The new memory architecture consists of 4 major components

* Memory Nodes
* Memory Clerks
* Memory objects
* Memory Broker

**Memory Node:** Memory nodes are internal SQLOS objects. The number of memory nodes depends on the NUMA configuration of the server. Memory nodes can be thought of as a large pool of Memory, which different components (Clerks) can allocate memory from. Memory Nodes expose a set of Allocators which are used by all memory Clerks to allocate the desired amount of memory from the Node.

* *Page Allocators*: There are 4 primary types of page allocators exposed by the memory node. These allocators are used to allocate a set of 8KB pages. Remember that SQL Server page granularity is 8KB and any memory allocated by these page allocators would always be in multiple of 8KB.
* *Single Page Allocator(BPool usage :total Single Page Allocator value in all nodes in the output of dbcc memorystatus )*: As the name suggests, this allocator is used to allocate one 8KB page at a time.
* *Multi Page Allocator*: (*MTL usage :total multi Page Allocator value in all nodes in the output of dbcc memorystatus*) Allocates Multiple contigous 8KB Pages

*This value can be recognized through*

**Select sum(multi\_pages\_kb) from sys.dm\_os\_memory\_clerks- represents MTL usage**

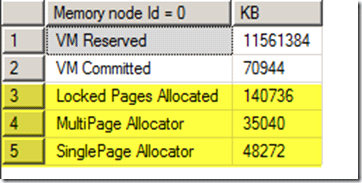
**Select sum(single\_pages\_kb) from sys.dm\_os\_memory\_clerks- represents Bpool usage**

Notes: we can also identify which memory clears consuming majority of memory using in MTL and Bpool using following

**Select \* from sys.dm\_os\_memory\_clearks order by multi\_pages\_kb (or) single\_pages\_kb desc**

* *Large Page Allocators*: These are used to allocate Large Pages. This feature is only present in IA64 or X64 editions of SQL, running on servers with more than 8GB of memory. Trace Flag 834 has to be enabled in order for SQL to use Large Pages. On a X64 server the Large Page granularity is 2MB, while on IA64 its 16MB.
* *Reserved Pages*: Is special purpose allocator, which allocates a set of pages for emergencies. Typically the “Fail Safe” memory for SQL.
* *Virtual Memory Allocator*: Uses Windows VirtualAlloc API’s, when SQL needs to allocate contigous memory which does not necessarily fall in the 8K page boundary.
* *Shared Memory Allocator*: Uses windows file mapping API’s and provides LPC (shared memory) capabilities.

DBCC memory status gives some information about the Memory Nodes, but they are NOT exactly memory node but CPU nodes.



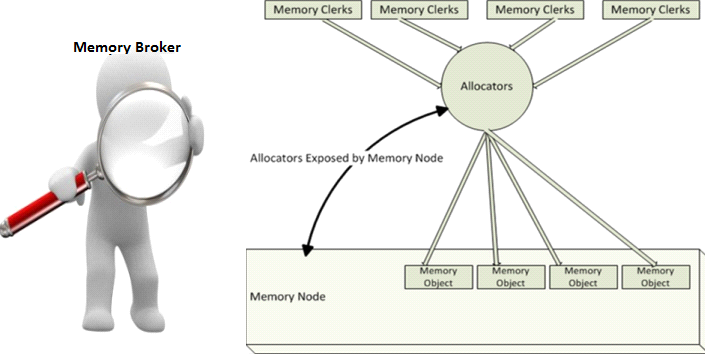
Locked Pages allocated would only show up when SQL Server has been configured to use Lock Pages in memory

**Memory Clerks:** Clerks are consumers of memory in SQL. In general there are 4 kinds of clerks, generic, cache store, object store, user store. The clerks make use of the memory node allocators to allocate memory. SQL has a vast number of Clerks predefined, and each of these clerks corresponds to the type of allocation which needs to be done. SQL exposes the clerk information using the “sys.dm\_os\_memory\_clerks” DMV. This information is also exposed through the DBCC MEMORYSTATUS. In the memory status output one can see the clerks being listed multiple times. This is because of the fact that the clerks are created per NUMA node. So if we have 4 NUMA nodes on the machine, we could see the clerks being reported 4 times.

**Memory Objects:** Memory objects are the memory allocations made through the clerks. There are 3 kinds of memory objects which are supported by SQL. Memory objects can be viewed using the “sys.dm\_os\_memory\_objects” DMV.

* Variable Memory Object: Supports variable size memory allocations.
* Mark/Shrink Objects: This allocations are typically done in two stages. During the first stage the allocations grows in size (more and more memory are added as part of this allocation) and during the second phase the memory is de-allocated. As Slava points out, these type of allocations are very useful in scenarios like compilation or execution.
* Fixed Size Memory objects: Supports memory allocations which are fixed in size.

**Memory Broker:** Memory broker can be though of as the Ring Master in a Circus. The purpose of Memory Broker is to provide a centralized mechanism to distribute memory or control the allocations made by each component in SQL Server. Memory Broker monitors the demand consumption of memory by each component and then based on the information collected, it calculates and optimal value of memory for each of these components. This information is then broadcast to each of the components, which grow or shrink their usage as required. Memory broker is exposed through the DMV “sys.dm\_os\_ring\_buffers” where the Ring\_buffer\_type is “RING\_BUFFER\_MEMORY\_BROKER



Let us start with basic components of a SQL Server Memory to understand better how its allocated,

SQL Memory has 2 basic components,

1. Memory to Leave(MemToleave)

2. Buffer Pool(Bpool)

**Memory to Leave** is a space used for external sources like COM objects(eg: sp\_OACreate), Extended store procedures, linked servers, OLEDB providers etc, MemToLeave is the smaller component and usually of a fixed size in 32 bit architecture 384 MB approximately. It 64 bit system VAS reservation does not happen at the start up, *We will see more about VAS allocation further in the topic.*

MemToLeave is virtual address space (VAS) that’s left un-used when SQL Server starts so that external components called by SQL Server are saved some address space. So in order for these technologies, .NET CLR, Linked Servers and extended stored procedures, to operate efficiently you must ensure that they too have access to sufficient memory.

This is why it is often recommended that you explicitly set the maximum amount of memory that SQL Server uses, as opposed to allowing it to consume all that is available on your server

It was named MemToLeave based on the way that SQL Server allocates memory when it starts up, but it lead to confusion over the idea that it was memory that SQL Server wouldn't allocate.  The appropriate terminology for what is commonly known as the MemToLeave is the VAS Reservation.

**Startup Allocation of Memory**

When SQL Server starts up, the first memory item that is reserved by the SQLOS is the VAS Reservation memory area.  This is a special contiguous memory area that is separate completely from the Bpool, and is for use by external consumers such as Extended Procedures, COM objects (OLE Automation calls), Linked Servers, OLEDB providers, SQL CLR, and is also used for allocations larger than 8KB (>8KB), also known as multi-page allocations, by the database engine (extemely large and complex query plans).  It has a calculated fixed size that is determined at startup based on the following formula:

**(MaxWorkerThreads \* StackSize) + DefautReservationSize**

The MaxWorkerThreads default on SQL Server 7.0 is 128 and on SQL Server 2000 it is 255.  However, on SQL Server 2005 and 2008 it is dynamic and depends on the number of schedulers (processors) you have configured in the SQL Server.  The formula for the number of **worker threads is**:

**((NumberOfSchedulers - 4) \* 8) + BaseThreadCount**

For 32 bit SQL Servers the BaseThreadCount is 256. For 64 bit SQL Servers the BaseThreadCount is doubled to 512.

Note: On SQL Server 2005 and 2008, if the number of schedulers is less than four, then the BaseThreadCount is the default value for the MaxWorkerThreads; 256 for 32 bit and 512 for 64 bit servers respectively.  You can also query this information from the system:

select max\_workers\_count   
from sys.dm\_os\_sys\_info

For a 32 bit SQL Server the default StackSize is 512KB so to determine the memory allocation to the VAS Reservation for a 2 processor 32 bit SQL Server the calculation would be (256\*512KB) + 256MB or 384MB of space.

For a 64 bit SQL Server the default StackSize is 2048KB or 2MB, so for the same server only 64 bit, it would be (512\*2048KB) + 256MB or 1280MB.  Keep in mind that only 256MB of this space is available for use by external objects both of these scenarios since the worker thread memory reservation is used to manage the worker threads in SQL Server.

Once this calculation and allocation has been completed, the next memory space setup by the SQL Server is the BPool or buffer pool.  The buffer pool size is determined by a number of factors, beginning with whether the server has AWE enabled on not.  If AWE is not enabled on the SQL Server, then the buffer pool is sized based on the remaining memory in the user address space minus the VAS Reservation allocation that has been previously calculated.  On 32 bit SQL Servers, the VAS is 4GB in size, and by default is split 50/50 into user address space and kernel address space, with 2GB assigned to each.  By using the /3GB or /USERVA startup switches in the Windows boot.ini file, this default configuration can be overridden to provide 3GB of VAS to the user address space, leaving 1GB for the kernel address space.

Once the user address space size has been determined by the SQL Server, the previously calculate VAS Reservation reservation is subtracted from it.  Then the resulting value is compared to the amount of available physical memory on the SQL Server, and the lower of the two numbers is the maximum available space for the buffer pool size.  This is then compared to the max server memory setting and the lower of the two values is then the maximum size of the buffer pool.  For a SQL Server with 1GB of Physical RAM and the default configuration of the Operating System and SQL Server options, this would equate to a maximum buffer pool size of 1GB based on the following calculations:

2048MB (2GB) user VAS - 384MB = 1664MB available VAS

1024MB (1GB) Physical Memory

Since the smaller of the two is the physical memory size, then this is the buffer pool size.  However, on a 2GB of Physical Memory server, the maximum size of the buffer pool would be 1664MB since the available user VAS is the smaller amount.  This is why adding the /3GB switch is often done in the boot.ini, which would change the calculation to be:

3072MB (3GB) user VAS - 384MB = 2688MB available VAS

4096MB (4GB) Physical Memory

Here the maximum size of the buffer pool is going to be 2688MB.  If however AWE is enabled on the SQL Server, then the maximum size of the buffer pool is the upper limit of the Physical RAM on the SQL Server or the max server memory setting, whichever is smaller.

**Note :**

**How do I determine my MemToLeave usage (multi\_pages\_kb in all nodes)?**

WITH VAS\_Summary AS

(

    SELECT Size = VAS\_Dump.Size,

    Reserved = SUM(CASE(CONVERT(INT, VAS\_Dump.Base) ^ 0) WHEN 0 THEN 0 ELSE 1 END),

    Free = SUM(CASE(CONVERT(INT, VAS\_Dump.Base) ^ 0) WHEN 0 THEN 1 ELSE 0 END)

    FROM

    (

        SELECT CONVERT(VARBINARY, SUM(region\_size\_in\_bytes)) [Size],

            region\_allocation\_base\_address [Base]

            FROM sys.dm\_os\_virtual\_address\_dump

        WHERE region\_allocation\_base\_address <> 0

        GROUP BY region\_allocation\_base\_address

        UNION

        SELECT

            CONVERT(VARBINARY, region\_size\_in\_bytes) [Size],

            region\_allocation\_base\_address [Base]

        FROM sys.dm\_os\_virtual\_address\_dump

        WHERE region\_allocation\_base\_address = 0x0 ) AS VAS\_Dump

        GROUP BY Size

    )

SELECT

    SUM(CONVERT(BIGINT, Size) \* Free) / 1024 AS [Total avail mem, KB],

    CAST(MAX(Size) AS BIGINT) / 1024 AS [Max free size, KB]

FROM VAS\_Summary WHERE FREE <> 0

**How can I tell if I need to allocate more memory to MemToLeave**

* If the above T-SQL script shows that the amount of available memory is small for the requirements of your platform. (For example, your application/development team may be able to advise on the expected memory requirements of the managed code components that have been developed).
* A more pressing indicator takes the form of a variety of warning/error messages raised by either SQL Server or the specific managed code component.

**How do I allocate more resources to MemToLeave?**

In the event that you determine that you require to increase the size of the MemToLeave area, this can be done by defining the -g startup option for the SQL Server Service.

The value that you assign to the parameter will determine the size of the MemToLeave area of memory.

If you are unsure of how much additional memory to allocate to MemToLeave, then it is best to increase the volume assigned in smaller increments for example 64MB (although dependent on your specific platform), until your original indicators are no longer being raised i.e. your application domain unload messages are no longer appearing.

**Note** :

It is important to note that increasing the size of the MemToLeave area will reduce the amount of available memory to the SQL Server Buffer Pool. For this reason it is wise to iterate your tweaking of this parameter in small increments.

**Buffer Pool**holds the majority of the SQL memory and is divided into multiple section,

* Data Cache/Buffer Cache - Holds the data pages uses in SQL operations.
* Procedure Cache/Query Cache - The name Procedure cache may be misleading but it stores all the execution plans of queries and store procedures.
* Log Cache  - stores data related to Transaction log pages.
* System Level Data Structure - Instance level data
* Connection context - stores connection information
* Stack Space - Windows allocated stack space

**Virtual Address Space:-**

The virtual address space(abbreviated **VAS**) for a process is the set of virtual memory addresses that it can use. The address space for each process is private and cannot be accessed by other processes unless it is shared.

A virtual address does not represent the actual physical location of an object in memory; instead, the system maintains a *page table* for each process, which is an internal data structure used to translate virtual addresses into their corresponding physical addresses. Each time a thread references an address, the system translates the virtual address to a physical address.

**Memory Allocation:-**

SQL server memory allocation has various scenario's based on the server Architecture(32bit or 64bit) and configurations (AWE,/3GB). Let us look at various scenario's

**Note :**

***1. 32 Bit with No AWE or /3G Option enabled:-***

All 32-bit applications(SQL Server) have a maximum of only 2^32 or 4-gigabyte (GB) of process address space. Microsoft Windows operating systems provide SQL Server with access to 2 GB of process address space, specifically known as user mode virtual address space(VAS). All threads owned by an SQL Server share the same user mode virtual address space. The remaining 2 GB are reserved for the operating system (also known as kernel mode address space). As we discussed already SQL Server memory is in turn divided into MemToLeave and Buffer Pool. On SQL Server Start up SQLOS reserves MemToLeave area which is usually a constant around 384 MB or *((MaxWorkerThreads \* StackSize) + DefautReservationSize)*and rest of the available space in the user mode VAS is taken by the Buffer Pool.

***2. 32 Bit with /3GB switch enabled:-***

All operating system editions starting with Windows 2000 Server, including Windows Server 2003, have a boot.ini switch that can provide applications with access to 3 GB of process address space, limiting the kernel mode address space to 1 GB, so in contrast to the above condition 4GB memory is split as 3 GB user mode VAS and 1 GB Kernel Mode VAS when /3GB swith is enabled. On SQL Server Start up SQLOS reserves MemToLeave area which is usually a constant around 384 MB or *((MaxWorkerThreads \* StackSize) + DefautReservationSize)*and rest of the available space (3072 MB - 384 MB) in the user mode VAS is taken by the Buffer Pool.

***3. 32 Bit with AWE option enabled:-***

Address Windowing Extensions (AWE) extend the capabilities of 32-bit SQL Server by allowing access to as much physical memory as the operating system supports with a max up to 64GB.  when AWE option is enabled on a SQL Server we have to keep in mind that only SQL server buffer cache(data cache) can utilize memory more than 4GB hence the MemToLeave remains the same around 384 MB or*((MaxWorkerThreads \* StackSize) + DefautReservationSize) .*The maximum size of the buffer pool is the upper limit of the Physical RAM on the SQL Server or the max server memory setting, whichever is smaller.

***4. 64 Bit Architecture:-***

64 bit applications can support Address space upto 2^64 or some xx Trillion GB which at present scenario much larger than the memory capacity, so the user mode VAS and Kernel mode VAS are capped to 8TB of space each. On start up VAS reservation does not occur in 64 bit system due to the size available. Also AWE Option is ignored in 64 bit Architecture. SQL Server allocates memory dynamically based on the requirement to the upper limit of the Physical RAM on the SQL Server or the max server memory setting, whichever is smaller.

