PASS SUMMIT 2016

Credits to Kalen Delaney, Jos de Bruijn, Sunil Agarwal, Cristian Diaconu, Craig Freedman, Alejandro Hernandez Saenz, Jack Li, Mike Zwilling and the entire Hekaton team

Latch free since 2007

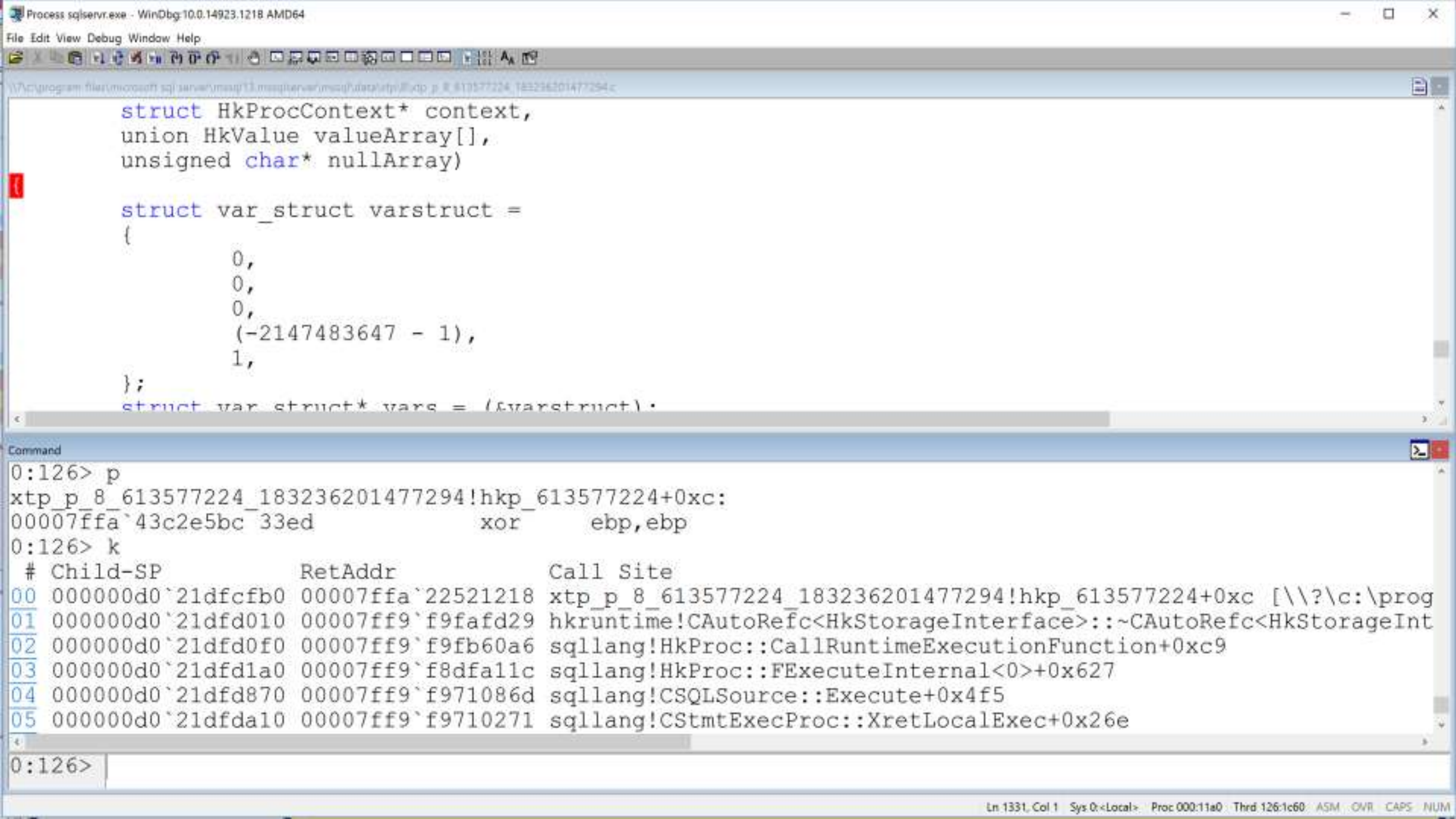
Inside SQL Server In-Memory OLTP

Based on this whitepaper, internal docs, source code, and reverse engineering

Bob Ward, Principal Architect, Microsoft, Data Group, Tiger Team

deck and demos at http://aka.ms/bobwardms

PASS



Process sqlservr.exe - WinDbg:10.0.14923.1218 AMD64
File Edit View Debug Window Help
struct HkProcContext* context,
union HkValue valueArray[],
unsigned char* nullArray)
{
struct var_struct varstruct =
{
0,
0,
0,
(-2147483647 - 1),
1,
};
Command
0:126> p
xtp_p_8_613577224_183236201477294!hkp_613577224+0xc:
00007ffa`43c2e5bc 33ed xor ebp,ebp
0:126> k
Child-SP RetAddr Call Site
00 000000d0`21dfcfb0 00007ffa`22521218 xtp_p_8_613577224_183236201477294!hkp_613577224+0xc [\\?\c:\prog
01 000000d0`21dfd010 00007ff9`f9fafd29 hkruntime!CAutoRefc<HkStorageInterface>::~CAutoRefc<HkStorageInt
02 000000d0`21dfd0f0 00007ff9`f9fb60a6 sqllang!HkProc::CallRuntimeExecutionFunction+0xc9
03 000000d0`21dfd1a0 00007ff9`f8dfa11c sqllang!HkProc::FExecuteInternal<0>+0x627
04 000000d0`21dfd870 00007ff9`f971086d sqllang!CSQLSource::Execute+0x4f5
05 000000d0`21dfda10 00007ff9`f9710271 sqllang!CStmtExecProc::XretLocalExec+0x26e
0:126>
Ln 1331, Col 1 Sys 0:<Local> Proc 000:11a0 Thrd 126:1c60 ASM OVR CAPS NUM

Great [primer](#)
and
customer
stories by
[@jdebruijn](#)

This is a 500 Level Presentation
It is ONE talk with a 15min break for 3 hours

=

Get and run the demo
yourself from [github](#)

Let's Do This

Based on SQL Server 2016

- What, How, Why, and Architecture
- Data and Indexes
- Concurrency, Transactions, and Logging
- Checkpoint and Recovery
- Natively Compiled Procedures
- Resource Management if time

Check out the Bonus Material Section

PASS

The Research behind the concept

OLTP Through the Looking Glass, and What We Found There

by Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, Michael Stonebraker , SIGMOD 2008

TPC-C hand-coded on top of the SHORE storage engine

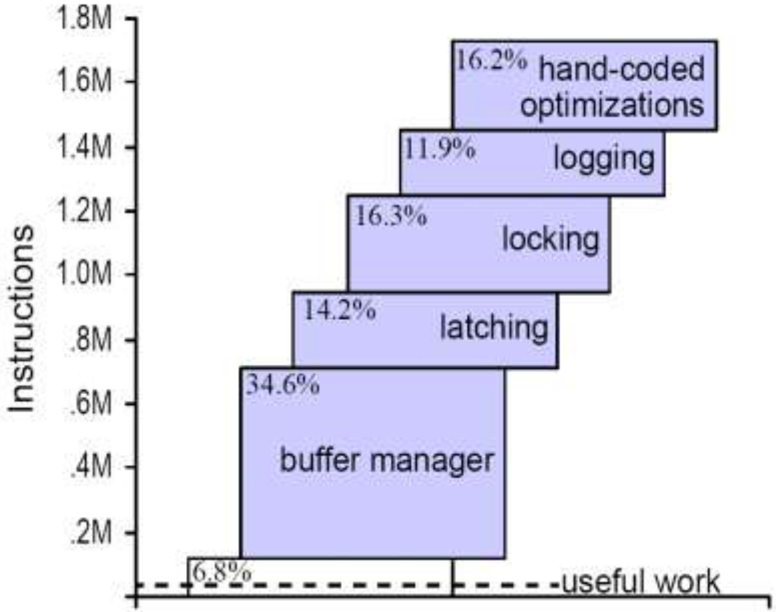


Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the transaction on a no-overhead kernel.

PASS

The Path to In-Memory OLTP

Project Verde

- 2007

"Early" Hekaton

- 2009/2010

Hekaton becomes In-Memory OLTP

- SQL Server 2014

In-Memory OLTP Unleashed

- SQL Server 2016

What, Why, and How

Keep SQL Server relevant in a world of high-speed hardware with dense cores, fast I/O, and inexpensive massive memory

The need for high-speed OLTP transactions at ***microsecond speed***

Reduce the ***number of instructions*** to execute a transaction

- Find areas of latency for a transaction and reduce its footprint
- Use multi-version optimistic concurrency and “lock free” algorithms
- Use DLLs and native compiled procs

Minimal diagnostics in critical path

XTP = eXtreme Transaction Processing
HK = Hekaton

In-Memory OLTP Capabilities

Hear the case studies

- Supported on Always On Availability Groups
- Supports ColumnStore Indexes for HTAP applications
- Supported in Azure SQL Database Preview
- Cross container transactions (disk and in-memory in one transaction)
- Table variables supported
- SQL surface area expanded in SQL Server 2016. Complete support here
- LOB data types (ex. Varchar(max)) supported
- BACKUP/RESTORE complete functionality
- Use Transaction Performance Analysis Report for migration

The Path of In-Memory OLTP Transactions

"Normal" INSERT

- INSERT LOG record
- Obtain locks
- Latch page
- Modify page
- Update index pages (more locks and latch)
- Release latches and locks

