Hekaton: SQL Server's Memory-Optimized OLTP Engine

Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson,
Pravin Mittal, Ryan Stonecipher, Nitin Verma, Mike Zwilling
Microsoft
{cdiaconu, craigfr, eriki, palarson, pravinm, ryanston, nitinver, mikezw}@microsoft.com

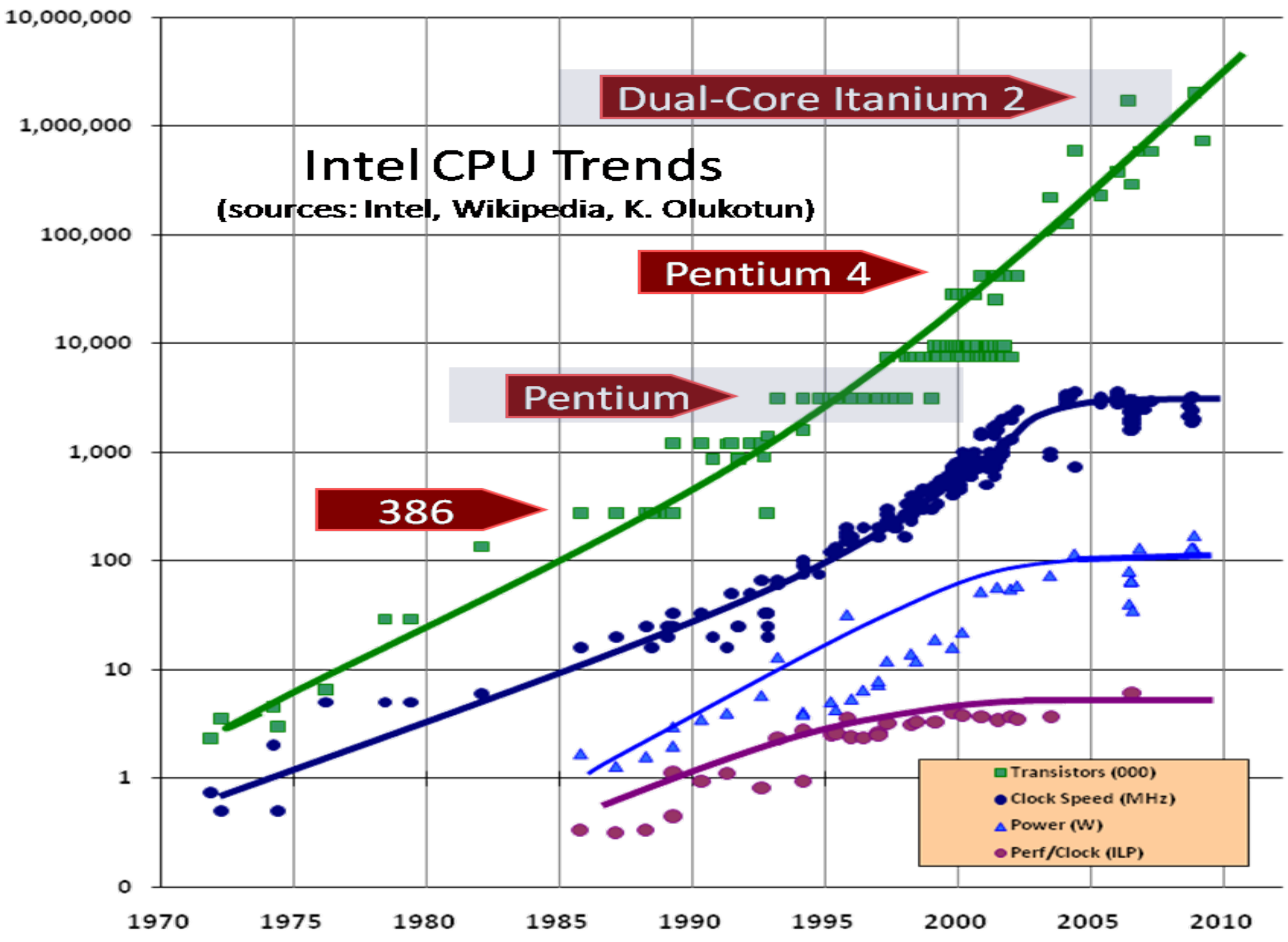
Presented by: Prateek Gulati

Agenda

- Why do you need in-memory processing?
- Hekaton engine overview
- How it is done
- Benefits
- Limitations

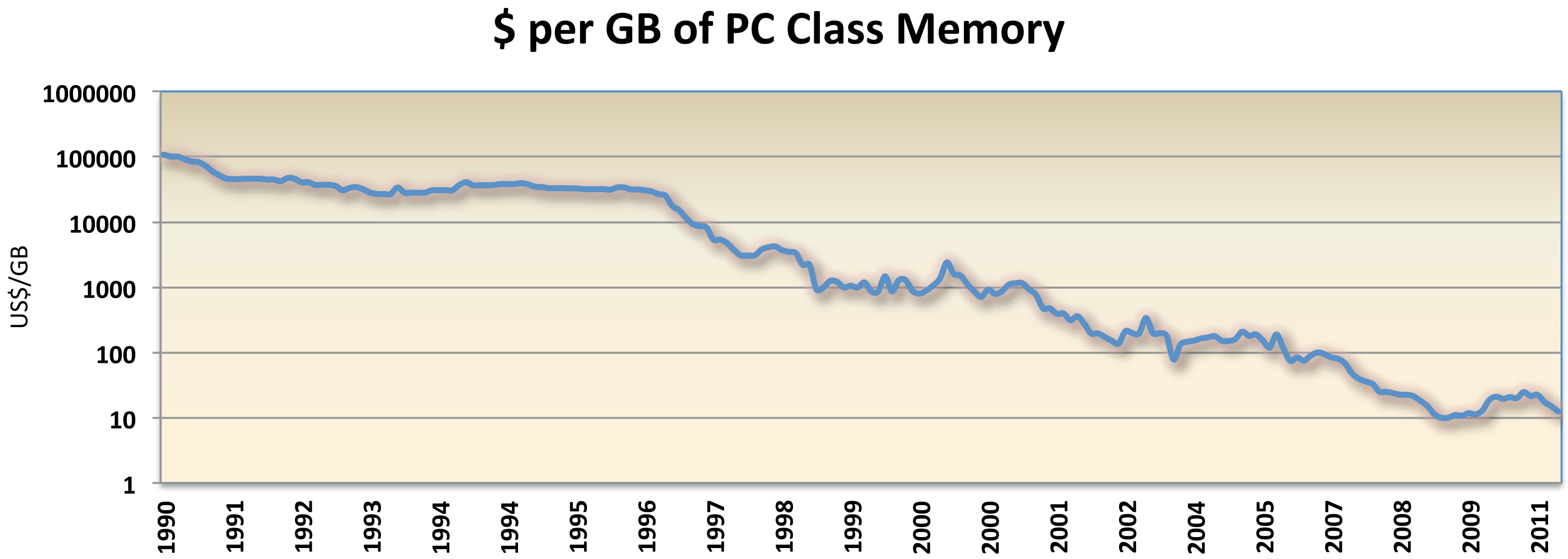
Industry Trends: CPU

- Computing power holds Moore Law due to parallelism
- CPU clock frequency stalled
- Parallel processing has its limits due to lock contention



Industry Trends: RAM

- RAM prices continue to fall
- Servers have HUGE memory
- DDR4 expected to hit mainstream in 2014-2015
- Traditional page based architecture has limitations, even when all pages are in memory

