



Database and Log Structure

Module 3

Learning Units covered in this Module

- Lesson 1: SQL Server Disk I/O
- Lesson 2: SQL Server Log File Structure
- Lesson 3: SQL Server TempDB File Structure

Lesson 1: SQL Server Disk I/O

SQL Server disk I/O patterns:

Data Files

- One .mdf file per database
- May have one or more .ndf file
- Random reads and writes
 - Read activity during backups; other activity varies depending on query activity and buffer pool size
 - Write activity during checkpoints, recovery, and lazy writes

Log Files

- One* .ldf file per database
- Sequential reads and writes
- Write activity during the log buffer flush operations
- Read activity during checkpoints, backups, and recovery
- Features such as database mirroring and replication will increase read and write activity

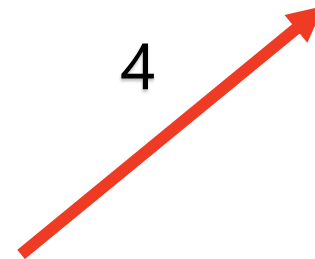
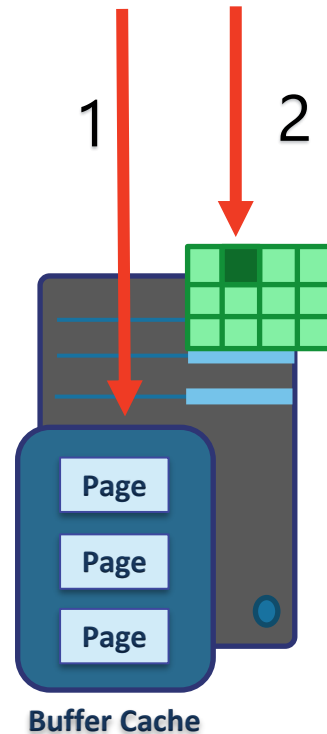
SQL Server Disk I/O (Write-Ahead Logging)

```
UPDATE Accounting.BankAccounts  
SET Balance -= 200  
WHERE AcctID = 1
```

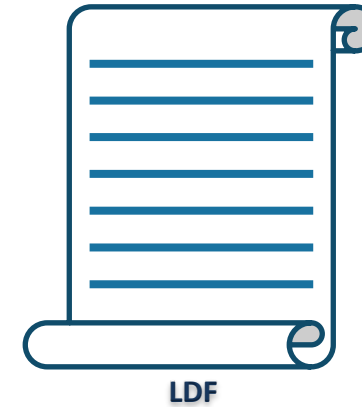
1. Data modification is sent to buffer cache in memory.

2. Modification is recorded in the log cache.

3. Data pages are located or read into the buffer cache and then modified.



4. Log cache record is flushed to the transaction log



5. At checkpoint, dirty data pages are written to the database file.



Log Buffer Flushing

SQL Server will flush the log buffer to the log file

- SQL Server gets a commit request of a transaction that changes data.
- The log buffer fills up. (Max size 60kb.)
- SQL Server needs to harden dirty data pages (checkpoints)
- Manually request a log buffer flush using the `sys.sp_flush_log` procedure

Log buffer flushing results in a WRITELOG wait type.

