



# Database and Log Structure

Module 3

## Learning Units covered in this Module

- Lesson 1: SQL Server Disk I/O
- Lesson 2: SQL Server Page Structure
- Lesson 3: SQL Server Data File Structure
- Lesson 4: SQL Server Log File Structure
- Lesson 5: 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

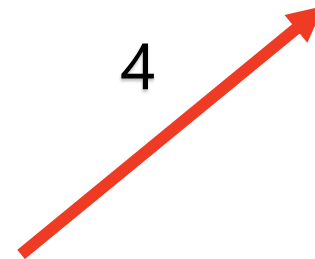
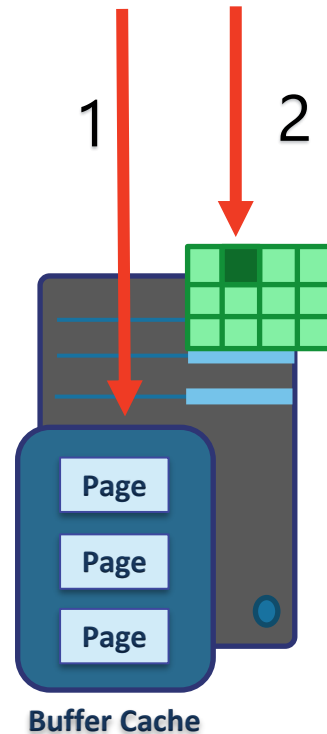
# SQL Server Disk I/O (Write-Ahead Logging)

```
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.

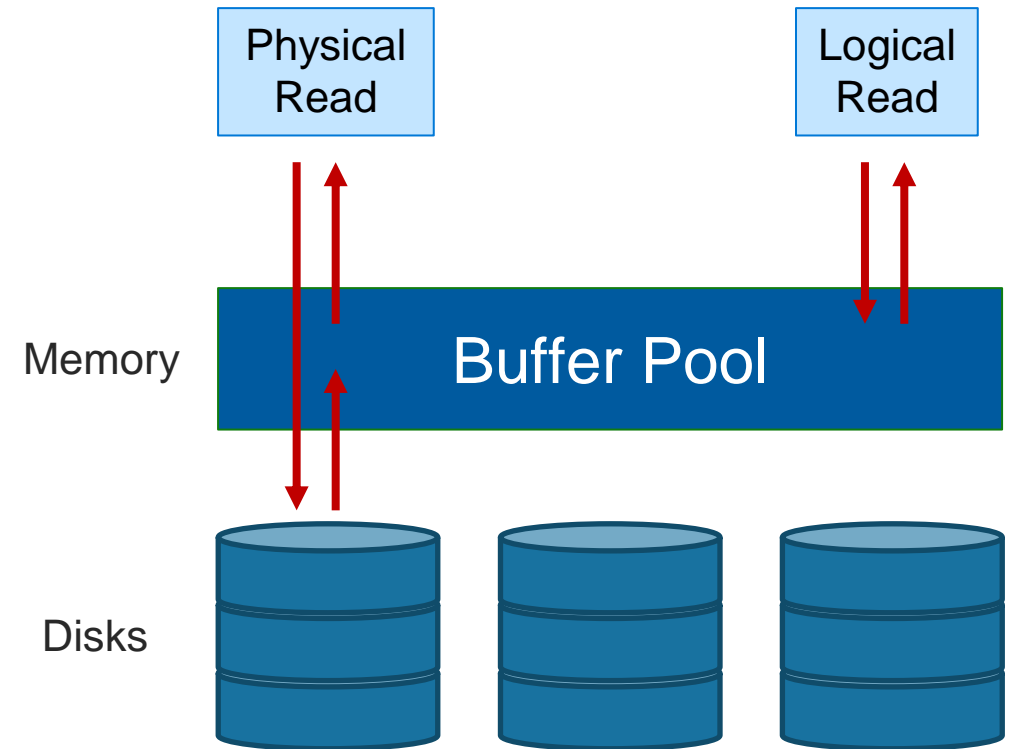
# SQL Server Buffer Pool

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

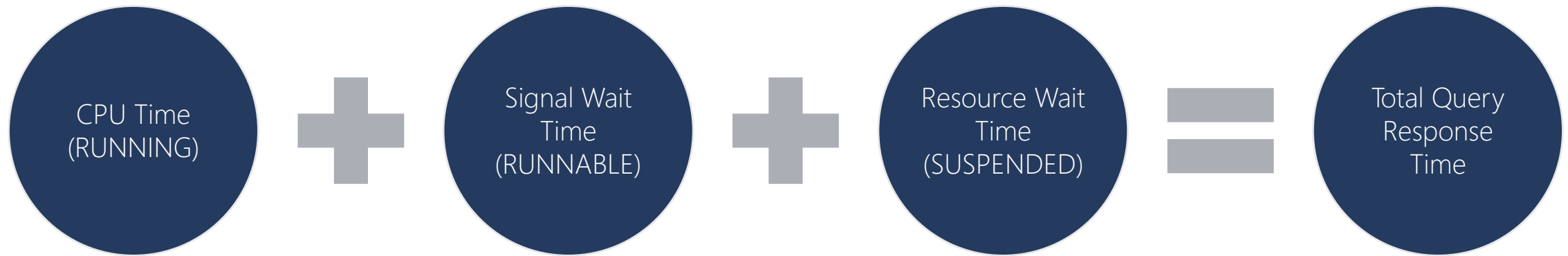
```
(121317 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server r
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server
Table 'SalesOrderDetail'. Scan count 1, logical reads 428, physical reads 0, pag
Table 'Product'. Scan count 1, logical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

# Objectives

After completing this learning, you will be able to:

- Understand the architecture of pages and extents.