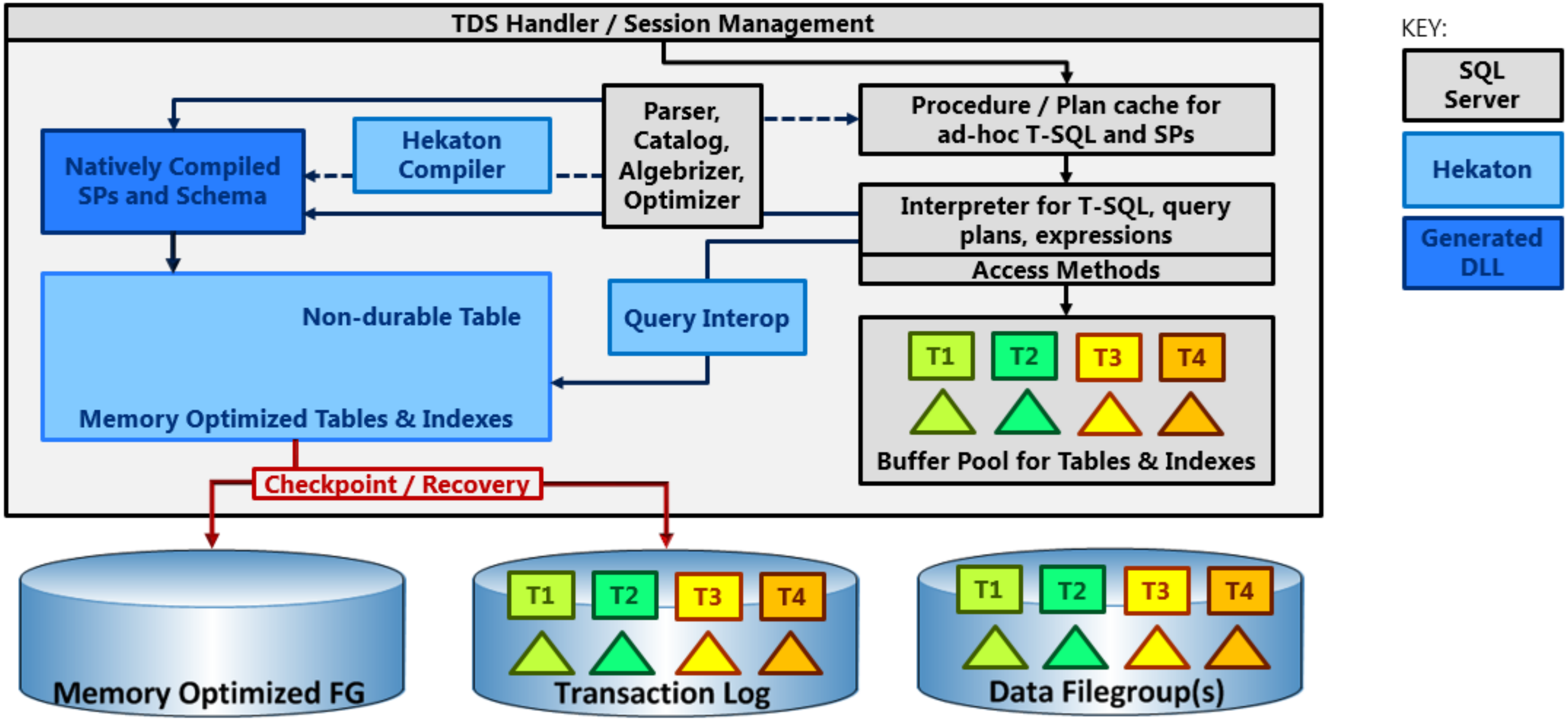


Hekaton-In-memory OLTP engine Architecture

Architectural Pillars

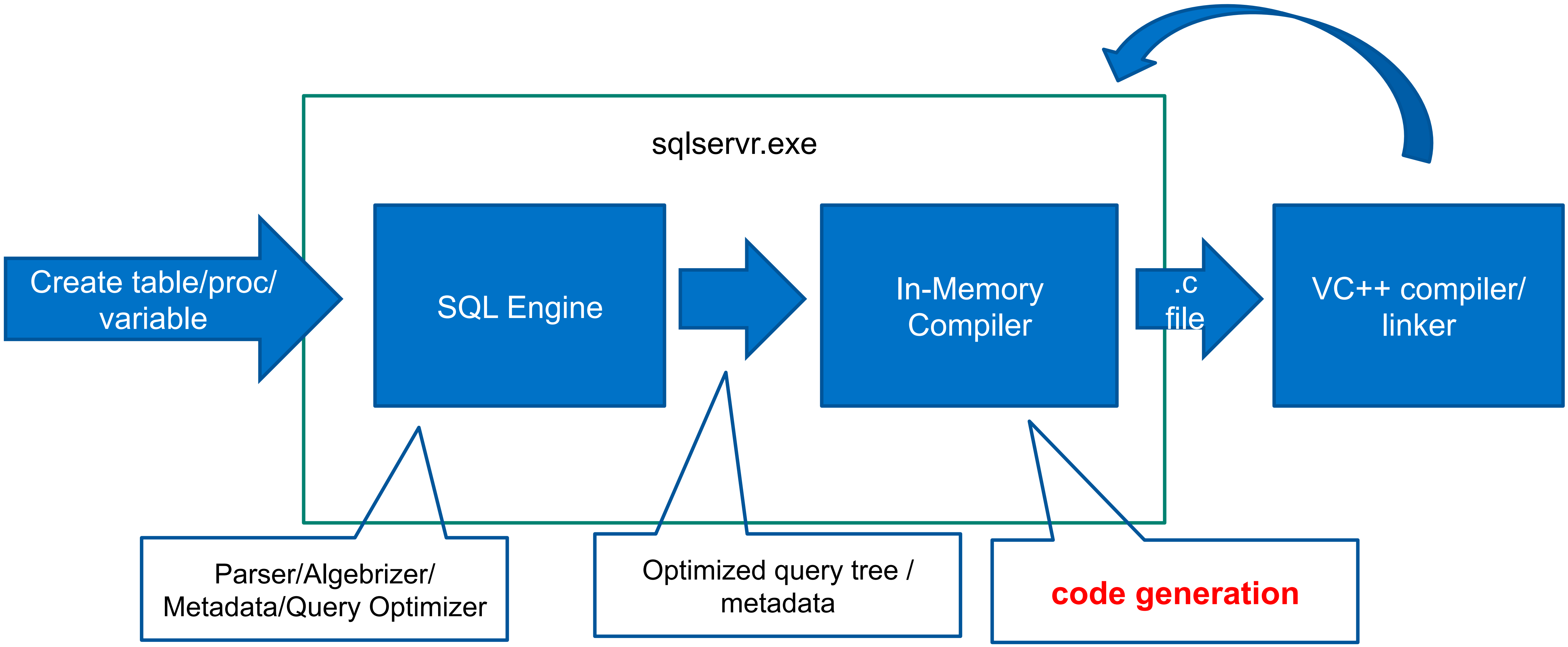
Main-Memory Optimized	T-SQL Compiled to Machine Code	High Concurrency	SQL Server Integration
• Optimized for in-memory data • Memory optimized Indexes (hash and range) exist only in memory • No buffer pool, B-trees • Stream-based storage • Transaction log optimization (block writes, no undo)	• T-SQL compiled to machine code via C code generator and VC • Invoking a procedure is just a DLL entry-point • Aggressive optimizations at compile-time	• Multi-version optimistic concurrency control (MVCC) with full ACID support • Core engine uses non blocking lock-free algorithms • No lock manager, latches or spinlocks • No TempDB	• Same manageability, administration & development experience • Integrated queries & transactions • Integrated backup/ restore • If SQL Server crashes data is fully recoverable.

Hekaton Integration with SQL Server



Native Compilation Process

Compile T-SQL statements ***and*** table data access logic into machine code



Native Compiled Stored Procedures

Interpreted T-SQL Access

- Access both memory- and disk-based tables
- Less performant
- Virtually all T-SQL functions supported
- **When to use**
  - Ad hoc queries
  - Reporting-style queries
  - Speeding up app migration

Natively Compiled Procs

- Access only memory optimized tables
- Maximum performance
- Limited T-SQL functions supported
- **When to use**
  - OLTP-style operations
  - Optimize performance critical business logic
  - More the logic embedded, better the performance improvement

