SYBASE
An SAP Company

APPLICATION MONITORING & DIAGNOSIS WITH MDA

JEFF TALLMAN
DIRECTOR, TECHNICAL ENGINEERING, SYBASE

JANUARY 2012

APPLICATION MONITORING & TUNING

Part 1 of the Series on Using MDA

- **Business Layer Monitoring**
  - ✓ Requirements
- **Resources & Key Performance Metrics**
  - ✓ Measuring Application Workloads & Resource Utilization
  - ✓ Baselining Performance
- **Objects**
  - ✓ Spotting indexing issues
  - ✓ Baselining for problem isolation
- **Stored Procs & Proc Cache**
  - ✓ Finding problematic statements/isolating application issues

BEFORE WE BEGIN….

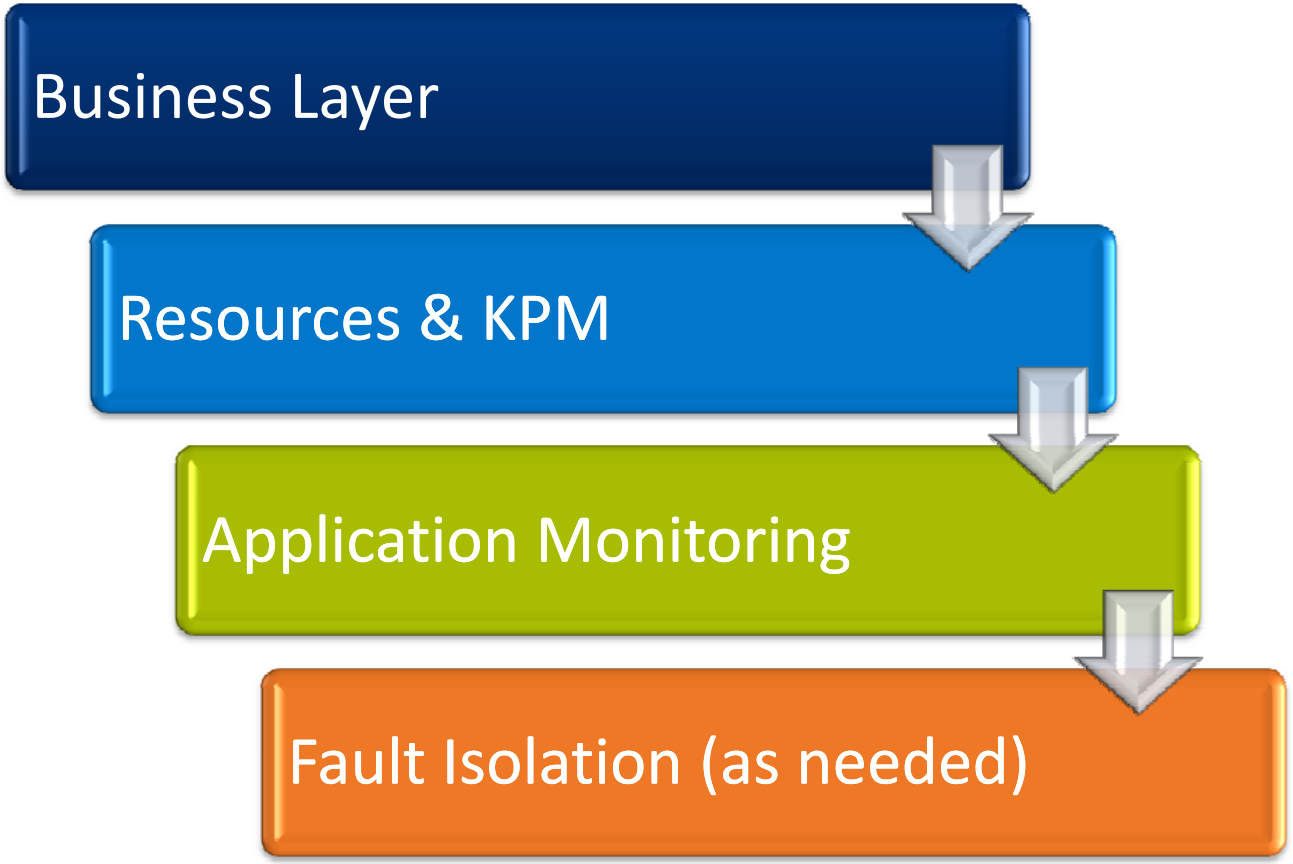
…A word about the customer examples in this presentation

