

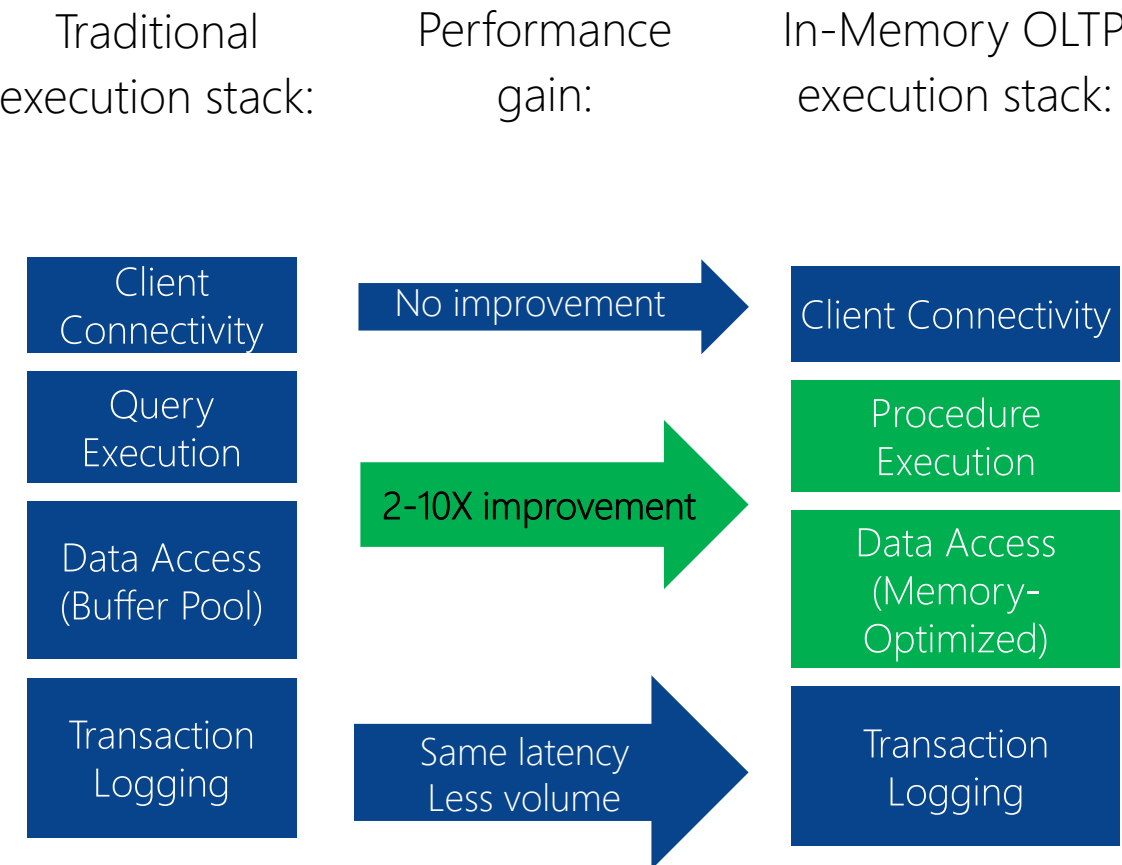
OLTP Application Bottlenecks

Logical
Application
bottlenecks
today

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

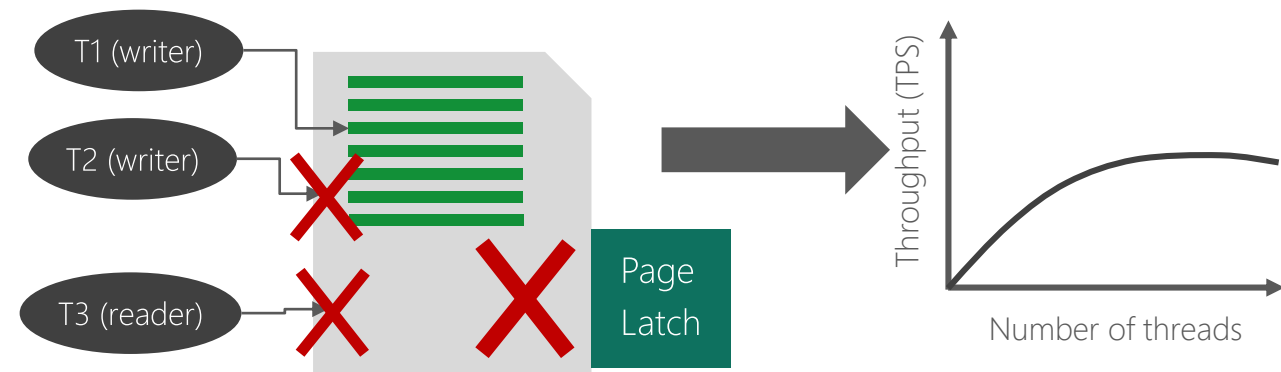
Performance Gains

Single Threaded

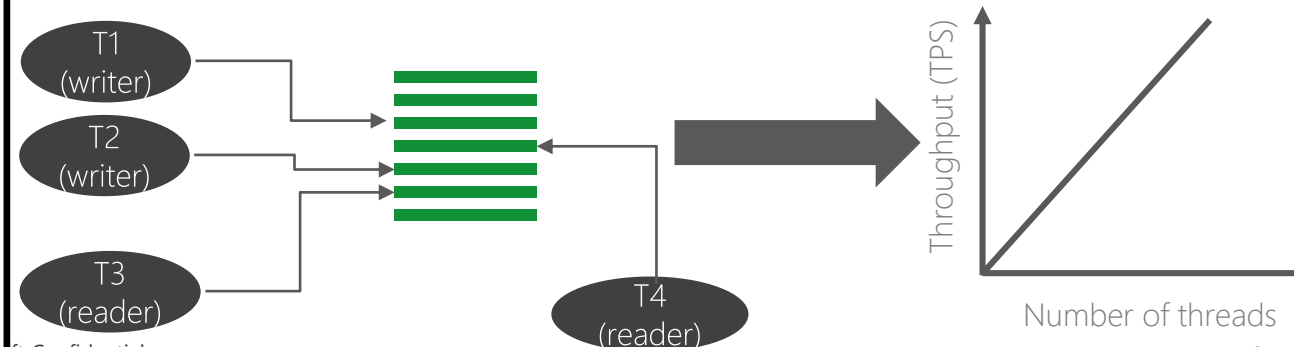


Multi-Threaded

Disk based tables:



Memory-optimized tables:



Workload Types

In-Memory OLTP is a good fit for:

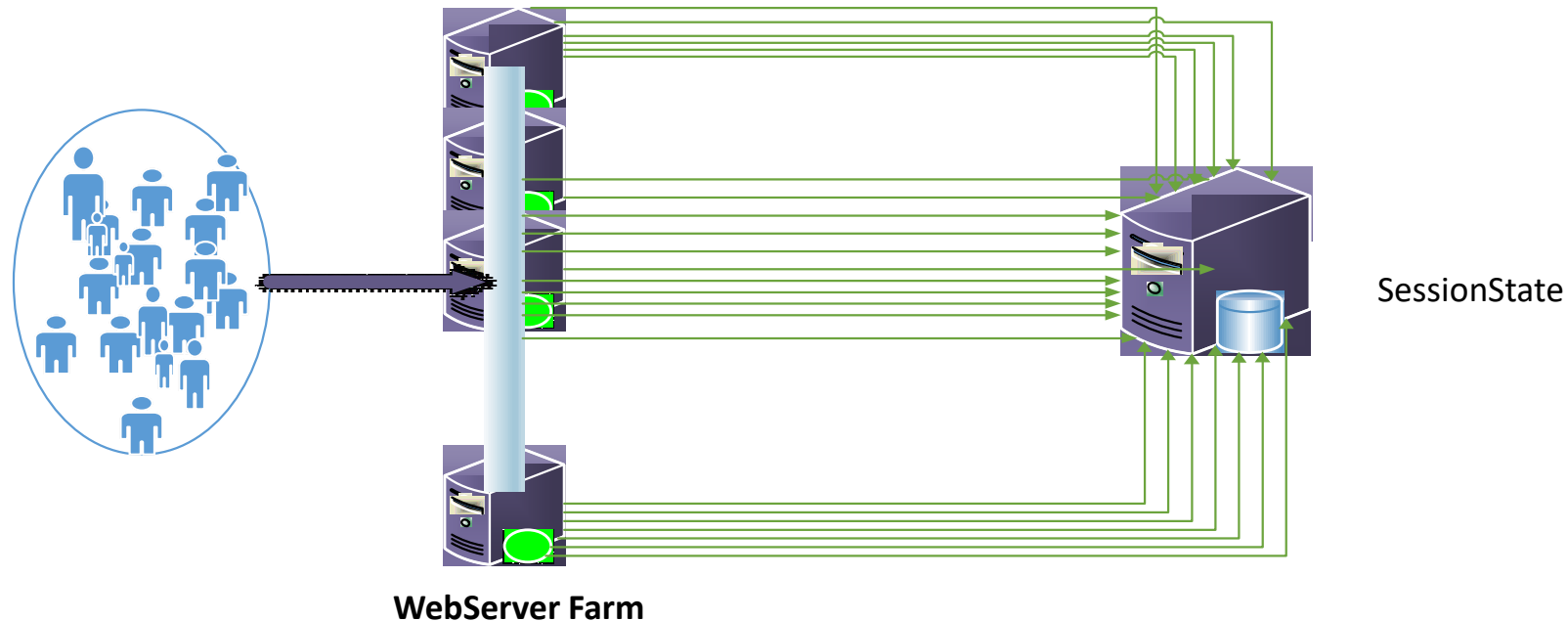
- Performance-critical OLTP (think order processing or trading)
- High data-input rate (nicknamed “Shock Absorber”)
- In-Memory OLTP as components of ETL
- Session state management
- Read scale

