Data Organization and Processing

Data Organization in Microsoft SQL Server 2012

(NDBI007)

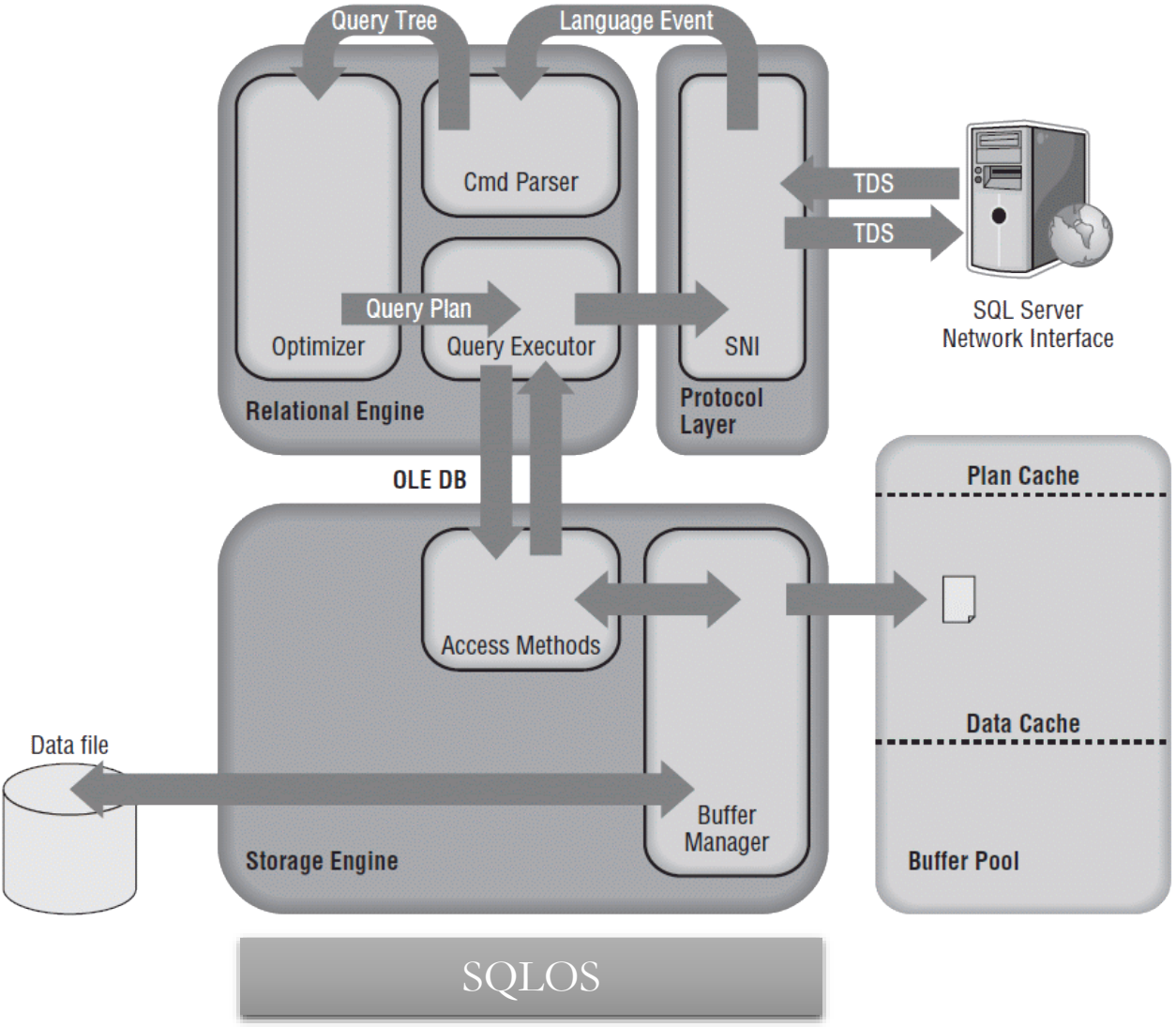
David Hoksza
http://siret.cz/hoksza

Outline

- Database server structure
  - databases
  - database files
  - memory
  - pages