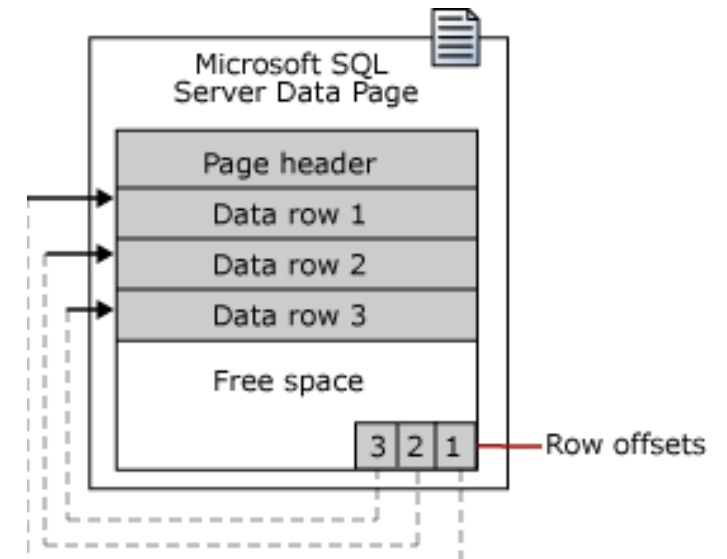


# Pages and Extents architecture

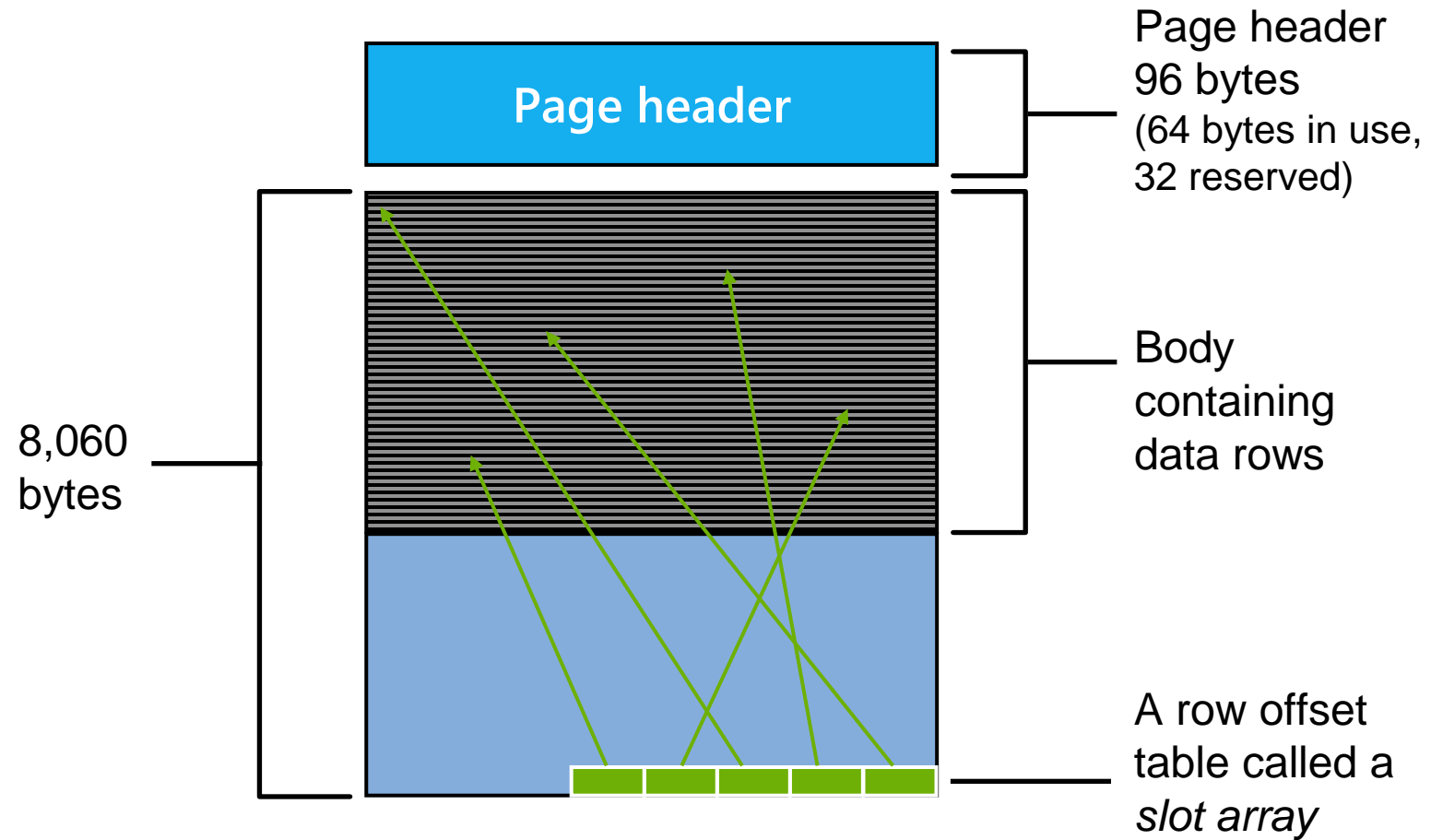
A data page is the fundamental unit of data storage in SQL Server.

- The disk space allocated to a data file (.mdf or .ndf) is logically divided into pages.
- Each page is 8 KB in size
- Pages are numbered contiguously from 0 to n.
- Disk I/O operations are performed at the page level.

Extents are a collection of eight physically contiguous pages (64KB) and are used to efficiently manage the pages.