In-Memory OLTP Structures summary

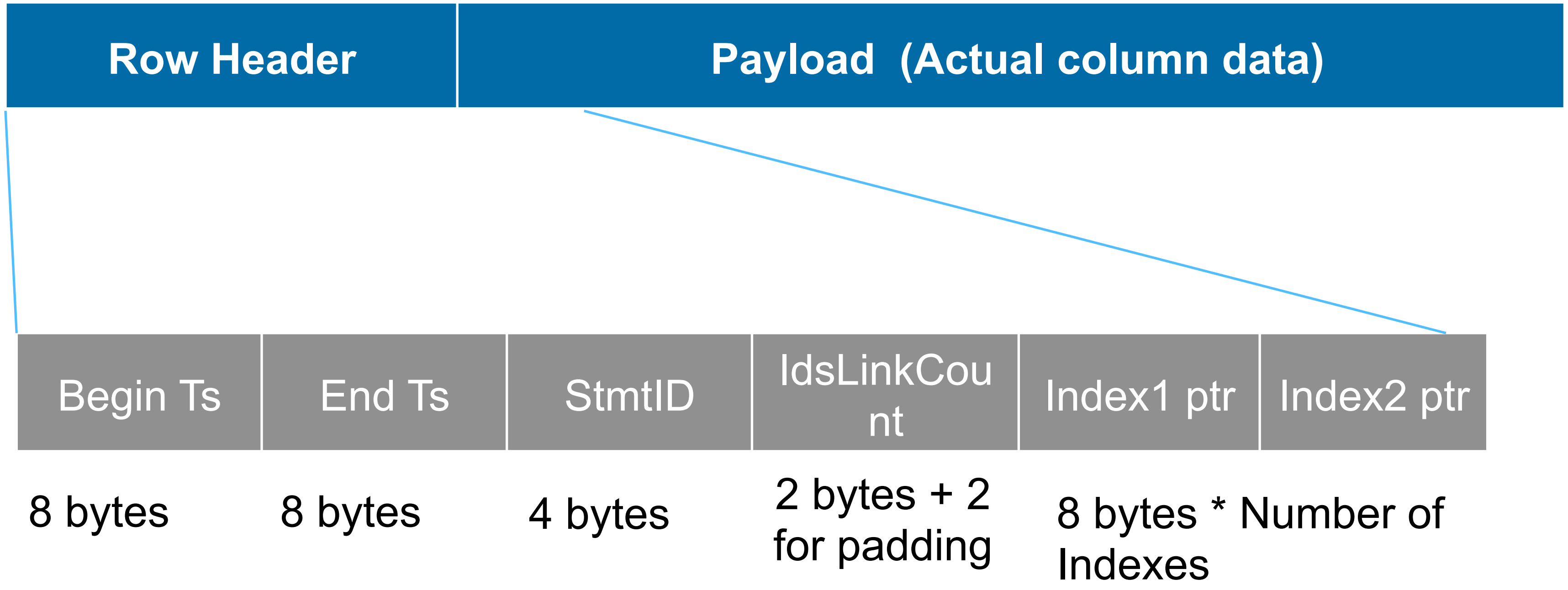
Rows

- Row structure is optimized for memory access
- There are no Pages
- Rows are versioned and there are no in-place updates
- Fully durable by default (but they don’t have to be)

Indexes

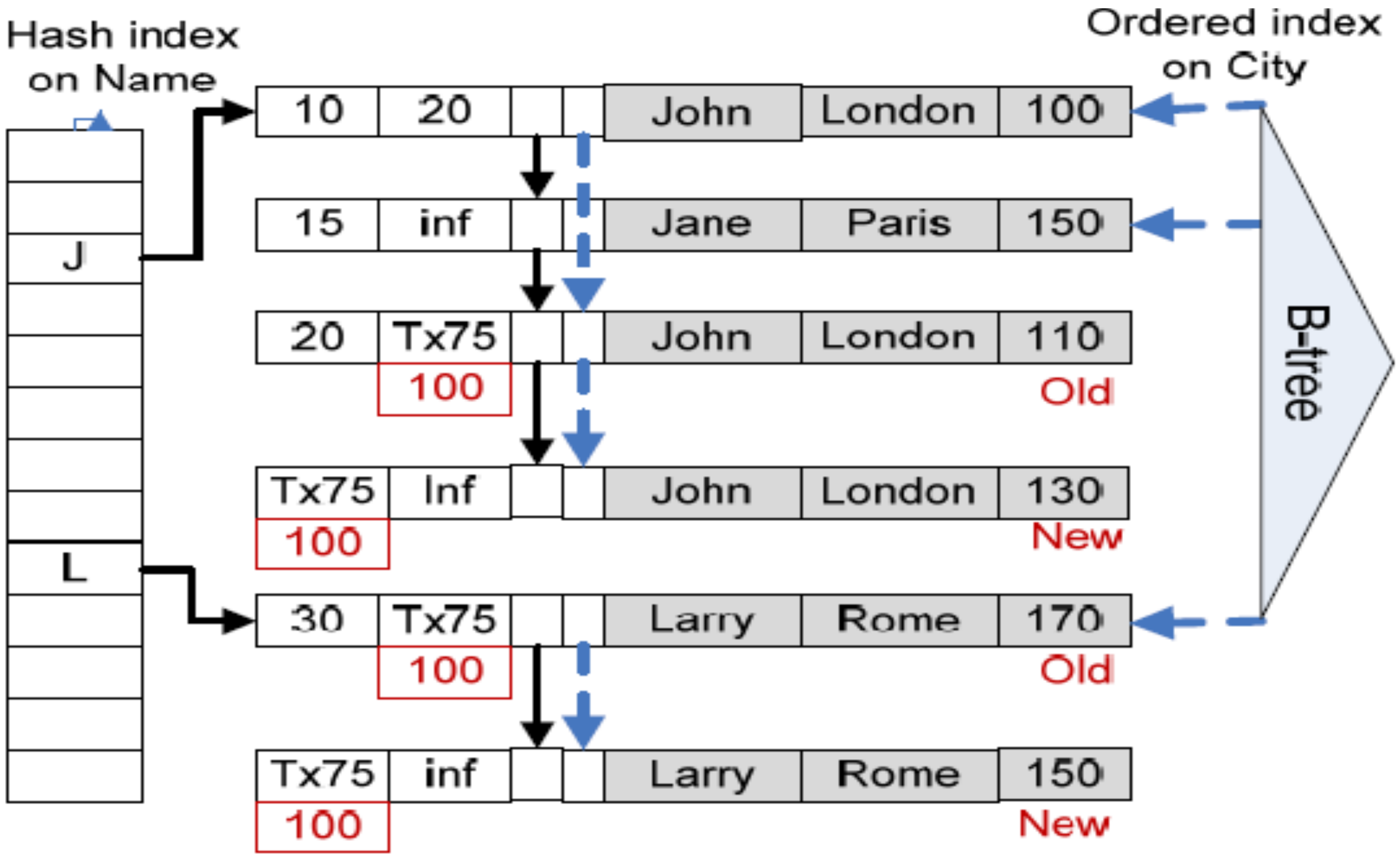
- There is no clustered index, only non-clustered indexes
- Indexes point to rows, access to rows is via an index
- Indexes do not exist on disk, only in memory, recreated during recovery
- Hash indexes for point lookups
- Range indexes for ordered scans and Range Scans

In-Memory Row Format



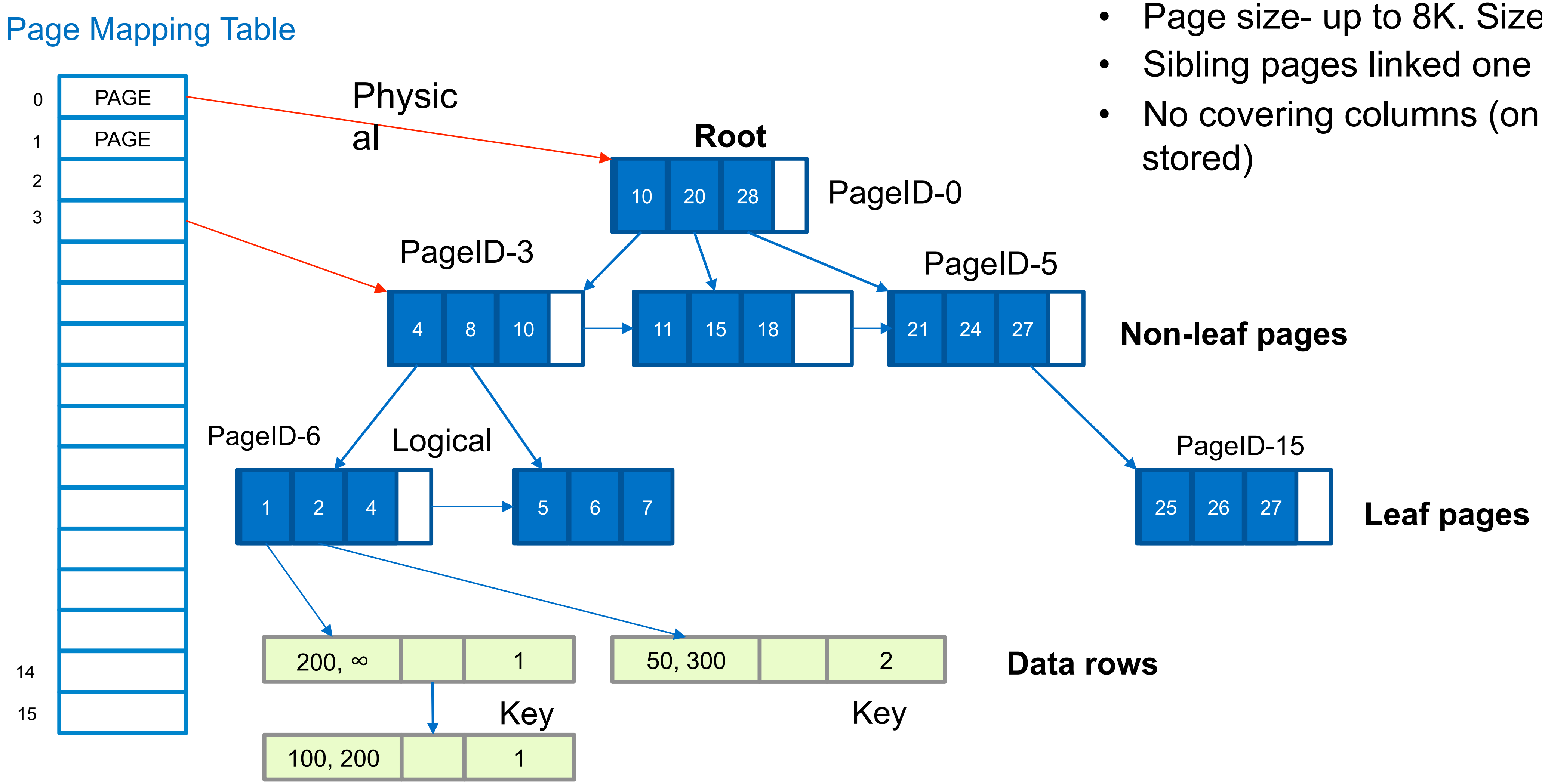
- Begin/End timestamp determines row's version validity and visibility
- No concept of data pages, only rows exist
- Row size limited to 8060 bytes (@table create time) to allow data to be moved to disk-based tables
- Not every SQL table schema is supported (Ex: LOB and SqlVariant)