- Data access
  - heaps
  - clustered indexes
  - nonclustered indexes

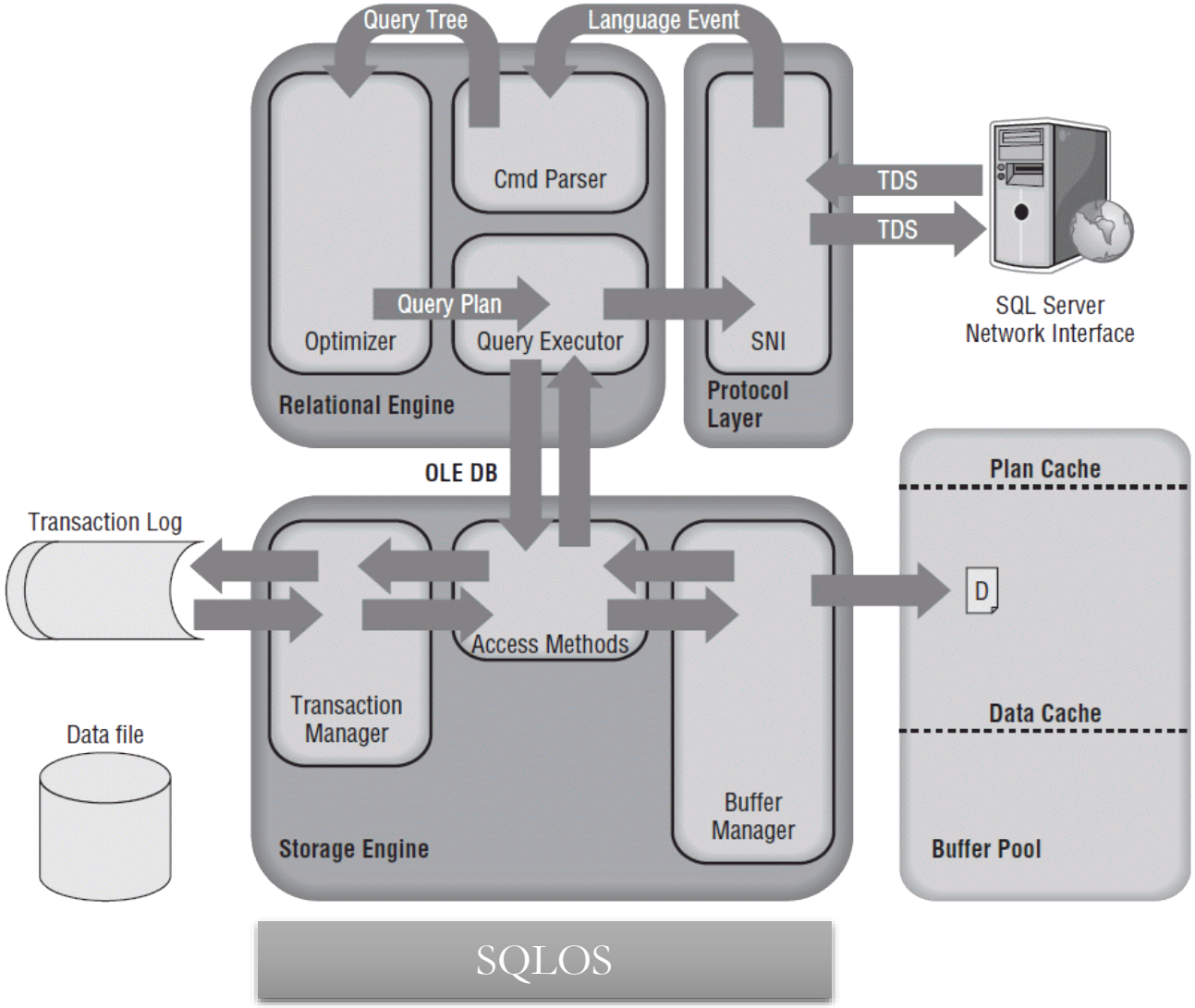
SELECT Query Lifecycle



1. *SNI* client-server connection using *TCP/IP* (or other protocol)
2. SELECT statement using *TDS (Tabular Data Streams) messages* between TDS endpoints
3. Unwrapping *TDS message*
4. Sending "SQL Command" to Command Parser
5. Checking *Plan Cache* in the *Buffer Pool*
6. If the plan is not cached, passing a query tree to the *Optimizer*
7. *Query plan* passed to *Query Executor*
8. Passing *Query Plan* to *Access Methods (AM)*
9. Checking the existence of the required page in the *Data Cache* by the *Buffer Manager*
10. Passing the requested page(s) back to the *AM*
11. Passing the results set to the *Relational Engine*

credit: SQL Server 2008 Internals and Troubleshooting

UDPATE Query Lifecycle



1. Identification of the page to update as in the SELECT lifecycle
2. The Access Methods require *Write-Ahead Logging* from the *Log Manager* (part of the *Transaction Manager*)
3. Page changes stored in the *Transaction Log*
4. The AM receive confirmation and pass the request on to the *Buffer Manager*
5. Modification of the page in cache
6. Confirmation sent to the AM and the client

credit: SQL Server 2008 Internals and Troubleshooting

Buffer Pool and the Data Cache

- **Buffer pool** (BP)
  - **main memory component**
  - BP **provides buffers** to other components which require memory
  - all memory not provided by other memory components is stored in the BP

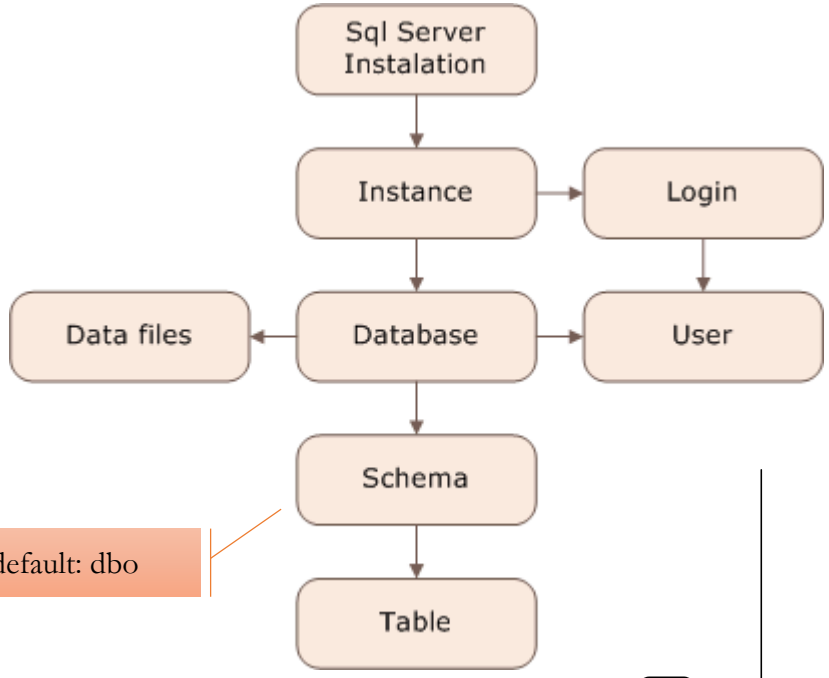
- **Buffer**
  - **a page in memory** having the same size as data or index page (8KB)
  - every **disk page** has to be **cached first** before it can be used anywhere in the server

- **Buffer manager**
  - **manages disk I/O** to fetch data and index pages into the data cache

