OLTP Application Bottlenecks

Logical Application bottlenecks today

- Locking/Latching/Spinlocks
- Logging latency
- Scale up associated latency

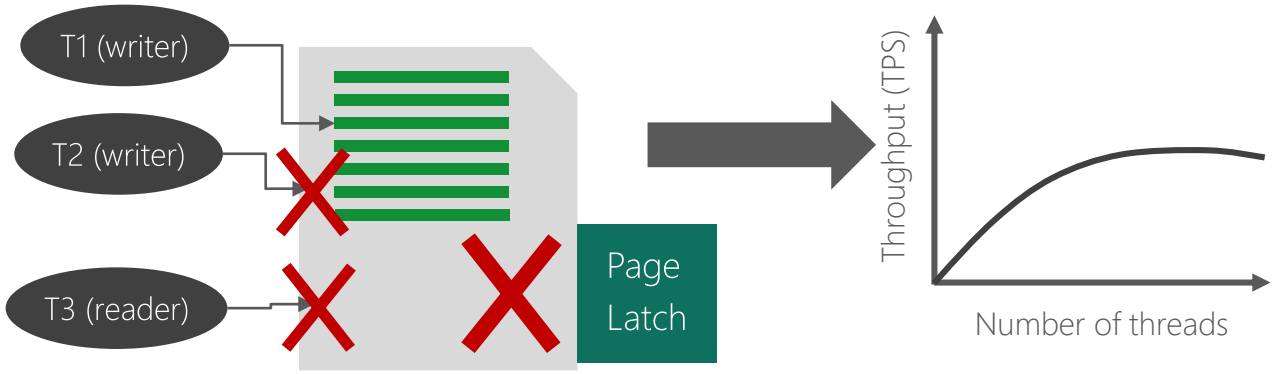
Performance Gains

Single Threaded

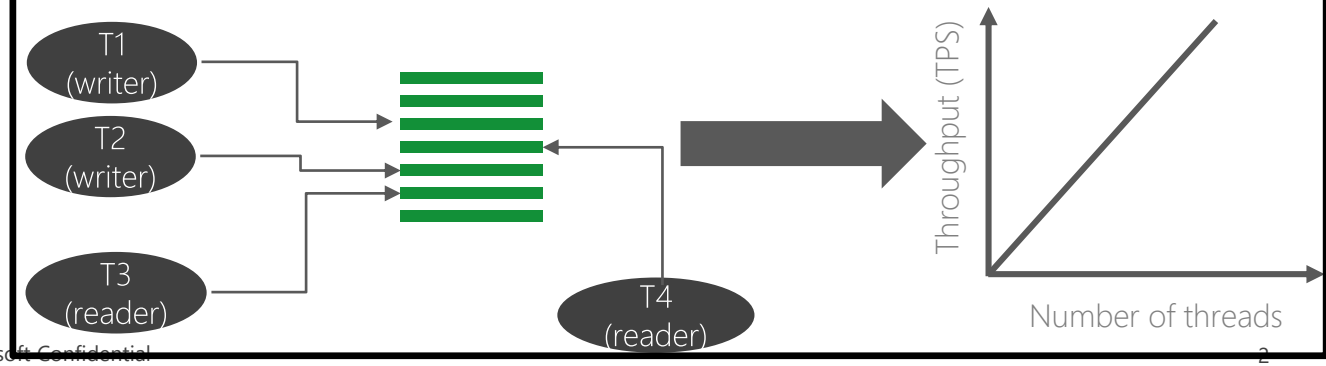
Traditional execution stack:	Performance gain:	In-Memory OLTP execution stack:
Client Connectivity	No improvement	Client Connectivity
Query Execution	2-10X improvement	Procedure Execution
Data Access (Buffer Pool)		Data Access (Memory-Optimized)
Transaction Logging	Same latency Less volume	Transaction Logging

Multi-Threaded

Disk based tables:



Memory-optimized tables:



Workload Types

In-Memory OLTP is a good fit for:

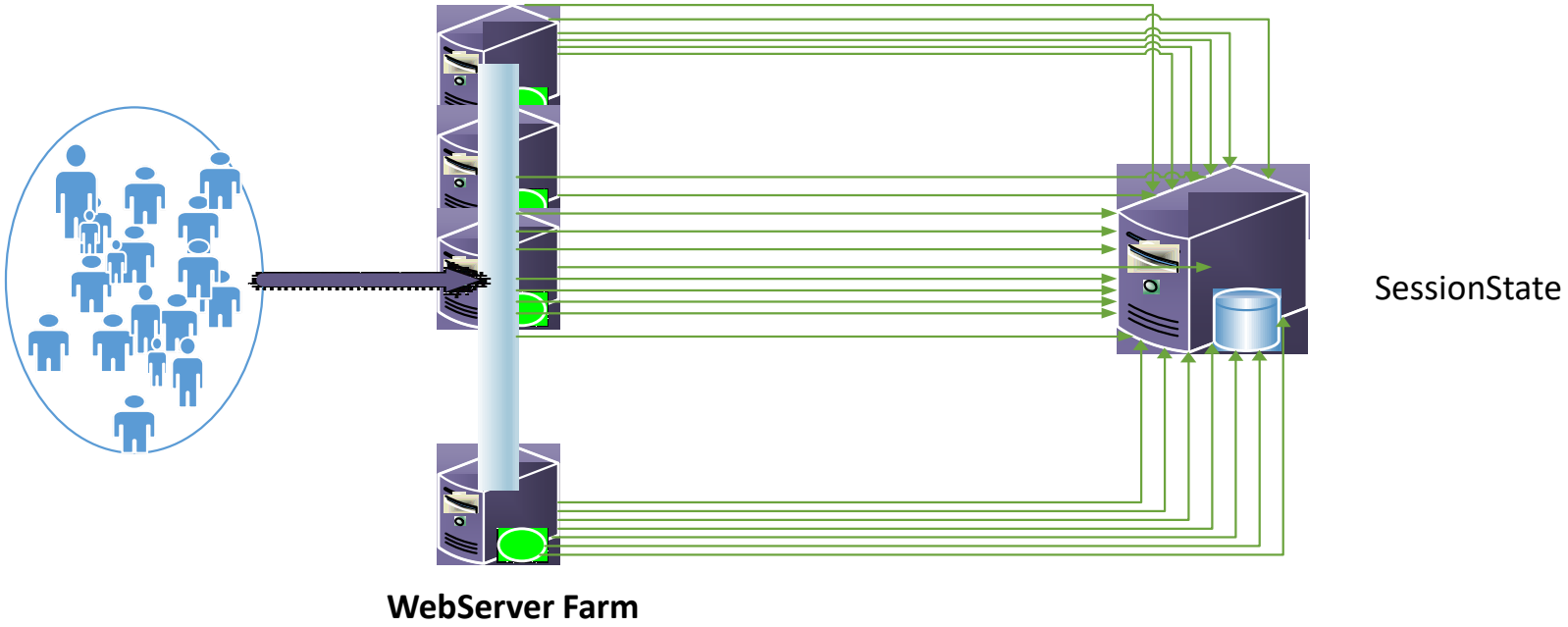
- Performance-critical OLTP (think order processing or trading)
- High data-input rate (nicknamed “Shock Absorber”)
- In-Memory OLTP as components of ETL
- Session state management
- Read scale

In-Memory OLTP is not a good fit for:

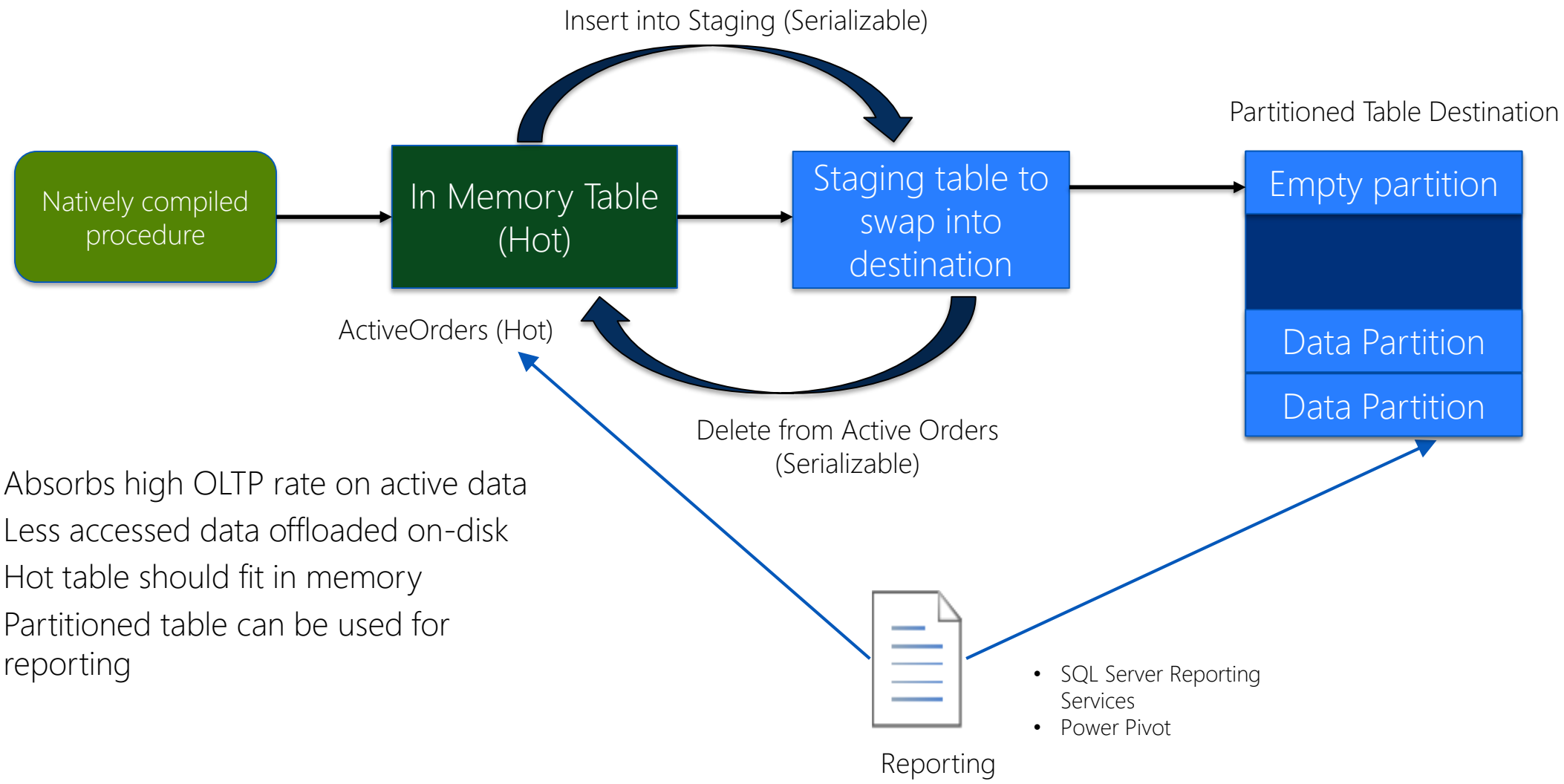
- No permission for code changes
- App depends on locking behavior
- Full data warehousing
- Long-running reporting workload (use Columnstore potentially instead)
- Use a lot of XML manipulation/Full-Text searches
- Heavily parallelized query
- Constrained on memory

Application Patterns – Session State

- Single-scale unit being the database servicing an ever increasing user workload
- Latch bottlenecks caused scale issues on single server, required scale-out
- If transient use SCHEMA_ONLY to eliminate logging
- App Tier caching may not be required
- Bwin.Party Case Study



Manual Partitioning Hot/Cold Data

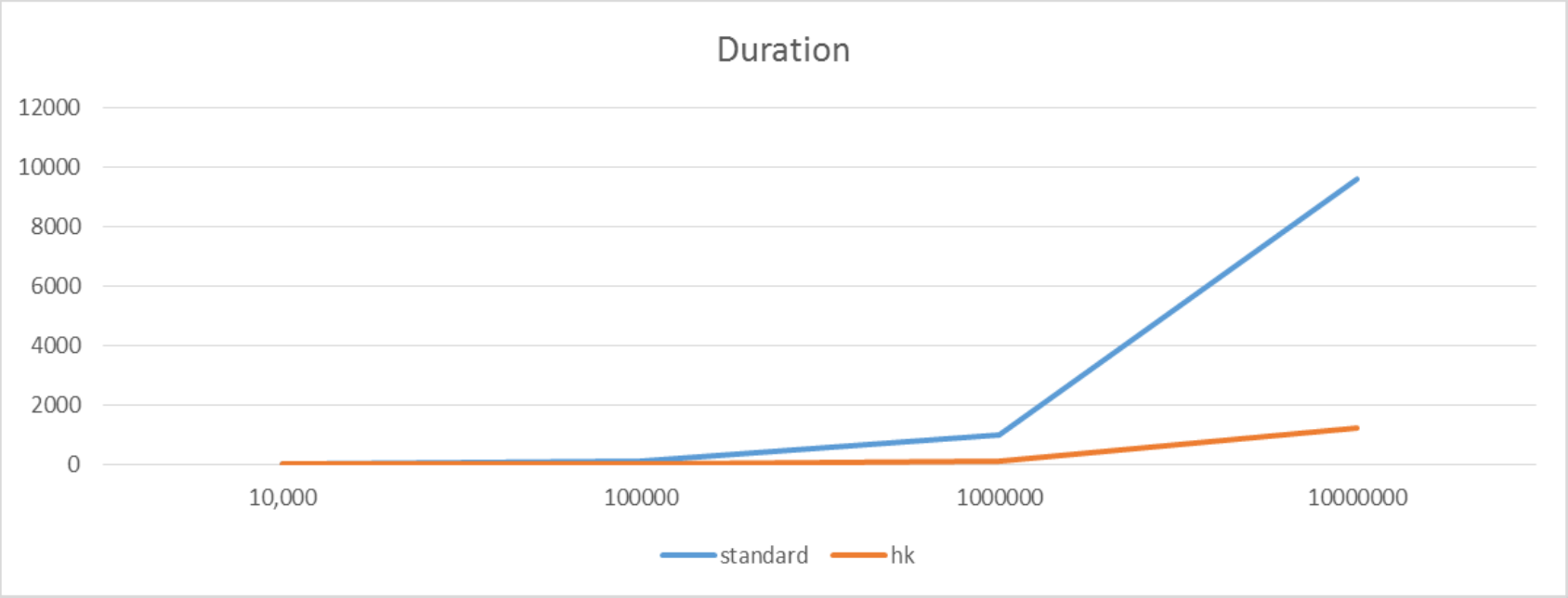


- Absorbs high OLTP rate on active data
- Less accessed data offloaded on-disk
- Hot table should fit in memory
- Partitioned table can be used for reporting

