SYBASE
An SAP Company

ANALYZING INDEXES AND DATA CACHE WITH MDA

JEFF TALLMAN
DIRECTOR, TECHNICAL ENGINEERING, SYBASE

JANUARY 2012

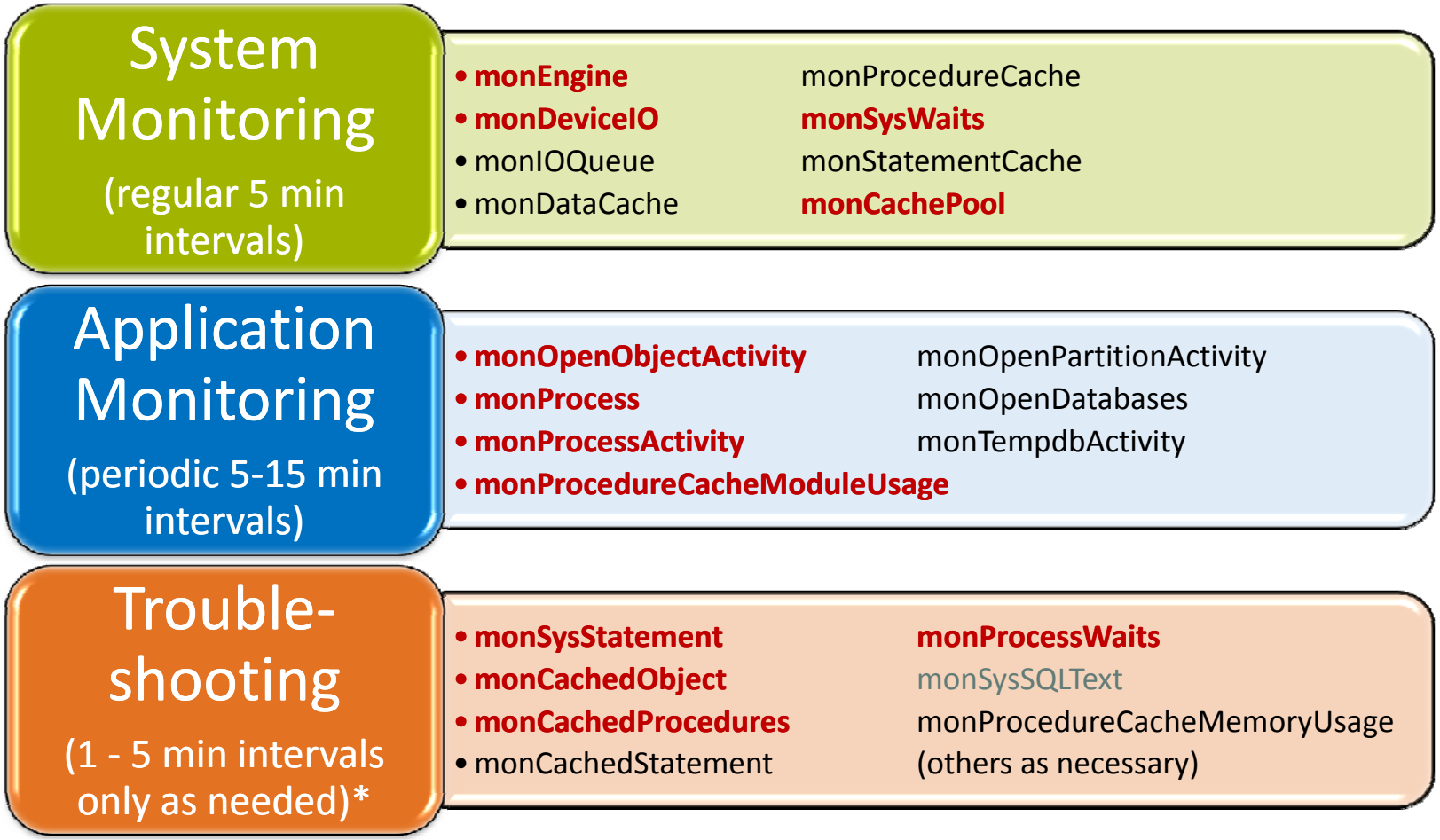
ANALYZING INDEXING, DATA CACHE, ETC

Part 2 of the Series on Using MDA

- **Data Cache Basics**
- **Tempdb & Log Cache Sizing**
- **Named Caches & Object Bindings**
  - ✓ Using LIO patterns and object volatility to determine
- **Indexing & Logic Issues**
  - ✓ How to spot potential indexing issues
  - ✓ Some key easy things to look for (table scans, unused indexes)
    - ….and what NOT to do…..
- **Proc Cache Sizing**

WHAT TO MONITOR

…and how often (some extremely useful tables highlighted)



*more frequent (e.g. 15 sec) intervals if necessary for troubleshooting purposes only

SYBASE | An SAP Company

POLLING/QUERY FREQUENCY

- **Did you notice intervals???**
  - ✓ Do not use less than 5 minute intervals other than troubleshooting
  - ✓ When troubleshooting, start with larger (e.g. 5 minute intervals) and reduce by steps until 15 sec minimum
    - Steps: 5 min, 2 min, 1 min, 30 sec, 15 sec…..10?....5?
    - monLocks is one exception (constant polling may be necessary)
    - Rational for 15 secs
      - based on time it takes to collect the data (e.g. walk buffer chains, engine statement events), generate results and send to client…and save to disk.
- **Polling too frequently adds to MDA performance impact**
  - ✓ Some tables such as cache detail tables (monCachedProcedures, monCachedObject, monCachedStatement) require scanning the proc headers or cache buffer chains and aggregating on the fly
  - ✓ This can take quite a while…and increase contention on cache spinlocks
    - Alternative is to “store” an aggregate which impacts every query even worse
  - ✓ Polling too frequently can have significant impact or worse
    - To be fixed by eliminating the spinlock grab…effectively a dirty read of cache contents

MDA CONFIGURATIONS

- **Configurations are dynamic**
  - ✓ Except SQL batch capture
  - ✓ So turn them on when needed - and then back off when done
- **Object level stats**
  - ✓ per object statistics active, object lockwait timing
  - ✓ Can make DES contention worse
    - Object stats are stored in DES/IDES/PDES
    - Concurrent queries will need to update DES
    - Be careful if DES contention is >7-10%
  - ✓ Look for recommendations to dbcc tune(desbind) in sp_sysmon
    - If present, give serious consideration to using on hot tables and hot procs before enabling object stats
    - DES is global where statement info (pipes) are per engine - hence the contention
- **Statement pipes**
  - ✓ Primarily sql text & plan text
    - Statement pipe to a much lesser degreee
  - ✓ Memory impact can have degradation on low memory systems

System Monitoring

```
enable monitoring = 1
errorlog pipe active = 1
errorlog pipe max messages = 1000
deadlock pipe active = 1
deadlock pipe max messages = 100
wait event timing = 1
SQL batch capture = 1
max SQL text monitored = 2048
```

Application Monitoring

```
object lockwait timing = 1
per object statistics active = 1
enable stmt cache monitoring = 1
process wait events = 1
```

Troubleshooting

```
statement pipe active = 1
statement pipe max messages = 25000
statement statistics active = 1
sql text pipe active = DEFAULT
sql text pipe max messages = DEFAULT
plan text pipe active = DEFAULT
plan text pipe max messages = DEFAULT
```

GOTCHA'S WITH 3RD PARTY TOOLS

- **This is not intended as a deterrent….**
  - ✓ …author personally likes several of them…but a few words of caution
- **Many just have a single "polling interval"**
  - ✓ Often set ridiculously low (1 second or 10 seconds)
  - ✓ Yes - it makes the pretty animated screens demo well
  - ✓ But…it can *severely impact* your server
    - Even if there isn't any contention - it takes time (all CPU time) to collect all of the data, aggregate it (if necessary) and return the results.
    - ….so….very rapid polling will drive CPU higher
- **Many will turn on every config when not necessary**
  - ✓ Common culprits are SQL text pipe and Plan text pipe
  - ✓ Currently executing queries do NOT need the pipes
    - monProcessSQLText & monProcess (for proc line currently executing)
  - ✓ After launching the tool, manually turn off the cfgs for SQL/Plan pipes

SOME ANALYSIS TIPS

Logic to Help You

- **Don't expect/ask for an "MDA Query" - no such thing**
  - ✓ Almost all analysis will require at least 2-3 steps to create the deltas between sample values, etc.
  - ✓ Soo….the analysis queries will most often be stored procedures
- **How do you do the deltas quickly with multiple keys**
  - ✓ E.g. SPID+KPID+SampleTime
  - ✓ Answer 1: Create a SampleInterval table

```
Select sample_interval=identity(20), SampleTime
  into #cur_intervals
  from monProcessActivity
```

    - Then join in queries ala

```
Select a.SampleTime, a.SPID, a.KPID,
          CPUTime=a.CPUTime - isnull(b.CPUTime,0)
          Transactions=a.Transactions - isnull(b.Transactions,0)
  from monProcessActivity a, monProcessActivity b,
          #cur_intervals i1, #cur_intervals i2
  where a.SPID=b.SPID and a.KPID=b.KPID
    and a.SampleTime=i1.SampleTime
    and i1.sample_interval=i2.sample_interval+1
    and i2.SampleTime=b.SampleTime
```

  - ✓ Answer2: similar to above, but do it inline with small numbers of cols

```
Select sample_num=identity(20), SPID, KPID, SampleTime, LogicalReads
  into #ordered_LIOs
  from monProcessActivity
  order by SPID, KPID, SampleTime
```

    - Then join in queries (similar to above)

DATA CACHE BASICS

BUFFER POOLS, CACHE PARTITIONS, CACHE STRATEGY, ETC.

SEVERAL KEY QUESTIONS

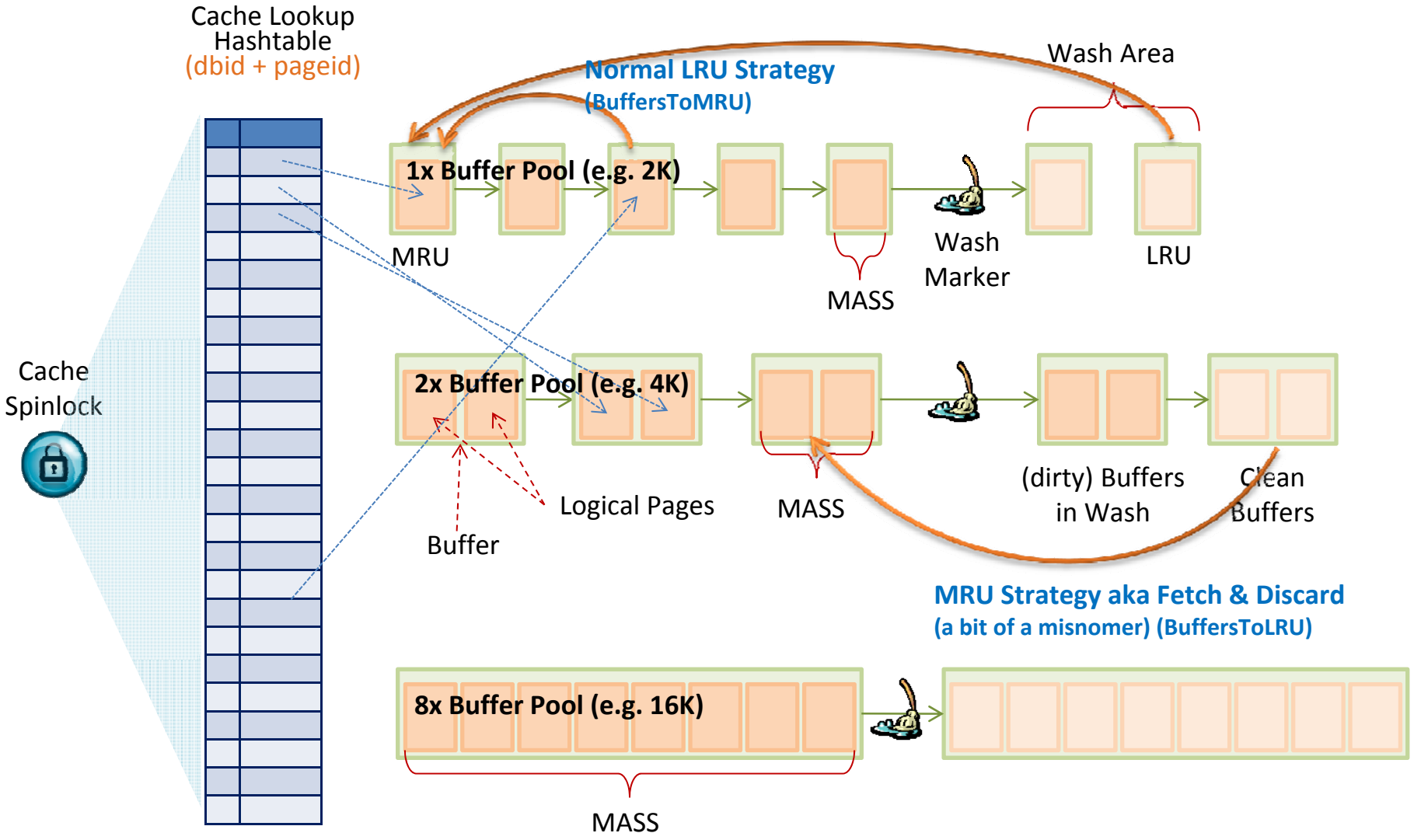
Common Questions We Often Face

- **Are PhysicalReads Due to Cache Configuration??**
  - ✓ Do we have the right mix of caches (and types)
- **Are PhysicalReads/LogicalReads a sign of poor indexing???**
  - ✓ …or bad query plan….can we tell the difference???
  - ✓ …or is it due to deliberate scattering via the clustered index
    - Which may lead to lower cache efficiency due to data scattering requiring more pages for same rows
    - For example, lets say you are doing claims processing…and….
      - Claims are clustered by member ID
      - Claims are clustered by claim date
- **Which Objects are having the most issues?**

SPINLOCKS & HASH BUCKETS

- **Locating items in memory is usually done via a hash table**
  - ✓ Size of the hash table may or may not be configurable
  - ✓ Each entry in the hash table is considered a hash bucket
  - ✓ Item attribute is hashed according to some hash function
  - ✓ Hash value determines which hash bucket is used
  - ✓ Hash bucket likely covers more than one value
  - ✓ Result is a serial scan through the hash chain associated with hash bucket
- **Modifying the hash chain requires grabbing the spinlock**
  - ✓ Need to find which spinlock guards that hash bucket
  - ✓ Grab the spinlock, add/remove item, release spinlock
- **Reading the hash chain may require grabbing the spinlock**
  - ✓ If you want a consistent picture vs. a 'dirty read'
- **Tuning ASE often is tuning hash buckets/spinlocks**
  - ✓ Understanding what you can change and what you can't for each
  - ✓ Understanding how to control concurrency
    - Upper limit of contention = concurrent processes = engines online

CACHE MANAGEMENT (DEFAULT)



UseMRU = PoolSize * 0.5 >= PoolSize - PagesScanned
(when the pagest to be scanned would use more than 50% of the configured PoolSize, the MRU strategy can be chosen by the optimizer for pages ***read from disk***)

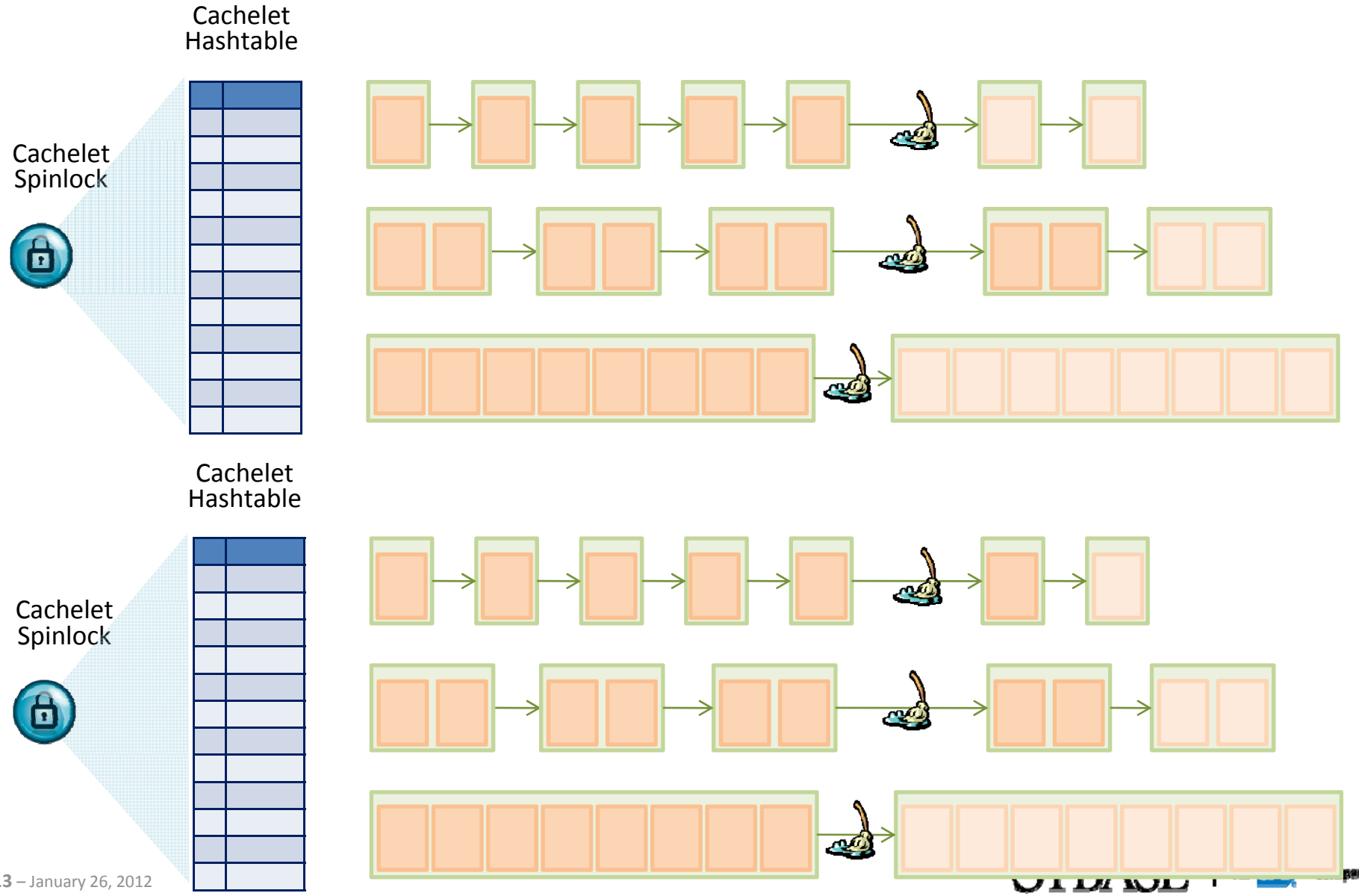
CACHE MANAGEMENT

Key Things to Remember When Monitoring the System

- **Changes to what is in cache is reflected in cache hash table**
  - ✓ The cache hash table is part of the cache overhead when sizing a cache
  - ✓ Changes to the cache hash table require grabbing the cache spinlock
    - Essentially, every physical read and every new page allocated (incl page splits)
    - Every new #temp table created
- **All Physical I/O's are done in MASS units**
  - ✓ A large I/O can be 4K, 8K or 16K
  - ✓ A MASS is just a group of contiguous pages on disk read in during a single IO operation
  - ✓ When writing, all pages are written - whether dirty or not
  - ✓ To block further changes when writing (DMA access), rather than using a spinlock, each cache buffer has a "MASS bit"
    - Rather than spinning - you sleep…..spinlock = spin mutex; mass bit = sleep mutex
- **Logical Reads count as a cache hit**
  - ✓ In a strict (default) cache replacement, this results in the buffer being relinked to the MRU end of the MRU→LRU chain
  - ✓ Any changes (relinkage) to the buffer chain requires grabbing the cache spinlock
    - Essentially any logical read will result in a buffer relink which means spinlock grab

