Hekaton

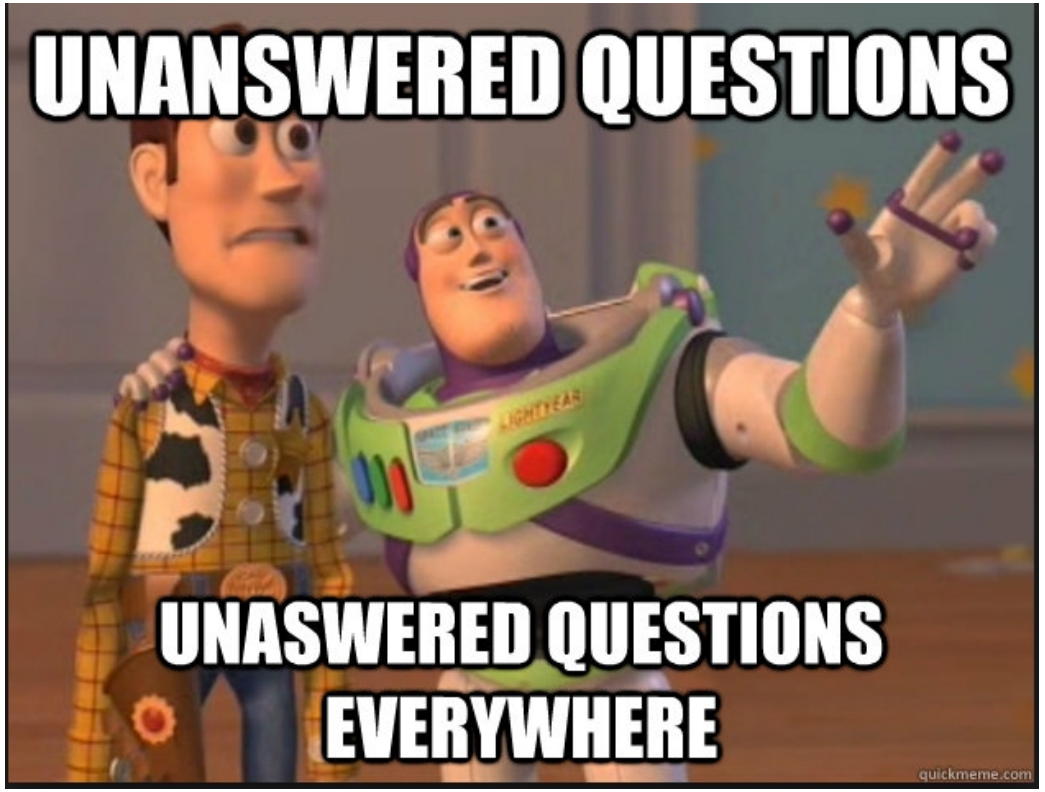
Biswarup Ghosh

Agenda

- Rationale behind In memory database
- Heckaton overview
- Hekaton deep dive

Why In memory databases??

What has changed ??

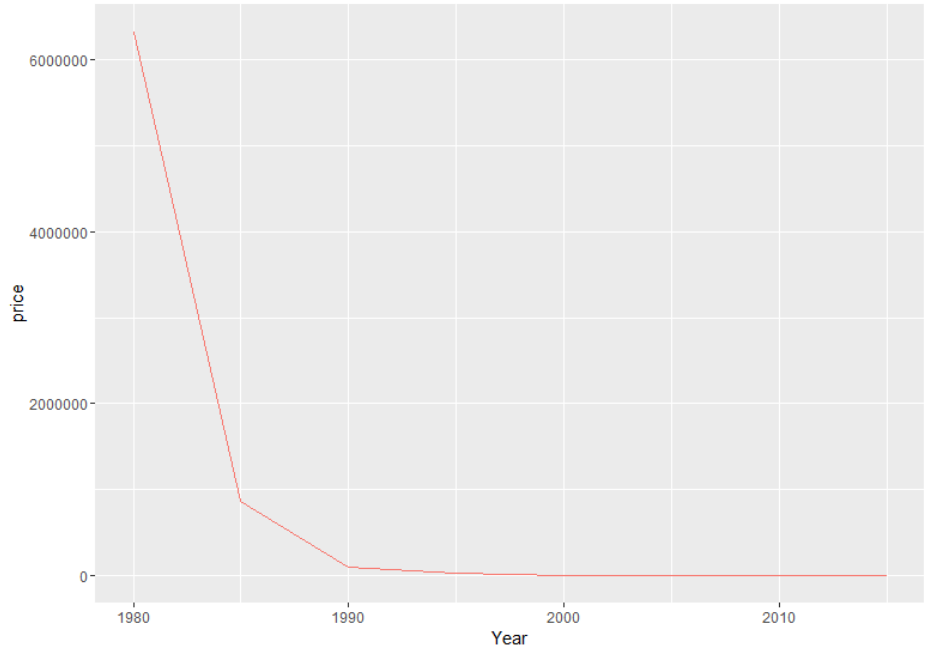


Business Trends

- Lower latency
- Higher Throughput
- Lower Cost

Technical Trends

- Drastic drop of memory price over the years
  - from $6,328,125 in 1980 per gb to $4.37 in 2015- www.statisticbrain.com/average-historic-price-of-ram
- CPU clock speed reached a plateau