Page Split

Sys Tran = Log Flush

Spinlocks

COMMIT Transaction = Log Record and Flush

In-Memory OLTP INSERT

Insert ROW into memory

No latch
No spinlock

No index logging

Maintain index in memory

SCHEMA_ONLY
no logging

COMMIT Transaction = Insert HK Log Record and Flush

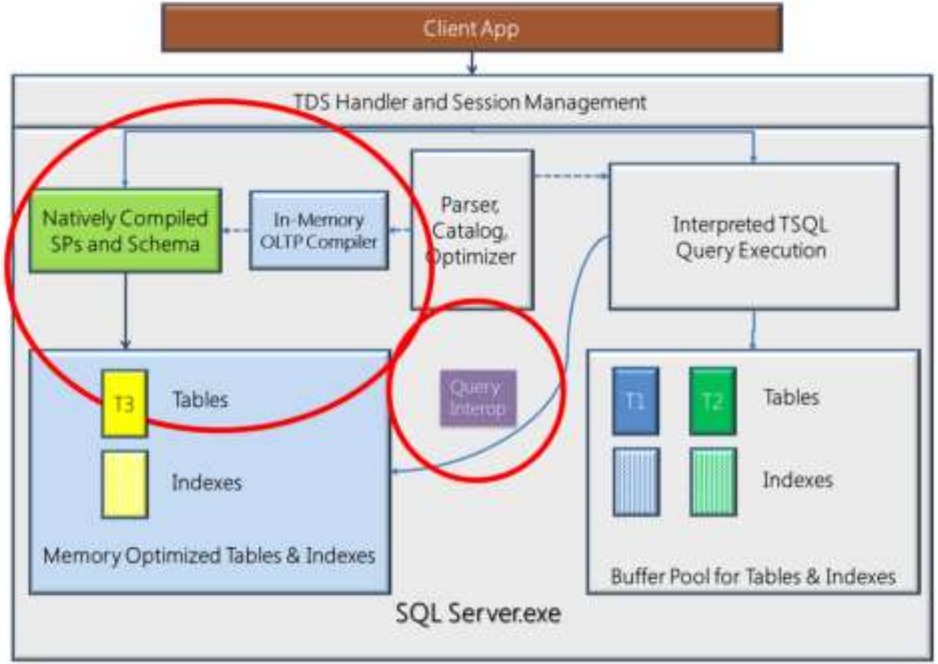
Just show it!

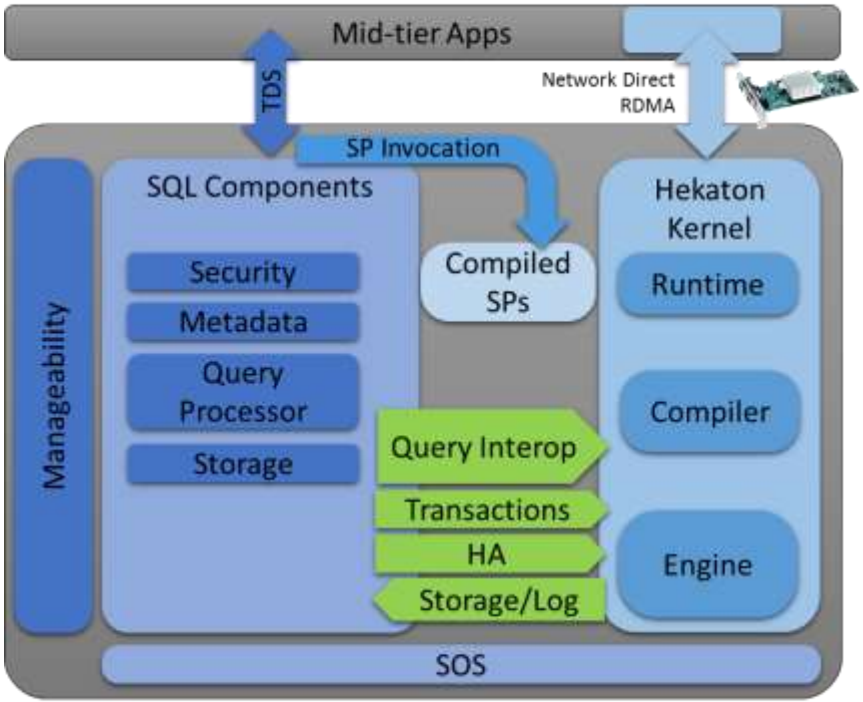
Demo

ἑκατόν

Architecture and Engine Integration

The In-Memory OLTP Architecture





In-Memory OLTP Engine Components

examples below

hkruntime.dll | hkcompile.dll | hkengine.dll

Hekaton Kernel

Runtime
- Call Native Procs
- Query Interop

Compile
- Build Schema DLL
- Build Native Proc DLL

The Engine
- Allocate Memory
- Checkpoint
- Transaction Logging
- System Tables

SQL Engine
- Creating and executing procs
- Query Interop
- SQL System Tables
- Log Manager
- HADR

SCHEMA DLL and Native Proc DLL

"HOST"

SQLOS (sqldk.dll)

PASS

Creating the database for in-memory OLTP

filegroup <fg_name> contains memory_optimized_data
(name = <name>, filename = '<drive and path>\<folder name>')

we create this

```
PS C:\temp\gocowboys_inmem> dir

    Directory: C:\temp\gocowboys_inmem

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        10/7/2016   7:28 AM                $FSLOG
d-----        10/7/2016   7:28 AM                $HKv2
-a----        10/7/2016   7:28 AM            447 filestream.hdr
```

filestream container

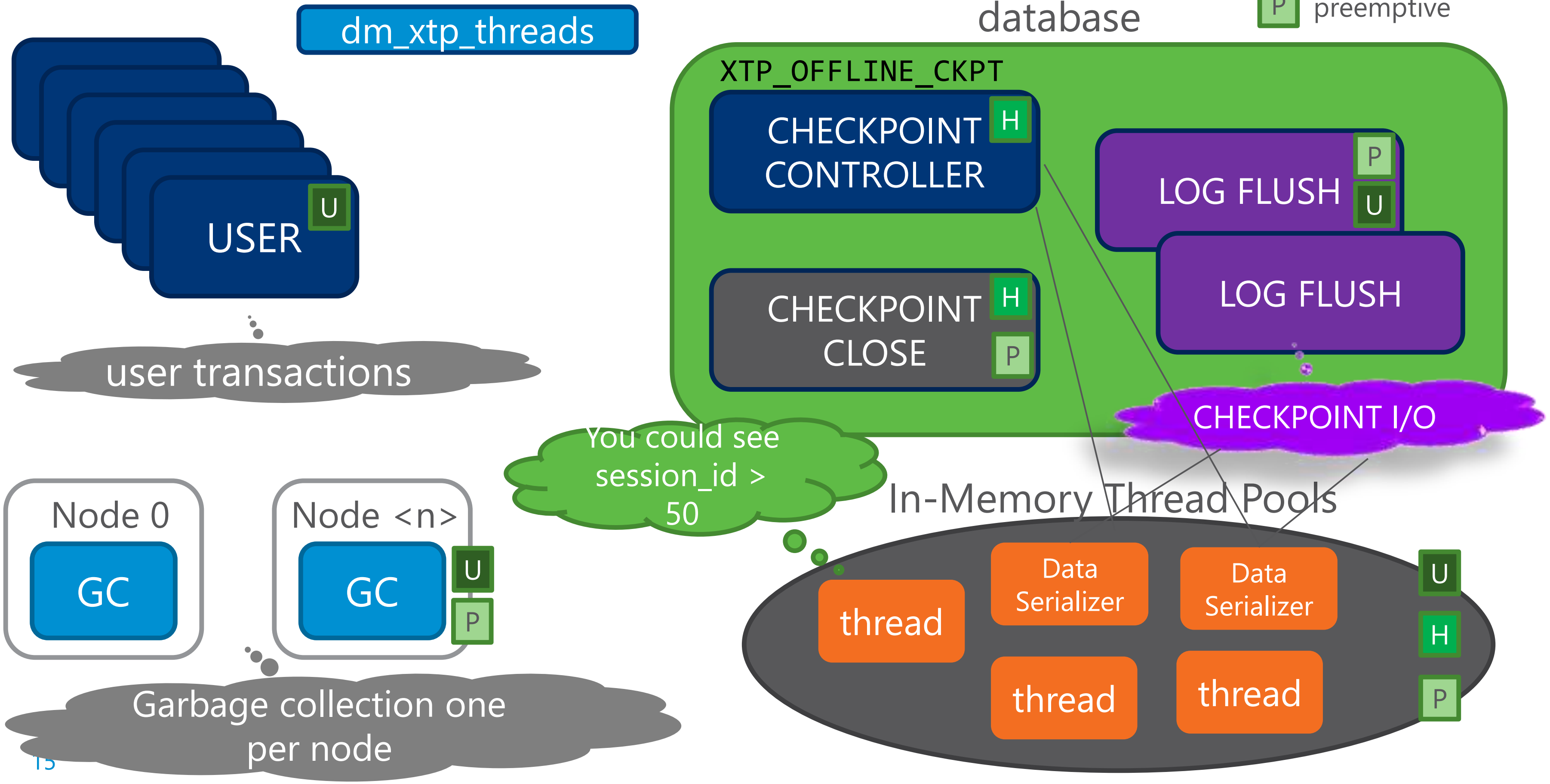
Create multiple across drives

MEMORYCLERK_XTP	• Name = DB_ID_<dbid> • Your data goes here so this gets big
MEMOBJ_XTPDB	• Two of these for each db • Much smaller but can grow some over time

HK System Tables created (hash index based tables)

In-memory OLTP Threads

U user scheduler
H hidden scheduler
P preemptive



Inside the HK architecture

Demo

Data and Indexes

Hash Index = single row lookup with '='
Range Index = searching for multi-rows

Creating a Table

All tables require at least one index

```
create table starsoftheteam
(player_number int identity
primary key nonclustered hash with (bucket_count = 1048576),
player_name char(1000) not null)
with (memory_optimized = on, durability = schema_and_data)
```

Default = range

Schema stored in SQL system catalog → Hash index built in-memory → Hk system tables populated → Schema DLL built → Perform a checkpoint

Ex. “root” table

FS container files created and written

Inside Data and Rows

Compute your estimated row size here

How do you find data rows in a normal SQL table?

- Heap = Use IAM pages
- Clustered Index = Find root page and traverse index

What about an in-memory table?