ETL Pipeline / Staging Tables

No Latching / Locking
Can bypass logging with SCHEMA_ONLY
Natively compiled procedures ideal for data transformation
Care should be taken of staging tables aggregated due to no parallelism

Edgenet case study



Considerations

Challenge	Resolution
Data in blobs > 8060 bytes	Wrapper to chunk the BLOB into multiple rows and then reconstruct the blob before output to client
IO latency at logging	Consider SCHEMA_ONLY for max performance if transient data, delayed durability for certain workloads
Hardware scaling	SAS drives, NUMA (may need to limit cores to a node)
Multi version concurrency	Modify code to handle write-write conflicts due to optimistic logging
Chatty applications	If possible combine multiple operations in a batch
Foreign key constraints	Manual checks in code logically implemented
Lack of parallelism	For queries requiring aggregations and the like

Testing Methodology

Test realistic workload

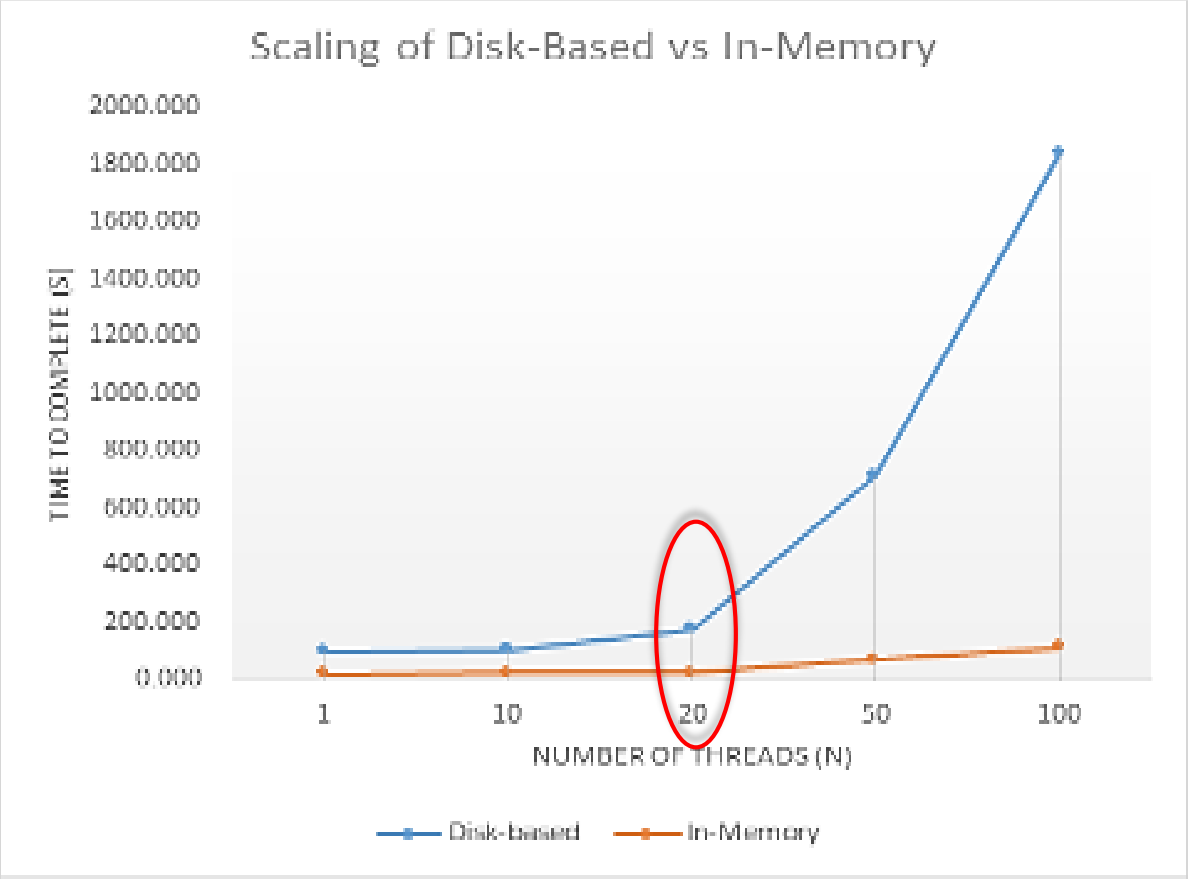
- Obtain an application metrics (logical business transactions or fixed workload)
- Right transaction mix
- Right table size and characteristics
- Test at scale for true comparison
- Production baseline is better

Goals and bottlenecks

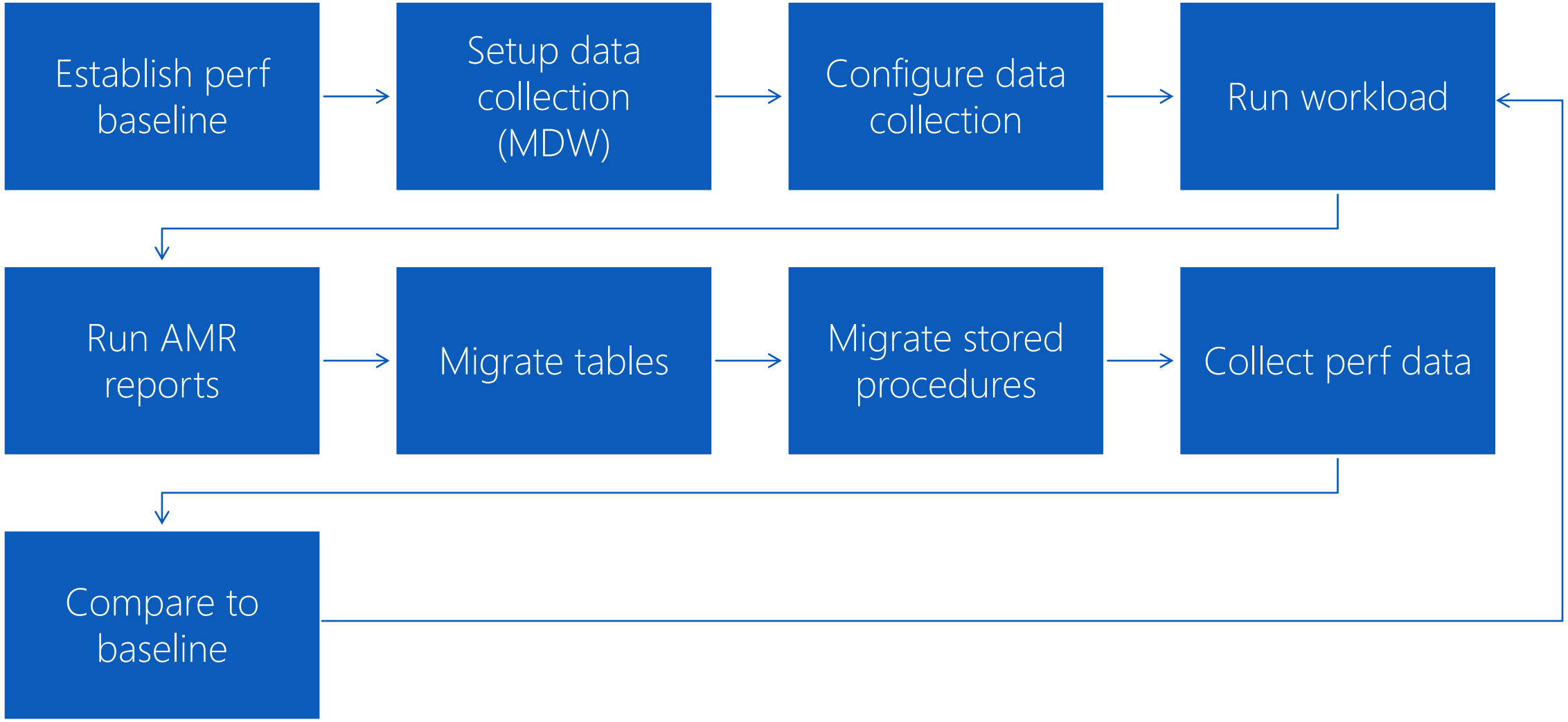
- Evaluate application bottleneck
- Is bottleneck addressed by In-Memory OLTP?