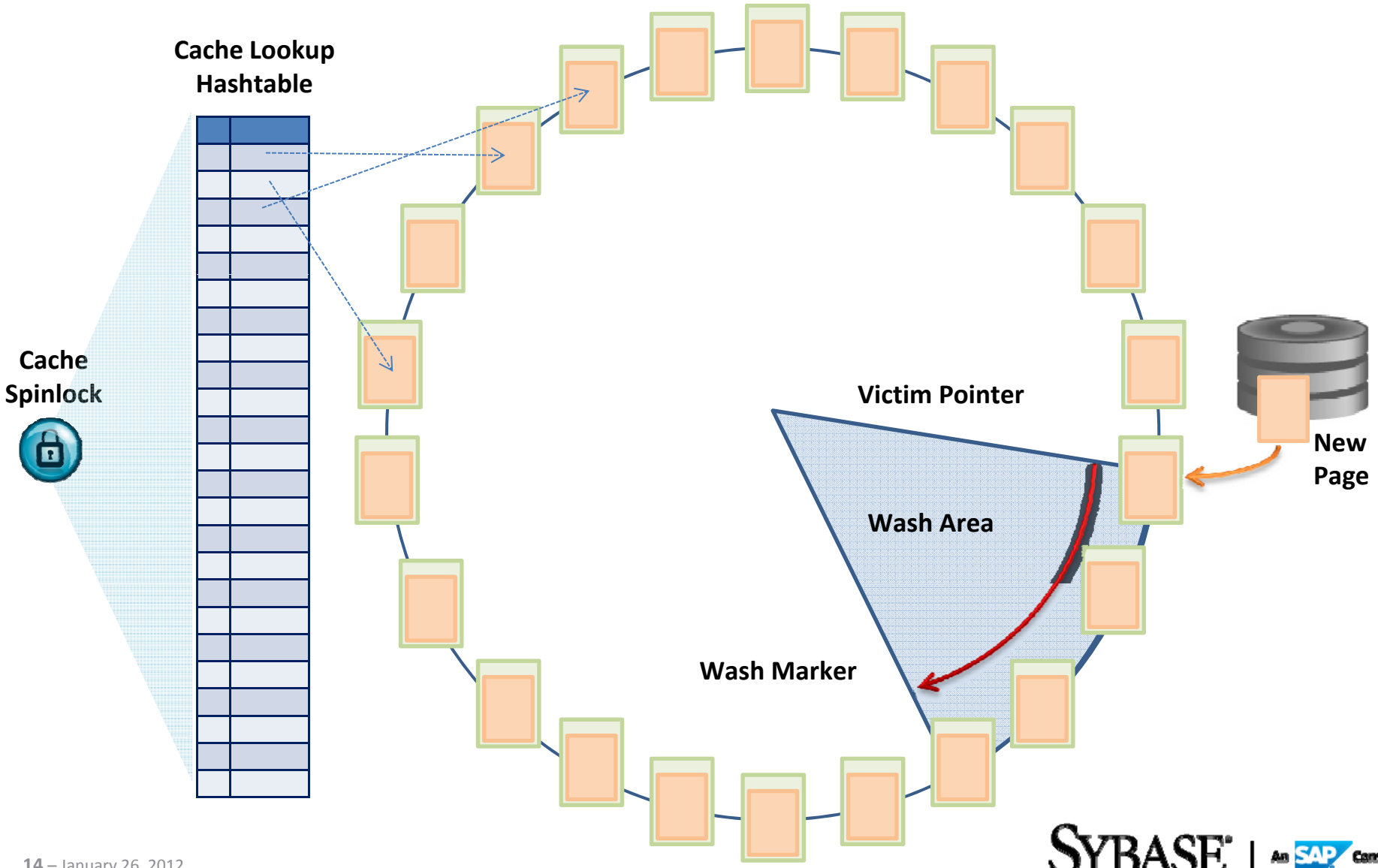
CACHE PARTITIONING

One Possible Answer to Cache Spinlock Contention



RELAXED CACHE REPLACEMENT

A Better Solution to Cache Spinlock Contention???



SYBASE | An SAP Company

RELAXED CACHE REPLACEMENT TIPS

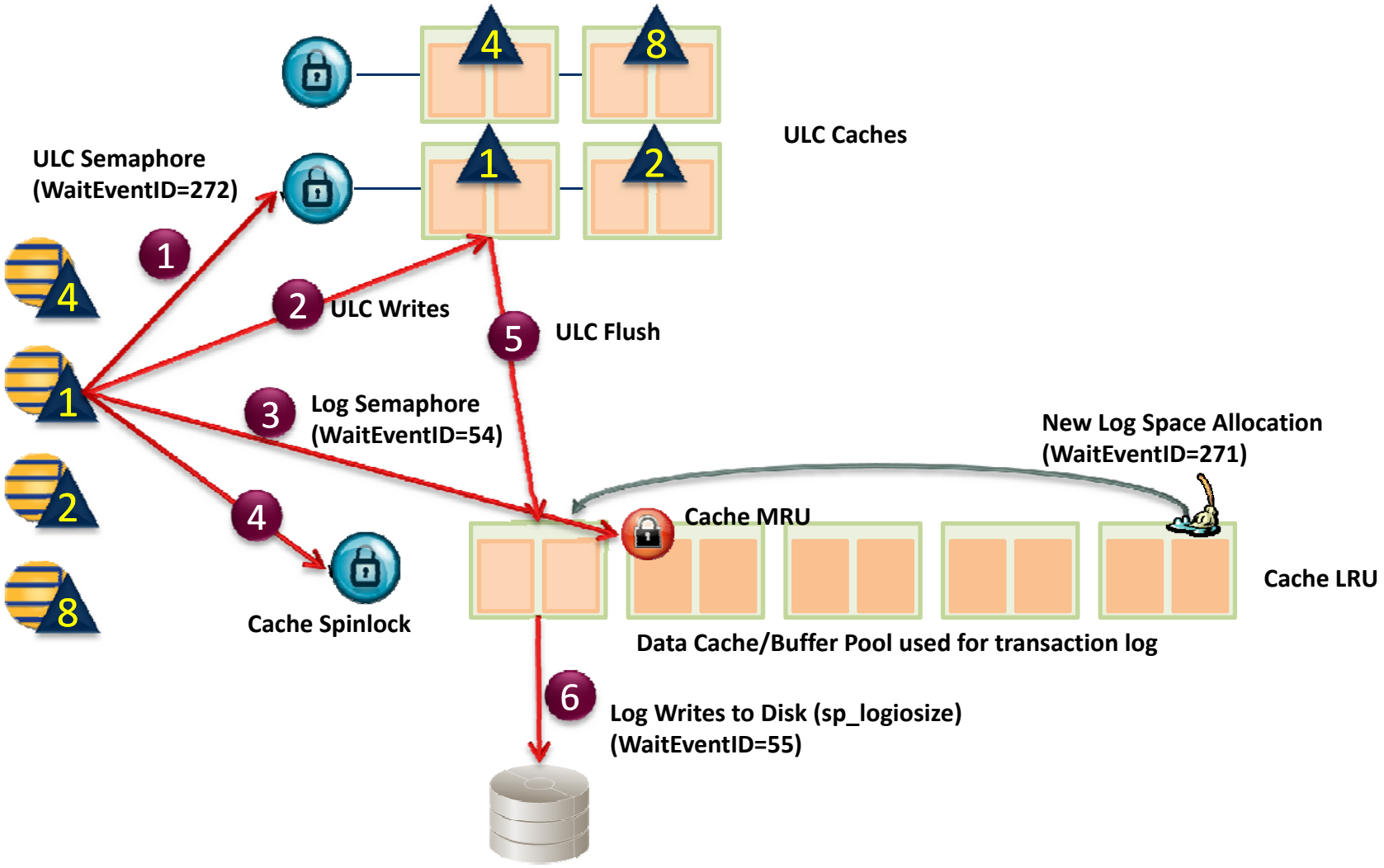
- **The trade-off:**
  - ✓ Reduces LRU→MRU relinkage driven spinlock contention
  - ✓ Increases physical IO if a lot of inserts as victim pushes wash marker around the ring
    - Wash is much more likely to hit recently modified pages since they are not moved to MRU
  - ✓ Potential increase in time it takes to find a clean page (cache stall)
  - ✓ Can decrease cache effectiveness as page cache overwrites are not dependent on how often re-read/re-written
- **Tips:**
  - ✓ Can be used for any db if db can be fully cached
    - If using multiple tempdb's, smaller OLTP tempdbs might benefit
  - ✓ Can be used for any table if the table can be fully cached and the table does not have:
    - a lot of inserts
    - updates that cause page splits
    - Consider DOL tables with exp_row_size
  - ✓ A good choice for indexes if index fully cachable and low turnover
    - Consider using a smaller fill factor though to reduce new page creations due to updates to index key values
      - Index key value updates → delete followed by insert (of key values)
    - Only if spinlock contention is a concern

REDUCING SPINLOCK CONTENTION:

The Choice Is Yours…Some Will Work Better In Some Situations…Others in Other Situations

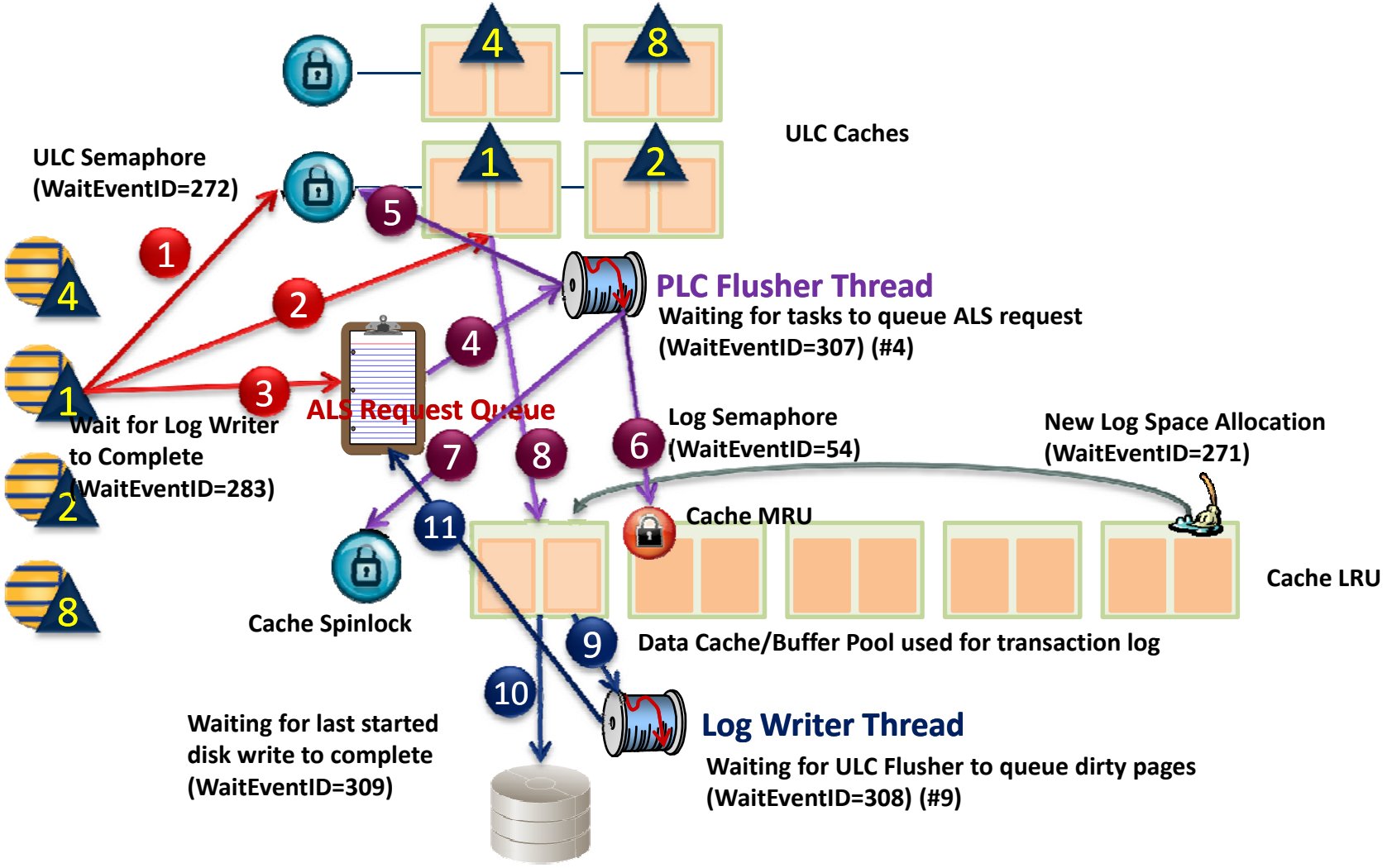
- **Increase number of spinlocks**
  - ✓ Decrease the spinlock ratio
  - ✓ Increase number of cache partitions (up to engine count)
- **Change cache replacement strategy**
  - ✓ Used relaxed cache replacement strategy
- **Use multiple different named caches**
  - ✓ E.g. split volatile tables and indexes into separate caches

LOG CACHE & LOG WRITES



SYBASE | An SAP Company

LOG WRITES WITH ASYNC LOG SERVICE



SYBASE | An SAP Company

TEMPDB & LOG CACHES

MONITORING CACHE SIZING

DATA CACHE CONFIGURATION

- **Too few DBA's configure data cache correctly**
  - ✓ Almost everything is in default data cache
  - ✓ There may be a log cache - usually oversized
  - ✓ There may be tempdb cache
- **A good starting configuration should minimally include:**

Named Cache	Sizing	Number	Partition	Cache Strategies	HK Ignore
Log cache	50-100MB (normal) 150-200MB (XXL SMP)	1-3	No	Log only	(implicit)
System tables	200MB-500MB	1	No or few	Relaxed	(implicit)
Tempdb caches	250MB-500MB (normal) 500MB-1GB (XXL SMP)	1 per tempdb/ tempdb group	YES*	HK Ignore Cache or Relaxed if ~100% cached	YES
Restriction	50-100MB (normal) 256-500MB (BLOB)	1	YES*	Strict (default)	(maybe)
Reference	50-100MB	1	No or few	Relaxed	(implicit)
Hot Tables (static size – update intensive) such as key sequence tables	10-50 MB	1	Few or more*	Relaxed	(implicit)
Hot Tables/Indexes	Size of volatile data	As necessary	Few or more*	Strict (default)	NO
Application Specific	As necessary	1-3	YES*	(depends)	NO
Default Data Cache	(most of memory)	(1)	YES*	(default)	NO

YOU DON'T BELIEVE ME???

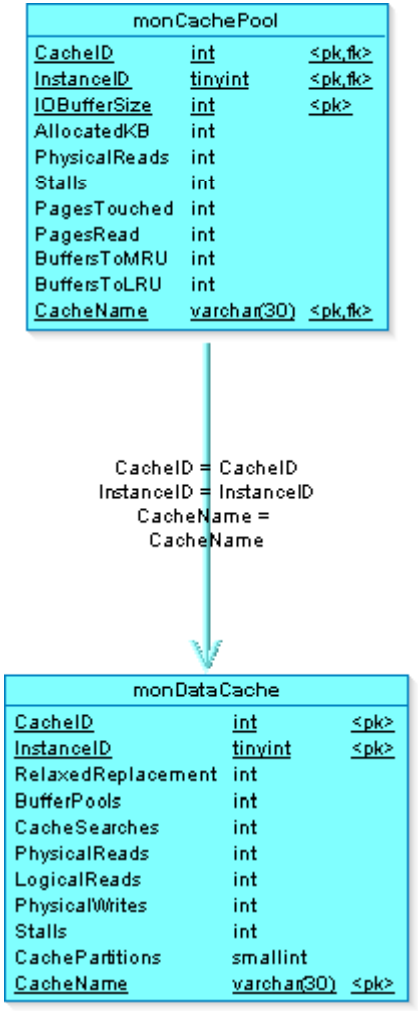
From monCachedObject….and you thought the metadata cache had you covered!!!

IndexID	ObjectID	DBName	ObjectName	PartitionName	Min(CachedKB)	Max(CachedKB)	Min(SizeKB)	Max(SizeKB)	Min(CachedPct)	Max(CachedPct)
0	8	batch_db1	**syslogs**	syslogs_8	8	822,620	32	822,620	25.0	100.0
0	8	batch_db3	**syslogs**	syslogs_8	8	603,328	32	603,328	25.0	100.0
0	8	batch_db2	**syslogs**	syslogs_8	8	70,752	32	70,752	25.0	100.0
255	21	master	**tsysattributes**	tsysattributes_21	4	46,488	4	46,488	100.0	100.0
0	5	lookup_db	**sysprocedures**	sysprocedures_5	668	32,052	83,168	83,404	0.8	38.4
0	5	stage_db1	**sysprocedures**	sysprocedures_5	80	26,968	202,372	202,372	0.0	13.3
0	5	work_db1	**sysprocedures**	sysprocedures_5	100	24,792	151,740	151,740	0.1	16.3
0	5	sybsystemprocs	**sysprocedures**	sysprocedures_5	1,260	20,752	113,536	113,636	1.1	18.3
0	5	stage_db3	**sysprocedures**	sysprocedures_5	20	12,680	24,644	24,644	0.1	51.5
0	8	master	**syslogs**	syslogs_8	8	12,048	32	12,048	25.0	100.0
0	5	batch_db1	**sysprocedures**	sysprocedures_5	4	8,740	49,128	49,140	0.0	17.8
0	6	lookup_db	**syscomments**	syscomments_6	4	8,304	7,820	8,304	0.1	100.0
0	3	main_db	**syscolumns**	syscolumns_3	900	7,256	15,836	15,836	5.7	45.8
2	3	main_db	**csyscolumns**	csyscolumns_3	580	6,500	6,232	6,500	9.3	100.0
0	5	work_db3	**sysprocedures**	sysprocedures_5	16	5,656	13,492	13,492	0.1	41.9
0	1	main_db	**sysobjects**	sysobjects_1	480	3,652	3,624	3,652	13.2	100.0
2	5	work_db1	**csysprocedures**	csysprocedures_5	12	2,432	8,168	8,168	0.1	29.8
2	5	stage_db1	**csysprocedures**	csysprocedures_5	16	2,412	11,052	11,052	0.1	21.8
0	9	main_db	**sysprotects**	sysprotects_9	1,160	3,340	3,300	3,340	35.2	100.0
0	24	work_db1	**sysstatistics**	sysstatistics_24	72	2,148	2,264	2,300	3.2	93.4
0	3	batch_db2	**syscolumns**	syscolumns_3	4	2,032	2,924	2,924	0.1	69.5
0	5	master	**sysprocedures**	sysprocedures_5	28	1,780	5,700	5,700	0.5	31.2
0	21	master	**sysattributes**	sysattributes_21	60	1,808	1,728	1,808	3.5	100.0
2	5	lookup_db	**csysprocedures**	csysprocedures_5	60	1,760	4,416	4,416	1.4	39.9
0	24	stage_db1	**sysstatistics**	sysstatistics_24	296	1,840	1,884	1,924	15.7	95.6

TYPICAL DBA CACHE MONITORING

monDataCache, monCachePool

- **The Problems**
  - ✓ Focusing on Stalls, PhysicalReads, Hit Ratio
  - ✓ Not understanding the differences in counters:
    - e.g Trying to get PhysicalReads in monCachePool to equal PhysicalReads in monDataCache
- **The Reality**
  - ✓ Stalls, PhysicalReads, Hit Ratio - all can be misleading stats
    - Some of the other stats can be more important to resolving the problem
    - Hit Ratio notoriously unreliable today with high memory systems
  - ✓ The counters in monCachePool refer to the pages *currently* in the cache pool
    - If pages are removed from the pool, the counters are decremented by the page stats



TEMPDB CACHE OR IMDB TEMPDB

…to be or not to be…..imdb….