Hash index scan is possible

- **Hash index table pointer** known in HK metadata for a table. *Hash* the index key, go to bucket pointer, traverse chain to find row
- Page Mapping Table used for **range indexes** and has a known pointer in HK metadata. Traverse the range index which points to data rows
- Data exists in memory as pointers to rows (aka a **heap**). No page structure

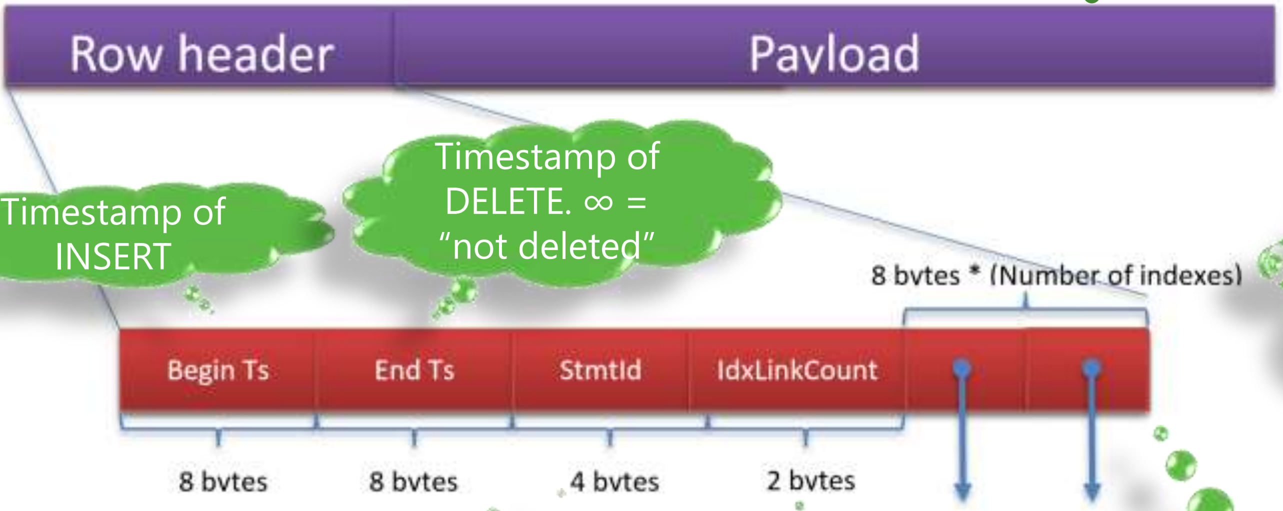
"bag of bytes"

All data rows have *known* header but data is ***opaque*** to HK engine

- Schema DLL and/or Native Compiled Proc DLL knows the format of the row data
- Schema DLL and/or Native Compiled Proc DLL knows how to find "key" inside the index

A In-Memory OLTP Data Row

Your data. "bytes" to HK engine



Timestamp of INSERT

Timestamp of DELETE. ∞ = "not deleted"

Rows don't have to be contiguous in memory

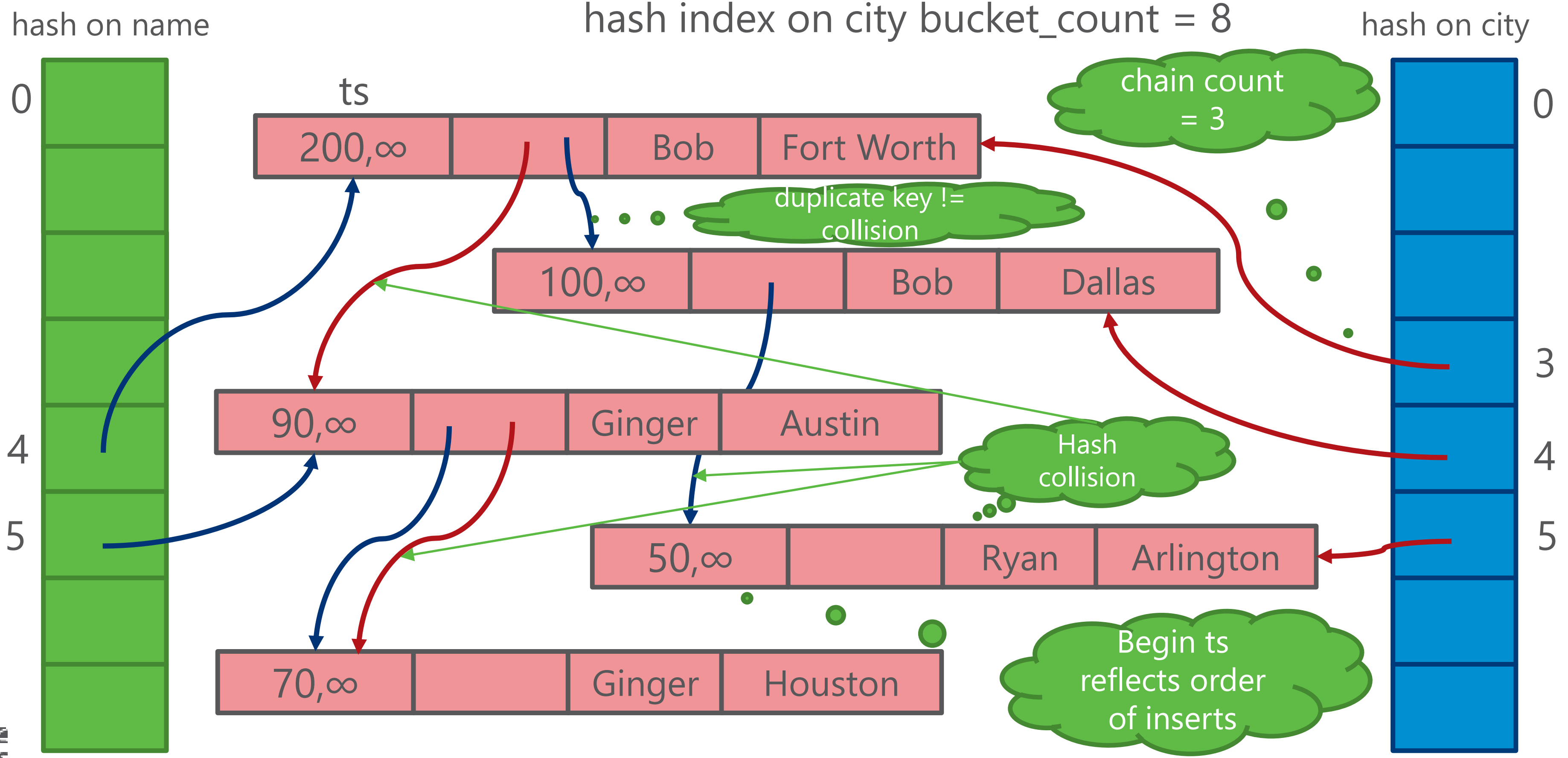
[Halloween](#) protection

Number of pointers = # indexes

Points to another row in the "chain". One pointer for each index on the table

Hash indexes

columns = name char(100), city char (100)
hash index on name bucket_count = 8
hash index on city bucket_count = 8



Hash Indexes

When should I use these?

Single row lookups by exact key value

Support multi-column indexes

Do I care about bucket counts and chain length?

The key is to keep the chains short

Key values with high cardinality the best and typically result in short chains

Larger number of buckets keep collisions low and chains shorter

Monitor and adjust with dm_db_xtp_hash_index_stats

SQL catalog view sys.hash_indexes also provided

Empty bucket % high ok except uses memory and can affect scans

Monitor average and max chain length (1 avg chain ideal; < 10 acceptable)

Rebuild with ALTER TABLE. Could take some time

REBUILD can block interop with SCH-S; Native proc rebuilt so blocked

PASS

Finding In-Memory Data Rows

Demo

Range Indexes

When should I use these?

Frequent multi-row searches based on a “range” criteria (Ex. >, <)

ORDER BY on index key

First column of multi-column index searches

“I don’t know” or “I don’t want to maintain buckets” – Default for an index

You can always put pkey on hash and range on other columns for seeks

What is different about them?

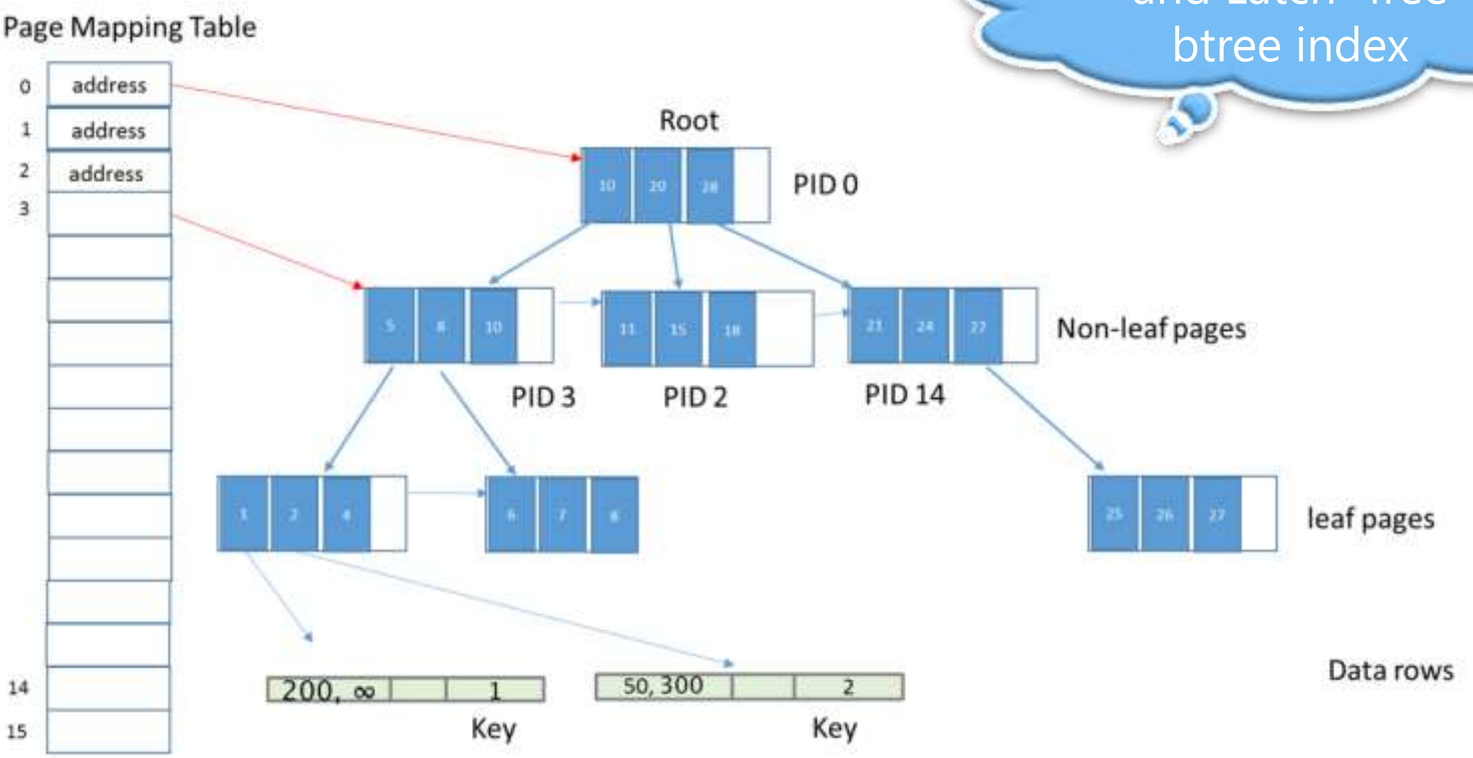
Contains a tree of pages but not a fixed size like SQL btrees