Profiling tools

- Ostress, Distributed Replay, Visual Studio Test System
- In-Memory OLTP's analysis, Migrate and Report toolset
- Performance Monitor



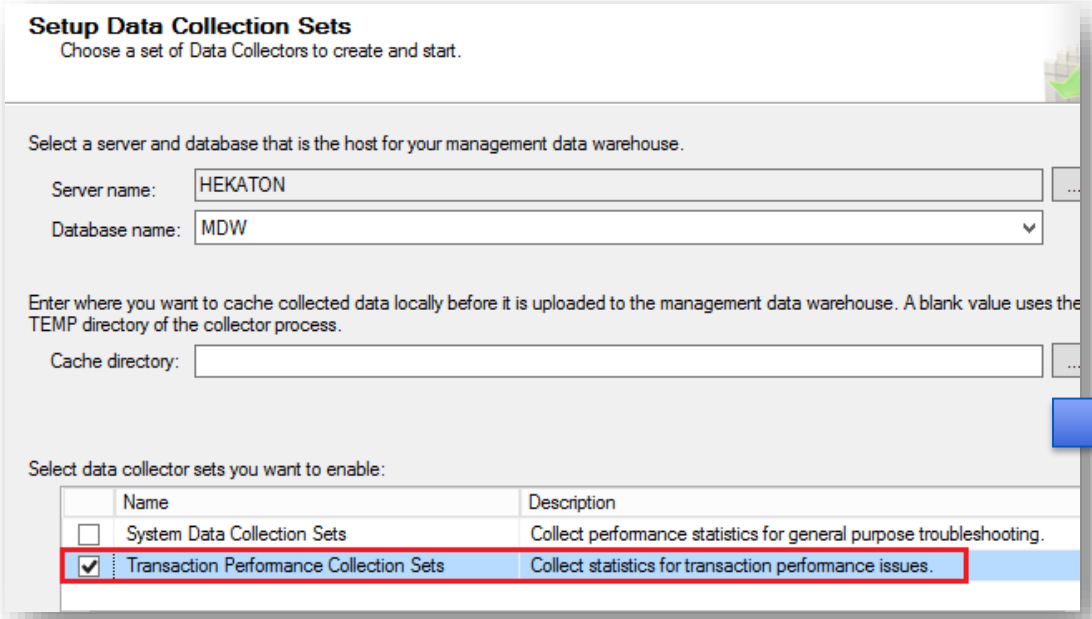
Iterative Methodology



AMR Toolset Data Collectors

Management data warehouse “New collectors” assist in analyzing migration candidates

Can be run remotely



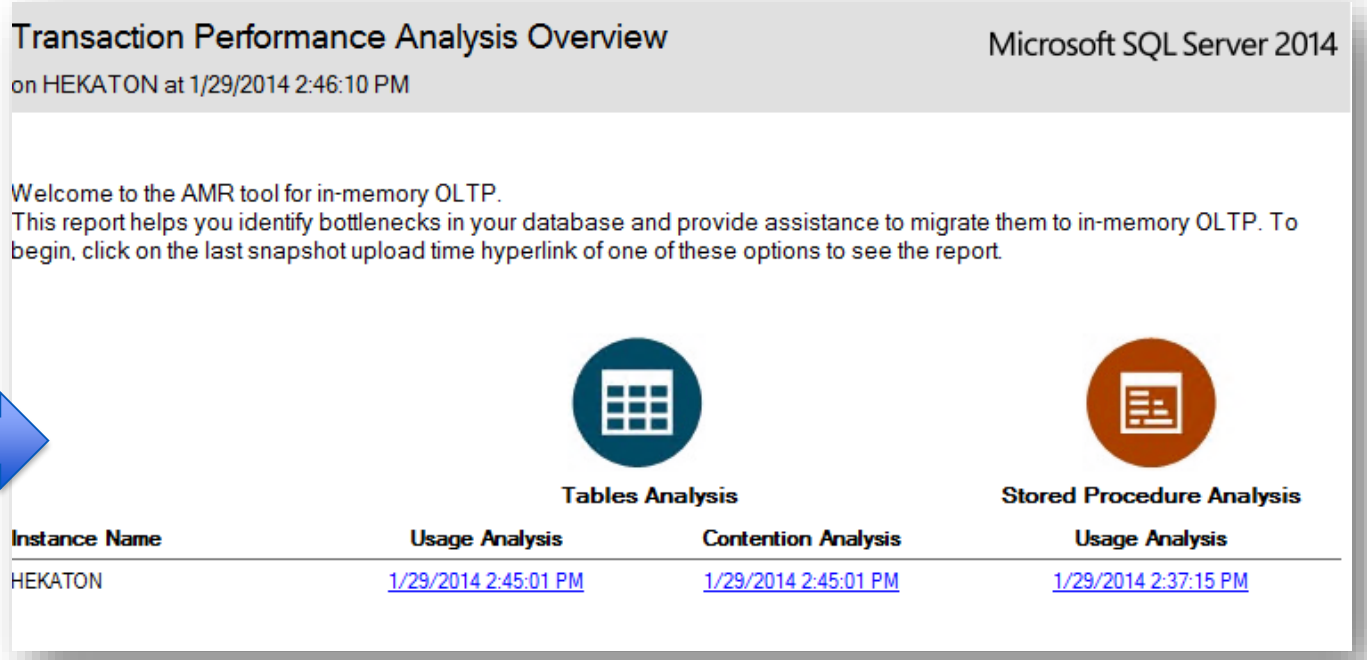
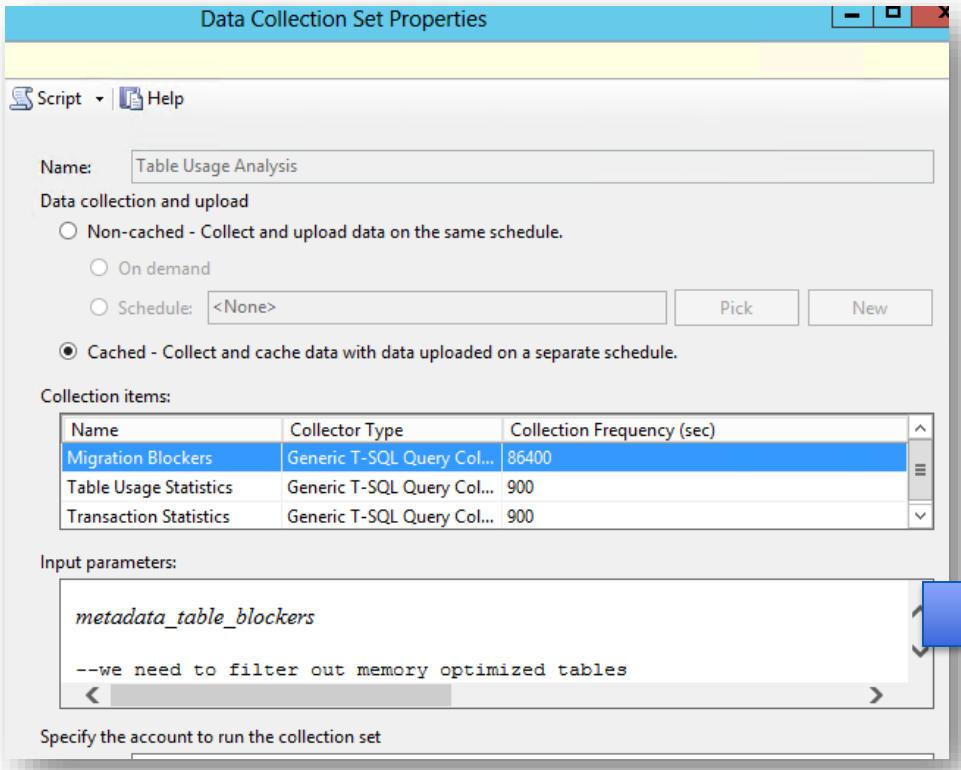