**What is Lock Pages in Memory?**

Under default configuration, all SQL Server's memory allocations are made in the user mode VAS. Any memory allocated in User VAS is pageable, meaning that the Windows OS can force this memory to be paged to disk, in response to memory pressure. periodically paging out a large amount of memory from SQL Server’s working set will have bad effect on SQL Server performance. Adding SQL server service account to ***Lock Pages in Memory*** will prevent buffer cache/data cache portion from paging to disk during windows OS low memory notification. In 32 -bit architecture when AWE is enabled we need to Lock Page in memory to exclusively lock the AWE memory from paging. In 64-bit architecture there is a debate still in SQL Server community whether its good to have lock page in memory to exclusively lock SQL server allocated memory. I would advise we can Lock pages in Memory in 64 bit architecture but we need to ensure we give enough memory to OS too by setting  '**max server memory**' setting to SQL Server.

**1.*To find total size of the physical memory on the Server:-***

select total\_physical\_memory\_kb/1024/1024 AS [total\_physical\_memory\_GB] from sys.dm\_os\_sys\_memory

total\_physical\_memory\_GB

------------------------

63

(1 row(s) affected)

***2. To Find Total Memory allocated to SQL Server :-***

select "Max\_Memory" =

CASE

WHEN a.value = 2147483647 THEN (select total\_physical\_memory\_kb/1024/1024 AS [total\_physical\_memory\_GB] fromsys.dm\_os\_sys\_memory)

ELSE a.value

END

from sys.configurations a where name = 'max server memory (MB)'

Max\_Memory

------------------

47344

(1 row(s) affected)

**3. *To Find the total memory allocated to Buffer Pool:-***

*a. 64- bit SQL:-*

select SUM(awe\_allocated\_kb)/1024/1024 from sys.dm\_os\_memory\_clerks where type = 'MEMORYCLERK\_SQLBUFFERPOOL'

--------------------

46

(1 row(s) affected)

*b. 32 Bit Awe Enable:-* To find if Awe is enable run select value from sys.configurations where name = 'awe enabled', The Value 1 indicates AWE is enabled.

select SUM(awe\_allocated\_kb)/1024/1024 from sys.dm\_os\_memory\_clerks where type = 'MEMORYCLERK\_SQLBUFFERPOOL'

--------------------

21

(1 row(s) affected)

*c. 32 bit AWE not allocated:-*

I could not find exact query but the below query will give you the almost the the correct value,

select SUM(virtual\_memory\_reserved\_kb)/1024 from sys.dm\_os\_memory\_clerks where type = 'MEMORYCLERK\_SQLBUFFERPOOL'

***4. To Find Total Memory assigned to the VAS(User Kernel mode):-***

*The below Query is taken from MSDN white paper "Troubleshooting performance problems in sql server 2008" written by Sunil Agarwal, Boris Baryshnikov, Keith Elmore, Juergen Thomas, Kun Cheng and  Burzin Patel (*[*http://msdn.microsoft.com/en-us/library/dd672789(v=sql.100).aspx*](https://usmail.cognizant.com/OWA/redir.aspx?C=Ersk920TJk-oqHdzmHTHNHC4OldAVs8IA5_85G-7m_9RX_U66nMRK6aVVu0qLRnmpAwL0IY_MMo.&URL=http%3a%2f%2fmsdn.microsoft.com%2fen-us%2flibrary%2fdd672789(v%3dsql.100).aspx)*) and Christian Bolton's Blog*[*http://sqlblogcasts.com/blogs/christian/archive/2008/01/07/sql-server-memtoleave-vas-and-64-bit.aspx*](https://usmail.cognizant.com/OWA/redir.aspx?C=Ersk920TJk-oqHdzmHTHNHC4OldAVs8IA5_85G-7m_9RX_U66nMRK6aVVu0qLRnmpAwL0IY_MMo.&URL=http%3a%2f%2fsqlblogcasts.com%2fblogs%2fchristian%2farchive%2f2008%2f01%2f07%2fsql-server-memtoleave-vas-and-64-bit.aspx)

WITH VASummary(Size,Reserved,Free) AS

(SELECT

Size = VaDump.Size,

Reserved = SUM(CASE(CONVERT(INT, VaDump.Base)^0)

WHEN 0 THEN 0 ELSE 1 END),

Free = SUM(CASE(CONVERT(INT, VaDump.Base)^0)

WHEN 0 THEN 1 ELSE 0 END)

FROM

(

SELECT CONVERT(VARBINARY, SUM(region\_size\_in\_bytes))

AS Size, region\_allocation\_base\_address AS Base

FROM sys.dm\_os\_virtual\_address\_dump

WHERE region\_allocation\_base\_address <> 0x0

GROUP BY region\_allocation\_base\_address

UNION

SELECT CONVERT(VARBINARY, region\_size\_in\_bytes), region\_allocation\_base\_address

FROM sys.dm\_os\_virtual\_address\_dump

WHERE region\_allocation\_base\_address = 0x0

)

AS VaDump

GROUP BY Size)

SELECT SUM(CONVERT(BIGINT,Size)\*Free)/1024/1024 AS [Total avail mem, MB] ,CAST(MAX(Size) AS BIGINT)/1024/1024 AS [Max free size, MB]

FROM VASummary

WHERE Free <> 0

Result:-

Total avail mem, MB Max free size, MB

-------------------- --------------------

288                         84

(1 row(s) affected)

We have seen how to find the total memory allocated to the in each section, now we can discuss how to find the amount of memory used in each parts of SQL memory.  
  
Before SQL 2005 we used DBCC MEMORYSTATUS output to find the memory components used space, post SQL 2005 we can get all the info from OS DMV sys.dm\_os\_memory\_clerks.

As we discussed in PART 1 MultiPage allocation happens in the VAS Area and Single page allocation comes from Buffer Pool, we can calculate the space used in Bpool and outside Bpool with the below query,

*The below Query is taken from MSDN white paper "Troubleshooting performance problems in sql server 2008" written by Sunil Agarwal, Boris Baryshnikov, Keith Elmore, Juergen Thomas, Kun Cheng and  Burzin Patel (*[*http://msdn.microsoft.com/en-us/library/dd672789(v=sql.100).aspx*](https://usmail.cognizant.com/OWA/redir.aspx?C=Ersk920TJk-oqHdzmHTHNHC4OldAVs8IA5_85G-7m_9RX_U66nMRK6aVVu0qLRnmpAwL0IY_MMo.&URL=http%3a%2f%2fmsdn.microsoft.com%2fen-us%2flibrary%2fdd672789(v%3dsql.100).aspx)*)*

-- amount of memory consumed by components outside the bBuffer pool

-- note that we exclude single\_pages\_kb as they come from BPool

select

    sum(multi\_pages\_kb

        + virtual\_memory\_committed\_kb

        + shared\_memory\_committed\_kb) as [Overall used w/o BPool, Kb]

from

    sys.dm\_os\_memory\_clerks

where

    type <> 'MEMORYCLERK\_SQLBUFFERPOOL'

-- amount of memory consumed by BPool

-- note that currenlty only BPool uses AWE

select

    sum(multi\_pages\_kb

        + virtual\_memory\_committed\_kb

        + shared\_memory\_committed\_kb

        + awe\_allocated\_kb) as [Used by BPool with AWE, Kb]

from

    sys.dm\_os\_memory\_clerks

where

    type = 'MEMORYCLERK\_SQLBUFFERPOOL'

We can also use Ring buffers to find memory used, we can discuss this in a separate article as i am planning write a blog on RING BUFFERS and extended events

**Buffer Counts:**

The target number of buffers represents the number of 8KB pages SQL Server can commit without causing paging. The value of **target buffers cannot exceed "max server memory" option. If this number diminishes, SQL Server might be experiencing external memory pressure**. If the target number of buffers is lower than the min server memory configuration setting, that indicates that SQL Server was never able to acquire the minimum amount of allocated memory because this memory isn't available through the operating system you'd need to check to see if other applications are consuming the memory on the same server. Recall that "min server memory" and "max server memory" control the size of the buffer pool.

**You should also examine the value of "stolen pages". If this value represents the large majority of target buffers then we might be experiencing internal memory issues**. You can query sys.dm\_os\_memory\_clerks view to figure out exactly which components are using the majority of memory. Note that you cannot alter the internal memory distribution, but you can tweak the amount of memory dedicated to each area of SQL Server memory.

**Buffer Counts Buffers**

**------------------------------ --------------------**

**Committed 2304**

**Target 12590**

**Hashed 1536**

**Stolen Potential 91620**

**External Reservation 0**

**Min Free 32**

**Visible 12590**

**Available Paging File 135354**

**Note:**

**If the *Target* value is low, but the *server Process: Private Bytes* is high, you might be facing internal SQL memory problems with components that use memory from outside of the buffer pool .** Such components include [linked servers](http://www.mssqltips.com/category.asp?catid=56), COM objects, extended stored procedures, [SQL CLR](http://www.mssqltips.com/category.asp?catid=48), etc.

You can compare the *Target* count against the max server memory values if it is set. Latter option limits the maximum memory consumption of the buffer pool. Therefore the *Target* value cannot exceed this value.  Also the low *Target* count can indicate problems: in case it is less than the min server memory setting, you should suspect external virtual memory pressure.

A high percentage (>75%) of *Stolen Pages* compared to *Target* can be a sign of internal memory pressure.

**Target > Max memory – external memory pressure**

**Target <min memory – internal memory pressure**

**stolen pages> Target – internal memory pressure**

**target < process: private bytes –internal memory pressure**

**Note:** Physical memory pressure could be imposed by OS, we called external or it could be imposed by the process itself we call it internal.

**External Memory pressure: RM and Buffer Pool**

Keep in mind that target commit can't be lower than configuration parameter specified through sp\_configure min server memory and can't be higher of max server memory. If new target commit is lower than currently committed buffers, BP starts shrinking until external physical memory pressure disappears. During this process BP tries to decommit or in case of AWE free physical memory back to OS. Remember that in SQL2000 BP didn't react to physical memory pressure when running in AWE mode

**Internal Memory Pressure: BP and Resource Monitor**

There are couple other ways for internal physical pressure to appear. It could be caused by dynamically changing max server memory. In addition it could raise when 75% of BP's pages are stolen using SQLOS's single page allocator interface. By triggering internal physical memory pressure BP reclaims its pages from caches and other components currently consuming them.

**32-Bit SQL Server memory architecture**

In the Win32 memory architecture, each process has a 4 GB address space. By default, 2 GB of that address space is accessible from user mode(Application like SQL Server) and the remaining 2 GB is accessible from kernel mode

So in 32 Bit windows architecture 2 GB of memory is maximum available for SQL Server.   
Note: When /3GB switch is enabled user mode address space becomes 3GB and kernel mode becomes 1 GB. When 32-Bit SQL Server is running on 64-Bit windows (WOW64) It gets 4GB of user address space .It can also leverage AWE on WOW64 mode and can use more than 4GB

SQL Server "User address space" is broken into two regions: MemToLeave and Buffer Pool

Size of MemToLeave (MTL) and Buffer Pool (BPool) is determined by SQL Server during start up as below.

**MTL (Memory to Leave)= (Stack size \* max worker threads) + Additional space (By default 256 MB and can be controlled by -g).   
Stack size =512 KB per thread for 32 Bit SQL Server   
I.e = (256 \*512 KB) + 256MB =384MB**

Additional space to load Dll’s= 256 MB from SQLServer2000. This space is used to store

1. COM objects

2. Extended stored procedure

3. Memory allocated by linked servers (loaded in process ) or other Dll’s loaded  in SQL Server proce

4. Memory allocated by SQL Server memory manger if the allocation size in greater than 8K and need’s contiguous memory (Multiple\_pages\_kb).

5. SQLCLR

**Note:** Additional space to load Dll’s can be modified using -g startup parameter.

on any machine with less than 4 processors the Maximum worker Thread’s is   
always 256 by default (unless we change the value using SP\_configure)

**SQL Server Buffer Pool is minimum of “Physical RAM “ or “user mode memory(2GB or 3GB) – MTL-  BUF structures”**

**BPool = Minimum (Physical memory, User address space – MTL) – BUF structures**

To ensure MemToLeave allocations are contiguous SQL Server reserves MTL first, then all the buffer pool regions and finally free MemtoLeave region.

What is in BPOOL?   
Data Pages/Index pages and Memory allocated by SQL Server memory  manager, which is accounted for  any of below memory clerk’s. If the memory

request is <= 8 KB

CACHESTORE\_PHDR   
CACHESTORE\_XMLDBTYPE   
CACHESTORE\_EVENTS   
MEMORYCLERK\_SQLSTORENG   
MEMORYCLERK\_XE   
CACHESTORE\_XPROC   
OBJECTSTORE\_SNI\_PACKET   
CACHESTORE\_BROKERRSB   
OBJECTSTORE\_SERVICE\_BROKER   
MEMORYCLERK\_SQLSERVICEBROKERTRANSPORT   
MEMORYCLERK\_XE\_BUFFER   
CACHESTORE\_XMLDBATTRIBUTE   
MEMORYCLERK\_SQLOPTIMIZER   
USERSTORE\_OBJPERM   
USERSTORE\_TOKENPERM   
CACHESTORE\_FULLTEXTSTOPLIST   
MEMORYCLERK\_SQLGENERAL   
MEMORYCLERK\_SQLHTTP   
CACHESTORE\_NOTIF   
CACHESTORE\_XMLDBELEMENT   
OBJECTSTORE\_LOCK\_MANAGER   
MEMORYCLERK\_SQLBUFFERPOOL   
MEMORYCLERK\_SQLSOAP   
MEMORYCLERK\_TRACE\_EVTNOTIF   
CACHESTORE\_CONVPRI   
MEMORYCLERK\_QSRANGEPREFETCH   
CACHESTORE\_BROKERREADONLY   
MEMORYCLERK\_SQLCLRASSEMBLY   
MEMORYCLERK\_SOSNODE   
CACHESTORE\_STACKFRAMES   
MEMORYCLERK\_SQLCONNECTIONPOOL   
MEMORYCLERK\_SQLSERVICEBROKER   
CACHESTORE\_OBJCP   
MEMORYCLERK\_SQLQUERYPLAN   
OBJECTSTORE\_SECAUDIT\_EVENT\_BUFFER   
OBJECTSTORE\_LBSS   
MEMORYCLERK\_FULLTEXT   
CACHESTORE\_TEMPTABLES   
CACHESTORE\_BROKERTBLACS   
MEMORYCLERK\_SQLXML   
USERSTORE\_SXC   
MEMORYCLERK\_BHF   
CACHESTORE\_SQLCP   
CACHESTORE\_SYSTEMROWSET   
USERSTORE\_SCHEMAMGR   
MEMORYCLERK\_SQLQUERYCOMPILE   
CACHESTORE\_BROKERTO   
CACHESTORE\_BROKERKEK   
MEMORYCLERK\_SNI   
MEMORYCLERK\_FULLTEXT\_SHMEM   
CACHESTORE\_BROKERUSERCERTLOOKUP   
USERSTORE\_DBMETADATA   
CACHESTORE\_VIEWDEFINITIONS   
MEMORYCLERK\_SQLQUERYEXEC   
CACHESTORE\_BROKERDSH   
MEMORYCLERK\_SQLSOAPSESSIONSTORE   
MEMORYCLERK\_SQLQERESERVATIONS   
MEMORYCLERK\_HOST   
MEMORYCLERK\_SQLCLR   
MEMORYCLERK\_SQLXP   
MEMORYCLERK\_SQLUTILITIES

**What is in MTL(Non-Bpool)?**

COM Objects   
SQL Server CLR   
Memory allocated by Linked Server OLEDB Providers and third party DLL’s loaded in SQL Server process   
Extended Stored Procedures:   
Network Packets   
Memory consumed by memory managers. If the memory request is > 8 KB and needs contiguous allocation.

