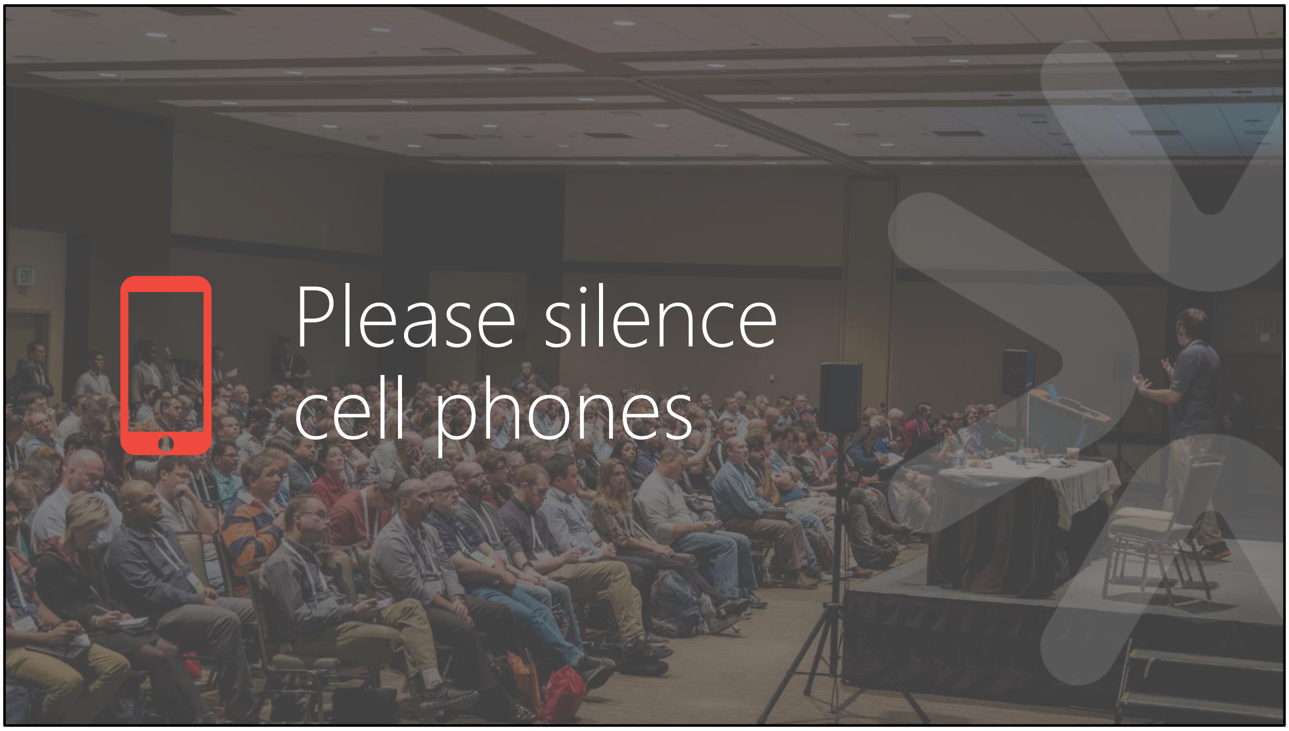
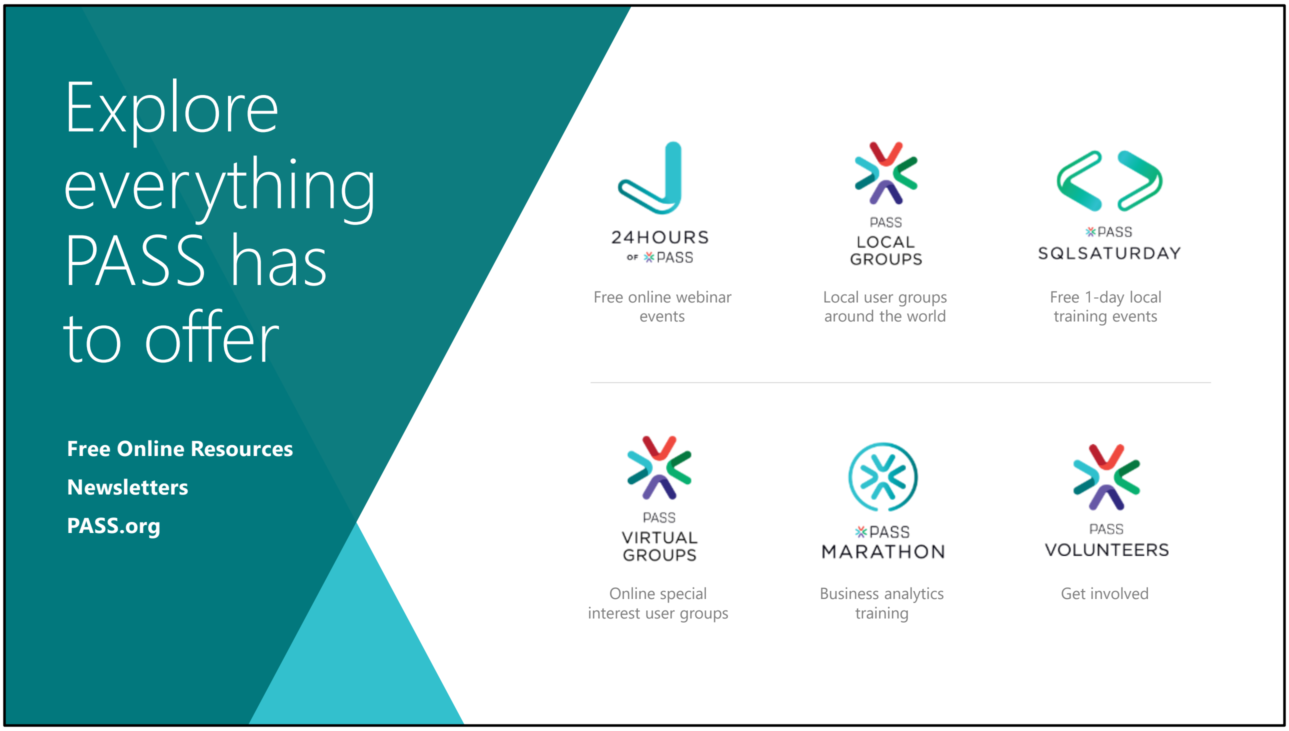
PASS SUMMIT2018

TempDB – The Good, The Bad, and The Ugly

Pam Lahoud, Sr. Program Manager, Microsoft



Please silence cell phones



Explore everything PASS has to offer

Free Online Resources

Newsletters

PASS.org

24HOURS
OF PASS

Free online webinar events

PASS
LOCAL
GROUPS

Local user groups around the world

PASS
SQLSATURDAY

Free 1-day local training events

PASS
VIRTUAL
GROUPS

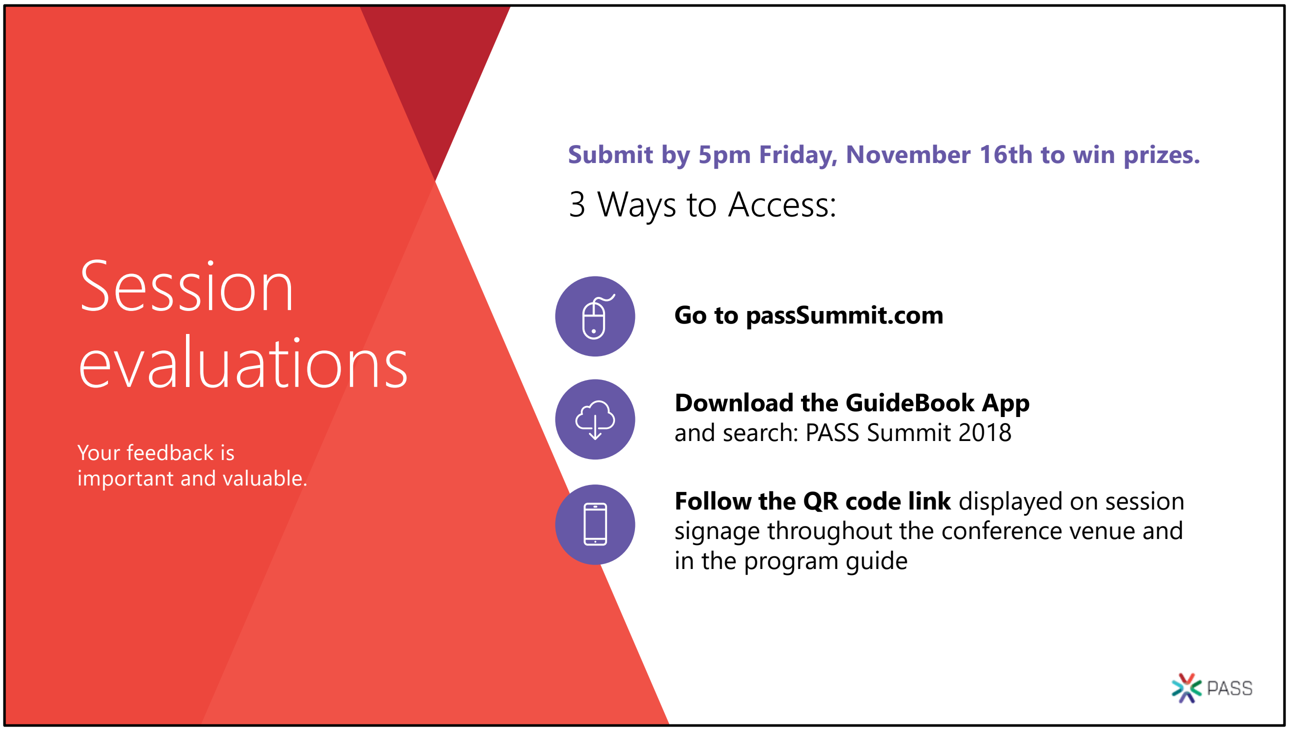
Online special interest user groups

PASS
MARATHON

Business analytics training

PASS
VOLUNTEERS

Get involved



Session evaluations

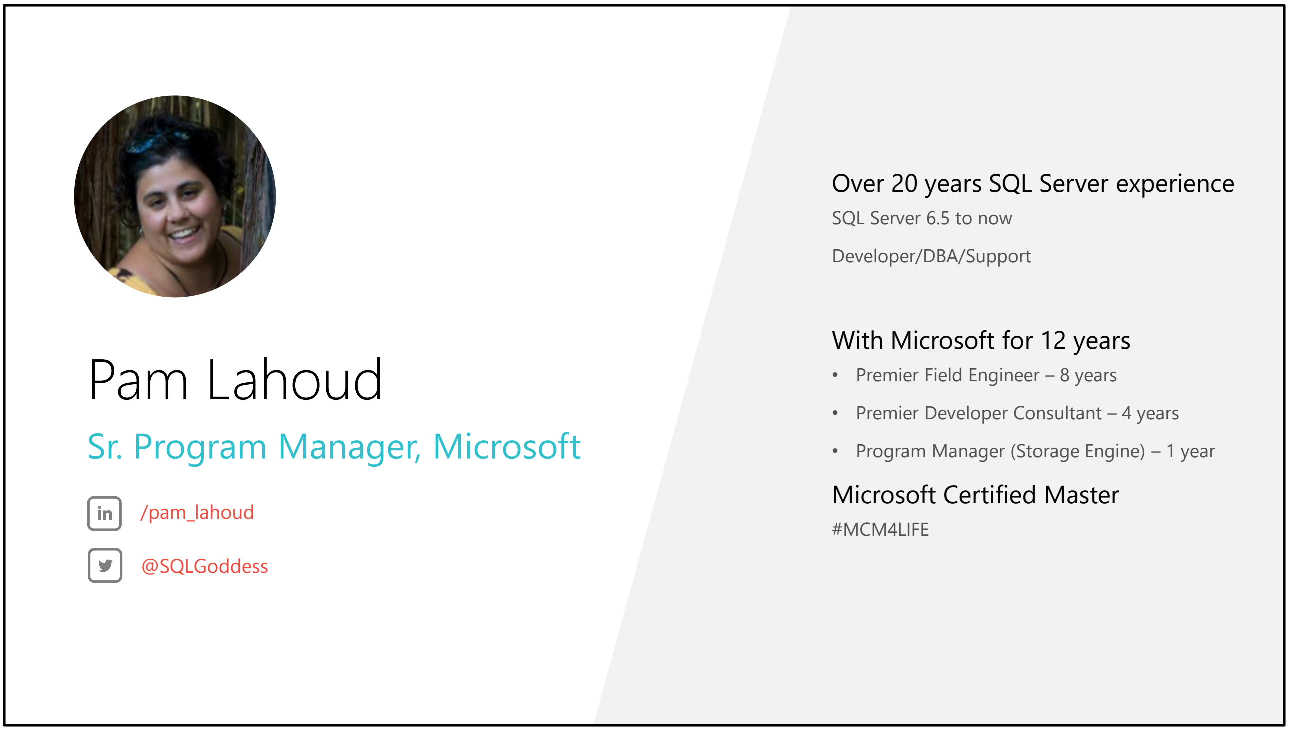
Your feedback is
important and valuable.