Hash Indexes

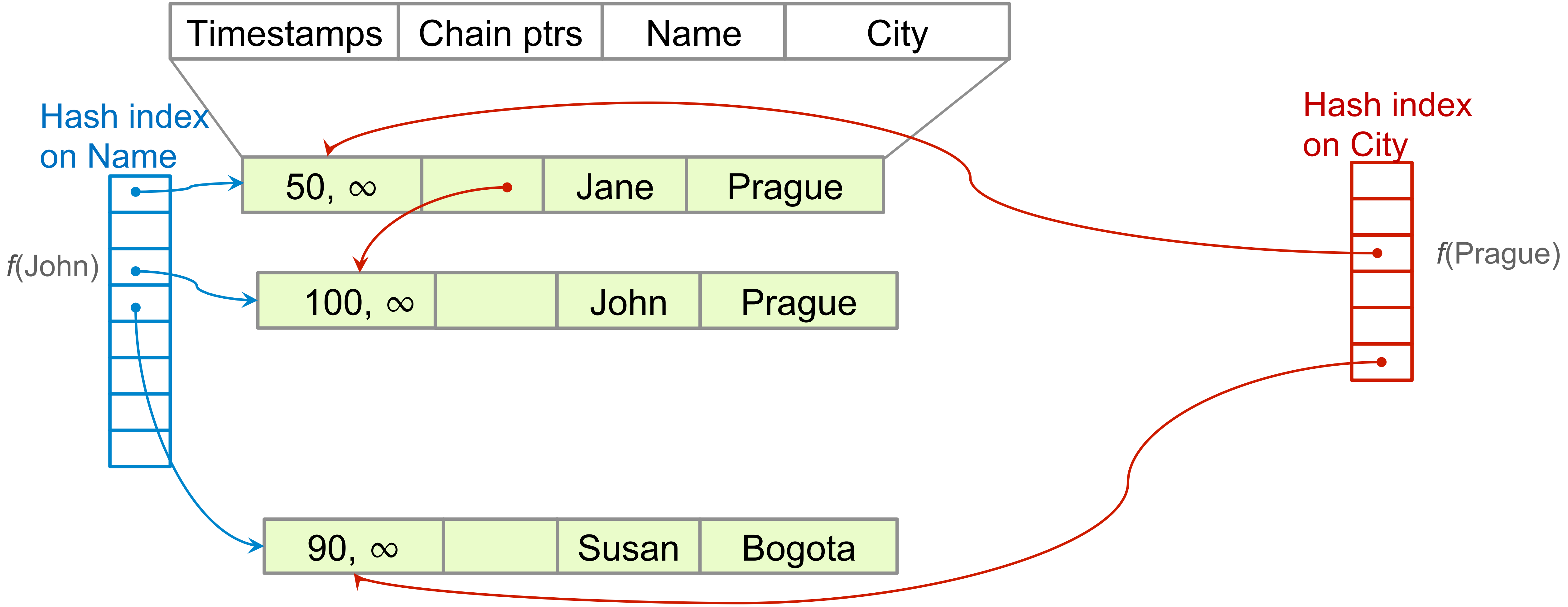


Non Clustered (Range) Index

- No latch for page updates
- No in-place updates on index pages
- Page size- up to 8K. Sized to the row
- Sibling pages linked one direction
- No covering columns (only the key is stored)