Pointers are logical IDs that refer to Page Mapping Table

Updates are contained in “delta” records instead of modifying page

Range Indexes

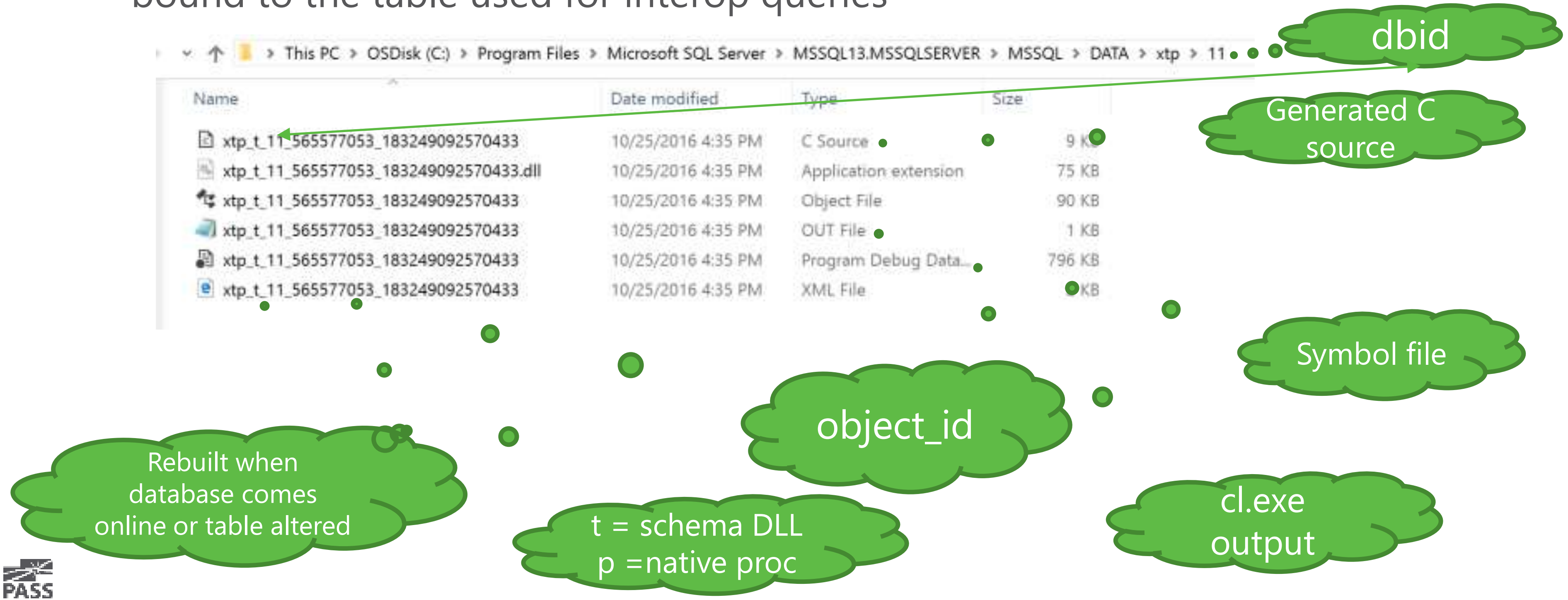
A Bw-Tree. "Lock and Latch" free btree index



PASS

The SCHEMA DLL

- The HK Engine doesn't understand row formats or key value comparisons
- So when a table is built or altered, we create, compile, and build a DLL bound to the table used for interop queries

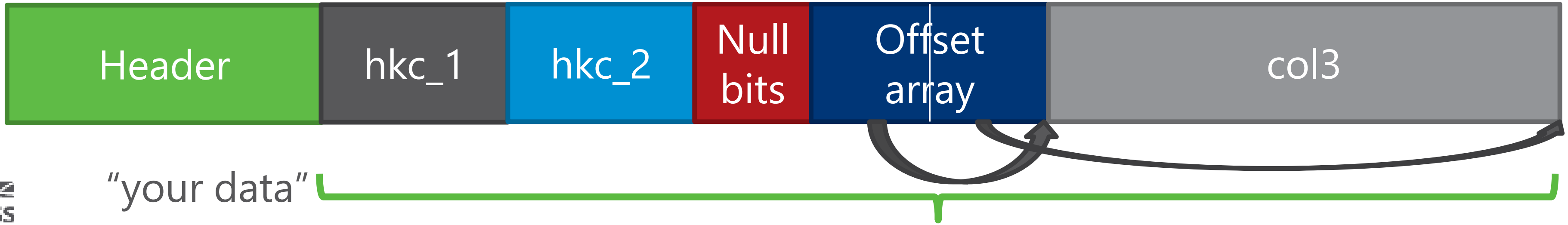


The Row Payload

```
create table letsgomavs
(col1 int not null identity primary key
nonclustered hash with (bucket_count =
1000000),
col2 int not null,
col3 varchar(100)
)
with (memory_optimized = on, durability
= schema_and_data)
```

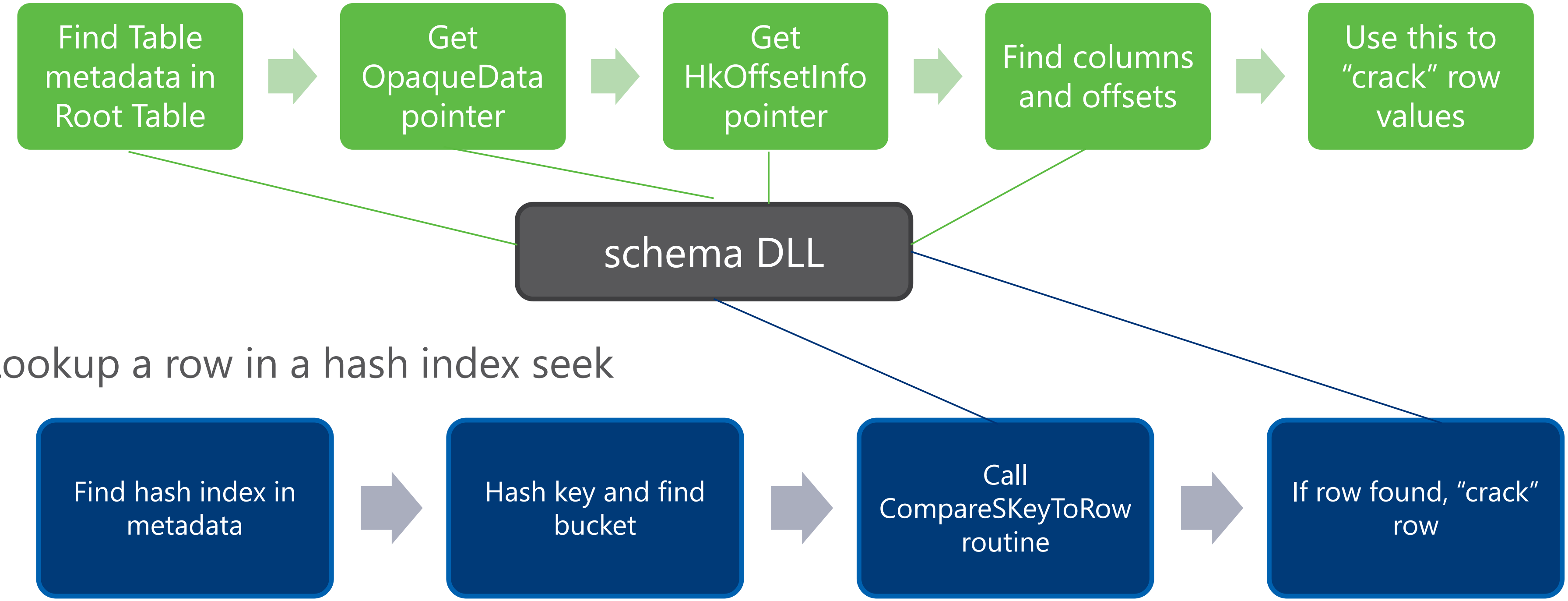
nullable

```
struct NullBitsStruct_2147483654
{
unsigned char hkc_isnull_3:1;
};
struct hkt_2147483654
{
long hkc_1;
long hkc_2;
struct NullBitsStruct_2147483654 null_bits;
unsigned short hkvdo[2];
};
```



Access data with the SCHEMA DLL

Interop Code to “crack” rows



Cracking In-Memory OLTP Data Rows

Demo

Concurrency, Transactions, and Logging

Multi-Version Optimistic Concurrency (MVCC)

Immutable

- Rows never change: UPDATE = DELETE + INSERT

Versions

- UPDATE and DELETE create versions
- Timestamps in rows for visibility and transactions correctness

