SQL Server 2014 In-Memory Tables (Extreme Transaction Processing)

Basics

Tony Rogerson, SQL Server MVP
@tonyrogerson
tonyrogerson@torver.net
http://www.sql-server.co.uk

Who am I?

- Freelance SQL Server professional and Data Specialist
- Fellow BCS, MSc in BI, PGCert in Data Science
- Started out in 1986 – VSAM, System W, Application System, DB2, Oracle, SQL Server since 4.21a
- Awarded SQL Server MVP yearly since 97
- Founded UK SQL Server User Group back in ’99, founder member of DDD, SQL Bits, SQL Relay, SQL Santa and Project Mildred
- Interested in commodity based distributed processing of Data.
- I have an allotment where I grow vegetables - I do not do any analysis on how they grow, where they grow etc. My allotment is where I escape technology.

Agenda

- Define concept “In-Memory”
- Implementation
  - Storage
  - Memory, Buffer Pool and using Resource Pools
  - Memory Optimised Tables and Index Types
  - Native Stored Procedures
- Queries
- HA / DR
- Planning

Define

“In-Memory”

What is IMDB / DBIM / IMT?

- Entire database resides in Main memory
- Hybrid - selective tables reside entirely in memory
- Row / Column store
  - Oracle keeps two copies of the data – in row AND column store – optimiser chooses, no app change required
  - SQL Server 2014 you choose –
    - Columnstore (Column Store Indexing) – traditionally for OLAP
    - Rowstore (memory optimised table with hash or range index) – traditionally for OLTP
- Non-Volatile Memory changes everything – it's here now!
  - SQL Server 2014 Buffer Pool Extensions

SQL Server “in-memory”

- Column Store (designed for OLAP)
  - Column compression to reduce memory required
  - SQL 2012:
    - One index per table, nonclustered only, table read-only
  - SQL 2014:
    - Only index on table – clustered, table is now updateable
- Memory Optimised Tables (designed for OLTP)
  - Standard CREATE TABLE with restrictions on FK’s and other constraints