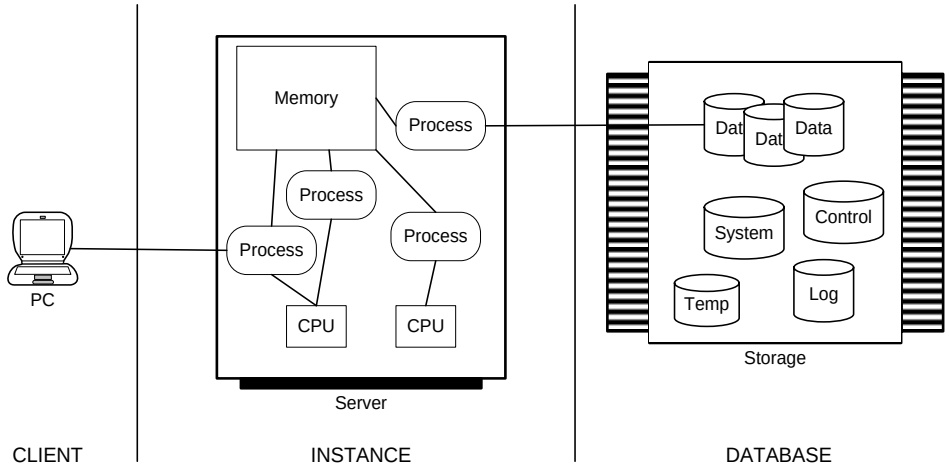
Access to In-Memory Pages

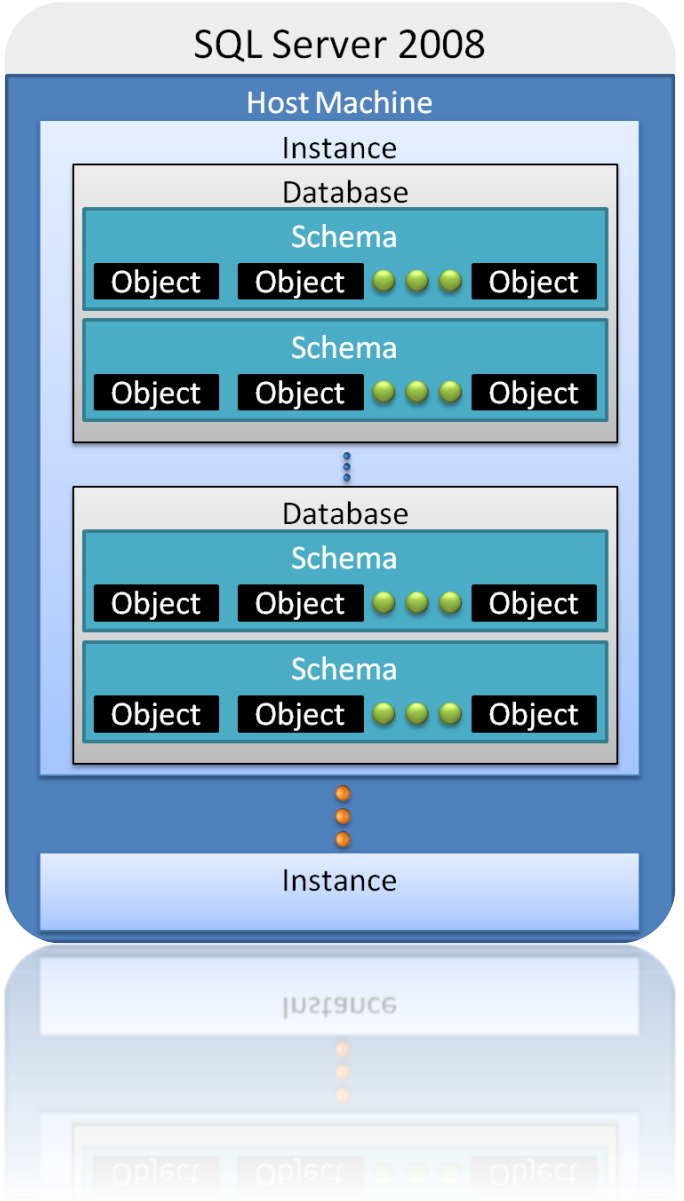
- The buffer pool can be huge (up to tens of GB) → **mechanism for identification of a page in memory** is needed
  - Disk **page is addressed** by a triplet **dbid:fileno:pageno**
    - **dbid** id of the database
    - **fileno** id of the database file
    - **pageno** id of the page
  - A **hash function** is used to **map page keys to buffers**

MSSQL Server Hierarchy



default: dbo



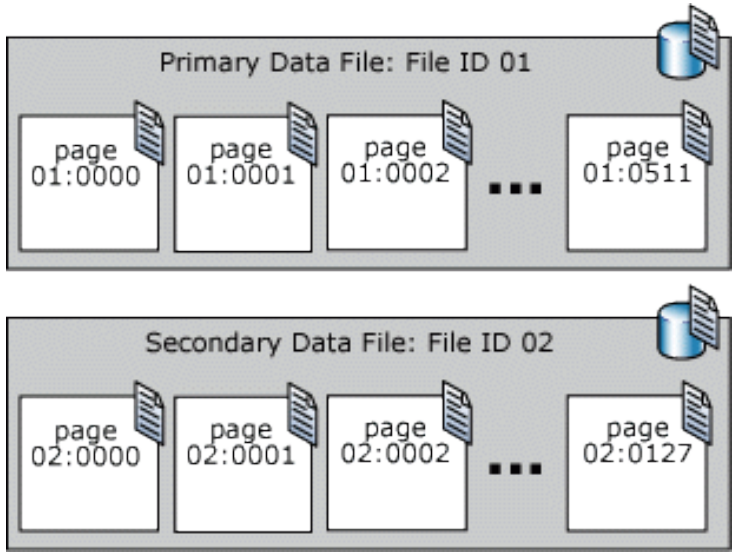


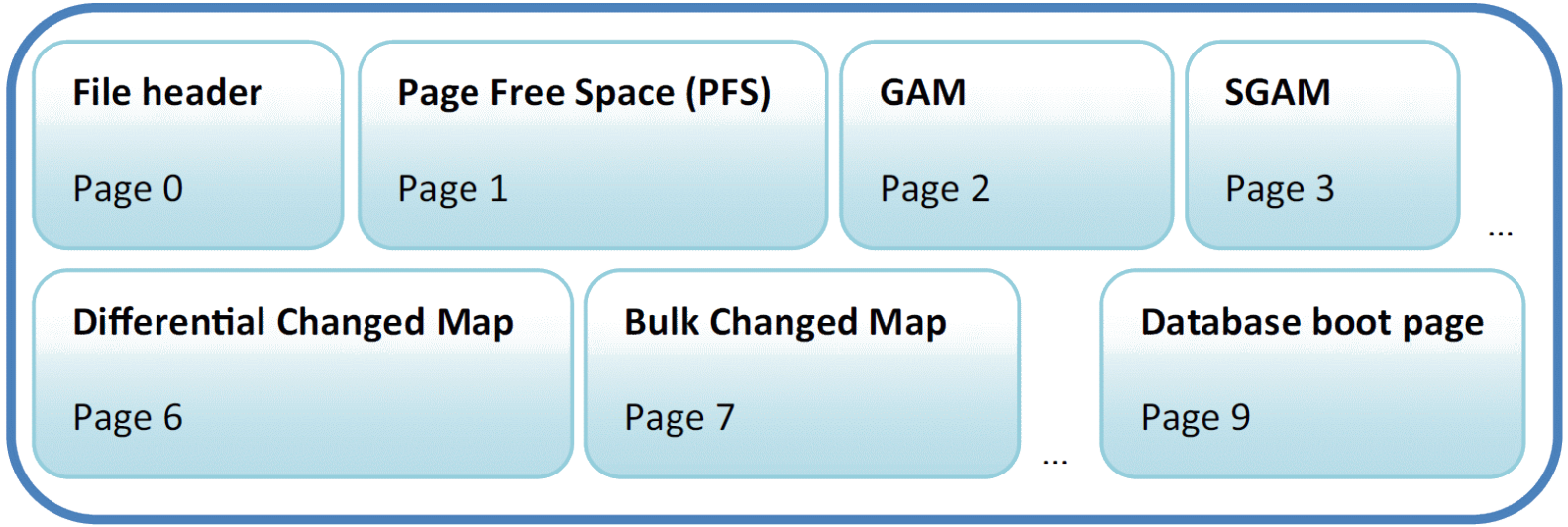
source: http://technet.microsoft.com/en-us/sqlserver/dd535524.aspx