**What is BUF structures?**SQL Server maintains a BUF structure for each page. This structure is used to track status information associated with each buffer, such as the buffer latch, a pointer to the actual 8 KB page, status bits that indicate whether the page is dirty, has an IO in progress etc.   
Note: When AWE is enabled BUF structure is maintained for entire RAM to adjust Max server memory with out restarting SQL Server.

**What is PAE?**   
PAE is the added ability of the 32 Bit processor to address more than 4 GB of physical memory. Enable /PAE   
in boot.ini to make operating system take advantage of physical memory over 4GB in system.

**What is AWE in SQL Server?**

When AWE is enabled, SQL Sever 32-Bit will be able to address more than 4 GB of physical memory using AWE allocator API’s.

**Note:**  In 32-Bit SQL Server Only data pages an index pages can be placed in AWE memory. So the memory available for other SQL Server memory objects is still limited to user address apace.

Memory allocated using  AWE allocator API’s are not part of Process working set ,hence can not be paged out and not visible in as private bytes or working set in task manger and perfmon.

Lock pages in memory privilege is required for startup account of SQL Server to use AWE allocator API’s.   
In 64-Bit systems sp\_configure ‘awe enabled’ does not have any functionality, If you have LPM privilege for Startup account of SQL Server AWE allocator API’s are used to allocate memory

**What is /3GB Switch?**

**/3GB** switch is used in the Boot.ini file.

When we enable /3GB. User address space of SQL Server or any application that uses IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE will increase to 3GB restricting kernel-mode address space to 1GB.

When the physical RAM in the system exceeds 16 GB and the /3GB switch is used, the operating system will ignore the additional RAM until the /3GB switch is removed. This is because of the increased size of the kernel required to support more Page table Entries

**64-Bit SQL Server memory architecture**

In the 64-Bit windows each process gets up to 8 TB of address space, Hence there was no need for SQL Server to leave certain amount of addressable memory for Non-Bpool allocations.

There are three types of memory model’s in 64-Bit SQL Server.

1. Conventional – Normal physical page size (4 / 8KB),memory can be paged, dynamic   
2. Locked – Normal physical page size (4 / 8KB), Bpool can not be paged, dynamic, Requires startup account of SQL Server to have "Lock pages in memory" privilege,Memory is allocated by  using Address Windowing Extensions (AWE) API’s   
3. Large – Large physical page size ( > = 2MB), Non-pageable, static, Memory is committed at startup,”Max server memory” is recommended, requires startup account of SQL Server to have "Lock pages in memory" privilege

Memory calculations in 64-Bit SQL Server are straight forward.   
SQL Server calculates the size of RAM during the startup and reserve it , minimum of (reserved space, “Max server memory”) is used as Bpool.

Similar to 32-Bit SQL Server, there will be memory allocations outside Bpool in 64-Bit SQL Server , which is called as Non-Bpool allocations.

**Who allocates memory outside Bpool?**

1. COM Objects   
2. SQL Server CLR   
3. Memory allocated by Linked Server OLEDB Providers and third party DLL’s loaded in SQL Server process   
4. Extended Stored Procedures:   
5. Network Packets   
6. Memory consumed by memory managers. If the memory request is greater than 8 KB and needs contiguous allocation.    
7. Backup   
8. Memory for threads (stack size is 2 MB in 64-BIT SQL )

Max server memory controls only the Bpool, it doesn’t control Non-Bpool allocations, this is the reason for SQL Server’s memory usage being greater than "Max Server memory".

**Key points:**

* When “Lock pages in memory is used” operating system can not page out Bpool, Non-Bpool allocations can still be paged.
* SP\_configure “awe enabled” option doesn’t have any use in 64-Bit SQL Server.
* “Max Server Memory” limits only Bpool, hence SQL Server memory usage will be greater than “Max server memory”
* If your operating system is windows2003 (Windows2008 is your call) make sure you cap the SQL Server MAX Server Memory after considering the memory required by other applications, Operating system, Drivers , SQL Server Non- Bpool allocations etc

**SQL Server performance degraded in 32-Bit SQL Server after adding additional RAM**

Do you know that adding additional RAM can affect the performance of SQL Server Sometimes?

I am not going to write how Optimizer can some times choose suboptimal plans when we have large amount of memory on the system but We will see how the memory which can be used by other memory clerks (aks: stolen memory) can shrink when we have large physical memory and AWE enabled.

If you notice  performance of 32-Bit SQL Server degraded after you added additional RAM or if you see SQL Server memory errors like ones below after adding RAM then it could be because of Large BUF structures which reduced the size of Bpool.

**Extract from** [**SQL Server memory design**](http://mssqlwiki.com/sqlwiki/sql-performance/basics-of-sql-server-memory-architecture/)

{

SQL Server "User address space" is broken into two regions: MemToLeave and Buffer Pool

Size of MemToLeave (MTL) and Buffer Pool (BPool) is determined by SQL Server during start up as below.

MTL (Memory to Leave)= (Stack size \* max worker threads) + Additional space to load Dll’s.

Stack size =512 KB per thread for 32 Bit SQL Server (904K under WOW)

I.e. = (256 \*512 KB) + 256MB =384MB

Additional space to load Dll’s= 256 MB from SQLServer2000. This space is used to store COM objects, Extended stored procedure, Linked server in SQL Server process

Note: Additional space to load Dll’s can be modified using -g startup parameter.

on any machine with less than 4 processors the Maximum worker Thread’s is always 256 by default (unless we change the value using SP\_configure)

**SQL Server Buffer Pool is minimum of “Physical RAM “ or “user mode memory(2GB or 3GB) – MTL-  BUF structures”**

**BPool = Minimum (Physical memory, User address space – MTL) – BUF structures**

}

When AWE is enabled in 32-Bit SQL Server M\_pbuf (part of BUF structures) which is mentioned earlier is calculated and allocated for entire physical memory on the system . Regardless of “MAX Server Memory”   This is to adjust Max server memory without restarting SQL Server.

SQL Server requires 8MB to create M\_pbuf for every 1GB of RAM available on the server.

Machine with 64 GB RAM can consume 64 (RAM) \*8MB (M\_pbuf for each GB) =512 MB just for the BUF array alone.

So the amount of BPOOL available for SQL Server is adversely affected.

Going back to the previous formula for BPOOL. Size of Bpool for 32-Bit SQL Server with AWE enabled and 64 GB of RAM would be.

BPool = Minimum (Physical memory, User address space – MTL) – BUF structures

BPool= Minimum (64GB, (2GB-384MB)) – **BUF structures (512+ MB)**

**Bpool would approximately become 1GB.  Since size BPOOL become very small we might end up with memory errors.**

**Note:  In 32-Bit SQL Server Only data pages an index pages can be placed in AWE memory. So the memory available for other SQL Server memory objects is still limited to BPOOL and MTL.**

**How to resolve this issue?**

Remove few GB of RAM from server  if you can convince your management that removing RAM will improve performance.

(Or)

There is a startup trace flag **TF 836** which you can use to indicate that BUF’s need to be allocated only for the configured max server memory setting. Enable this Trace Flag (836) and Reduce the “MAX Server Memory” of SQL Server.

(Or)

Enable /3GB. This will increase the Size of SQL Server BPOOL by 1GB providing relief to SQL Server BPOOL pressure.

Note: When the physical RAM in the system exceeds 16 GB and the /3GB switch is used, the operating system will ignore the additional RAM until the /3GB switch is removed.

**Understanding SQL Server Memory Usage**

Well first recall that SQL Server has two types of memory usage, VAS (aka MemToLeave on 32bit servers) and the Buffer Pool or BPool.  When SQL Server starts up the maximum size of the BPool is calculated based on the servers configuration.  For 32bit servers without AWE enabled, the VAS reservation is calculated and reserved first, then the remaining user mode VAS is compared to the amount of physical memory available on the server.  The smaller of the two values is then compared to the max server memory configuration option and the smallest number becomes the maximum size for the BPool.  For example, a 32bit server with 4GB RAM, the /3GB boot.ini switch, and a max server memory configuration of 2048MB the maximum size of the BPool would be 2048MB:

3072MB (3GB) user VAS - 384MB = 2688MB available VAS   
4096MB (4GB) Physical Memory > 2688MB available VAS   
2688MB available VAS > 2048MB max server memory configuration

However, when AWE is enabled the BPool maximum size isn’t restricted by the available user mode VAS.  Instead the maximum size of the BPool is based on the physical memory on the server or the max server memory configuration setting, whichever is smaller.  For example, a 32bit server with 8GB RAM, the /PAE boot.ini switch, AWE enabled, and a max server memory configuration of 6144MB the maximum size of the BPool would be 6144MB:

8192MB (8GB) Physical Memory > 6144MB max server memory configuration

For 64 bit servers, there is no need to use AWE to allocate memory above 3GB for SQL Server since the user mode VAS is 8TB there is always ample VAS to utilize all of the physical memory available on the server.  However, the AWE mechanism for allocating memory is still useful and available on 64 bit servers.  It is used whenever the  the Lock Pages in Memory security right has been granted to the SQL Server Service Account and can improve the stability and performance of a 64 bit system

For 64 bit servers, the maximum size of the BPool is the size of physical memory or the max server memory configuration, whichever is smaller.  For example, a 64 bit server with 16GB RAM, and a default max server memory configuration, the maximum size of the Bpool would be 16GB.

There are couple of things you need be aware of:

A. You only need to consider setting max server memory if you expect load on the machine to be memory bound

B. ***You need to look at the machine during heavy load to come up with appropriate max server memory setting so that you keep available memory in recommended range***

C. The more RAM you have on the machine (32GB+)  the more important to have max server memory setting on.

D. ***My recommendations are "personal" recomendations, you will need to tune your box to find what is the best combination for you.***

E. Max worker threads setting does affect how much memory server will require under heavy concurrent load. Consider on x64 platform each SQL Server thread can consume 2MB of physical memory. So if you configure SQL Server for 2000 threads, under heavy load in addition to max server memory it might require additional 4GB of physical memory (2MB \* 2000), it is even higher on IA64, i.e 8GB (4MB \* 2000)

F. Max server memory setting only controls size of the Buffer Pool and hence doesn't affect allocations going through Multi Page Allocator (MPA). If your load does require MPA you will have to take that into account as well. You can monitor your MPA usage by using sys.dm\_os\_memory\_clerks DMV.

G. Memory allocations requested by external components such as xps, COM (sp\_OACreate), and others are not controlled by max server memory setting

**Note : - in sql server 64 bit version AWE enable is not require but 32 bit version AWE should enable to utilize more than 4GB of SQL Server but in 64 bit version SQL server max memory setting should be enable to avoid page locks.**

**Lock pages in memory**

Why SQL Server is using so LESS memory

Today I want to talk about a phenomenon regarding memory management in SQL Server. A few weeks ago I had a consulting customer where we made some SQL Server performance improvements. As a side note the customer asked me, why SQL Server isn't using as much memory as possible on their production system. Their database was about 500 GB, and when we looked into the Task Manager of Windows, we saw that SQL Server was just consuming a few hundred MBs:

In the first step this behavior of SQL Server was a little bit amazing, but the answer was found very fast, when we looked into the details of their production configuration. The problem was that the customer granted the Locked Pages in Memory privilege to the service account under which SQL Server was running. You can check if you have granted this privilege when you look into the current SQL Server Error Log. If you have granted that privilege, you can see following informational message during the startup of SQL Server:

In the first step of this posting I want to explain why and when you should use that privilege for SQL Server, and in the second step you will see, how SQL Server uses that privilege and why Task Manager is lying to you regarding the memory consumption.

Every time when the Windows OS gets into memory pressure, the Windows OS raises a so-called Memory Resource Notification Event to all processes that are currently running on the box (see <http://msdn.microsoft.com/en-us/library/windows/desktop/aa366541(v=vs.85).aspx> for the corresponding API function). SQL Server subscribes to this event and processes it internally through a component called the Resource Monitor. Additional information about the Resource Monitor component is reported through the Ring Buffer type RING\_BUFFER\_RESOURCE\_MONITOR that is stored inside the DMV sys.dm\_os\_ring\_buffers. When SQL Server receives the Memory Resource Notification Event, SQL Server must trim its internal caches, like the Buffer Pool, or the Plan Cache to get out of memory pressure. But there are some scenarios in which SQL Server reacts to slow to memory pressure. In that case the Windows OS will do a so-called Hard Working Set Trim, and pages the process (in our case SQL Server) out to the page file. The Windows OS just want to survive and get out of its memory pressure. Imagine that – the Buffer Pool, the Plan Cache – In-Memory buffers – are paged out to the page file! Accessing the Buffer Pool – our in-memory cache - means reading pages from the physical disk! You can imagine that the performance of SQL Server will decrease massively…

Another case when a Hard Working Set Trim can occur is, when you have device drivers that have bugs, or when the Windows OS has bugs that leads to Hard Working Set Trims. Under Windows Server 2003 there were several bugs reported that caused Hard Working Set Trims, like:

Copying large files from the SQL Server box

Establishing a Remote Desktop Connection to your SQL Server box

But how you can find out if your performance problems occur because of Hard Working Set Trims? When you are running a version of SQL Server prior to SQL Server 2005 SP2, you have to monitor the overall performance of your SQL Server instance, there are no messages or indications inside SQL Server that can tell you, if a Hard Working Set Trim degraded the performance of SQL Server. Beginning with SQL Server 2005 SP2 Microsoft has added an error message to the SQL Server Log, as soon as a Hard Working Set Trim occurred, like:

" A significant part of sql server process memory has been paged out. This may result in a performance degradation. Duration: 0 seconds. Working set (KB): 1086400, committed (KB): 2160928, memory utilization: 50%"

In such cases you can enable the Locked Pages in Memory privilege to get rid of this issue. In this case the Windows OS is not allowed to page out pages that SQL Server allocated for the Buffer Pool. Yes, you read correct: Locked Pages in Memory is only used for Buffer Pool allocations inside SQL Server!

But on the other hand you also have to investigate WHY a Hard Working Set Trim occurred on your system. A Hard Working Set Trim is just the result of another problem that you have on your Windows box. So you should find the root cause, and eliminate it, instead of using the shortcut (enabling Locked Pages in Memory) infinitely. I know a lot of DBAs who are using Locked Pages in Memory by default, and there are several pro and cons for this which I don't want to discuss here again. Glen Berry has a great article which describes more of those pros and cons (see <http://sqlserverperformance.wordpress.com/2011/02/14/sql-server-and-the-lock-pages-in-memory-right-in-windows-server>). One thing that I want to mention here again is the fact that you should limit the memory that SQL Server can allocate for the Buffer Pool when you use Locked Pages in Memory. You can limit the memory through the Max Memory Setting of your SQL Server instance.

By now you know when and how you can enable Locked Pages in Memory for your SQL Server instance. But why the heck Task Manager is reporting a wrong Working Set size when SQL Server is using that privilege. For this explanation we have to dig a little bit deeper into the Win32API. By default (without Locked Pages in Memory) SQL Server allocates memory for the Buffer Pool through the VirtualAlloc function of the Win32API (see <http://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx>). But when the SQL Server service account has the Locked Pages in Memory privilege, SQL Server internally uses the AllocateUserPhysicalPages Win32API function (see <http://msdn.microsoft.com/en-us/library/windows/desktop/aa366528(v=vs.85).aspx>) to do Buffer Pool allocations. This function can be only called by a process, when the process has the SeLockMemoryPrivilege – in other words the Locked Pages in Memory privilege which is the user-friendly name. The Win32API function AllocateUserPhysicalPages is part of the Address Windowing Extensions API (AWE). Therefore those memory allocations can't be paged out by the Windows OS, because AWE memory regions are not page able by design. Those memory regions are just locked in memory, therefore the name of this privilege. When you have Locked Pages in Memory for your SQL Server service account enabled, it also means that you are using AWE indirectly – funny isn't it? As you can see from this explanation, Locked Pages in Memory is rather a Windows OS concept than a SQL Server concept. It has nothing to do directly with SQL Server. SQL Server is just a consumer of it.