LogicalName	Reads	APFReads	APF_Pct	Writes	TotalIOs	Write_Pct	DevSemaphoreRequests	IOTime_ms	msPerIO
dv16tmp	193	3	1	2,541,486	2,541,679	99	0	8,635,100	3
dv08log	4,483	0	0	1,739,428	1,743,911	99	977	4,088,600	2
dev_dbname_01_log	2,416	0	0	1,466,073	1,468,489	99	0	2,732,600	1
dv27dat	434,076	89,055	20	810,705	1,244,781	65	0	9,273,500	7
dv19log	3,272	73	2	1,140,213	1,143,485	99	0	2,494,600	2
dv48tmp	0	0	0	942,282	942,282	100	0	23,563,700	25
dv58tmp	0	0	0	811,819	811,819	100	0	6,424,800	7
dv49tmp	5	0	0	808,135	808,140	99	0	17,661,600	21
dv60tmp	1	0	0	802,349	802,350	99	0	799,300	0
dv61tmp	0	0	0	764,290	764,290	100	0	935,900	1
dv54dat	271,754	60,068	22	476,876	748,630	63	0	17,518,000	23
dv02tmp	0	0	0	622,277	622,277	100	0	2,156,200	3
dv07dat	223,083	74,401	33	353,299	576,382	61	0	4,733,900	8
dv05dat								4,160,500	12
dv42dat								6,118,400	26
dv23dat								2,138,200	11
dv38dat								1,358,100	8
dv01webdb	146,211	103,084	70	10,185	156,396	6	0	1,590,300	10
dv55dat	85,575	0	0	67,361	152,936	44	0	4,198,700	27
sybsecurity_log1	0	0	0	139,992	139,992	100	0	719,300	5
dv25dat	76,002	29	0	49,655	125,657	39	0	1,751,900	13
master	1,402	1,092	77	111,237	112,639	98	0	521,300	4
dv24dat	23,065	19	0	88,382	111,447	79	0	587,800	5
dv26dat	40,816	9	0	39,221	80,037	49	0	746,200	9
sybsecurity_data1	52	2	3	74,125	74,177	99	0	429,400	5

…with a few more KB (~200 pages) in tempdb cache, the answer is YES!!!

…and eliminate 7,292,638 writes out of 14,280,289 (50%) (System CPU, disk I/O structures, OS AIO resources, HBA bandwidth)….not to mention save 10's of GB of file system space/SAN storage

CPU and I/O accounting flush interval (sp_configure)….writes are noise level (<1%) - but impacts master db log…and if master log fills - you are in a world of hurt!!!

TEMPDB CACHE OR IMDB TEMPDB

A Common Application Question Today

- **Tempdb Cache**
  - ✓ A lot of times, it can be created large enough to "fully cache" tempdb
  - ✓ …however, writes to tempdb still get flushed to disk
    - Log Writes are the big issue as they are synchronous to transaction
      - Although we do a lot less of this now in 15.0.2+
    - Writes to data devices in tempdb still happen (less so in 15.5)
      - If HKGC hits tempdb table dirty pages in the cache (can be disabled)
      - Checkpoints in tempdb (yep)
        - Think about it - if we are going to simply truncate the log, if we don't flush the dirty pages, what do we do when the cache fills??? (wash marker & cache stalls??)
        - Physical reads in tempdb happen from pages written to disk by checkpoint or HKGC
  - ✓ Any write operation puts your SPID to sleep
    - So that tempdb log flush (session tempdb ULC) will put you to sleep
      - Can happen more than once if session tempdb ULC not large enough
    - Luckily, checkpoint/HKGC flushes don't affect you…sort of
      - If you are writing (or trying to) in the same MASS as a checkpoint or HK, you will see MASS contention (common)
    - Even if the write was to cached UFS devices, ASE doesn't know the write is super fast
      - you go to sleep and then wakeup and get put on back of run queue…behind the dozens of other users.
- **IMDB Tempdb avoids all of this**
  - ✓ Application response time can be faster

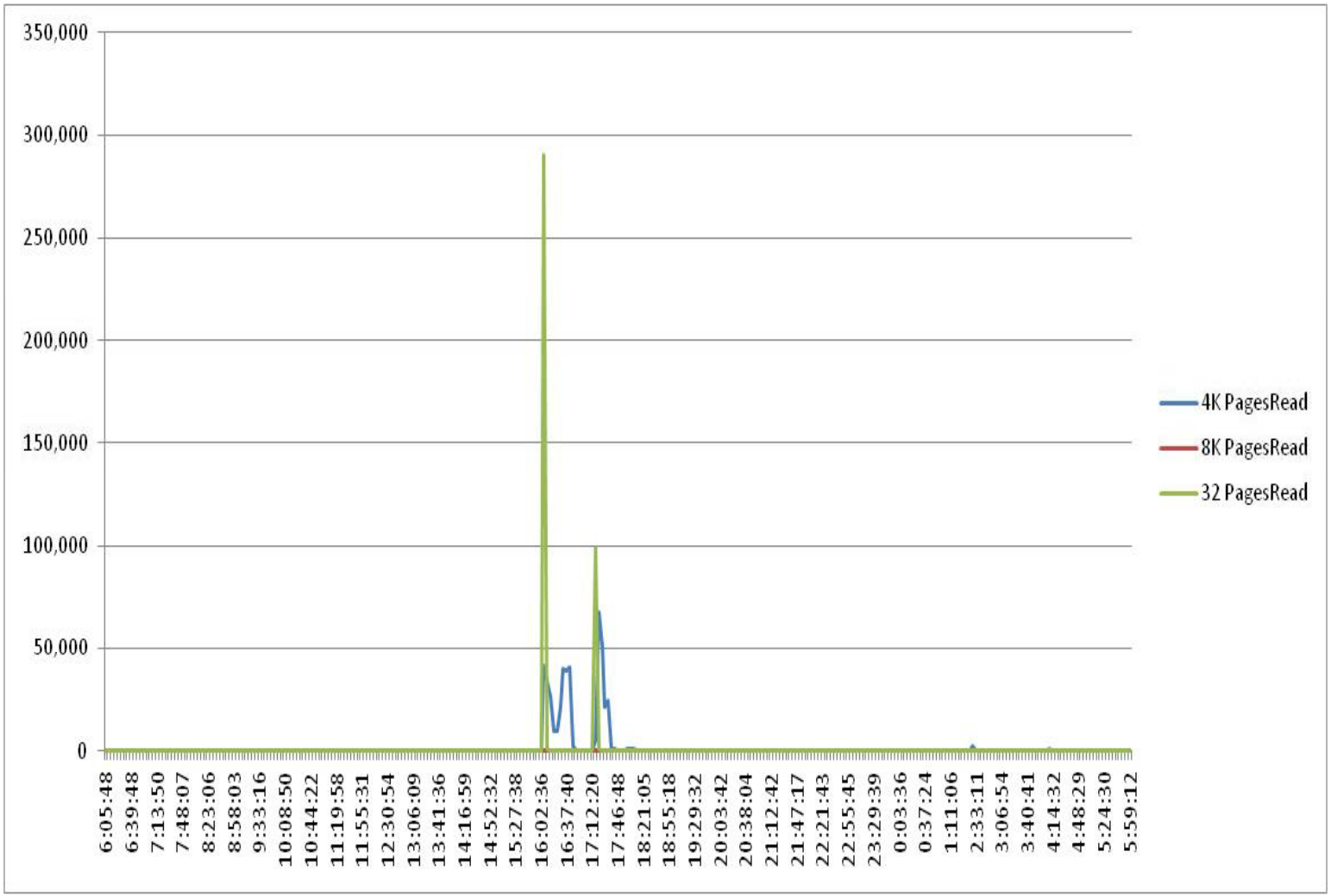
TEMPDB….A HARDER ONE

LogicalName	Reads	APFReads	APF_Pct	Writes	TotalIOs	IOTime_ms	msPerIO
tempdb_d_001	487,263	1,405	0	3,623,453	4,110,716	19,833,400	4.8
tempdbrr9_d_001	18,777	0	0	2,147,083	2,165,860	3,371,700	1.5
tempdbrr2_d_001	76,239	504	0	1,216,982	1,293,221	1,324,400	1.0
tempdbrr11_d_001	21,127	0	0	776,769	797,896	715,200	0.8
tempdb_d_002	6,557	0	0	775,960	782,517	812,300	1.0
tempdbbat_d_001	24,449	2	0	662,144	686,593	3,287,400	4.7
tempdbrr12_d_001	23,408	0	0	538,402	561,810	97,562,611	173.6
tempdbrr2_d_002	6,043	0	0	454,036	460,079	592,200	1.2
tempdbrr10_d_002	507	0	0	455,814	456,321	812,100	1.7
tempdbbat_d_002	116	0	0	386,096	386,212	453,500	1.1
tempdbrr10_d_001	20,674	0	0	355,932	376,606	265,100	0.7
tempdb_l_001	4,080	0	0	336,216	340,296	8,186,400	24.0
tempdbrr8_d_002	788	0	0	316,510	317,298	223,800	0.7
tempdbsa_data01	25,331	0	0	272,868	298,199	329,400	1.1
tempdbrr8_d_001	20,456	0	0	277,595	298,051	218,500	0.7
tempdbrr3_d_001	36,920	44	0	166,182	203,102	200,300	0.9
tempdbrr_d_001	36,188	43	0	164,220	200,408	224,300	1.1
tempdbrr6_d_001	37,303	62	0	160,199	197,502	184,600	0.9
tempdbrr11_d_002	263	0	0	192,748	193,011	183,300	0.9
tempdbrr7_d_001	36,415	26	0	154,845	191,260	162,800	0.8
tempdb_l_002	1,024	0	0	164,880	165,904	3,961,200	23.8
tempdbrr12_d_002	443	0	0	141,049	141,492	2,175,800	15.3
tempdbrr5_d_001	21,200	2	0	58,419	79,619	51,700	0.6
tempdbrr9_d_002	147	0	0	76,584	76,731	88,400	1.1
tempdbrr4_d_001	21,213	3	0	53,556	74,769	48,700	0.6
tempdbrr2_l_001	4,087	0	0	36,254	40,341	836,800	20.7
tempdbrr3_l_001	4,040	0	0	29,348	33,388	395,500	11.8
tempdbrr6_l_001	4,045	0	0	27,486	31,531	290,400	9.2

27h:06m:02s !!!!

…MAYBE NOT SO HARD

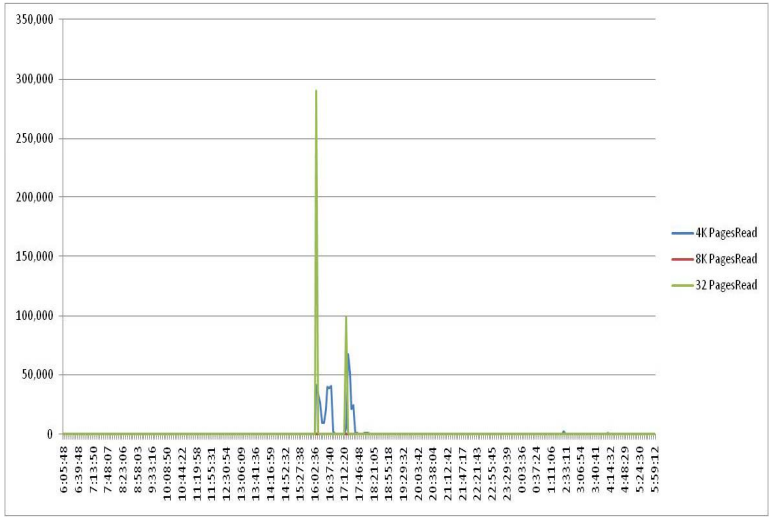
MDA making life simple: monCachePool (tempdb cache)



SYBASE | An SAP Company

WHAT TO DO

- **Check for bad query**
  - We are scanning 300,000 pages in for a #temp table!!! Ouch!!!
  - Soo…either we are scanning a table multiple times (unlikely) or the temp table is >1.2GB in size
    - Whoa!!! Scan is 1.2GB plus cache size
- **Split into two caches**
  - OLTP cache for tempdb can be much smaller
  - Batch cache - eh…size it to what memory we feel like giving it and let it spill to disk

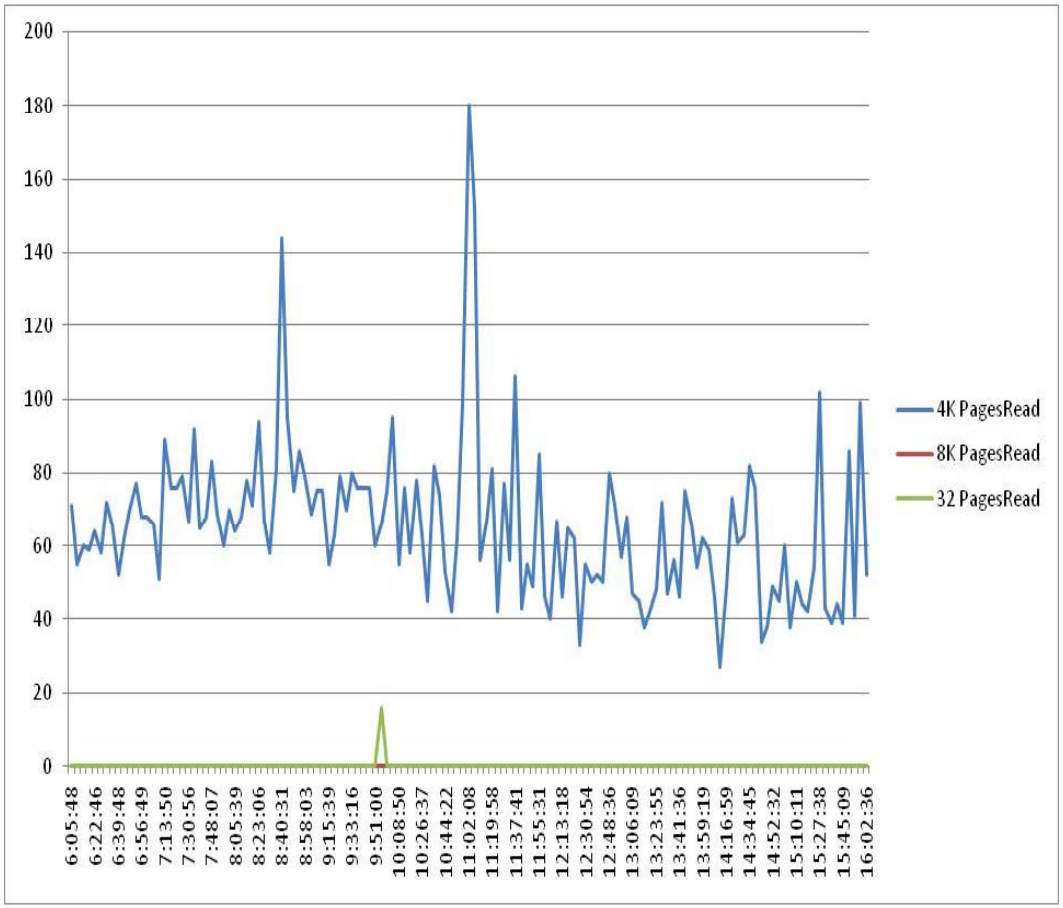


To cover the 300K pages for the batch process, we'd have to add 1.2GB to the cache just in 32K pool. Hmmm…a bit expensive.

Focusing on 4K pool, a ~75k page addition would still be 300MB…still a tad pricey…

JUST THE OLTP TIMEFRAME

monCachePool: tempdb cache



If we add ~200 pages to 4K pool (200 buffers) and 32 pages to 32K pool (1 buffer), we might be able to avoid physical reads. In MB, we just add at total of 2MB to the cache - 1MB to 4K pool and 1MB to 32K pool.

To cover the 300K pages for the batch process, we'd have to add 1.2GB to the cache. Hmmm…a bit expensive.

MONCACHEPOOL & TEMPDB CACHE

PagesRead=PhysicalReads * (IOBufferSize/@@maxpagesize)

Results

	CacheID	InstanceID	IOBufferSize	AllocatedKB	PhysicalReads	Stalls	PagesTouched	PagesRead	BuffersToMRU	BuffersToLRU	CacheName
1	0	0	4,096	2,883,576	9,941	0	7,221	9,941	9,932	9	default data cache
2	0	0	32,768	262,144	33,096	0	64,427	264,768	33,792	230,976	default data cache
3	1	0	4,096	10,240	0	0	0	0	0	0	ec_keys_cache
4	2	0	4,096	563,200	0	0	0	0	0	0	imdb_big_cache
5	3	0	4,096	143,360	0	0	81	0	0	0	imdb_small_cache
6	4	0	4,096	51,200	0	0	0	0	0	0	log_cache
7	4	0	32,768	102,400	0	0	0	0	0	0	log_cache
8	5	0	4,096	102,400	1	0	1	1	1	0	system_cache
9	6	0	4,096	393,216	361	0	723	361	361	0	tempdb_cache
10	6	0	32,768	131,072	0	0	631	0	0	0	tempdb_cache

Results Messages

Line 1 Column 27 | 10 rows

PhysicalReads in tempdb could be system table reads or MJ spills to disk…most likely the former (system tables)

Pages added due to inserts and dropped (select/into's used 32K pool & large IO)

RecentlyUsedKB = PagesTouched*@@maxpagesize/1024

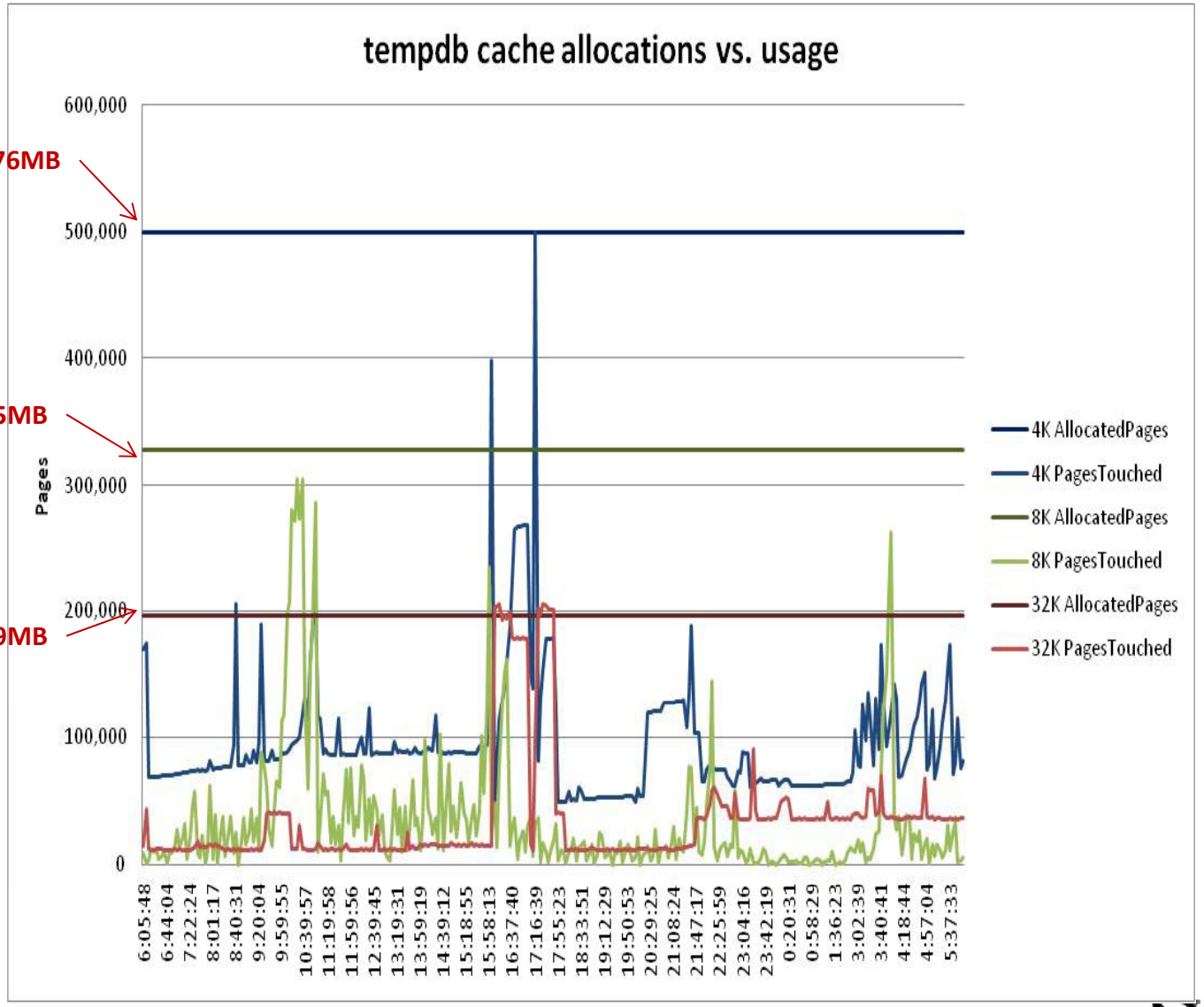
Results

	CacheID	InstanceID	IOBufferSize	AllocatedKB	PhysicalReads	Stalls	PagesTouched	PagesRead	BuffersToMRU	BuffersToLRU	CacheName
1	0	0	4,096	2,883,576	9,995	0	6,223	9,995	9,986	9	default data cache
2	0	0	32,768	262,144	46,289	0	59,536	370,312	58,160	312,152	default data cache
3	1	0	4,096	10,240	0	0	0	0	0	0	ec_keys_cache
4	2	0	4,096	563,200	0	0	0	0	0	0	imdb_big_cache
5	3	0	4,096	143,360	0	0	81	0	0	0	imdb_small_cache
6	4	0	4,096	51,200	0	0	0	0	0	0	log_cache
7	4	0	32,768	102,400	0	0	0	0	0	0	log_cache
8	5	0	4,096	102,400	1	0	1	1	1	0	system_cache
9	6	0	4,096	393,216	479	0	556	479	479	0	tempdb_cache
10	6	0	32,768	131,072	0	0	0	0	0	0	tempdb_cache

Results Messages

Line 1 Column 27 | 10 rows

IS THE TEMP CACHE OVERSIZED???