Database Files

- 1 database – N files ($N > 1$)
- **DB files**
  - **primary**
    - one per database
    - stores information about the remaining database files
    - stores **data**
    - .mdf
  - **secondary**
    - stores data
    - .ndf
  - **transaction log**
    - stores **information about transactions** for recovery purposes
    - .ldf

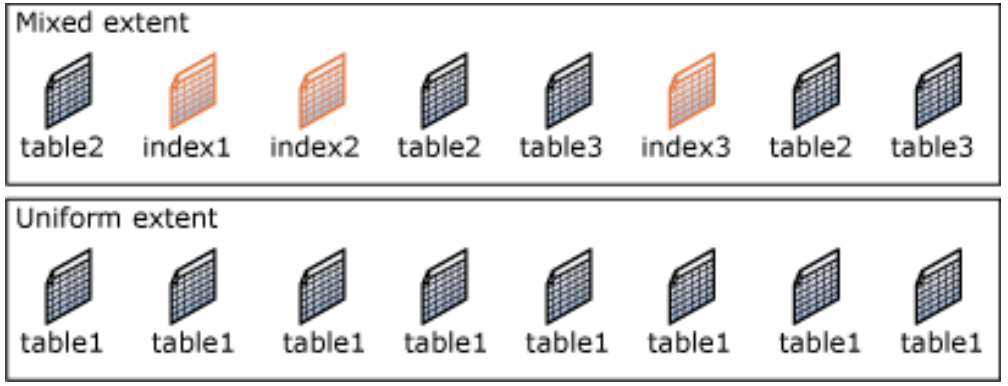
Database File Structure





Extent

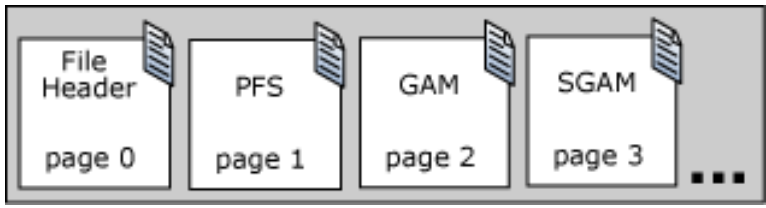
- The **basic unit in which space is managed**
  - **8** contiguous **pages** ⇒ 64KB
- **Types**
  - uniform
  - mixed
- First page of a new table → mixed extent
- New index
  - large → uniform extents
  - small → mixed extents



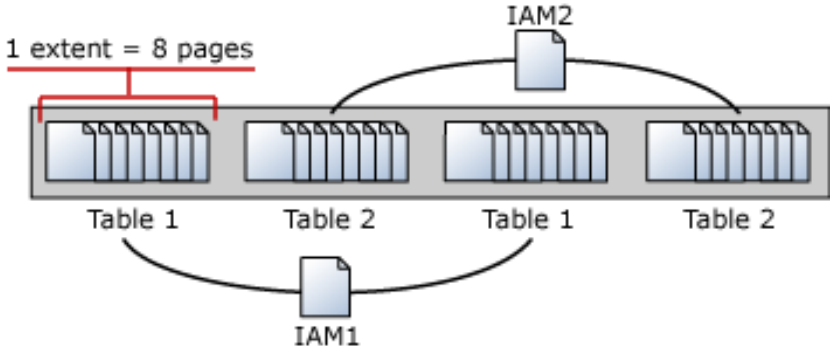
Space Management

Extent status	GAM bit setting	SGAM bit setting
Free	1	
Uniform extent (allocated)	0	0
Mixed extent with free pages	0	1
Full mixed extent	0	0

- **G**lobal **A**llocation **M**ap (**GAM**)
  - records **what extents have been allocated**
  - 1 = free extent, 0 = allocated extent
  - covers 64,000 extents
- **S**hared **G**lobal **A**llocation **M**ap (**SGAM**)
  - records **which extents** are currently **being used as mixed extents** and **have at least one unused page**
  - covers 64,000 extents
  - 1 = extent is being used as a mixed extent and has a free page, 0 = extent is not used as a mixed extent, or it is a mixed extent and all its pages are being used
- **P**age **F**ree **S**pace (**PFS**)
  - allocation status of each page – 1B

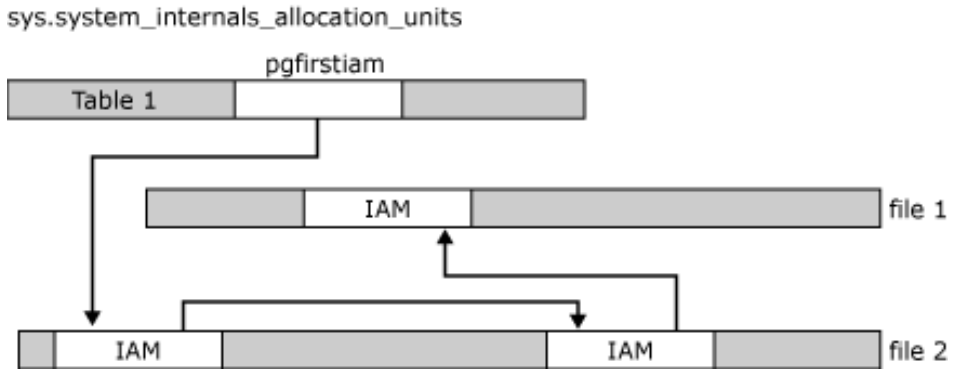


Space Management (cont.)