Submit by 5pm Friday, November 16th to win prizes.

3 Ways to Access:

- **Go to passSummit.com**
- **Download the GuideBook App**
  and search: PASS Summit 2018
- **Follow the QR code link** displayed on session signage throughout the conference venue and in the program guide

PASS



Pam Lahoud

Sr. Program Manager, Microsoft

/pam_lahoud

@SQLGoddess

Over 20 years SQL Server experience

SQL Server 6.5 to now

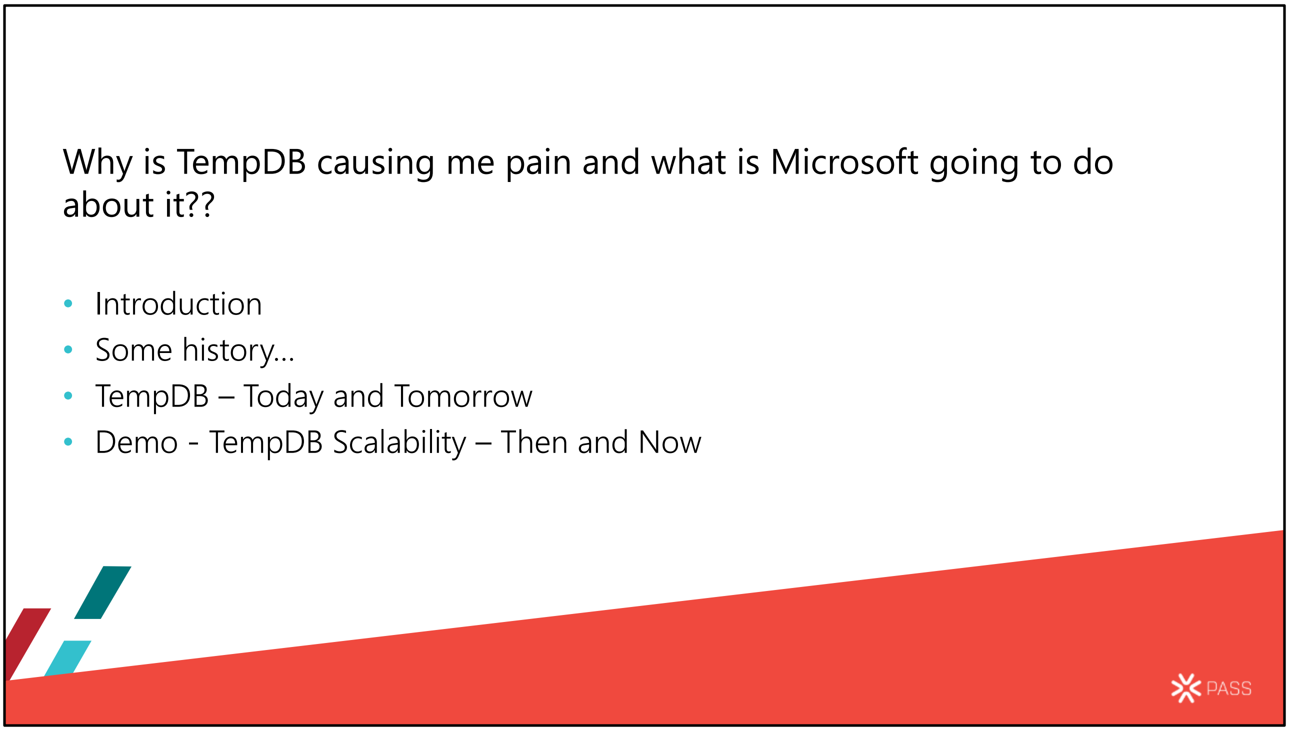
Developer/DBA/Support

With Microsoft for 12 years

- Premier Field Engineer – 8 years
- Premier Developer Consultant – 4 years
- Program Manager (Storage Engine) – 1 year

Microsoft Certified Master

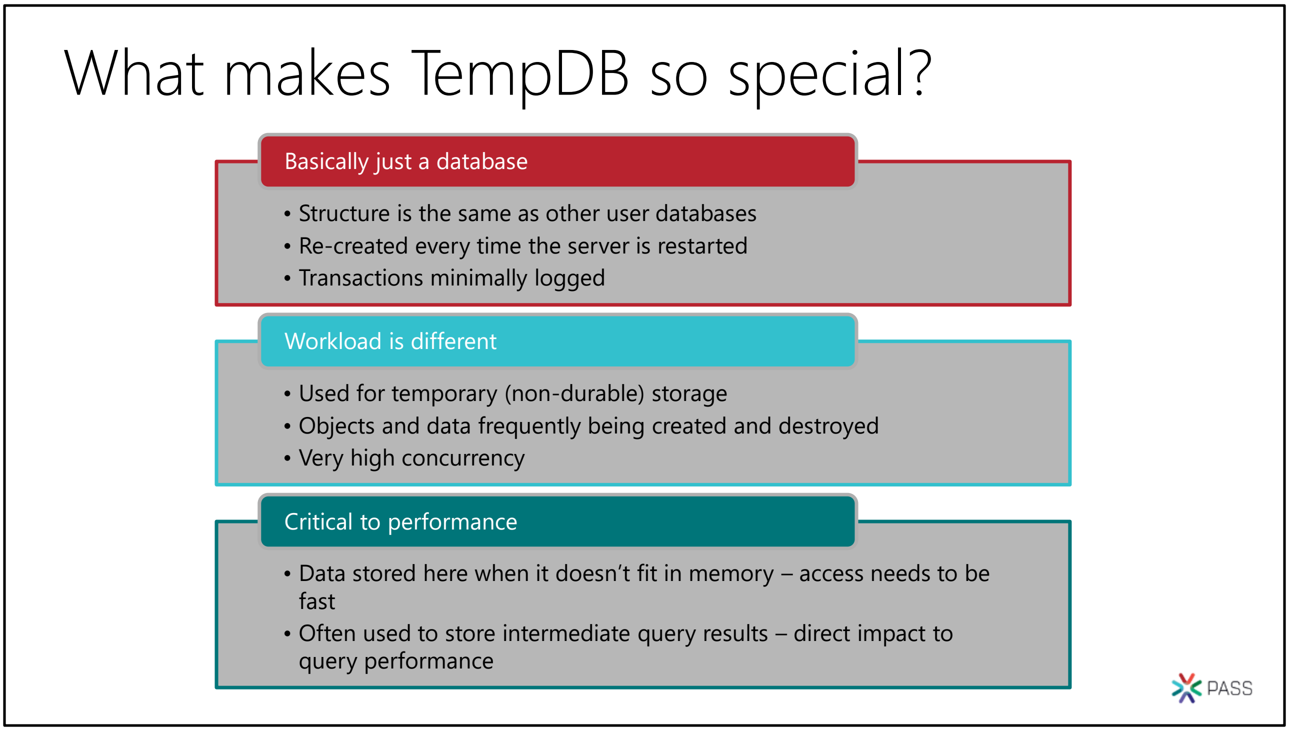
#MCM4LIFE



Why is TempDB causing me pain and what is Microsoft going to do about it??

- Introduction
- Some history…
- TempDB – Today and Tomorrow
- Demo - TempDB Scalability – Then and Now

Introduction



What makes TempDB so special?

Basically just a database

- Structure is the same as other user databases
- Re-created every time the server is restarted
- Transactions minimally logged

Workload is different

- Used for temporary (non-durable) storage
- Objects and data frequently being created and destroyed
- Very high concurrency

Critical to performance

- Data stored here when it doesn't fit in memory – access needs to be fast
- Often used to store intermediate query results – direct impact to query performance

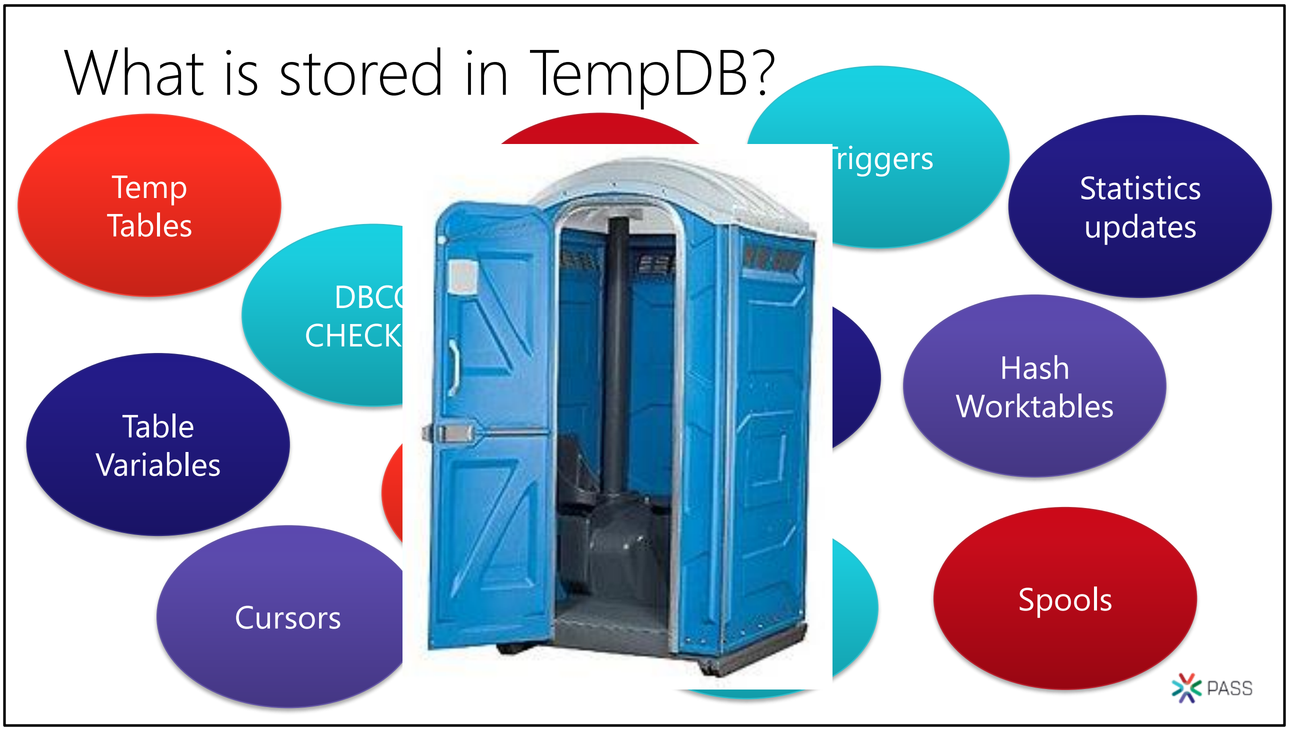
The primary purpose of tempdb is in the name – temporary storage. It's used by all sorts of things that need some scratch space, particularly things that don't fit in memory and need to spill to disk. For the most part, it's information that we only need to store for a short period of time – just until we're ready to read it back into memory.