# Data Page structure



# Page types

Types	Page Type (ID)	Description
Data and Index	Data (1)	Data rows with all data, except text, ntext, image, nvarchar(max), varchar(max), varbinary(max), and xml data, when <b>text in row</b> is set to <b>ON</b>
	Index (2)	Index Entries
	Text/Image (3 or 4)	Large Object Data Type, variable length columns when the data row exceeds 8 kilobytes (KB)
Allocation	GAM, SGAM (8 and 9)	Extent Allocation information
	PFS (11)	Information about page allocation and free space available on pages
	IAM (10)	Information about extents used by a table or index per allocation unit
Restore	Bulk Changed Map (17)	Information about extents modified by bulk operations since the last BACKUP LOG statement per allocation unit
	Differential Changed Map (16)	Information about extents that have changed since the last BACKUP DATABASE statement per allocation unit
Metadata	Boot (13)	Information about the database; Each database has only one Boot page
	File Header (15)	Information about the file. It is the first page (page 0) in every file

# DBCC IND

```
USE AdventureWorks2012
DBCC TRACEON(3604) -- Print to results pane
DBCC IND(0, 'HumanResources.Employee', -1)
-- Parameter 1: Is the DatabaseName, 0 is current database
-- Parameter 2: The table name
-- Parameter 3: Index ID, -1 Shows all indexes, -2 shows only IAM Pages
```

10 %															
		Results Messages													
	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType	IndexLevel	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	1	874	NULL	NULL	1237579447	1	1	72057594045136896	In-row data	10	NULL	0	0	0	0
2	1	875	1	874	1237579447	1	1	72057594045136896	In-row data	2	1	0	0	0	0
3	1	1048	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1049	0	0
4	1	1049	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1050	1	1048
5	1	1050	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1051	1	1049
6	1	1051	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1052	1	1050
7	1	1052	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1053	1	1051
8	1	1053	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1054	1	1052
9	1	1054	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	0	0	1	1053
10	1	9287	NULL	NULL	1237579447	2	1	72057594050510848	In-row data	10	NULL	0	0	0	0
11	1	9286	1	9287	1237579447	2	1	72057594050510848	In-row data	2	0	0	0	0	0
12	1	9289	NULL	NULL	1237579447	3	1	72057594050576384	In-row data	10	NULL	0	0	0	0

Query executed successfully. STUDENTSERVER (12.0 RTM) STUDENTSERVER\Student ... AdventureWorks2012 00:00:0



# DBCC PAGE

```
DBCC TRACEON(3604) -- Print to results pane
DBCC PAGE (0,1,0,3)
-- Parameter 1: Is the DatabaseName, 0 is current database
-- Parameter 2: The File ID
-- Parameter 3: The Page ID
-- Parameter 4: The print option, 3 is verbose
```

.00 %

Messages

PAGE HEADER:

Page @0x000000027757A000

m_pageId = (1:0)	m_headerVersion = 1	m_type = 15
m_typeFlagBits = 0x0	m_level = 0	m_flagBits = 0x208
m_objId (AllocUnitId.idObj) = 99	m_indexId (AllocUnitId.idInd) = 0	Metadata: AllocUnitId = 6488064
Metadata: PartitionId = 0	Metadata: IndexId = 0	Metadata: ObjectId = 99
m_prevPage = (0:0)	m_nextPage = (0:0)	pminlen = 0
m_slotCnt = 1	m_freeCnt = 6989	m_freeData = 7831
m_reservedCnt = 0	m_lsn = (181:50952:34)	m_xactReserved = 0
m_xdesId = (0:0)	m_ghostRecCnt = 0	m_tornBits = -820886669
DB Frag ID = 1		

# New DMF

## sys.dm\_db\_page\_info

- Returns information about a page in a database.
- The function returns one row that contains the **header** information from the page, including the object\_id, index\_id, and partition\_id.
- This function replaces the need to use DBCC PAGE in most cases.
- Output sample:

database_id	file_id	page_id	page_header_version	page_type	page_type_desc	page_flag_bits	page_flag_bits_desc	page_lsn	page_level	object_id	index_id	partition_id
5	1	11712	1	1	DATA_PAGE	0x200	HAS_CHECKSUM	00000025:000004cc:0133	0	1029578706	1	72057594047889408

sys.dm\_db\_page\_info is currently supported only in SQL Server 2019 (15.x) and later.

# Demonstration

Examining Data Pages



# Lesson 3: SQL Server Data File Structure

# Objectives

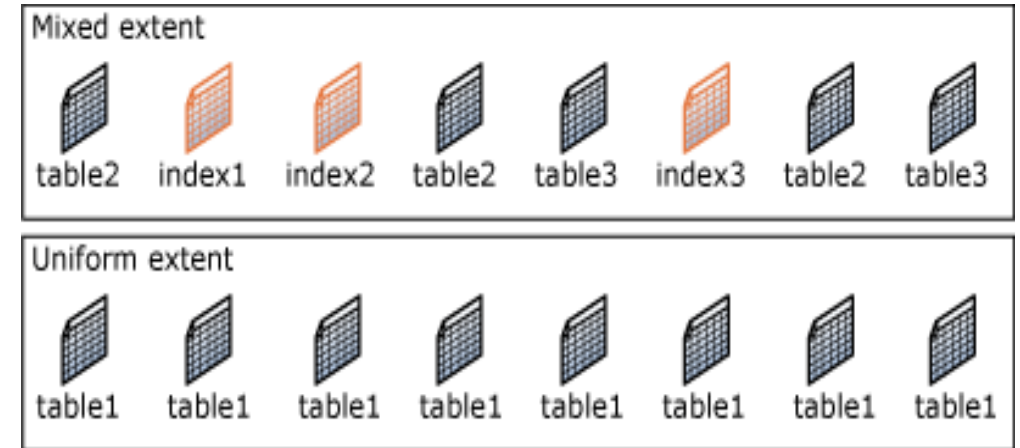
After completing this learning, you will be able to:

- Understand the data file structure.
- Describe SQL Server allocation objects, and object allocation process.