- **The examples in this presentation are from two primary (anonymous) customers**
  - ✓ Table names/proc names may not make sense as they were obfuscated
  - ✓ Customer A: Bank → metrics and application baselining
  - ✓ Customer B: HealthCare → process activity, application fault isolation
  - ✓ Other anecdotal examples from other customers
- **None of the customers did anything "wrong"**
  - ✓ In fact, the one used for most of the presentation (A) generally followed TS advice and sp_sysmon recommendations
    - Problem is sp_sysmon aggregates too highly and evidence of what was happening was not visible as applications swapped characteristics or symptoms sp_sysmon exposed where due to other causes sp_sysmon did not diagnose entirely
  - ✓ Just pointing out that if you are going to continue to use sp_sysmon - you will need MDA to really diagnose what is happening and to verify the recommendations it is making

BUSINESS LAYER MONITORING

A CRITICAL COMPONENT FOR ANY ENVIRONMENT

MONITORING FRAMEWORK



BUSINESS LAYER MONITORING (1)

A Key Foundation Requirement

- **Report Statistics in Business Terms**
  - ✓ Transactions by type per minute (e.g. ticket sales per minute)
  - ✓ Reports by type per hour (e.g. regional sales report per hour)
  - ✓ Queries by type per minute (e.g. product inventory check)
- **Break out by application/system**
  - ✓ Customer Service application did 15 price lookups per minute using appserver farm E and DBMS service T (service not server)
  - ✓ Customer portal web application did 1500 stock performance trend reports per hour using appserver cluster X and DBMS service Y
  - ✓ Trading desk application did 15,000 stock performance trend reports per hour using CEP pipeline A and DBMS service B

BUSINESS LAYER MONITORING (2)

A Key Foundation Requirement

- **Don't confuse technology metrics**
  - ✓ Queue depth in application servers or application response times
    - These are symptoms of a problem - can't be used to explain a problem
  - ✓ Transactions per minute
    - Unless you know whether the transactions are similar to other periods, this metric is only a symptom as well
- **But …DON'T just rely on business level metrics**
  - ✓ Reporting performance differences in X business units per hour is not helpful unless you are also collecting and analyzing the corresponding technology metrics

THE GOOD, BAD & UGLY….

Justifying Business Level Monitoring

- **The Business Will (or Should) Pay For It**
  - ✓ Helps them (and you) justify IT expenditures (xxx% growth vs. last year)
  - ✓ Helps them better quantify real-time business behavior and unit performance with the perspective of IT capacity
    - Business analytics systems typically can't help completely as the business performance metrics are not associate with which applications or systems were used.
- **Critical for your debugging….**

TRUE CUSTOMER STORY

Why Business Level Monitoring is Critical

- **Customer ALARGEBANK**
  - ✓ System was 100% pegged for several weeks/months
  - ✓ Business was suffering repeated 'slow-downs' and aborted processes as DBA's manually killing reports and other (assumed) resource consumptive processes
- **Culprit:**
  - ✓ A web-service that reported short-term interest rates was being called 5,000 times per second by 1 partner
  - ✓ Kicker: The interest rate only changed once every 7 days
  - ✓ Why did it take so long to find:
    - Nothing in ASE pointed out any issues as the query ran very fast.
    - Only when the applications were isolated by using engine groups was it noticed that it was a specific application causing the problem and detailed checking of the middletier components revealed which web services were busy.
  - ✓ Lesson: Too many customers rely exclusively on ASE monitoring vs. monitoring the entire stack

ANOTHER CUSTOMER STORY (1)

Another Large Bank

- **Migration to ASE 15 was a nightmare**
  - ✓ customer's perception…sorta reality sorta not
  - ✓ Took over a year to stabilize
- **Ended up using compatibility mode until application fixes implemented**
  - ✓ Needed about 30 new indexes
  - ✓ A small number of ad-hoc queries needed to be fixed (bad SQL)
  - ✓ Compatibility mode blocked use of semantic partitions and other desired features
- **Customer had implemented crude middle-tier and database contention monitoring as bell weather**
  - ✓ Tracked the application queue depths for requests pending connections from the ASE connection pools
  - ✓ Tracked the contention via number of blocked processes (but not on what)

ANOTHER CUSTOMER STORY (2)