You might be wondering what makes tempdb such a big deal? Why is everyone so worried about it? Isn't it just a regular database like all the others on the server?

The answer is yes. And no. At it's core, tempdb is just another database on the server, pretty much the same as all your user databases. One of the big differences with tempdb is that it's not durable. Every time you restart SQL Server, it gets re-created from scratch based on the configuration it had before the service was stopped. Since it's not durable, all the transactions are minimally logged, just enough so that we can rollback any transactions that involve temporary objects. The big difference with tempdb is the workload. Because of it's transient nature, objects in tempdb are CONSTANTLY being created and destroyed on the fly. It's a very different pattern than you see with your user databases for the most part. In "regular" databases, the tables and indexes get created up front and only the data changes over time. You might have a few new tables added or old tables removed once in a while, but this doesn't happen frequently, and certainly not several times a second.

Because of this unique workload, tempdb experiences stresses and strains that other databases on the server do not.

But that doesn’t completely answer the question. If the data is only stored for a short period of time, and the whole thing gets thrown away on restart, why is it so important? The reason is performance. Tempdb is primarily being used to store things that we would normally want to keep in memory, but that won’t fit for whatever reason. This means we need the access to be fast. Any delays that are experienced in tempdb can have wide-ranging implications on the instance. It’s also used to store things like temporary query results, so again, we need it to be fast or it’s gong to slow down our queries. Given that it’s in the critical path for performance, and it has unique contention points because of the workload, it requires special care. Both on your part with configuration, and on our part in optimizing the code as best we can.



What is stored in TempDB?

Temp Tables

DBCC CHECK

Table Variables

Cursors

Triggers

Statistics updates

Hash Worktables

Spools

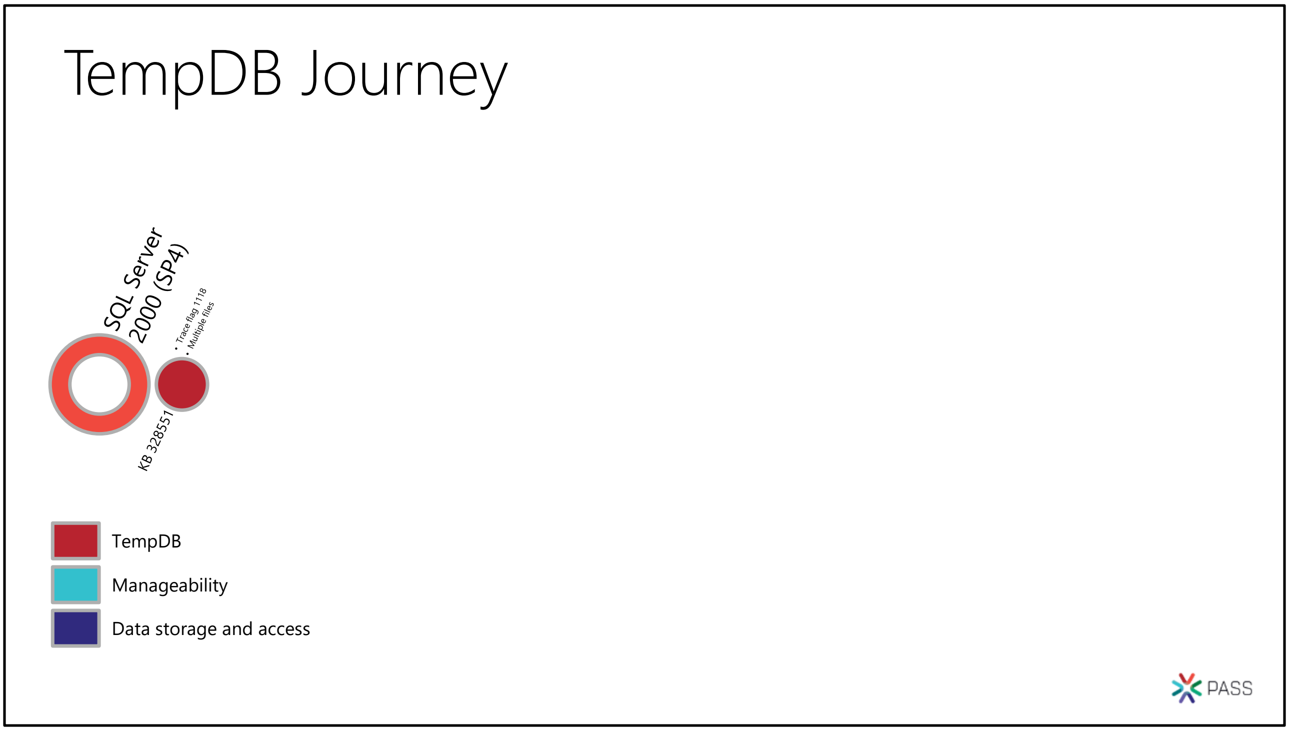
The **tempdb** system database is a global resource that is available to all users connected to the instance of SQL Server or connected to SQL Database. Tempdb is used to hold:

- Temporary **user objects** that are explicitly created, such as: global or local temporary tables and indexes, temporary stored procedures, table variables, Tables returned in table-valued functions, or cursors.
- **Internal objects** that are created by the database engine. These include:
    - Work tables to store intermediate results for spools, cursors, sorts, and temporary large object (LOB) storage.
    - Work files for hash join or hash aggregate operations.
    - Intermediate sort results for operations such as creating or rebuilding indexes (if SORT_IN_TEMPDB is specified), or certain GROUP BY, ORDER BY, or UNION queries.
- **Version stores**, which are a collection of data pages that hold the data rows that are required to support the features that use row versioning. There are two version stores: a common version store and an online-index-build version store. The version stores contain:
    - Row versions that are generated by data modification transactions in a database that uses read-committed using row versioning isolation or snapshot isolation transactions.

- Row versions that are generated by data modification transactions for features, such as: online index operations, Multiple Active Result Sets (MARS), and AFTER triggers.

There's only one tempdb on an instance of SQL Server and everyone uses it at some point. It has been referred to by some as the public toilet of SQL Server :-P

Some history…



TempDB Journey

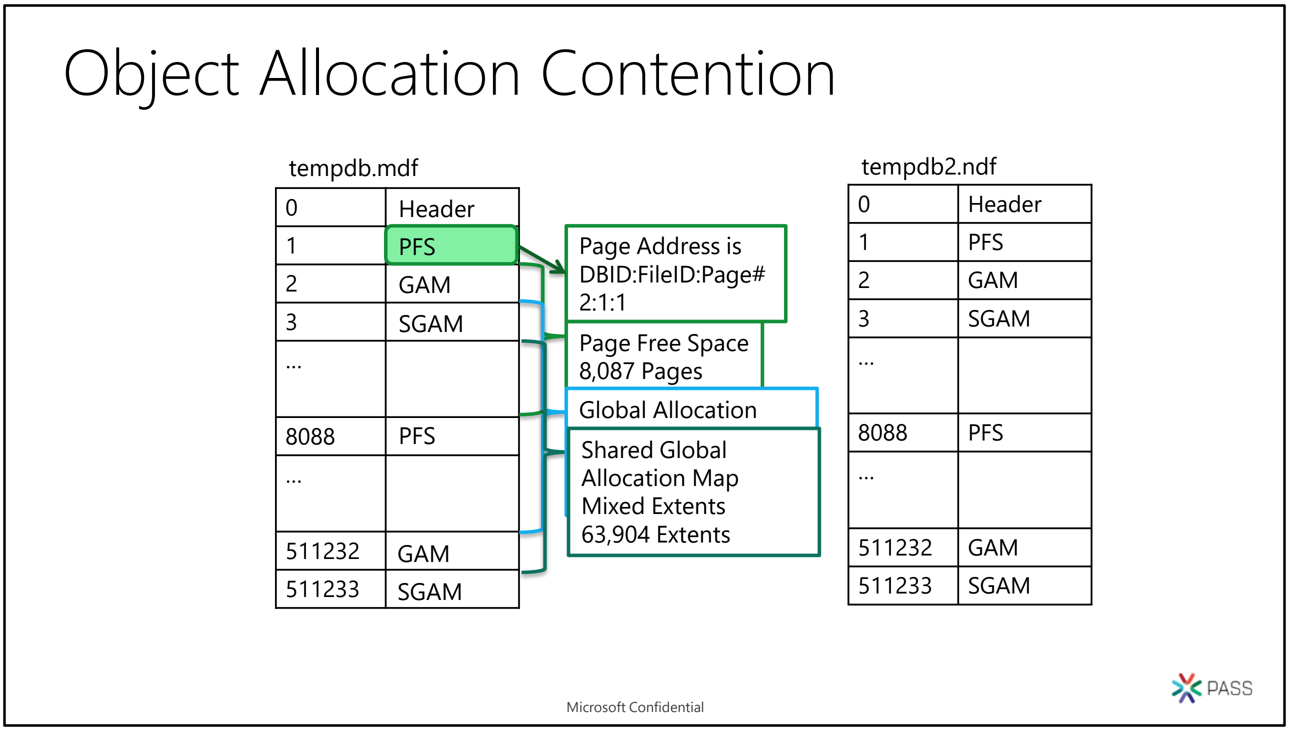
🟥 TempDB

🟦 Manageability

🟪 Data storage and access

For the storage engine, we are tracing a few different journeys: TempDB, Manageability, Data storage and access, File management and I/O.

For TempDB, our journey begins in SQL Server 2000. It was this time where 64-bit servers (both IA and x64 if you remember ☺) with a large number of cores become more common. NUMA machines start to emerge in this time frame as well. Along with this new hardware, new concurrency issues emerged in tempdb in two areas: object allocation and metadata contention. To address this, with SP4 we first introduced trace flag 1118 and the recommendation to create multiple data files in tempdb, at this point one per core was the recommendation.



Object Allocation Contention

tempdb.mdf

0	Header
1	PFS
2	GAM
3	SGAM
…	
8088	PFS
…	
511232	GAM
511233	SGAM

Page Address is
DBID:FileID:Page#
2:1:1

Page Free Space
8,087 Pages

Global Allocation

Shared Global
Allocation Map
Mixed Extents
63,904 Extents

tempdb2.ndf

0	Header
1	PFS
2	GAM
3	SGAM
…	
8088	PFS
…	
511232	GAM
511233	SGAM

Microsoft Confidential

SQL Server databases are composed of 64 kilobytes (KB) extents

Extents are the basic unit used by SQL Server to manage space. An extent is eight physically contiguous pages, or 64 kilobytes (KB). SQL Server has two types of extents:

- Uniform extents are owned by a single object, and only the owning object can use the eight pages in the extent.
- Mixed extents contain eight pages that can be used by different objects.

Each extent is composed of eight pages of 8 KB each

To make its space allocation efficient, SQL Server does not allocate an entire extent to tables that contain small amounts of data. A new table or index is usually allocated using pages from mixed extents. When the table or index grows to the point where it needs to allocate more than 8 pages, then all allocations of new space will be done in uniform extents, but the original pages on mixed extents will not be moved into uniform extents. If you create an index on an existing table that has enough rows to generate eight pages in the index, all allocations of new space are uniform extents.

Allocation maps track allocations

SQL Server uses two types of allocation maps to record the allocation of extents:

- Global Allocation Map (GAM). GAM pages record all extents. Each GAM covers 63,904 extents, or nearly 4 gigabytes (GB) of data. The GAM has one bit for each extent in the interval that it covers. If the bit is 1, the extent is free; if the bit is 0, the extent is allocated.
- Shared Global Allocation Map (SGAM). SGAM pages record those extents that are currently used as mixed extents and have at least one unused page. Each SGAM covers 63,904 extents, or nearly 4 GB of data. The SGAM has one bit for each extent in the interval that it covers. If the bit is 1, the extent is being used as a mixed extent and has free pages; if the bit is 0, the extent is not used as a mixed extent, or it is a mixed extent whose all pages are in use.

When a database has multiple files, the files are numbered from 1 to n, n being the total number of files. Each file consists of a collection of pages numbered from 0 to n-1, n being the number of pages in the file (total number of pages will depend on the size of the file).