Tempdb cache of 2000MB…some notes:

Some heavy logged DML statements at 09:30 to 11:00

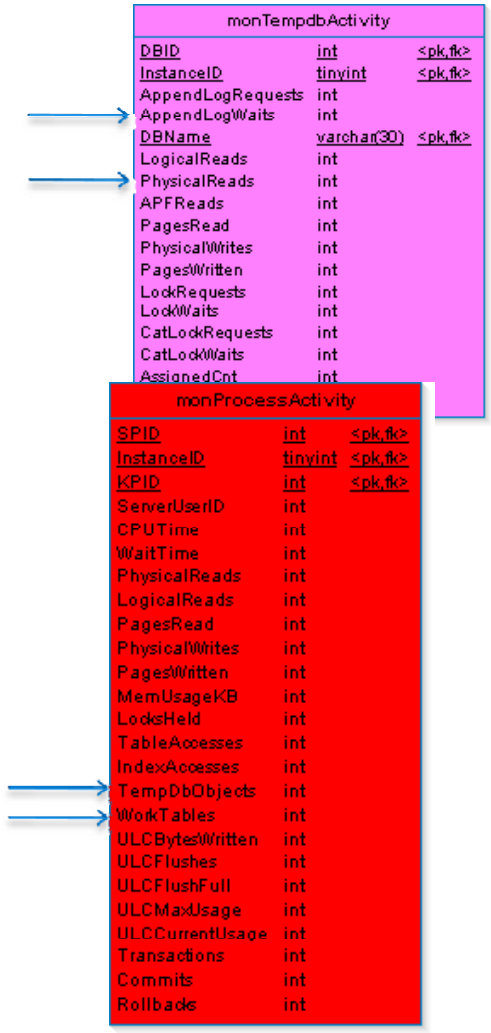
Apparent batch processing at 16:00 to 18:00

…if not for that, it would all fit in <500MB - saves 1500MB in cache

MULTIPLE TEMPDB'S & CACHES

monTempdbActivity, monProcessActivity

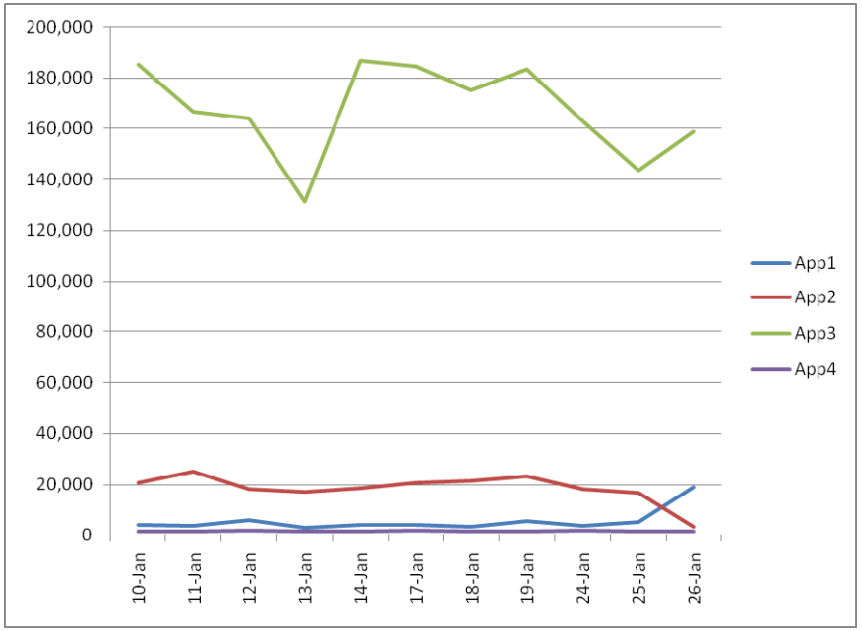
- **Catalog contention should be gone**
  - ✓ 15.0.2 with RLC
- **Log semaphore contention still there**
  - ✓ Contention%=AppendLogWaits/AppendLogRequests
  - ✓ Increase 'session tempdb log cache size'
    - Recommend 32KB (min) → 128KB (max)
  - ✓ …..or add another tempdb to tempdb group
- **Writes should be a lot less (especially 15.5+)**
  - ✓ No more checkpoint flushes of dirty pages
  - ✓ Session tempdb log cache
  - ✓ No more SLR's or synchronous page splits
  - ✓ Make sure directio=false and dsync=false (cached UFS)
- **Configuration**
  - ✓ Turn off HK Wash in all tempdb caches
    - Add 'cache status = HK ignore cache' in cfg file
  - ✓ 3-4 small tempdbs for OLTP
    - Separate named caches for each to reduce spinlock contention during #temp creation/dropping
    - Candidates for IMDB or relaxed cache strategy
    - Watch PhysicalReads for cache sizing…ideally 0
  - ✓ 1-2 tempdbs for batch processes
    - Can share named cache with others and sa tempdb (below)
    - PhysicalReads likely due to table size vs. cache size
  - ✓ 1 tempdb for SA (update stats, etc)



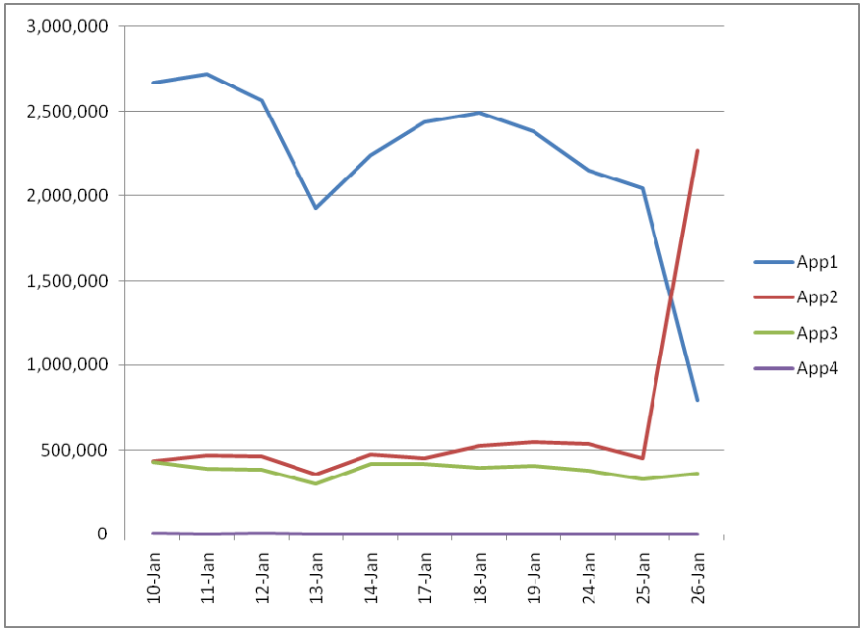
TEMP TABLES VS. WORK TABLES

monProcessActivity

TempDbObjects



WorkTables



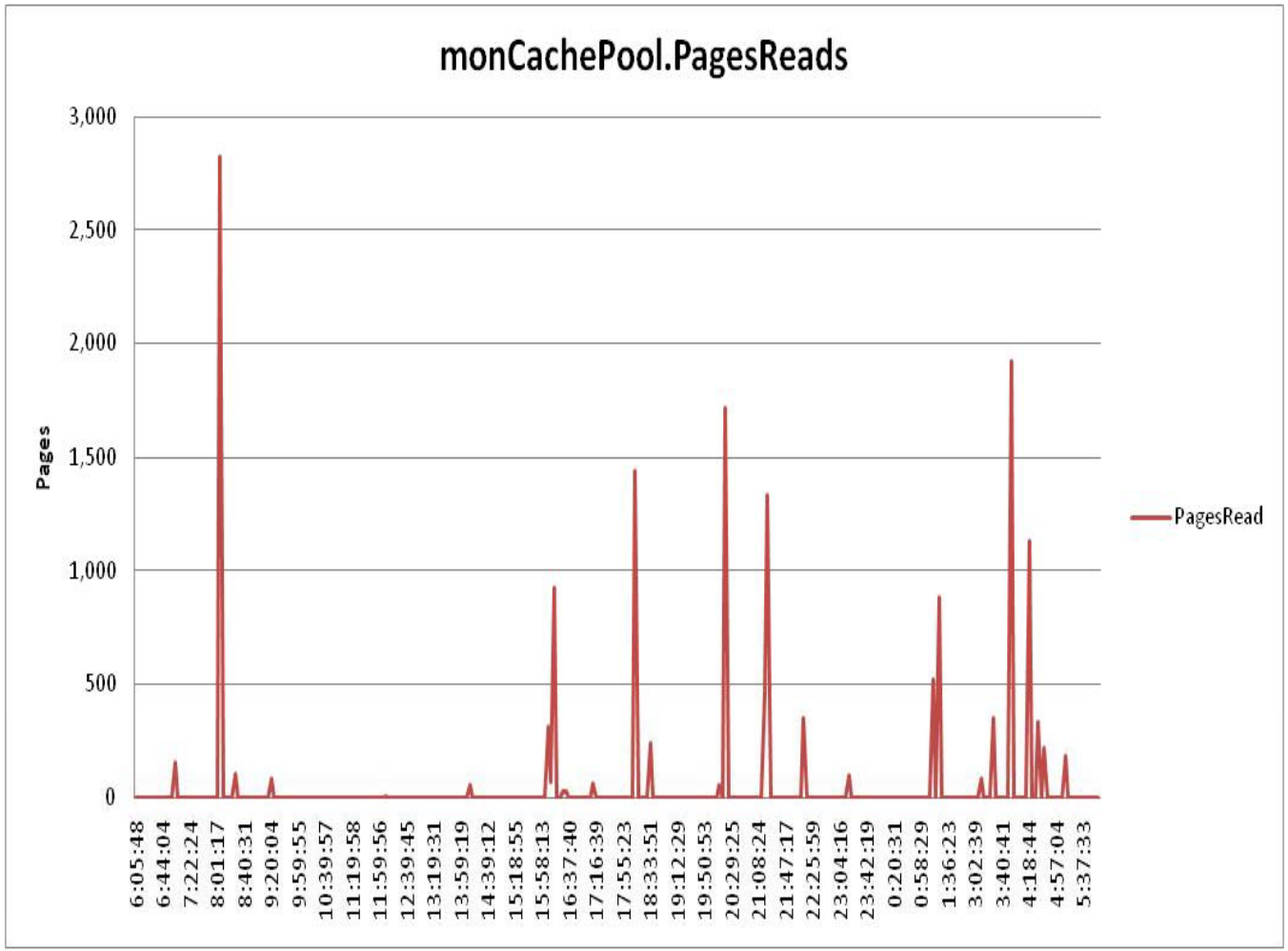
SYBASE | An SAP Company

LOG CACHE

The Easy Stuff

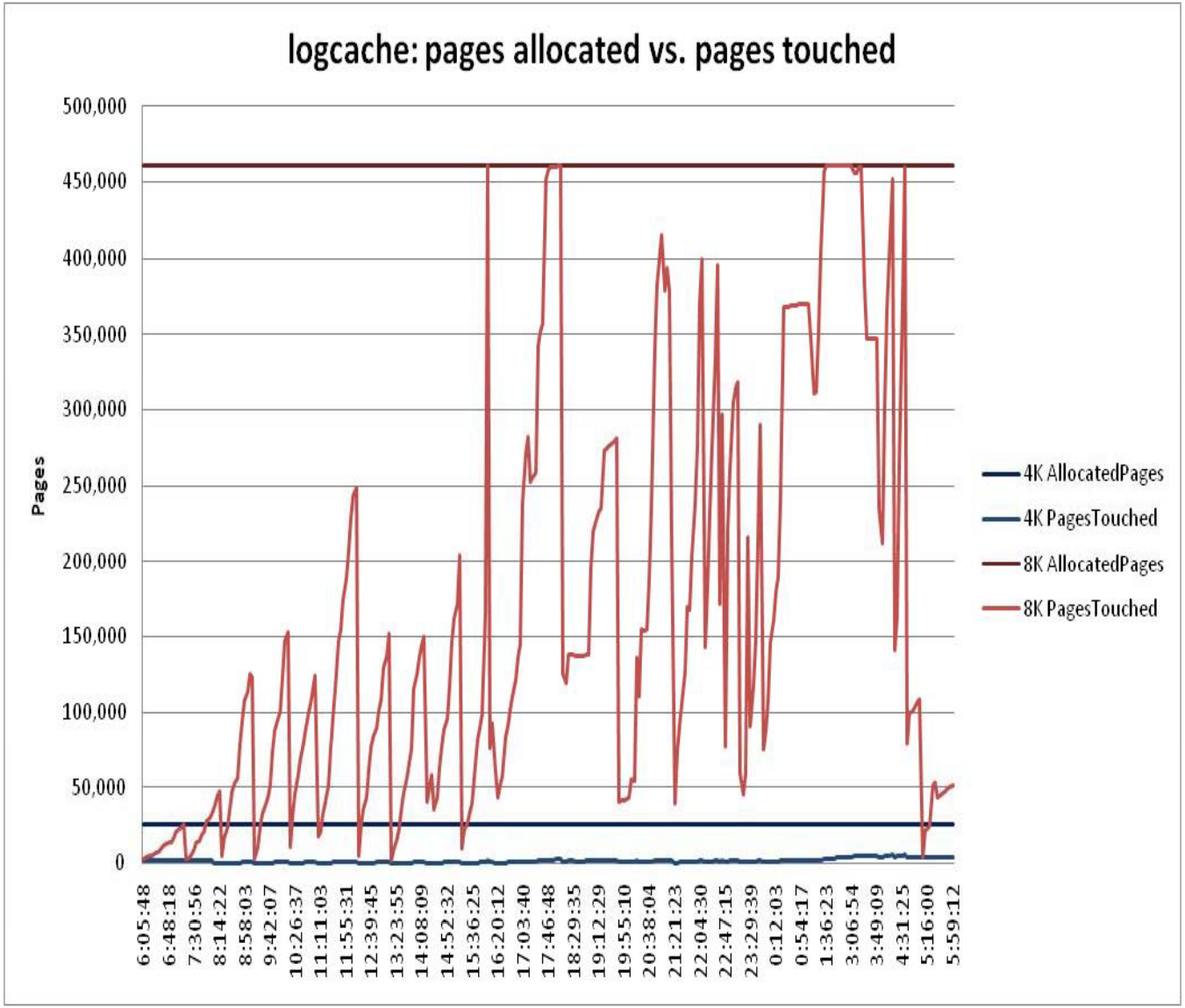
- **Who reads from the log???**
  - ✓ Replication Agent
  - ✓ Checkpoint process
  - ✓ Dump transaction/dump database
  - ✓ Some dbcc commands
    - check storage, checktable on syslogs, etc.
  - ✓ User transactions on tables with triggers
    - Needs to build inserted/deleted tables
  - ✓ User transactions with deferred operations (updates, deletes, etc.)
  - ✓ User transactions with inserts into tables with ignore dupe rows
    - Logs CLR records to “undo” duplicate inserts
- **How to tell if it is too small**
  - ✓ Physical Reads on log device in monDeviceIO
  - ✓ Physical Reads on CheckPoint process in monProcessActivity
  - ✓ Physical Reads in monCachePool (or monDataCache)
    - monDataCache would be most accurate due to transient nature of monCachePool
- **But …how to tell if it is too big???**
  - ✓ Can we use the PagesTouched???

LOGCACHE: PHYSICALREADS



SYBASE | An SAP Company

LOGCACHE: PAGESTOUCHED



SYBASE | An SAP Company

SO…IS IT UNDERSIZED OR OVERSIZED??