In-Memory OLTP is not a good fit for:

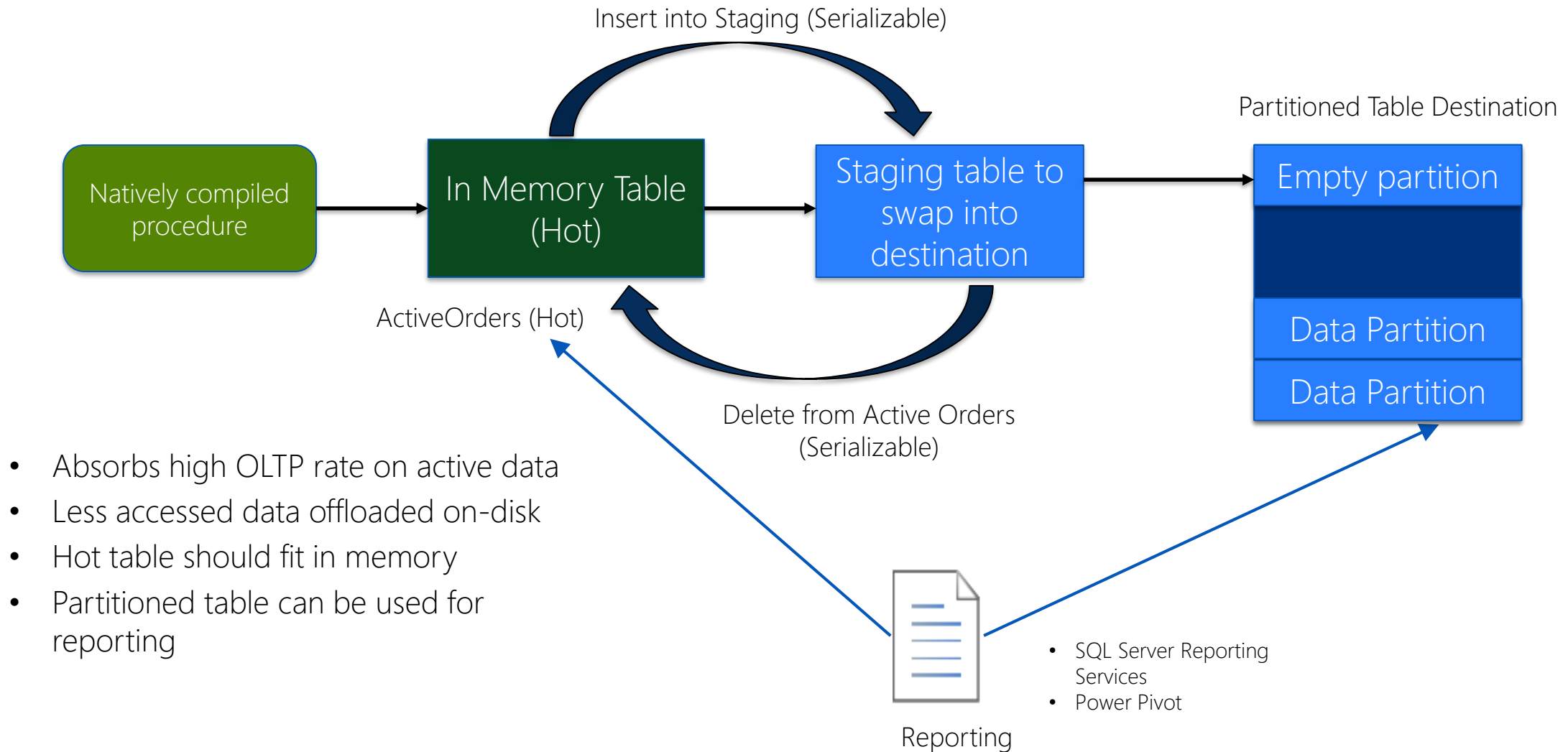
- No permission for code changes
- App depends on locking behavior
- Full data warehousing
- Long-running reporting workload (use Columnstore potentially instead)
- Use a lot of XML manipulation/Full-Text searches
- Heavily parallelized query
- Constrained on memory