The special Object Allocation pages will always appear in the same location in each data file contained in a database. A PFS page is the first page after the file header page in a data file and is marked as page 1. Following the PFS page is a GAM, marked as page 2, followed by an SGAM, marked as page 3. After the first PFS, there will be a new PFS page after every 8,087 pages. There is a new GAM for each 63,904 extents after the first GAM on page 2, and a new SGAM each 63,904 extents after the first SGAM on page 3.

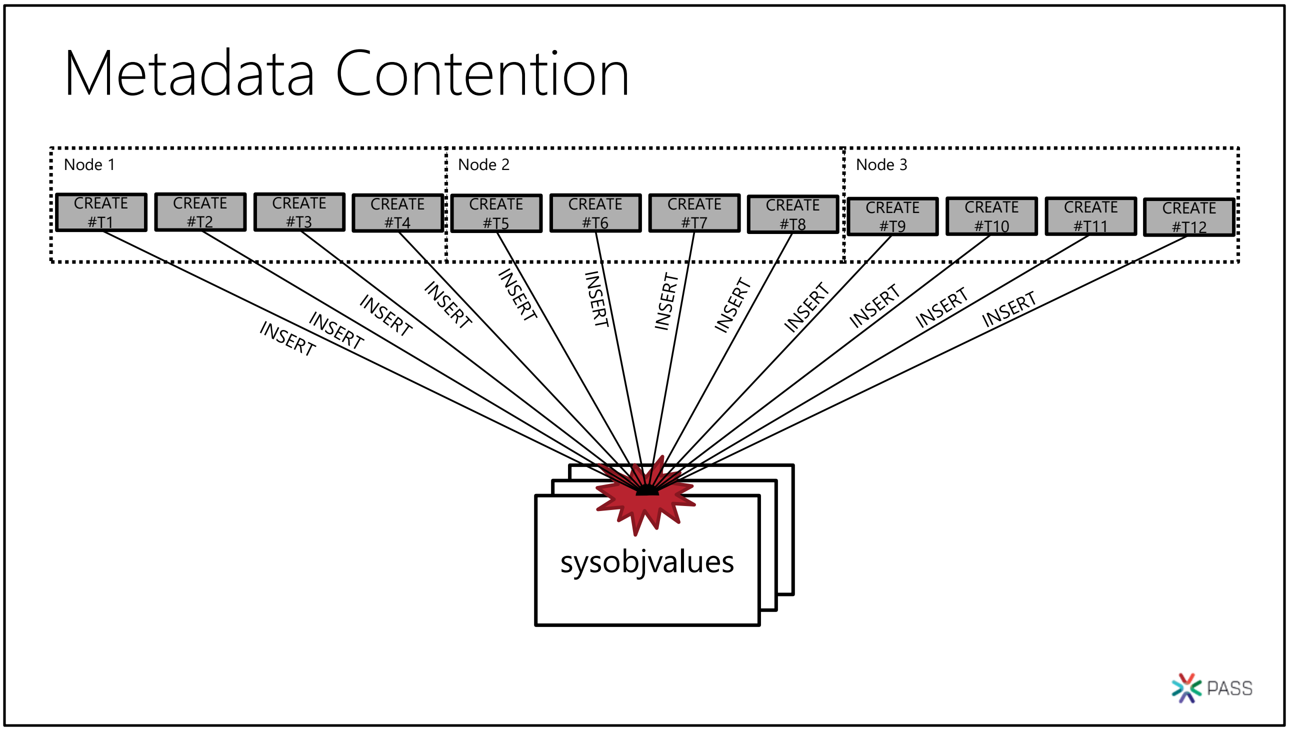
The SQL Server engine addresses each of these pages using a combination of database id, file id, and page number. For example, the SGAM page in the first data file of tempdb (which has a database ID of 2) will be notated as **2:1:3**. This becomes important when troubleshooting any type of page-related wait as the wait resource will be listed using this notation.

Because of the frequent creation of objects in tempdb, this database in particular is subject to a bottleneck on these object allocation pages, most commonly the PFS pages. The first allocation for any new object in SQL Server is a mixed extent by default, so tempdb will also often see additional contention on the SGAM page. This problem became a critical performance issue in SQL Server 2000, so it was this timeframe where the following recommendations were first made:

1. Create multiple data files
   You should start with 1 tempdb data file per CPU core up to 8 files. You may need to add more depending on your workload. All the files must be equally sized.
2. If you are on SQL Server 2014 or earlier, turn on trace flags 1117 and 1118 (this behavior is the default for tempdb in SQL Server 2016).

By having multiple files that are equally sized, each new object allocation hits a different file, and thus a different PFS and SGAM page, in a round robin fashion. Trace flag 1118 turns off mixed extent allocations in all databases on the server, so even the first object allocation goes to a uniform extent. This reduces SGAM contention because we will allocate a full extent up front rather than making 8 separate page allocations to mixed extents. Trace flag 1117 causes all files in the same filegroup to autogrow at the same time, this ensures that the files remain the same size which is required to maintain the round-robin algorithm. If the files are not the same size, SQL Server uses a proportional fill algorithm, so the file with the highest percentage of empty space (e.g. the one that just grew) will get hit more often and the contention may return.

Keep in mind that having multiple files can cause fragmentation of objects when doing something like a SELECT INTO. This means you'll have some extra overhead for operations that have to scan the entire object. If you're still using spinning media, put the files on separate drives to help improve throughput.



Metadata Contention

Node 1

CREATE #T1 CREATE #T2 CREATE #T3 CREATE #T4

Node 2

CREATE #T5 CREATE #T6 CREATE #T7 CREATE #T8

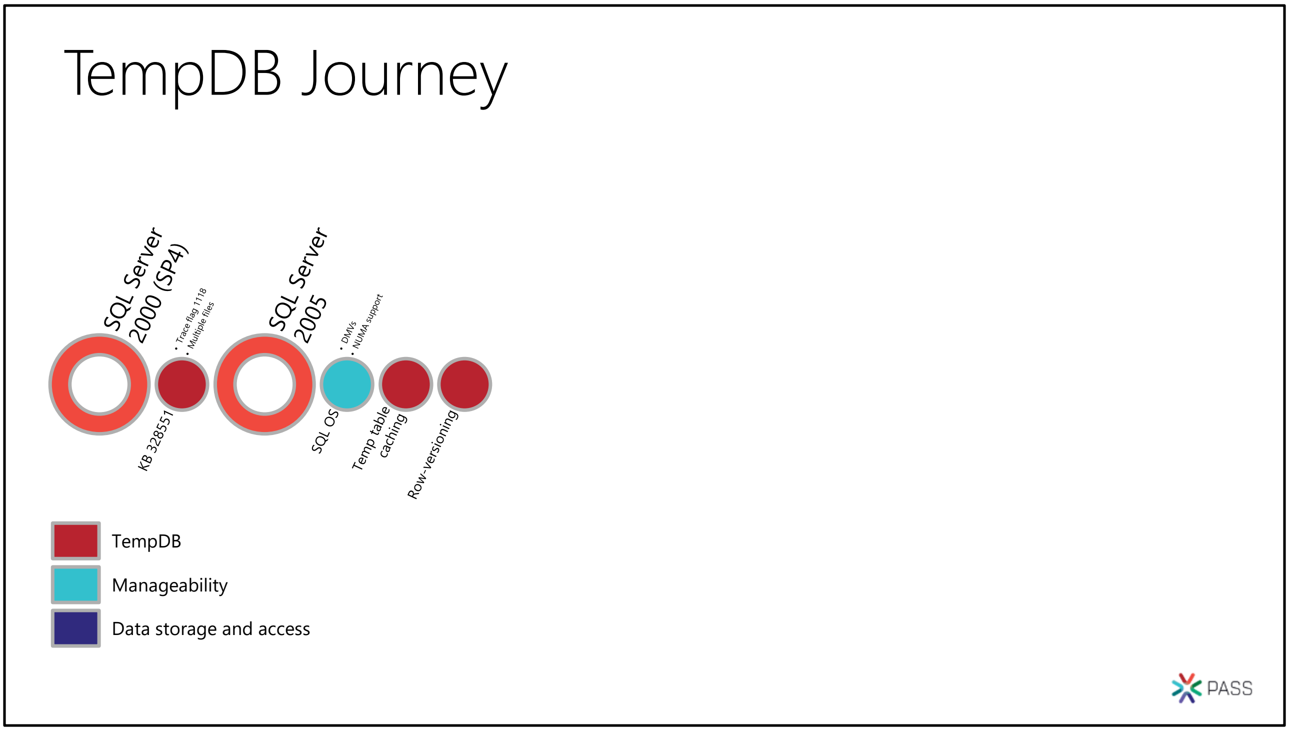
Node 3

CREATE #T9 CREATE #T10 CREATE #T11 CREATE #T12

INSERT INSERT INSERT INSERT INSERT INSERT INSERT INSERT INSERT INSERT INSERT INSERT

sysobjvalues

Along with the object allocation contention that emerged in SQL Server 2000, contention on tempdb metadata tables also began to be observed. Object allocation contention occurs during the process of allocating space for new objects, metadata contention occurs when we are writing to the system tables that store the metadata for the objects being created. The metadata structures in SQL Server databases are not as simple as a single table, there are many system tables that need to have rows added whenever an object is created (there are something like 90, about a dozen are used regularly). When metadata contention occurs, it will manifest itself similar to object allocation contention in that you will see PAGELATCH waits against tempdb pages. The difference here is that rather than seeing pages 2:1:1 and 2:1:3 in contention, you'll see seemingly random page numbers. The only way to diagnose that this is metadata contention is to execute a DBCC PAGE command against these pages and see which objects they belong to. If they are system tables, you're seeing metadata contention.



TempDB Journey

SQL Server 2000 (SP4)

- Trace flag 1118
- Multiple files

KB 328551

SQL Server 2005

- DMVs
- NUMA support

SQL OS

Temp table caching

Row-versioning

🟥 TempDB

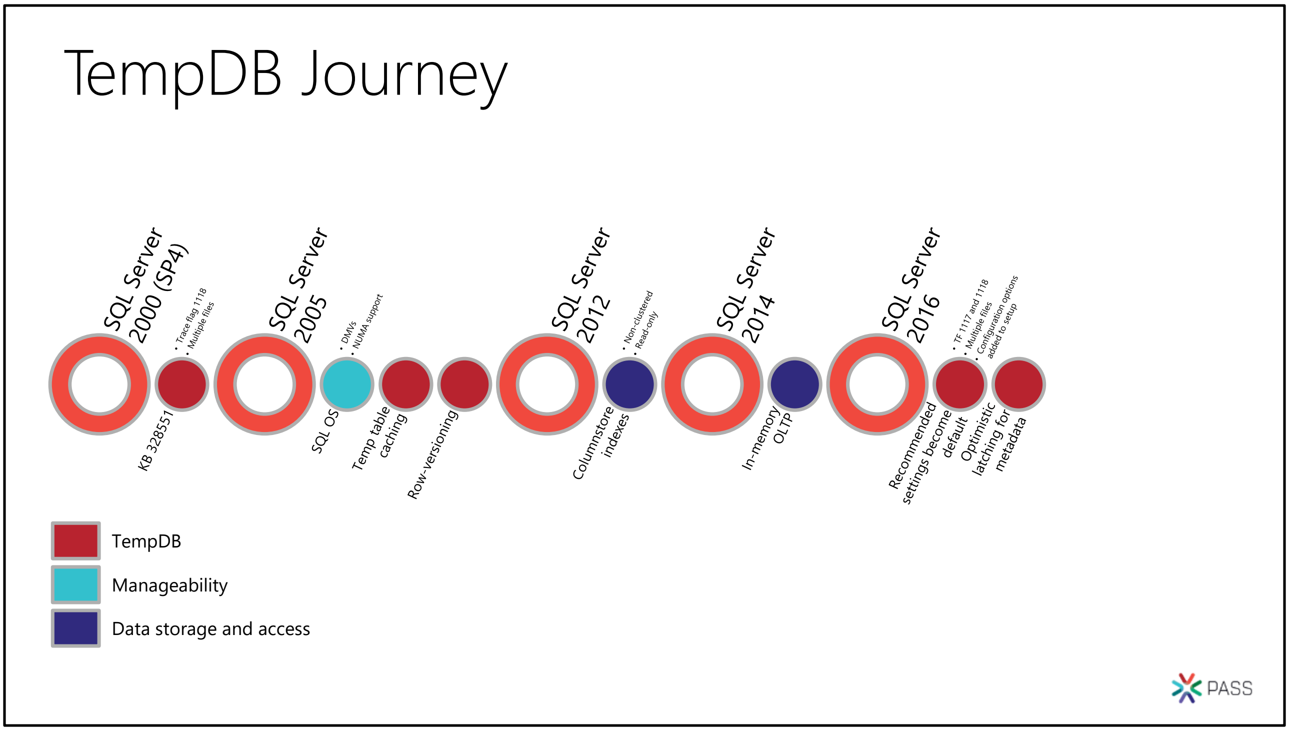
🟦 Manageability

🟪 Data storage and access

In SQL Server 2005 we had a major overhaul in the SQL Server engine which resulted in many of the aspects of SQL Server that we still have today. Not only does our tempdb journey continue, this is where the manageability story begins:

1. The SQLOS (DMVs, full NUMA support, new memory manager, new scheduler etc.)
2. SQL Server Management Studio replaces the old Enterprise Manager and Query Analyzer to provide a single tool for both DBAs and developers to manage and maintain SQL Server
3. Temp table caching which attempted to address the metadata contention we saw in SQL 2000
4. Row-versioning (to support Snapshot and Read-committed Snapshot Isolation, triggers etc.) which added more load to tempdb

The introduction of temp table caching together with the best practice recommendations and trace flags introduced in SQL Server 2000 and reinforced in 2005 mitigated most of the issues that were observed in TempDB. Over time, the story changes though.



TempDB Journey

🟥 TempDB

🟦 Manageability

🟪 Data storage and access

SQL Server 2012 brings data warehousing capabilities into the database engine with the first generation of columnstore indexes. Even though you're most likely not adding columnstore indexes to your temp tables, that still means more metadata tables.

SQL Server 2014 marks another milestone release for SQL Server:

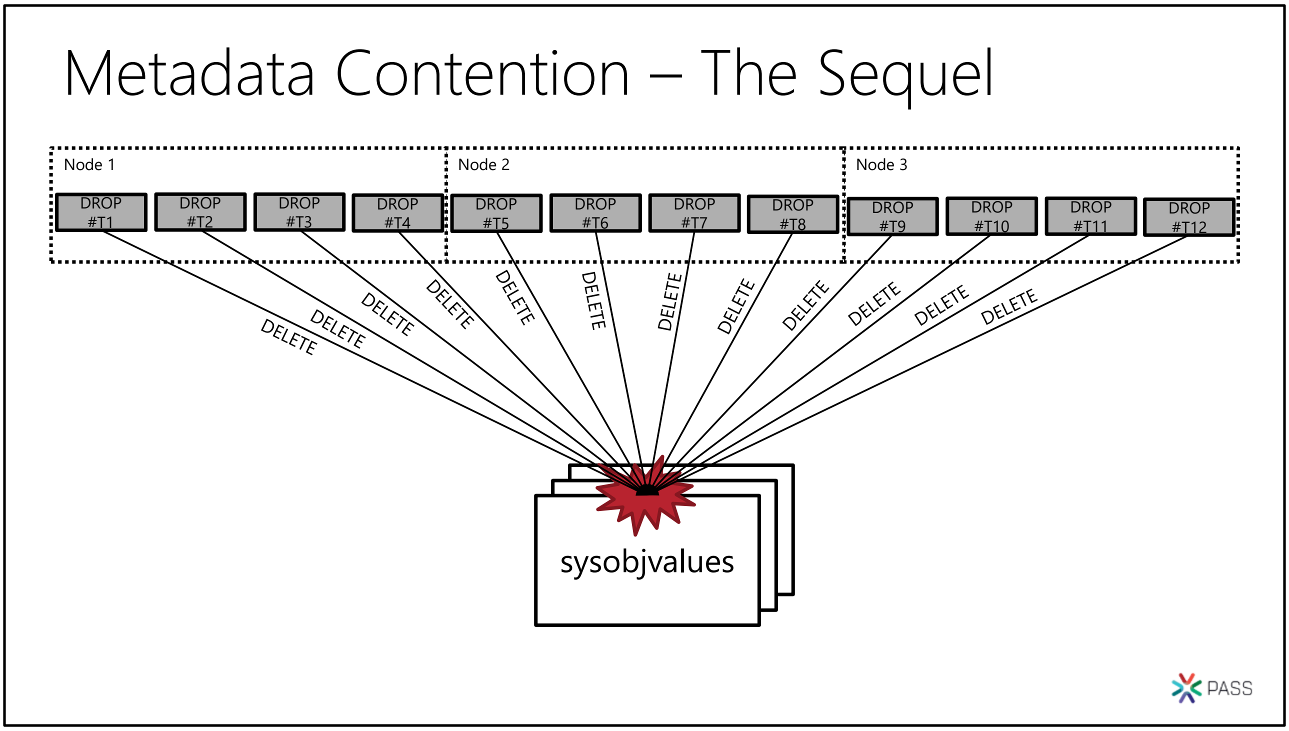
1. In-memory OLTP provides a new way of both storing and retrieving OLTP data with extremely low-latency, even in high-concurrency environments
2. Columnstore indexes are extended to be both clustered and non-clustered as well as updateable, making them a more viable option for modern data warehouse workloads on SQL Server

SQL Server 2016 provides major performance and scalability enhancements across the board. See the "It just runs faster" blog series for more details (https://blogs.msdn.microsoft.com/bobsql/tag/it-just-runs-faster/)

1. Automatic soft NUMA improves memory management and scalability on servers with a large number of cores per NUMA node (> 8)

2. Recommended tempdb configuration becomes the default
    1. Trace flags 1117 and 1118 become the default behavior for tempdb. 1118 is also the default for all user databases, you can opt out using a database level option. 1117 is default only for tempdb, you can opt in on user databases using a database level option.
    2. Multiple tempdb files are created by default based on the server configuration (one file per logical processor up to a maximum of 8)
3. Setup gets smarter with options to enable instant file inititialization and configure tempdb as needed for your environment (multiple files, multiple logical drives etc.)
4. File-snapshot backups in Azure provide a fast and easy way to snap backups in Azure IaaS-hosted VMs

So what happens when you make everything go faster? The throughput increases, the workload changes and the bottleneck moves somewhere else. All these scalability improvements in the engine, along with bigger and faster hardware, have increased the concurrent workload and put more pressure on tempdb. The temp table caching introduced in SQL Server 2005 was no longer enough to overcome the metadata contention, we just shifted the contention from the CREATE to the DROP.



Metadata Contention – The Sequel

Node 1

DROP #T1

DROP #T2

DROP #T3

DROP #T4

Node 2

DROP #T5

DROP #T6

DROP #T7

DROP #T8

Node 3

DROP #T9

DROP #T10

DROP #T11

DROP #T12

DELETE

DELETE

DELETE

DELETE

DELETE

DELETE

DELETE

DELETE

DELETE

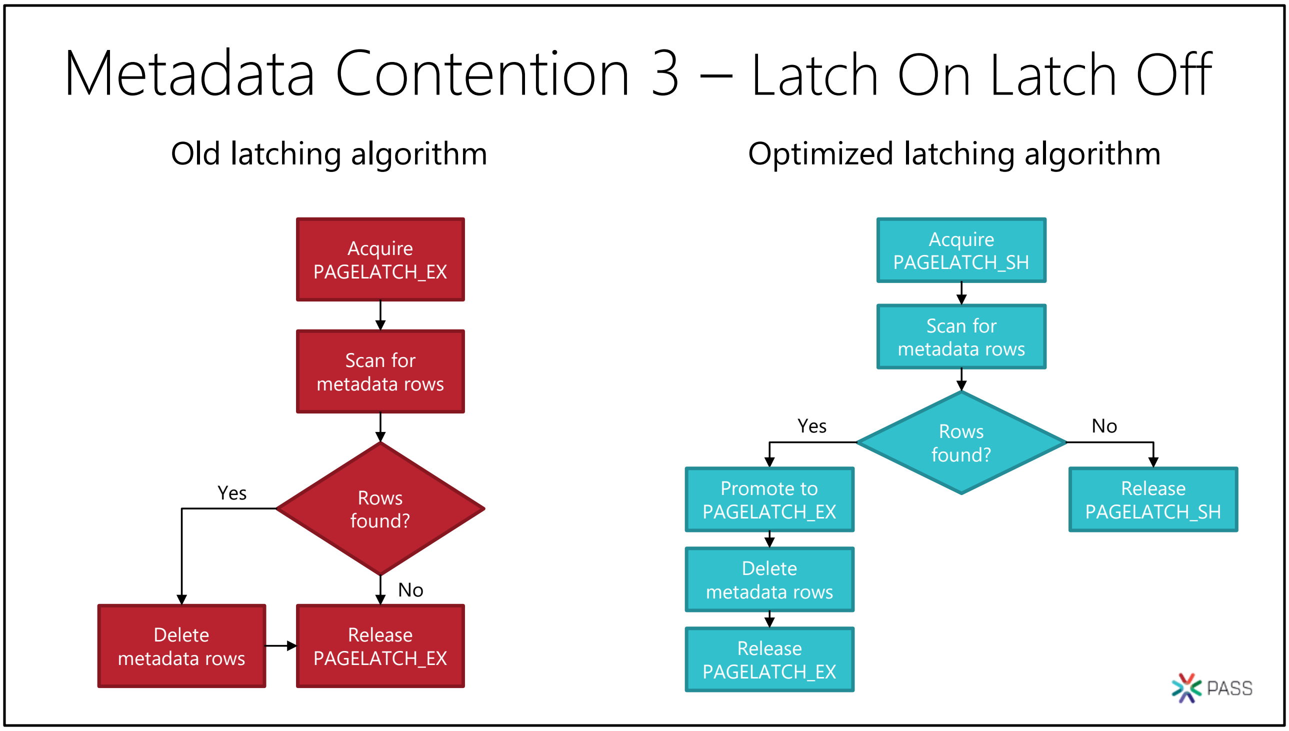
DELETE

DELETE

DELETE

sysobjvalues

Recall that in SQL Server 2000 we started seeing contention on the system metadata tables when lots of temp tables were created at once. Temp table caching in SQL Server 2005 resolved this issue by allowing us to reuse tables that didn't change between stored procedure executions. As long as the table was not altered after it was created, it would be eligible to be reused by another execution of the same stored procedure. This isn't really a special cache per se, we just don't bother to delete the metadata. If the table IS altered somewhere during the stored procedure (e.g. adding an index or a column), then it can't be reused and must be dropped when the stored procedure completes. Keep in mind that it isn't just one system table, there are several different tables that we need to delete metadata from in order to completely drop the table, and this was all being done synchronously at the end of the stored procedure execution. Also keep in mind we are adding all kinds of new features to SQL Server: columnstore indexes, temporal tables, stretch DB etc. All these new features require new metadata, so the number of tables we need to delete from is increasing, which makes the process take longer.



Metadata Contention 3 – Latch On Latch Off

Old latching algorithm

Acquire PAGELATCH_EX

Scan for metadata rows

Rows found?

Yes

No

Delete metadata rows

Release PAGELATCH_EX

Optimized latching algorithm

Acquire PAGELATCH_SH

Scan for metadata rows

Rows found?