# SQL Server object allocation

- Data files are structured in extents.
- Each extent (64 KB) has eight pages of 8 KB each.
- Two types of extents:
  - Mixed extents
    - Up to 8 distinct objects can reside in the same Mixed Extent.
    - Controlled by SGAM (Shared Global Allocation Map) Object.
  - Uniform extents
    - All pages belong to the same object.
    - Controlled by GAM (Global Allocation Map) object.



# The Role of Allocation Pages in Object Allocation

PFS and IAM are used to determine when an object needs a new extent allocated

GAMs and SGAMs are used to allocate the extent

# Allocation objects

## Page Free Space (PFS)

- Tracks free space on a page uses 1 Byte/Page
- Covers 64 MB worth of pages

## Global Allocation Map (GAM)

- Tracks Uniform extents that have been allocated uses 1 Bit/extent
- Covers 64,000 extents (4 GB worth of data)

## Shared Global Allocation Map (SGAM)

- Tracks Mixed extents that have all 8 pages allocated uses 1 Bit/extent
- Covers 64,000 extents (4 GB worth of data)

## Index Allocation Map (IAM)

- Tracks extents that are allocated to an allocation unit
- Covers 4 GB worth of data
- One IAM chain per table, per index, per partition, per allocation unit type





# Database File page layout

MyDBFile1.mdf

0	Header
1	PFS
2	GAM
3	SGAM
...	IAM/Data /Index/etc.
8088	PFS
...	IAM/Data /Index/others
511232	GAM
511233	SGAM

Page Address is  
DBID:FileID:Page#  
5:1:1

Page Free Space  
8,087 Pages

Global Allocation Map

Shared Global  
Allocation Map  
Mixed Extents  
63,904 Extents

MyDBFile2.ndf

0	Header
1	PFS
2	GAM
3	SGAM
...	IAM/Data /Index/oth ers
8088	PFS
...	IAM/Data /Index/oth ers
511232	GAM
511233	SGAM

# SQL Server object allocation

Up to, and including, SQL Server 2014 (12.x)

- On object creation 2 pages from a mixed extents are allocated
- After 8 data pages, new allocation comes from Uniform Extent

Starting with SQL Server 2016 (13.x)

- Default for allocations in the database is Uniform extents for Data pages
- Decreases contention on SGAM objects

# SQL Server object allocation (Trace Flag 1118)

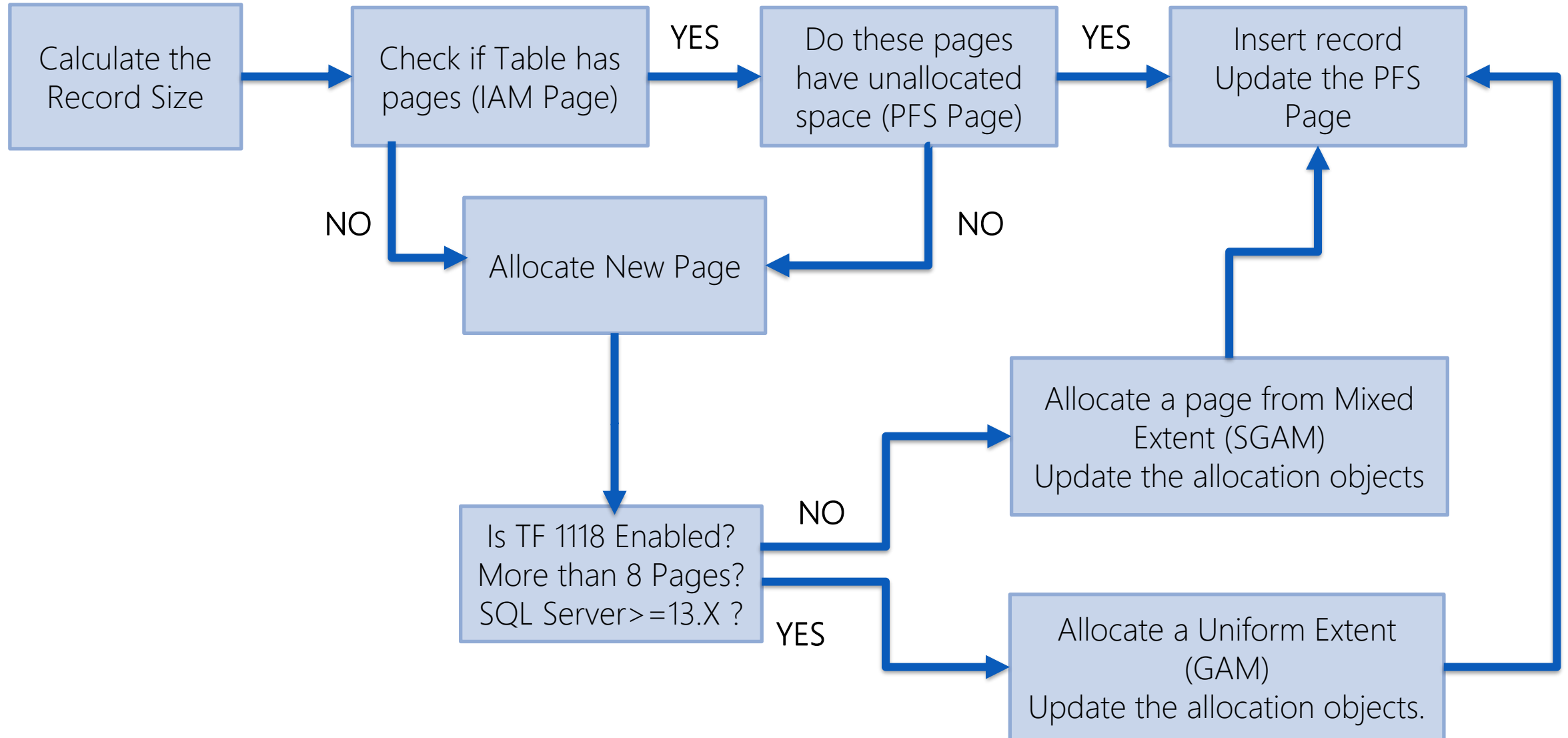
Up to, and including, SQL Server 2014 (12.x)

- Trace flag 1118 is used to change default allocation to use uniform extents

Starting with SQL Server 2016 (13.x)

- For TempDB: TF 1118 functionality is automatically enabled.
- For user databases: behavior controlled with SET MIXED\_PAGE\_ALLOCATION option of ALTER DATABASE
  - default value set to OFF
  - TF 1118 has no effect.