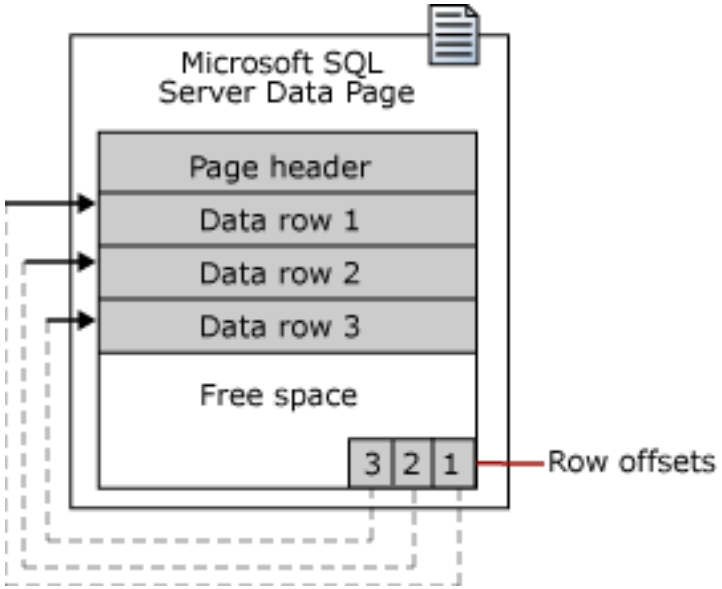
- Index **A**llocation **M**ap (**IAM**)
  - a bitmap tracking extents in 4GB used by an allocation unit
  - can be linked → IAM chain
  - **maps space allocation** for
    - heaps and b-trees
    - LOB data
    - row-overflow data
  - **IAM page**
    - 96-byte page header
      - single-page array (length 8)
    - **bitmap for extents belonging to the allocation unit**

- **Inserting a new row**
  - **IAM** → extents for an allocation unit
  - **PFS** → free pages in identified extents



Pages

- **Data** stored in **pages**
  - **8KB**
    - header
    - rows
    - row offsets



- **Allocation units**
  - **collection of pages within a table**
    - **IN_ROW_DATA** (hobt - Heap Or B-Tree)
      - data or index rows that contain all data, except large object (LOB) data (majority of data)
    - **LOB_DATA (LOB)**
      - TEXT, NTEXT, IMAGE, …
      - stored out-row
    - **ROW_OVERFLOW_DATA (SLOB - Small-LOB)**
      - variable length data stored in VARCHAR, VARBINARY, .NET, …
      - stored in-row unless the row size exceeds 8KB

Tables Organization

- **Heap**
  - a table with no clustered index
- **Clustered table/index**
  - a table over which a clustered index is defined
  - leaf level contains the actual data
- **Nonclustered index**
  - index over a table which can be both heap or clustered table

Internals

- DataBase Consistency Check (till 2000)
- DataBase Console Commands

- **Informational**
  - collecting and presenting various information about the system
- **Validation**
  - validation of the consistency of DB, tables, indexes, catalogues, page allocation

- **Maintenance**
  - tasks related to maintaining databases, indexes, …
- **Miscellanous**
  - various tasks such as turning on the flags, removing DLLs from library, …

- http://msdn.microsoft.com/en-us/library/ms188796.aspx

DBCC IND

- **DBCC IND** ({ **'dbname'** | dbid }, { **'objname'** | objid }, { **nonclustered indid** | 1 | 0 | -1 | -2 } [,**partition_number**])
  - **0 …** Displays information for in-row data pages and in-row IAM pages of the specified object.
  - **1 …** Displays information for all pages, including IAM pages, data pages, and any existing LOB pages or row-overflow pages of the requested object. If the requested object has a clustered index, the index pages are included.
  - **–1 …** Displays information for all IAMs, data pages, and index pages for all indexes over the specified object. This includes LOB and row-overflow data.
  - **–2 …** Displays information for all IAMs for the specified object
  - **Nonclustered index ID (2 – 250, 256 – 1005**) … Displays information for all IAMs, data pages, and index pages for one index. This includes LOB and row-overflow data that might be part of the index's included columns
    - **sys.indexes** … list of indexes

DBCC IND Output

PageFID	File ID containing the page
PagePID	Page number within that file
IAMFID	File ID containing the IAM managing this page
IAMPID	Page number within that file of the IAM managing this page
ObjectID	Object ID of the parent table
IndexID	Index ID—valid values are 0–250 and 256–1005
PartitionNumber	Partition number within the table or index for this page
PartitionID	ID for the partition containing this page (unique in the database)
iam_chain_type	Type of allocation unit this page belongs to: in-row data, row-overflow data, or LOB data
PageType	Page type: 1 = data page, 2 = index page, 3 = TEXT_MIXED_PAGE, 4 = TEXT_TREE_PAGE, 10 = IAM page
IndexLevel	Level of index; 0 is the leaf level and levels are counted up from the leaf to the root
NextPageFID	File ID containing the next page at this level
NextPagePID	Page number within that fi le for next page at this level
PrevPageFID	File ID containing the previous page at this level
PrevPagePID	Page number within that fi le for previous page at this level

DBCC PAGE

- **DBCC PAGE** ({dbid | dbname}, filenum, pagenum[, printopt])

- p**rintopt**

  - **0** prints the buffer header and page header
  - **1** prints the buffer header, page header, each row separately, and the row offset table
  - **2** prints the buffer and page headers, the page as a whole, and the offset table
  - **3** prints the buffer header, page header, each row separately, and the row offset table; each row is followed by each of its column values listed separately