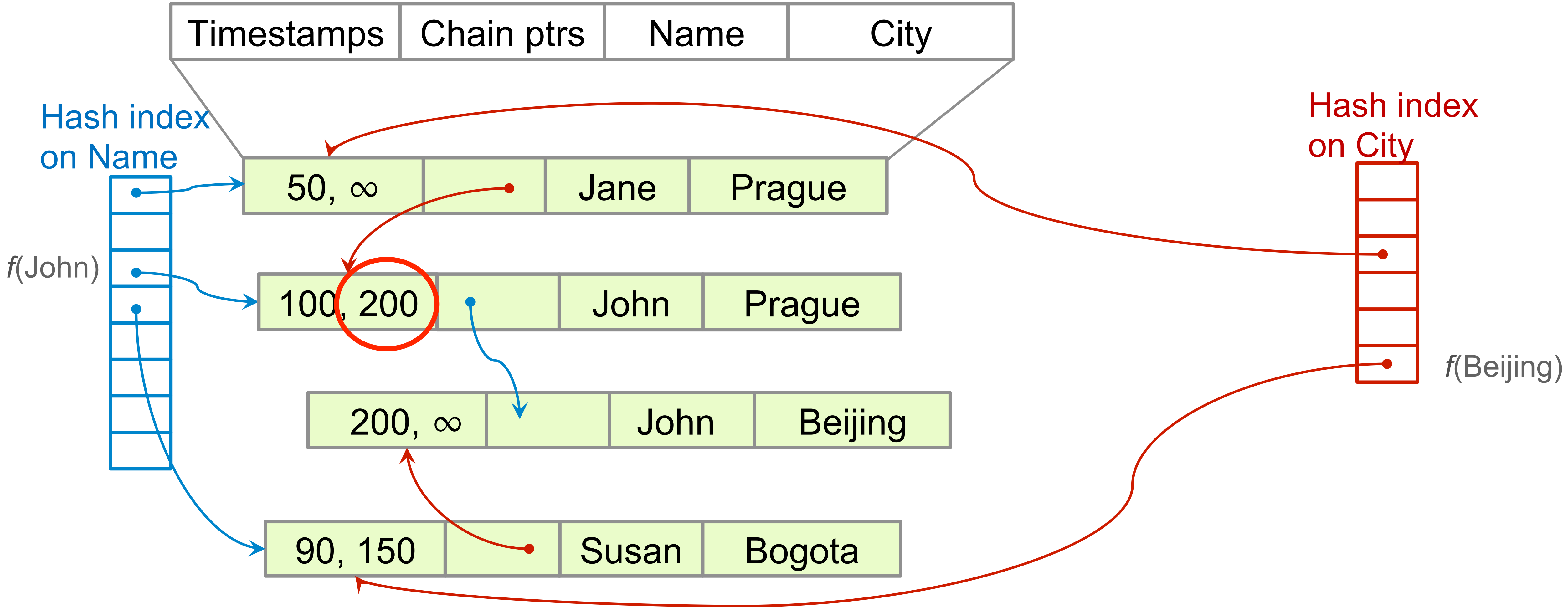


Memory Optimized Table Insert



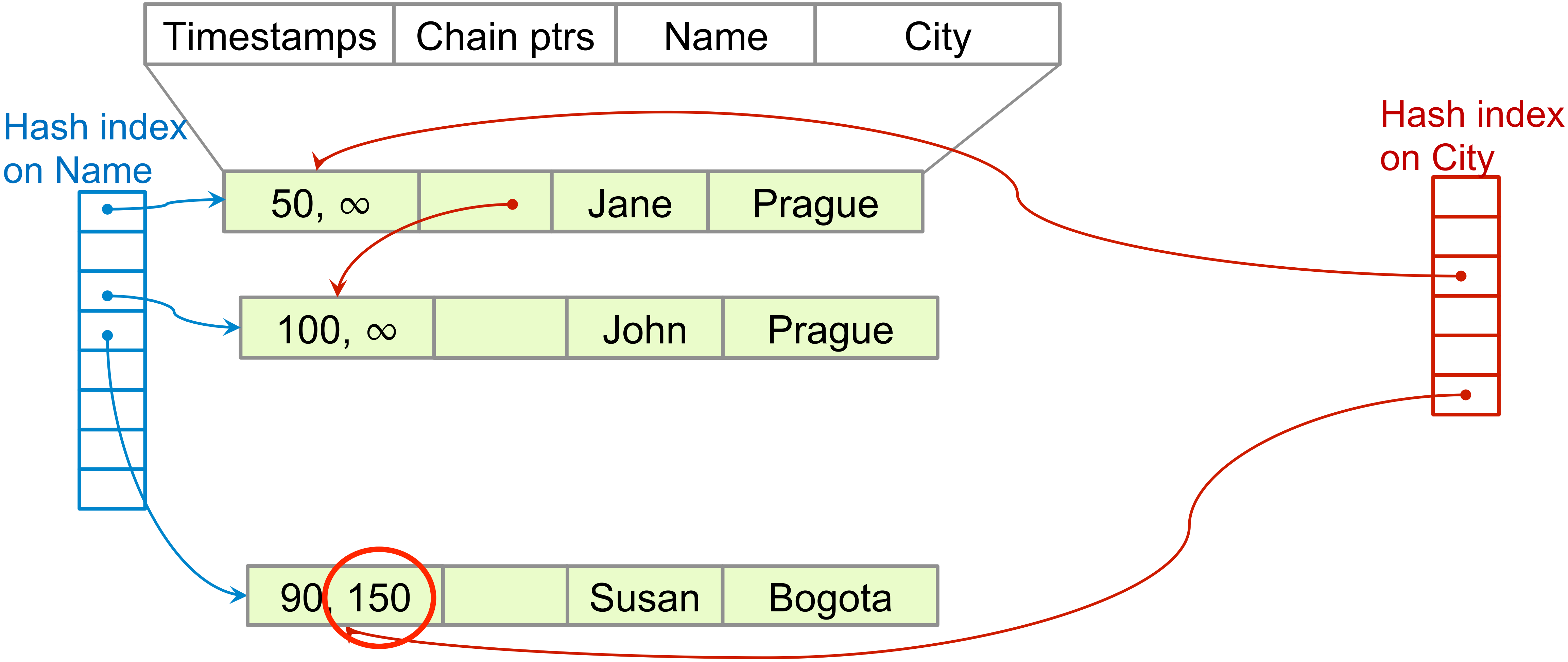
T100: INSERT (John, Prague)

Memory Optimized Table Update



T200: UPDATE (John, Prague) to (John, Beijing)

Memory Optimized Table Delete



T150: DELETE (Susan, Bogota)

Transaction Durability

- Transaction durability is ensured to allows system to recover memory-optimized table after a failure.
- Log streams contain the effects of committed transactions logged as insertion and deletion of row versions
- Checkpoint streams come in two forms:
  - a) data streams which contain all inserted versions during a timestamp interval,
  - b) delta streams, each of which is associated with a particular data stream and contains a dense list of integers identifying deleted versions for its corresponding data stream
- Hekaton table can be durable or non-durable
- Stored in a single memory-optimized FILEGROUP based on FILESTREAM implementation
- Sequential IO pattern (no random IO)

Transaction Logging

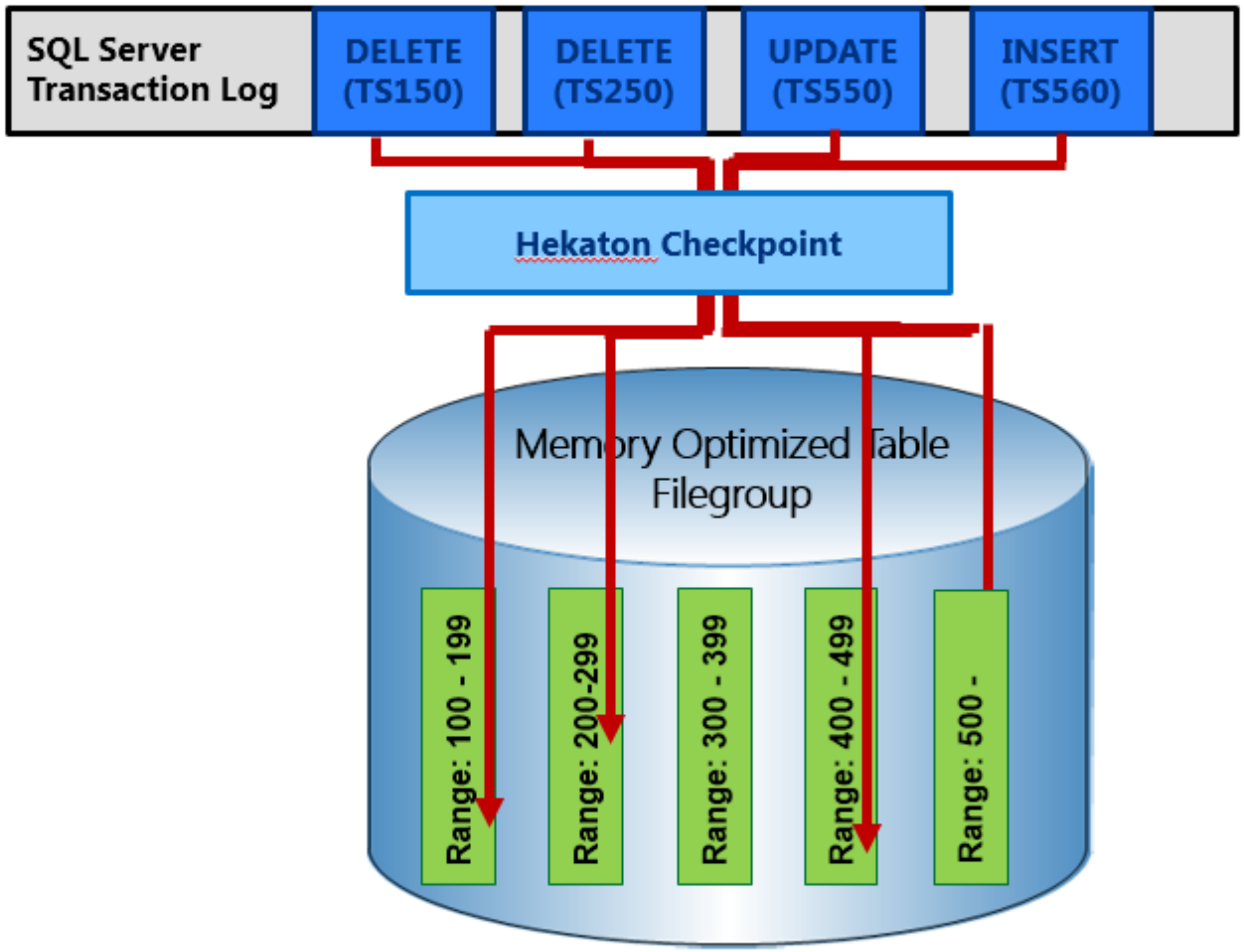
- Uses database's transaction log to store content
- Each Hekaton log record contains a transaction log record header, followed by Hekaton-specific log content
- All logging in Hekaton is logical
  - No physical log records for physical structure modifications
  - No index-specific / index-maintenance log records
  - Redo-only log records in transaction log

Checkpoints

Hekaton Checkpoint

- Not tied to recovery interval or SQL checkpoint. Has its own log truncation
- Gets triggered when generated log exceeds a threshold (1GB) or internal min time-threshold has crossed since last checkpoint or manual checkpoint
- Checkpoint is a "set of {Data, Delta} files and checkpoint file inventory to apply transaction log from"