Another Large Bank

- **Customer perception:**
  - ✓ every time compatibility mode was turned off:
    - CPU utilization climbed in ASE
    - blocking became more frequent
    - application queue depths climbed
- **Reality:**
  - ✓ Business workload changes was driving CPU differences
    - DBA's were operating under the assumptions that workloads would be similar each day, just minor variances in volume.
    - Business was launching new campaigns periodically, had scheduled payment dates to partners, etc.
  - ✓ Business workload changes caused increased log writes - log devices were slow - which increased lock hold times - which increased contention.
- **Literally million$$$ was spent chasing the problem**
  - ✓ Not to mention executive level meetings, calls, etc……

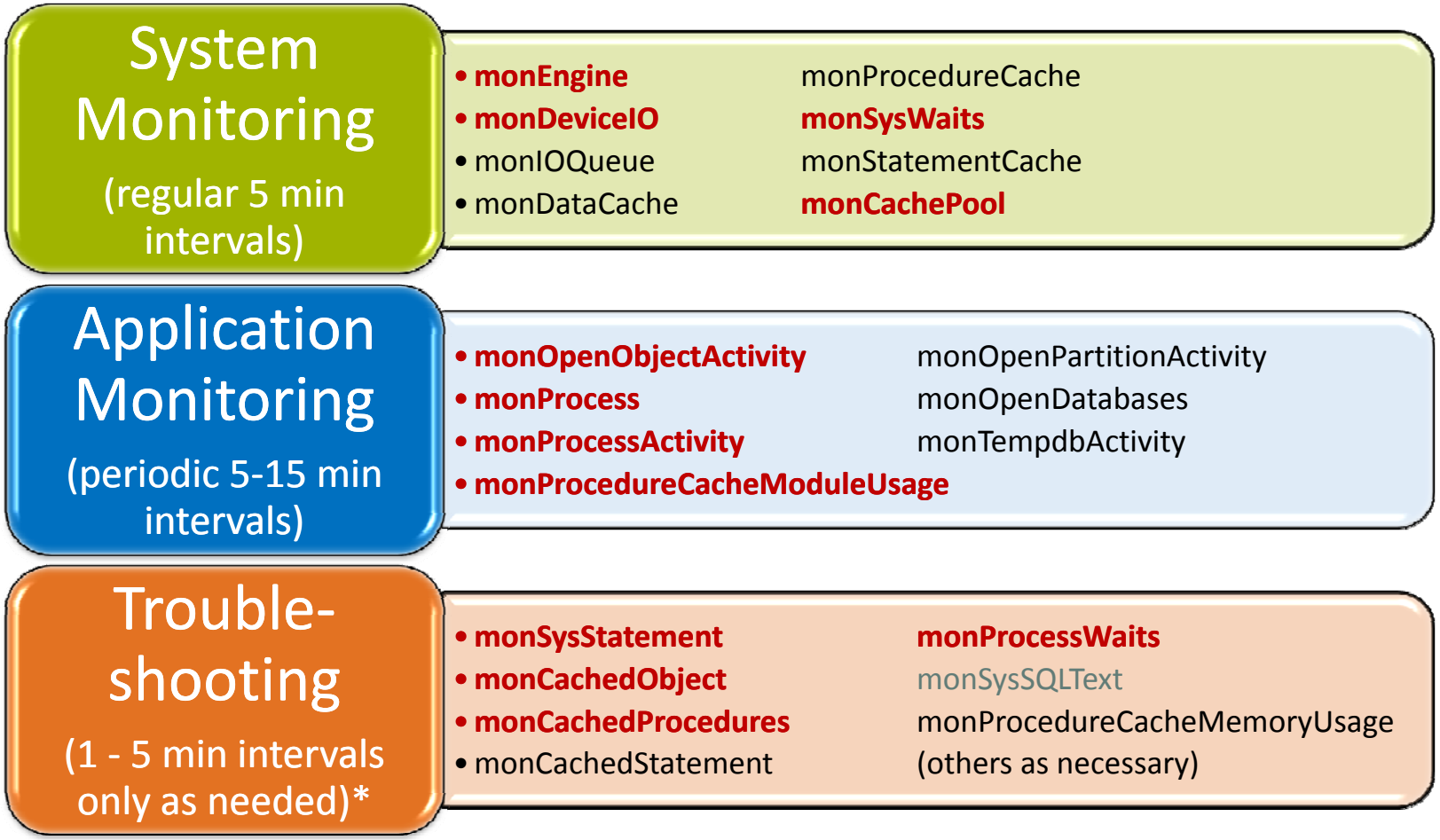
KEY BUSINESS MONITORING POINTS

Beyond Just Collecting the Data