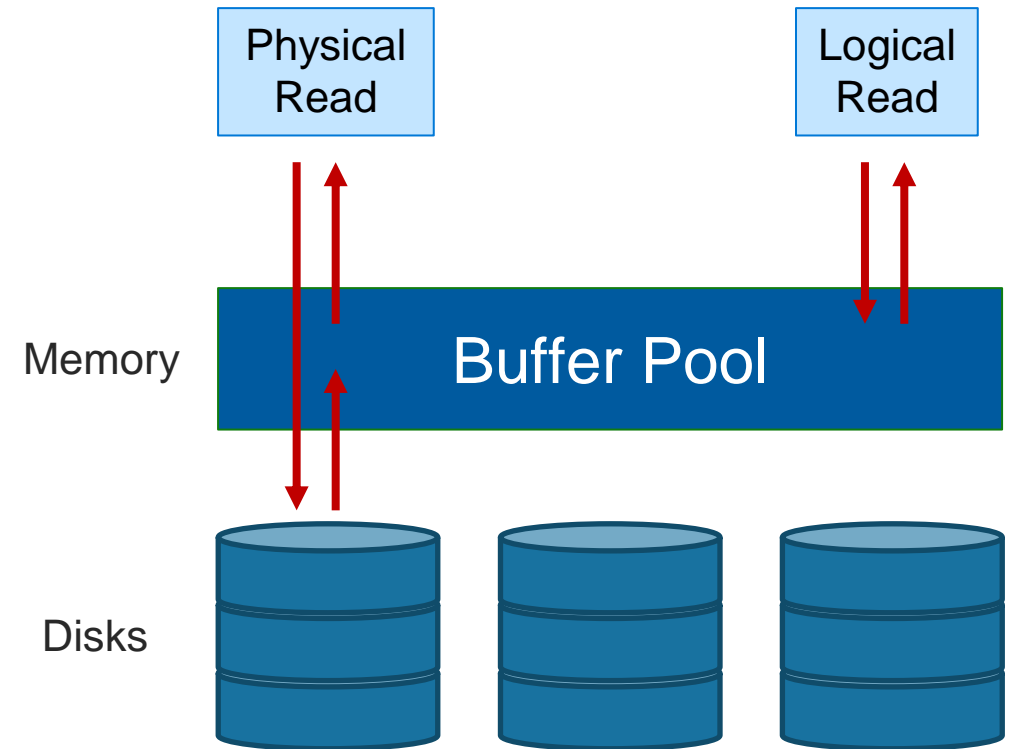
SQL Server Buffer Pool

Stores 8 kilobytes (KB) pages of data to avoid repeated disk I/O.

- Pages held in the buffer until the space is needed by something else.

Lazy Writer searches for eligible buffers.

- If the buffer is dirty, an asynchronous write (lazy write) is posted so that the buffer can later be freed.
- If the buffer is not dirty, it is freed.



SET STATISTICS IO

```
SET STATISTICS IO ON
GO
SET STATISTICS TIME ON
SELECT SOH.SalesOrderID, SOH.CustomerID,
OrderQty, UnitPrice, P.Name
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN Production.Product AS P
ON P.ProductID = SOD.ProductID
SET STATISTICS IO, TIME OFF
```

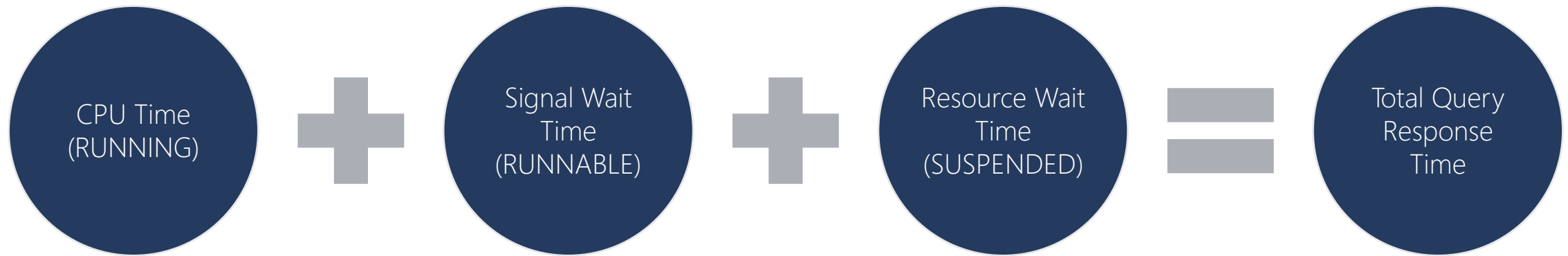
Used to identify physical reads and logical reads for a query

```
(121317 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server r
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server
Table 'SalesOrderDetail'. Scan count 1, logical reads 428, physical reads 0, pag
Table 'Product'. Scan count 1, logical reads 15, physical reads 0, page server r
Table 'SalesOrderHeader'. Scan count 1, logical reads 57, physical reads 0, page

SQL Server Execution Times:
    CPU time = 94 ms,  elapsed time = 1653 ms.
```


Total Query Response Time

- The full cycle between the several task states, for how many times it needs to cycle, is what we experience as the total query response time.



Checkpoints

Flushes dirty pages from the buffer pool to the disk. Frequency of checkpoints varies based on the database activity and recovery interval.