Populating Data / Delta files



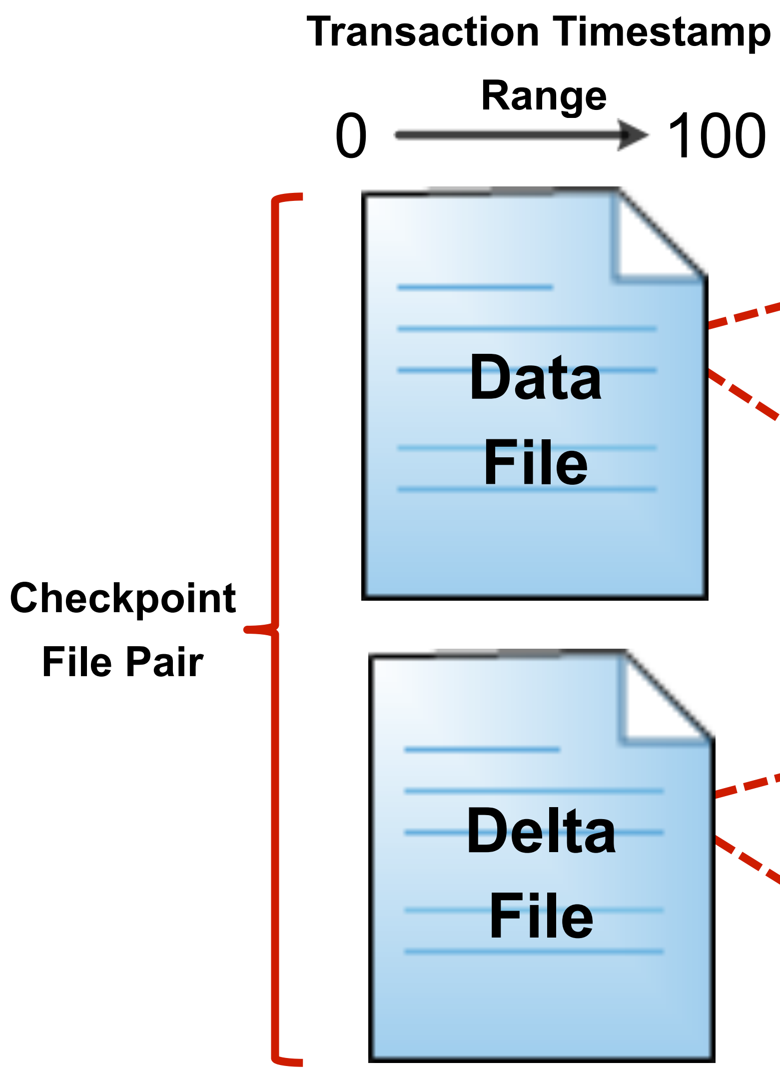
Data files:

- Pre-allocated size (128 MB)
- Hekaton Engine switches to new data file when it estimates that current set of log records will fill the file
- Stores only the inserted rows
- Indexes exist only in memory, not on disk
- Once a data file is closed, it becomes read-only

Delta files:

- File size is not constant, write 4KB pages over time
- Stores IDs of deleted rows

Data / Delta Files



Timestamp (INSERT)	TableID	RowID	Payload
Timestamp (INSERT)	TableID	RowID	Payload
Timestamp (INSERT)	TableID	RowID	Payload

Data file contains rows inserted within a given transaction range

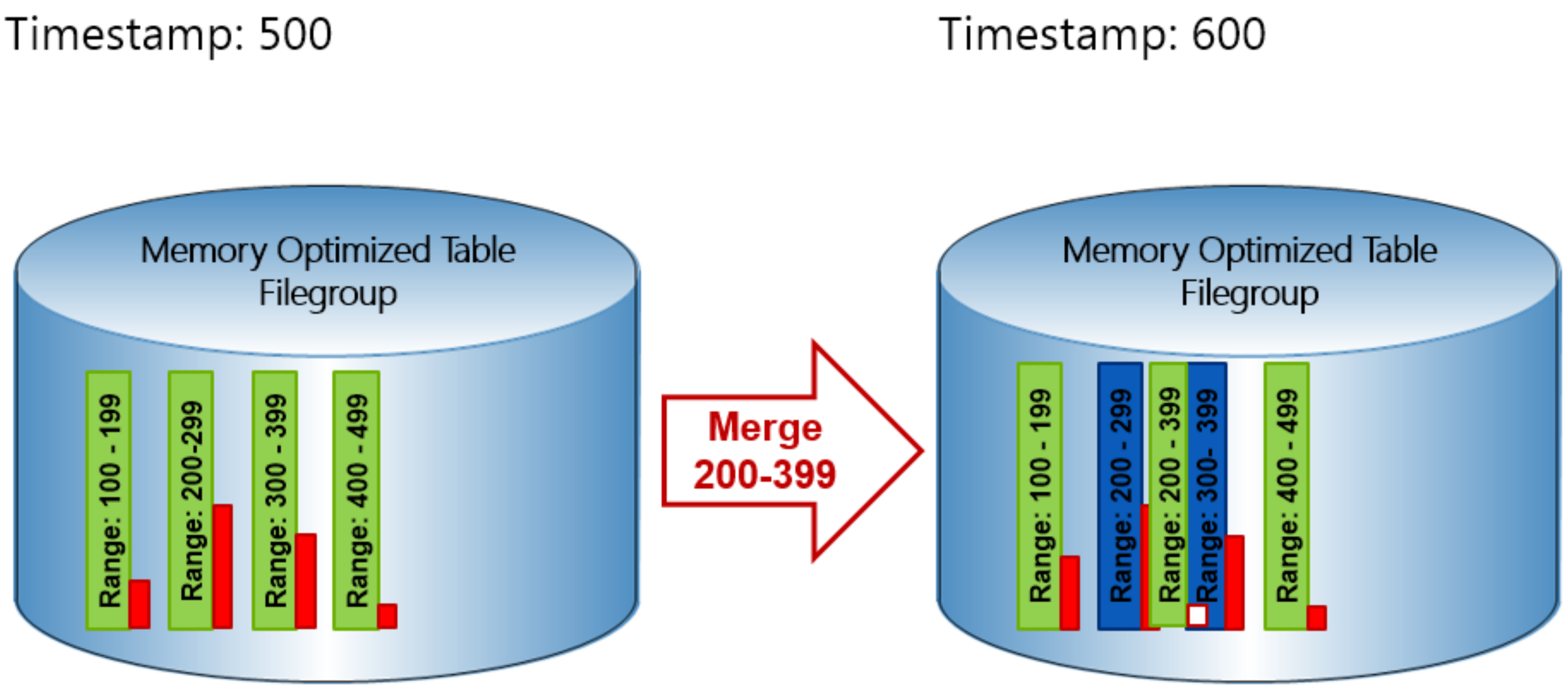
Timestamp (INSERT)	Timestamp (DELETE)	RowID
Timestamp (INSERT)	Timestamp (DELETE)	RowID
Timestamp (INSERT)	Timestamp (DELETE)	RowID