- **Collect In Business Terms…associated with apps/systems**
- **Data collection will require application changes**
  - ✓ Middle-tier needs to track as it is much closer to business context and user role than database server is.
  - ✓ Database performance metrics can be used to help diagnose application issues, …
    - …but without application monitoring, you will only be able to conclude a workload change ….
    - ….You won't know the business event that triggered it.
- **DBA & SysAdmins**
  - ✓ should have (near) real time access to the data
  - ✓ ….only way to correlate the information

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

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

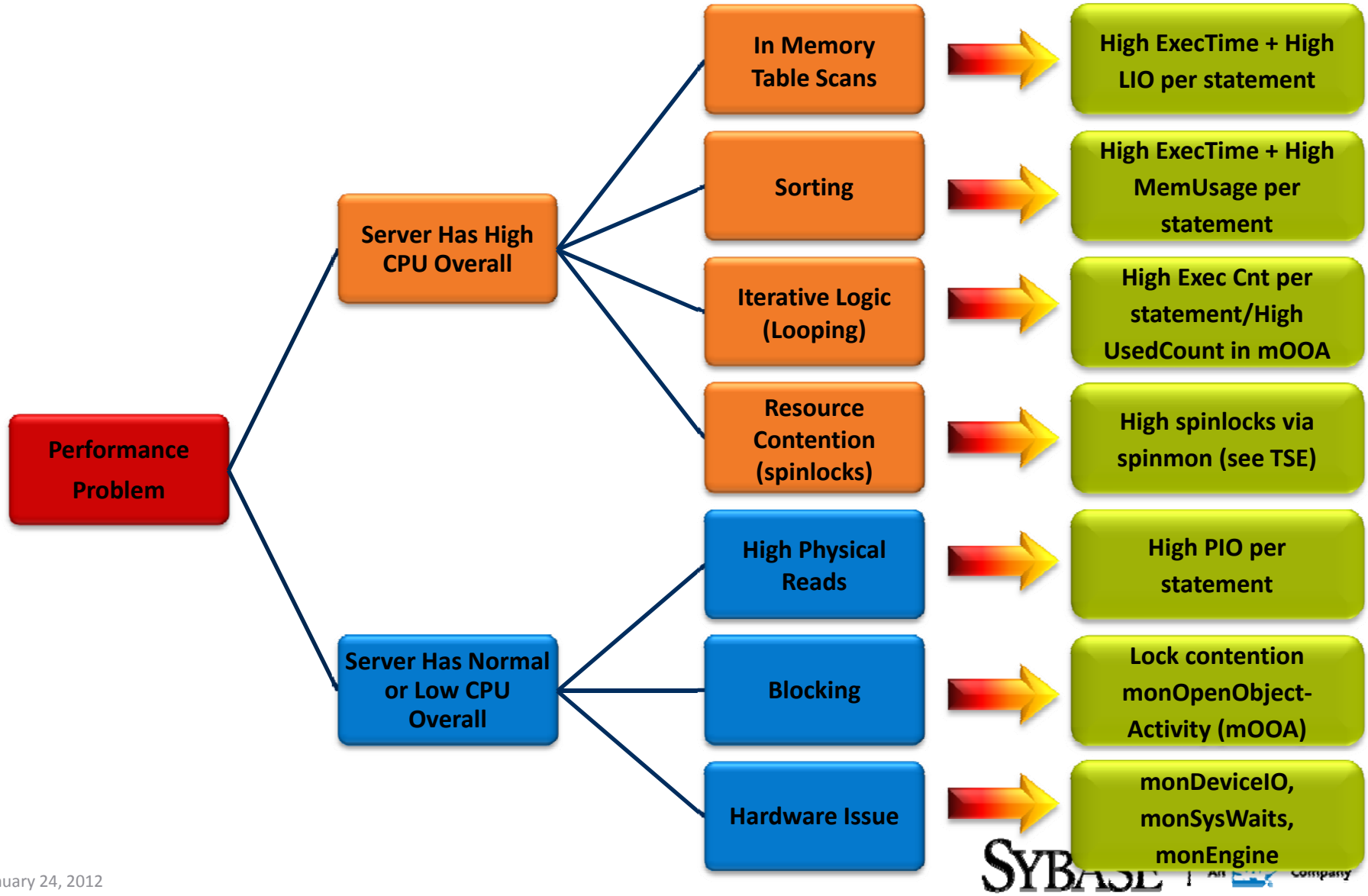
```
Select sample_num=identity(20), SPID, KPID, SampleTime, LogicalReads
  into #ordered_LIOs
  from monProcessActivity
  order by SPID, KPID, SampleTime
```