Yes

No

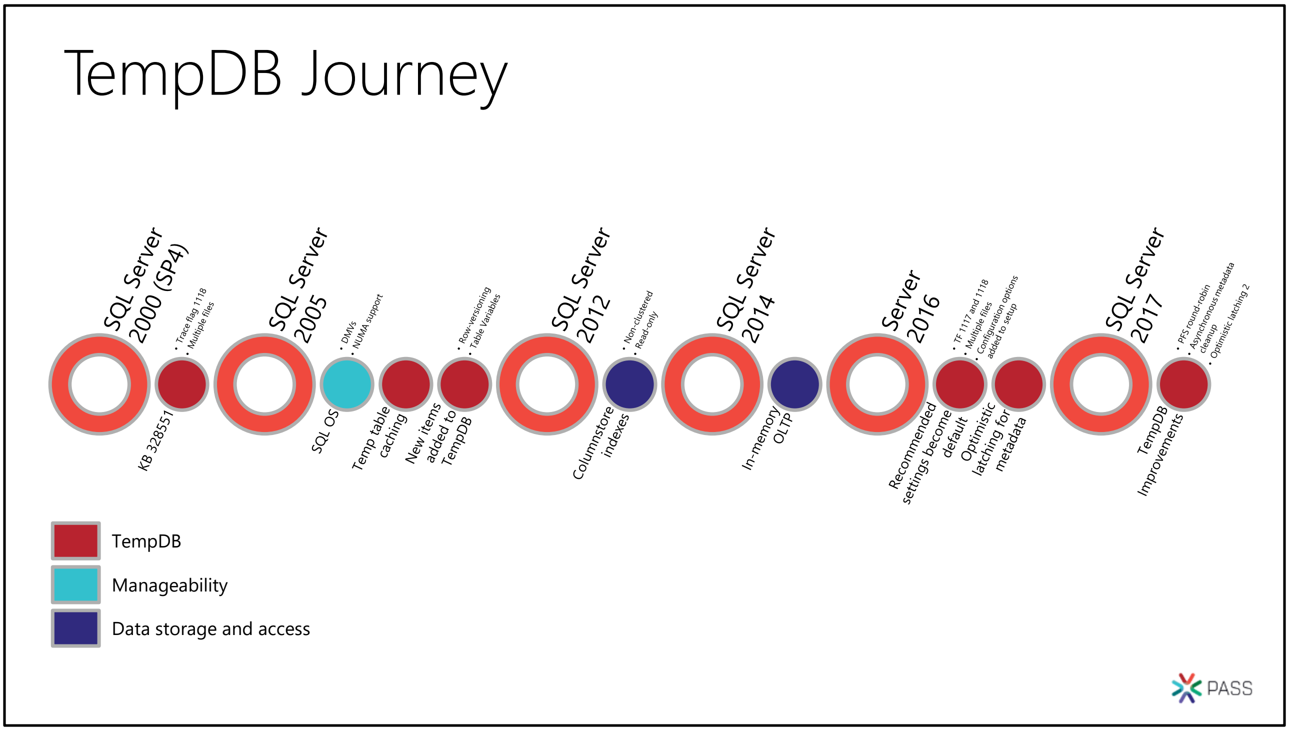
Promote to PAGELATCH_EX

Delete metadata rows

Release PAGELATCH_EX

Release PAGELATCH_SH

In response to the increase in metadata contention on deletes (cache invalidation), SQL Server 2016 RTM implemented a new “optimistic” latching mechanism for tempdb metadata when searching for metadata to delete. There are several metadata tables that have to be searched for rows to delete when we remove a table from the cache, but in tempdb many of them are going to be empty. Making the latching mechanism optimistic avoids getting exclusive latches altogether for these empty tables and makes the process a lot less prone to contention.



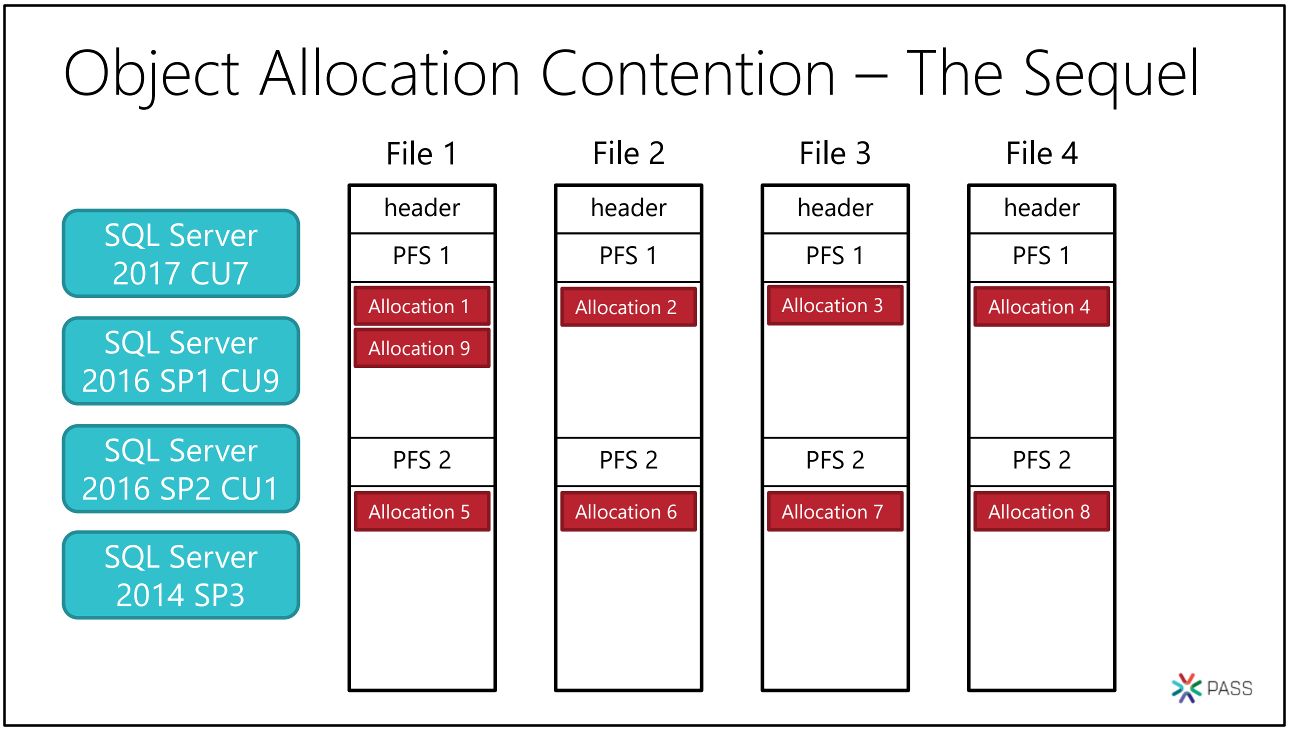
TempDB Journey

🟥 TempDB

🟦 Manageability

🟪 Data storage and access

With all of these scalability enhancements and new features and workloads on SQL Server, new and more intense bottlenecks emerged on tempdb which triggered further improvements here to both object allocation with the introduction of a new intra-file round robin algorithm for PFS pages and metadata concurrency enhancements.



Object Allocation Contention – The Sequel

SQL Server 2017 CU7

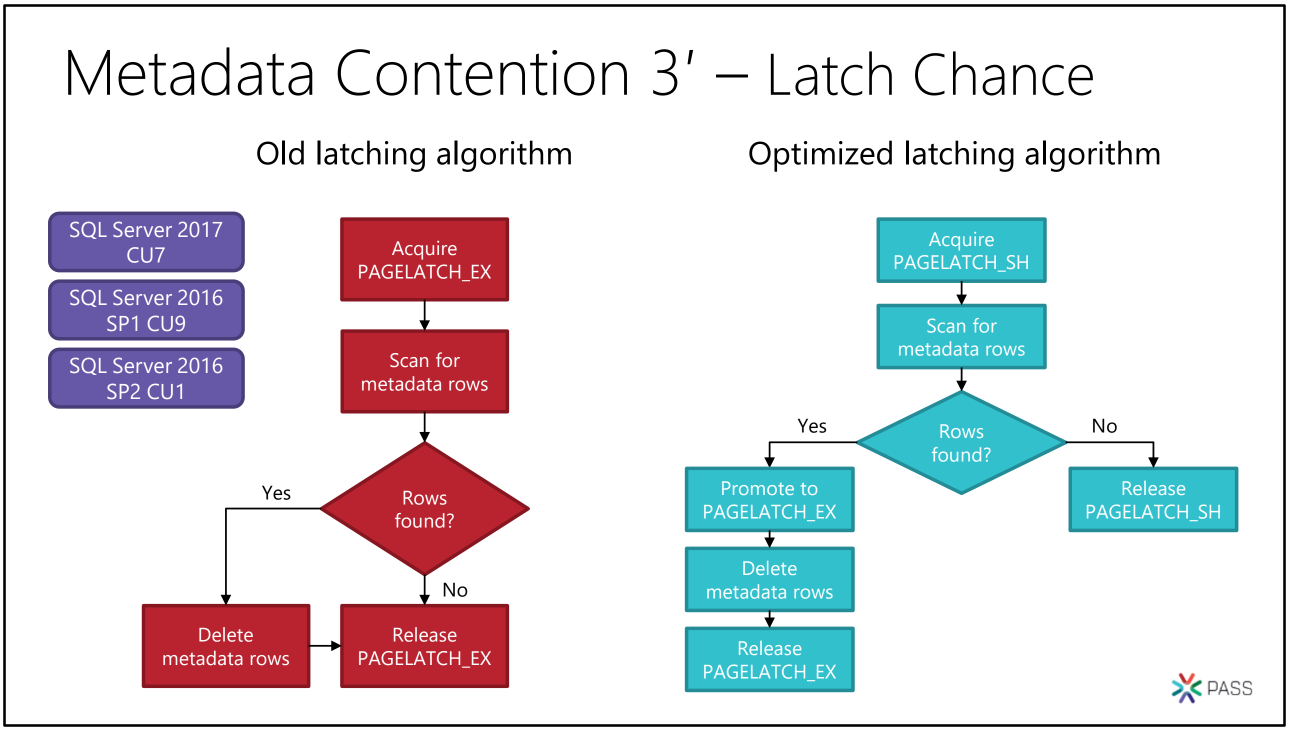
SQL Server 2016 SP1 CU9

SQL Server 2016 SP2 CU1

SQL Server 2014 SP3

File 1	File 2	File 3	File 4
header	header	header	header
PFS 1	PFS 1	PFS 1	PFS 1
Allocation 1	Allocation 2	Allocation 3	Allocation 4
Allocation 9			
PFS 2	PFS 2	PFS 2	PFS 2
Allocation 5	Allocation 6	Allocation 7	Allocation 8

With very high concurrency workloads, it's still possible to get PFS contention, even with multiple files. You could just keep adding files as the contention increases, but that can be cumbersome to manage. In SQL Server 2017 CU7, we introduced a fix that implements round-robining across all the PFS within a file as well as across the files. This allows us to further reduce PFS contention without increasing the number of files. This fix was also backported to SQL Server 2016 SP1 CU9, SQL Server 2016 SP2 CU1 and SQL Server 2014 SP3.



Metadata Contention 3′ – Latch Chance

Old latching algorithm

SQL Server 2017 CU7

SQL Server 2016 SP1 CU9

SQL Server 2016 SP2 CU1

Acquire PAGELATCH_EX

Scan for metadata rows

Rows found?

Yes

No

Delete metadata rows

Release PAGELATCH_EX

Optimized latching algorithm

Acquire PAGELATCH_SH

Scan for metadata rows

Rows found?