What is Hekaton

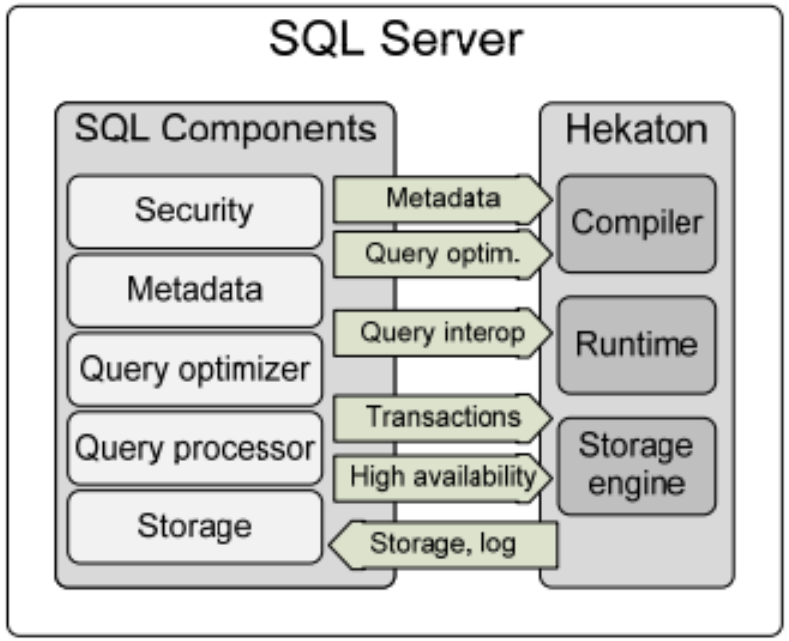
SQL Server’s Memory optimized OLTP engine

Why Memory optimized

Select student_address from student_table where student_id =123

Heckaton Design and Architechture overview

High level Architecture



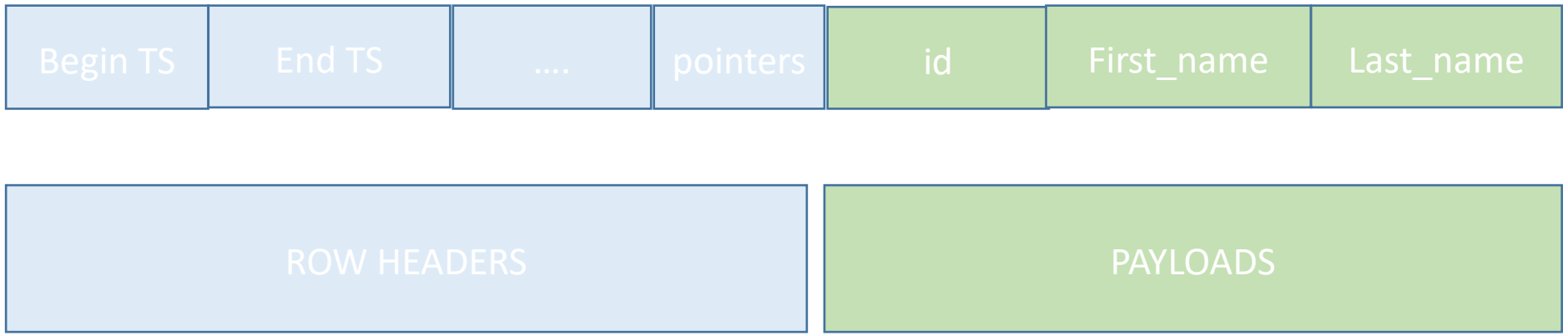
Data storage in hekaton 101

How the table creation in hekaton works

```
Create table student(
Id int not null,
first_name varchar(100),
last_name varchar(100))
With (MEMORY_OPTIMIZED = ON,DURABILITY = SCHEMA_AND_DATA)
```

- Lob data types ,xml ,clr types are not supported
- Tables ,indexes cant be altered once created
- Foreign key constraints are not supported
- Need to have atleast one index,either hash based or range based non clustered index