NOT in tempdb

Optimistic

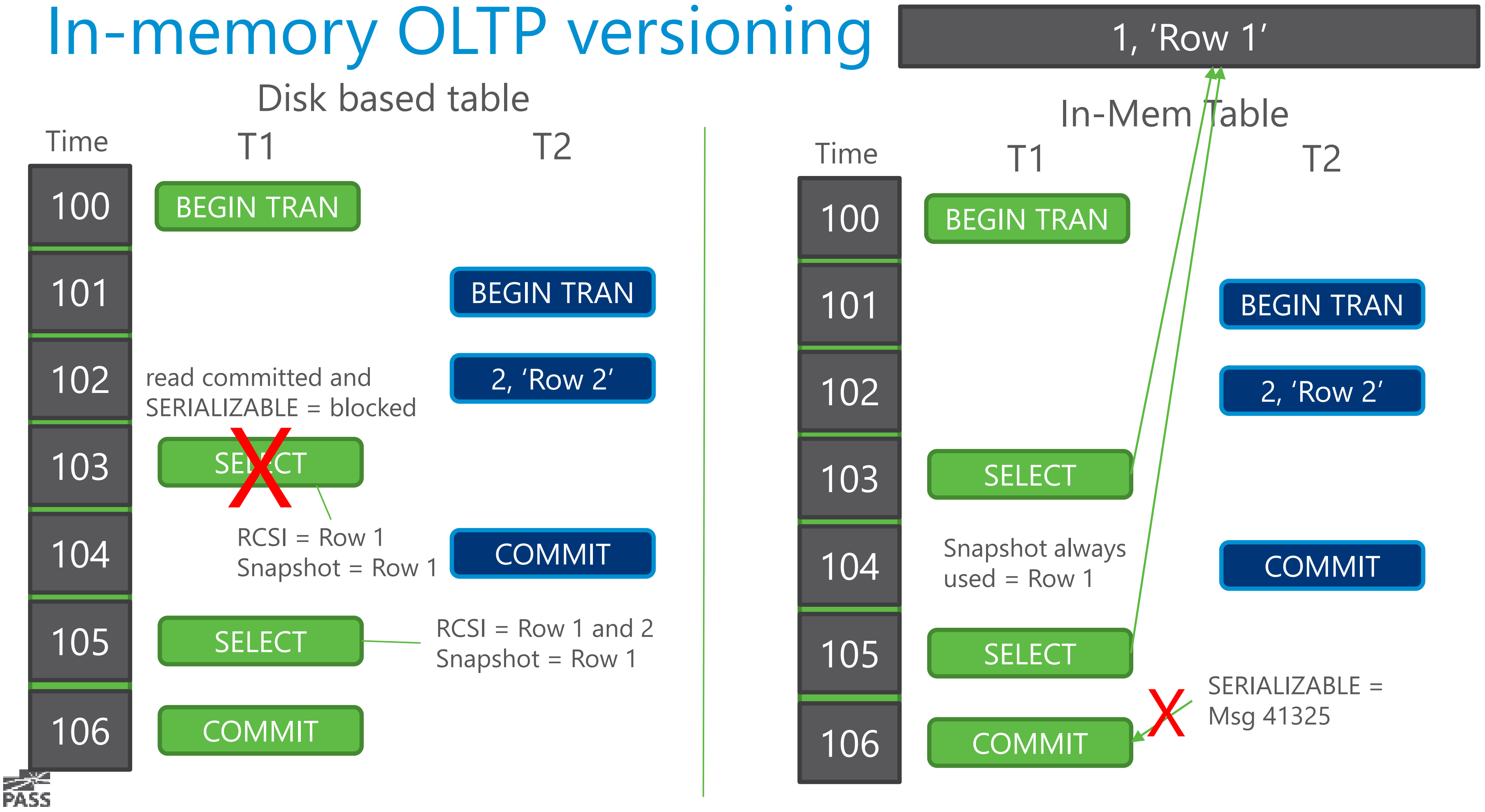
- Assume no conflicts
- Snapshots + conflict detection
- Guarantee correct transaction isolation at commit

Pessimistic = locks

Errors may occur

PASS

In-memory OLTP versioning

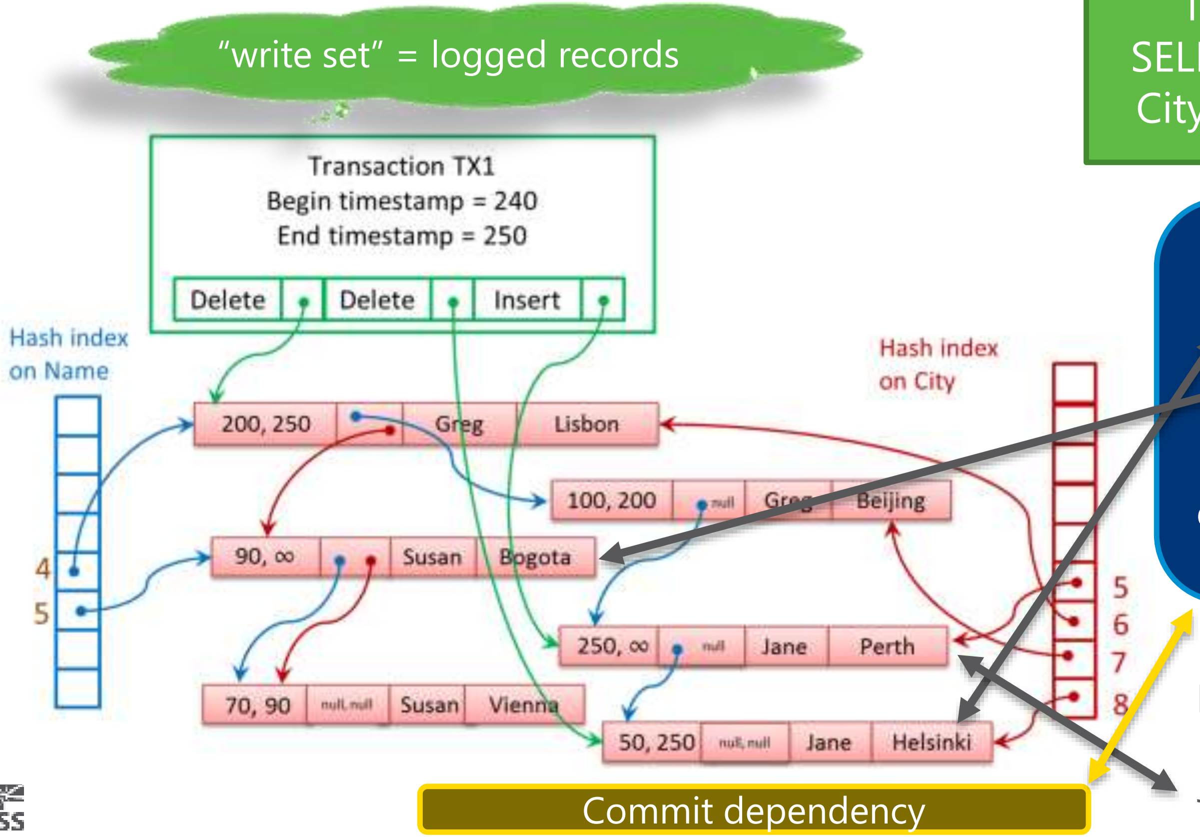


What were those timestamps for?

"read set"

TS = 243
SELECT Name, City FROM T1

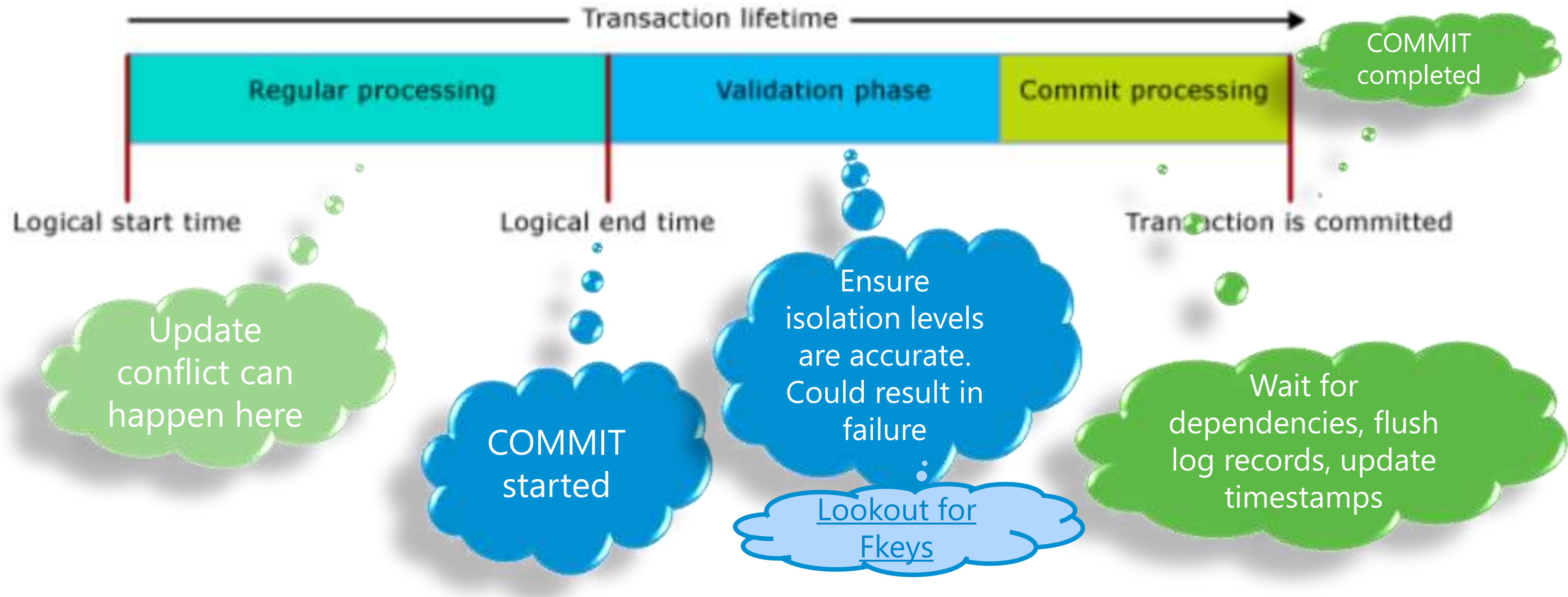
Greg, Lisbon
Susan Bogata
Jane Helsinki



BEGIN TRAN TX3 – TS 246
SELECT City FROM T1 WITH (REPEATABLEREAD) WHERE Name = 'Jane';
UPDATE T1 WITH (REPEATABLEREAD) SET City 'Helinski' WHERE Name = 'Susan';
COMMIT TRAN -- commits at timestamp 255

FAIL = 'Jane' changed after I started but before I committed and I'm REPEATABLEREAD. With SQL update to Jane would have been blocked

Transactions Phases and In-Memory OLTP



dm_db_xtp_transactions

It is really "lock free"

The "host" may wait – tLog, SQLOS

Code ***executing transactions in the Hekaton Kernel*** are lock, latch, and spinlock free

deadlock free

Locks
- Only SCH-S needed for interop queries
- Database locks

Latches
- No pages = No page latches
- Latch XEvents don't fire inside HK

Spinlocks
- Spinlock class never used

CMED_HASH_SET fix

We use Thread Local Storage (TLS) to "guard" pointers

We can "walk" lists lock free. Retries may be needed

XTP wait types not in transaction code

Atomic "Compare and Swap" (CAS) to modify

Great blog post here

Spinlocks use CAS ('cmpxchg') to "acquire" and "release"