Yes

No

Promote to PAGELATCH_EX

Delete metadata rows

Release PAGELATCH_EX

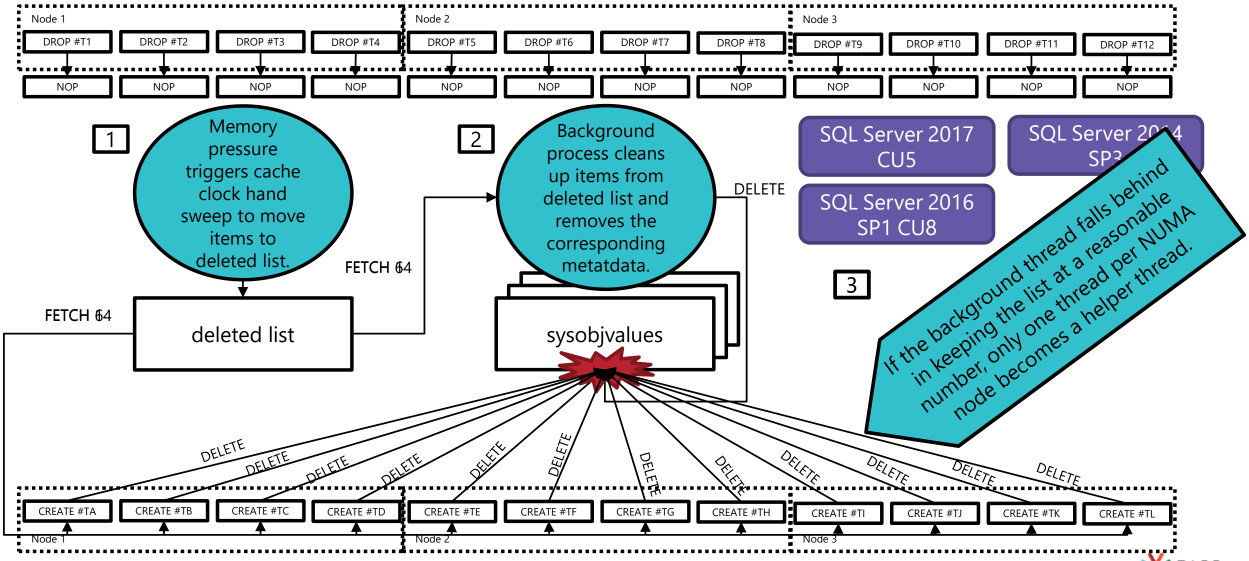
Release PAGELATCH_SH

PASS

Turns out with SQL Server 2016 RTM, some metadata tables were missed with the optimized latching mechanism. In SQL Server 2017 CU7, SQL Server 2016 SP1 CU9 and SQL Server 2016 SP2 CU1, we completed the job by applying the new mechanism to the remaining metadata tables.

Performance issues occur in the form of PAGELATCH_EX and PAGELATCH_SH waits in TempDB when you use SQL Server 2016
https://support.microsoft.com/kb/4131193

Metadata Contention 4 – Going Async



If the background thread falls behind in keeping the list at a reasonable number, only one thread per NUMA node becomes a helper thread.

By SQL Server 2016, we had outgrown the SQL Server 2005-era temp table caching. That caching was introduced to overcome CREATE TABLE contention, now we started seeing DROP TABLE contention when something gets removed from the cache.

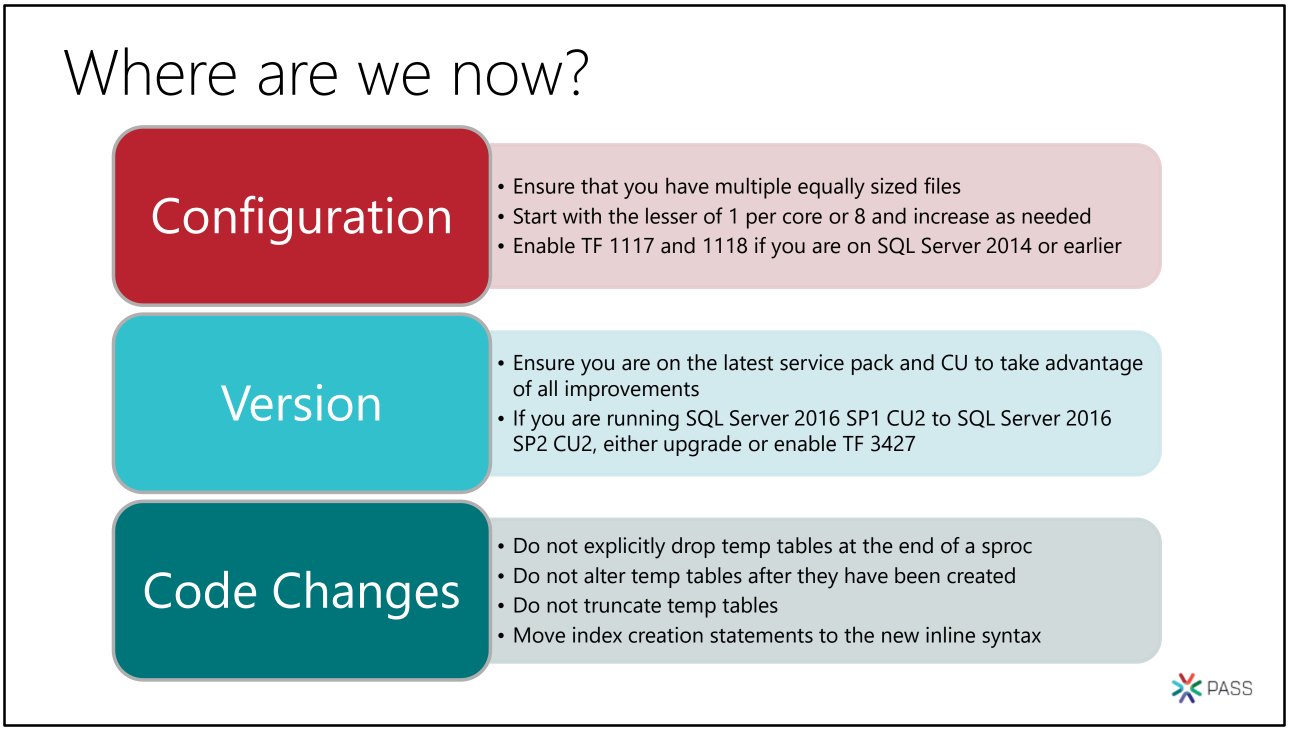
The first step to overcoming this contention is to make the deletes from cache asynchronous. When a table needs to be removed from the cache, it is not immediately dropped, it is moved to a deleted list and dropped later by another thread.

With this behavior change, the only thing that triggers a delete from the cache is memory pressure, which is handled by a cache clock hand sweep, the same as other caches in SQL Server. The clock hand sweep moves items to a deleted list and a background thread cleans up the metadata based on this list. When the deleted list reaches a certain size, any thread which needs to add something to the cache becomes a helper thread and has to remove an object from the deleted list first (FYI, this mechanism has been in place since the temp table cache was introduced in SQL Server 2005). With very intense tempdb workloads under memory pressure, the helper threads can get into the same contention situation as we had earlier on deletes. In order to get around this problem, we reduced the helper threads to one per NUMA node, and the threads will remove 64 objects at a time rather than 1.

FIX: Heavy tempdb contention occurs in SQL Server 2016 or 2017
https://support.microsoft.com/kb/4058174

TempDB

Today and tomorrow



Where are we now?

Configuration

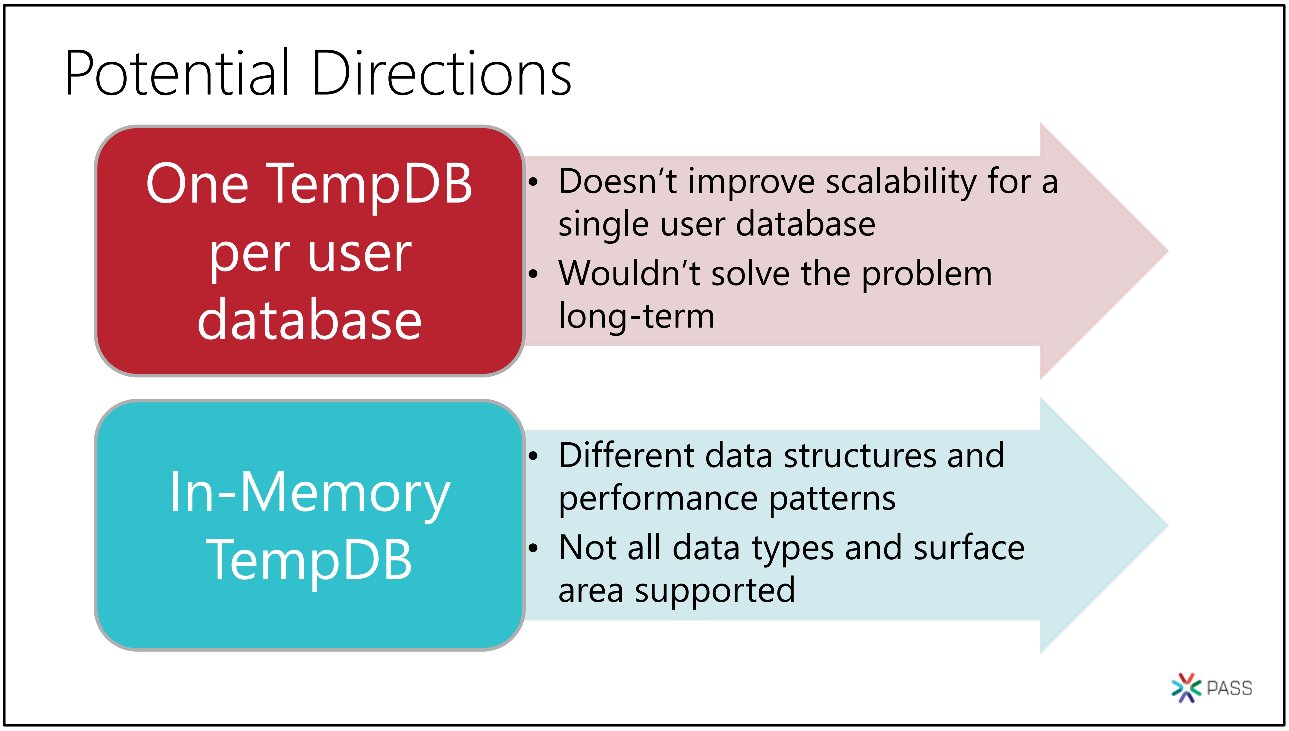
- Ensure that you have multiple equally sized files
- Start with the lesser of 1 per core or 8 and increase as needed
- Enable TF 1117 and 1118 if you are on SQL Server 2014 or earlier

Version

- Ensure you are on the latest service pack and CU to take advantage of all improvements
- If you are running SQL Server 2016 SP1 CU2 to SQL Server 2016 SP2 CU2, either upgrade or enable TF 3427

Code Changes

- Do not explicitly drop temp tables at the end of a sproc
- Do not alter temp tables after they have been created
- Do not truncate temp tables
- Move index creation statements to the new inline syntax



Potential Directions

One TempDB per user database

- Doesn't improve scalability for a single user database
- Wouldn't solve the problem long-term