Implementation

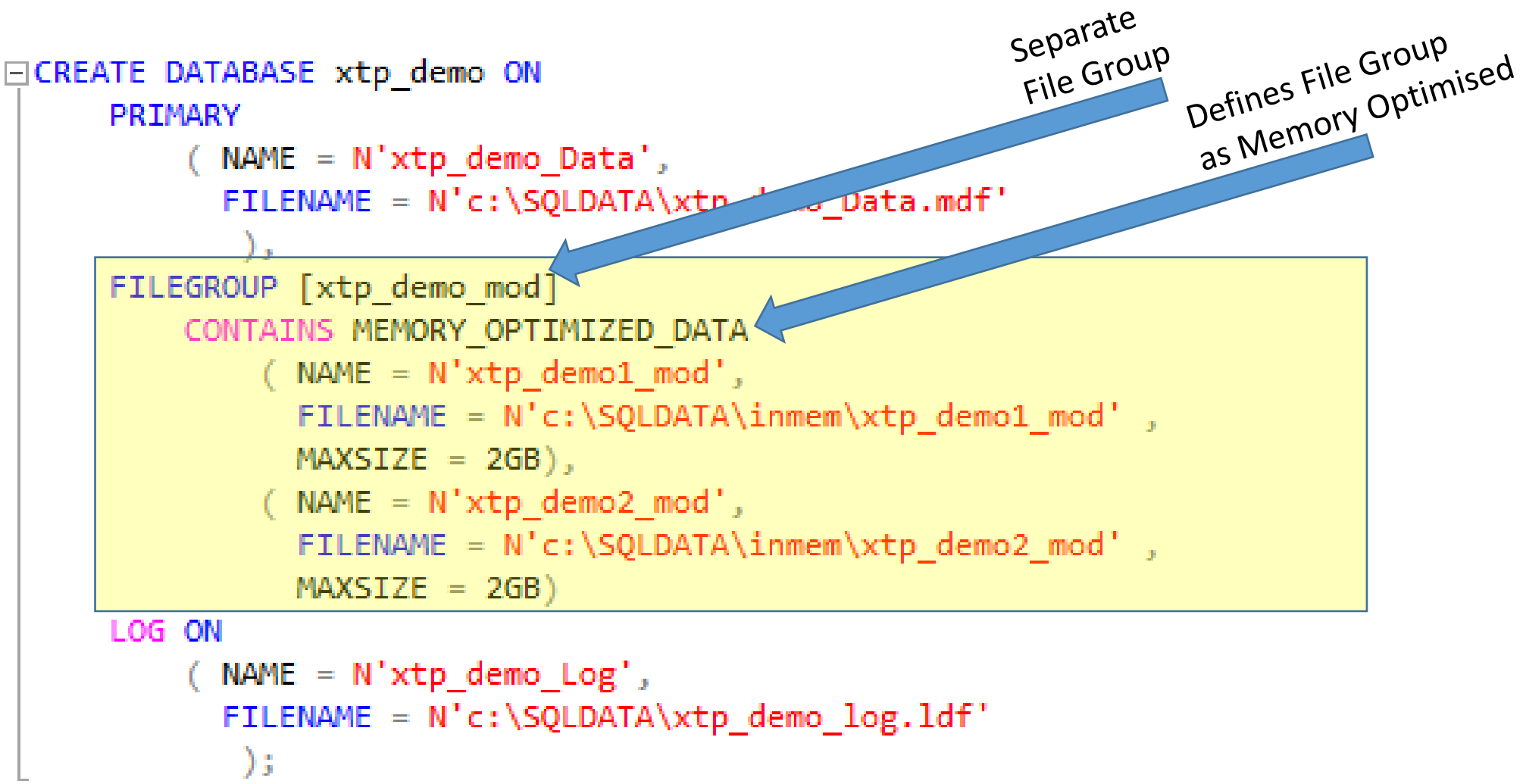
FILESTREAM underpins XTP

```
CREATE DATABASE xtp_demo ON
    PRIMARY
        ( NAME = N'xtp_demo_Data',
          FILENAME = N'c:\SQLDATA\xtp_demo_Data.mdf'
           ),
    FILEGROUP [xtp_demo_mod]
        CONTAINS MEMORY_OPTIMIZED_DATA
            ( NAME = N'xtp_demo1_mod',
              FILENAME = N'c:\SQLDATA\inmem\xtp_demo1_mod' ,
              MAXSIZE = 2GB),
            ( NAME = N'xtp_demo2_mod',
              FILENAME = N'c:\SQLDATA\inmem\xtp_demo2_mod' ,
              MAXSIZE = 2GB)
    LOG ON
        ( NAME = N'xtp_demo_Log',
          FILENAME = N'c:\SQLDATA\xtp_demo_log.ldf'
           );
```

Storage - FILESTREAM

```
CREATE DATABASE xtp_demo ON
    PRIMARY
        ( NAME = N'xtp_demo_Data',
          FILENAME = N'c:\SQLDATA\xtp_demo_Data.mdf'
           ),
    FILEGROUP [xtp_demo_mod]
        CONTAINS MEMORY_OPTIMIZED_DATA
            ( NAME = N'xtp_demo1_mod',
              FILENAME = N'c:\SQLDATA\inmem\xtp_demo1_mod' ,
              MAXSIZE = 2GB),
            ( NAME = N'xtp_demo2_mod',
              FILENAME = N'c:\SQLDATA\inmem\xtp_demo2_mod' ,
              MAXSIZE = 2GB)
    LOG ON
        ( NAME = N'xtp_demo_Log',
          FILENAME = N'c:\SQLDATA\xtp_demo_log.ldf'
           );
```