IsXTPSupported() = cmpxchg16b

PASS

In-Memory OLTP Concurrency and Transactions

Demo

Logging for a simple disk based INSERT

use dontgobluejays

create table starsoftheteam (player_no int primary key nonclustered, player_name char(100) not null)

One row already exists

begin tran

insert into starsoftheteam (1, 'I really don''t like the Blue Jays')

commit tran

LOP_BEGIN_XACT	124
LOP_INSERT_ROWS – heap page	204
LOP_INSERT_ROWS – ncl index	116
LOP_COMMIT_XACT	84

4 log records

528 bytes

Logging for a simple in-mem OLTP INSERT

```
use gorangers
create table starsoftheteam (player_no int primary key nonclustered,
player_name char(100) not null)
with (memory_optimized = on, durability = schema_and_data)_
begin tran
insert into starsoftheteam values (1, 'Let''s go Rangers')
commit tran
```

fn_dblog_xtp()

3 log records

436 bytes

LOP_BEGIN_XACT	144
LOP_HK	208
LOP_COMMIT_XACT	84

LOP_HK:
- HK_LOP_BEGIN_TX
- HK_LOP_INSERT_ROW
- HK_LOP_COMMIT_TX

Multi-row INSERT disk-based heap table

100 rows

- LOP_BEGIN_XACT
- LOP_INSERT_ROWS – heap page
- LOP_INSERT_ROWS – ncl index
- LOP_BEGIN_XACT
- LOP_MODIFY_ROW – PFS
- LOP_HOBT_DELTA
- LOP_FORMAT_PAGE
- LOP_COMMIT_XACT
- LOP_SET_FREE_SPACE - PFS
- LOP_COMMIT_XACT

1 pair for every row

Alloc new page twice

PFS latch

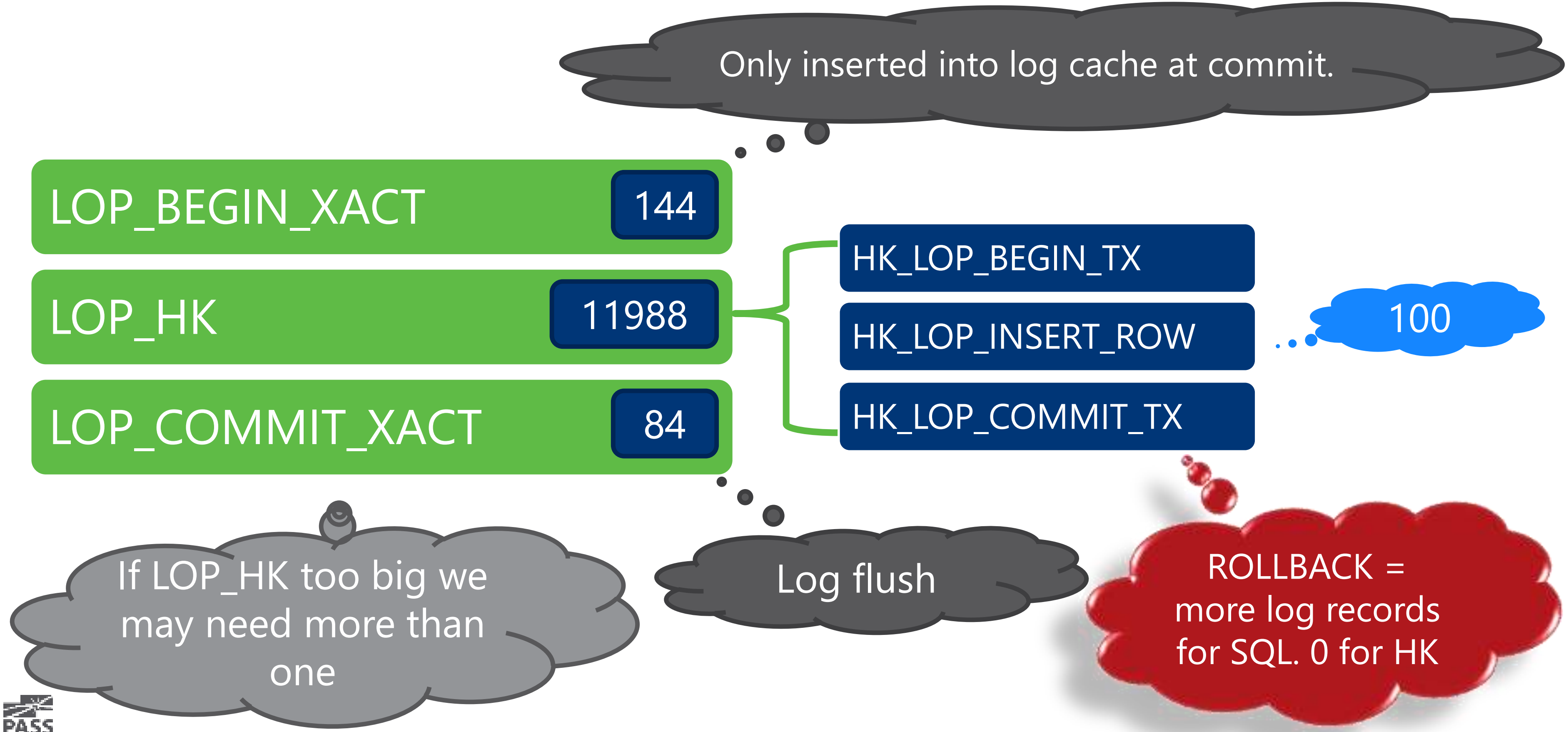
Metadata access

Log flush

PFS latch multiple times

213 log records @ 33Kb

Compared to In-Memory OLTP



Checkpoint and Recovery

In-Memory OLTP CHECKPOINT facts

Log truncation eligible at CHECKPOINT event

Why CHECKPOINT?

Speed up database startup. Otherwise we would have to keep around a huge tlog around.

We only write data based on committed log records

No WAL protocol

Independent of SQL recovery intervals or background tasks (1.5Gb log increase)

All data written in pairs of data and delta files

Preemptive tasks using WriteFile (async with FILE_FLAG_NO_BUFFERING)

Data is always appended

I/O is continuous but CHECKPOINT ***event*** is based on 1.5Gb log growth

Instant File Initialization matters for PRECREATED files

Data = INSERTs and UPDATEs

Typically 128Mb but can be 1Gb

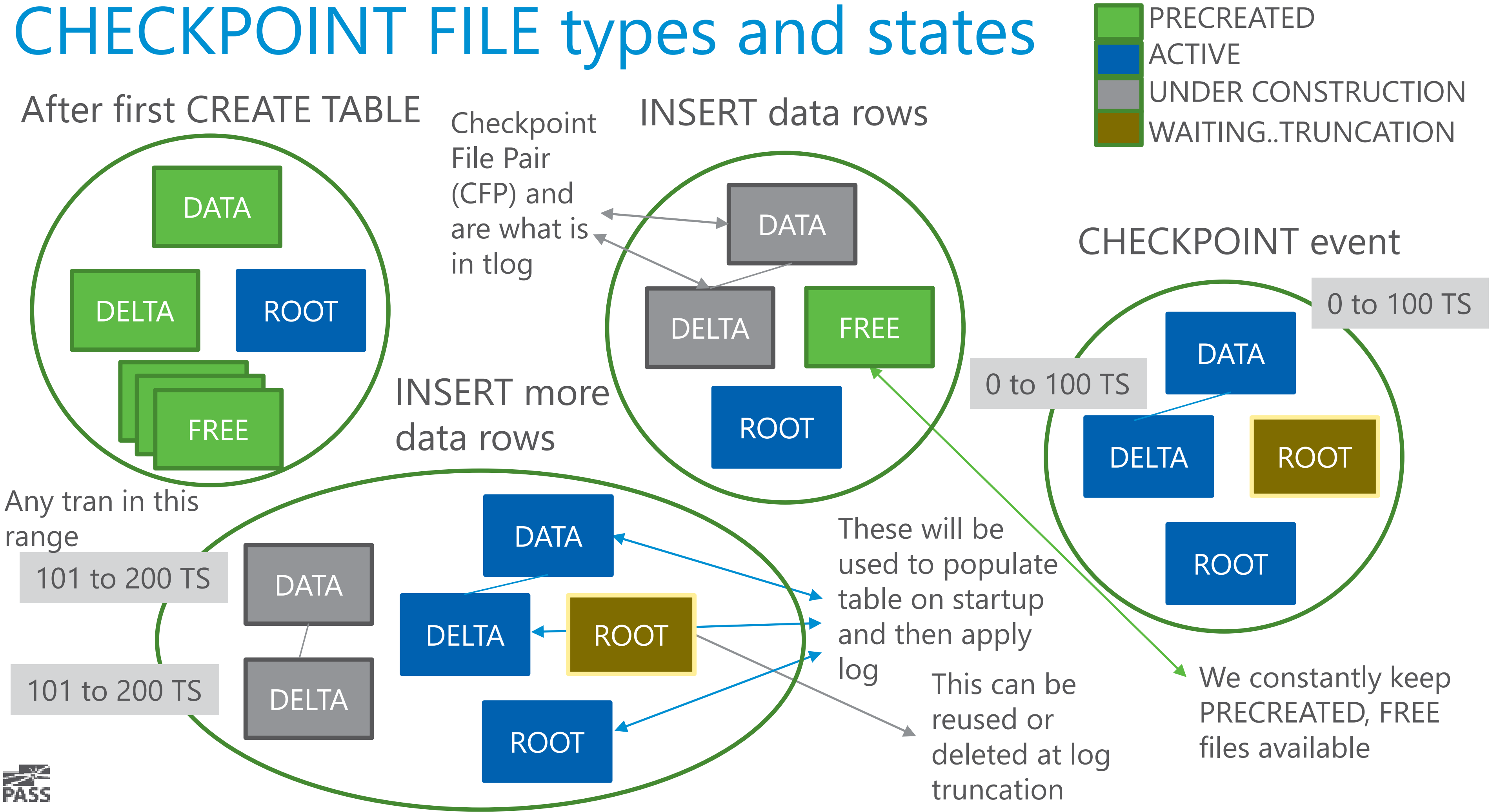
Delta = filter for what rows in Data are actually deleted