Table Usage Analysis



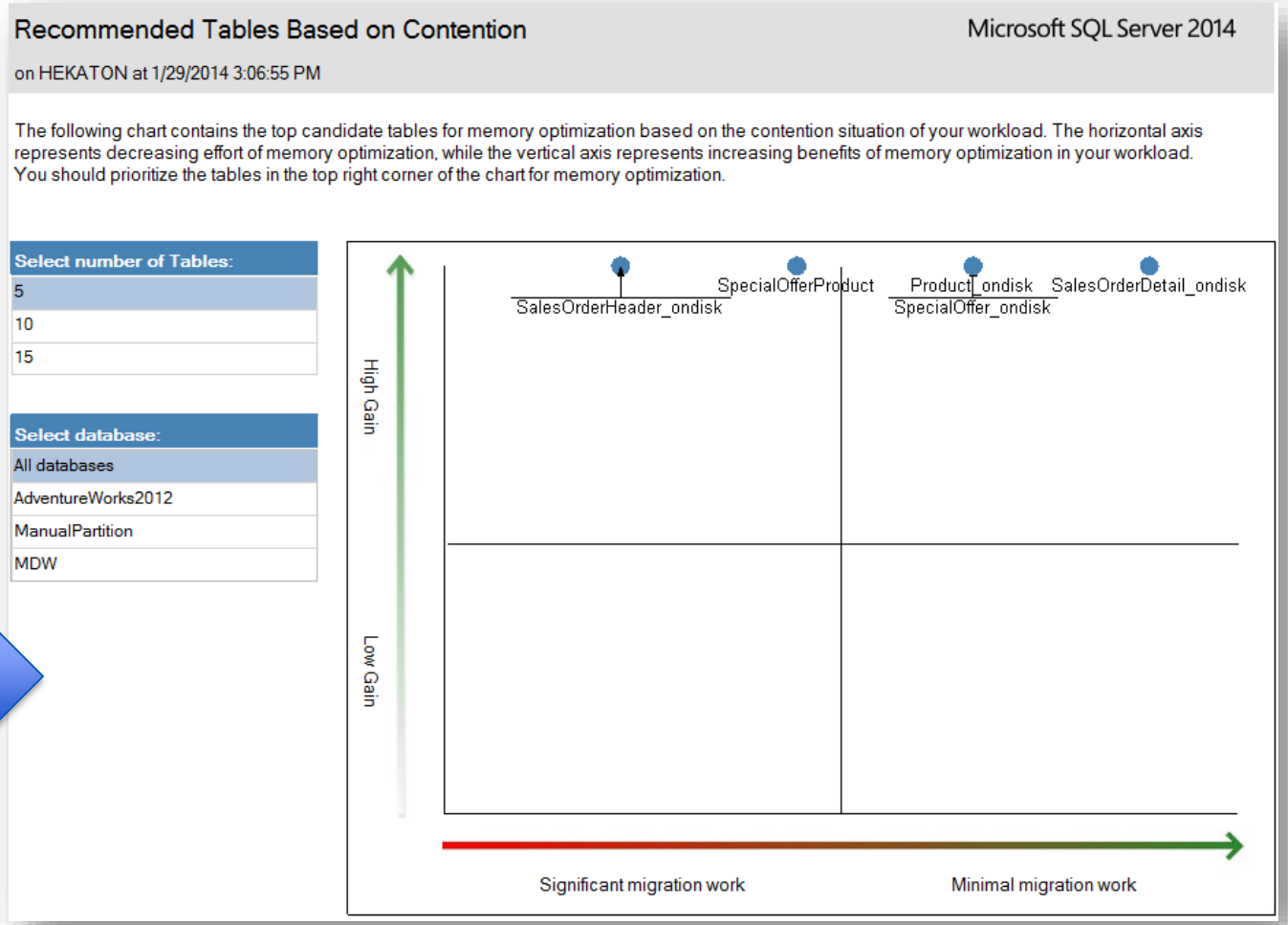


Table Usage Based Reporting

Table Usage Statistics

- sys.dm_db_index_operational_stats
- Should primarily be looked at by contention (Latches/Locks)
- Can be looked at by Usage (Seeks/Scans)