- **Erk - Hard to tell…..**
  - ✓ We can't see if a user hit a table with a trigger (too easily….monSysStatement is only way)
  - ✓ monProcess.Command will show 'DUMP DATABASE' or 'DUMP TRANSACTION'
    - But since dump database uses sybmultibufs, no metrics will be visible to MDA
    - Dump tran may be too fast for MDA polling interval - especially if threshold fired
- **But since CHECKPOINT & RepAgent SPIDs are fairly persistant**
  - ✓ It looks like Physical Reads are done by dump tran
  - ✓ Adding 3,000 pages * 4K pagesize = 11.7MB
    - But wait a sec!!! We had a 4K pool defined at 100MB….barely used….just move 12MB from 4K→8K

Application	CPUTime	PhysicalReads	LogicalReads	PagesRead	PhysicalWrites	PagesWritten	TableAccesses	IndexAccesses	Transactions
HK CHORES	5,198,300	1,460	8,525,386	1,460	1,788,154	3,566,784	144,344	138,549	0
HK WASH	2,060,600	9	23,839	9	23,891,474	57,234,617	0	0	0
CHECKPOINT SLEEP	1,859,800	21	2,387,534	21	31,762,147	32,364,356	126,352	79,715	34,373
<rep agent>	1,032,100	113	17,669,879	113	3,177	3,177	17,306,797	152,690	0
HK GC	168,500	107,262	3,458,536	107,262	348,299	694,317	99,871	0	0
Dump Tran (threshold)	2,300	1,265	5,513	2,530	40,879	40,944	81	132	1
Dump Tran (threshold)	3,900	156	1,074	312	33,452	33,498	81	132	1

NAMED CACHES & OBJECT BINDINGS

USING LOGICALREADS & TABLE IO PATTERNS

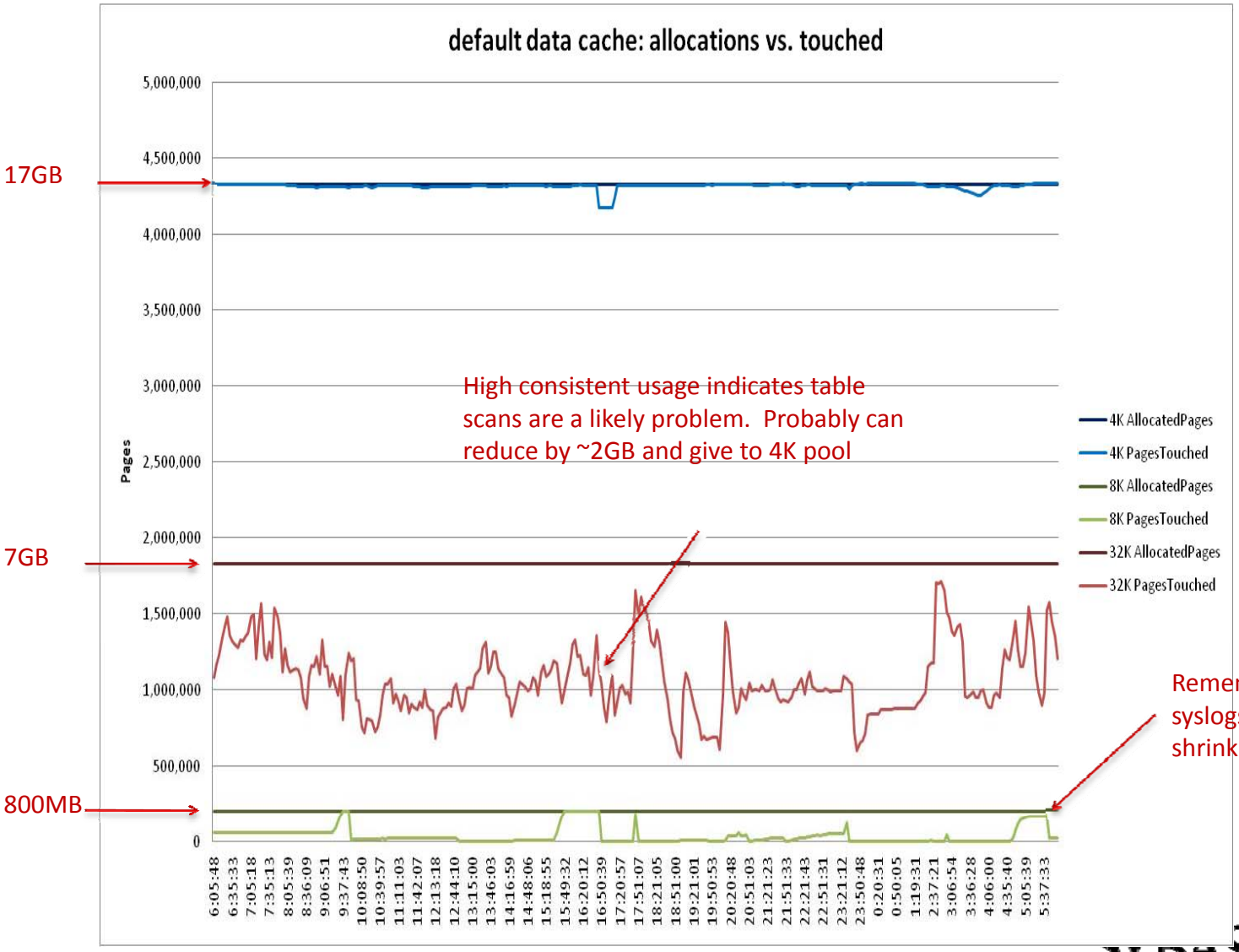
NAMED CACHES

The obvious problems

- **Too Many DBA's Focus on CacheHit Ratio's**
  - ✓ If I have a 4GB cache and only 100MB of data - I should have a 100% cache hit ratio…but I also have wasted 3.9GB of cache
  - ✓ If I have a 4GB table that is fully cached and I do 100,000 table scans on it, what does that do to my cache hit ratio???
- **What to Really Be Concerned About**
  - ✓ Are the cache sizes and cache buffer pool sizes appropriate???
  - ✓ Which objects and which named caches should they be bound to???
  - ✓ Do the buffer pool usage patterns suggest a broader problem???

DEFAULT DATA CACHE SIZING

via monCachePool



REAL CACHE CONFIGURATION

Use monOpenObjectActivity

- **Common problem**
  - ✓ Usual mash of 2-3 named caches and that is it
  - ✓ Dumping stat counters from sp_sysmon shows some cache partitions at high spinlock contention
    - Others with 0 contention results in low average
- **Conventional Wisdom**
  - ✓ Add cache partitions
- **MDA Quick Tip**
  - ✓ monOpenObjectActivity is a ***critical*** monitoring table.
    - The amount of application tuning and troubleshooting information it provides is far beyond your imagination

SYBASE | An SAP Company

YOUR FIRST REAL TEST FOR TODAY

What Cache Configs are You Going to Do??? (4 hour sample)

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11107604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1041762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2570372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1029486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2158504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156

ANSWER #1

Customer (Reference) Data Cache (relaxed cache strategy, ~1GB?2GB)

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11107604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1041762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2570372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1029486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2158504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156

ANSWER #2

Volatile Data Cache #1 (default or relaxed cache strategy, ~50MB→100MB, #partitions = #engines)….IMDB candidate (or cached UFS device + segments)

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11107604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1041762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2570372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1029486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2158504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156

ANSWER #3

Volatile Data Cache #2 (default cache strategy, ~50MB→100MB, #partitions = #engines)….or let this remain in default data cache since it is real business xactns

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11107604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1041762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2570372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1029486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2158504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156

ANSWER #4

Reference Data Cache (relaxed cache strategy, ~50MB→100MB)

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			143,11
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1,041,762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809,656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

ANSWER #5

Key Sequence Data Cache (relaxed cache strategy, ~5MB)

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			143,11
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1,041,762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809,656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

PUTTING IT ALL TOGETHER

Cache configuration changes

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11107604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1041762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2570372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1029486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2158504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7156

PUTTING IT ALL TOGETHER EXPLAINED

The Benefits of Actively Managing Your Cache vs. Not Managing It

- **Significant Reduction in MRU→LRU relinkages**
  - ✓ >300M in 4 hours (23,000/second)
  - ✓ Should see a big drop in cache spinlock contention
  - ✓ Regardless, the tasks will complete slightly quicker each
- **Volatile data no longer pushing others out of cache**
  - ✓ Tables with a lot of insert/delete pairs - especially DOL - will allocate new pages (which grab LRU and go to MRU)
    - Also applies to volatile indexes on actively updated tables
  - ✓ DOL tables need to have 'enable housekeeper GC=5'
    - Watch monOpenObjectActivity.HkgcOverflows if you want to try 4 first
- **What to use dbcc tune(desbind) on….**
  - ✓ Any table …
    - …we bound to a named cache
    - …top 20 tables in default data cache by LogicalReads
  - ✓ Not forgetting the triggers, defaults, rules on those tables

THERE IS AN EASIER WAY

Working Smarter….Not Harder

- **MDA Analysis is Analytics**
  - ✓ A single result set sorted by one attribute makes hard work
- **Slice & Dice**
  - ✓ Top LIO Tables w/ Indexes
  - ✓ Top 20 Indexes by LIO
  - ✓ Top Small Tables scanned by LIO
  - ✓ Top Possible Sequence Key Tables
  - ✓ Top 50 Static Tables by LIO
  - ✓ Top 50 Volatile Tables by Writes
  - ✓ Top Inserted Tables
  - ✓ Top Updated Tables

MONOPENOBJECTACTIVITY TIPS

Some Things to Think About

- **How do we know an object was flushed from cache???**
  - ✓ Answer: monOpenObjectActivity is based on reading the object descriptors for open objects…
    - Stats are stored in IDES (Index) descriptors
  - ✓ …if an object is removed from cache, the descriptor will most likely be reclaimed.
  - ✓ …When object is re-read in, it gets a new descriptor
  - ✓ Result: Subsequent metrics are lower than former ones
    - If doing deltas on a before/after basis - the results will be negative - but not near the rollover point (e.g. not -2B)…..can still be in negative millions….
- **But the above is a complete flush from cache - rare….**
- **How do we know when an object is fluctuating in cache???**
  - ✓ Some pages being bumped out to make room for other pages.
  - ✓ Answer 1: When PhysicalReads exceeds number of pages in table
    - Works for small tables
    - Can also be a “bell-ringer” on larger key transaction tables…needs follow-up
  - ✓ Answer 2: monCachedObject

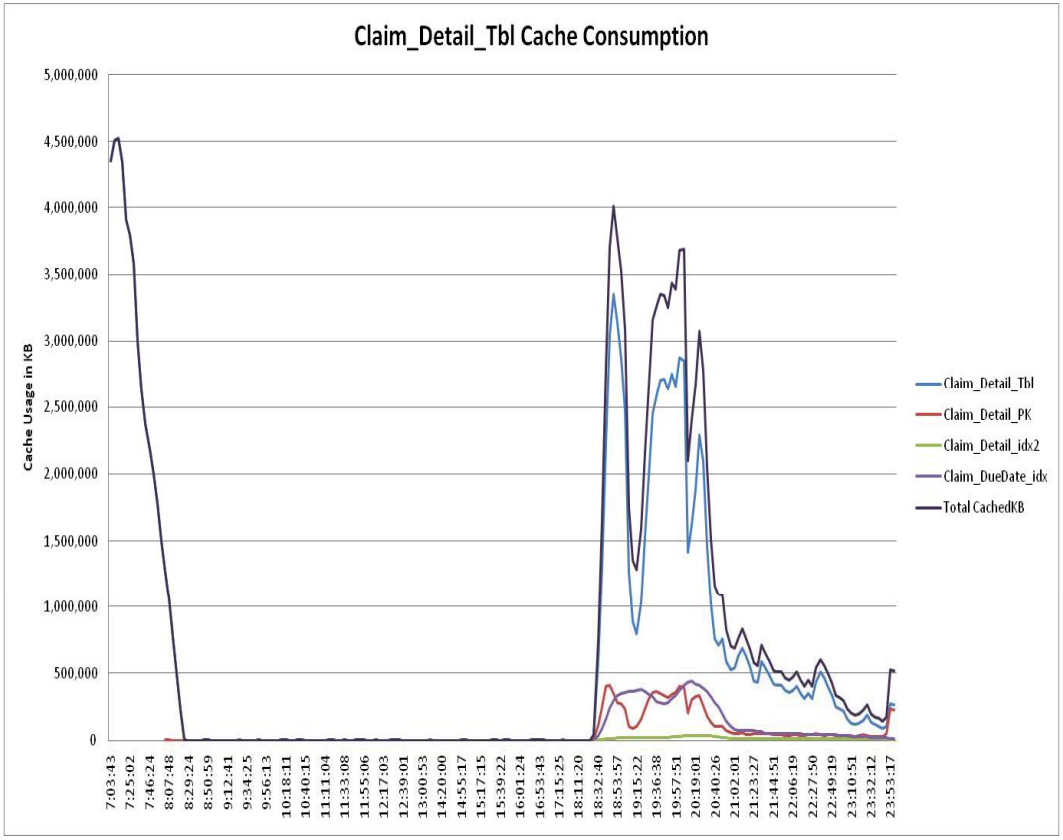
MONITORING DATA CACHE BY OBJECT

monCachedObject

- **CAREFUL!!!!**
  - ✓ Prior to ASE 15.5 ESD #2, this would grab a spinlock on each cachelet as it scanned the pages to dynamically add up the cache usage
  - ✓ This gave an accurate count of how much cache was used…….but……
  - ✓ Causes severe performance degradation if sampled more than once every few minutes
    - Unfortunately, most GUI tools are set to poll every few seconds ….which results in severe degradation
  - ✓ ASE 15.5 ESD #3 and higher no longer grab the spinlock….
    - You can have accurate counters or non-intrusive counters - pick only one
- **One of the more useful for cache sizing/bindings**
  - ✓ Tracks all tables (including system tables) down to the index or partition level

monCachedObject		
CacheID	int	<pk,fk>
InstanceID	tinyint	<pk,fk>
DBID	int	<pk>
IndexID	int	<pk>
PartitionID	int	<pk>
CachedKB	int	
CacheName	varchar(30)	<pk,fk>
ObjectID	int	<pk>
DBName	varchar(30)	<pk>
OwnerUserID	int	
OwnerName	varchar(30)	
ObjectName	varchar(30)	<pk>
PartitionName	varchar(30)	
ObjectType	varchar(30)	
TotalSizeKB	int	
ProcessesAccessing	int	

PHYSREADS, CACHING & LIO



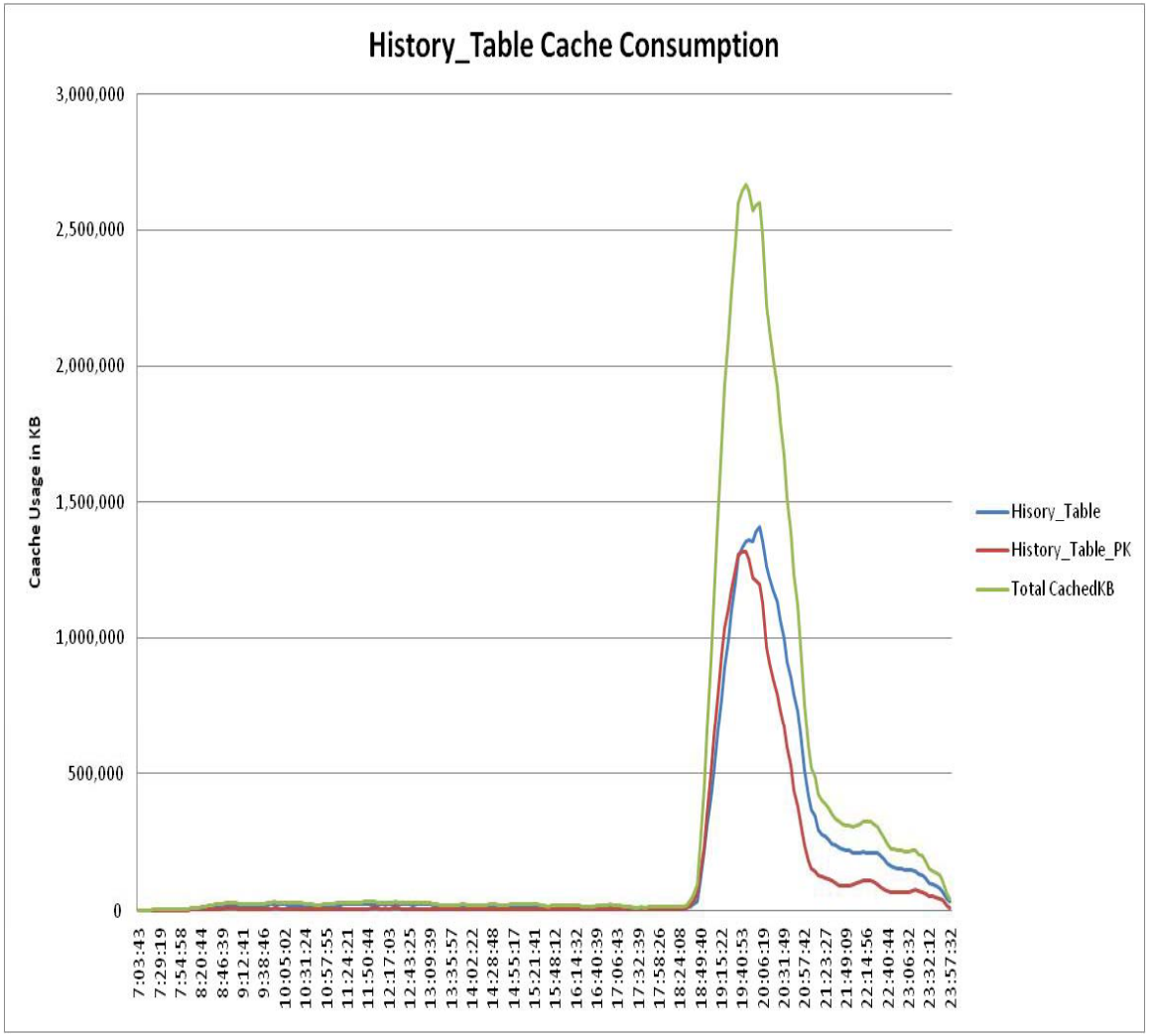
The price of table/partition scans (including PK starting point scans) or bad queries (most likely) that cause 2M PhysReads (5.5M PagesRead)

On the other hand, since this is a partitioned table, depends on clustered index keys vs. partition key - this could just point to a really bad partitioning decision in which data is scattered helter-skelter driving all PIO for every page access.

ObjectName	Index ID	Logical Reads	Physical Reads	APF Reads	Pages Read	Physical Writes	Pages Written	Rows Inserted	Rows Deleted	Rows Updated	Used Count
Claim_Detail_Tbl	0	26,081,269	1,914,716	1,309,377	5,697,551	89,942	337,294	819,955	48,415	0	819,953
Claim_Detail_PK	2	13,691,870	102,647	73,709	436,680	121,644	546,887	0	48,415	0	1,049,849
Claim_Detail_idx2	3	3,691,746	27,062	0	27,062	35,733	35,754	819,965	48,415	0	0
Claim_Detail_Due_Date_idx	4	6,071,684	276,570	11,856	276,570	118,391	118,398	819,966	48,415	0	308,829

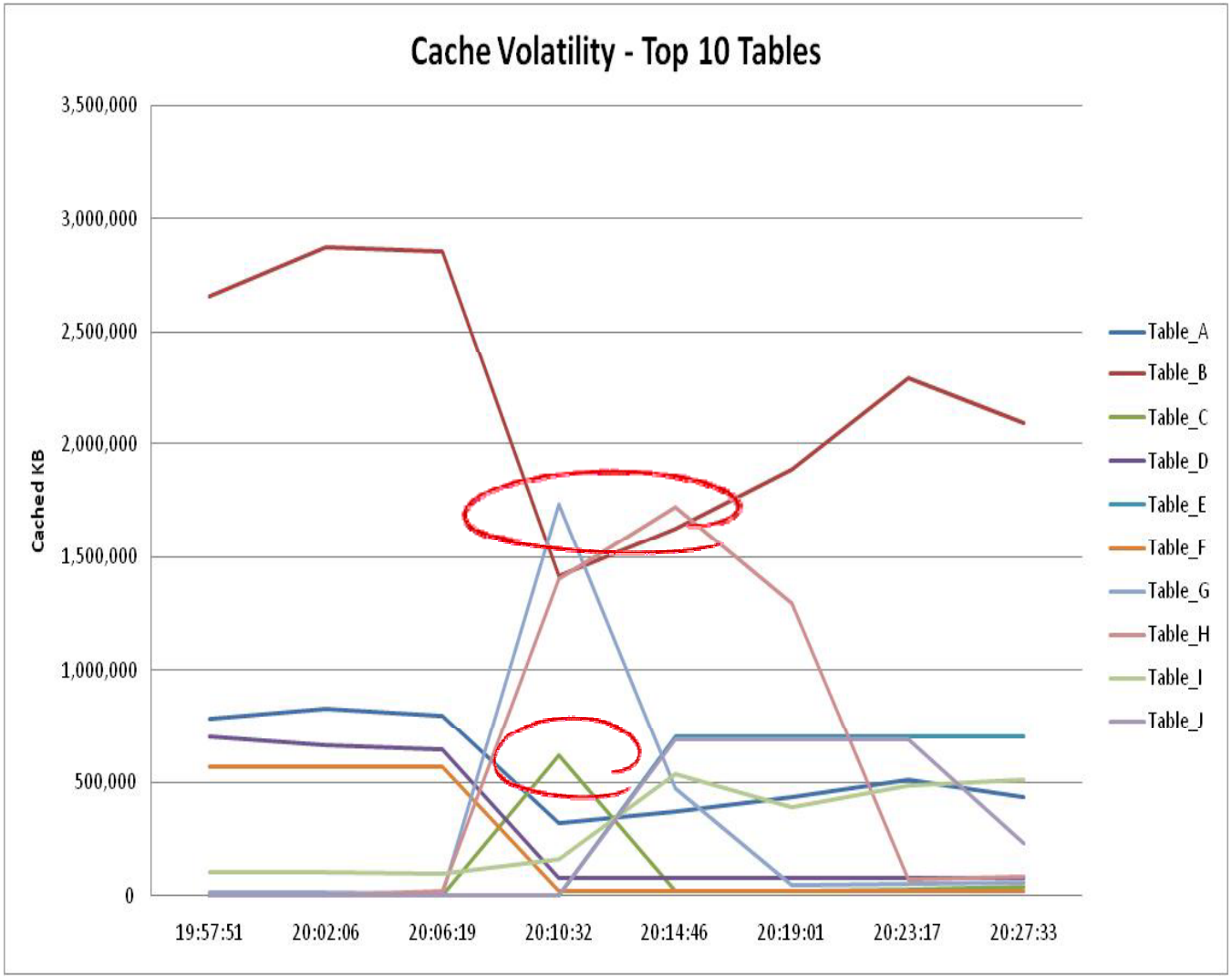
THE NEED FOR NAMED CACHES (1)