Typically 8Mb but can be 128Mb

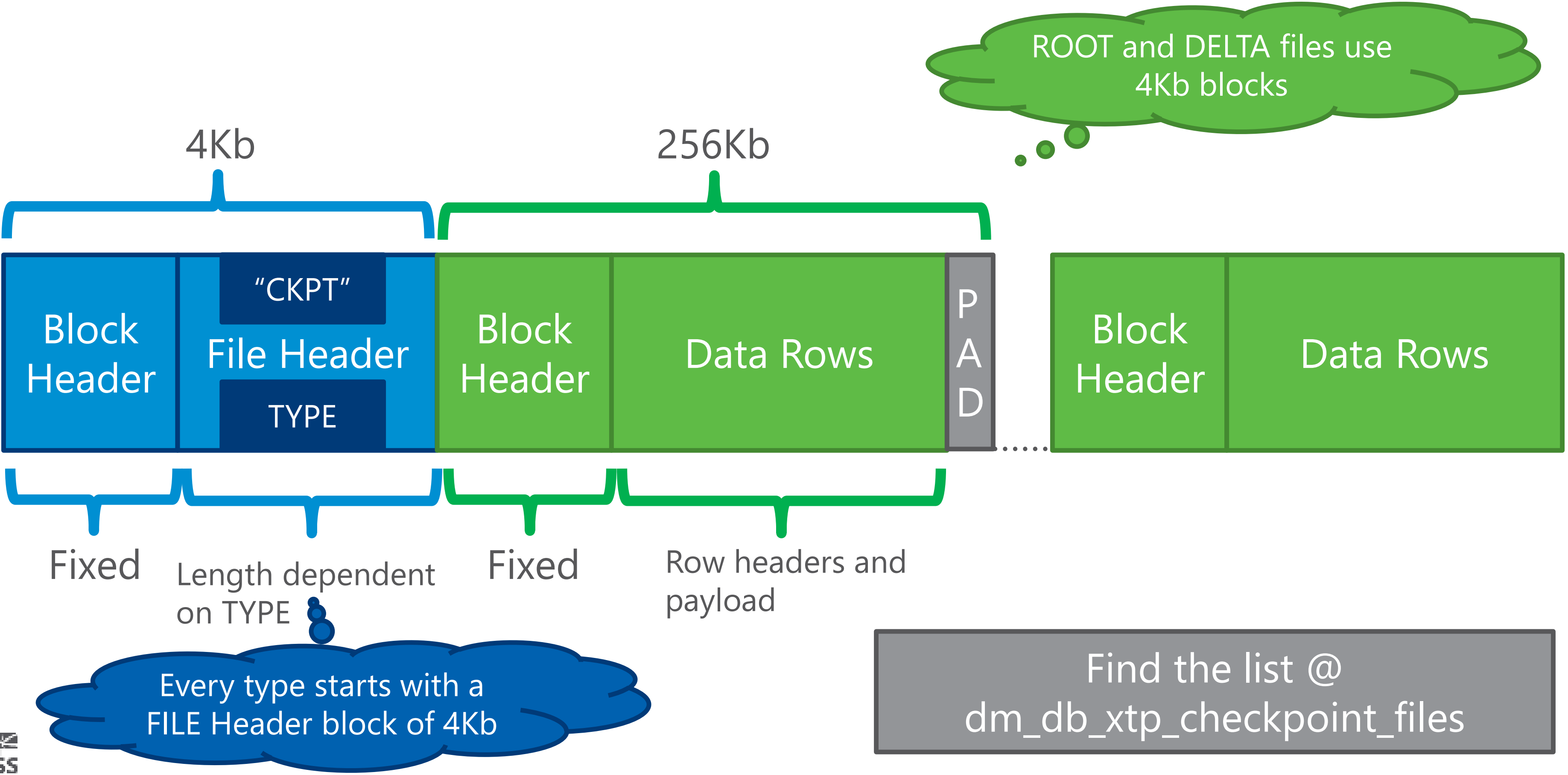
Periodically we will MERGE several Data/Delta pairs.

PASS

CHECKPOINT FILE types and states



CHECKPOINT DATA file format



In-Memory OLTP Recovery

Checksum failures = RECOVERY_PENDING on startup

Read details from database Boot Page (DBINFO)

Load ROOT file for system tables and file metadata

We are a bit ERRORLOG "happy"

Load ACTIVE DATA files filtered by DELTA streamed in parallel

Redo COMMITED transactions greater than last CHECKPOINT LSN

We can create a thread per CPU for this

CHECKPOINT and RECOVERY

Demo

Natively Compiled Procedures

Native = C code vs T-SQL
Built at CREATE PROC time or recompile
Rebuilt when database comes online
'XTP Native DLL' in dm_os_loaded_modules

The structure of a natively compiled proc

Compile this
Into a DLL

Required. No referenced object/column
can be dropped or altered.
No SCH lock required

```
create procedure cowboys_proc_scan
with native_compilation, schemabinding as
begin atomic with
(transaction isolation level = snapshot, language = N'English')
select player_number, player_name from dbo.starsoftheteam
..
end
```

Everything in block a single tran

These are required. There are other options

Your queries

Iso levels still use MVCC

No "*" allowed	Fully qualified names	Bigger surface area in SQL Server 2016	Restrictions and Best Practices are here

The lifecycle of compilation

Done in memory

Mixed Abstract Tree (MAT) built

- Abstraction with flow and SQL specific info
- Query trees into MAT nodes
- XML file in the DLL dir

xtp_matgen XEvent

Converted to Pure Imperative Tree (PIT)

- General nodes of a tree that are SQL agnostic. C like data structures
- Easier to turn into final C code

Final C code built and written

- C instead of C++ is simpler and faster compile.
- No user defined strings or SQL identifiers to prevent injection

Call cl.exe to compile, link, and generate DLL

- Many of what is needed to execute in the DLL
- Some calls into the HK Engine

All files in BINN\XTP

VC has compiler files, DLLs and libs

Gen has HK header and libs

Call cl.exe to compile and link

Debugging a Natively Compiled Proc

Demo

Resource Management

Inside the Memory of In-Memory OLTP

Hekaton implements its own memory management system built on SQLOS

- **MEMORYCLERK_XTP** (DB_ID_<dbid>) uses SQLOS Page allocator
- Variable *heaps* created per table and range index
- Hash indexes using partitioned memory objects for buckets
- "System" memory for database independent tasks
- Memory only limited by the OS (**24TB in Windows Server 2016**)
- Details in **dm_db_xtp_memory_consumers** and **dm_xtp_system_memory_consumers**

Locked and Large apply

Allocated at create index time

In-Memory does recognize SQL Server Memory Pressure

- Garbage collection is triggered
- If OOM, no inserts allowed but you may be able to DELETE to free up space

Resource Governor and In-Memory OLTP

Remember this is ALL memory

SQL Server limits HK to ~70-90% of target depending on [size]

Freeing up memory = drop db (immediate) or delete data/drop table (deferred)

Binding to your own Resource Pool

no classifier function

Best way to control and monitor memory usage of HK tables

sp_xtp_bind_db_resource_pool required to bind db to RG pool

What about CPU and I/O?

RG I/O does not apply because we don't use SQLOS for checkpoint I/O

RG CPU does apply for user tasks because we run under SQLOS host

Not the same as memory

Let’s Wrap It Up

Walk away with this

- ✓ Want 30x faster for OLTP? Go In-Memory OLTP
- ✓ Transactions are truly lock, latch, and spinlock free
- ✓ Hash index for single row; Range for multi-row
- ✓ Reduce transaction latency for super speed
- ✓ SQL Server 2016 major step up from 2014

We are not done

- Move to GA for Azure SQL Database
- Increase surface area of features support for SQL Server
- Push the limits of hardware
  - Speeding up recovery and HA with NVDIMM and RDMA
- Explore further areas of code optimization
  - SQL 2016 CU2 fix for session state workloads Read more here
  - Range index performance enhancements

PASS

Resources

- SQL Server In-Memory OLTP Internals for SQL Server 2016
- In-Memory OLTP Videos: What it is and When/How to use it
- Explore In-Memory OLTP architectures and customer case studies
- Review In-Memory OLTP in SQL Server 2016 and Azure SQL Database
- In-Memory OLTP (In-Memory Optimization) docs

Bonus Material

Troubleshooting

From the CSS team

Validation errors. Check out this blog

Upgrade from 2014 to 2016 can take time

Large checkpoint files for 2016 could take up more space and increase recovery times

Log growths, XTP_CHECKPOINT waits. Hotfix 6051103

Checkpoint file shut down and with no detailed info: https://support.microsoft.com/en-us/kb/3090141

Unable to rebuild log 6424109 (by design)

Set filegroup offline. NEVER do it because you can't set it online again (filestream limitation)