Delta file contains deleted rows within a given transaction range

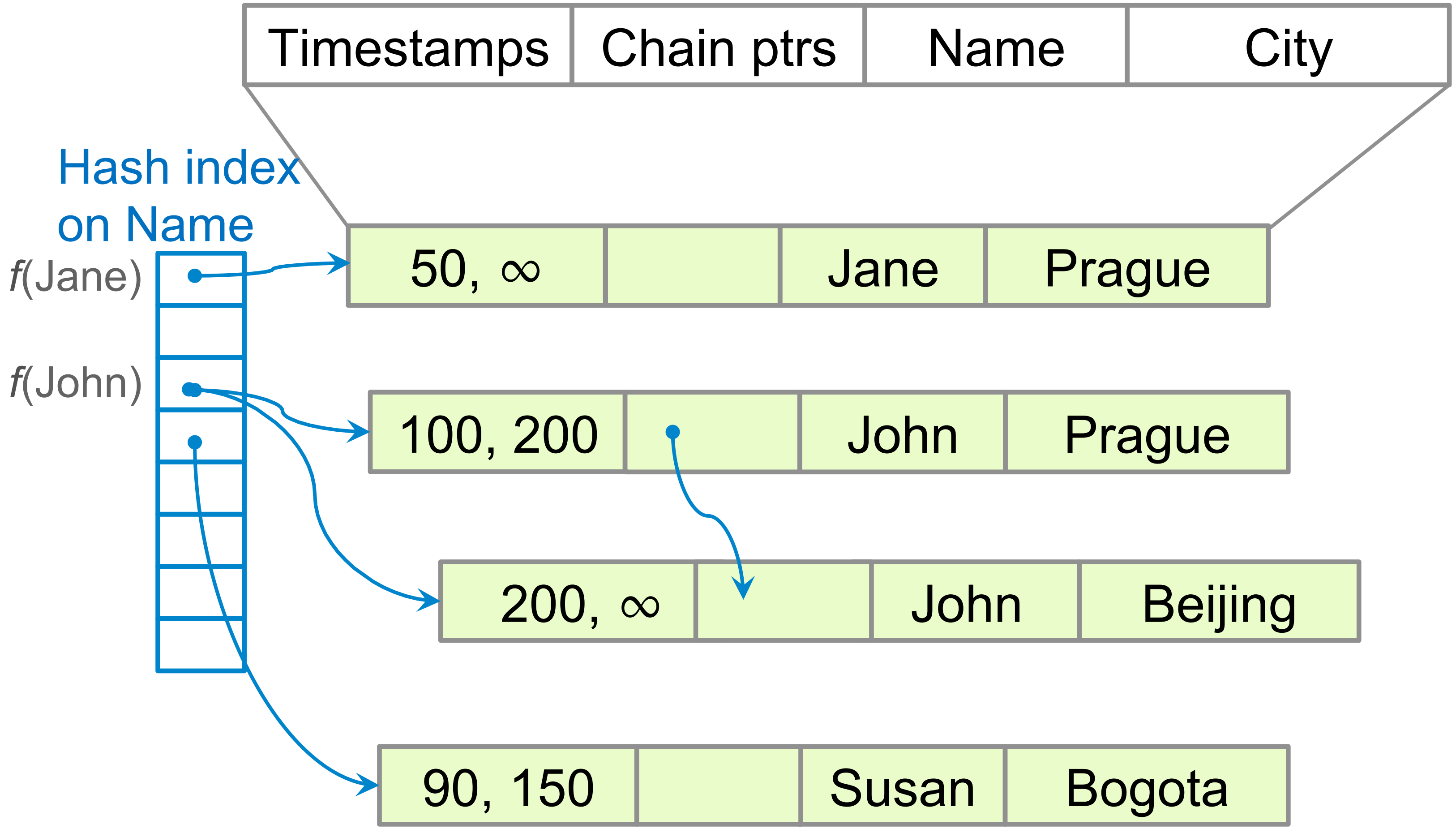
Merge Operation

- Merges 2+ adjacent data / delta files pairs into 1 pair
- Need for merge - Deleting rows causes data files to have stale rows
- Manual checkpoints closes file before it is “full”
- Reduces storage required to “store” active data rows
- Improves the recovery time
- Stored Procedure provided to invoke merge manually