Restriction Cache for History Tables



SYBASE | An SAP Company

THE NEED FOR NAMED CACHES (2)



INDEXING & LOGIC ISSUES

USING LOGICALREADS & TABLE IO PATTERNS

DATABASE ACCESS METHODS 201

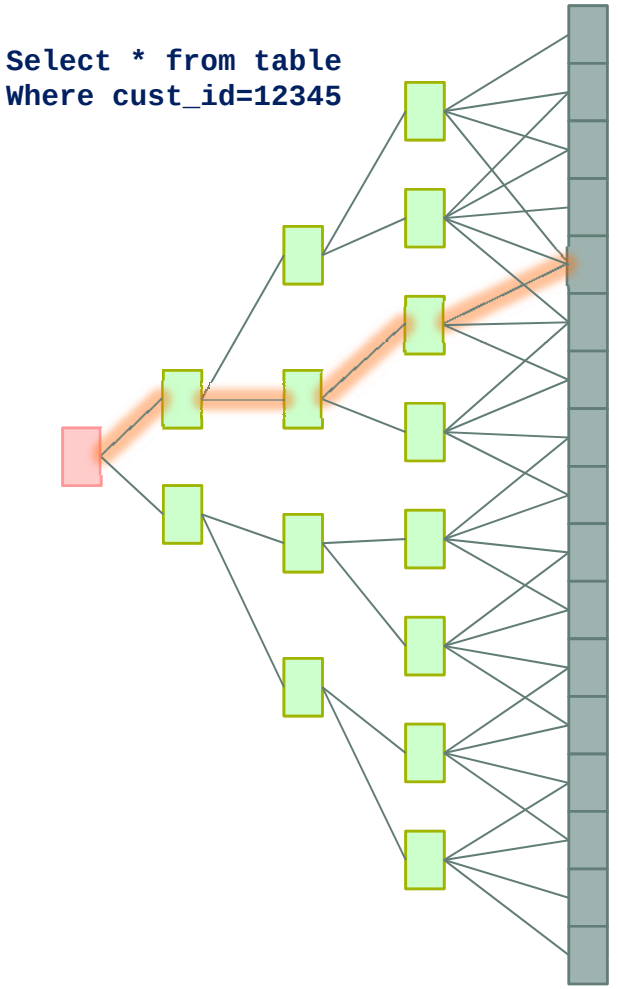
A Quick Review of How Indexes are Used and IO Patterns

- Single Row Lookup/Nested Loop Join
- Index Scan/Merge Join
- Covered Queries
- Range Scan
- Heap Tables (No Clustered Index)

SINGLE ROW LOOKUP/NESTED LOOP JOIN

IO Patterns for a Well Indexed Access Strategy

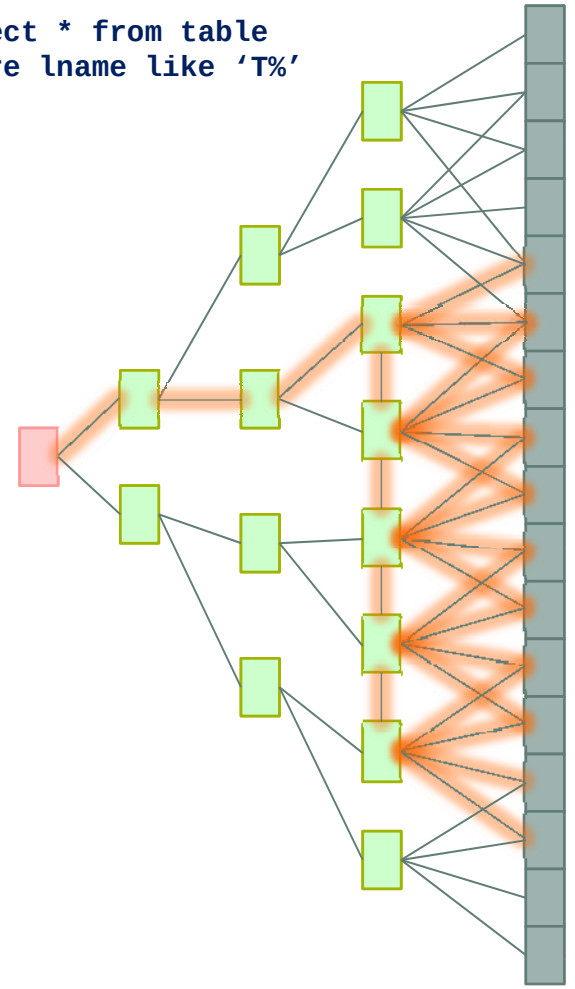
- **Quick Review**
  - ✓ Index is a b-tree+ with root node, intermediate nodes, leaf level
  - ✓ Index is unique or pkey or similar
  - ✓ Index levels depends on number of rows in the table, but typically
    - 2-3 levels of indexing for 1-10M rows
    - 4-5 levels of indexing for 10's-100's of millions of rows
- **Data Access Method**
  - ✓ Each row seek is a full index traversal to the leaf and then the data page read
    - So with 5 levels of index, we would expect 6 IOs total (5 index + 1 datapage)
  - ✓ Non-indexed SARGS are then evaluated
  - ✓ So, for a join with 1000 outer rows, the full index tree will be traversed 1,000 times
- **Net Effect**
  - ✓ We expect to see not only index optimizer usage, but a IO ratio of 4:1 or so (depending on size of table)



INDEX SCAN/MERGE JOIN

IO Patterns for a Partial Index Usage

- **Quick Review**
  - ✓ Index isn't very distinct (lots of duplicates) or search range is broad
- **Data Access Method**
  - ✓ Index is traversed down the first key value
  - ✓ Index pages are scanned until the index key no longer matches
  - ✓ As each index leaf row is read, the corresponding datapage is fetched and non-indexed SARGs applied
- **Net Effect**
  - ✓ We see a few index IO's but a lot of datapage IO's due to number of rows per page at index leaf level
    - Usually 10:1 or 20:1 data to index
  - ✓ Not always avoidable, but sometimes better indexing should be considered if non-indexed predicates are common

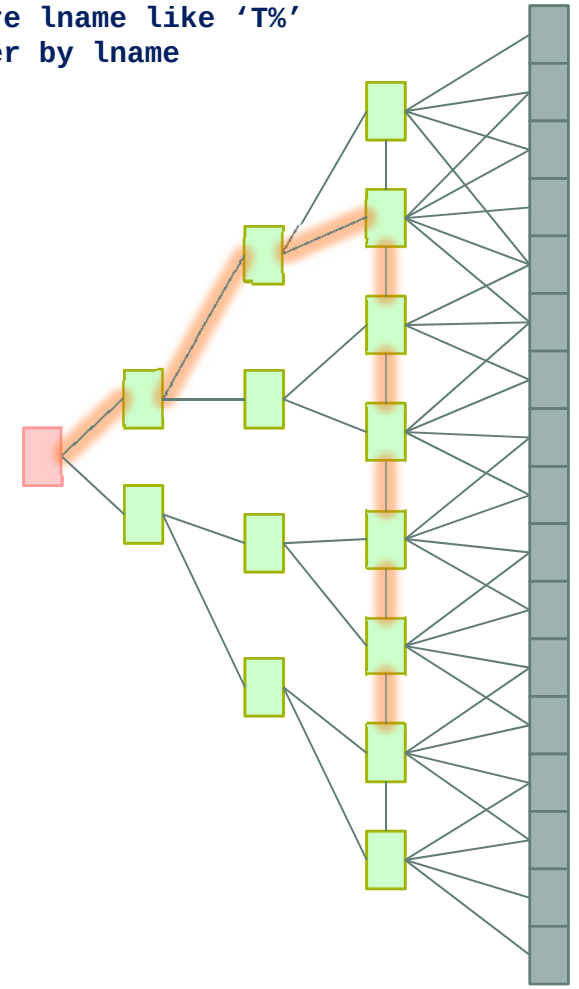


COVERED QUERIES

IO Patterns for Index Covered Queries

- **Quick Review**
  - ✓ Index contains all data in both projection as well as predicates
- **Data Access Method**
  - ✓ Index is traversed down the first key value
  - ✓ Index pages are scanned until the index key no longer matches
- **Net Effect**
  - ✓ Very high index IO with no datapage IO
  - ✓ For some reporting queries, this could be a full scan of the index

```
Select top 10 lname from table
Where lname like 'T%'
Order by lname
```

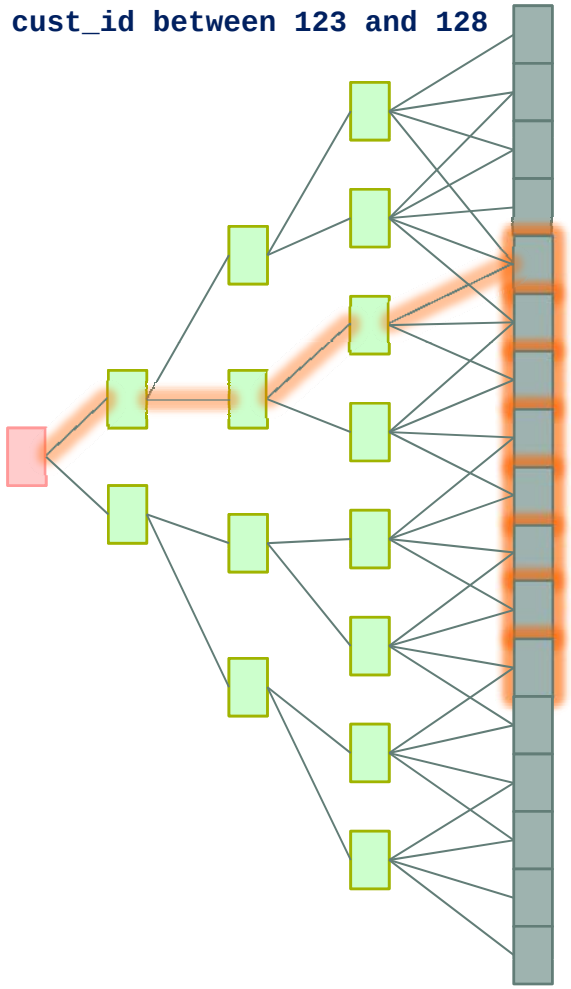


RANGE SCAN

IO Patterns for a Clustered Index Range Scan

- **Quick Review**
  - ✓ Clustered index on APL table
  - ✓ For DOL table, this would be an index scan instead
  - ✓ Applies to:
    - Col between X and Y
    - Col like 'Pat%' (becomes between)
    - Min()/Max()/top n
- **Data Access Method**
  - ✓ Index tree is traversed until starting point is found
    - Beginning for between/range/min/top n
    - End for max
  - ✓ Datapages are scanned until predicates no longer match or the min/max is found
- **Net Effect**
  - ✓ We expect to see very low index IO on clustered index followed by a lot of datapage IO's
  - ✓ May be confused with clustered index driven table scan
  - ✓ Needs to be evaluated to see if a covered index is a better solution (e.g. min/max)

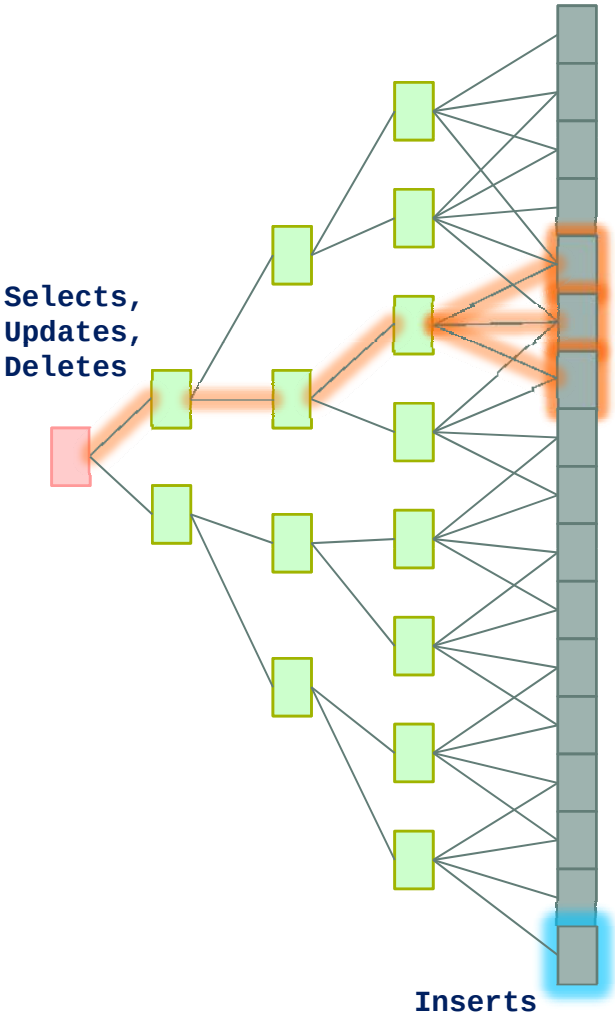
```
Select * from table
Where cust_id between 123 and 128
```



HEAP TABLES (NO CLUSTERED INDEX)

Tempdb plus a lot of other common tables (e.g. history tables)

- **Fact vs. Fiction**
  - ✓ Fiction: All DOL tables are heap tables
  - ✓ Fact: WRONG – Clustered indexes on DOL tables are still placement indexes – it just doesn't guarantee sequential sort order, but it will try to put the data as close as possible
- **Selects, Updates & Deletes**
  - ✓ Generally will operate like an index scan in that an index will be used to find the rows
  - ✓ So, a lot of datapages and a few index IO's depending on the uniqueness of the index selected
- **Inserts**
  - ✓ Since no clustered index, rows are inserted at the end of the table
  - ✓ From an optimization standpoint, this will look like a table scan (indid=0 picked)
  - ✓ However, the number of LIO's will be significantly less than the size of the table AND the number of rows inserted will be a considerable portion of the UsedCount



KEEP IO EXPECTATIONS REALISTIC

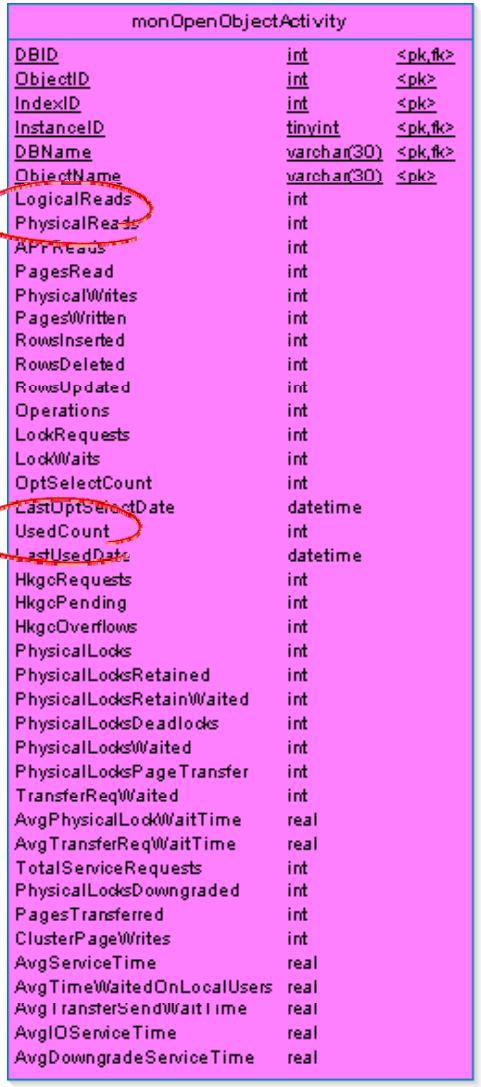
Wishful Thinking Followed by a Cold Does of Reality

- **You may think you have a lot of range scans**
  - ✓ But the reality is that often these are in combination with joins and other query structures that cause index IO's to outweigh data IO's
- **Pick an index, any index …..is not a good solution**
  - ✓ Just because the QP says it is using an index, look at the cost of the query in terms of CPU and LIO.
- **Partial Indexes are Partial Solutions**
  - ✓ If you have 3 queries on different subsets of columns, creating a single index on the common columns for all 3 queries saves you some space, but it means none of the queries is efficient
  - ✓ Yes, this works well in dev with 100 rows…..Try production with 100 million
  - ✓ Use capture missing stats and look at the missing densities to identify sets of columns to index
- **What you really want**
  - ✓ More index IO's than datapage IO's
  - ✓ The higher the ratio the better

MONOPENOBJECTACTIVITY & INDEXING

Some key considerations about using for determining indexing quality

- **LogicalReads: Index vs. Table**
  - ✓ Generally speaking, one expects to see LogicalReads on indexes a lot higher than base table
    - NL Joins will traverse 3+ levels of indexing for each page read
    - If exists() clauses could be covered by index
    - Covered queries (e.g. select count(*))
  - ✓ Exceptions to the rule:
    - Range scans (especially clustered index by date)
    - Backwards scans where SARG not fully indexed
- **UsedCount**
  - ✓ Table Scans: IndexID=0 and UsedCount > 0
    - Compare LogicalReads to UsedCount → estimate pages per scan
    - If <10 pages, small table….verify with actual rowcounts
    - Heap table inserts will also show a IndexID =0 and UsedCount >0
      - Check if IndexID=1 exists
      - If DOL, a clustered index may be IndexID>1 but this would not be a heap insert (so positive UsedCounts is a tablescan)
  - ✓ Unused/Low utilized indexes: IndexID>=0 and UsedCount=0
    - A lot of indexes are created at schema time based on query expectations that are never used or maybe only a few times per day
    - Compare RowsInserted vs. RowsDeleted - remember, an index key is never updated - it has to move to keep the index in sort order.



monOpenObjectActivity		
DBID	int	<pk,fk>
ObjectID	int	<pk>
IndexID	int	<pk>
InstanceID	tinyint	<pk,fk>
DBName	varchar(30)	<pk,fk>
ObjectName	varchar(30)	<pk>
LogicalReads	int	
PhysicalReads	int	
APFReads	int	
PagesRead	int	
PhysicalWrites	int	
PagesWritten	int	
RowsInserted	int	
RowsDeleted	int	
RowsUpdated	int	
Operations	int	
LockRequests	int	
LockWaits	int	
OptSelectCount	int	
LastOptSelectDate	datetime	
UsedCount	int	
LastUsedDate	datetime	
HkgcRequests	int	
HkgcPending	int	
HkgcOverflows	int	
PhysicalLocks	int	
PhysicalLocksRetained	int	
PhysicalLocksRetainWaited	int	
PhysicalLocksDeadlocks	int	
PhysicalLocksWaited	int	
PhysicalLocksPageTransfer	int	
TransferReqWaited	int	
AvgPhysicalLockWaitTime	real	
AvgTransferReqWaitTime	real	
TotalServiceRequests	int	
PhysicalLocksDowngraded	int	
PagesTransferred	int	
ClusterPageWrites	int	
AvgServiceTime	real	
AvgTimeWaitedOnLocalUsers	real	
AvgTransferSendWaitTime	real	
AvgIOServiceTime	real	
AvgDowngradeServiceTime	real	

POTENTIAL INDEXING ISSUES (1)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14,311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1,041,762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809,656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

Blue & Green are good. Yellow, Orange, Pink are bad

POTENTIAL INDEXING ISSUES (2)

Explaining the problems

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
										0
										29
										0
										11
										52
										56
										0
										0
										0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

Site table is not small ala the "id" tables at the bottom - we can tell by the fact it is not scanned (UsedCount=0) and therefore an index is being used to access it….however, the index isn't in the top 30 by LogicalReads - so the index is being used as a starting point for a scan - or an index based scan. Likely clustered index used to start a table scan or index most often used is not very unique. If the most commonly selected index is not very unique, a possible solution is to add one or more indexes that cover the most common SARGS (use capture missing stats to find). Table is fairly static (no DML rows), so, adding an index will not impede application performance (it will only help).