Automatic (default) – Database engine issues checkpoints automatically based on the server level “recovery interval” configuration option

Indirect (new in SQL Server 2012) – Database engine issues checkpoints automatically based on the database level TARGET_RECOVERY_TIME

```
ALTER DATABASE [AdventureWorksPTO] SET TARGET_RECOVERY_TIME = 60 SECONDS
```

Manual – Issued in the current database for your connection when you execute the T-SQL CHECKPOINT command

Internal – Issued by various server operations

Lesson 2: SQL Server Log File Structure

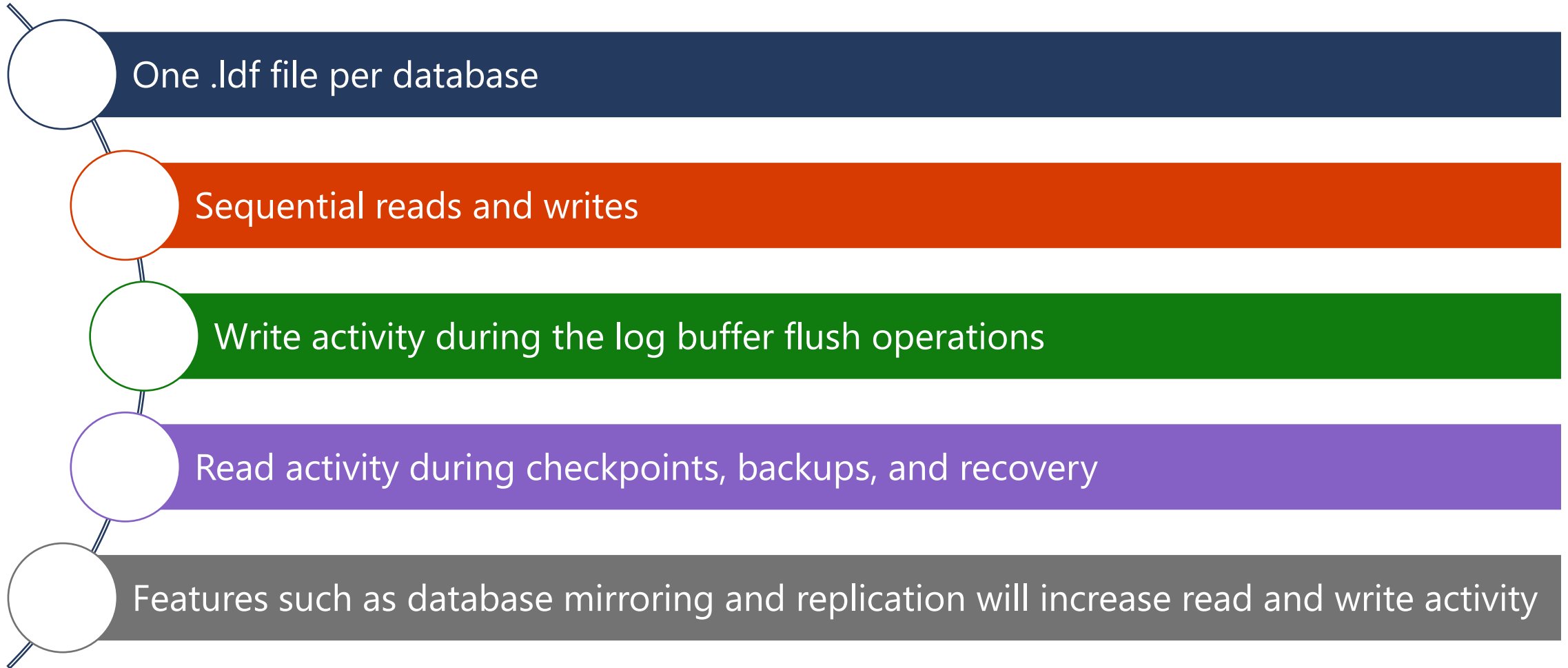
Objectives

After completing this learning, you will be able to:

- Understand the log file structure.
- Explain SQL Server logging process.