# Tying it all together



# Lesson 4: SQL Server Log File Structure

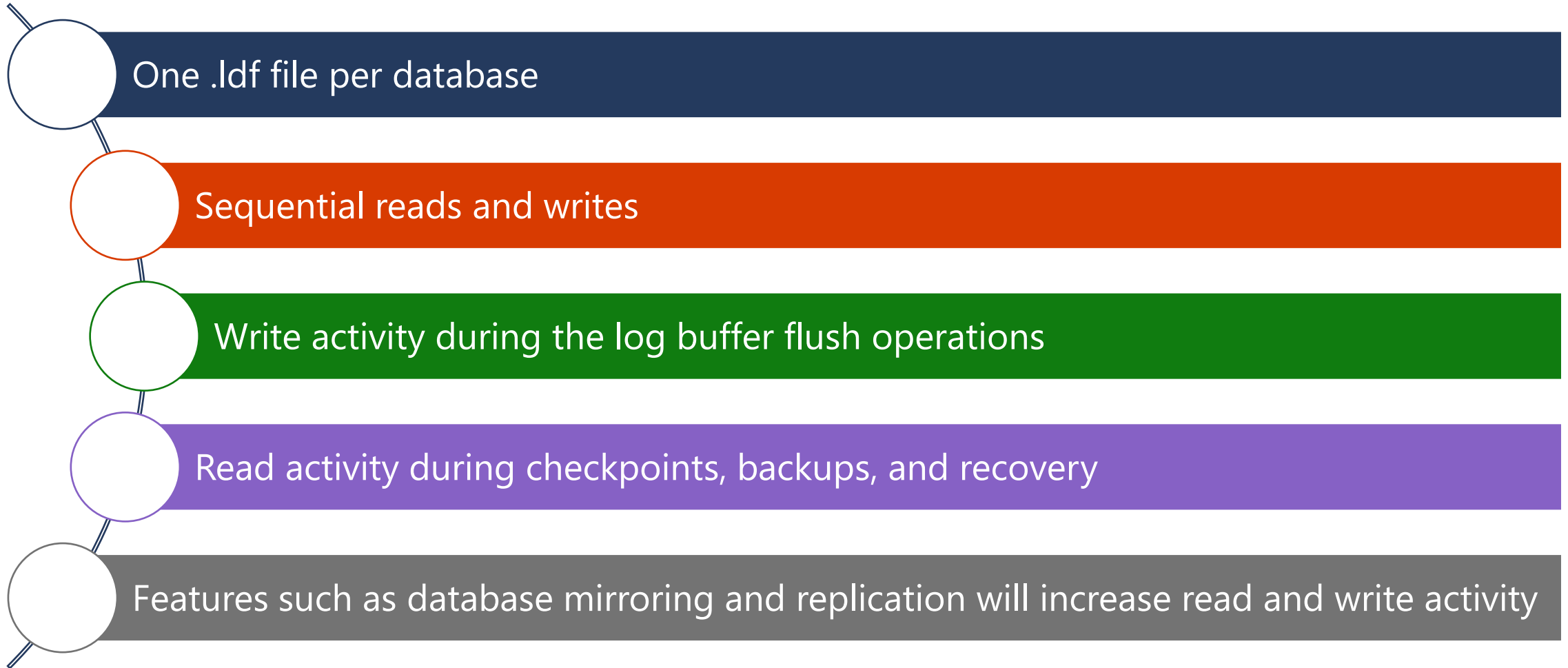
# Objectives

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

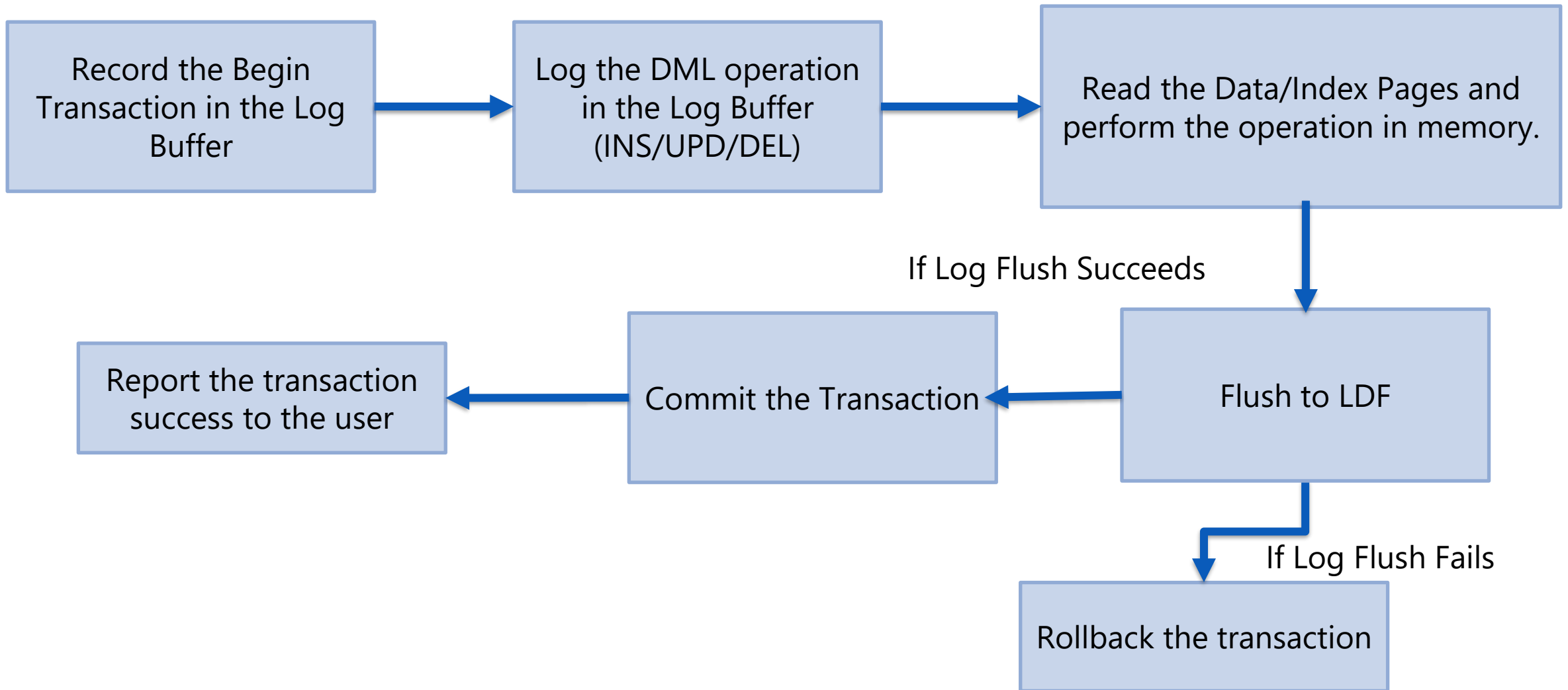
Records identified by increasing Logical Sequence Numbers (LSNs)

Recorded operations:

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

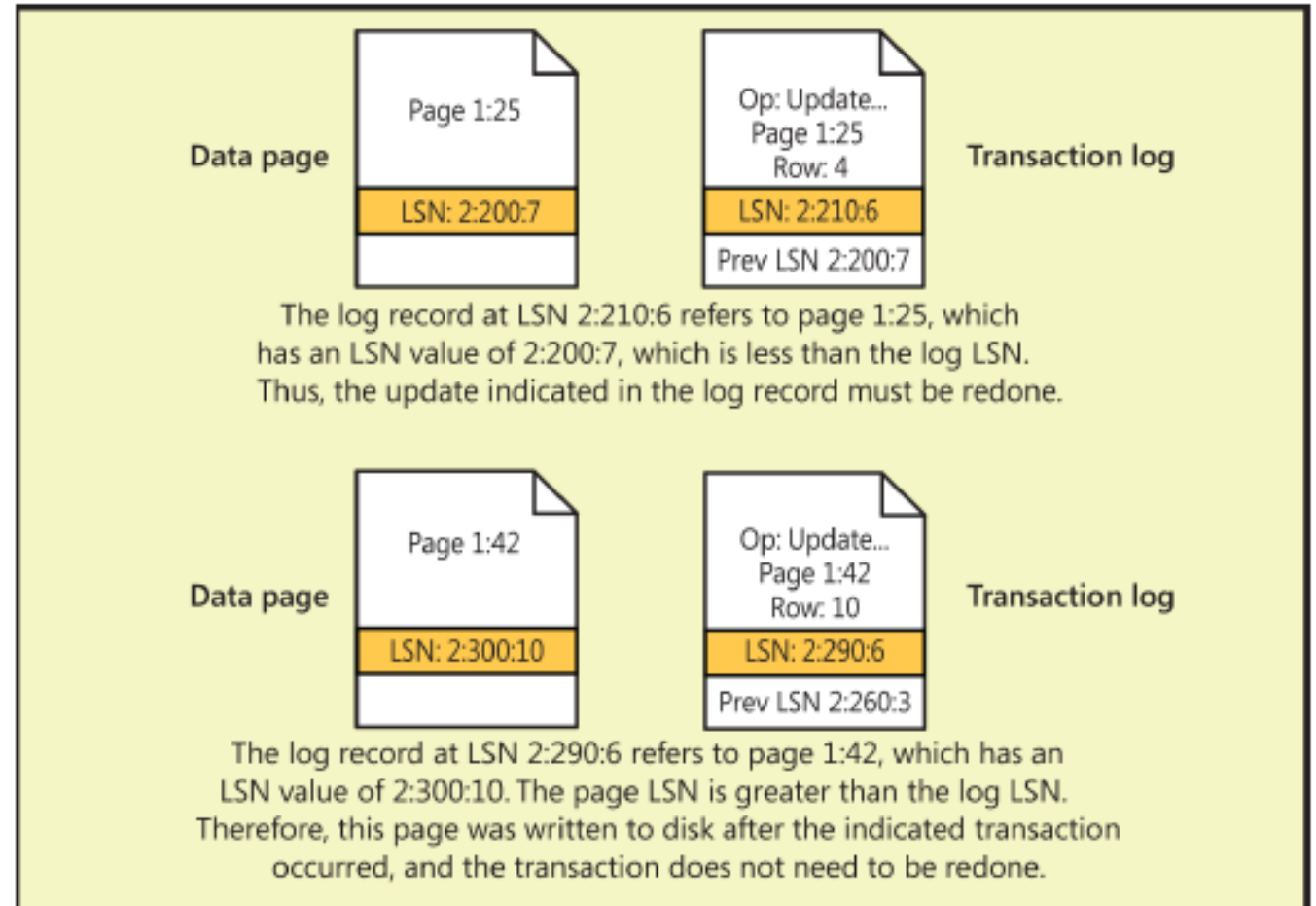


# Microsoft SQL Server Transaction Logging



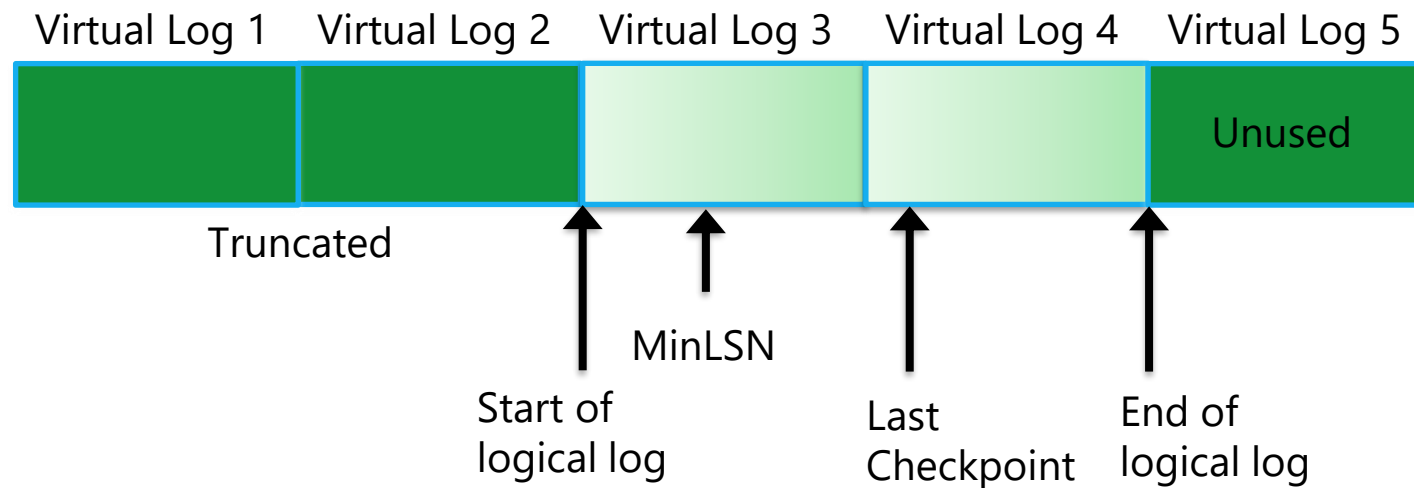