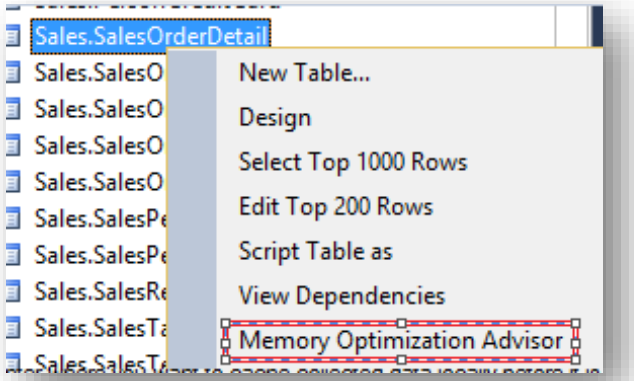
Migration Blockers

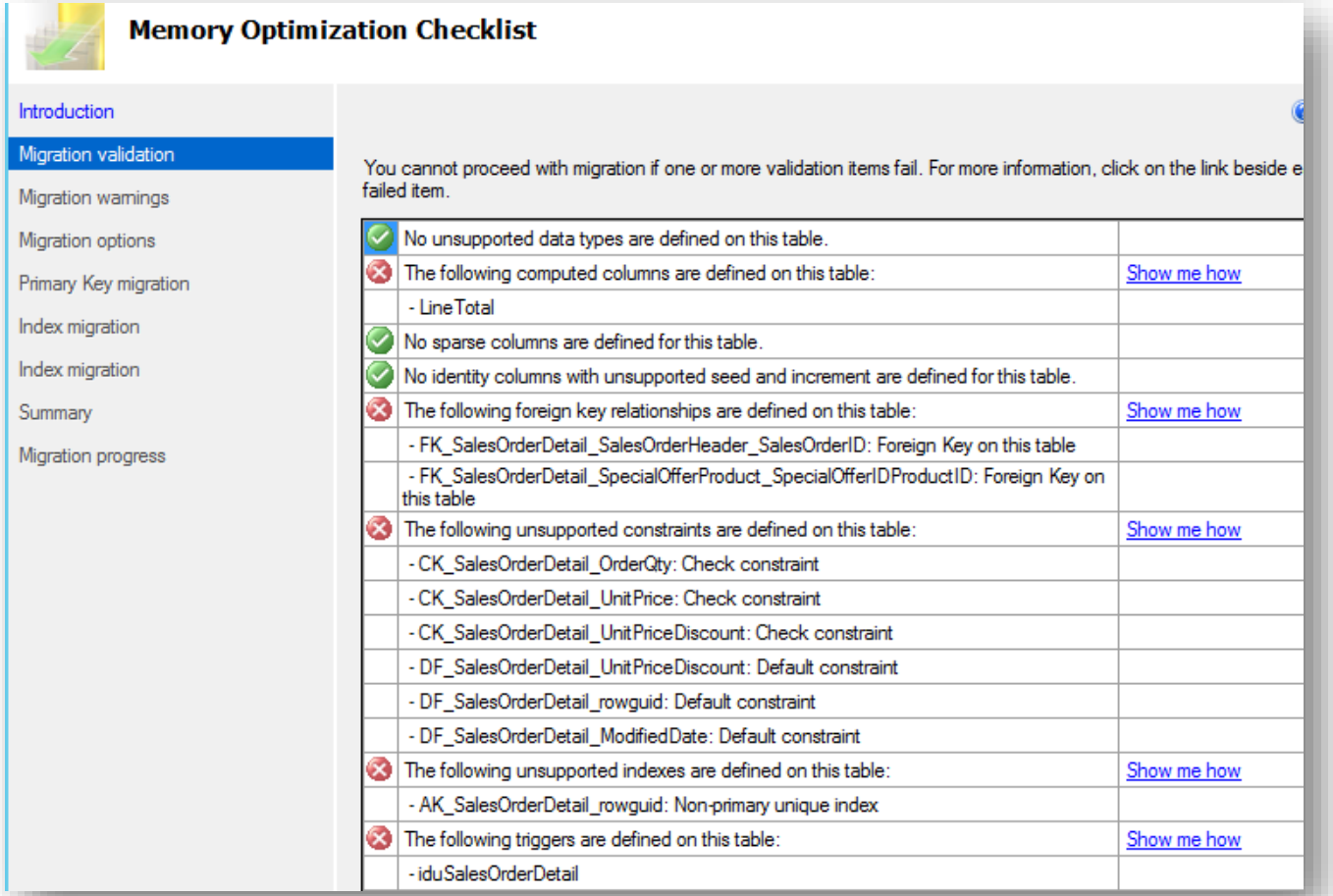
- Evaluates table metadata
- Number of constraints, unsupported column types, other blockers

Transaction Statistics

- Looks at Perfmon counter 'Transactions/sec'
- Memory-optimized tables transactions reflected in XTP transactions\sec

Memory Optimization Advisor





Can be used against down level clients 2008 and up

Table Integrity Limitations

Constraints

Some default constraints are supported, columns are not unique outside of primary keys

- Use stored procedures to enforce check constraints
- Constraints Workaround

Foreign Keys

Foreign keys are not supported from or to a memory-optimized table

- Re-evaluate necessity of foreign keys
- Use stored procedures to enforce foreign-key relationships

Triggers

- Re-evaluate necessity of DML triggers
- Use stored procedures to emulate trigger functionality
- Unable to create memory optimized when using DDL Trigger for CREATE_TABLE
- Migrating Triggers

Column Type Limitations

LOB data types or rows > 8060 bytes

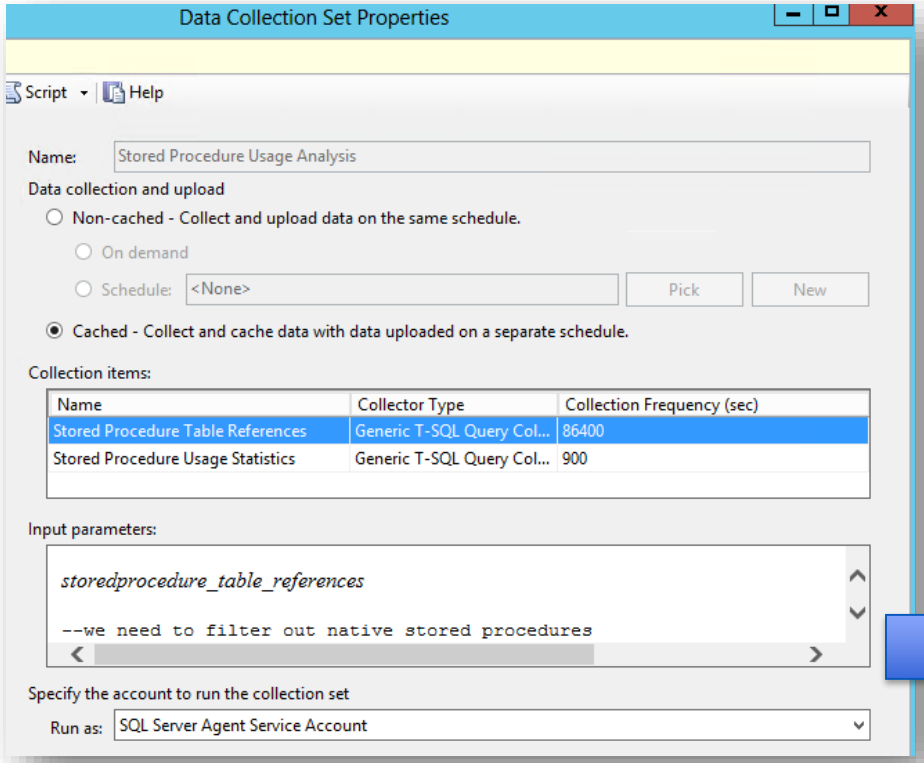
VARCHAR(max) and NVARCHAR(max) are considered LOB columns and are not supported