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



ETL Pipeline / Staging Tables

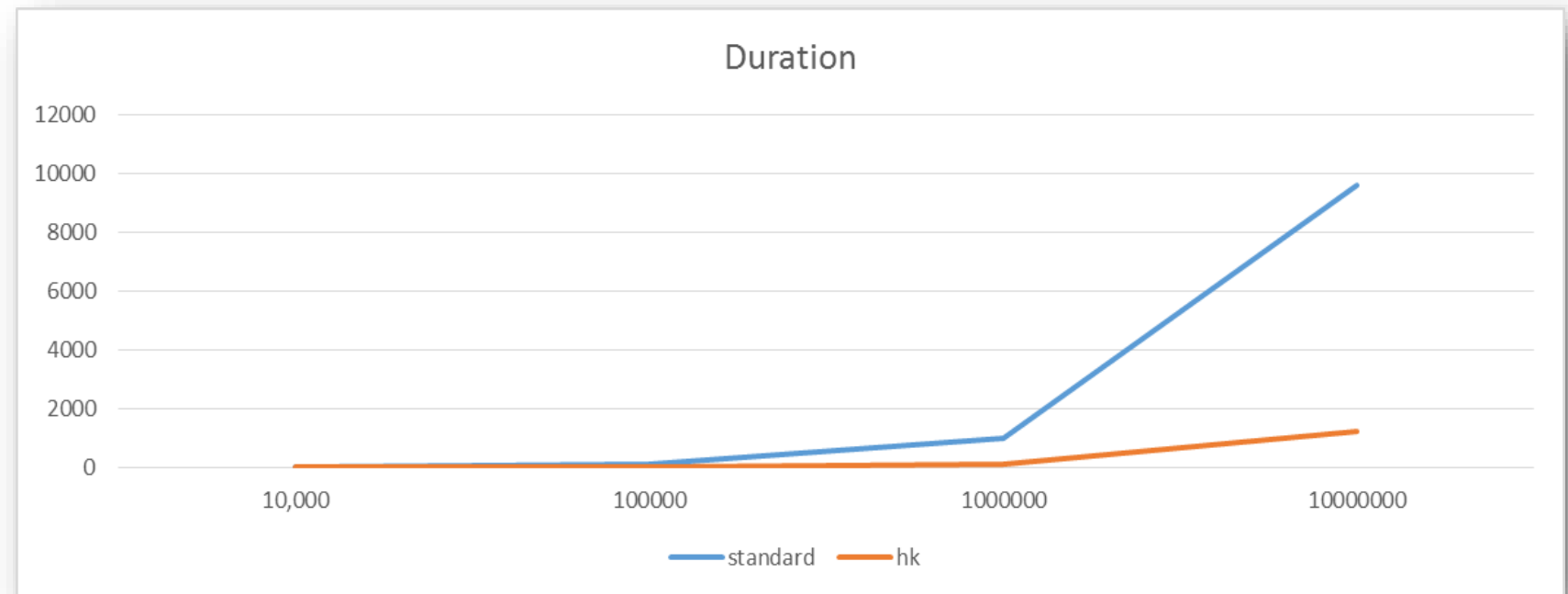
No Latching / Locking

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

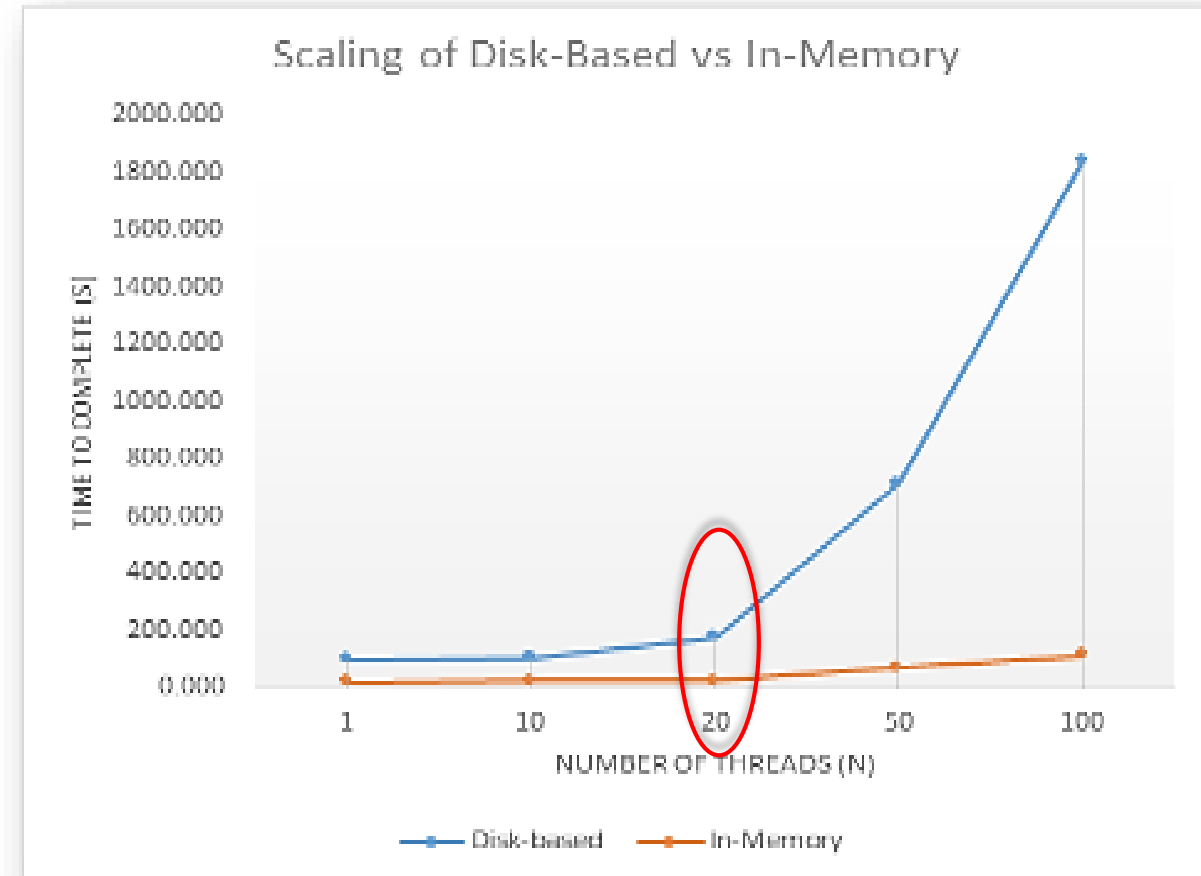
- Obtain an application metrics (logical business transactions or fixed workload)
- Right transaction mix
- Right table size and characteristics
- Test at scale for true comparison
- Production baseline is better