SQL Server disk I/O patterns: transaction log



Transaction Log

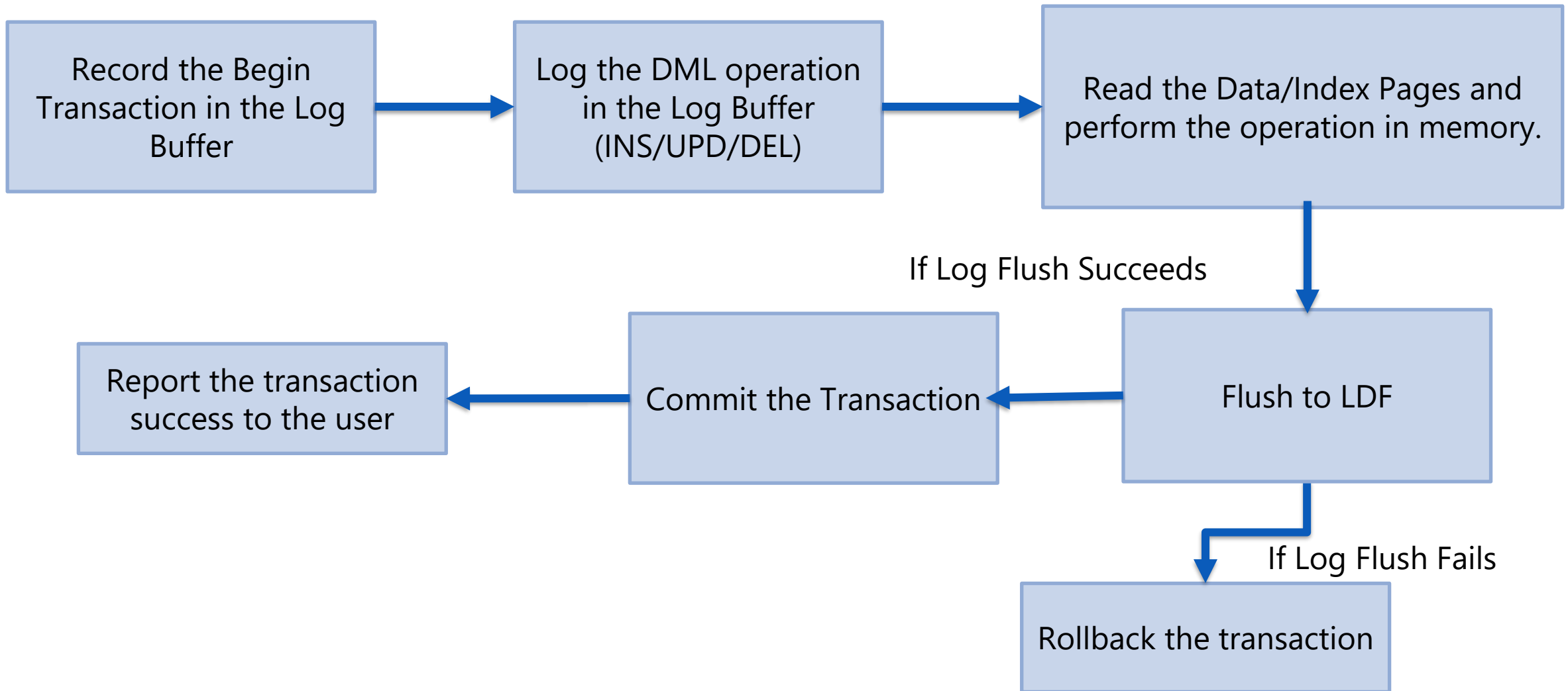
Sequence of log records contained in one or more physical files

Records identified by increasing Logical Sequence Numbers (LSNs)

Recorded operations:

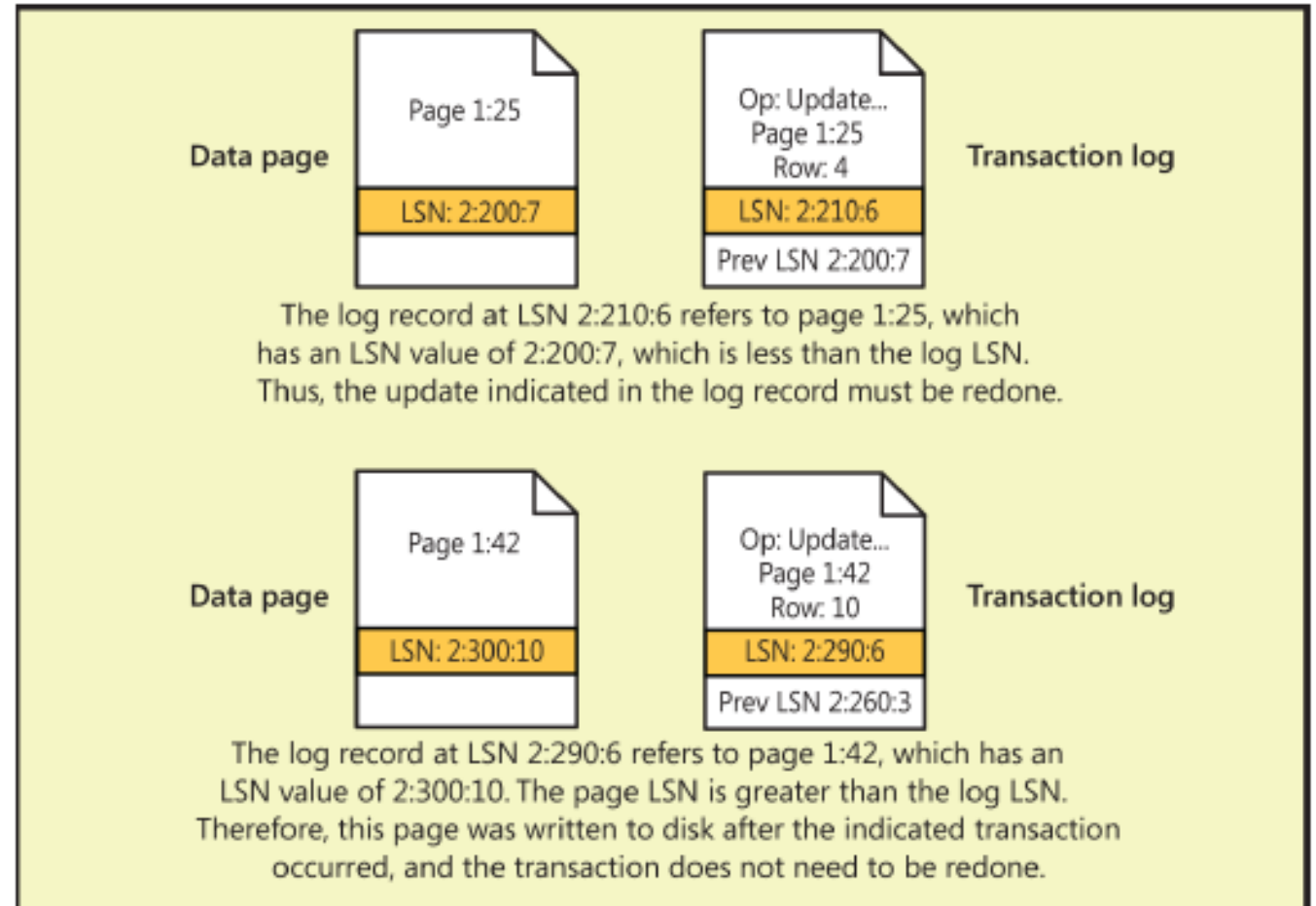
- Start and end of each transaction
- All database modifications
- Extent and page allocation or deallocation
- Creating and dropping objects