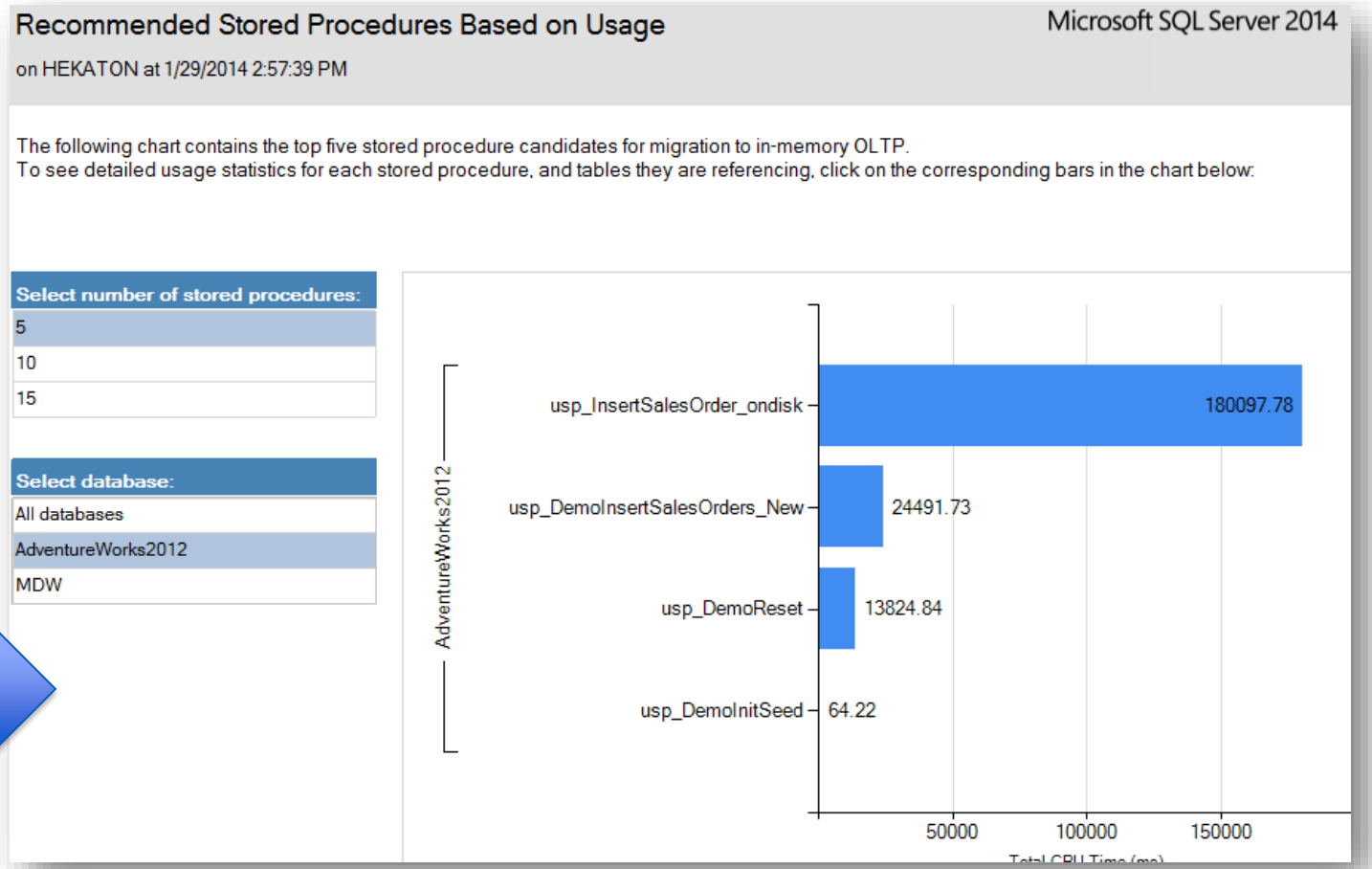
- If data size is known, can convert columns to have the maximum available data size
- Can move above columns to disk-based tables
- Can split above columns into many rows
- LOB Workarounds

Computed Columns

- Can calculate before insert
- Can use stored procedures to return non-persisted computed columns
- Migrating Computed Columns

Stored Procedure Usage Analysis

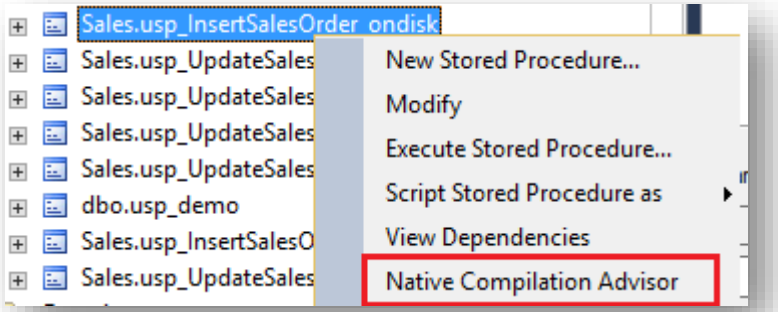


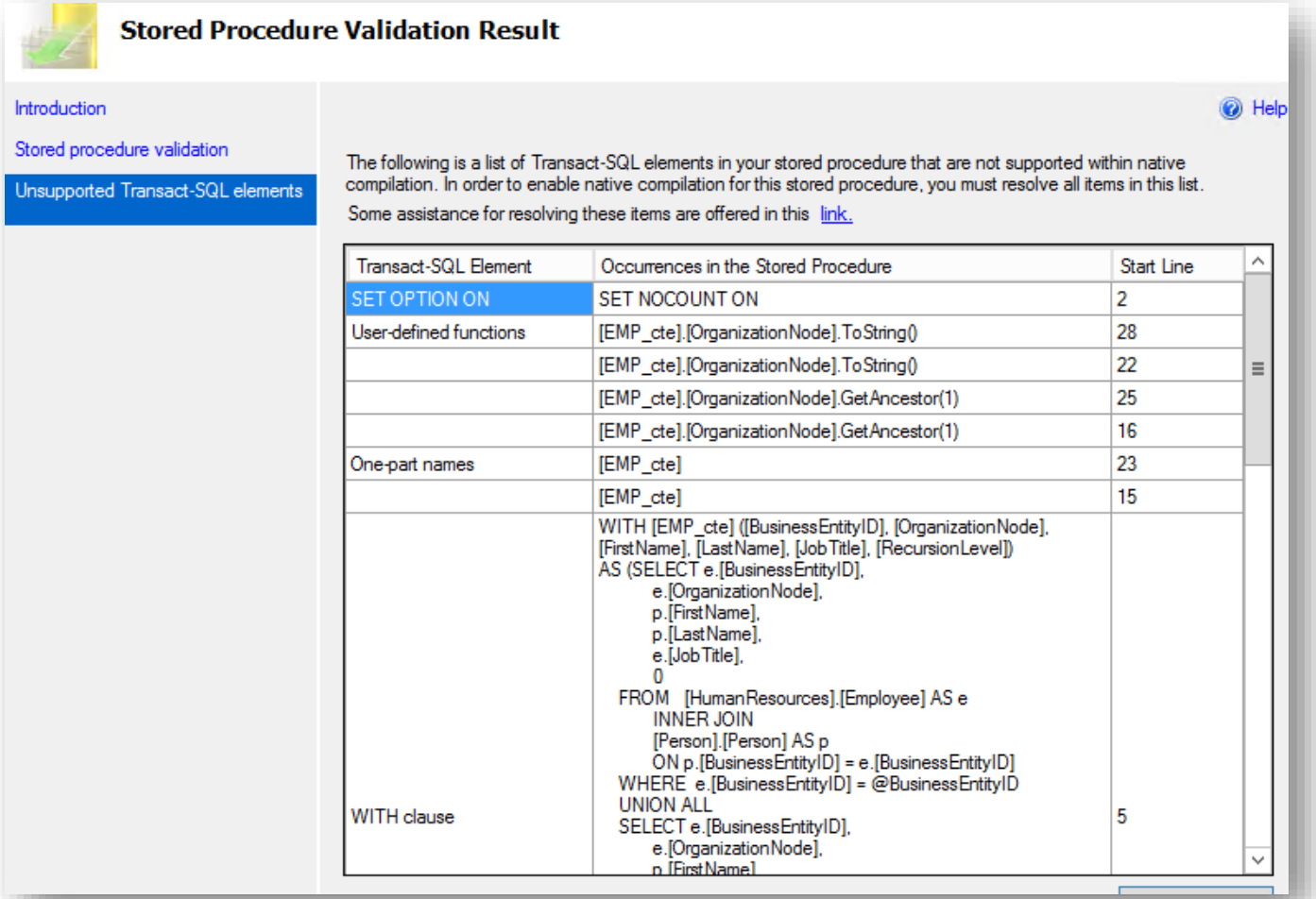


Primarily looks at *sys.dm_exec_procedure_stats*

Drill down to referencing objects

Native Compilation Advisor





Transact-SQL Element	Occurrences in the Stored Procedure	Start Line
SET OPTION ON	SET NOCOUNT ON	2
User-defined functions	[EMP_cte].[OrganizationNode].ToString()	28
	[EMP_cte].[OrganizationNode].ToString()	22
	[EMP_cte].[OrganizationNode].GetAncestor(1)	25
	[EMP_cte].[OrganizationNode].GetAncestor(1)	16
One-part names	[EMP_cte]	23
	[EMP_cte]	15
WITH clause	WITH [EMP_cte] ([BusinessEntityID], [OrganizationNode], [FirstName], [LastName], [JobTitle], [RecursionLevel]) AS (SELECT e.[BusinessEntityID], e.[OrganizationNode], p.[FirstName], p.[LastName], e.[JobTitle], 0 FROM [HumanResources].[Employee] AS e INNER JOIN [Person].[Person] AS p ON p.[BusinessEntityID] = e.[BusinessEntityID] WHERE e.[BusinessEntityID] = @BusinessEntityID UNION ALL SELECT e.[BusinessEntityID], e.[OrganizationNode], p.[FirstName]	5