With this basic knowledge in your hand, it is now very easy to explain why Task Manager doesn't show the correct Working Set of the SQL Server process: Task Manager is not reporting those memory allocations that are done through the Win32API function AllocateUserPhysicalPages – that's all about this phenomenon.

But how can you now find out, how much memory SQL Server is really using? There are several possibilities. Inside SQL Server you can use the DMV sys.dm\_os\_process\_memory and the column physical\_memory\_in\_use\_kb. This returns you the physical memory that SQL Server is currently using including AWE memory allocations. If you want to have a more detailed breakdown of the memory consumption of SQL Server, you can use the DMV sys.dm\_os\_memory\_clerks. A memory clerk is a component inside SQL Server, which tracks memory allocations for a specific component inside SQL Server. SQL Server gives you a memory clerks for each major component for each available NUMA node in your system. The column awe\_allocated\_kb shows you the AWE memory allocations in kb that were allocated by SQL Server through the Win32API function AllocateUserPhysicalPages.

Update:

Aaron Bertrand (see <http://sqlblog.com/blogs/aaron_bertrand>) mentioned that it would also make sense to reference a Performance Monitor counter to track the memory consumption of SQL Server. So here it is:

Outside of SQL Server you can use the Performance Monitor counter SQLServer:Memory Manager/Total Server Memory (KB) to track how large the Buffer Pool currently is.

As you have seen with this posting, memory management inside SQL Server is a really complicated topic, and we just have touched the tip of the iceberg. So don't trust Task Manager blindly regarding the memory consumption of SQL Server, you really have to know how SQL Server is configured to get the correct picture.

**SQL Server and the “Lock pages in memory” Right in Windows Server**

**Another Windows setting you might want to enable is “Lock pages in memory”. There is some controversy within the SQL Server community and within Microsoft about whether and when you should enable this setting. Before you decide whether you want to do this, let me give you some background. When SQL Server 2005 was first released (in early 2006), and was installed on systems running x64 Windows Server 2003 or x64 Windows Server 2003 R2, it quickly became pretty common to enable “Lock pages in memory”, to try to prevent the operating system from periodically paging out a large amount of memory from SQL Server’s working set, which would cause a very noticeable bad effect on SQL Server performance.**

**The reason that this usually happened is because the operating system would run low on available memory (typically due to buggy device drivers), and then the operating system would page out a large portion of the working set of the SQL Server process in order to free up memory for the operating system. This made the operating system happier, but had a terrible effect on SQL Server, since it would have to heavily access the storage subsystem to read back in the data that had recently been the buffer pool.**

**SQL Server 2005 and above is designed to perform dynamic memory management based on the memory requirements of the current load on the system. On a Windows Server 2003 or later operating system, SQL Server can use the memory notifications from the QueryMemoryResourceNotification Windows API. This is meant to keep the operating system from paging out the working set of the SQL Server process, and it helps keep more database pages available in memory to reduce your physical I/O needs.**

**The problem was that SQL Server 2005 did not always react quickly enough to a low memory notification from the operating system, so the operating system would take matters into its own hands, and force SQL Server to release a large portion of its working set. Using “Lock pages in memory” prevents this from happening, at the cost of masking the root cause of the issue.**

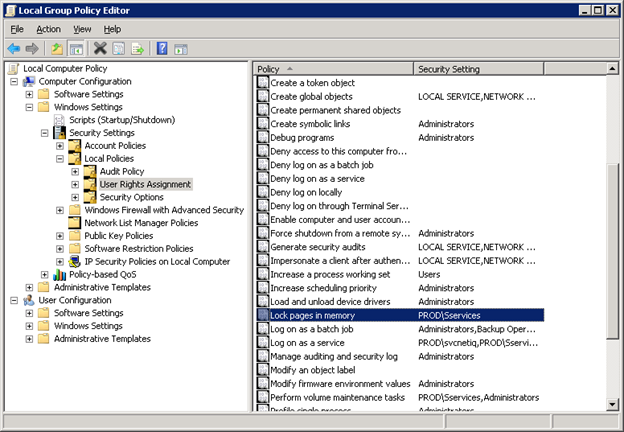
**Initially, this setting would only be honored by SQL Server 2005 Enterprise Edition. Microsoft’s official stance was that if you encountered this issue, you should open a support incident with Microsoft CSS to try to find the root cause of the problem. Their opinion was that it was better to find the cause of the problem instead of using “Lock pages in memory” as a Band-Aid to cover up the problem.**

**Many production DBAs disagreed with this idea. Once you had been affected by this issue a couple of times, and had to explain to your management team or customers why your application suddenly started timing out for no apparent reason, you were usually quite ready to enable” Lock pages in memory”. Later, after a lot of pressure from the SQL Server MVP community, Microsoft finally made “Lock pages in memory” available in SQL Server Standard Edition (in later builds of SQL Server 2005, 2008, and 2008 R2), by using a startup trace flag of 845.**

**Microsoft has published conflicting information about this issue over the years. The current official stance is that you should not have to use “Lock pages in memory” with Windows Server 2008 and newer, because of improvements in memory management and improved device drivers. Off the record, I have heard several Microsoft employees concede that it may still be necessary to use this setting in some situations. I don’t want people to race out and immediately enable “Lock pages in memory”. Instead, you should consider your specific situation. and use your own judgment before you decide what to do. If I was running SQL Server 2005 on top of Windows Server 2003, I would be more tempted to use this setting. If I was running SQL Server 2008 R2 on top of Windows Server 2008 R2, I would be much less tempted to use this setting. You also have to consider whether you have run into the problem that LPIM was designed to alleviate. Remember the old saying “Don’t fix it if it ain’t broke”…**

**You have to grant this right to the Windows account that the SQL Server Service is using. This would normally be a Windows domain account. You can do this by using the Local Group Policy Editor on the machine where SQL Server will be running. You can just type GPEDIT.MSC in a Run window, which will bring up the Local Group Policy Editor shown in Figure 1. Then you go to Computer Configuration, Windows Settings, Security Settings, Local Policies, User Rights Assignment.**

**Next, in the right hand portion of the dialog window, you simply right-click on “Lock pages in memory” and select Properties, and click on the Add User or Group button. Then you need to add the name of SQL Server Service account, and click Ok. You have to restart the SQL Server Service to have this setting take effect.**



**Figure 1: Using the Local Group Policy Editor to grant the “Lock pages in memory” right to the SQL Server Service account**

**If you do enable “Lock Pages in Memory”, it is very important that you also set an explicit MaxServerMemory setting for your SQL Server instance, which I discussed in more detail here. This will control how much memory can be used by the SQL Server Buffer Pool, allowing a cushion for the operating system and anything else (including other SQL Server components) that may be running on the instance or machine.**

**You can confirm that the user right is used by the instance of SQL Server by making sure that the following message is written in the SQL Server Error Log at startup: “Using locked pages for buffer pool”.**

Suggested Max Memory Settings for SQL Server 2005/2008

**It is pretty important to make sure you set the Max Server memory setting for SQL Server 2005/2008 to something besides the default setting (which allows SQL Server to use as much memory as it wants, subject to signals from the operating system that it is under memory pressure). This is especially important with larger, busier systems that may be under memory pressure.**

**This setting controls how much memory can be used by the SQL Server Buffer Pool. If you don’t set an upper limit for this value, other parts of SQL Server, and the operating system can be starved for memory, which can cause instability and performance problems. It is even more important to set this correctly if you have “Lock Pages in Memory” enabled for the SQL Server service account (which I always do for x64 systems with more than 4GB of memory).**

**These settings are for x64, on a dedicated database server, only running the DB engine, (which is the ideal situation).**

**Physical RAM MaxServerMem Setting**

**2GB 1500**

**4GB 3200**

**6GB 4800**

**8GB 6400**

**12GB 10000**

**16GB 13500**

**24GB 21500**

**32GB 29000**

**48GB 44000**

**64GB 60000**

**72GB 68000**

**96GB 92000**

**128GB 124000**

**If you are running other SQL Server components, such as SSIS or Full Text Search, you will want to allocate less memory for the SQL Server Buffer Pool. You also want to pay close attention to how much memory is still available in Task Manager. This is how much RAM should be available in Task Manager while you are under load (on Windows Server 2003):**

**Physical RAM Target Avail RAM in Task Manager**

**< 4GB 512MB – 1GB**

**4-32GB 1GB – 2GB**

**32-128GB 2GB – 4GB**

**> 128GB > 4GB**

**You can use T-SQL to set your MaxServerMemory setting. The sample below sets it to 3500, which is the equivalent of 3.5GB. This setting is dynamic in SQL Server 2005/2008, which means that you can change it and it goes into effect immediately, without restarting SQL Server.**

**– Turn on advanced options**

**EXEC sp\_configure‘Show Advanced Options’,1;**

**GO**

**RECONFIGURE;**

**GO**

**– Set max server memory = 3500MB for the server**

**EXEC sp\_configure‘max server memory (MB)’,3500;**

**GO**

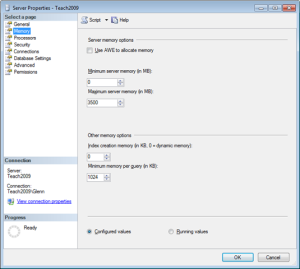
**RECONFIGURE;**

**GO**

**-- See what the current values are**

**EXEC sp\_configure;**

**You can also change this setting in the SSMS GUI, as you see below:**



**Finally, I have learned that it is a good idea to temporarily adjust your MaxServerMemory setting downward by a few GB if you know you will be doing a large file copy on your database server (such as copying a large database backup file).**

**SQL Server Service Startup Options**

**Default startup options Description**

**-d master\_file\_path**

**The fully qualified path for the master database file (typically, C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Data\master.mdf). If you do not provide this option, the existing registry parameters are used.**

**-e error\_log\_path**

**The fully qualified path for the error log file (typically, C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\LOG\ERRORLOG). If you do not provide this option, the existing registry parameters are used.**

**-l master\_log\_path**

**The fully qualified path for the master database log file (typically C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Data\mastlog.ldf).**

**Other startup options Description**

**-c**

**Shortens startup time when starting SQL Server from the command prompt. Typically, the SQL Server Database Engine starts as a service by calling the Service Control Manager. Because the SQL Server Database Engine does not start as a service when starting from the command prompt, use -c to skip this step.**

**-f**

**Starts an instance of SQL Server with minimal configuration. This is useful if the setting of a configuration value (for example, over-committing memory) has prevented the server from starting.**

**-g memory\_to\_reserve**

**Specifies an integer number of megabytes (MB) of memory that SQL Server will leave available for memory allocations within the SQL Server process, but outside the SQL Server memory pool. The memory outside of the memory pool is the area used by SQL Server for loading items such as extended procedure .dll files, the OLE DB providers referenced by distributed queries, and automation objects referenced in Transact-SQL statements. The default is 256 MB.**

**Use of this option might help tune memory allocation, but only when physical memory exceeds the configured limit set by the operating system on virtual memory available to applications. Use of this option might be appropriate in large memory configurations in which the memory usage requirements of SQL Server are atypical and the virtual address space of the SQL Server process is totally in use. Incorrect use of this option can lead to conditions under which an instance of SQL Server may not start or may encounter run-time errors.**

**Use the default for the -g parameter unless you see any of the following warnings in the SQL Server error log:**

**"Failed Virtual Allocate Bytes: FAIL\_VIRTUAL\_RESERVE <size>"**

**"Failed Virtual Allocate Bytes: FAIL\_VIRTUAL\_COMMIT <size>"**

**These messages might indicate that SQL Server is trying to free parts of the SQL Server memory pool in order to find space for items such as extended stored procedure .dll files or automation objects. In this case, consider increasing the amount of memory reserved by the -g switch.**

**Using a value lower than the default will increase the amount of memory available to the buffer pool and thread stacks; this may, in turn, provide some performance benefit to memory-intensive workloads in systems that do not use many extended stored procedures, distributed queries, or automation objects.**

**-h**

**Reserves virtual address space for Hot Add memory metadata when AWE is enabled with 32-bit SQL Server 2005. Required for Hot-Add memory with 32-bit AWE, but consumes about 500 MB of virtual address space and makes memory tuning more difficult. Not required for 64-bit SQL Server. Hot Add Memory is only available for Windows Server 2003, Enterprise and Datacenter editions. It also requires special hardware support from the hardware vendor.**

**-m**

**Starts an instance of SQL Server in single-user mode. When you start an instance of SQL Server in single-user mode, only a single user can connect, and the CHECKPOINT process is not started. CHECKPOINT guarantees that completed transactions are regularly written from the disk cache to the database device. (Typically, this option is used if you experience problems with system databases that should be repaired.) Enables the sp\_configure allow updates option. By default, allow updates is disabled.**

**-n**

**Does not use the Windows application log to record SQL Server events. If you start an instance of SQL Server with -n, we recommend that you also use the -e startup option. Otherwise, SQL Server events are not logged.**

**-s**

**Allows you to start a named instance of SQL Server 2005. Without the -s parameter set, the default instance will try to start. You must switch to the appropriate BINN directory for the instance at a command prompt before starting sqlservr.exe. For example, if Instance1 were to use \mssql$Instance1 for its binaries, the user must be in the \mssql$Instance1\binn directory to start sqlservr.exe -s instance1.**

**-T trace#**

**Indicates that an instance of SQL Server should be started with a specified trace flag (trace#) in effect. Trace flags are used to start the server with nonstandard behavior. For more information, see Trace Flags (Transact-SQL).**

**-x**

**Disables the keeping of CPU time and cache-hit ratio statistics. Allows maximum performance.**

**SQL Server Memory**

On a 32-bit system there is a maximum of 4GB of virtual address space (VAS). This is made up of: 2GB for the kernel and 2GB user mode for the Buffer Pool minus the Mem-To-Leave amount. Not all the memory is committed at start-up. The Mem-To-Leave is an area in the user mode portion of the 4GB address space that is reserved only at start-up and then released after start-up, so that external items to SQL Server can have contiguous blocks of memory to draw upon. This is typically extended stored procedures, linked servers, SQLCLR etc. However, this area of memory is not guaranteed to be contiguous. The Mem-To-Leave amount is calculated using the following formula:

(Stack Size x Max Worker Threads) + -g startup parameter.

The default setting for Max Worker Threads (in 2005/2008) is 0 which lets SQL Server allocate this value based on the number of CPU’s. In SQL 2000 it was set at 255.

Default thread stack size is 512k (512k on x86 and 2Mb on x64)

Run the following query to confirm both these running values…

SELECT max\_workers\_count

, stack\_size\_in\_bytes / 1024 AS stack\_size\_kb

FROM sys.dm\_os\_sys\_info ;

The default setting for –g parameter is 256MB. If it is set to more than 50% of VAS, it is ignored. On a 32-bit dual proc system, the calculation would be…

Mem-To-Leave = StackSize x MaxWorkerThreads + 256MB

= (512k x 256) + 256MB

= 128MB + 256MB

= 384MB

Now, calculate how much physical memory SQL Server has…

2048MB VAS – 384MB Mem-To-Leave = 1664MB

This will explain why on a 32-bit SQL Server with 4GB RAM that SQL will max out at approx 1.6GB. If there is less than 2GB physical memory on the server then SQL Server will reserve that amount – not more than it actually has. The buffer pool cannot reserve more physical memory than actually exists on the server. The VAS for the kernel is still 2GB.

With /3GB or /USERVA the user VAS is increased to 3GB (or a user defined amount with USERVA) and the kernel is restricted to 1GB. This is done in the boot.ini file or in Windows 2008 using BCDEdit.exe as there is no boot.ini in Windows 2008. Using the /3GB switch will increase memory to things like the procedure cache. This doesn’t change the Mem-To-Leave calculation but SQL Server gets 3GB minus Mem-To-Leave. This can cause problems for applications that require memory other than SQL Server which they would acquire from the Mem-To-Leave region if there isn’t enough free contiguous memory available. An example of this was Red-Gate SQL Backup wouldn’t run on one of our systems with the /3GB switch enabled but once it was removed – no problems. There is no way to defragment the Mem-To-Leave region.