Microsoft SQL Server Transaction Logging



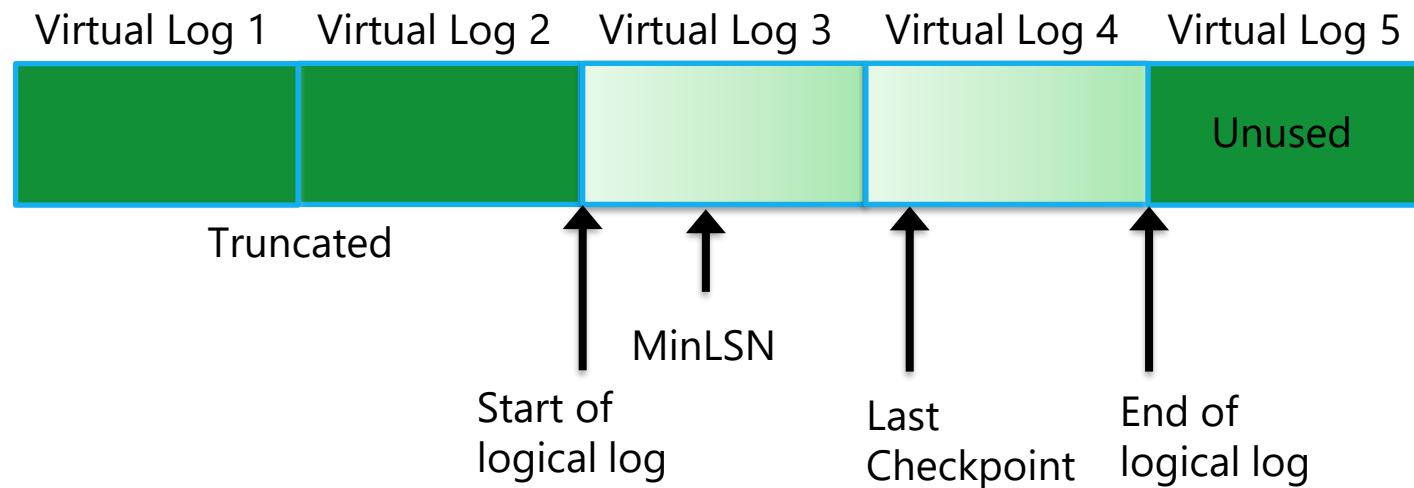
Page LSN and recovery

- Last LSN in the page header
- Log Record has two LSNs
 - Previous LSN from the data page
 - Current log record LSN
- All used for recovery