Goals and bottlenecks

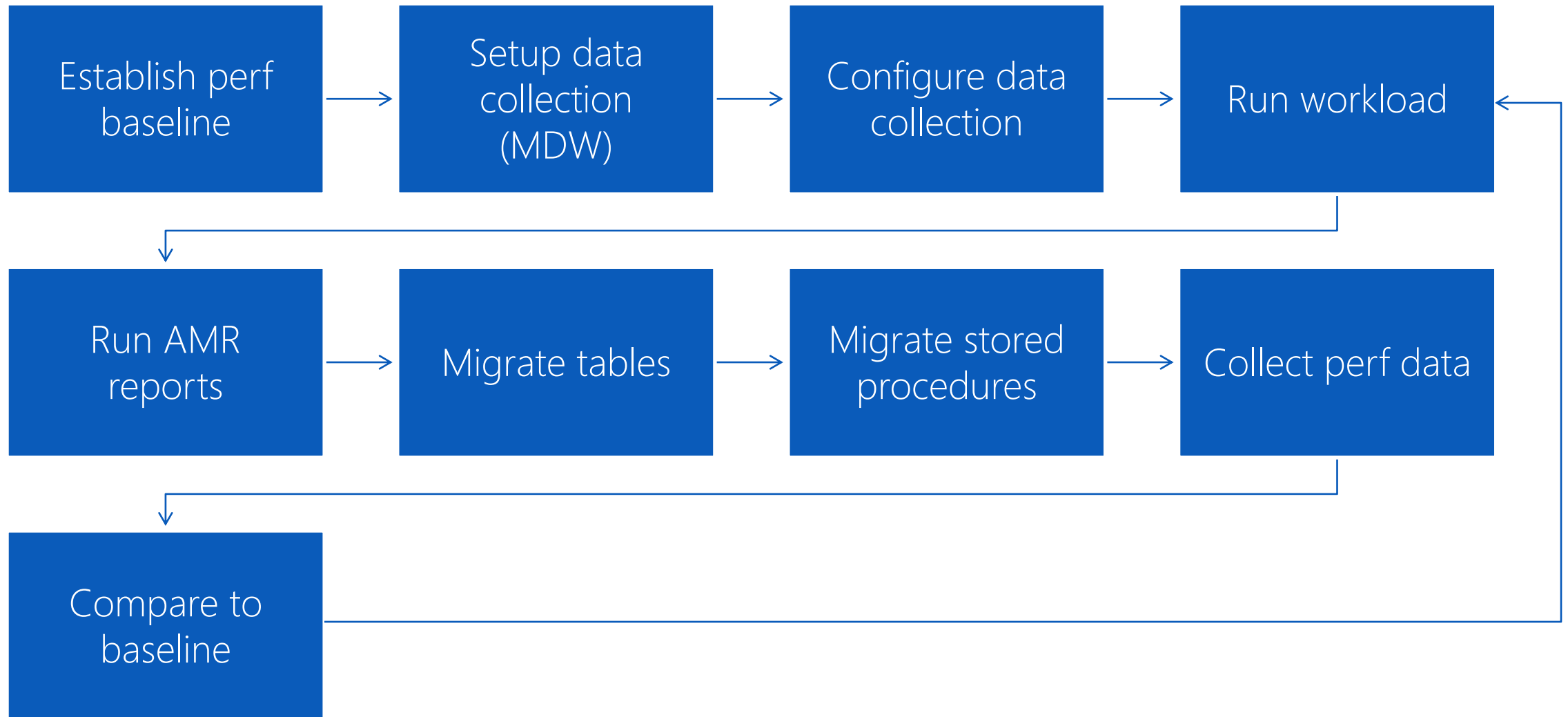
- Evaluate application bottleneck
- Is bottleneck addressed by In-Memory OLTP?