In-Memory TempDB

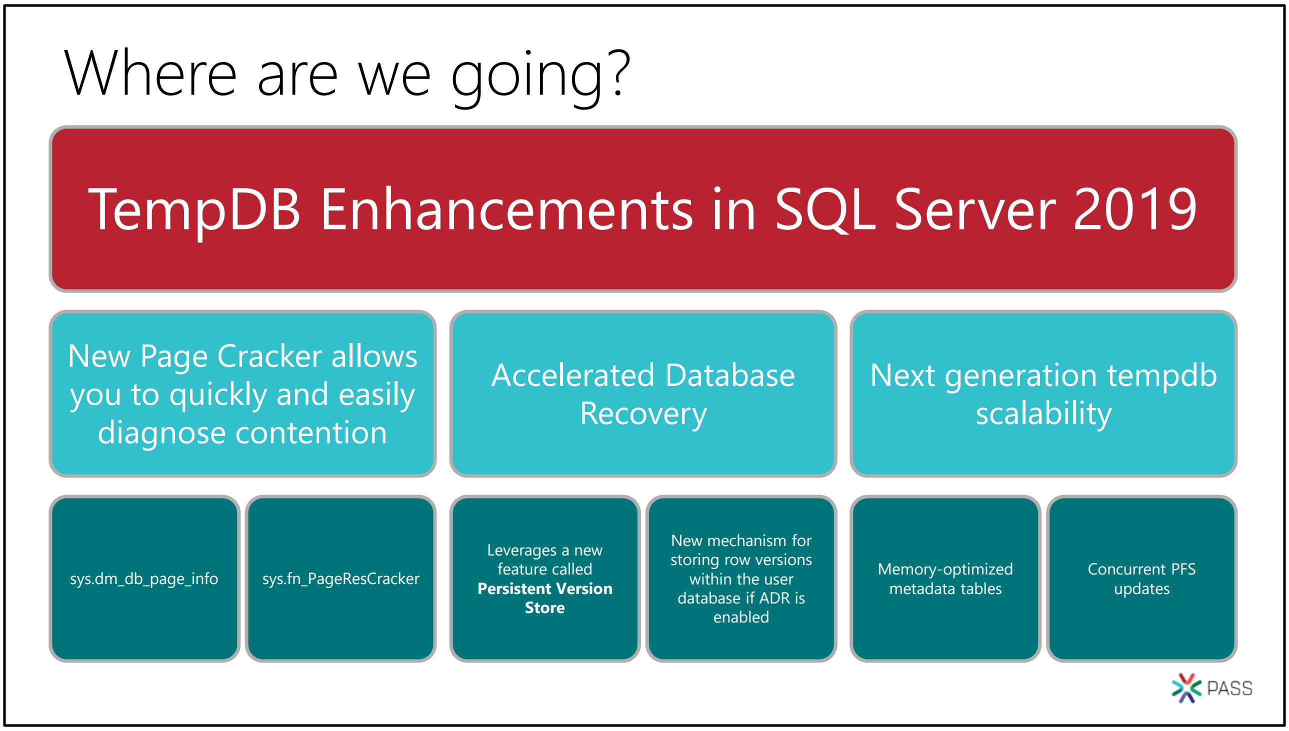
- Different data structures and performance patterns
- Not all data types and surface area supported

In spite of all the improvements we have made, we still have scalability issues with tempdb today.

There have been many suggestions for changing tempdb over the years. One of the most common is to have multiple tempdbs, specifically one per user database. The problem with this solution is that many of the scalability issues we have seen occur on a server with a single user database. Even if this would help some customers, it wouldn't buy much time before the scalability becomes a roadblock again.

Leveraging In-Memory OLTP is another option that has been suggested, but memory optimized tables are not the same as disk-based tables. Indexing and access patterns are different, plus not all data types and T-SQL constructs are supported. There may be a case for implementing memory-optimized tables in tempdb, but it wouldn't necessarily replace the disk-based tempdb.

What we really want is a solution that solves the current problems for the long-term, something that we know will scale through the foreseeable future.



Where are we going?

TempDB Enhancements in SQL Server 2019

New Page Cracker allows you to quickly and easily diagnose contention

- sys.dm_db_page_info
- sys.fn_PageResCracker

Accelerated Database Recovery

- Leverages a new feature called **Persistent Version Store**
- New mechanism for storing row versions within the user database if ADR is enabled

Next generation tempdb scalability

- Memory-optimized metadata tables
- Concurrent PFS updates

PASS

We continue our tempdb journey in SQL Server 2019 by merging it with the data storage innovation of In-Memory OLTP. Leveraging memory-optimized tables for tempdb metadata will release a major friction point in tempdb and set us up for the next generation of tempdb scalability. Memory-optimized tables are ideal for tempdb metadata because the tables are relatively small, concurrency is very high, and tempdb is re-created at startup so we can leverage non-durable tables.

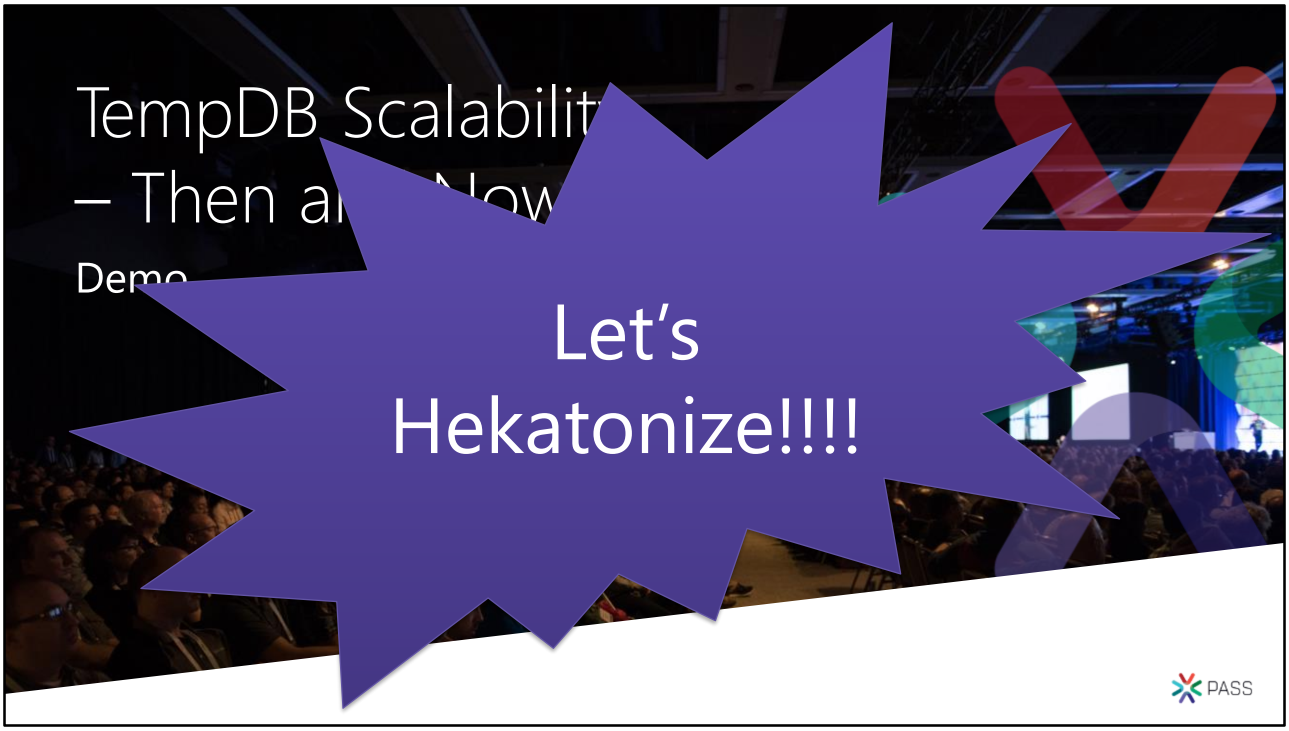
Another improvement which addresses object allocation contention is concurrent PFS updates, which is on by default in tempdb and configurable in user databases in SQL Server 2019. If you recall our latching discussion, an exclusive latch is required to update a PFS page (really any page in SQL Server) in order to maintain consistency. Because PFS pages are updated in a different way than user pages, using what's called an interlocked operation, we are actually able to perform this operation under a shared latch. With an interlocked operation, the locking is handled at a lower level (at the CPU), so the exclusive latch is not needed. This change allows multiple sessions to latch the PFS page at the same time, and since the CPU is handling the serialization, we can maintain consistency as a much lower cost than using the latch construct.

We also introduce a new DMF that helps troubleshoot page-related waits.

In addition to improvements in scalability, SQL Server 2019 will introduce a new mechanism for row-versioning called Persistent Version Store. This will move the row-versioning from tempdb to the user database and enables a new set of features called Accelerated Database Recovery (ADR). ADR is a new database scoped SQL Server Engine feature that greatly improves database availability (especially in the presence of long running transactions) by completely redesigning the SQL Server recovery process.

ADR provides the following benefits:

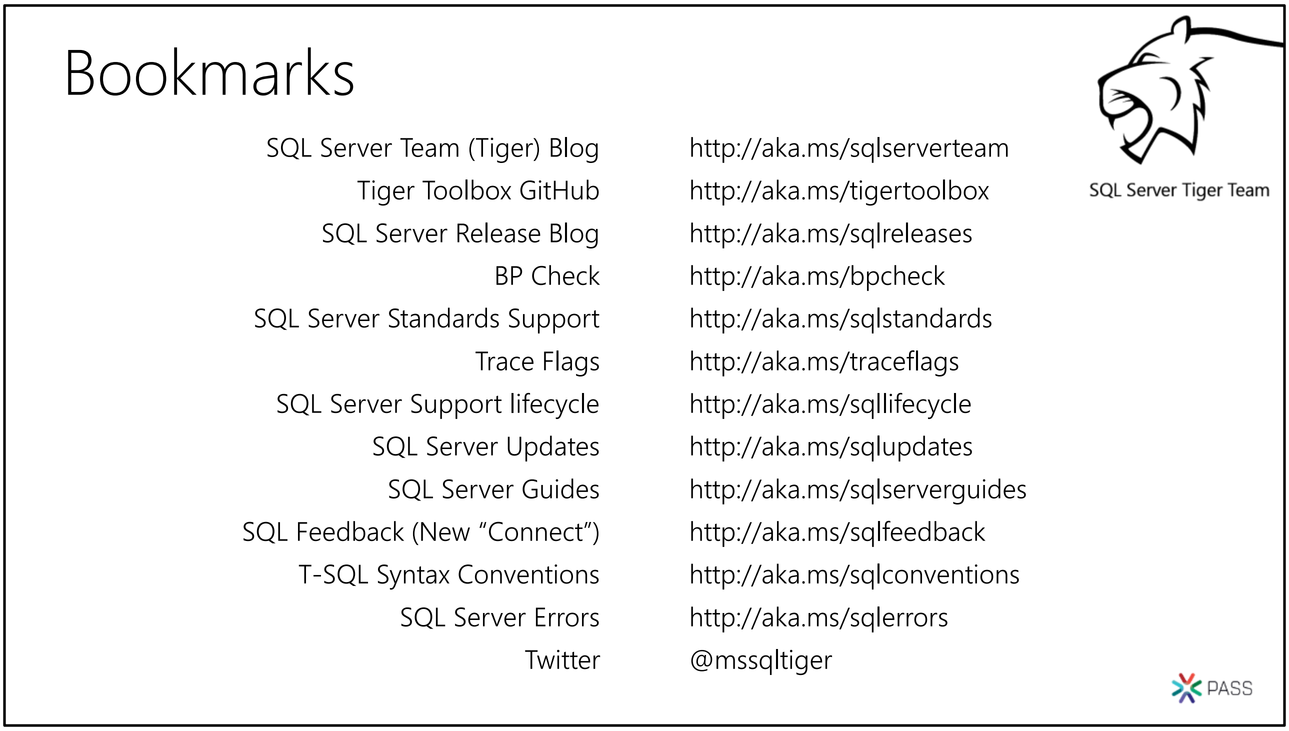
- Fast and consistent Database Recovery
    - Redo phase starts from last checkpoint rather than oldest active transaction
    - Special Log Stream used for non-versioned operations
    - Database fully available after Redo phase completes
- Instantaneous Transaction rollback
  Uses logical revert to last committed row version
- Aggressive Log Truncation



TempDB Scalabilit
– Then a Now

Demo

Let’s
Hekatonize!!!!



Bookmarks

SQL Server Tiger Team

SQL Server Team (Tiger) Blog	http://aka.ms/sqlserverteam
Tiger Toolbox GitHub	http://aka.ms/tigertoolbox
SQL Server Release Blog	http://aka.ms/sqlreleases
BP Check	http://aka.ms/bpcheck
SQL Server Standards Support	http://aka.ms/sqlstandards
Trace Flags	http://aka.ms/traceflags
SQL Server Support lifecycle	http://aka.ms/sqllifecycle
SQL Server Updates	http://aka.ms/sqlupdates
SQL Server Guides	http://aka.ms/sqlserverguides
SQL Feedback (New “Connect”)	http://aka.ms/sqlfeedback
T-SQL Syntax Conventions	http://aka.ms/sqlconventions
SQL Server Errors	http://aka.ms/sqlerrors
Twitter	@mssqltiger

PASS

PASS SUMMIT2018

Thank You

Learn more from Pam Lahoud

@SQLGoddess

pamela@microsoft.com