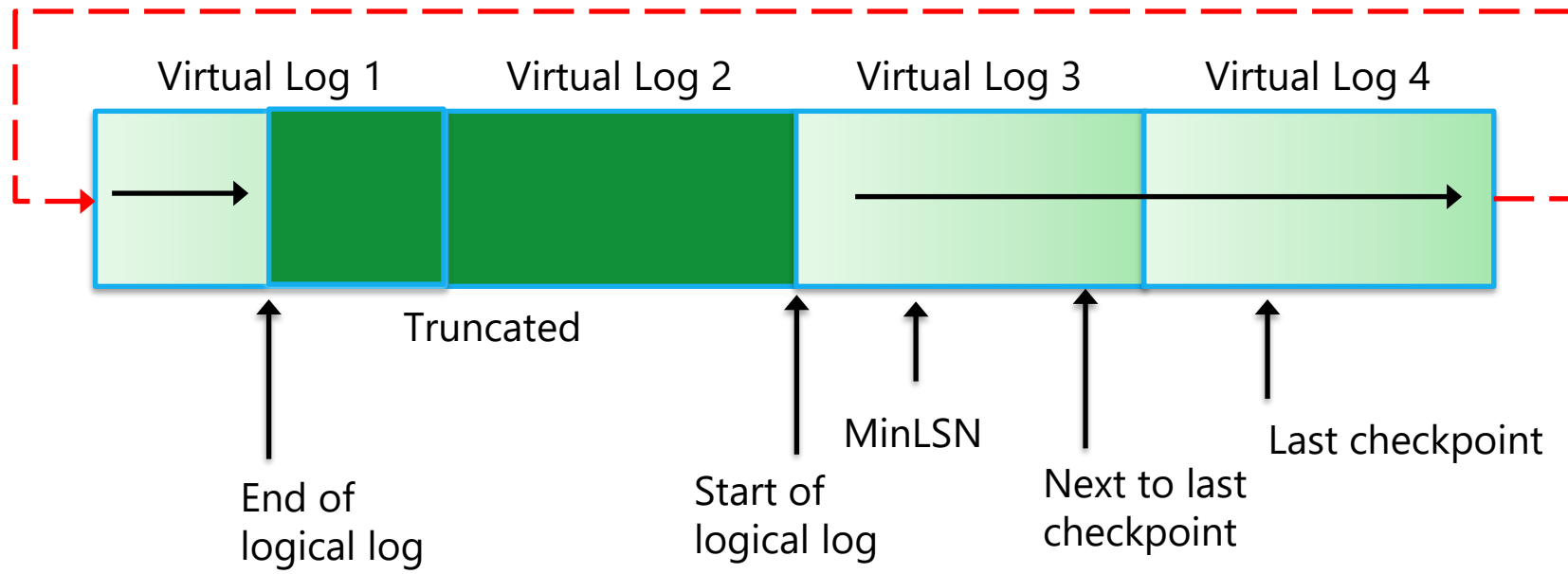
Physical Log File Structure

- The physical file is divided into virtual log files (VLFs).
- SQL Server works to keep the number of VLFs small.
- VLF size and number is dynamic and cannot be configured or set.



Physical Log File Structure (Continued)

- Log file is circular.
- When the end of logical log meets the beginning, the physical log file is extended.



Addressing VLF Counts

VLF count T-LOG file creation

Size \leq 64 MB \rightarrow 4 VLFs

Between 64 MB and 1 GB \rightarrow 8 VLFs

Size $>$ 1GB \rightarrow 16 VLFs

VLF count at T-LOG file growth

Growth \leq 64 MB \rightarrow 4 VLFs

Between 64 MB and 1GB \rightarrow 8 VLFs

Growth $>$ 1GB \rightarrow 16 VLFs

(SQL 2014 and higher)

If growth is less than 1/8th of
current size \rightarrow add 1 VLF

SQL Server 2014 VLF Growth Improvement

- Is the growth size less than 1/8 the size of the current log size?
 - Yes: create 1 new VLF equal to the growth size
 - No: use the previous formula
- Example of a 256 MB log file with an autogrowth setting of 5 MB
 - 2012 and earlier: 10 auto-grows of 5MB would add 4 VLFs x 10 auto-grows
 - 2014 and later: 10 auto-grows of 5MB each would only create 10 VLFs

Grow Iterations + Log size	Up to SQL Server 2012	From SQL Server 2014
0 (256 MB)	8	8
10 (306 MB)	48	18
20 (356 MB)	88	28
80 (656 MB)	328	88
250 (1.2 GB)	1008	258
3020 (15 GB)	12091	3028

Addressing VLF Counts

To avoid VLF fragmentation

- Pre-size log files to avoid unplanned file growth
- Set autogrowth to an appropriate fixed size

To view VLFs

- DBCC LOGININFO (<database_id>)
- sys.dm_db_log_info (SQL Server 2016 SP2 and higher)
- Active VLFs have a status of 2
- SQL Server Error Log (SQL 2012 and higher)

Demonstration

Examining log use



Lesson 3: SQL Server TempDB File Structure

Objectives

After completing this learning, you will be able to:

- Understand the TempDB database structure.
- Optimize TempDB database.
- Use the newest improvements in SQL Server for TempDB database.