The maximum virtual address space on a 64-bit Windows server it is 8TB. This is a Windows limitation as the actual amount is 16EB. However, no Windows server supports more than 2TB of RAM. Note that 64-bit does not have a Mem-To-Leave as there is no need because of the much increased VAS. Again, the kernel addressable space is also 8TB.

The following query will show the amount of physical memory on the server and the maximum virtual address space…

SELECT physical\_memory\_in\_bytes / 1024 / 1024 AS physical\_memory\_mb

, virtual\_memory\_in\_bytes / 1024 /1024 AS max\_possible\_virtual\_memory\_mb

FROM sys.dm\_os\_sys\_info ;

The CTE query below will show the size of the Mem-To-Leave VAS and the largest available block in the Mem-To-Leave. If the largest block free is less than 4MB the server is considered to be experiencing VAS memory pressure.

WITH VASummary ( Size, Reserved, Free )

AS ( SELECT Size = VaDump.Size

, Reserved = SUM(CASE ( CONVERT(INT, VaDump.Base) ^ 0 )

WHEN 0 THEN 0

ELSE 1

END)

, Free = SUM(CASE ( CONVERT(INT, VaDump.Base) ^ 0 )

WHEN 0 THEN 1

ELSE 0

END)

FROM (SELECT CONVERT(VARBINARY, SUM(region\_size\_in\_bytes)) AS Size

, region\_allocation\_base\_address AS Base

FROM sys.dm\_os\_virtual\_address\_dump

WHERE region\_allocation\_base\_address <> 0x0

GROUP BY region\_allocation\_base\_address

UNION

SELECT CONVERT(VARBINARY, region\_size\_in\_bytes)

, region\_allocation\_base\_address

FROM sys.dm\_os\_virtual\_address\_dump

WHERE region\_allocation\_base\_address = 0x0

) AS VaDump

GROUP BY Size

)

SELECT SUM(Size \* Free) / 1024 AS total\_available\_mem\_kb

, CAST(MAX(Size) AS INT) / 1024 AS max\_free\_size\_kb

FROM VASummary

WHERE Free <> 0 ;

This query below shows memory usage. It also shows available\_virtual\_memory\_mb to show you how much virtual memory SQL Server has left (if you suspect you are getting low). SQL 2008 only

SELECT total\_physical\_memory\_kb / 1024 AS total\_physical\_memory\_mb

, available\_physical\_memory\_kb / 1024 AS available\_physical\_memory\_mb

, system\_cache\_kb / 1024 AS system\_cache\_mb

, ( kernel\_paged\_pool\_kb + kernel\_nonpaged\_pool\_kb ) / 1024 AS kernel\_pool\_mb

, total\_page\_file\_kb / 1024 AS total\_virtual\_memory\_mb

, available\_page\_file\_kb / 1024 AS available\_virtual\_memory\_mb

, system\_memory\_state\_desc

FROM sys.dm\_os\_sys\_memory

Total memory used by SQL Server – this is for SQL2008 only

SELECT (SUM(virtual\_address\_space\_committed\_kb)

+ SUM(locked\_page\_allocations\_kb)

+ SUM(multi\_pages\_kb)) / 1024 AS total\_sql\_memory\_usage\_mb

FROM sys.dm\_os\_memory\_nodes ;

On systems with more than 4GB RAM, PAE gives access to the additional physical memory. PAE is enabled in the boot.ini file and it has nothing to do with AWE or /3GB. Individually an application cannot access all this extra memory given by PAE but in totality they can. AWE is a Windows API mechanism that allows an application like SQL Server to access the additional memory over the 4GB VAS, provided by PAE. Memory allocated with AWE is not part of the processes working set so it cannot be paged out. To enable AWE you must grant the “Lock pages in memory” priviledge to the SQL Server service account. This priviledge is granted automatically if the service account is running under the local system account. Also, using sp\_configure set “AWE Enabled” to 1. On 64-bit systems the AWE mechanism is still used however it is known as Locked Pages. The AWE setting in sp\_configure is ignored on 64-bit systems.

The following query shows how much memory is allocated through AWE

SELECT SUM(awe\_allocated\_kb) / 1024 AS awe\_allocated\_mb

FROM sys.dm\_os\_memory\_clerks ;

Check SQL Server errorlog to see a message that AWE can be used.

AWE extends database pages in the buffer pool only. Other memory objects in 32-bit SQL Server like the Procedure Cache cannot use AWE memory. Technically, single pages allocated for the Procedure Cache come from the buffer pool. This is why the Mem-To-Leave area is still important on 32-bit systems with more than 4GB RAM. Virtualalloc() is the routine that SQL Server uses for allocating memory. For your standard database page that is 8k, single page allocator is used and this comes from the buffer pool. For multi page allocations, that is greater than 8k, SQL uses the multi page allocator. These pages are allocated from outside the buffer pool – the mem-to-leave area.

The following query show the top 10 consumers of buffer pool memory

SELECT TOP 10

type

, SUM(single\_pages\_kb) AS buffer\_pool\_consumers\_kb

FROM sys.dm\_os\_memory\_clerks

GROUP BY type

ORDER BY SUM(single\_pages\_kb) DESC ;

The following shows how much memory is allocated through the multi-page allocator. Remember that the multi-page allocator allocates memory from outside the buffer pool.

SELECT type

, SUM(multi\_pages\_kb)

FROM sys.dm\_os\_memory\_clerks

WHERE multi\_pages\_kb <> 0

GROUP BY type ;

The amount of reserved memory on a 32-bit system without AWE is the visible amount that can actually be accessed. The visible amount also equals the Target amount, the amount of memory SQL Server would like to use. The committed amount will increase as physical memory is accessed. With AWE enabled, the target amount will increase to include the amount of AWE memory. The visible counter will stay the same.Visible always equals target on 64-bit. The following query shows these amounts.

SELECT ( bpool\_committed \* 8192 ) / 1024 / 1024 AS buffer\_pool\_committed\_mb

, ( CAST(bpool\_commit\_target AS BIGINT) \* 8192 ) / 1024 / 1024 AS buffer\_pool\_target\_mb

, ( bpool\_visible \* 8192 ) / 1024 / 1024 AS buffer\_pool\_visible

FROM sys.dm\_os\_sys\_info ;

The following query will show all memory allocations. Look at the single\_pages\_kb column where the value is greater than 0 to see what processes are allocated memory from the buffer pool.

SELECT \*

FROM sys.dm\_os\_memory\_clerks

WHERE ( single\_pages\_kb > 0 )

OR ( multi\_pages\_kb > 0 )

OR ( virtual\_memory\_committed\_kb > 0 ) ;

SQL Server myths…busted!

AWE API’s are not used on 64-bit SQL Server – FALSE (the “locked pages” uses the AWE API’s)

SQL Server never allocates more than “max server memory” – FALSE (virtual memory can be more)

PAE required for SQL Server 32-bit to use AWE on Windows 64-bit – FALSE (PAE overcomes the 32-bit limitation)

“Lock Pages” is required to avoid working set trim – FALSE (lock pages only locks buffer pool memory)

SQL Server allocates all of its memory at start-up – ONLY LARGE PAGES

Many thanks to Bob Ward at Microsoft for reviewing this article for accuracy.code

**Importance of setting Max Server Memory in SQL Server**

Of late, I'm observing that some of the customers are not setting up Max. Server Memory Properly or they never set Max. Server Memory at all in 64-bit SQL Server installations.

*What will happen if I don't set Max. Server Memory?*

Working Set trimming, Operating System unresponsiveness, Performance Problems in other applications running on the same server, Downgraded Backup Buffers etc...

*How should I set Max. Server Memory?*

sp\_configure 'max server memory (MB)',<Memory in MB>

*Is Max. Server Memory the upper limit to which SQL Server will consume Memory in the box?*

No. Max. Server Memory just limits the memory consumed by SQL Server Buffer Pool. Memory consumed by Non-buffer pool portion of SQL Server and memory consumed by external dll's loaded into SQL Server memory space is not controlled using Max. Server Memory

*How can I calculate the value for Max. Server Memory?*

In general, you should calculate Max. Server Memory using the formula:

***Max. Server Memory for a SQL Server Instance =***

***(Total RAM available to the OS) -***

***{***

***(Memory needed by Operating System which gets allocated to*** [***memory pool***](http://msdn.microsoft.com/en-us/library/aa965226.aspx)***, [filesystem cache](http://msdn.microsoft.com/en-us/library/aa364218(v=vs.85).aspx" \o "filesystem cache" \t "_blank),***[***PTE***](http://en.wikipedia.org/wiki/Page_table)***,*** [***desktop heap***](http://www.microsoft.com/downloads/en/details.aspx?familyid=5cfc9b74-97aa-4510-b4b9-b2dc98c8ed8b&displaylang=en)***,  Driver Images etc...) +***

***(Memory needed by Non-buffer Pool region of SQL Server which gets allocated to Multi Page Allocators, Worker Threads, COM, Extended SPs, Backup Buffers, CLR, Linked Server...) +***

***Memory required for SQL Server Agent, Replication Agents, Bulk Copy, SSRS, SSAS, SSIS, and Full Text  +***

***Memory required for Log shipping file copy depending on the size of log backups (if LS is configured) +***

***Memory required for other SQL Server instances running in the box +***

***Memory required for other applications running in the box  (Antivirus, Monitoring Softwares, Compression softwares etc...)***

***}***

I'm sure this is not a tough formula. ***Please note*** *that stack size of SQL Server in x64 is 2 MB so depending on the amount of worker threads* [*calculated by SQL Server*](http://msdn.microsoft.com/en-us/library/ms187024.aspx) *(SELECT max\_workers\_count FROM sys.dm\_os\_sys\_info) you may need to deduct memory in the above formula. Also if you have enabled* ***-g*** *startup parameter then you need to deduct the memory in the above formula accordingly. Make sure that you collect these memory requirements during the peak load.*

I'm giving some scenarios explaining how to set Max. Server Memory here:

**Scenario #1:**

I have Active-Active SQL Server instances running on Node A and Node B. I have 65 GB of RAM in each Node A and Node B. I have set Max. Server Memory for both the instances InstA and InstB to 60 GB leaving 5 GB to operating systems. Is it a right configuration?

NO.

Why?

Imagine a situation where both the instances are running on same node due to some issues on the other node. So if both the instances are memory hungry then you will have the worst performance possible. So for a perfect configuration, you should have 125 GB RAM on each node so that at any point, there will be no memory bottleneck immaterial on which node both the instances are running. Atleast you should have physical memory more than sum of Min. Server Memory of both the instances.

**Scenario #2:**

I have 3 SQL Server instances running on box SQLSRVR. I have 65 GB of RAM in SQLSRVR. I have set Max. Server Memory on each of the three instances to 60 GB leaving 5 GB to operating systems. Is it a right configuration?

NO.

Why?

This setting means that all the three instances can grab upto 60 GB of memory and if that happens then all the three instances will be performing poorly. Refer to the formula above and set Max. Server Memory accordingly to all the three instances. If that is something not possible, atleast you should have physical memory on the box which is more than sum of Min. Server Memory of all the instances.

Here is snippet out of a doc I prepared for a customer:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **x64** | **x86 - AWE** | **x86** |
| Set Max. Server Memory | Must | Must | Not Needed |
| sp\_configure 'awe enabled' | Not Needed | Needed | Not Needed |
| Lock Pages in Memory | Optional *(To use AWE API, Large Page Support)* | Needed | Not Needed |

**What components within SQL Server use Memory from BPool?**

Database Page Cache  
Log caches  
Procedure/Plan cache  
Query Workspace  
Locks  
Connection context   
Optimizing queries  
System-level data structures etc.,

**There is no “MemToLeave” for the 64bit version of the SQL Server Engine!**

Let me explain why….

This concept has to do with virtual address space (VAS) memory and not physical (or virtual memory) for the computer. We created the concept of “MemToLeave” memory for the virtual address space of a 32bit SQLSERVR.EXE because the virtual address space of a 32bit process in Windows is a limited resource. (2Gb by default, up to 3Gb with 4GT tuning options, and 4Gb in WoW64). And since the VAS is limited, the designers of the engine felt that by default the buffer pool should reserve as much of the VAS it could at startup (provided the amount of physical memory is 2Gb or higher. No sense reserving a big portion of the VAS if the amount of RAM is less than the max VAS for the process). So… since we are reserving most of the VAS for the Buffer Pool, we realized we can’t “take it all”. Why? Memory may be needed in the VAS for other “things” such as:

* Thread Stacks – When a thread is created, reserved memory is required for a thread stack (this is why you might seen some type of error like *17189* *SQL Server failed with error code 0xc0000000 to spawn a thread to process a new login or connection. Check the SQL Server error log and the Windows event logs for information about possible related problems.* This error typically means the thread stack could not be reserved)
* Heaps – Default Windows heap and any other heap created by a DLL
* SQL Server Multi-Page Allocations (MPA) – The engine itself may allocate memory outside the Buffer Pool because the size required is bigger than a SQL page which is 8Kb.
* Any other DLL that needs to use VirtualAlloc

What are the other “things” besides thread stacks and SQL MPA? Extended Stored Procedures, Linked Server Providers, COM objects, and host of other edge cases where DLLs get loaded in the process space of SQL Server.

The actual term “MemToLeave” comes from an internal variable name in the code that refers to the total amount of memory we “leave around” for these “things”.

The general algorithm at server startup calls for us to reserve this space using VirtualAlloc() and then after reserving the space for the buffer pool, we free this reservation. Now we have “left it” for these other “things” to reserve and commit memory. The general algorithm for the amount to “leave” is:

thread stack size \* ‘max worker threads’ + “the value of –g startup parameter” (measured in Mb)

* The thread stack size is platform dependent (for example 512Kb on x86 and 2Mb on x64)
* ‘max worker threads’ when set to 0 is dynamic and based on number of CPUs. See BOL for details
* The default for –g is 256 (which stands for 256Mb)

By now I hope you can guess that this algorithm and code are not needed for 64bit SQL Engine systems because the virtual address space for 64bit Windows applications are not as limited as with 32bit. In theory, the VAS for 64bit applications is 16 *Exabytes*.but in practical terms for Windows it is 8TB. (you should also note that the physical memory limit of Windows Server 2008 is 2TB so it is not even possible today to address the maximum VAS for a 64bit Windows application). Because of this, SQL Server doesn’t need to make any special VAS reservation at startup.  In fact, on 64bit SQL Server systems, the engine doesn’t reserve a large amount of memory for the buffer pool as with 32bit. So there is no need to leave any VAS space around since we don’t reserve most of it. You might be fooled on a x64 machine with 2Gb or greater if you monitor the “Virtual Bytes” of SQLSERVR.EXE right after startup. On my laptop with 3Gb of physical RAM, my virtual bytes was close to this number of 3Gb. But that is not the engine reserving space at startup. That is just the natural growth of the Buffer Pool when it allocates memory for overhead structures at startup such as SQLOS and lock manager. When we “grow” the Buffer Pool we reserve space in large, small number of blocks instead of small, larger number of blocks.

I hope this can help you and other bust the myth that “MemToLeave” exists on the x64 versions of SQL Server. It only applies to 32bit versions of SQL Server (even those 32bit versions running in Wow64).

**Lock Pages in Memory on 64-bit SQL Server**

The SQL Server Buffer Pool can dynamically grow and shrink. Its size will never be greater than the size of physically installed RAM. In addition it will never grow beyond the configuration setting "max server memory (MB)". Nevertheless, it is possible that Windows pages out a huge part of the buffer pool to the Windows page file, in particular when running an SAP application server on the same box.  
  
Paging out the buffer pool may slow down SQL Server. Reading a page from SQL Server data cache would then result in a physical I/O on the Windows page file. The 64-bit editions of SQL Server 2005 and newer prevent the paging of the buffer pool by using "Lock pages in memory". As a result, the whole buffer pool is removed from the normal Windows memory management. It cannot be used by Windows for other applications any more. When using "Lock pages in memory" you can see the the following message within the first 20 lines of the SQL Server Error Log:

* "Using locked pages for buffer pool"

In order to use "locked pages", the Windows account running the SQL Server service needs a special privilege. The "Local System" account already has this privilege. Therefore you don't have to take care about this, when running SQL Server using the "Local System" account.  
  