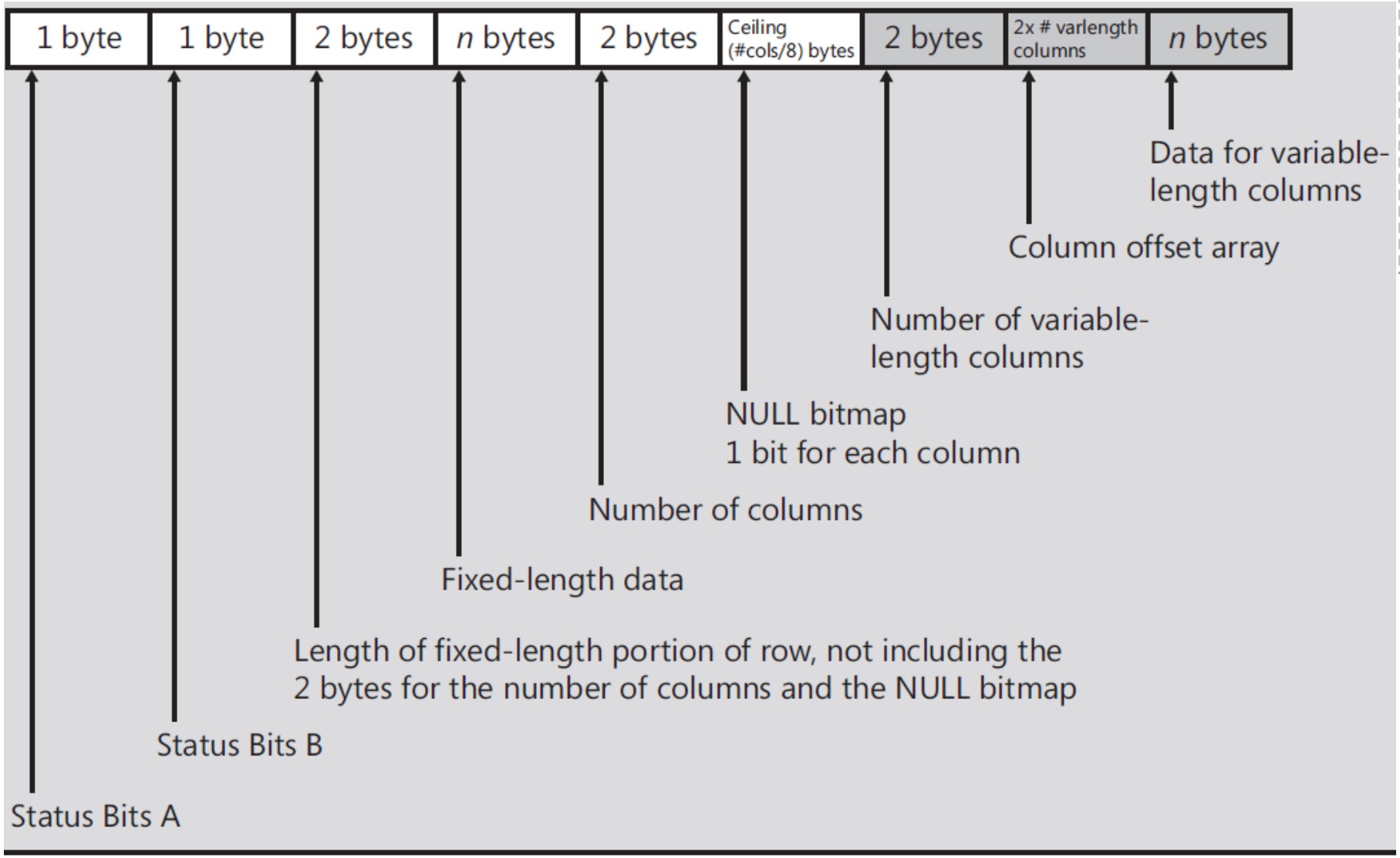
Page Structure

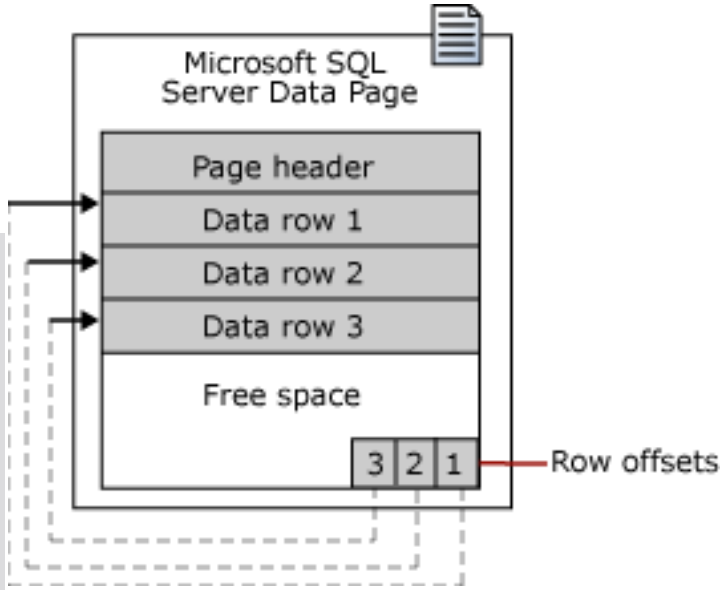
- Four main sections
  - **BUFFER**
    - information about the buffer for the given page
  - **PAGE HEADER**
    - header information common to every row
  - **DATA**
    - information for each row
    - differs based on printopt parameter
  - **OFFSET TABLE**
    - row offset array

Page Header

Field	Meaning
pageID	The file number and page number of this page in the database
nextPage	The file number and page number of the next page if this page is in a page chain
prevPage	The file number and page number of the previous page if this page is in a page chain
Metadata: ObjectId	The ID of the object to which this page belongs
Metadata: PartitionId	The ID of the partition that this page is part of
Metadata: AllocUnitId	The ID of the allocation unit that contains this page
LSN	The Log Sequence Number (LSN) corresponding to the last log entry that changed this page
slotCnt	The total number of slots (rows) used on this page
Level	The level of this page in an index (which always is 0 for leaf pages)
indexId	The index ID of this page (always 0 for data pages)
freeData	The byte offset of the first free space on this page
Pminlen	The number of bytes in fixed-length portion of rows
freeCnt	The number of free bytes on the page
reservedCnt	The number of bytes reserved by all transactions
xactreserved	The number of bytes reserved by the most recently started transaction
tornBits	A bit string containing 1 bit per sector for detecting torn page writes (or checksum information if torn_page_detection is not on)
flagBits	A 2-byte bitmap that contains additional information about the page

Data Row Structure





Data Row Structure (cont.)