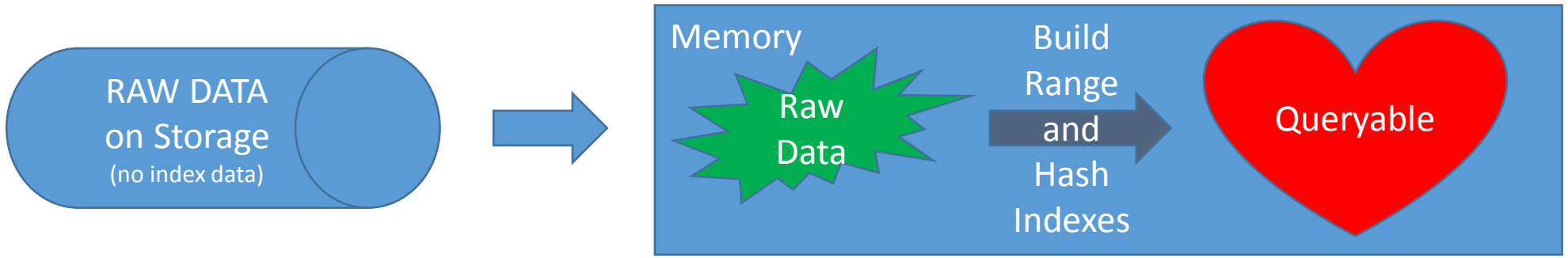
Storage - FILESTREAM



DEMO

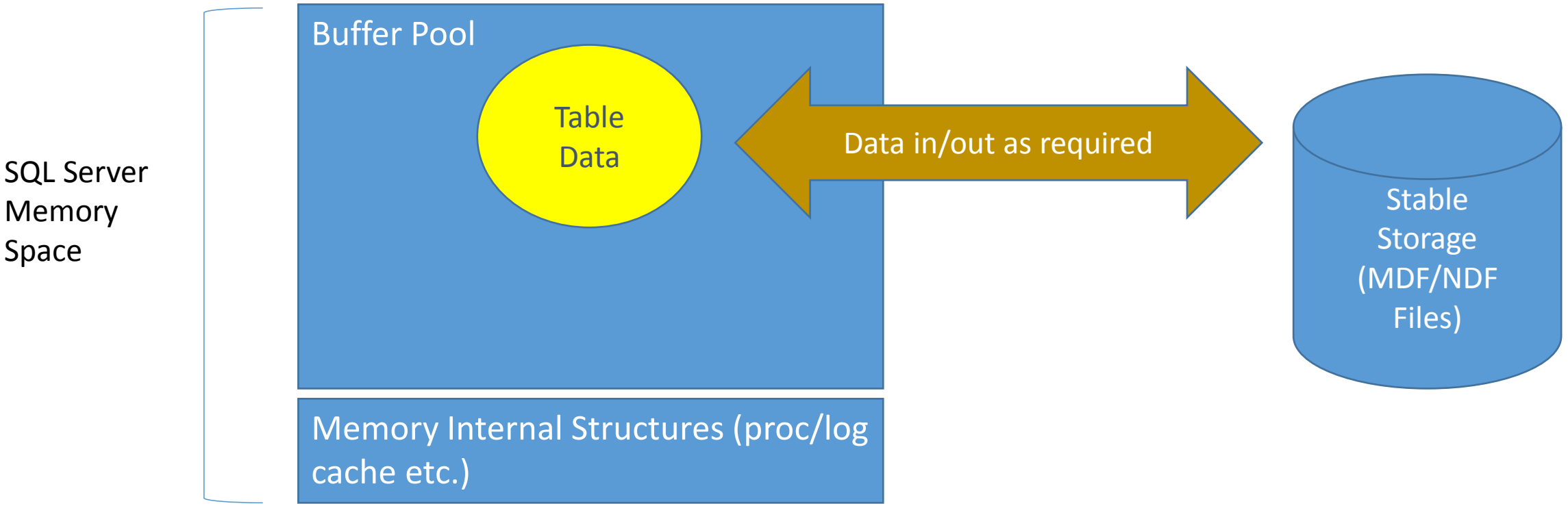
- CREATE DATABASE.sql

Database Recovery

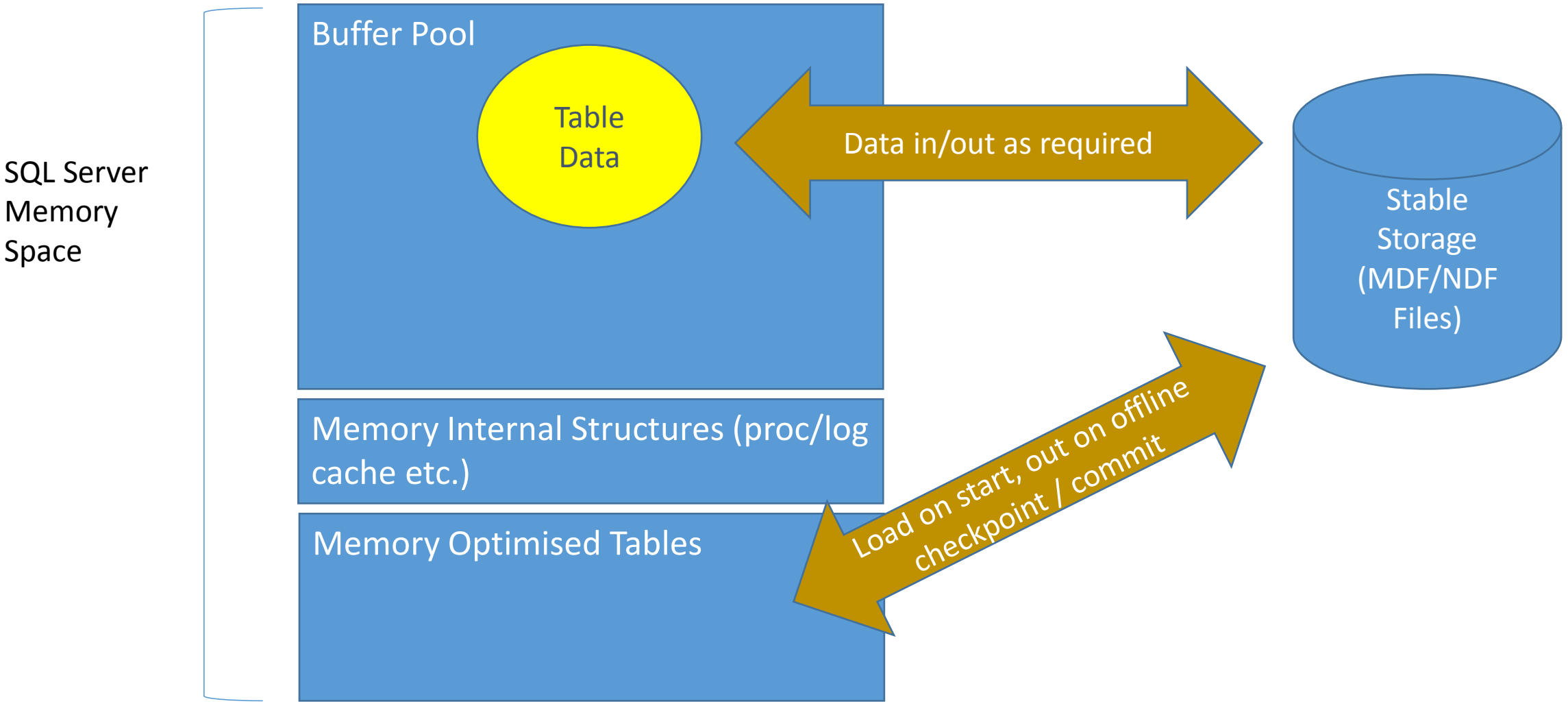


- Make sure the IO Sub-system can handle the load – it will swamp the IO Sub-System
- Only the Raw Data is stored – no need to hold indexes (everything fits in memory!)

SQL Server Memory Pools



SQL Server Memory Pools



Create Resource Pool

```
CREATE RESOURCE POOL mem_xtp_pool
       WITH ( MAX_MEMORY_PERCENT = 50,
              MIN_MEMORY_PERCENT = 50
              );

ALTER RESOURCE GOVERNOR RECONFIGURE;

EXEC sp_xtp_bind_db_resource_pool 'xtp_demo', 'mem_xtp_pool';

ALTER DATABASE xtp_demo SET OFFLINE WITH ROLLBACK IMMEDIATE;

ALTER DATABASE xtp_demo SET ONLINE;
```

DEMO

- RESOURCE POOL.sql

Memory Optimised Table

Table Structure

- New option on WITH (
  - MEMORY_OPTIMIZED = ON
  - DURABILITY = SCHEMA_AND_DATA | SCHEMA_ONLY

)
- Maximum 8 indexes per table
- Choice of two index types:
  - Hash (only good for equality expressions)
  - Range (good for everything)