    - Then join in queries (similar to above)

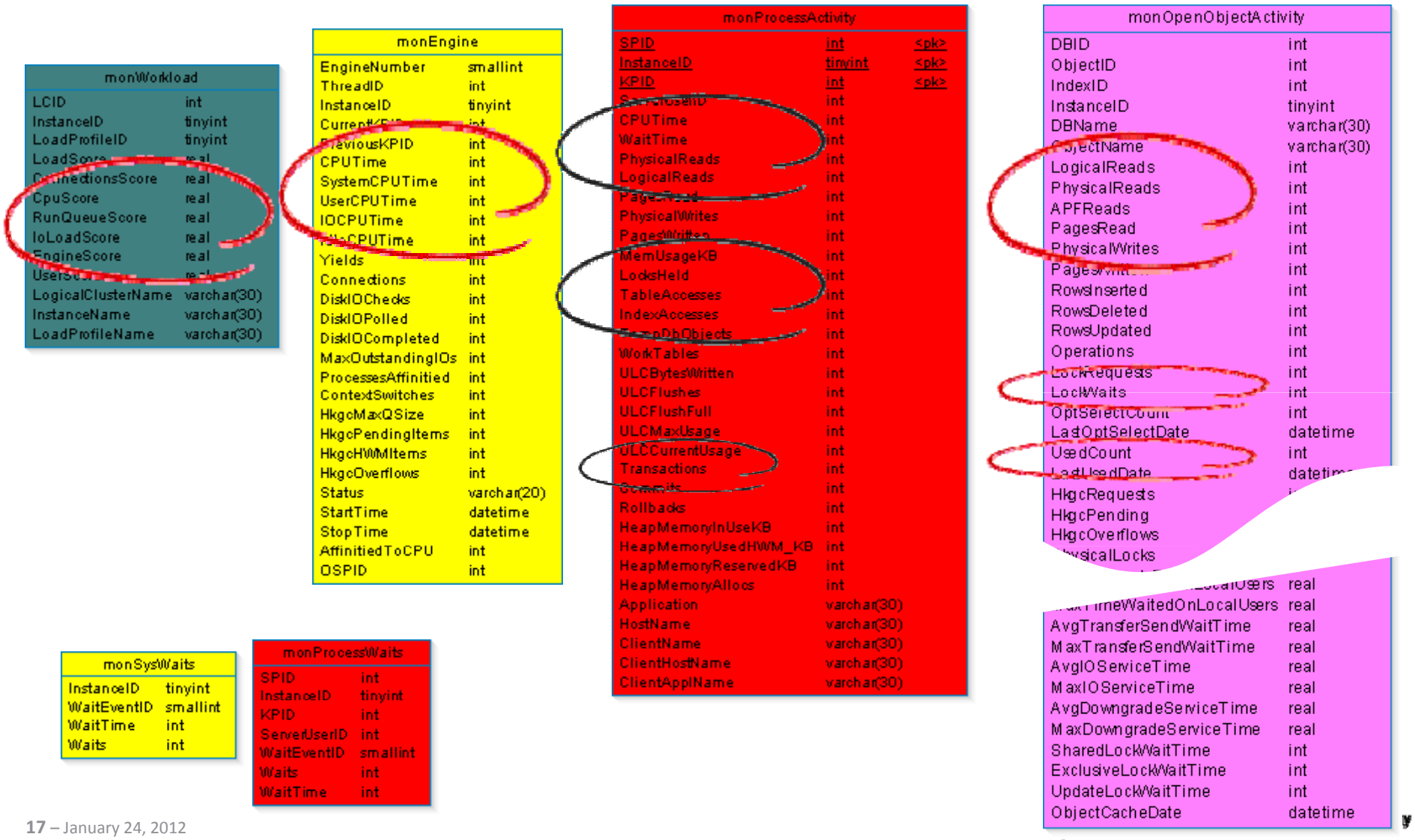
FAULT ISOLATION

HOW TO ACTUALLY FIND THE CAUSES