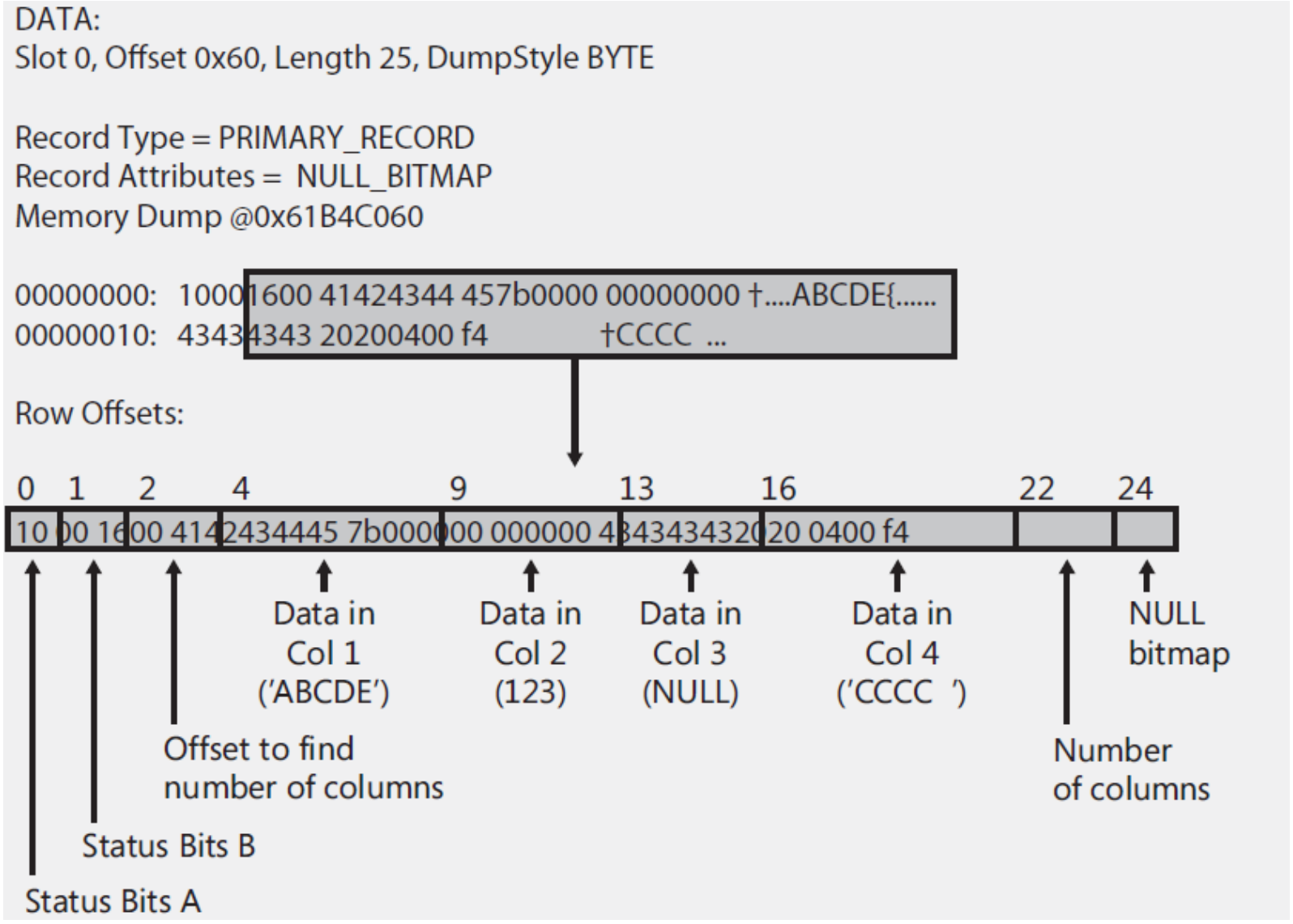
- **Status Bits A**
  - **Bit 0** contains versioning information. In SQL Server 2012, this is always 0.
  - **Bits 1 through 3** are taken as a three-bit value, 0 indicates a primary record, 1 indicates a forwarded record, 2 indicates a forwarding stub, 3 indicates an index record, 4 indicates a blob fragment or row-overflow data, 5 indicates a ghost index record, 6 indicates a ghost data record, and 7 indicates a ghost version record.
  - **Bit 4** indicates that a NULL bitmap exists. In SQL Server 2012, a NULL bitmap is always present, even if no NULLs are allowed in any column.
  - **Bit 5** Indicates that variable-length columns exist in the row.
  - **Bit 6** Indicates that the row contains versioning information.
  - **Bit 7** Not used in SQL Server 2012.

Fixed-Length Records (1)

```
CREATE TABLE Fixed
(
     Col1       CHAR(5)   NOT NULL,
     Col2       INT       NOT NULL,
     Col3       CHAR(3)   NULL,
     Col4       CHAR(6)   NOT NULL
);

INSERT Fixed VALUES ('ABCDE', 123, NULL, 'CCCC');
```

Fixed-Length Records (2)



Variable-Length Records (1)

```
CREATE TABLE Variable
(
      Col1 CHAR(3)          NOT NULL,
      Col2 VARCHAR(250)     NOT NULL,
      Col3 VARCHAR(5)       NULL,
      Col4 VARCHAR(20)      NOT NULL,
      Col5 SMALLINT         NULL
);
INSERT Variable VALUES ('AAA',
REPLICATE('X', 250), NULL, 'ABC', 123);
```

Variable-Length Records (2)

```
Col1 CHAR(3)          NOT NULL,
Col2 VARCHAR(250)     NOT NULL,
Col3 VARCHAR(5)       NULL,
Col4 VARCHAR(20)      NOT NULL,
Col5 SMALLINT         NULL
```

```
INSERT Variable VALUES ('AAA',
REPLICATE('X', 250), NULL, 'ABC', 123);
```

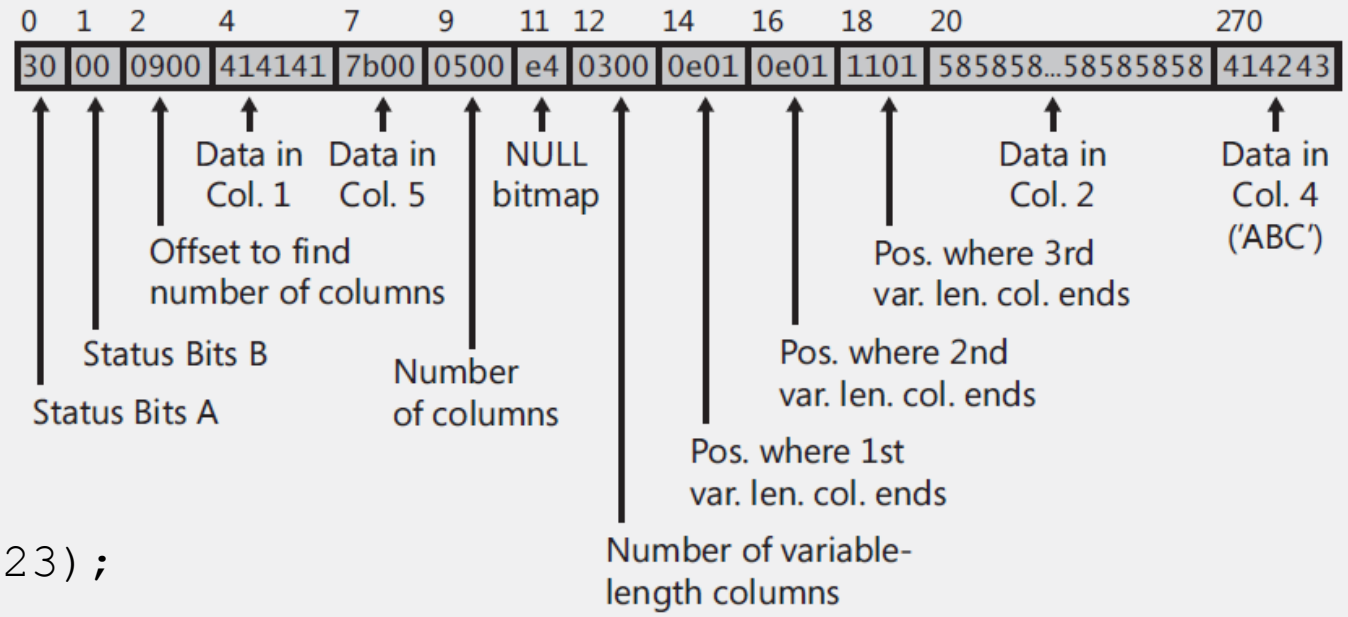
```
DATA:

Slot 0, Offset 0x60, Length 273, DumpStyle BYTE

Record Type = PRIMARY_RECORD
Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
Memory Dump @0x61C4C060

00000000:   30000900 4141417b 000500e4 03000e01 †0...AAA{........
00000010:   0e011101 58585858 58585858 58585858 †....XXXXXXXXXXXX
00000020:   58585858 58585858 58585858 58585858 †XXXXXXXXXXXXXXXX
00000030:   58585858 58585858 58585858 58585858 †XXXXXXXXXXXXXXXX
...                                             ...
000000F0:   58585858 58585858 58585858 58585858 †XXXXXXXXXXXXXXXX
00000100:   58585858 58585858 58585858 58584142 †XXXXXXXXXXXXXXAB
00000110:   43                                  †C

Row Offsets:
```