Storage format



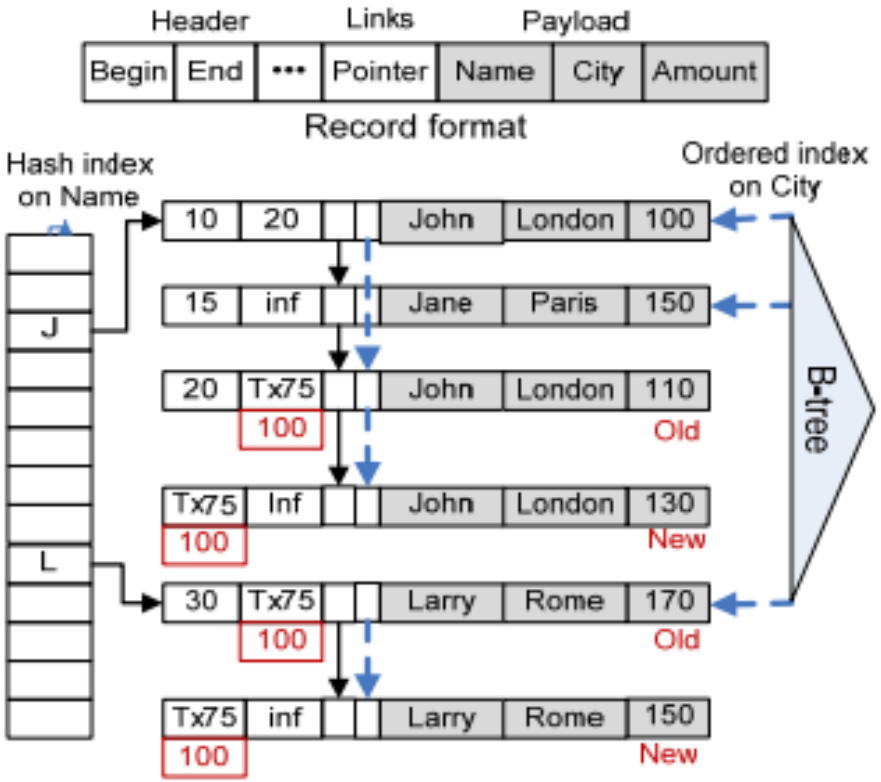


Figure 2: Example account table with two indexes. Transaction 75 has transferred $20 from Larry's account to John's account but has not yet committed.

- Black arrow indicates versions which are linked together by the hash index
- Blue dotted arrow indicates If multiple records have the same key value, the duplicates are linked together using the second link field in the records

How read works

- For a particular read operation , versions which have overlapping duration with the ***read***'s time are only fetched

How update/Insert works

- Update creates a new version with begin time stamp as the time stamp of the transaction
- The previous version’s end time stamp is updated in this process by the same
- Insert creates a new version with begin time stamp as the time stamp of thtransaction

Query Processing in Hekaton

- Queries and stored procedure are converted into native C code to maximize the efficiency
- Memory optimized tables and stored procedures or ad hoc queries on memory optimized tables query plan is transformed into C codes

Stored Proc in Hekaton

```
CREATE PROCEDURE SP_Example @id INT
WITH NATIVE_COMPILATION, SCHEMABINDING,
     EXECUTE AS OWNER
AS BEGIN ATOMIC
WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
      LANGUAGE = 'English')
   SELECT Name, Address, Phone
   FROM dbo.Customers WHERE Id = @id
```

Gotchas

- Limited set of operations
- compiled stored procedures must execute in a predefined security context so that we can run all permission checks once at creation time
- Must be schema bound

Query Interop

- Import and export data to and from hekaton based tables
- Ad hoc queries
- Transactions can leverage both memory optimized table and normal table

Transaction management

- Hekaton uses optimistic multiversion concurrency control
- Snapshot,serializable and repeatable read transcation isolation

3 key concept of transaction

- Logical read time
- End time
- Valid time

Transaction Processing

- Validation and Dependencies
  - Need to make sure the versions are not updated and no phantom read occurred
  - If T1 is dependent on T2, T1 is allowed to commit only if T2 commits. If T2 aborts, T1 must also abort
- Commit Logging and Post-processing
- Transaction Rollback

Durability of Transaction

Hekaton achieves transaction durability with

- Transaction logs
- Checkpoint logs

In order to support high through put following concepts were applied

- Index operations are not logged
- No undo information is logged
- A single large log record is used

Heckaton supports multiple concurrently generated log streams per database for I/O scaling

Checkpoints/Recovery

- Checkpoints logs are compressed T log
- Optimized for sequential access
- Checkpoint related I/O occurs incrementally and continuously
- Indexes are built during recovery

CPU efficiency for look ups

- Random lookups in a table with 10M rows
- All data in memory
- Intel Xeon W3520 2.67 GHz

Transaction size in #lookups	CPU cycles (in millions)		Speedup
	SQL Table	Hekaton Table	
1	0.734	0.040	10.8X
10	0.937	0.051	18.4X
100	2.72	0.150	18.1X
1,000	20.1	1.063	18.9X
10,000	201	9.85	20.4X

- Hekaton performance: 2.7M lookups/sec/core

CPU efficieny for updates

- Random updates, 10M rows, one index, snapshot isolation
- Log IO disabled (disk became bottleneck)
- Intel Xeon W3520 2.67 GHz

Transaction size in #updates	CPU cycles (in millions)		Speedup
	SQL Table	Hekaton Table	
1	0.910	0.045	20.2X
10	1.38	0.059	23.4X
100	8.17	0.260	31.4X
1,000	41.9	1.50	27.9X
10,000	439	14.4	30.5X

- Hekaton performance: 1.9M updates/sec/core

Throughput efficiency

Workload: read/insert into a table with a unique index

Insert txn (50%): append a batch of 100 rows