You can't remove HK filegroup after you add it 2016 LOB can cause memory issues

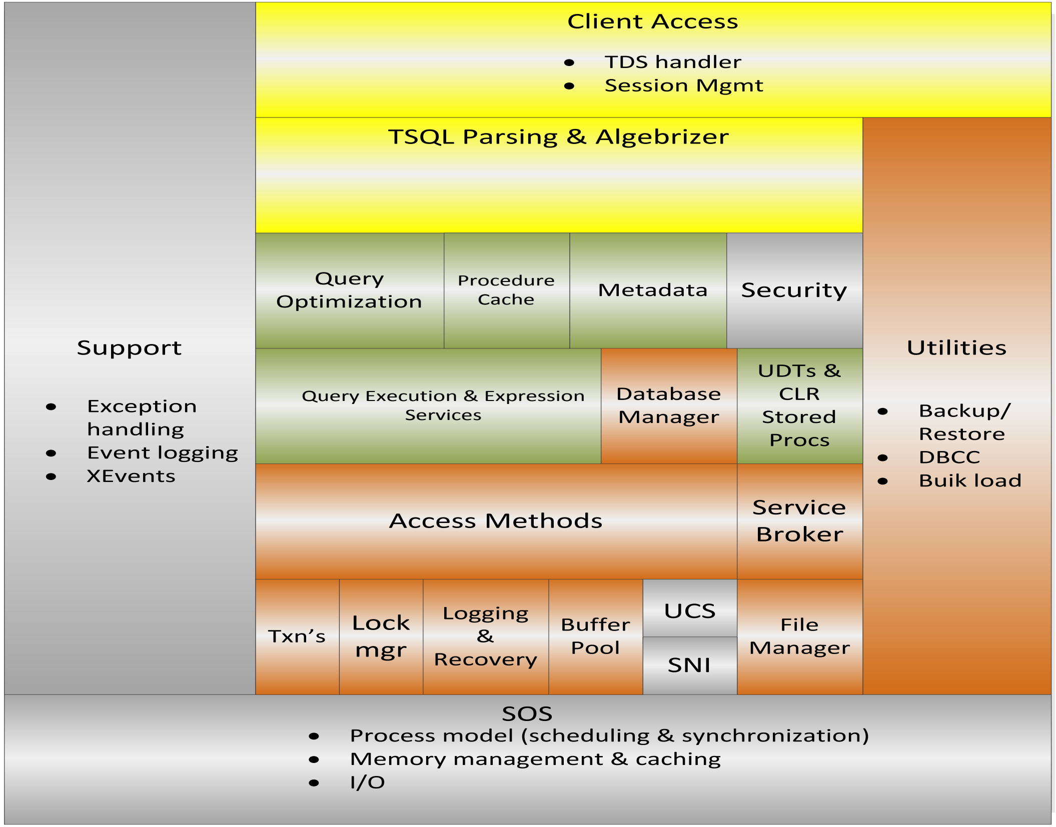
Other CSS Observations

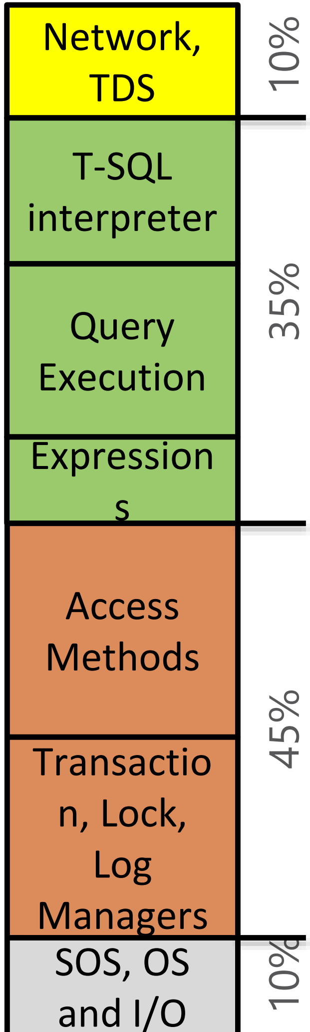
- Hash index doesn't have a concept of "covered index"
- Low memory can cause recovery to fail (because we need everything in memory)
- No dbcc checkdb support (but checkpoint files and backup have checksum)

Architecture Pillars

Customer Benefits	High performance data operations	Efficient, business-logic processing	Frictionless scale-up	Hybrid engine and integrated experience
Hekaton Tech Pillars	**Main-Memory Optimized** • Optimized for in-memory data • Indexes (hash and range) exist only in memory • No buffer pool • Stream-based storage for durability	**T-SQL Compiled to Machine Code** • T-SQL compiled to machine code via C code generator and VC • Invoking a procedure is just a DLL entry-point • Aggressive optimizations @ compile-time	**High Concurrency** • Multi-version optimistic concurrency control with full ACID support • Core engine uses lock-free algorithms • No lock manager, latches or spinlocks	**SQL Server Integration** • Same manageability, administration & development experience • Integrated queries & transactions • Integrated HA and backup/restore
Drivers	**Hardware trends**			**Business**
	Steadily declining memory price, NVRAM	Stalling CPU clock rate	Many-core processors	TCO

The Challenge





The In-Memory OLTP Thread Model

SQLOS task and worker threads are the foundation

“User” tasks to run transactions

Hidden schedulers used for critical background tasks

Some tasks dedicated while others use a “worker” pool

SQLOS workers dedicated to HK

In-Memory OLTP Thread Pool

H hidden scheduler

U user scheduler

NON-PREEMPTIVE POOL U

Background workers needed for on-demand

Parallel ALTER TABLE

WORKERS POOL U

Other workers needed for on-demand (non-preemptive)

Parallel MERGE operation of checkpoint files

PREEMPTIVE POOL H

Background workers needed on-demand - preemptive

Serializers to write out checkpoint files

Command = XTP_THREAD_POOL wait_type = DISPATCHER_QUEUE_SEMAPHORE

Command = UNKNOWN TOKEN wait_type = XTP_PREEMPTIVE_TASK

Ideal count = < # schedulers> ; idle timeout = 10 secs

Data Modifications

- Multi-version Optimistic Concurrency prevents all blocking
- ALL UPDATEs are DELETE followed by INSERT
- DELETED rows not automatically removed from memory
- Deleted rows not visible to active transactions becomes ***stale***
- Garbage Collection process removes stale rows from memory
- TRUNCATE TABLE not supported

Page deallocation in SQL Server

Garbage Collection

Stale rows to be removed queued and removed by multiple workers

User workers can clean up stale rows as part of normal transactions

Dedicated threads (GC) per NODE to cleanup stale rows (awoken based on # deletes)

- SQL Server Memory pressure can accelerate
- Dusty corner scan cleanup – ones no transaction ever reaches

This is the only way to reduce size of HK tables

Long running transactions may prevent removal of deleted rows (visible != stale)

Diagnostics

- **rows_expired** = rows that are stale
- **rows_expired_removed** = stale rows removed from memory
- Perfmon -SQL Server 2016 XTP Garbage Collection
- dm_db_xtp_gc_cycle_stats
- dm_xtp_gc_queue_stats

Diagnostics for Native Procs

Read the docs

Query Plans and Stats

SHOWPLAN_XML is only option

Operator list can be found here

sp_xtp_control_query_exec_stats and sp_xtp_control_proc_exec_stats

Query Store

Plan store by default

Runtime stats store needs above stored procedures to enable (need recompile)

XEvent and SQLTrace

sp_statement_completed works for Xevent but more limited information

SQLTrace only picks up overall proc execution at batch level

Undo, tempdb, and BULK

Multi-row insert test

- Undo required 415 log records @46Kb
  - Hekaton required no logging
- Tempdb has similar # log records as disk-based table but less per log record (aka minimal logging).

Remember SCHEMA_ONLY has no logging or I/O

BULK INSERT for in-mem executes and logged just like INSERT

Minimally logged BULK INSERT took 271 log records @ 27Kb

Latches required for GAM, PFS, and system table pates

In-Memory OLTP pushes the log to the limit

Multiple Log Writers in SQL Server 2016

One per NODE up to four

We were able to increase log throughput from 600 to 900Mb/sec

You could go to delayed durability

Database is always consistent

You could lose transactions

Log at the speed of memory

Tail of the log is a “memcpy” to commit

Watch the video

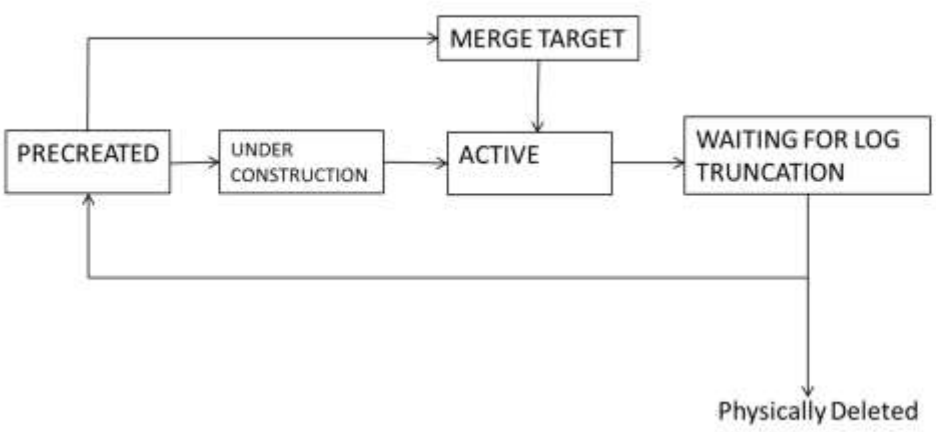
The Merge Process

Over time we could have many CFPs

Why not consolidate data and delta files into a smaller number of pairs?

MERGE TARGET type is target of the merge of files. Becomes ACTIVE

ACTIVE files that were source of merge become WAITING FOR LOG TRUNCATION



Bob Ward

Principal Architect, Microsoft

Bob Ward is a Principal Architect for the Microsoft Data Group (Tiger Team). Bob has worked for Microsoft for 23 years supporting and speaking on every version of SQL Server shipped from OS/2 1.1 to SQL Server 2016. He has worked in customer support as a principal escalation engineer and Chief Technology Officer (CTO) interacting with some of the largest SQL Server deployments in the world. Bob is a well-known speaker on SQL Server often presenting talks on internals and troubleshooting at events such as SQL PASS Summit, SQLBits, SQLIntersection, and Microsoft Ignite. You can find him on twitter at @bobwardms or read his blog at **https://blogs.msdn.microsoft.com/bobsql**..

in /bobwardms   @bobwardms   f Bob Ward

Explore Everything PASS Has to Offer

24 HOURS OF PASS

FREE ONLINE WEBINAR EVENTS

PASS SQL saturday

FREE 1-DAY LOCAL TRAINING EVENTS

VOLUNTEERING OPPORTUNITIES

PASS CHAPTERS

LOCAL USER GROUPS AROUND THE WORLD

PASS VIRTUAL CHAPTERS

ONLINE SPECIAL INTEREST USER GROUPS

PASS CommunityCONNECTOR

PASS COMMUNITY NEWSLETTER

PASS BLOG

WHITE PAPERS

SESSION RECORDINGS

FREE ONLINE RESOURCES

PASS BUSINESS ANALYTICS CONFERENCE 2016

BUSINESS ANALYTICS TRAINING

BA INSIGHTS NEWSLETTER

PASS

Session Evaluations

Your feedback is important and valuable.

3

ways to access

Submit by 5pm Friday November 6th to **WIN** prizes

Go to **passSummit.com**

Download the GuideBook App and search: **PASS Summit 2016**

Follow the QR code link displayed on session signage throughout the conference venue and in the program guide

Thank You

Learn more from

Bob Ward

bobward@company.com or follow @bobwardms