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

Setup Data Collection Sets

Choose a set of Data Collectors to create and start.

Select a server and database that is the host for your management data warehouse.

Server name: HEKATON

Database name: MDW

Enter where you want to cache collected data locally before it is uploaded to the management data warehouse. A blank value uses the TEMP directory of the collector process.

Cache directory:

Select data collector sets you want to enable:

	Name	Description
<input type="checkbox"/>	System Data Collection Sets	Collect performance statistics for general purpose troubleshooting.
<input checked="" type="checkbox"/>	Transaction Performance Collection Sets	Collect statistics for transaction performance issues.

Transaction Performance Analysis Overview

Microsoft SQL Server 2014

on HEKATON at 1/29/2014 2:46:10 PM

Welcome to the AMR tool for in-memory OLTP.

This report helps you identify bottlenecks in your database and provide assistance to migrate them to in-memory OLTP. To begin, click on the last snapshot upload time hyperlink of one of these options to see the report.



Tables Analysis



Stored Procedure Analysis

Instance Name	Usage Analysis	Contention Analysis	Usage Analysis
HEKATON	1/29/2014 2:45:01 PM	1/29/2014 2:45:01 PM	1/29/2014 2:37:15 PM

Table Usage Analysis

Data Collection Set Properties

Script Help

Name: Table Usage Analysis

Data collection and upload

☐ Non-cached - Collect and upload data on the same schedule.

☐ On demand

☐ Schedule: <None> Pick New

☒ Cached - Collect and cache data with data uploaded on a separate schedule.

Collection items:

Name	Collector Type	Collection Frequency (sec)
Migration Blockers	Generic T-SQL Query Col...	86400
Table Usage Statistics	Generic T-SQL Query Col...	900
Transaction Statistics	Generic T-SQL Query Col...	900

Input parameters:

```
metadata_table_blockers
```

```
--we need to filter out memory optimized tables
```

Specify the account to run the collection set

Recommended Tables Based on Contention

Microsoft SQL Server 2014

on HEKATON at 1/29/2014 3:06:55 PM

The following chart contains the top candidate tables for memory optimization based on the contention situation of your workload. The horizontal axis represents decreasing effort of memory optimization, while the vertical axis represents increasing benefits of memory optimization in your workload. You should prioritize the tables in the top right corner of the chart for memory optimization.

Select number of Tables:

5
10
15

Select database:

All databases
AdventureWorks2012
ManualPartition
MDW

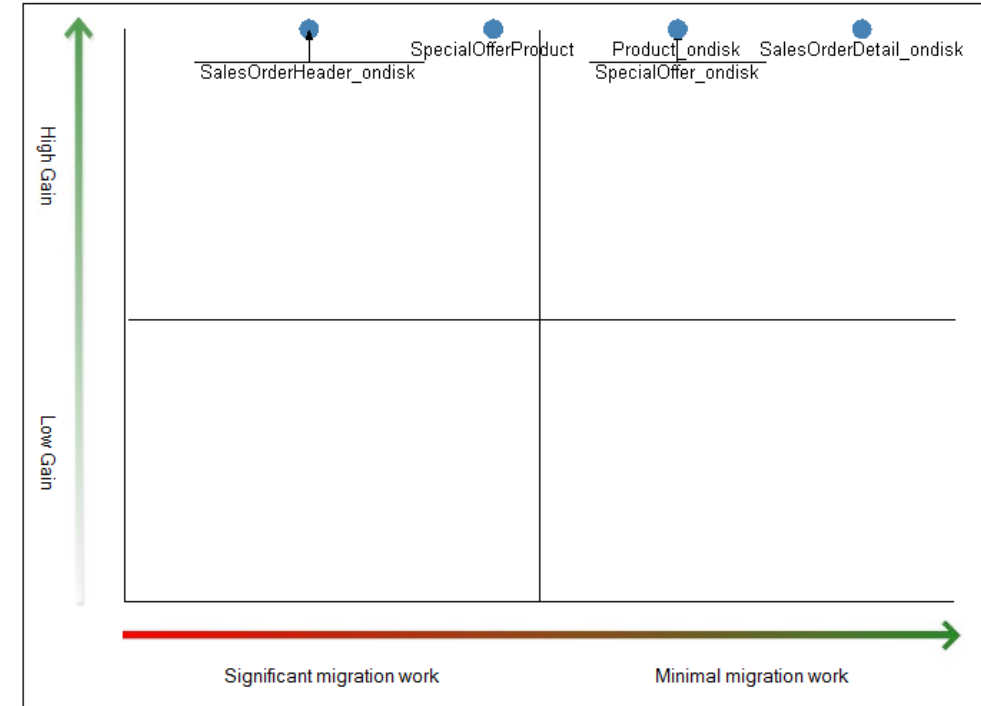


Table Usage Based Reporting

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)