Merge Operation



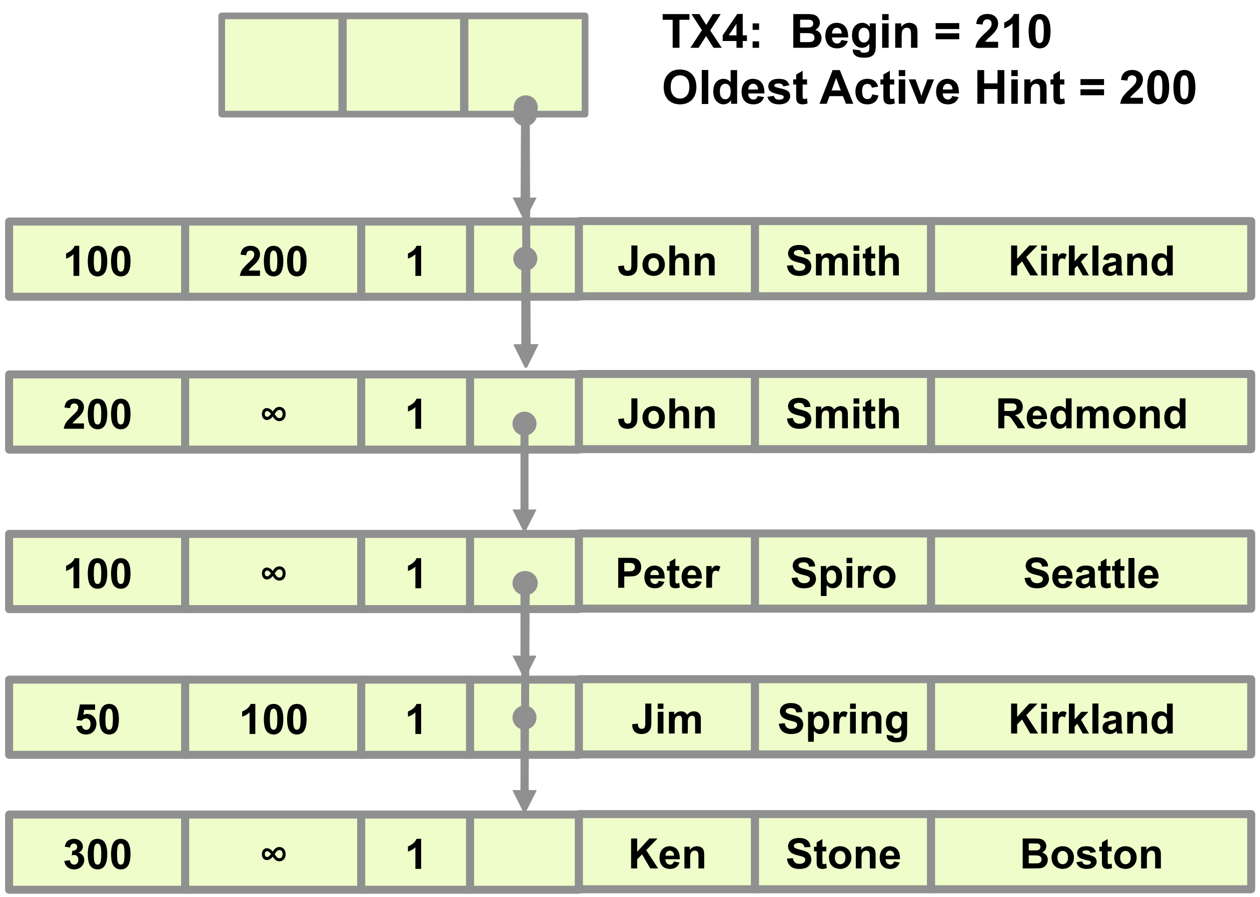
Garbage Collection

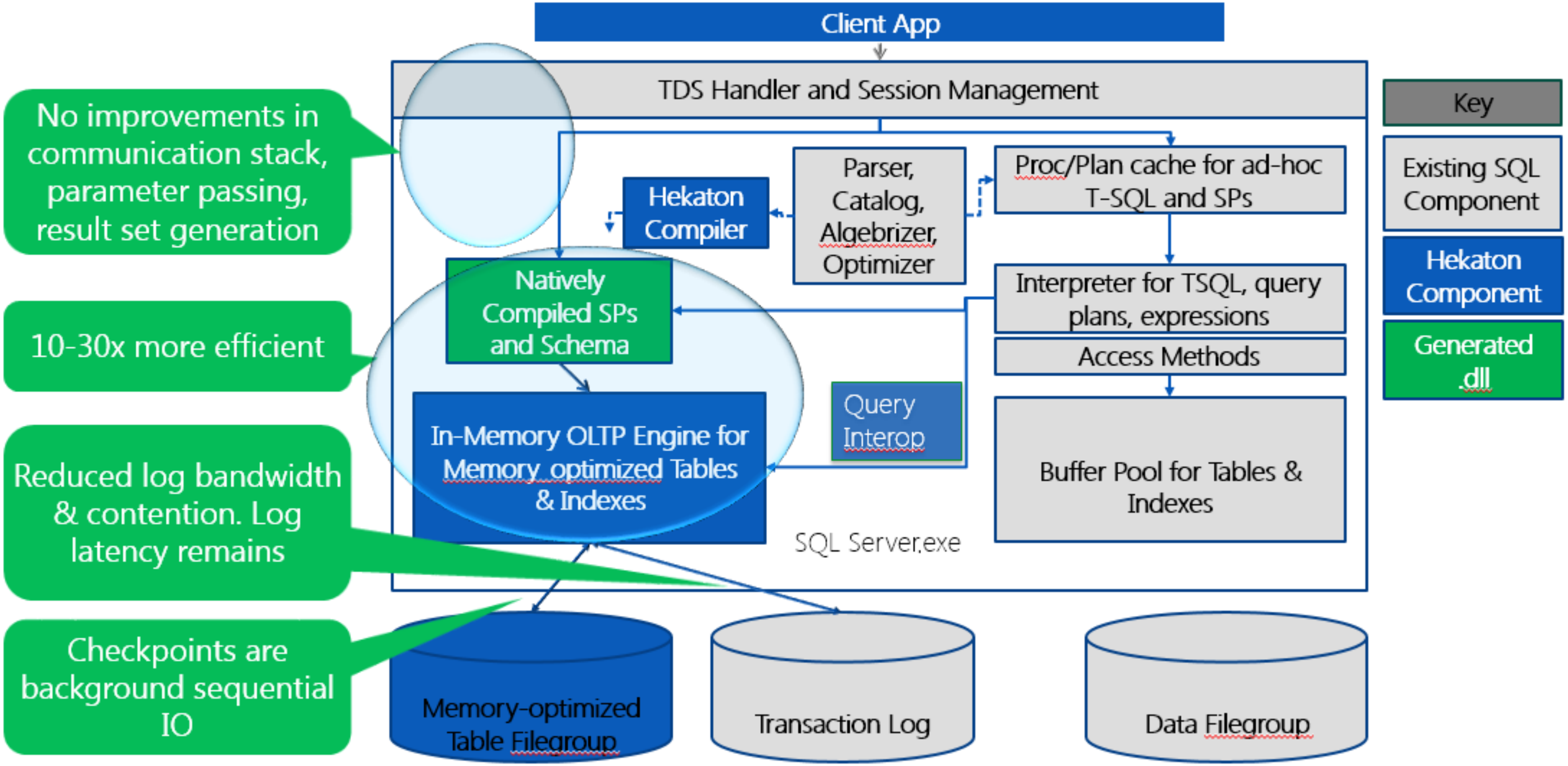


T250: ***Garbage collection***

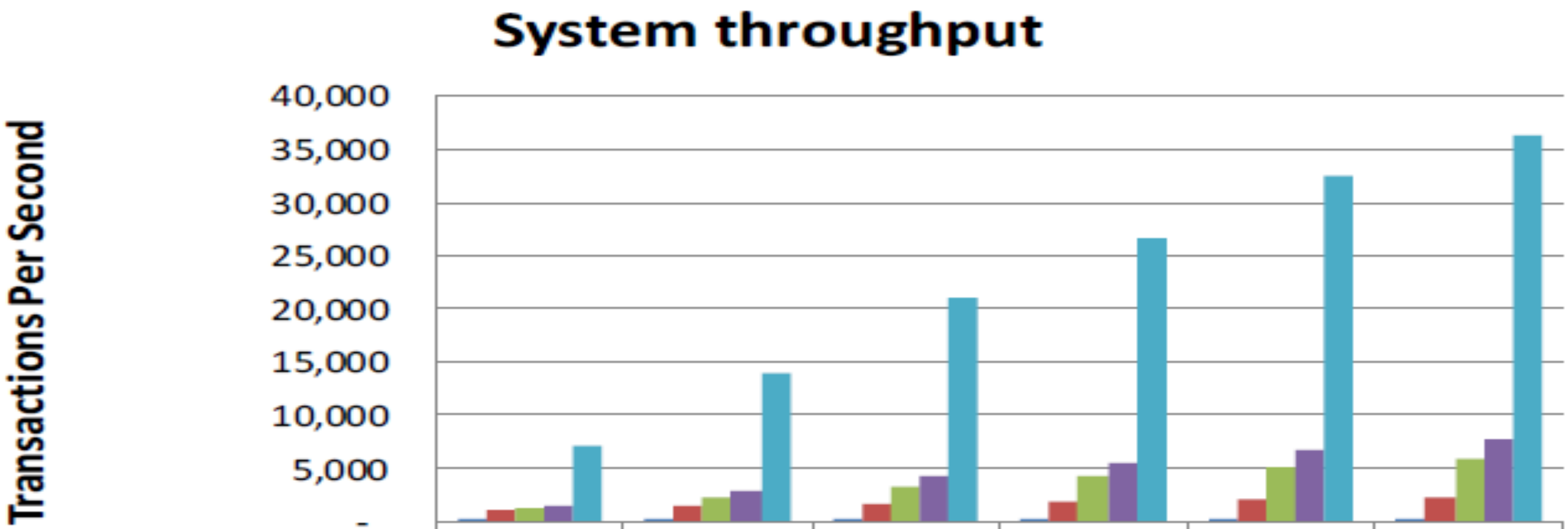
Cooperative Garbage Collection

- Scanners can remove expired rows when they come across them
- Offloads work from GC thread
- Ensures that frequently visited areas of the index are clean of expired rows




Client App
TDS Handler and Session Management
No improvements in communication stack, parameter passing, result set generation
Hekaton Compiler
Parser, Catalog, Algebrizer, Optimizer
Proc/Plan cache for ad-hoc T-SQL and SPs
Natively Compiled SPs and Schema
Interpreter for TSQL, query plans, expressions
Access Methods
10-30x more efficient
Query Interop
In-Memory OLTP Engine for Memory_optimized Tables & Indexes
Buffer Pool for Tables & Indexes
Reduced log bandwidth & contention. Log latency remains
SQL Server.exe
Checkpoints are background sequential IO
Memory-optimized Table Filegroup
Transaction Log
Data Filegroup
Key
Existing SQL Component
Hekaton Component
Generated .dll

Hekaton Engine’s Scalability



	1	2	3	4	5	6
Number of cores	2	4	6	8	10	12
SQL with contention	984	1,363	1,645	1,876	2,118	2,312
SQL without contention	1,153	2,157	3,161	4,211	5,093	5,834
Interop	1,518	2,936	4,273	5,459	6,701	7,709
Native	7,078	13,892	20,919	26,721	32,507	36,375

Memory Optimized Table Limitations

Optimized for high-throughput OLTP

- No XML and no CLR data types

Optimized for in-memory

- Rows are at most 8060 bytes
- No Large Object (LOB) types like varchar(max)
- Durable memory-optimized tables are limited to 512 GB. (Non-durable tables have no size limit.)

Scoping limitations

- No FOREIGN KEY and no CHECK constraints
- No schema changes (ALTER TABLE) – need to drop/recreate table
- No add/remove index – need to drop/recreate table
- No Computed Columns
- No Cross-Database Queries

Natively Compiled Procedures Restrictions

- Not all operators/TSQLs are supported
- Only Nested Loop join, no TSQL MERGE or EXISTS, cursors, nested queries
- No CASE statement, CTEs, user-defined functions, UNION statement, DISTINCT statement
- Transaction isolation level
- SNAPSHOT, REPEATABLEREAD, and SERIALIZABLE
- READ COMMITTED and READ UNCOMMITED is not supported
- Cannot access disk-based tables
- No TEMPDB! Use In-Memory Table variables
- No automatic recompile on statistics changes
- Need to stop & start SQL or drop & create procedure

Create Filegroup

FileGroup Container

```
CREATE DATABASE [Hekaton]
ON PRIMARY
(NAME = N'Hekaton_data', FILENAME = N'C:\Data\Data\Hekaton_data.mdf'),
FILEGROUP [Hekaton_InMemory] CONTAINS MEMORY_OPTIMIZED_DATA
(NAME = N'Hekaton_mem', FILENAME = N'C:\Data\Mem\Hekaton_Lun1.mdf')
LOG ON
(NAME = N'Hekaton_log', FILENAME = N'C:\Data\Log\Hekaton_log.ldf')


ALTER DATABASE [Hekaton]
ADD FILE (NAME = N'Hekaton_mem', FILENAME = N'C:\Data\Mem
\Hekaton_Lun2.mdf')
TO FILEGROUP [Hekaton_InMemory]
```

Create Memory Optimized Table

```
CREATE TABLE [Customer] (
[CustomerID] INT NOT NULL
PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1000000),
[AddressID] INT NOT NULL INDEX [IxName] HASH WITH (BUCKET_COUNT =
1000000),
[LName] NVARCHAR(250) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IXLName] NONCLUSTERED (LName)
)
WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA);
```

Hash Index

Collation BIN2

Range Index

This table is memory optimized

This table is durable

References

http://www.enterprisetech.com/2014/03/18/microsoft-turbocharges-transactions-hekaton-memory/

http://blogs.msdn.com/b/sqlcat/archive/2013/06/25/sql-server-in-memory-oltp-internals-overview-for-ctp1.aspx

http://blogs.msdn.com/b/arvindsh/archive/2013/07/03/sql-2014-in-memory-oltp-hekaton-training-videos-and-white-papers.aspx

https://www.simple-talk.com/sql/database-administration/exploring-in-memory-oltp-engine-(hekaton)-in-sql-server-2014-ctp1/

http://www.sqlskills.com/blogs/bobb/category/hekaton/

http://thomaslarock.com/2013/08/sql-server-2014-in-memory-oltp-hekaton-useful-links

http://blogs.msdn.com/b/carlnol/archive/2013/09/16/implementing-lob-storage-in-memory-optimized-tables.aspx

http://www.sqlpassion.at/archive/2013/08/12/extreme-transaction-processing-xtp-hekaton-the-solution-to-everything/

http://mattsql.wordpress.com/2013/07/08/in-memory-oltp-with-sql-server-2014/

http://research.microsoft.com/en-us/news/features/hekaton-122012.aspx

www.slideshare.net/RaviOkade/

Thank you for your time!

Q&A