Blue & Green are good. Yellow, Orange, Pink are bad

POTENTIAL INDEXING ISSUES (3)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

client_event table is accessed by at least one index - most common one is likely shown…. but either a not very unique index or likely a backward scan on a non-indexed SARG (e.g. not just find their last event, but find the last event for a particular attribute). SARG should be added to the index. Likely APL locked and updates all trigger blocking at the index. **Follow-up Action:** Really needs verified as IndexID=9 suggests a lot of indexes for this table already….probably some not used.

Blue & Green are good. Yellow, Orange, Pink are bad

POTENTIAL INDEXING ISSUES (4)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14,311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1,041,762
cpt_grouping_desc	0						0	4,123,724	0	809,656
permission_trans	0						0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

Table scans. Enough said.

Blue & Green are good. Yellow, Orange, Pink are bad

POTENTIAL INDEXING ISSUES (5)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
cli										
re										
cu										
sit										
pe										
cp										
pe										
cu										
cli										
cli										
cu										
ev										
cp										
gr										
pe										
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

Okay, this one is special. Some other index is used to access the table - but it too is not in the top 30 (by LIO). So the most commonly selected index is not very unique…. and table is static, so adding more indexes will not likely impact DML performance.

Pop Quiz: If the table is *static* (no rows modified), why would an index have LogicalReads but never be picked by the optimizer??

Possible Answer: If iso 3 query, both APL and DOL tables use locks on indexes to prevent modifications…one possible cause….2nd and more likely is DRI constraint enforcement, but then we would expect this on the PKEY index only (which it might be) and ~15M (1:3 or 1:4) cumulative row ins/upd/del elsewhere on FKEY tables….there may be others (e.g. trans rolled back)…but is app using iso 3 by accident….or too many DRI layers (e.g. grandchildren with direct DRI to grandparent vs. parent tables only)???

NO INDEXING ISSUES (1)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client							0			14,311
permission							0	4,167,064	0	1,041,762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809,656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

All small tables - 1-2 pages total in table

Blue & Green are good. Yellow, Orange, Pink are bad

NO INDEXING ISSUES (2)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
cu										
sit										
pe										
cp										
pe										
cu										
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372
customer_rules	0	2,463,521	0	0	0	0	0	4,866,200	0	0
event_status_summary	2	2,130,222	3	0	183,968	184,028	0			335,729
cpt_group_addon	0	2,058,968	0	2,058,969	0	0	0	0	0	1,029,486
group_trans	2	2,043,431	9	0	184,183	184,184	0			2,158,504
permission_relation	0	1,926,847	0	0	0	0	0	13,048,693	0	0
client_event	9	1,690,384	0	0	42,897	35,800	0			15,253
result_selection_comments	0	28,258	0	28,258	0	0	0	7	0	14,129
call_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156
client_event_id	0	14,312	0	14,312	0	0	7,156	14,312	0	7,156

permission_relation: Index to table ratio 33:2….suggests covered queries or index scans as well as ~1:3 ratio (index height of ~3)…likely covered queries such as if exists() clauses

Blue & Green are good. Yellow, Orange, Pink are bad

NO INDEXING ISSUES (3)

The following is top 30 monOpenObjectActivity rows sorted by LogicalReads

TableName	Index ID	Logical Reads	Physical Reads	APFReads	Rows Inserted	Rows Deleted	Rows Updated	Lock Requests	Lock Waits	Used Count
site	0	110,215,414	0	0	0	0	0	0	0	0
customer	2	101,301,758	0	0	0	0	0			14,311
client_event	0	86,471,323	0	0	7,235	0	35,800	186,953	20,550	0
customer_eligibility	3	56,935,200	0	0	0	0	0			0
customer_eligibility	0	46,460,814	0	0	0	0	0	0	0	0
permission_relation	2	33,331,924	0	0	0	0	0			11,107,604
letter_request	0	21,517,184	0	19,245,616	0	0	0	0	0	6,968
permission_trans	3	13,261,549	329	0	1,569,209	1,569,210	0			616,745
permission_trans	2	12,619,479	0	0	1,569,212	1,569,204	0			14,168
client_steerage_detail	3	9,435,417	0	0	0	0	0			0
result_selections	0	8,646,750	0	8,307,689	0	0	0	243,215	0	14,129
customer_site	3	8,123,220	0	0	0	0	0			0
site_client	2	6,371,175	0	0	0	0	0			14,311
permission	0	4,167,037	0	0	0	0	0	4,167,064	0	1,041,762
cpt_grouping_desc	0	4,123,720	0	0	0	0	0	4,123,724	0	809,656
permission_trans	0	3,453,833	2,227	0	1,569,211	1,569,213	0	13,930,857	3,133	0
customer_rules	2	3,020,451	0	0	0	0	0			0
client_search_list	0	2,570,376	0	0	0	0	0	2,599,612	0	0
client_search_list	1	2,570,376	0	0	0	0	0			2,570,372

permission_trans: Index to table ratio ~1:4 for each which is good…however, we do have some blocking so likely should use DOL (with none at index, likely we are using datapage instead of datarows locking)…since we are picking NC indexes, it should not affect the queries (e.g. no one appears to be default on clustered index sort order and not using order by clause)…another potential issue is "*Used Count*" to LIO ratio on indexes is low…but with 1.5M inserts and 1.5M deletes or 3M row modifications …depending on size of table, the LIO's could be skewed due traversing the indexes to insert/remove index keys (vs. iso 3 from earlier)

Blue & Green are good. Yellow, Orange, Pink are bad

A MORE BLATANT EXAMPLE

TableName	IndexID	TotalObjLIO	Logical Reads	Logical PerSec	Physical Reads	APFReads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	UsedCount
LookupList	0	709,696,229	160,264,636	1,857	3,105	0	30,341	5,564,082	0	0	102,335
LookupList	2	709,696,229	549,431,593	6,369	3,412	19,713	105,348	5,564,082	0	0	159,692,371
CentralEntityList	0	537,599,673	830,119	9	5	0	8	448	48	714	21
CentralEntityList	2	537,599,673	536,769,554	6,222	3	0	5	448	48	0	536,769,044
PriceTable	0	497,820,034	160,105,909	1,856	1,259,792	482,210	131	950	865	0	1
PriceTable	1	497,820,034	337,714,125	3,915	12,538	482,210	131	950	865	0	112,570,568
ProcedureDefinition	0	483,176,760	475,713,008	5,515	109,734	362,106	23,868	67,228	301	0	2
ProcedureDefinition	1	483,176,760	5,726,171	66	23,828	362,106	23,868	67,228	301	0	1,835,063
ProcedureDefinition	2	483,176,760	1,737,581	20	9,462	0	9,924	67,228	301	0	0
EntityPayment	0	388,848,558	298,876,091	3,464	1,204,405	2,180,663	479,766	2,266,615	1,496,147	0	12
EntityPayment	1	388,848,558	89,972,467	1,043	301,532	2,180,663	479,766	2,266,615	1,496,147	0	20,177,319
ProviderSchedule	0	363,662,057	93,384,674	1,082	921,643	529,227	4,764	72,266	3	0	3
ProviderSchedule	1	363,662,057	270,277,383	3,133	9,589	529,227	4,764	72,266	3	0	90,017,828
ScheduleModification	0	349,035,595	141,100,184	1,635	737,403	69,297,838	0	0	0	0	0
ScheduleModification	1	349,035,595	6	0	6	69,297,838	0	0	0	0	4
ScheduleModification	2	349,035,595	207,731,934	2,408	639,524	217,233	7	0	0	0	51,382,966
ScheduleModification	3	349,035,595	203,471	2	160	36,674	0	0	0	0	21
ProviderProvisions	0	341,192,246	252,341,557	2,925	182,868	31,191,349	795	156	0	1,861	2,666
ProviderProvisions	2	341,192,246	76,644,611	888	11,090	8	111	0	0	0	40,850,838
ProviderProvisions	3	341,192,246	14,821	0	982	1,314	717	912	756	0	77
ProviderProvisions	4	341,192,246	11,841	0	1,632	499	628	910	754	0	1,394
ProviderProvisions	5	341,192,246	1,833,979	21	16,316	13,719	270	814	658	0	3,400,017
ProviderProvisions	6	341,192,246	3,355,969	38	56,264	40,539	356	329	173	0	2,823,304
ProviderProvisions	7	341,192,246	4,873	0	120	3	211	814	658	0	47
ProviderProvisions	8	341,192,246	7,504	0	374	40	252	814	658	0	248
ProviderProvisions	9	341,192,246	824,806	9	14,116	646	194	227	71	0	207,618
ProviderProvisions	10	341,192,246	1,600,890	18	20,397	0	283	849	693	0	529,967
ProviderProvisions	11	341,192,246	4,551,395	52	6,779	12,336	292	875	719	0	1,864,902
CustomUserField_D	0	298,322,608	294,498,452	3,414	42,181	752,021	140	52	0	246	52
CustomUserField_D	1	298,322,608	3,824,156	44	517	752,021	140	52	0	246	1,274,658
#tiv_bas00058950021666137	0	284,067,888	97,866,203	1,946	24	1,095	860,375	46,306,910	0	0	355
#tiv_bas00058950021666137	1	284,067,888	186,201,685	3,703	18	1,095	860,375	46,306,910	0	0	46,252,240

LOOP ALERT → BAD APP LOGIC = SLOOWWW

TableName	IndexID	TotalObjLIO	Logical Reads	Logical PerSec	Physical Reads	APFReads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	UsedCount
LookupList	0	709,696,229	160,264,636	1,857	3,105	0	30,341	5,564,082	0	0	102,335
LookupList	2	709,696,229	549,431,593	6,369	3,412	19,713	105,348	5,564,082	0	0	159,692,371
CentralEntityList	0	537,599,673	830,119	9	5	0	8	448	48	714	21
CentralEntityList	2	537,599,673	536,769,554	6,222	3	0	5	448	48	0	536,769,044
PriceTable	0	497,820,034	160,105,909	1,856	1,259,792	482,210	131	950	865	0	1
PriceTable	1	497,820,034	337,714,125	3,915	12,538	482,210	131	950	865	0	112,570,568
ProcedureDefinition	0	483,176,760	475,713,008	5,515	109,734	362,106	23,868	67,228	301	0	2
ProcedureDefinition	1	483,176,760	5,726,171	66	23,828	362,106	23,868	67,228	301	0	1,835,063
ProcedureDefinition	2	483,176,760	1,737,581	20	9,462	0	9,924	67,228	301	0	0
EntityPayment	0	388,848,558	298,876,091	3,464	1,204,405	2,180,663	479,766	2,266,615	1,496,147	0	12
EntityPayment	1	388,848,558	89,972,467	1,043	301,532	2,180,663	479,766	2,266,615	1,496,147	0	20,177,319
ProviderSchedule	0	363,662,057	93,384,674	1,082	921,643	529,227	4,764	72,266	3	0	3
ProviderSchedule	1	363,662,057	270,277,383	3,133	9,589	529,227	4,764	72,266	3	0	90,017,828
ScheduleModification	0	349,035,595	141,100,184	1,635	737,403	69,297,838	0	0	0	0	0
ScheduleModification	1	349,035,595	6	0	6	69,297,838	0	0	0	0	4
ScheduleModification	2	349,035,595	207,731,934	2,408	639,524	217,233	7	0	0	0	51,382,966
ScheduleModification	3	349,035,595	203,471	2	160	36,674	0	0	0	0	21
ProviderProvisions	0	341,192,246	252,341,557	2,925	182,868	31,191,349	795	156	0	1,861	2,666
ProviderProvisions	2	341,192,246	76,644,611	888	11,090	8	111	0	0	0	40,850,838
ProviderProvisions	3	341,192,246	14,821	0	982	1,314	717	912	756	0	77
ProviderProvisions	4	341,192,246	11,841	0	1,632	499	628	910	754	0	1,394
ProviderProvisions	5	341,192,246	1,833,979	21	16,316	13,719	270	814	658	0	3,400,017
ProviderProvisions	6	341,192,246	3,355,969	38	56,264	40,539	356	329	173	0	2,823,304
ProviderProvisions	7	341,192,246	4,873	0	120	3	211	814	658	0	47
ProviderProvisions	8	341,192,246	7,504	0	374	40	252	814	658	0	248
ProviderProvisions	9	341,192,246	824,806	9	14,116	646	194	227	71	0	207,618
ProviderProvisions	10	341,192,246	1,600,890	18	20,397	0	283	849	693	0	529,967
ProviderProvisions	11	341,192,246	4,551,395	52	6,779	12,336	292	875	719	0	1,864,902
CustomUserField_D	0	298,322,608	294,498,452	3,414	42,181	752,021	140				52
CustomUserField_D	1	298,322,608	3,824,156	44	517	752,021	140				1,274,658
#tiv_bas00058950021666137	0	284,067,888	97,866,203	1,946	24	1,095	860,375	46,306,910	0	0	355
#tiv_bas00058950021666137	1	284,067,888	186,201,685	3,703	18	1,095	860,375	46,306,910	0	0	46,252,240

Mostly Heap Inserts

BBBAAAADDD INDEXING

Including Table Scans

TableName	IndexID	TotalObjLIO	Logical Reads	Logical PerSec	Physical Reads	APFReads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	UsedCount
LookupList	0	709,696,229	160,264,636	1,857	3,105	0	30,341	5,564,082	0	0	102,335
LookupList	2	709,696,229	549,431,593	6,369	3,412	19,713	105,348	5,564,082	0	0	159,692,371
CentralEntityList	0	537,599,673	830,119	9	5	0	8	448	48	714	21
CentralEntityList	2	537,599,673	536,769,554	6,222	3	0	5	448	48	0	536,769,044
PriceTable	0	497,820,034	160,105,909	1,856	1,259,792	482,210	131	950	865	0	1
PriceTable	1	497,820,034	337,714,125	3,915	12,538	482,210	131	950	865	0	112,570,568
ProcedureDefinition	0	483,176,760	475,713,008	5,515	109,734	362,106	23,868	67,228	301	0	2
ProcedureDefinition	1	483,176,760	5,726,171	66	23,828	362,106	23,868	67,228	301	0	1,835,063
ProcedureDefinition	2	483,176,760	1,737,581	20	9,462	0	9,924	67,228	301	0	0
EntityPayment	0	388,848,558	298,876,091	3,464	1,204,405	2,180,663	479,766	2,266,615	1,496,147	0	12
EntityPayment	1	388,848,558	89,972,467	1,043	301,532	2,180,663	479,766	2,266,615	1,496,147	0	20,177,319
ProviderSchedule	0	363,662,057	93,384,674	1,082	921,643	529,227	[illegible]	[illegible]	[illegible]	0	3
ProviderSchedule	1	363,662,057	270,277,383	3,133	9,589	529,227	[illegible]	[illegible]	[illegible]	0	90,017,828
ScheduleModification	0	349,035,595	141,100,184	1,635	737,403	69,297,838	[illegible]	[illegible]	[illegible]	0	0
ScheduleModification	1	349,035,595	6	0	6	69,297,838	[illegible]	[illegible]	[illegible]	0	4
ScheduleModification	2	349,035,595	207,731,934	2,408	639,524	217,233	7	0	0	0	51,382,966
ScheduleModification	3	349,035,595	203,471	2	160	36,674	0	0	0	0	21
ProviderProvisions	0	341,192,246	252,341,557	2,925	182,868	31,191,349	795	156	0	1,861	2,666
ProviderProvisions	2	341,192,246	76,644,611	888	11,090	8	111	0	0	0	40,850,838
ProviderProvisions	3	341,192,246	14,821	0	982	1,314	717	912	756	0	77
ProviderProvisions	4	341,192,246	11,841	0	1,632	499	628	910	754	0	1,394
ProviderProvisions	5	341,192,246	1,833,979	21	16,316	13,719	270	814	658	0	3,400,017
ProviderProvisions	6	341,192,246	3,355,969	38	56,264	40,539	356	329	173	0	2,823,304
ProviderProvisions	7	341,192,246	4,873	0	120	3	211	814	658	0	47
ProviderProvisions	8	341,192,246	7,504	0	374	40	252	814	658	0	248
ProviderProvisions	9	341,192,246	824,806	9	14,116	646	194	227	71	0	207,618
ProviderProvisions	10	341,192,246	1,600,890	18	20,397	0	283	849	693	0	529,967
ProviderProvisions	11	341,192,246	4,551,395	52	6,779	12,336	292	875	719	0	1,864,902
CustomUserField_D	0	298,322,608	294,498,452	3,414	42,181	752,021	140	52	0	246	52
CustomUserField_D	1	298,322,608	3,824,156	44	517	752,021	140	52	0	246	1,274,658
#tiv_bas00058950021666137	0	284,067,888	97,866,203	1,946	24	1,095	860,375	46,306,910	0	0	355
#tiv_bas00058950021666137	1	284,067,888	186,201,685	3,703	18	1,095	860,375	46,306,910	0	0	46,252,240

Not Heap Insert
(Indid=1 is clustered index)

STORED PROCEDURES

ISOLATING APPLICATION ISSUES

MONITORING PROC CACHE

monProcedureCacheModuleUsage and monCachedProcedures

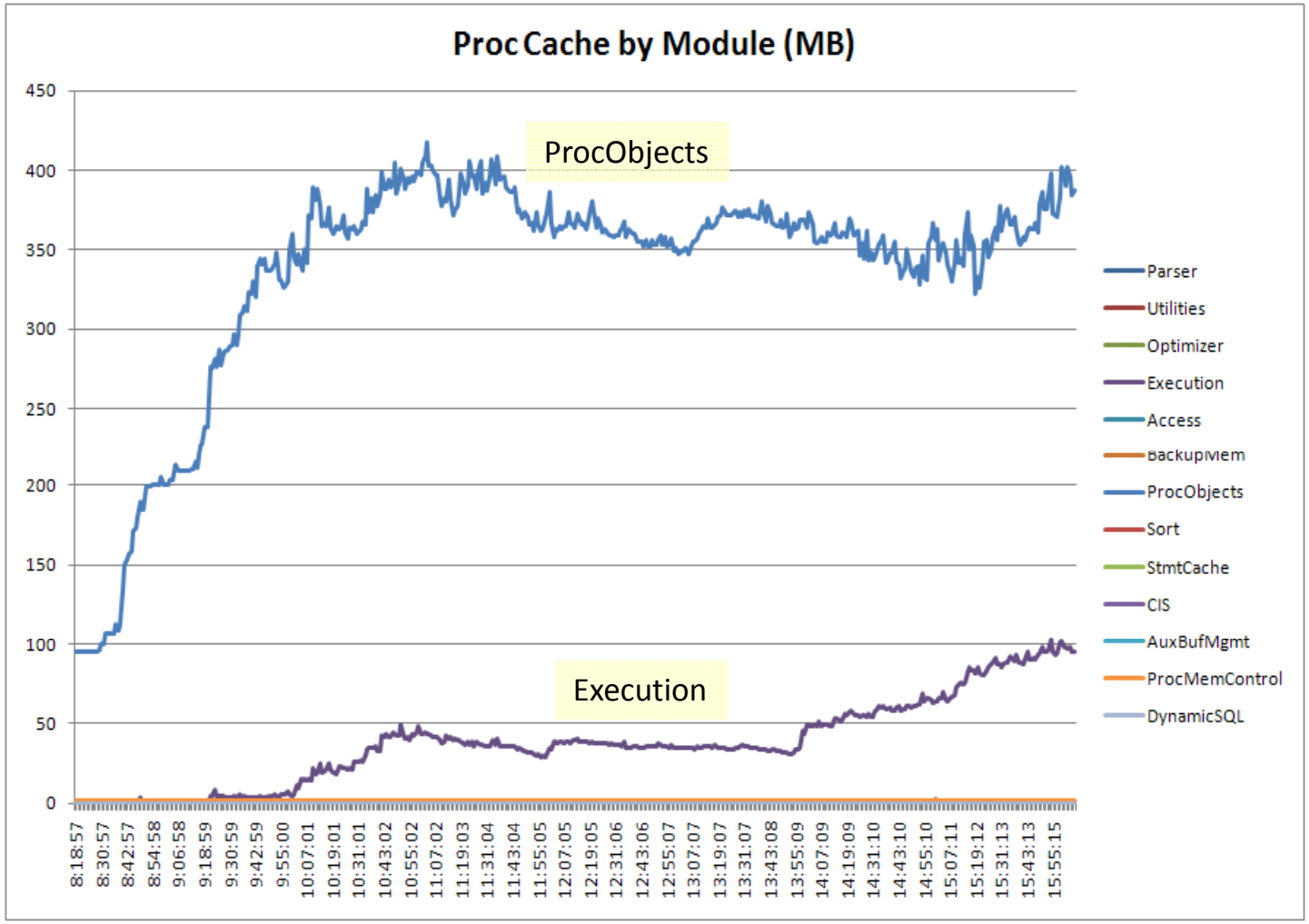
- **monProcedureCacheModuleUsage**
  - ✓ Good for watching the HWM of non proc usage of proc cache (e.g. sorts, etc.)
  - ✓ Although the HWM's can occur at different times, use the max(HWM) for each module for configuration
  - ✓ Try to avoid NumPagesReused > 0
- **monCachedProcedures**
  - ✓ Interesting stats
    - PlanID → Proc Concurrency (count)
    - CompileDate → Plan Lifespan
    - RequestCnt → Proc Execs during lifespan
    - MemUsageKB → Proc cache usage (sum)
  - ✓ May be more accurate (or shall we say more conservative) than using monProcedureCacheModuleUsage for sizing….
    - …you'll see in a minute

monProcedureCacheModuleUsage		
InstanceID	tinyint	<pk>
ModuleID	int	<pk>
Active	int	
HWM	int	
NumPagesReused	int	
ModuleName	varchar(30)	

monCachedProcedures		
ObjectID	int	<pk>
InstanceID	tinyint	<pk>
OwnerUID	int	<pk>
DBID	int	<pk>
PlanID	int	<pk>
MemUsageKB	int	
CompileDate	datetime	<pk>
ObjectName	varchar(30)	<pk>
ObjectType	varchar(32)	
OwnerName	varchar(30)	<pk>
DBName	varchar(30)	<pk>
RequestCnt	int	
TempdbRemapCnt	int	
AvgTempdbRemapTime	int	