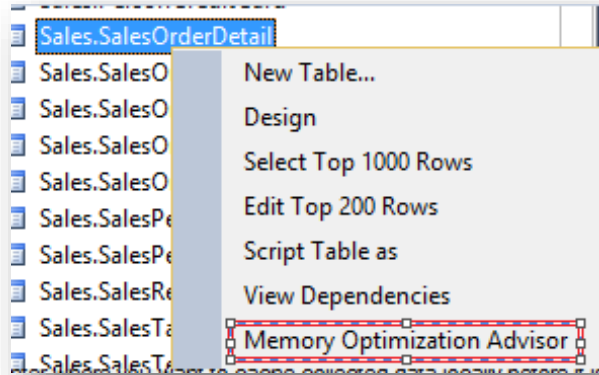
Migration Blockers

- 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

Memory Optimization Checklist

[Introduction](#)
Migration validation
[Migration warnings](#)
[Migration options](#)
[Primary Key migration](#)
[Index migration](#)
[Index migration](#)
[Summary](#)
[Migration progress](#)

You cannot proceed with migration if one or more validation items fail. For more information, click on the link beside each failed item.

✓	No unsupported data types are defined on this table.	
✗	The following computed columns are defined on this table: - LineTotal	Show me how
✓	No sparse columns are defined for this table.	
✓	No identity columns with unsupported seed and increment are defined for this table.	
✗	The following foreign key relationships are defined on this table: - FK_SalesOrderDetail_SalesOrderHeader_SalesOrderID: Foreign Key on this table - FK_SalesOrderDetail_SpecialOfferProduct_SpecialOfferIDProductID: Foreign Key on this table	Show me how
✗	The following unsupported constraints are defined on this table: - CK_SalesOrderDetail_OrderQty: Check constraint - CK_SalesOrderDetail_UnitPrice: Check constraint - CK_SalesOrderDetail_UnitPriceDiscount: Check constraint - DF_SalesOrderDetail_UnitPriceDiscount: Default constraint - DF_SalesOrderDetail_rowguid: Default constraint - DF_SalesOrderDetail_ModifiedDate: Default constraint	Show me how
✗	The following unsupported indexes are defined on this table: - AK_SalesOrderDetail_rowguid: Non-primary unique index	Show me how
✗	The following triggers are defined on this table: - iduSalesOrderDetail	Show me how

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

Data Collection Set Properties

Script | Help

Name:

Data collection and upload

☐ Non-cached - Collect and upload data on the same schedule.

☐ On demand

☐ Schedule:

☒ Cached - Collect and cache data with data uploaded on a separate schedule.

Collection items:

Name	Collector Type	Collection Frequency (sec)
Stored Procedure Table References	Generic T-SQL Query Col...	86400
Stored Procedure Usage Statistics	Generic T-SQL Query Col...	900

Input parameters:

```
storedprocedure_table_references
--we need to filter out native stored procedures
```

Specify the account to run the collection set

Run as:



Recommended Stored Procedures Based on Usage

Microsoft SQL Server 2014

on HEKATON at 1/29/2014 2:57:39 PM

The following chart contains the top five stored procedure candidates for migration to in-memory OLTP. To see detailed usage statistics for each stored procedure, and tables they are referencing, click on the corresponding bars in the chart below.

Select number of stored procedures:

5

10

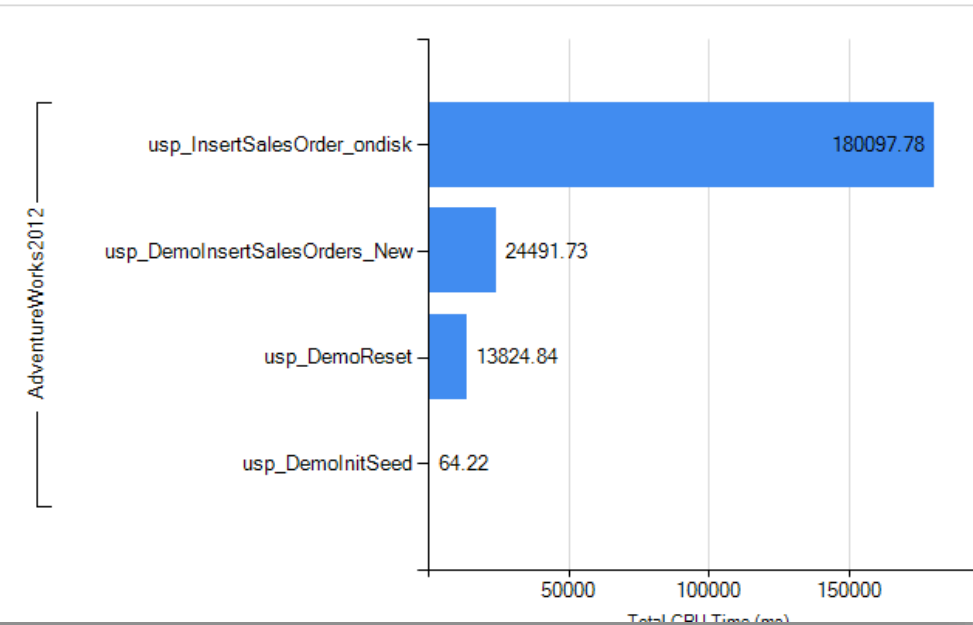
15

Select database:

All databases

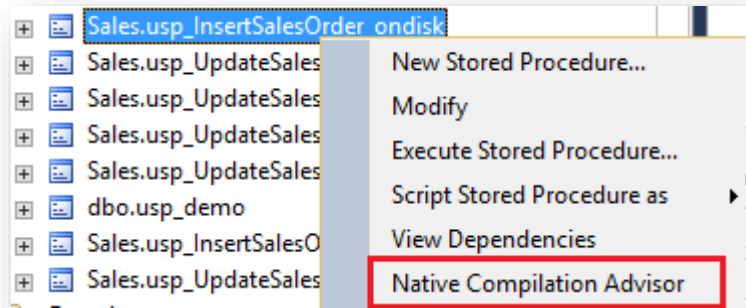
AdventureWorks2012

MDW




Primarily looks at *sys.dm_exec_procedure_stats*
Drill down to referencing objects

Native Compilation Advisor



TSQL Constructs Not Supported



Stored Procedure Validation Result

[Introduction](#)
[Stored procedure validation](#)
Unsupported Transact-SQL elements

The following is a list of Transact-SQL elements in your stored procedure that are not supported within native compilation. In order to enable native compilation for this stored procedure, you must resolve all items in this list. Some assistance for resolving these items are offered in this [link](#).

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], [Job Title], [RecursionLevel]) AS (SELECT e.[BusinessEntityID], e.[OrganizationNode], p.[FirstName], p.[LastName], e.[Job Title], 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

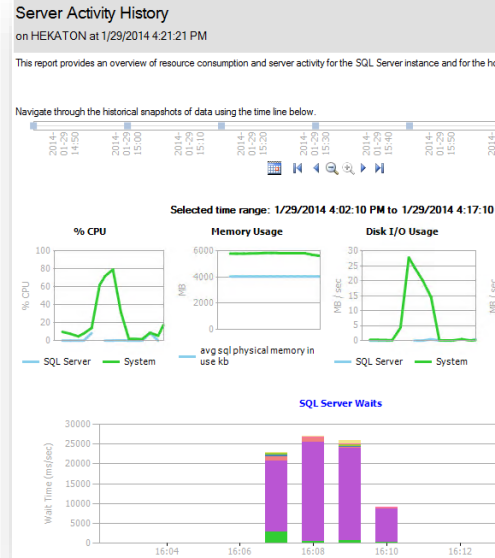
Other Tools

DMVs

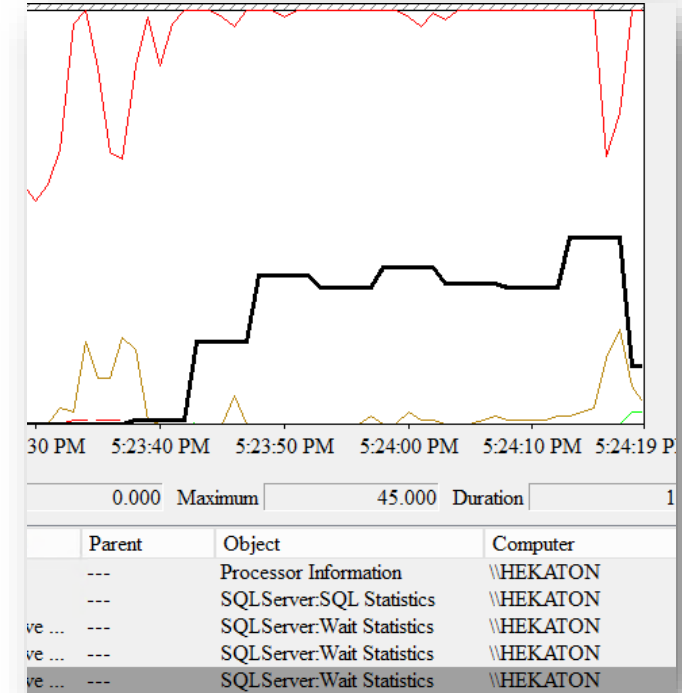
```
-- Check waits, No Latch waits.
WITH WaitCategoryStats ( wait_category, wait_type, wait_time_ms, waiting_tasks_count, max_wait_time_ms) AS
(
    SELECT
        CASE
            WHEN wait_type LIKE 'LCK%' THEN 'LOCKS'
            WHEN wait_type LIKE 'PAGEIOL%' THEN 'PAGE I/O LATCH'
            WHEN 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
```

```
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