TempDB database

System database

- Available to all users with the same structure as user databases.
- Operations are minimally logged.
- Re-created every time SQL Server is started.

Workload

- Used for temporary (non-durable) storage.
- Object and data frequently being created and destroyed.
- Very high concurrency.
- Backup and restore operations are not allowed on TempDB.

What is stored in TempDB?

Temporary user objects

- Global or local temporary tables and indexes
- Temporary stored procedures
- Table variables
- Tables returned in table-valued functions, or cursors

Internal objects

- Worktables to store intermediate results for spools, cursors, sorts, and temporary 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

- Common version store
- Online-index-build version store

Physical properties of TempDB

File	Logical name	Physical name	Initial size	File growth
Primary data	tempdev	tempdb.mdf	8 MB	Autogrow by 64 MB until the disk is full
Secondary data files*	temp#	tempdb_mssql_#.ndf	8 MB	Autogrow by 64 MB until the disk is full
Log	templog	templog.ldf	8 MB	Autogrow by 64 megabytes to a maximum of 2 terabytes

If the number of logical processors is less than or equal to eight (8), use the same number of data files as logical processors.

If the number of logical processors is greater than eight (8), use eight data files.

If contention continues, increase the number of data files by multiples of four (4) up to the number of logical processors

Alternatively, make changes to the workload or code.

Optimizing TempDB performance

Consider instant file initialization

Pre-allocate space for all TempDB files

Divide TempDB into multiple data files of equal size

Put the TempDB database on a fast I/O subsystem

Use disk striping if there are many directly attached disks.

Put the TempDB database on separate disks from user databases

Performance improvements in TempDB

Starting with SQL Server 2016 (13.x)

Trace Flags 1117 and 1118 behavior enabled by default for TempDB

Temporary tables and table variables are cached.

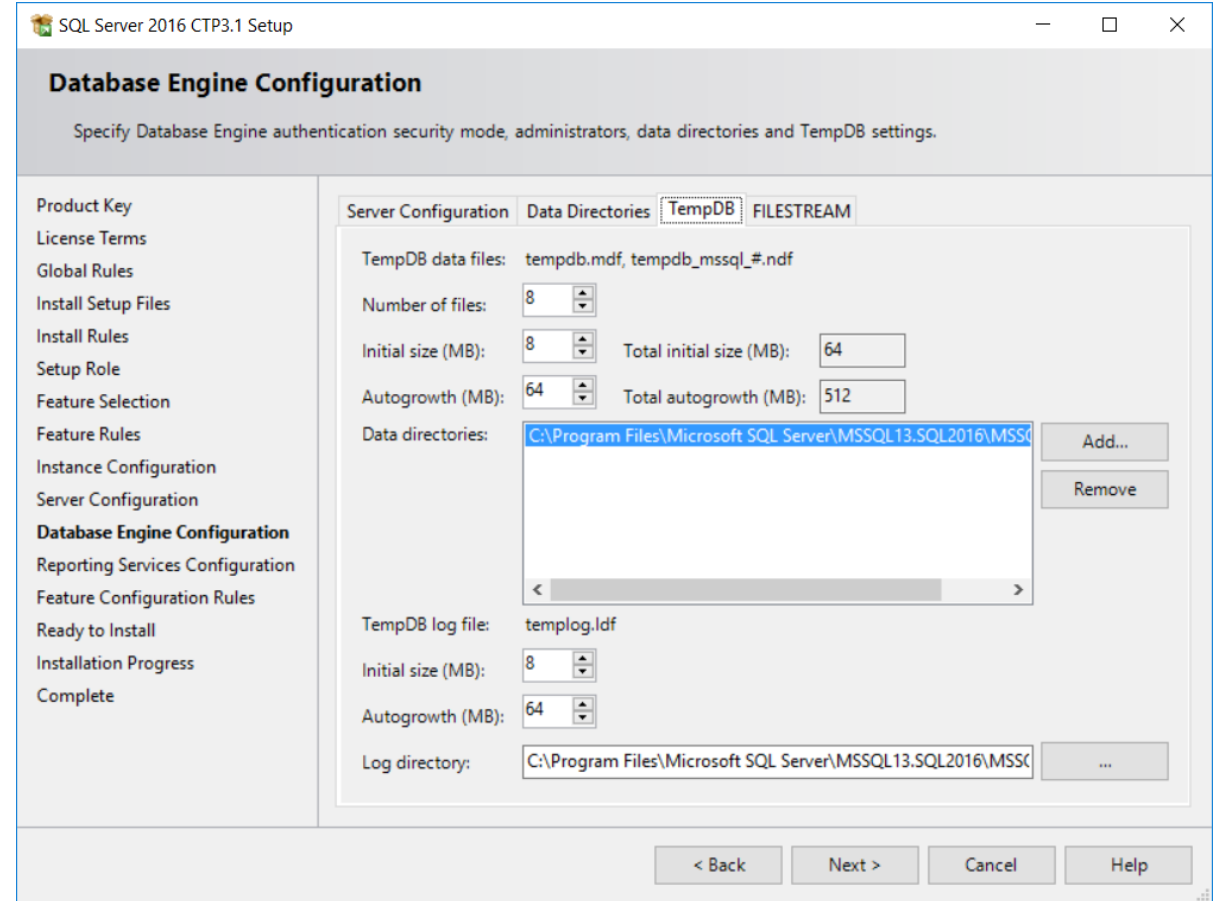
Allocation page latching protocol improved.