For some reason you may not want to use the "Local System" account. On a Windows Cluster Server you have to use a Windows domain account. In these cases you have to explicitly grant the privilege to use "locked pages" to the Windows account running the SQL Server service. Even the members of the Windows "Administrators" group do not have this privilege per default.  
  
Almost all of the memory consumed by SQL Server is taken from the SQL Server Buffer Pool. You can configure the maximum size of the buffer pool with the SQL Server configuration option "max server memory (MB)". Nevertheless, SQL Server also needs some memory in addition to the Buffer Pool (for contiguous memory > 8KB). This memory is directly allocated from Windows (within the address space of SQL Server Mem-to-leave memory).  
Therefore you should not configure "max server memory (MB)" to the physical RAM size or higher once you use "Lock pages in memory"

**Address Windowing Extensions (AWE) on 32-bit SQL Server**

It's only necessary to use AWE if you want to use more than 4GB RAM for a 32-bit SQL Server. In this case SAP strongly recommends to use a 64-bit edition of SQL Server running on 64-bit edition of Windows (x64 or IA64) instead.

**Solution**

The following describes how to set the privilege "Lock pages in memory" for the Windows account, running the SQL Server service. This is recommended when running either a 64-bit SQL Server or when using AWE on a 32-bit SQL Server.  
  
The SQL Server service account has be added to the Windows accounts, which have the privilege "Lock pages in memory":

* open gpedit.msc
* expand the tree
  + "Local Computer Policy"
  + "Computer Configuration"
  + "Windows Settings"
  + "Security Settings"
  + "Local Policies"
  + "User Rights Assignments"
* double click "Lock pages in memory"
* add the Windows account, which is used to run the SQL Server service

After that you have to restart the SQL Server service.  
A detailed description is available at:

* http://support.microsoft.com/default.aspx?scid=kb;en-us;918483

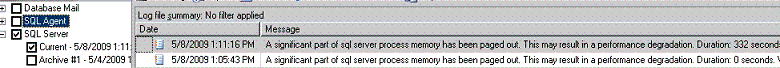
When using "Lock pages in memory" we recommend, not to give all memory to the SQL Server Buffer Pool. To do this, you have to reduce the SQL Server configuration setting "max server memory (MB)".

* When running SQL Server and an SAP instance on the same box, we recommend to set about a third of the memory to the SQL Server Buffer Pool. See SAP note 879941 for details.
* For a dedicated SQL Server you should roughly leave between 1.5GB (when having 8GB or less physical RAM) and 4GB (when having 32GB or more physical RAM) for the operating system and SQL Server Mem-to-leave memory. For example, on a Windows Server with 16GB RAM running a dedicated SQL Server you may configure the Buffer Pool to 13000 MB with the following SQL command:
  + sp\_configure 'max server memory (MB)', 13000  
    reconfigure with override
* **SQL Error A significant part of sql server process memory has been paged out**

**Issue:**

On Windows 2003 64bit running SQL Server 2005 Service Pack 2 (64bit) you find the following error recorded in the SQL Server log.

A significant part of the sql server process memory has been paged out.  This may result in a performance degradation.  Duration: xxx seconds



You may also have the following issues

* The performance of SQL Server 2005 decreases suddenly.
* SQL Server 2005 stops responding for a short time.
* A time-out occurs for applications that connect to SQL Server 2005.
* Problems occur when you run even simple commands or use applications on the system.

**Cause:**

By default SQL server will use all available memory, and dynamically release memory as required.  A common cause of this problem occurs when SQL Server does not release memory because the server is under heavy load.

The typical scenario that causes this issue is when the Windows OS notifies all processes on the server that memory is running low.  By default SQL server should release memory, but it is unable to process the request in a timely manner because of a heavy load on the SQL Server.  When this happens the Windows OS is forced to require all processes to release memory.  This will then force SQL Server to log the above events in the SQL Server Log, and page its entire buffer pool memory to disk.  Since the request to release memory applies to all processes, all processes will be forced to release memory as well.

This issue is normally limited to 64bit Windows 2003 running SQL Server 2005, but may occur on 32bit Windows 2003 and SQL Server 2005 as well.

There may be other causes of this issue, please see the More Information section for details of those issues.

**Solution:**

Configure SQL **minimum** and **maximum server memory** settings to allow the OS to and other services enough RAM to function.  To determine the amount of memory to reserve for the OS and other services use Performance Monitor to record the **maximum working set size** for each process.  Total those values, and subtract that amount from the total physical RAM.  Use this value as the **maximum server memory** setting for SQL Server.  However, because the other processes can be memory starved these maximum values may not be adequate.  Therefore, it may be necessary to repeat this process several times.

Configure SQL Server to use a **minimum server memory** of 25% to 50% of the **maximum server memory** setting.  Configuring this value will help prevent all of the SQL Buffer Pool from being paged out should the request to release all buffer pool memory occur again.

Given that the SQL Server is under a heavy load, it may also be necessary to add more memory to your server.  You should monitor performance of SQL server to determine if more memory is needed.

Verify that **File and Printer Sharing for Microsoft Networks** is configured for **Maximize data throughput for network applications**.  This setting will help reduce the amount of **system cache** used by the by the OS, and is the recommended setting for SQL Server 2005.

**More Information:**

[How to reduce paging of buffer pool memory in the 64-bit version of SQL Server 2005](http://support.microsoft.com/kb/918483)

[Omar Blog - A significant part of sql server process memory has been paged out](http://msmvps.com/blogs/omar/archive/2007/09/19/a-significant-part-of-sql-server-process-memory-has-been-paged-out-this-may-result-in-performance-degradation.aspx)

[Slavao Blog - Memory Pressure](http://blogs.msdn.com.urrsa2.com/slavao/archive/2005/02/01/364523.aspx)

[Slavao BLog - SQL memory manager: responding to memory pressure](http://blogs.msdn.com/slavao/archive/2005/02/19/376714.aspx)

Dynamic Memory Management and Memory Architecture of SQL 2005

By default Microsoft SQL Server 2005 dynamically acquires and frees memory as needed. It is typically not necessary for an administrator to specify how much memory should be allocated to SQL Server, although the option still exists and is required in some environments.

The default memory management behavior of the Microsoft SQL Server Database Engine is to acquire as much memory as it needs without creating a memory shortage on the system.  The Database Engine does this by using Memory Notification functions of the [Memory Management APIs](http://msdn.microsoft.com/en-us/library/aa366781(VS.85).aspx) in Microsoft Windows 2003.

Under Windows Server 2003, SQL Server uses the memory notification function **QueryMemoryResourceNotification** to determine when the buffer pool may allocate memory and release memory.  This is essentially a dedicated thread that listens for one of two notification types: **low-memory-resource** or **high-memory-resource**.  The default level of available memory that signals a **low-memory-resource** notification event is approximately 32 MB per 4 GB, to a maximum of 64 MB. The default level that signals a **high-memory-resource** notification event is three times the default low-memory value (96MB by default).  As of Windows Server 2003 SP1, Windows does not balance memory across applications with the Memory Notification functions.  It merely provides global feedback as to the availability of memory on the system.

When available memory becomes low the Windows OS will turn on the **low-memory-resource** notification. SQL Server and other applications that listen for this notification will be given the opportunity to shrink their memory usage before the Windows OS does it system wide.  When the system is under heavy load SQL Server and other processes may not be able release memory in time.  When this happens the OS will eventually intervene by paging (trimming) the working sets for all processes to disk.

Allowing SQL Server to use memory dynamically is recommended; however, you can set the memory options manually and restrict the amount of memory that SQL Server can access. Before you set the amount of memory for SQL Server, determine the appropriate memory setting by subtracting, from the total physical memory, the memory required for Windows Server 2003 and any other instances of SQL Server (and other system uses, if the computer is not wholly dedicated to SQL Server). This difference is the maximum amount of memory you should assign to SQL Server.  However, because the other processes can be memory starved these maximum values may not be adequate.  Therefore, it may be necessary to repeat this process several times.

Working Set Size Option

The **set working set size** option is still present in the **sp\_configure** stored procedure, but its functionality is unavailable in Microsoft SQL Server 2005. (The setting has no effect.)

**Large Pages, theory and usage with SQL Server 2008**

This article focuses on “Large Pages” on Windows Server systems, how they work and how they can be used to enhance the performance of SQL Server on 64-bit systems. It is part of a two-piece article on large pages. The [second part](http://www.GabesVirtualWorld.com/large-pages-transparent-page-sharing-and-how-they-influence-the-consolidation-ratio) is written by Gabrie van Zanten from [GabesVirtualWorld](http://www.gabesvirtualworld.com) and explains impact when large pages are used on VMWare ESX / vSphere.

**Windows Memory Management Features**

Before diving into the definition of large pages, let me introduce you to a small subset of memory management features of the Windows operating system. As you might know, a system can have both physical and virtual memory addresses. For starters, the physical memory addresses obviously refers to the address on the physical memory, as present in your system.

Virtual addresses are another story. By default, each process within Windows gets 2GB of virtual address space to use. While most of the time there will be more virtual address space inside a system then there is physical address space, there has to be some kind of process that takes control of this situation and controls which parts of the virtual addresses get mapped to the physical memory and which parts do not. This process is called the Memory Manager. The Memory Manager has –in short- a few tasks:

* Translating the virtual address space of a process into the physical address space.
* Moving content from memory to the page file when there’s no more free space available in memory.
* Various other functions, from which one is Large Page Support.

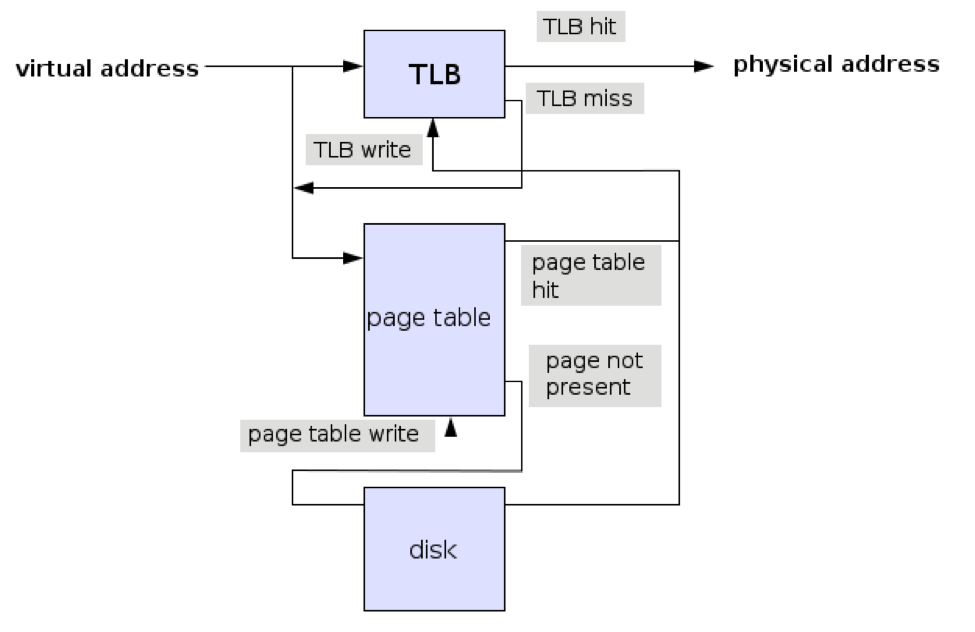
**Large Page Definition**

What exactly are large pages and what are their advantages? [MSDN](http://msdn.microsoft.com/en-us/library/aa366720.aspx) defines large pages as follows:

*Large-page support enables server applications to establish large-page memory regions, which is particularly useful on 64-bit Windows. Each large-page translation uses a single translation buffer inside the CPU. The size of this buffer is typically three orders of magnitude larger than the native page size; this increases the efficiency of the translation buffer, which can increase performance for frequently accessed memory.*

In detail: The physical and virtual address spaces consist of pages. A page is the smallest unit (entity) with which the Memory Manager translates from virtual to physical address space. These pages can have two sizes named “small” and “large”. For an x64 system, a small page is 4 KB in size, where a large page is 2 MB in size.

So what is the advantage of having larger page sizes? Well, for starters we can tell that a large page can hold more data than a small page does (hence the name). Second, the Translation Look Aside Buffer (TLB) is a CPU cache that gets called upon when a page is first accessed and will generate an entry holding a cache with information about references to data in that page. The task of the TLB is to generate a cache which holds page-data with a high hit-ratio. The following image from [Wikipedia](http://en.wikipedia.org/wiki/File:Page_table_actions.svg) explains the situation in graphical form:



So when you use small pages, the TLB holds more entries with page-caches. When large pages are used, fewer entries can be used to cache the same amount of page-data.

Because the cache buffer of the TLB is not very big, entries inside the TLB get swapped for other entries often. (This is done by invalidating the TLB entry by the program that is using the large pages pool), When an accessed page is not present in the TLB, the pagetable will be referenced to look-up information, which in turn is cached again by the TLB. The page table is where the operating system stores its mappings of virtual addresses to physical addresses.

So when using large pages, fewer entries can be used to store more data, making the space usage more efficient. A process can make use of these large pages when it is programmed to do so.

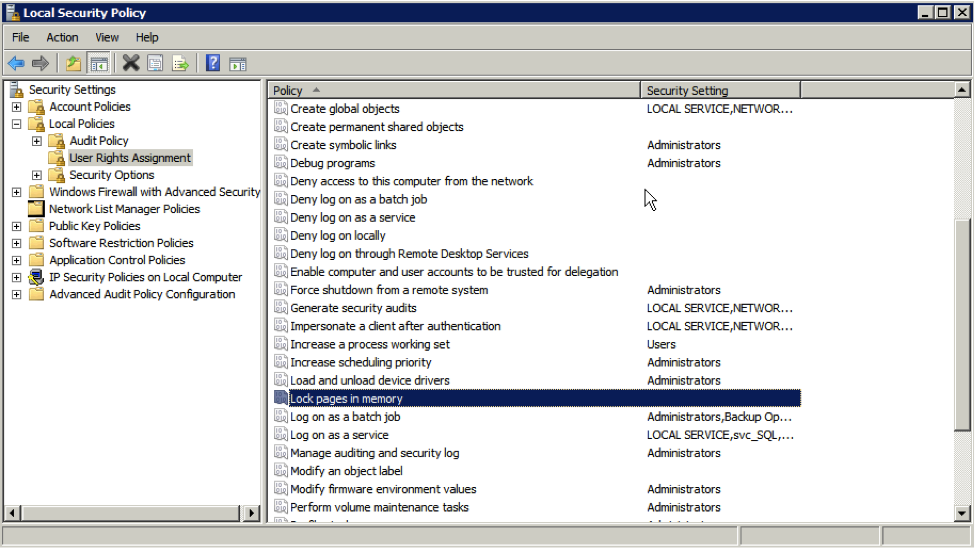
The story above can be summarized in the following sentence ([Source](http://blogs.msdn.com/b/slavao/archive/2005/02/11/371063.aspx)): SQLOS and SQL Server use large pages to minimize TLB misses when accessing hot data structures.

**Locking Pages in Memory**

On a 64-bit system, Microsoft SQL Server can only make use of large pages by first enabling the “Lock pages in memory” security policy for the account which runs the server. By default, this feature only works on SQL Server 2005 / 2008 Enterprise or Developer 64-bit editions. Install the following [hotfix](http://support.microsoft.com/kb/970070) to add this feature to the standard version of SQL Server 2005 and 2008. Also, please note that the service has to be restarted for the policy to become active.

Detailed steps:

* Navigate to Start, Administrative Tools and Select “Local Security Policy”
* Navigate to Local Policies, User Rights Assignment and Look for “Lock Pages in Memory”



* Add the service account that you use for SQL to the policy.
* Click start, run and fire up the services.msc
* Look for the SQL Server (MSSQLSERVER) and restart it.
* Navigate to the log file of the SQL Server (standard path is C:Program FilesMicrosoft SQL ServerMSSQL10\_50.MSSQLSERVERMSSQLLogERRORLOG
* If the policy was enabled for the SQL Instance, you will see the following line in the errorlog:

2010-12-19 17:41:39.64 Server      Using locked pages for buffer pool.