Table Structure

- Compiled to .C structure
- Cannot ALTER TABLE – only DROP/CREATE
- Data is stored in FILESTREAM files (not MDF)
- Logging still done to LDF (SCHEMA_AND_DATA only)
- Statistics are not automatically updated – require:
  - UPDATE STATISTICS {table} WITH FULLSCAN, NORECOMPUTE, ALL

Table Declaration

- IDENTITY(1, 1) only – use SEQUENCE if not suitable
- DEFAULT constraints ok
- No Foreign Key constraints
- No Check constraints
- No UNIQUE constraints – only PRIMARY KEY allowed
- CREATE VIEW works (but view not usable from Native Stored Procedure)

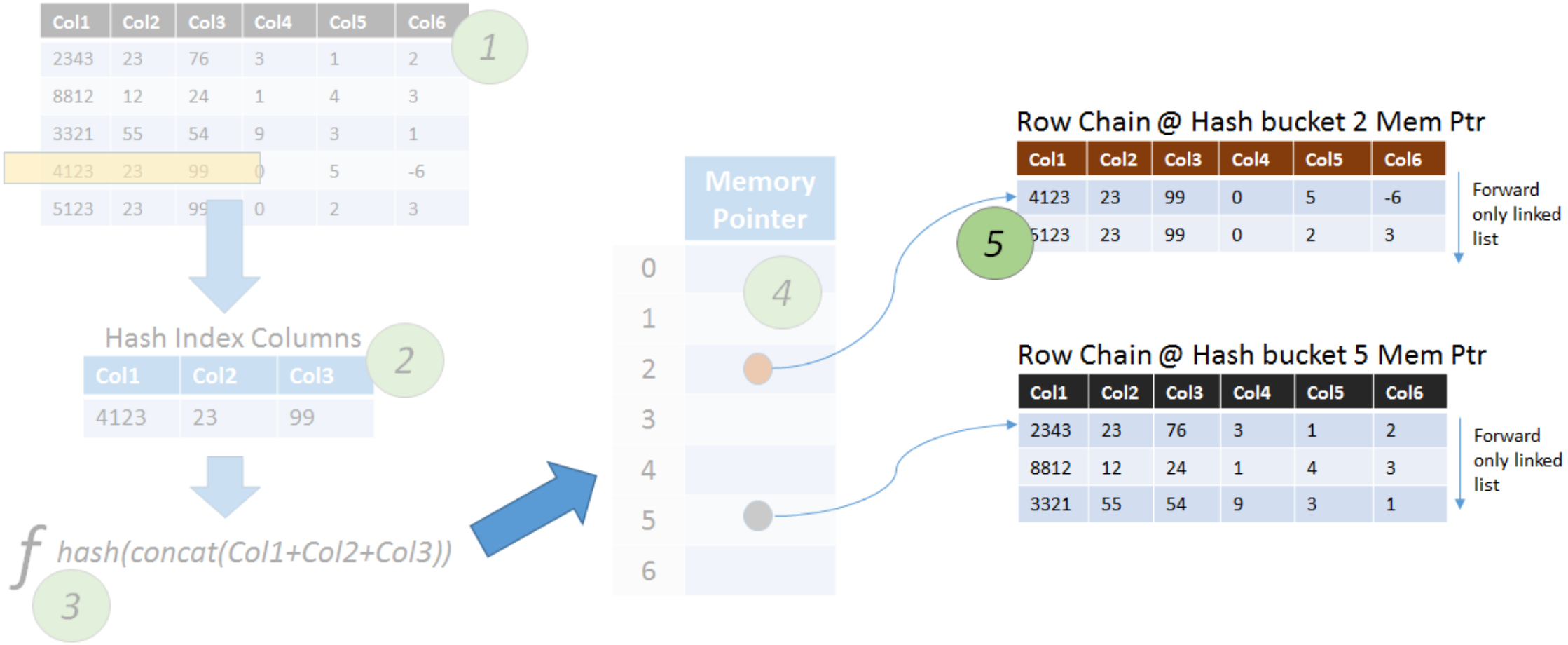
Useful DMV’s

- sys.hash_indexes
- sys.dm_db_xtp_hash_index_stats
- sys.dm_db_xtp_table_memory_stats
- sys.dm_db_xtp_index_stats
- sys.dm_db_xtp_object_stats