# Page LSN and recovery

- 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



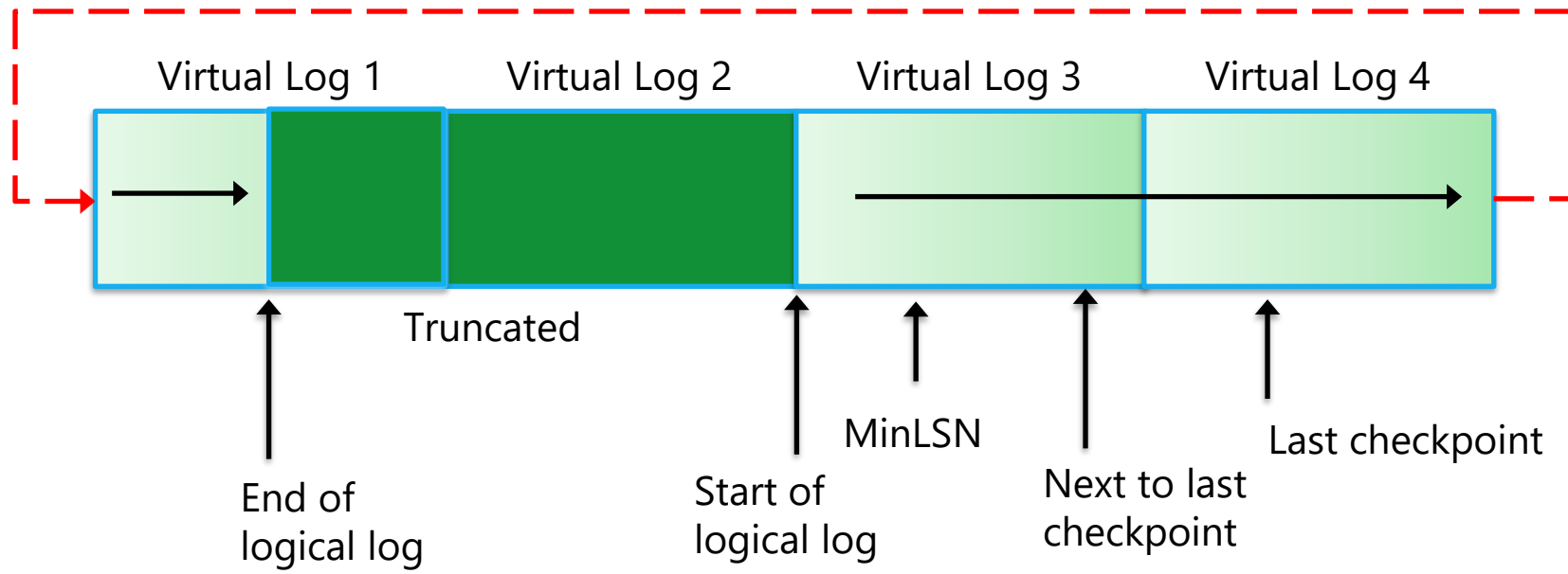
# Physical Log File Structure

- 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.