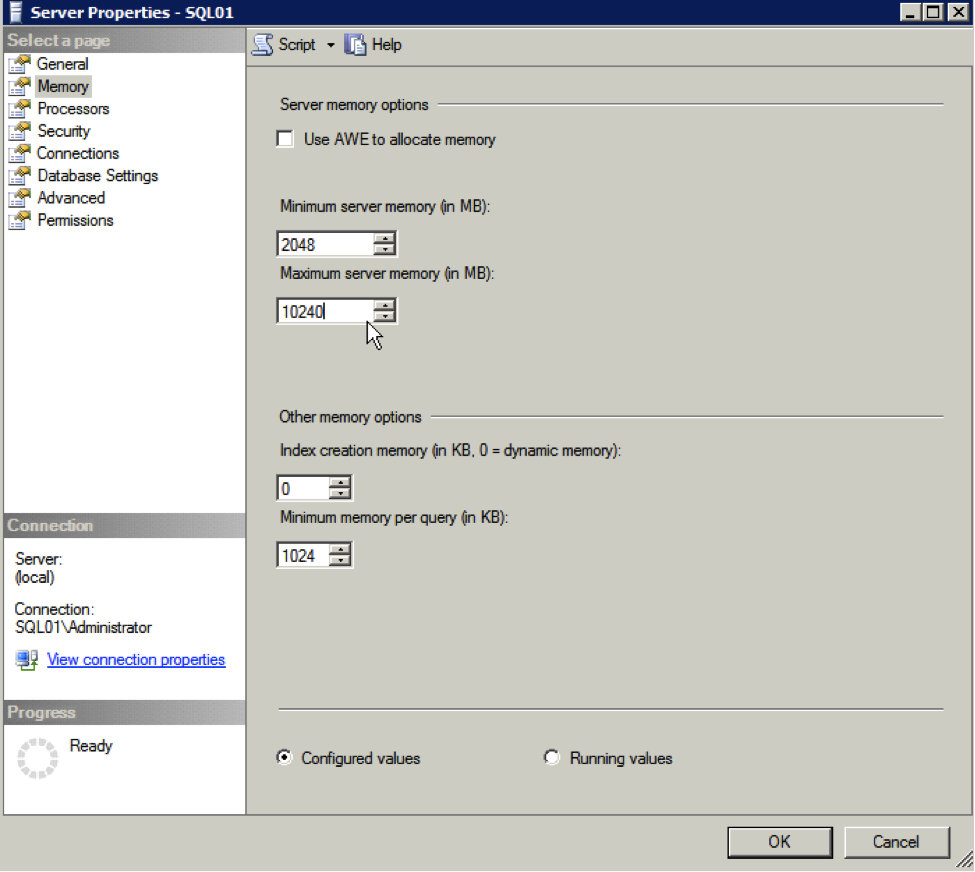
The “Lock Pages in Memory” setting allows locking of the pages SQL uses in physical memory, so the OS doesn’t have the right to swap away the pages to a page file.

**Min and Max Memory Setting**

Microsoft recommends setting the min and max server memory setting for the buffer pool of the SQL instance after you have enabled the policy. If you don’t set the values for min and max server memory (which is in fact the min and max memory for the buffer pool), SQL Server will fill up all the memory there is present in the machine, leaving no memory left for other processes, such as the operating system.

To set the values for min and max memory values, do the following:

* Fire up the SQL Server Management Studio
* Right click the server and choose properties
* Select the Memory Page
* Set the memory values accordingly



Please note that the use of sp\_configure to configure the min and max memory values is deprecated and can be removed in a future version of Microsoft SQL Server!

As from Windows Server 2003, SQL Server queries the QueryMemoryResourceNotification API to discover the amount of free available memory on the system so it can adjust the SQL buffer pool to allocate / release memory. By setting the min and max server memory setting, this process will be omitted. You can use the following formula to determine the appropriate memory setting:

Max server memory = Total Physical Memory – Memory required by the OS + other Instances of SQL (and even other processes, if the system is not only used for hosting SQL databases).

Pretty simple isn’t it? Please note that the server will not fill up the buffer pool immediately with the amount of memory specified in the Max server memory setting. It will increase the amount of memory as needed by the SQL server until it hits the max server memory value. It will not exceed this value unless you configure a higher value.

The Min server memory option is used to guarantee a minimum amount of memory assigned to the buffer pool of the SQL instance. Also, the amount of memory set here will not be assigned to the buffer pool on startup, it will be assigned as needed, but once it has reached the min server memory value, it can never drop below this threshold.

When you set min and max server memory to the same values, it will fill up the buffer pool until it reaches it’s min / max value, and will never drop below that point. This is done to stop SQL releasing and acquiring for the memory pool again.

**Setting Trace Flag 834**

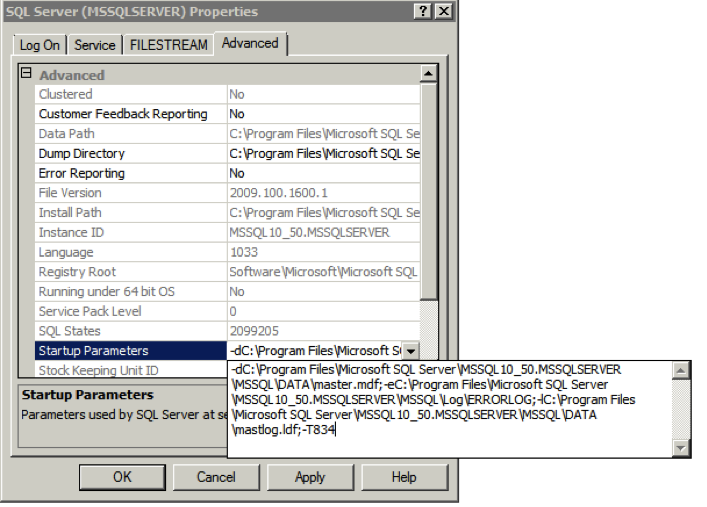
After setting the policy and modifying the min and max memory values, trace flag 834 has to be set for SQL Server to actually use large page allocations for the buffer pool. Setting this trace flag requires a few prerequisites:

* The “Lock pages in memory” option must be enabled
* The machine needs to have 8 GB of Physical Memory
* A 64-bit version of Microsoft SQL Server must be used.
* Best used only by servers dedicated to servicing SQL Databases.

As stated in [this kb article](http://support.microsoft.com/kb/920093) from Microsoft, large pages should be test thoroughly before implemented in production.

Set it up as follows:

* Start the SQL Server Configuration Manager
* Click SQL Server Services and right click SQL Server (Instancename)
* Choose properties
* Click the advanced tab
* Add “;-T834” to the “Startup Parameters” section.
* Restart the SQL Server Service



When the service has been rebooted, check the errorlog again. It now shows the following lines:

2010-12-22 16:12:29.79 Server Large Page Extensions enabled.

2010-12-22 16:12:31.79 Server Large Page Granularity: 2097152

2010-12-22 16:12:45.38 Server Large Page Allocated: 32MB

2010-12-22 16:13:03.15 Server Using locked pages for buffer pool.

When you want to know how much kb of memory is used for large pages, use the following command in SQL 2008: “select large\_page\_allocations\_kb from sys.dm\_os\_process\_memory”

Right now you have your SQL Server running with large pages!

**Notice on usage of Large pages when your OS is hosted on a Hypervisor**

Please take into account that enabling large pages on a virtual machine running on VMware ESX / vSphere can have a negative effect. Please see [this blogpost](http://www.GabesVirtualWorld.com/large-pages-transparent-page-sharing-and-how-they-influence-the-consolidation-ratio) on [GabesVirtualWorld](http://www.gabesvirtualworld.com) for more information on this subject.

[**Cannot use Large Page Extensions: lock memory privilege was not granted**](http://www.sqlserver-dba.com/2012/05/cannot-use-large-page-extensions-lock-memory-privilege-was-not-granted.html)

**Question :**This message appears in the SQL Server Logs “Cannot use Large Page Extensions: lock memory privilege was not granted”.  It’s a  SQL Server 2008 64 bit on Windows 2008 R2 64-bit Operating System. The Server memory is 64 GB.

 AWE is not enabled and the SQL Server Maximum Server Memory is 47 GB

 Should I be worried?

**Answer :**  It depends. When SQL Server is booted  - the SQL Server decides if large page support is used.   The Windows memory (x64) is 4kb versus 2MB for large pages.

The decision is based on the three criteria being true :

1)      The server having at least 8GB memory

2)      SQL Server Enterprise Edition

3)       [**Lock Pages in Memory**](http://www.sqlserver-dba.com/2011/04/powershellpseexec-and-lock-pages-in-memory-made-easy.html) is set

Lock pages in memory keeps the pages in memory and doesn’t allow the OS to page them out. It is possible performance is improved – **but does require testing**.

To view large page allocation use the sys.dm\_os\_process\_memory DMV.

**Great SQL Server Debates: Lock Pages in Memory**

Recently, I responded to the following, seemingly-innocuous question, on Twitter: "*Should I use Lock Pages in Memory as a Default Configuration?*" My answer was *yes*, if it's 64-bit, you have more than 16-32 GB RAM installed, you've, set '**max server memory**' appropriately, and you monitor the **Memory\Available Mbytes** counter, then you *should* enable it by default. In the ensuing debate, it became clear that my advice was somewhat out-of-step with that offered on this topic by the CSS team at Microsoft, as well as several respected SQL Server MVPs; not on the point that the **Lock Pages in Memory** privilege was still sometimes needed, in response to the Operating System forcing SQL Server to trim its working memory set, but on the point that it should be assigned *by default*.

With their kind permission, I'm going to single out posts by two SQL Server MVPs, and well-respected members of our SQL Server community, Brent Ozar and Glenn Alan Berry, which summarize the countervailing opinion:

* [SQL Server 2005/2008 Setup Checklist](http://www.brentozar.com/archive/2008/03/sql-server-2005-setup-checklist-part-1-before-the-install/) – "*Configuring Lock pages in memory used to be a best practice. It's not such a great idea anymore, especially for virtual machines…*"
* [SQL Server and the "Lock pages in Memory" Right in Windows Server](http://sqlserverperformance.wordpress.com/2011/02/14/sql-server-and-the-lock-pages-in-memory-right-in-windows-server/) – "*If I was running SQL Server 2005 on top of Windows Server 2003, I would be more tempted to use this setting. If I was running SQL Server 2008 R2 on top of Windows Server 2008 R2, I would be much less tempted to use this setting*"

In other words, on older Windows Server versions, which responded aggressively to memory pressure by trimming SQL Server's working set, then use of LPIM was highly advisable. However, with new operating systems, with improved memory management, it's better to not to assign the LPIM privilege, unless it's really required. In many ways, this is sound advice, and even as recently as early 2011, when I first joined SQLskills as a consultant, I would have agreed with it entirely. However, my experiences since then have convinced me otherwise. In this article, I hope to set out, definitively, the major issues surrounding use of LPIM, and to explain why I recommend that **Lock Pages in Memory** be used as a default configuration option on all 64-bit instances of SQL Server, unless you have a good reason not to (such as use of a virtualized environment).

## Essentials of SQL OS Memory allocation and management

The Windows Operating System runs every process, including the SQL Server process, in its own dedicated area of virtual memory, known as the Virtual Address Space (VAS). The VAS is divided into two regions; **kernel mode** (or system) space and **user mode** (or application) space. The kernel mode VAS is used by the OS, for mapping various system data structures such as the file cache, Paged and Non-Page pools. The user mode VAS is used to map memory for the currently-executing application process i.e. SQL Server.

Under default configuration, all SQL Server's memory allocations are made in the user mode VAS, using calls to the **VirtualAlloc()** Windows API function. Any memory allocated by **VirtualAlloc()** is *pageable*, meaning that the Windows OS can force this memory to be paged to disk, in response to memory pressure. Memory allocated by **VirtualAlloc** doesn't have to be physically present but Windows ensures that the amount of memory committed by SQL Server will be less than, or equal to, the installed physical memory, plus page file capacity.

When the **Lock Pages in Memory** permission is assigned to the SQL Server service account, then memory allocations for the buffer pool are made using calls to a function in the AWE API called **AllocateUserPhysicalPages()**.All of the memory that is allocated using the **AllocateUserPhysicalPages()** API are considered locked, i.e. *non-pageable*, and must be backed by physical memory on the server.

As a general rule, SQL Server will use as much memory as you can give it, and it will not release the memory that it has allocated under normal operations, unless the Windows Server OS sets the **memory low** resource notification flag. A component of the SQLOS called the Resource Monitor monitors the **QueryMemoryResourceNotification** Windows Server API and when Windows sets the low memory resource notification, the SQLOS will respond by sweeping its caches internally to reduce the process working set, and release memory back to Windows.

At this point any memory that is pageable, i.e. memory allocated via **VirtualAlloc()**, may be paged to disk in order to free up more memory for the OS. Conversely, any memory allocated using **AllocateUserPhysicalPages()** is locked and cannot be paged. In cases where large amounts of memory are locked, it can limit how much memory the Windows OS can reclaim under pressure, and this could lead to system instability.

Nevertheless, the Windows OS will do what it can to reduce memory consumption and any of the SQLOS structures that aren't locked, for example the thread stacks, and any other non-buffer pool memory, can still be paged out. This can cause problem performance issues, but it will rarely impact SQL Server performance in the same way as having 50 GB+ of buffer pool paged to disk, immediately.

The potential problem with using Lock Pages in Memory is that if the SQLOS can't respond quickly enough to a low memory notification, to release the memory that Windows needs, it can cause Out of Memory (OOM) errors in Windows, and instability. The way to avoid this, as we'll discuss a little alter, is to make appropriate configuration changes to prevent Windows from experiencing memory pressure in the first place.

## A Brief History of Locked Pages in Memory

Let me start by saying that this whole debate of *whether or not* to use LPIM is framed entirely within the context of a 64-bit environment. However, in order to understand fully why this topic has caused so much confusion over the years, it's worth briefly revisiting the bad old days of 32-bit.

If you are running 32-bit SQL Server, and need access to more than 2 GB of user mode VAS, then you *have* to use Lock Pages in Memory; there is no debate there. You must configure the OS to use Physical Address Extensions (**PAE**), enable Address Windowing Extensions (**AWE**), and then assign the **Lock Pages in Memory** permission to the SQL Server account so that it can allocate AWE memory, via calls to **AllocateUserPhysicalPages()**. So, in a typical 32-bit server using LPIM you'd have, in addition to the 2 GB of pageable user mode VAS, a separate AWE-mapped area, up to 64 GB (the PAE pointer was 36-bit), of non-pageable memory. This AWE-mapped area is for exclusive use by the *data cache portion* of the buffer pool. The rest of the buffer pool (mainly the plan cache), and other non-buffer pool allocations are still mapped within the 2 GB of user mode VAS.

However, the advent of a 64-bit SQL Server process completely changed the dynamics of memory allocation by the SQLOS. In place of the default 2 GB user mode VAS, for a 32-bit process, a 64-bit process has access to up to 8 TB of user mode VAS out-of-the-box, without any need for further configuration changes! 64-bit users now have a potentially-vast amount of memory for the buffer pool, but all of which is allocated via **VirtualAlloc**, and backed by user mode VAS, and so is pageable *i.e.* in the absence of LPIM, the memory allocated for the data and plan cache is pageable.