Identification of Data Pages of an Allocation Unit

```
SELECT
    OBJECT_NAME(p.object_id) AS name, i.name as ix_name, rows, a.type_desc as
page_type_desc, total_pages AS pages,
    dbo.convert_RIDs(a.first_iam_page) AS iam_first_page,
    dbo.convert_RIDs(root_page) AS index_root,
    dbo.convert_RIDs(first_page) AS first_page
FROM
        sys.partitions p
JOIN    sys.system_internals_allocation_units a ON p.partition_id = a.container_id
JOIN    sys.indexes AS i ON p.index_id = i.index_id AND i.object_id = p.object_id
WHERE
    p.object_id = OBJECT_ID('Person.Address')
```

```
DBCC IND('AdventureWorks2014', 'Person.Address', -1);
```

Heap RID decomposition

```
CREATE FUNCTION convert_RIDs(@rid BINARY(8))
RETURNS VARCHAR(30)
--FileID:PageID:SlotNumber
AS
BEGIN
    RETURN (
        CONVERT (VARCHAR(5),
            CONVERT(INT, SUBSTRING(@rid, 6, 1)
            + SUBSTRING(@rid, 5, 1)) )
        + ':' +
        CONVERT(VARCHAR(10),
            CONVERT(INT, SUBSTRING(@rid, 4, 1)
            + SUBSTRING(@rid, 3, 1)
            + SUBSTRING(@rid, 2, 1)
            + SUBSTRING(@rid, 1, 1)) )
        + ':' +
        CONVERT(VARCHAR(5),
            CONVERT(INT, SUBSTRING(@rid, 8, 1)
            + SUBSTRING(@rid, 7, 1)) ) )

END;
```

NULLS

- **Fixed-length columns**
  - always **the same length**, even if the column contains NULL
- **Variable-length columns**
  - **NULLs don't take any space** in the variable-length data part of the row (except for the 2 byte column offset information field)

```
CREATE TABLE dbo.null_varchar(
id INT PRIMARY KEY IDENTITY(1,1),
col1 VARCHAR(10) NULL,
col2 VARCHAR(10) NULL,
col3 VARCHAR(10) NULL,
col4 VARCHAR(10) NULL,
col5 VARCHAR(10) NULL,
col6 VARCHAR(10) NULL,
col7 VARCHAR(10) NULL,
col8 VARCHAR(10) NULL,
col9 VARCHAR(10) NULL,
col10 VARCHAR(10) NULL
);
```

```
INSERT INTO null_varchar(col10)
SELECT 'a';

INSERT INTO null_varchar(col1)
SELECT 'b';

INSERT INTO null_varchar
SELECT
'','','','','','','','','','c';

INSERT INTO null_varchar
SELECT
'd','','','','','','','','','';
```

(Non)clustered Index Structure

- An index structure corresponds to a **B+-tree** with both leaf and inner nodes linked together
- The process of traversing clustered or nonclustered index is identical
- **Leaf level differs**
  - **clustered index**
    - **leaf level** contains the **data pages** themselves
  - **NC index on a heap**
    - **leaf level** contains **pointers** (dbid:fileid:pageno) to the heap
  - **NC index on a clustered index**
    - **leaf level** contains **keys of the clustered index** (possibly enriched with uniquifier)

```
CREATE TABLE sp_tablepages
(
    PageFID     tinyint,
    PagePID         int,
    IAMFID      tinyint,
    IAMPID          int,
    ObjectID        int,
    IndexID     tinyint,
    PartitionNumber tinyint,
    PartitionID     bigint,
iam_chain_type varchar(30),
    PageType       tinyint,
    IndexLevel     tinyint,
    NextPageFID  tinyint,
    NextPagePID      int,
    PrevPageFID   tinyint,
    PrevPagePID     int,
  CONSTRAINT sp_tablepages_PK
        PRIMARY KEY (PageFID,
PagePID)
);
```

Index Traversal (1)

```
TRUNCATE TABLE sp_tablepages;
INSERT sp_tablepages
EXEC ('DBCC IND (''AdventureWorks2014'', ''Person.Person'', 1)');
SELECT IndexLevel
     , PageFID
     , PagePID
     , PrevPageFID
     , PrevPagePID
     , NextPageFID
     , NextPagePID
FROM sp_tablepages
ORDER BY IndexLevel DESC, PrevPagePID;
```

IndexLevel	PageFID	PagePID	PrevPageFID	PrevPagePID	NextPageFID	NextPagePID
2	1	234	0	0	0	0
1	1	232	0	0	1	233
1	1	233	1	232	1	235

…

Index Traversal (2)

```
DBCC TRACEON(3604)

DBCC PAGE (AdventureWorks2014, 1, 234, 3) --
first page of level 2 (FileId, PageID,
detail_level)
```

```
FileId  PageId  Row Level   ChildFileId    ChildPageId    EmployeeID (key)   KeyHashValue
------  -----------    ------  ------  -----------    -----------    ----------------      -------
---------
1       232            0       1       1              168            NULL                NULL
1       232            1       1       1              169            21                      NULL
1       232            2       1       1              170            41                      NULL
1       232            3       1       1              171            61                      NULL
1       232            4       1       1              172            81                      NULL
1       232            5       1       1              173            101                     NULL
...
```

Index Structure - Example

- Traversing clustered index when searching for an employee with ID 9002
- Referencing a clustered index record from leaf level (SSN = 060-06-1991)
- Referencing a heap record from leaf level

Deleting from a Heap

- When deleting a record from a heap
  - the free space is not reclaimed
  - the offset for the given record is set to 0