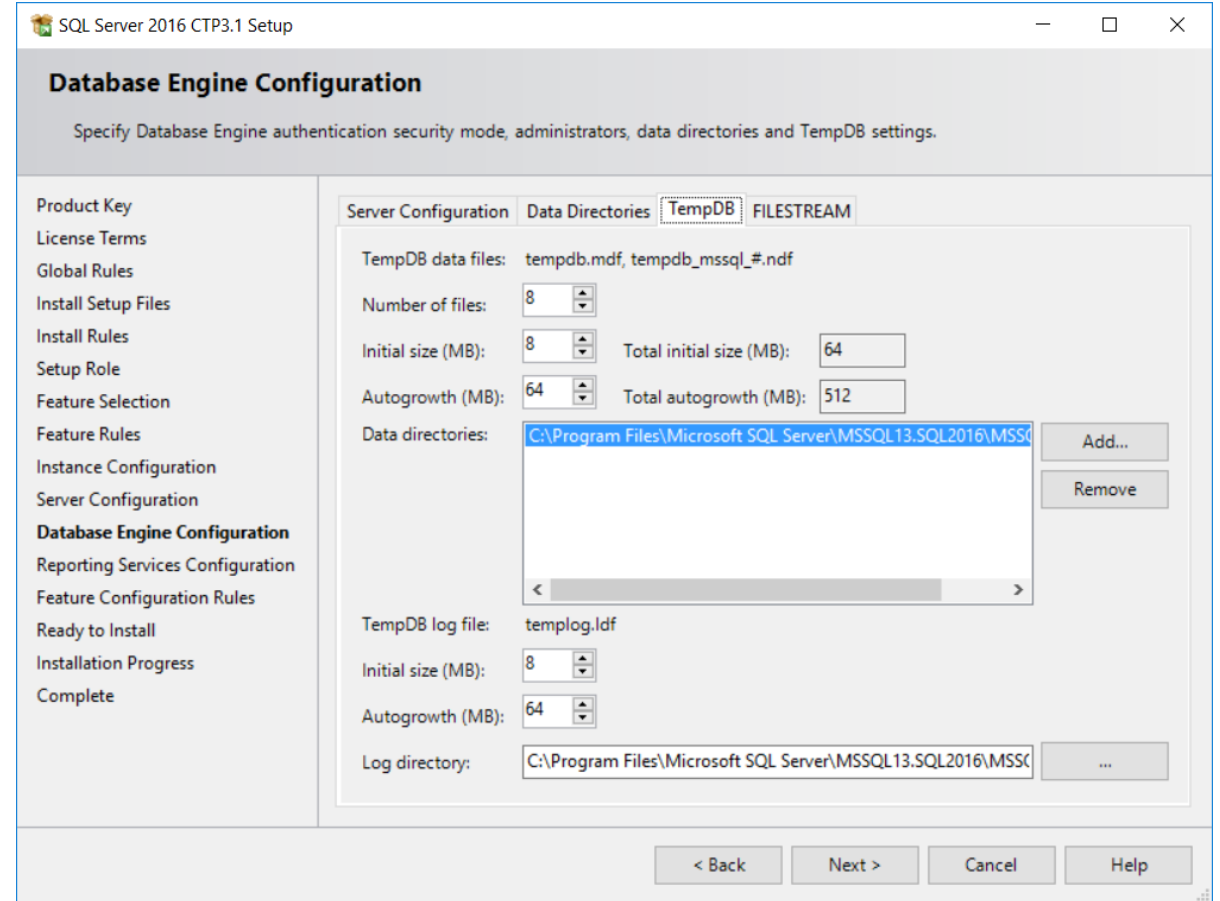
Allocation page latching protocol improved.

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

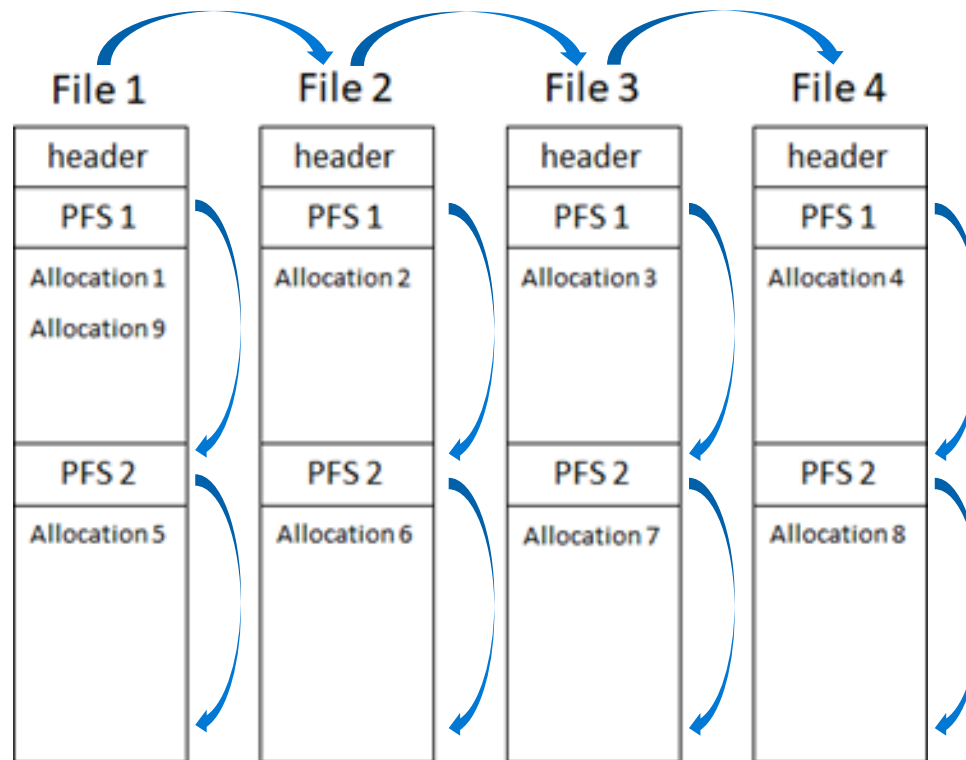
[TEMPDB Files, Trace Flags, and Updates by Pam Lahoud](#)

[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.