[TSQL Constructs Not Supported]

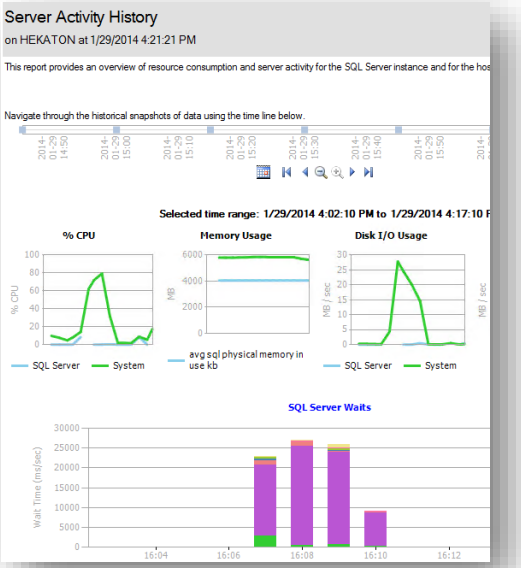
Other Tools

DMVs

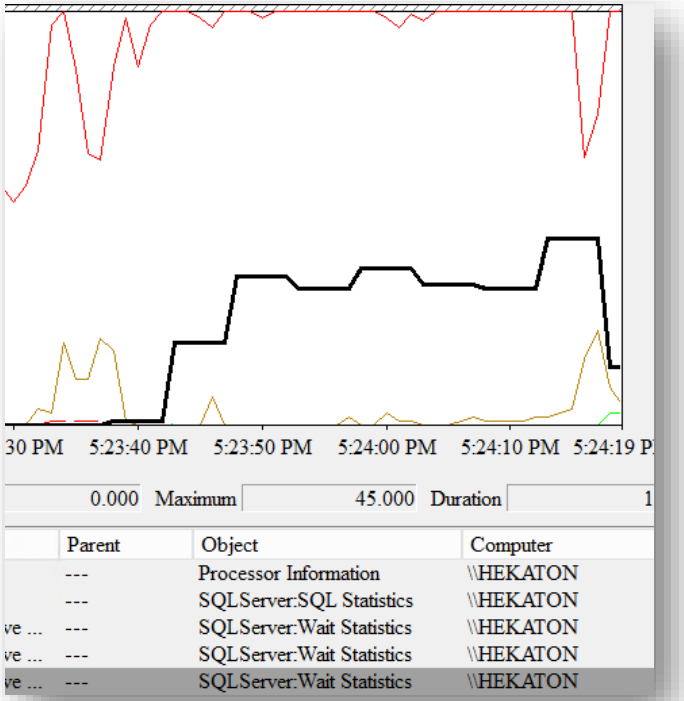
```sql
-- Check waits, No Latch waits.
WITH WaitCategoryStats ( wait_category, wait_type, wait_time_ms, waiting_tasks_count, max_wait_time_ms) AS
( SELECT
  CASE
    WHEN wait_type LIKE 'LCK%' THEN 'LOCKS'
    WHEN wait_type LIKE 'PAGEIO%' THEN 'PAGE I/O LATCH'
    WHEN wait_type LIKE 'PAGELATCH%' THEN 'PAGE LATCH (non-I/O)'
    WHEN wait_type LIKE 'LATCH%' THEN 'LATCH (non-buffer)'
    WHEN wait_type LIKE 'LATCH%' THEN 'LATCH (non-buffer)'
    ELSE wait_type
  END AS wait_category, wait_type, wait_time_ms, waiting_tasks_count, max_wait_time_ms
  FROM sys.dm_os_wait_stats
)
SELECT TOP 25 wait_category, sum(wait_time_ms) as wait_time_ms,
sum(waiting_tasks_count) as waiting_tasks_count
from WaitCategoryStats
where
 wait_type NOT IN ('WAITFOR', 'LAZYWRITER_SLEEP', 'SQLTRACE_BUFFER_FLUSH', 'CXPACKET', 'EXCHANGE', 'REQUEST
    'CHKPT', 'DBMIRROR_WORKER_QUEUE', 'DBMIRRORING_CMD', 'DBMIRROR_DBM_EVENT','XE_DISPATCHER_WAIT')
    and wait_type not like 'PREEMPTIVE%'
GROUP BY wait_category
ORDER BY sum(wait_time_ms) DESC
```

```sql
SELECT
  CONVERT (varchar, getdate(), 126) AS runtime,
  LEFT (p.cacheobjtype + ' (' + p.objtype + ')', 35) AS cacheobjtype,
  p.usecounts, PlanStats.total_worker_time/1000 AS tot_cpu_ms, PlanStats.total_elapsed_time/1000 AS tot_duration_ms,
  PlanStats.total_physical_reads, PlanStats.total_logical_writes, PlanStats.total_logical_reads,
  PlanStats.CpuRank, PlanStats.IORank, PlanStats.DurationRank,
  object_name(sql.objectid, sql.dbid) AS procname,
  REPLACE (REPLACE (SUBSTRING (sql.[text], PlanStats.statement_start_offset/2 + 1,
      CASE WHEN PlanStats.statement_end_offset = -1 THEN LEN (CONVERT(nvarchar(max), sql.[text]))
        ELSE PlanStats.statement_end_offset/2 - PlanStats.statement_start_offset/2 + 1
      END), CHAR(13), ' '), CHAR(10), ' ') AS stmt_text
FROM
(  SELECT
    stat.plan_handle, statement_start_offset, statement_end_offset,
    stat.total_worker_time, stat.total_elapsed_time, stat.total_physical_reads,
    stat.total_logical_writes, stat.total_logical_reads,
    ROW_NUMBER() OVER (ORDER BY stat.total_worker_time DESC) AS CpuRank,
    ROW_NUMBER() OVER (ORDER BY (stat.total_physical_reads + stat.total_logical_reads
                        + stat.total_logical_writes) DESC  ) AS IORank,
    ROW_NUMBER() OVER (ORDER BY stat.total_elapsed_time DESC) AS DurationRank
  FROM sys.dm_exec_query_stats stat ) AS PlanStats
INNER JOIN sys.dm_exec_cached_plans p ON p.plan_handle = PlanStats.plan_handle
OUTER APPLY sys.dm_exec_plan_attributes (p.plan_handle) pa
OUTER APPLY sys.dm_exec_sql_text (p.plan_handle) AS sql
WHERE ( PlanStats.CPURank < 100 )
AND pa.attribute = 'dbid'
ORDER BY tot_cpu_ms DESC
```

Waits & Queues



Perfmon



SQLDIAG / SQL Nexus / XEvent