DEMO

- CREATE TABLE.sql
- DML.sql

Hash Index and Row Chains



BUCKET_COUNT options

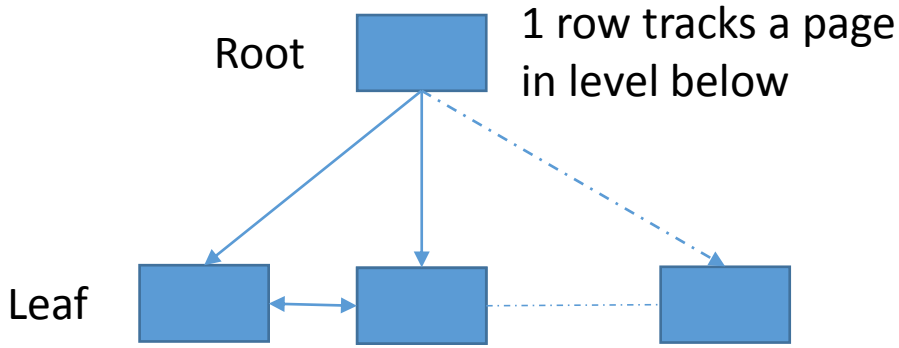
1024 – 8 kilobytes
2048 – 16 kilobytes
4096 – 32 kilobytes
8192 – 64 kilobytes
16384 – 128 kilobytes
32768 – 256 kilobytes
65536 – 512 kilobytes

131072 – 1 megabytes
262144 – 2 megabytes
524288 – 4 megabytes
1048576 – 8 megabytes
2097152 – 16 megabytes
4194304 – 32 megabytes
8388608 – 64 megabytes

16777216 – 128 megabytes
33554432 – 256 megabytes
67108864 – 512 megabytes
134217728 – 1,024 megabytes
268435456 – 2,048 megabytes
536870912 – 4,096 megabytes
1073741824 – 8,192 megabytes

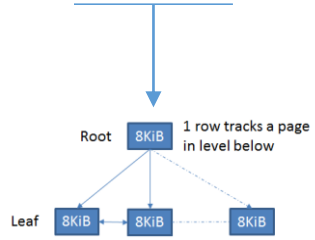
Range Index (B^w Tree ≈ B^+Tree)

B^+Tree

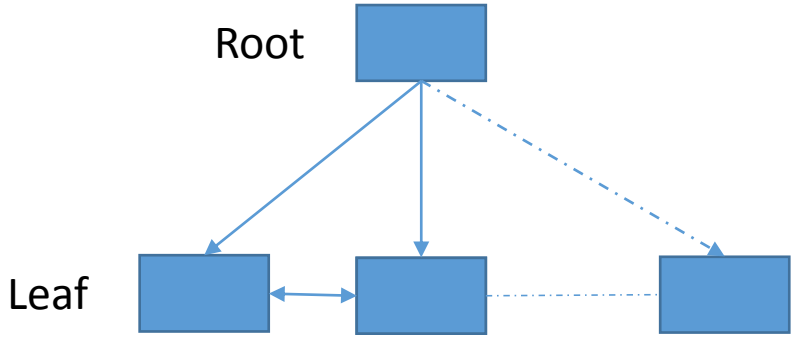


Leaf row per *table row*

{index-col}{pointer}



B^w Tree



Leaf row per *unique value*

{index-col}{memory-pointer}

Row Chain

account_no	active
7	1
6	1
5	1
4	1
3	1
2	1
1	1

account_no	active
14	0
13	0
12	0
10	0

DEMO

- INDEXING.sql

Index recommendations

- Use HASH when
  - Mostly unique values
  - Query uses expressions “Equality” i.e. = or !=
  - Set BUCKET_COUNT appropriately
  - Seriously think if you are considering composite index
- Use RANGE when
  - Expressions are not Equality e.g. < > between
  - Composite Index
  - Low Cardinality

Multi-Version Concurrency Control (MVCC)