Logging overhead for TempDB is reduced.

Performance improvements in TempDB

Starting with SQL Server 2016 (13.x)

- Setup adds multiple TempDB data files during instance installation.
- By default, setup adds as many TempDB data files as the logical processor count or eight, whichever is lower.



Performance improvements in TempDB

Starting with SQL Server 2019 (15.x)

Default

- Temp table cache improvements
- Concurrent PFS updates

Opt-in

- Memory-Optimized TempDB Metadata

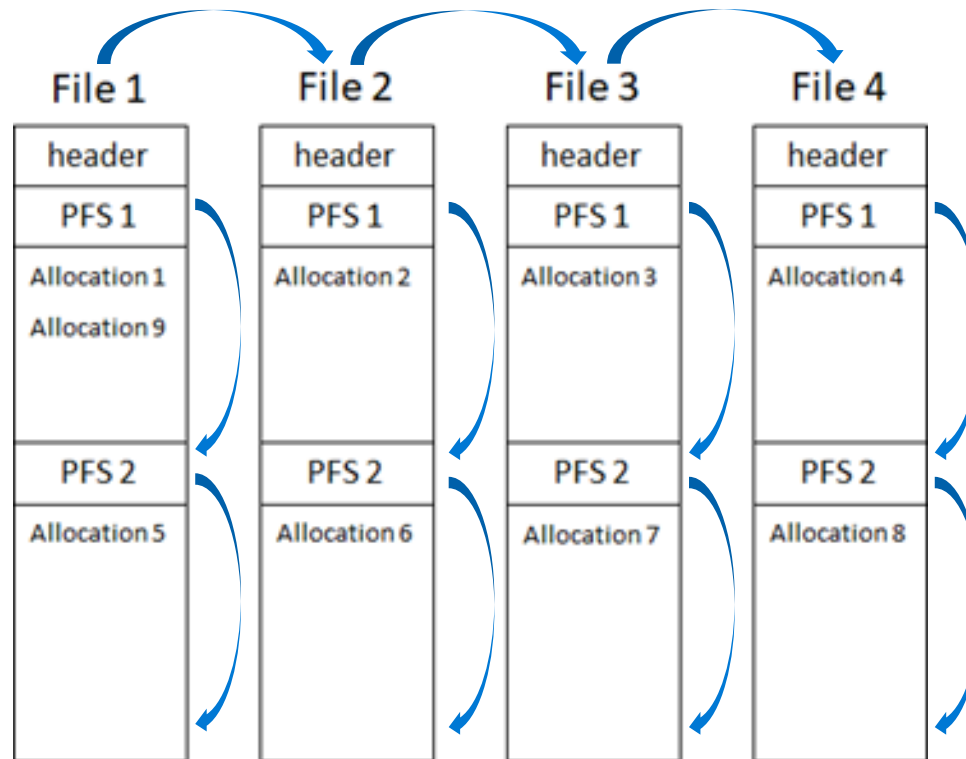
[TEMPDB Files, Trace Flags, and Updates by Pam Lahoud](#)

[How \(and When\) To: Memory Optimized TempDB](#)

Concurrent PFS updates

Starting with SQL Server 2019 (15.x)

New algorithm to round robin between files, and between PFS pages within the files.



With this change, not only will increasing the number of files help with PFS contention also increasing the size of the files.