In 64-bit SQL Server, the SQL Server account still requires the **Lock Pages in Memory** permission in order to be able to allocate locked pages, via **AllocateUserPhysicalPages()**, but there are a couple of big differences:

* **The underlying reliance on AWE-mapped memory is removed**. You *do not* need AWE in 64-bit SQL Server; the **awe enabled sp\_configure** option has no meaning. The continued use of the same AWE API function is purely to ensure that the allocated pages are locked.
* **Memory allocated via AllocateUserPhysicalPages() can be used for both the data cache and plan cache**. In 64-bit SQL Server, the plan cache is no longer allocated separately (it now uses stolen pages from the buffer pool)

The crux of the problem is that, under a number of conditions, the Windows Server OS may trigger hard working set trims of the running processes, forcing large amounts of memory allocated by SQL Server to be paged out to disk, and leading to performance degradation in the SQL Server environment. Some of the specific scenarios where this can occur have been documented by the Product Support Services group at Microsoft in KB 918483 (<http://support.microsoft.com/kb/918483>).

This was a particular problem for early 64-bit environments – SQL Server 2005 on Windows Server 2003 – where the OS was aggressive in its requests to trim SQL Server's working set, in response to memory pressure. The problem was greatly exacerbated by the fact that in early 64-bit SQL Server **Lock Pages in Memory** was an Enterprise-only feature. If you were running Standard Edition, there was nothing you could do to prevent these working set trims.

So, whereas 32-bit users had a relatively small area of pageable memory (the 2 GB of user mode VAS) and then a bigger area of locked memory for the data cache, which was protected from hard trims, early Standard edition 64-bit users had no defense against the OS hard trimming the most significant portion of the SQL Server working set, in response to memory pressure.

If you have a SQL Server with 64GB RAM and 52GB of that is allocated to the buffer pool, these hard trims have a significant performance impact on the server operation, since the entire purpose of the buffer pool is to minimize disk access by caching frequently, or at least recently used pages in memory where the access time is significantly faster than it would by retrieving the pages from disk.

I can only guess at the number of product support cases that were created as a result of this issue, and it took a lot of pressure from the community, and specifically from the MVPs, to have LPIM added to Standard Edition. Finally, the pressure paid off and Bob Ward announced, first at PASS Europe 2009 and then on his [blog](http://blogs.msdn.com/b/psssql/archive/2009/04/24/sql-server-locked-pages-and-standard-sku.aspx) that SQL Server Standard Edition would finally include the option to use Lock Pages in Memory. This change was released in May 2009 with CU4 for SQL Server 2005 Service Pack 3 and CU2 for SQL Server 2008 Service Pack 1 (<http://support.microsoft.com/kb/970070>). If you were on Standard Edition, you needed to apply the appropriate cumulative update to be able to enable Trace Flag 845 to make use of **Lock Pages in Memory** for the buffer pool.

The situation stabilized, and subsequently, in Windows Server 2008, changes made to the memory manager (also documented in the previously-referenced KB article) greatly reduced the problem of hard working set trims for SQL Server. This prompted Microsoft to announce, soon after the release of Windows Server 2008, that **Lock Pages in Memory** was no longer required.

This brings us more or less back to the current situation, and the advice from the Brent, Glenn and others that if you're running SQL Server 2005 on Windows Server 2003, you need LPIM; if you're running Windows Server 2008, or later, you don't, at least not as a default.

## Why Lock Pages in Memory should be a default configuration

I'll present my reasons for the continued use of LPIM, as a default choice, even on Windows Server 2008 and Windows Server 2008 R2, in terms of the major counter-arguments:

* **Improvements in memory management mean it's no longer required** – my experience suggests otherwise
* **On 64-bit, use of LPIM can cause OS instability during memory pressure,** as it limits the memory Windows can rapidly reclaim through paging – these issues can be avoided by careful configuration of SQL Server memory settings

As a note of caution however, before we start, I refer you to this recent [blog post](http://blogs.msdn.com/b/psssql/archive/2012/03/20/setfileiooverlappedrange-can-lead-to-unexpected-behavior-for-sql-server-2008-r2-or-sql-server-2012-denali.aspx) from the SQL Server Support team, regarding a potential bug that could lead to corruption when using LPIM on certain builds of SQL Server 2008 R2 and 2012, on certain unpatched Windows Server installations.

## Hard trims still happen on recent Windows Server versions

It's certainly true that changes made to Windows Server 2008 memory manager make the problem much less drastic than it was under Windows Server 2003. However, some of the problems listed in the [KB article](http://support.microsoft.com/kb/918483) *still* occur under Windows Server 2008 and Windows Server 2008 R2 and can still result in hard trims of the working set and to serious problems for SQL Server.

I've worked with numerous clients, who were Windows Server 2008 and Windows Server 2008 R2 and were nevertheless suffering from performance problems that had their root cause in hard working set trims issued at the behest of Windows.

In some cases, these trims weren't actually being caused by memory pressure on the system; in the worst case the server had 64 GB RAM installed in it, and at the point that the hard trims were being triggered by Windows Server 2008 R2, the server had over 48 GB of available memory!

What is really insidious about this particular case is that because less than 50% of the SQL Server process memory was getting trimmed, no notifications about the trim were being logged in the SQL Server error log! The only way to track down this problem was to monitor paging, via the Performance Monitor counters for the Process object (as documented in http://support.microsoft.com/kb/918483).

Over time, I have engaged with countless customers where this has proven to be the cause of their performance issues with SQL Server, and as a result I have reverted to the stance that **Lock Pages in Memory** should be used as a default configuration if you are running SQL Server on a 64-bit instance of SQL, with more than 16-32 GB RAM, regardless of the version and edition of Windows Server OS that you are running.

If you are suffering performance problems related to memory trims, then it's very likely that enabling **Lock Page in Memory** will help. Of course, if you can find out the underlying cause of the working set trim and stop it happening then this is even better. Unfortunately, tracking down the cause of the sorts of problems detailed in the KB article can take a long time, even with the help of Microsoft Customer Support Services, and until you do, you will continue to have performance problems with SQL Server. I would rather prevent issues from occurring in an environment than wait to find out if it might occur and then try to react to performance issues after the fact.

## Preventing Problems with Lock Pages in Memory

With Lock Pages in Memory, as with everything, there is no such thing as a free lunch and while it can help prevent potential problems associated with hard working set trims of SQL Server, it can also lead to out-of-memory conditions for the Windows Server OS, if appropriate configuration of the overall system has not been made to prevent the OS from getting into memory pressure. Even though the SQLOS is designed to monitor for low memory notifications from the Windows OS through the **QueryMemoryResourceNotification** API, it is possible that under load, the SQLOS won't be able to respond quickly enough to a low memory condition, and the Windows OS could become unstable as a result.

As a best practice, even when you're not using **Lock Pages in Memory** and certainly before enabling it, you need to set an appropriate value for the **'max server memory' sp\_configure** option, in order to limit the amount of memory that SQL Server allocates for its buffer pool and to leave enough memory available for the Windows Server OS, and other applications running on the server, to be able to operate without triggering memory pressure on the server. These "other applications" include anti-virus software, Integration Services, and any multi-page allocations by SQL Server that occur outside of the buffer pool.

Unfortunately, there is no hard and fast rule that determines what the optimal value for '**max server memory**' will be, for a given instance of SQL Server. The best recommendation I can make would be to set this value artificially low and then gradually fine tune the value, based on monitoring of the **Memory\Available Mbytes** performance counter in Windows, till you reach the optimum value for the server.

I tend to start out by reserving 1-2 GB RAM for the OS, and then an additional 1GB for each 4 GB of RAM installed from 4-16 GB, and then 1 GB for every 8 GB RAM installed above 16 GB RAM. I then monitor the **Memory\Available Mbytes** counter over time to determine peak memory usage is for the system. Memory in excess of what's required to support this peak memory usage can be added to the '**max server memory**' option.

At a more technical level, you can perform the necessary calculations for the size of the SQL Server Thread Stack, estimate the usage of memory from multi-page allocators in SQL Server, such as SQLCLR, add in the additional memory requirements for the SQL Server process, and then for each of the applications or services that are running, to try to arrive at a reasonable the starting value for **'max server memory'**. My personal experience has been that trying to select a value in this manner tends to result in setting the value too high, and it isn’t always clear that this is the case until you have a problem.

## Considerations for Virtual environments

Brent, in his previously-referenced blog post, gives the basis of a compelling argument against the use of LPIM, for SQL Servers running in a virtual environment. Having SQL Server running on Virtual machines certainly does pose an interesting problem regarding use of **Lock Pages in Memory**, since the potential for memory overcommit exists, depending on the hypervisor being used. Memory overcommit is a scenario where the memory allocated to the virtual machines running on the host exceeds the total amount of physical RAM available in the server.

When memory overcommit occurs, one of the first ways that the hypervisor reacts is to make use of a special driver, known as a balloon driver, which is installed in the VM as part of the VM tools. In essence, the hypervisor sets the balloon driver the task of reducing memory consumption in the VM to a target level, and the balloon driver responds by acquiring memory in the VM. This 'ballooning' activity creates memory pressure in the VM, which in turn prompts the guest OS to reduce the physical memory usage of processes in the VM. The released memory is then available to the hypervisor for allocation to the other VMs running on the host, as necessary to prevent memory pressure at the hypervisor level. Ultimately, if not enough memory can be released, the hypervisor will also begin hard paging VM memory to disk, which can have a huge impact on performance.

The situation is even more complex when SQL Server is using **Lock Pages in Memory**, since the ballooning causes memory pressure in the VM, but the guest OS is limited in the amount of memory it can free up by paging to disk. SQLOS will still respond to the memory pressure by reducing its memory usage internally but it may fail to respond quickly enough, resulting in an OOM condition for Windows OS running in the guest, just as if a physical machine ran out of memory.

There are a number of ways to deal with the balloon driver issues associated with VMs, the worst of which is to disable the balloon driver entirely for the SQL Server VM. The balloon driver exists to allow the hypervisor to manage memory pressure in the best way possible for overall performance of the VMs running on the host, so it should not be disabled as a general rule.

Instead, if the VM has a condition such that **Lock Pages in Memory** is needed to prevent hard paging of the SQL Server working set, or the VM needs to be guaranteed to have a minimum set of memory resources, then a reservation should be configured for the VM so that the hypervisor only balloons its memory as a last resort. In a situation like this, it would be best to set the **'min server memory' sp\_configure** option so that the SQL Server only reduces its memory down to the minimum level required for appropriate application performance.

[SQL Server: Database Page Basics](http://blogs.extremeexperts.com/2012/11/22/sql-server-database-page-basics/)

As you have been reading multiple posts around the database fundamentals topic in the past few weeks. Here is yet another post that will discuss around the basics of SQL Server Database Pages concept. I have seen there are many junior DBA’s who come from other platforms and ask what is a page, why is it restrictive, Is there a way to configure, are there optimizations I need to be aware etc. For most part the answer to this question is – there is hardly anything we can configure as parameters, but there is a reason why one must learn why SQL Server uses these defaults, what are the nuances we need to be aware and how can we understand the inner working. In this post, we just take a look at the basics of what constitute a database page and what are the various functions of each of these page types. This is not exhaustive, but still the learning can never stop right :).

1. Logical pages are 8KB in size by design and default.
2. The pages are numbered sequentially from 0 to N in each file, while the size of the file decides the number of pages that it contains.
3. SQL Server can identify any page using the database-id, file-id and page-number combination. Many times there are error messages with this nomenclature for reference.
4. With all math done, it means SQL Server has 128 pages per MB.
5. When new space is allocated to a file because of expanding file, the first page of the newly created space is page# is N + 1.
6. Though shrinking removes pages, SQL Server also ensures page numbers within a file are always contiguous.

SPACE ALLOCATIONS

1. Space in a database file is managed in units called **extents.**
2. Extent is made up of 8 logically contiguous pages therefore having a capacity of 64KB.
3. 96 byte of each page is allocated for header information such as what type of page, amount of free space on page, object owing it etc.
4. Post the header information is the data rows placed serially.
5. Bottom of the page contains a row offset – there is one entry for every row inside that page and its offset from the start of the page.
6. Row Offsets are in reverse sequence from the sequence of the rows.
7. Maximum amount of data that one can store in a single row is **8060 bytes**. Whenever we are able to fit the rows in this limit – it is called In-Row Data.
   1. From SQL 2008, the page restriction has been relaxed with an concept called as Row-Overflow Data. Read more from [MSDN](http://msdn.microsoft.com/en-us/library/ms186981(v=sql.105).aspx).
8. SQL Server doesn’t allocate entire extents to tables which have small amounts of data.
9. Extents are of 2 Types:
   1. **Uniform extents**: As name suggests it is owned by a single object; all eight pages in the extent can be used only by this single object
   2. **Mixed extents**: These are shared between objects. Up to eight objects can share an extent (a.k.a each object using a single page)
10. Whenever a new table or index needs allocation it is given a mixed extents; when the table or index grows to eight pages, future allocations can use uniform extents. Between versions of SQL Server the # of pages post which uniform extents are given may vary.
11. If a table or index needs more space and is still less than 8 pages total, SQL Server must find a mixed extent with space available, else if it is 8 pages or larger, uniform extents are located.
12. When there is no mixed extents with free space, a new extent marked as mixed extent is allocated and SGAM updated accordingly.
13. Basic Page Layouts:
    1. Page 0 is the File Header
    2. Page 1 is the Page Free Space (PFS)
    3. Page 2 is GAM
    4. Page 3 is SGAM
    5. Page 6 is Differential Changed Map (DCM)
    6. Page 7 is Bulk Changed Map (BCM)
14. 2 special types of pages to record which extents have been allocated and what type it is being use for:
    1. **Global Allocation Map (GAM)** pages (always on page 2)
       1. These pages record which extents have been allocated for any type of use.
       2. GAM has a bit to indicate value 0 means in use and 1 means free extent.
       3. After header and other overhead are accounted, there are 8000 bytes or 64000 bits to cover 64000 extents.
       4. Each GAM page covers 4 GB of data and a GAM page exists in a file for every 4 GB of file size.
    2. **Shared Global Allocation Map (SGAM)** pages (always on page 3)
       1. Record which extents are currently used as mixed events and have at least one unused page.
       2. Similar to GAM, it covers 64000 extents every 4GB of data.
       3. Has a bit for each extent in the interval it covers: 1 if the extent is being used a mixed extent and has free pages, 0 if the extent isn’t being used a mixed extent or it’s a mixed extent with no free pages

Index Allocation Maps (IAM)

1. Any structure of data or index needs an IAM page.
2. IAM pages keep track of the extents in a 4 GB section of a database file.
3. An allocation unit is a set of pages belonging to a single partition in a table or index. The pages of can be of three storage types:
   1. Pages with regular **in-row data**
   2. Pages with **Large Object (LOB) data**
   3. Pages with **row-overflow data**
4. IAM page contains a 96 byte page header, followed by IAM header which contains 8 page-pointer slots.
5. IAM pages contain a set of 8 bits that map a range of extents onto a file. The IAM header has the address of the first extent in the range mapped by the IAM.
6. A bit in the IAM bitmap represents an extent in the range: 1 means extent is allocated to the object owning the IAM, 0 means the extent isn’t allocated to the object owning the IAM.
7. IAMs are allocated as needed for each object. Each of the IAM covers a possible range of 512,000 pages.

Misc. Notes

1. PFS (page 1) – keeps track of how each particular page in a file is used.
2. PFS also tracks if the page is empty, 1 to 50 percent full, 51 to 80 percent full, 81 to 95 percent full or 96 to 100 percent full.
3. There is a PFS page approximately 8,000 pages in size after the first PFS page.
4. Differential Changed Map (DCM page 6) – keeps track of which extents in a file have been modified since the last full database backup.
5. Bulk Changed Map (BCM page 7)- is used when a page in a file is in a minimally or bulk-logged operation.
6. Like the GAM and SGAM pages, DCM and BCM pages have 1 bit for each extent in the section of the file they represent.
7. There is a GAM page 64,000 extents after the first GAM page on page 2 and another SGAM page 64,000 extents after the first SGAM page on page 3.
8. Log files donot have the concepts of pages, they are a series of log entries written sequentially.
9. In the previous versions the text, ntext and image data are stored in separate pages and are not inside the Data pages.

I think I have touched most parts of how database pages exist inside SQL Server. Though there are even more subtle nuances on the page functioning, we will keep it outside of this post for later. Please refer to MSDN BOL for more information on the page architectures.