- Compare And Swap replaces Latching
- Update is actually INSERT and Tombstone
- Row Versions [kept in memory]
- Garbage collection task cleans up versions no longer required (stale data rows)
- Versions no longer required determined by active transactions – may be inter-connection dependencies
- Your in-memory table can double, triple, x 100 etc. in size depending on access patterns (data mods + query durations)
- Row chains can expand dramatically and cause really poor performance

Versions in the Row Chain

	From Hash Bucket	TS_START	TS_END	FirstName	Surname	Pints Drank
Newest	23212	43	NULL	Mark	Whitehorn	7
	23212	26	42	Mark	Whitehorn	6
	23212	13	25	Mark	Whitehorn	5
	23212	9	12	Mark	Whitehorn	4
	23212	6	8	Mark	Whitehorn	3
	23212	3	5	Mark	Whitehorn	2
Oldest	23212	1	2	Mark	Whitehorn	1

TS_START	TS_END	Connection
2	NULL	55
7	NULL	60
50	NULL	80

Versions in the Row Chain

	From Hash Bucket	TS_START	TS_END	FirstName	Surname	Pints Drank
Newest	23212	43	NULL	Mark	Whitehorn	7
	23212	26	42	Mark	Whitehorn	6
	23212	13	25	Mark	Whitehorn	5
	23212	9	12	Mark	Whitehorn	4
Oldest	23212	6	8	Mark	Whitehorn	3
	23212	3	5	Mark	Whitehorn	2
	23212	1	2	Mark	Whitehorn	1

Tombstone

TS_START	TS_END	Connection
2	10	55
7	NULL	60
50	NULL	80

Versions in the Row Chain

Current

From Hash Bucket	TS_START	TS_END	FirstName	Surname	Pints Drank
23212	43	NULL	Mark	Whitehorn	7
23212	26	42	Mark	Whitehorn	6
23212	13	25	Mark	Whitehorn	5
23212	9	12	Mark	Whitehorn	4
23212	6	8	Mark	Whitehorn	3
23212	3	5	Mark	Whitehorn	2
23212	1	2	Mark	Whitehorn	1

Tombstone

TS_START	TS_END	Connection
2	NULL	55
7	NULL	60
50	NULL	80

Natively Compiled Procedures

- Compiled via C code
- Table statistics [which are not auto-updated] used when proc is created
- *{very}*Restricted functionality (they are working on it)
- Massive speed improvements possible

DEMO

- NATIVE PROCS.sql

Querying

Understanding Isolation and mixing memory optimised and storage optimised tables

SNAPSHOT Isolation

- Your connection has a Snapshot of the Data that you just read – Snapshot is persistent and isolated from other connections using Versioning (MVCC).
- Default SQL Server isolation is READ COMMITTED
- Inside a transaction you must specify WITH (SNAPSHOT) to override the default isolation level
- Cannot use SET TRANSACTION ISOLATION SNAPSHOT with transactions and in-memory tables
- MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT database setting

DEMO

- QUERYING.sql

Querying (interop)

- **Not** supported:
  - Linked Tables
  - Cross database joins (except table vars and temp tables)
  - Reference from indexed view
  - MERGE (memory optimised table as target)
  - TRUNCATE TABLE
  - CLR using context connection
  - A load of table hints, see: http://msdn.microsoft.com/en-us/library/dn133177.aspx
- Must use WITH (SNAPSHOT) when statement is not auto-commit.

HADR - DEMO

- HADR.sql

Backup / Restore

- Meta-data only for Non-Durable tables (no data!)
- Backup is self-contained
  - Full backup
  - Differentials
  - Logs
  - Restore in Standby
- Remember
  - Data is loaded into memory on recovery [standby performs a semi-recovery]
  - Make sure you have enough memory on the secondary

Availability Groups

- Yep – they work too
- Differs from Log restore
  - Data and Indexes already in memory
- Readable secondary replica’s work
- SCHEMA_ONLY (Non-Durable) tables will be empty!

Planning

Thinking through the possibilities

Migration - DEMO

- Table: Memory Optimiser Advisor
- Stored Procedure: Native Compilation Advisor

Migrating Tables

- Throw it all in memory?
  - Probably not a good idea!
- Partition through View – mix memory and storage optimised
- Only index what you need + monitor and remove ones not required
- Application Transaction duration/number of updates suitable?