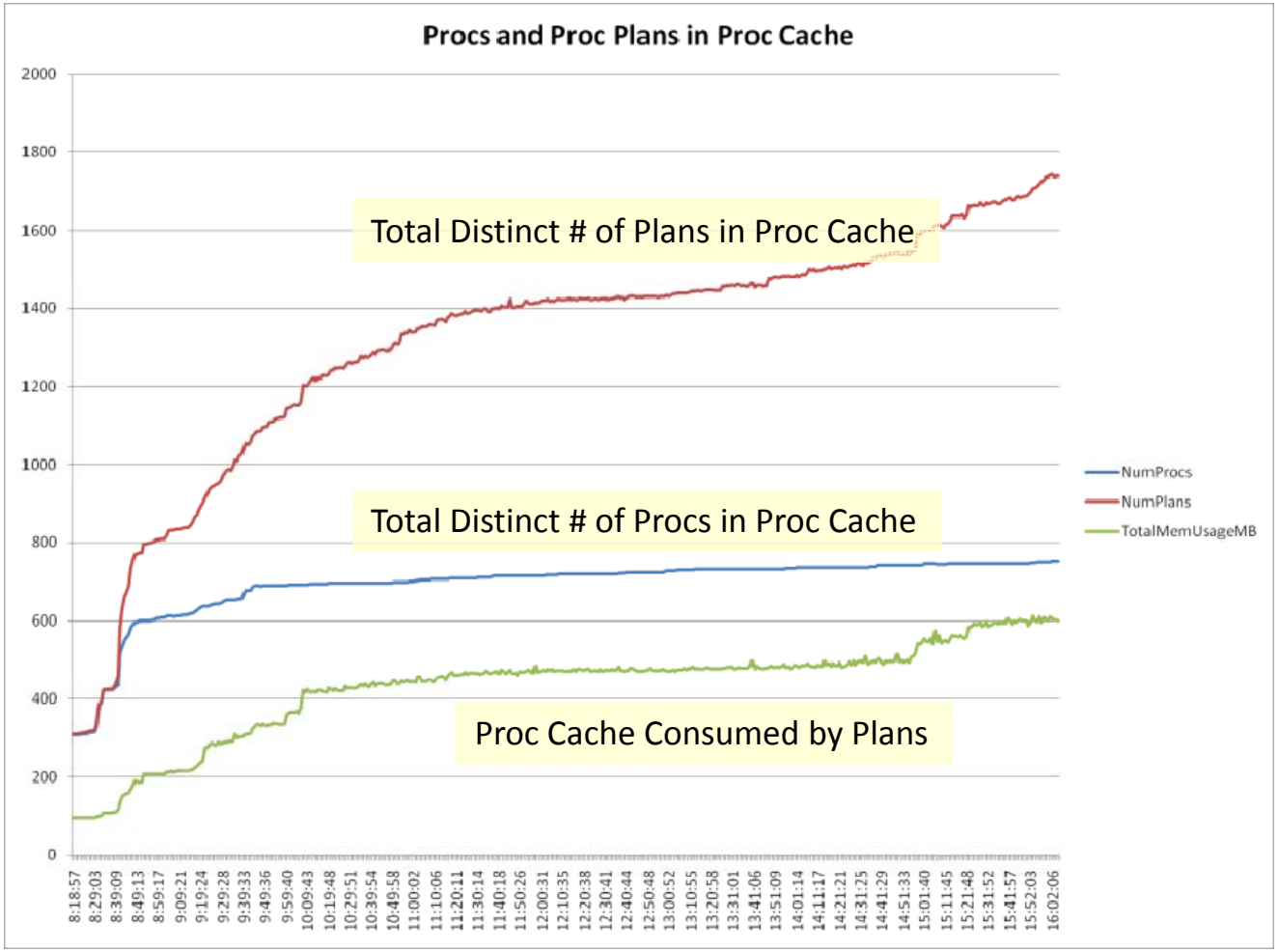
PROC CACHE USAGE BY MODULE

monProcedureCacheModuleUsage



…A DIFFERENT WAY OF LOOKING AT IT

Aggregating monCachedProcedures

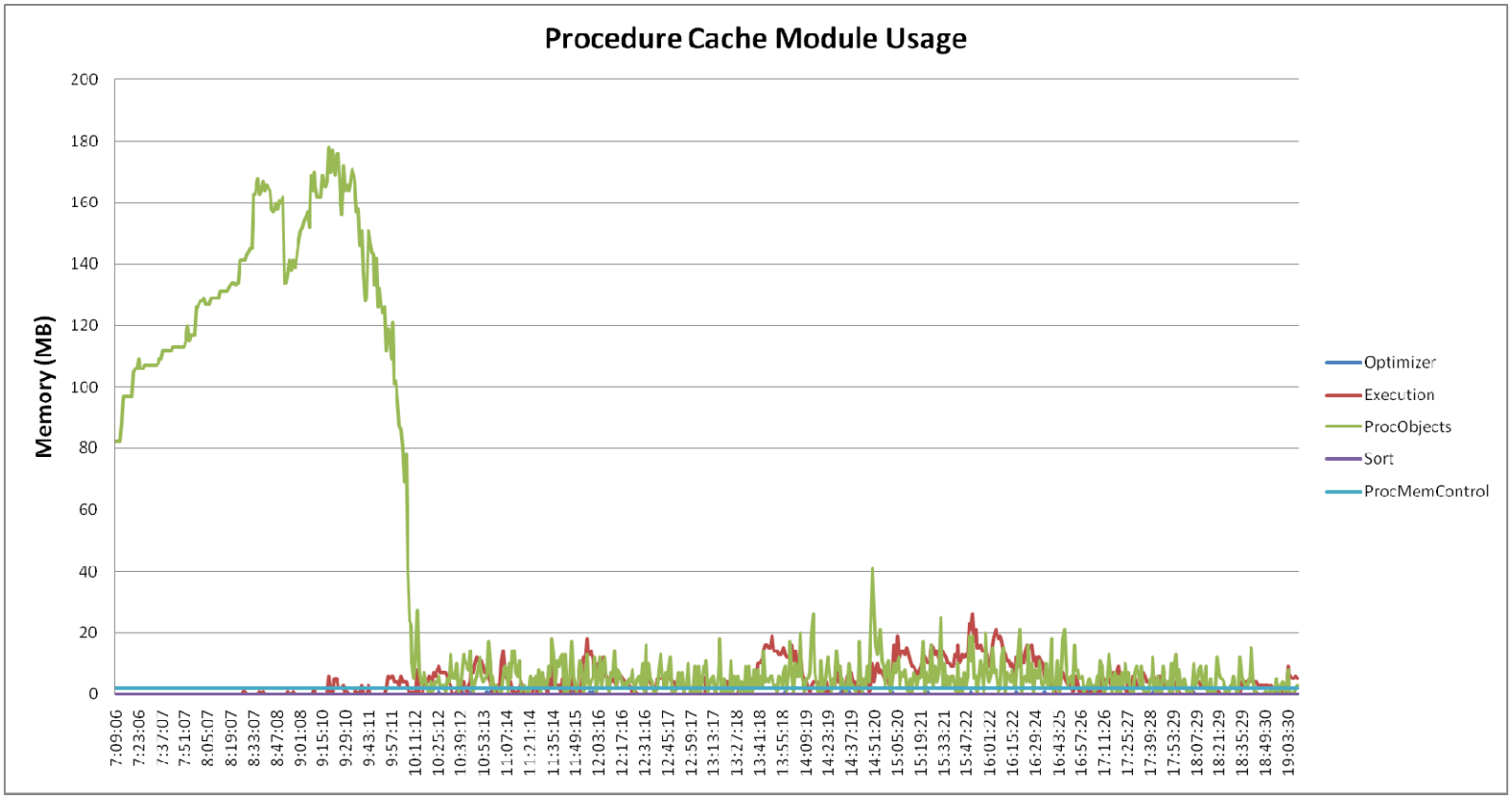


SYBASE | An SAP Company

OUR CUSTOMER: LIFE GETS INTERESTING

So….How Do You Explain This One???...and yes, the system is busy (50-60% CPU)

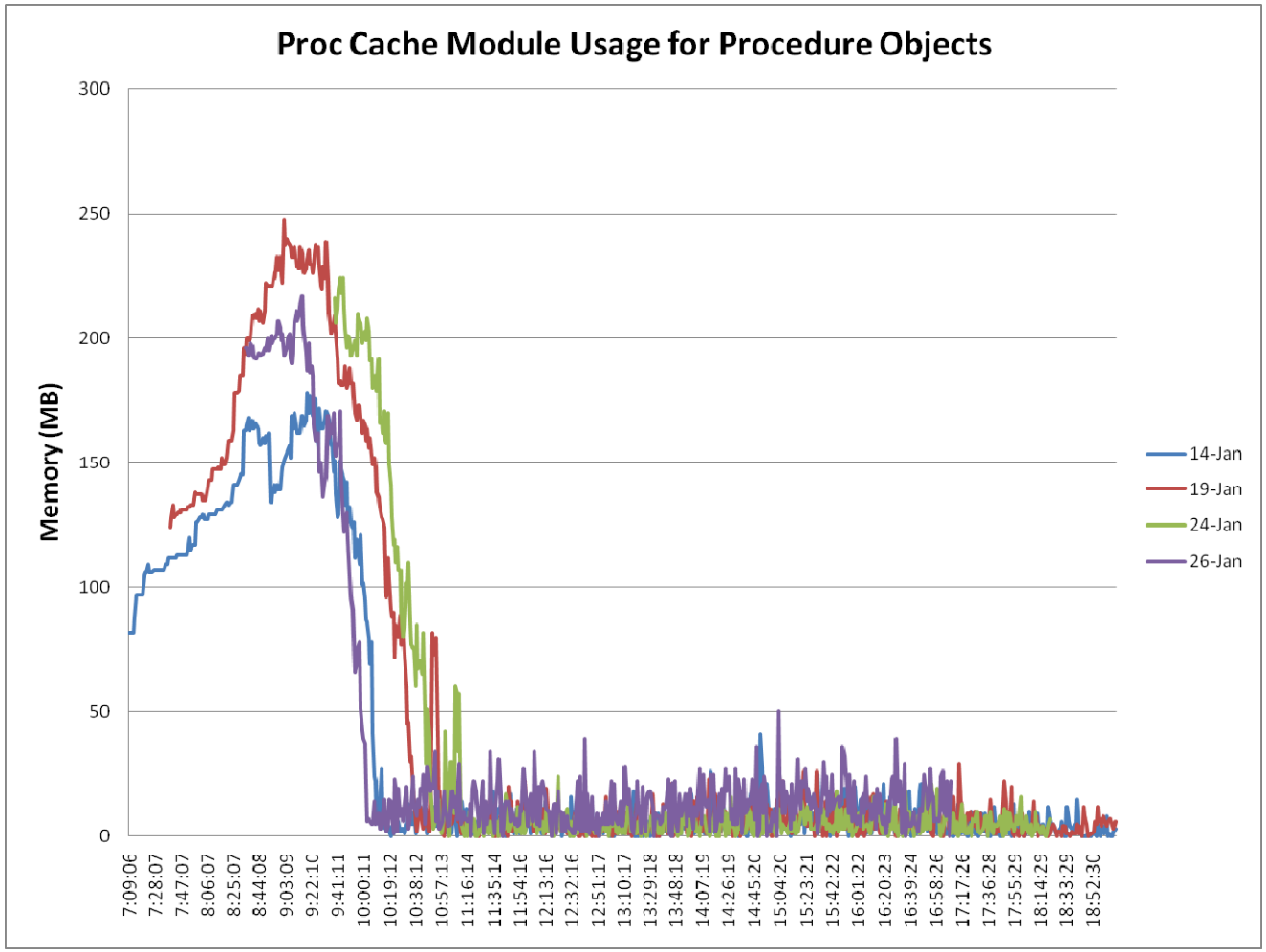
Further info to confuse the issue: Adhoc queries=2,890,000; monSysStatement Proc Lines executed=17,041,767; Distinct PlanID's in monSysStatement= 238,447



SYBASE | An SAP Company

BEST PART - IT IS NORMAL

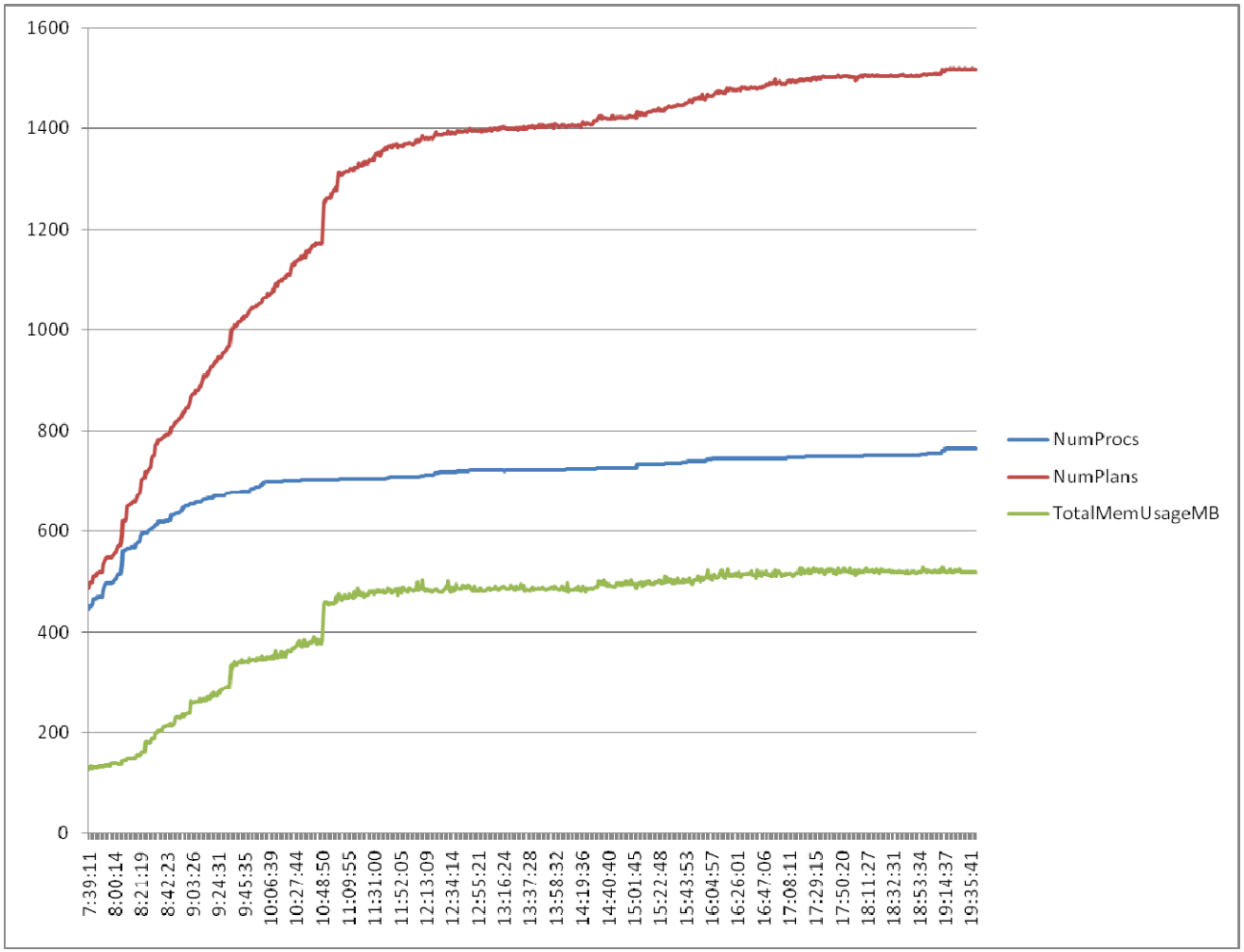
Baselining monProcedureCacheModuleUsage



SYBASE | An SAP Company

….BUT…

monCachedProcedures for same day shows ~550MB of proc cache used…



SYBASE | An SAP Company

HOW MUCH PROC CACHE

Do you really need 5GB???

- **Procedures recompile more frequently than you think**
  - ✓ Schema change
    - Which happens every time a sub-proc hits a #temp table that wasn't defined in the sub-proc
  - ✓ Deferred Proc Recompile
    - If enabled, when proc hits @var or #temp for the first execution
  - ✓ Other times….
  - ✓ Result of a recompile is that the old plan is removed and new plan created
  - ✓ Looking at monCachedProcedures.CompileDate….
    - This customer had some frequently executed procs with the longest lifespan measured in less than 100ms
    - A few primarily batch procs stayed in cache infinitely
    - Customer (and TS?) had bought into the over exuberance of proc cache fragmentation in ASE 15 and ran dbcc proc_cache(free_unused) every morning at 6am…..rather needlessly
      - This was based on sp_sysmon output showing high recompile/cache removals/proc reads from disk that seemed to suggest proc cache was full
- **Proc Cache Sizing**
  - ✓ Use combination of monProcedureCacheModuleUsage and monCachedProcedures
    - Use the higher of the two…and then pad by a bit….
    - Then add in the HWM for other modules

BEST PRACTICES: NOT JUST MDA DATA

Data to collect along with MDA data

- **Object Metadata**
  - ✓ Loosely based on sysobjects
  - ✓ May be needed to decode ObjectID's, especially if a schema change
    - E.g. dropping and recreating a proc - it gets a new objectid
  - ✓ Information to collect
    - Collection, Scenario, Server
    - Database name, Database ID, Object Name, Object ID
    - Owner UID, Owner Name
    - Locking (datarows, allpages, etc.)
    - Row Count
- **Index Metadata**
  - ✓ Loosely based on sysindexes + syscolumns
  - ✓ Helpful for later determining which columns are in an index
  - ✓ Information to collect
    - Collection, Scenario, Server
    - Database name, Database ID, Object Name, Object ID, Index Name, Index ID
    - Key count, IsUnique?, IsPKey?, IsFKey?, IsPKeyUniqueConstraint?, IsFunction?
    - List of indexed columns


SYBASE®
An SAP Company