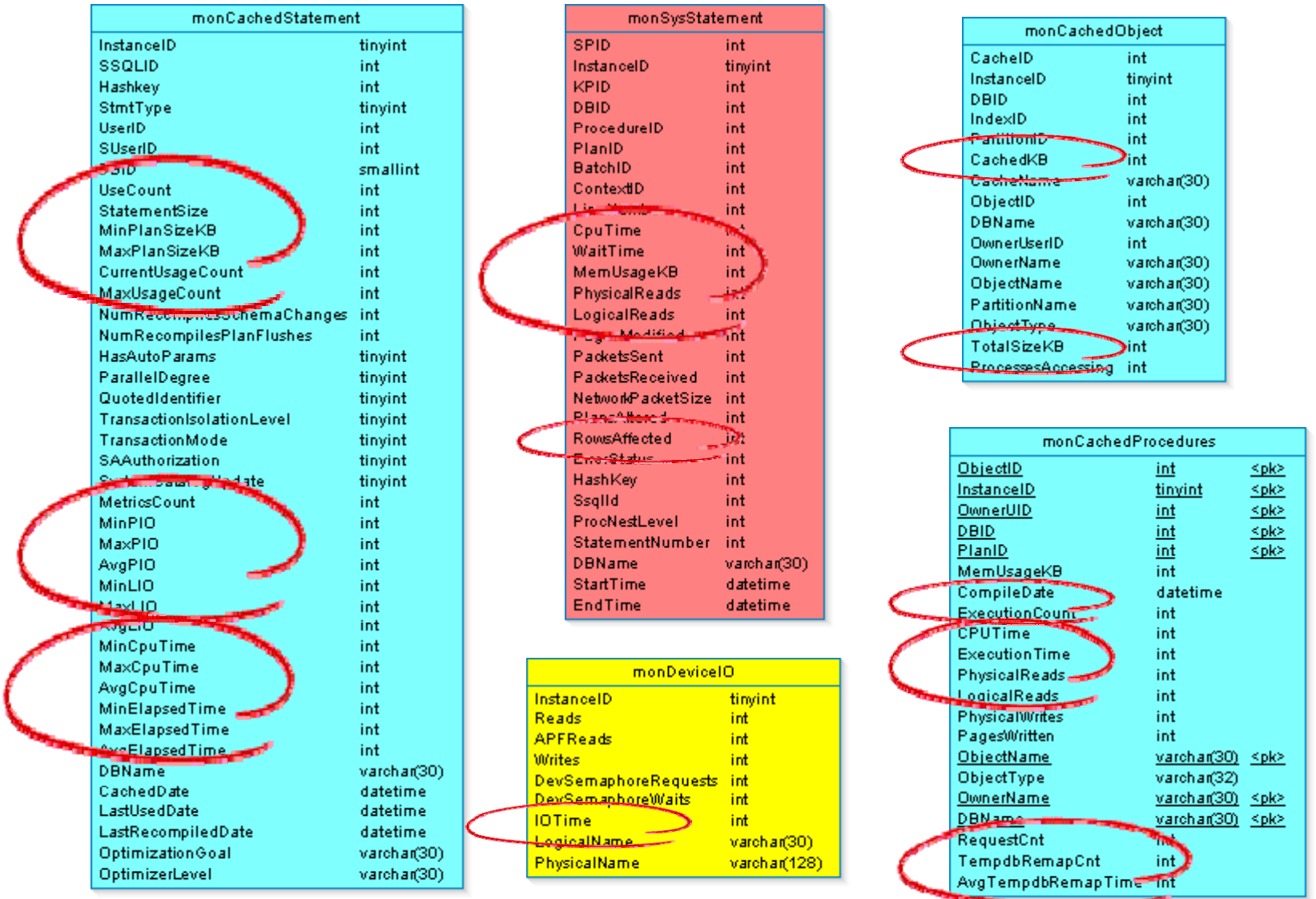
PERFORMANCE FAULT ISOLATION



"QUICK VALIDATION" MDA TABLES



“FAULT DETERMINATION” MDA TABLES



SYBASE | An SAP Company

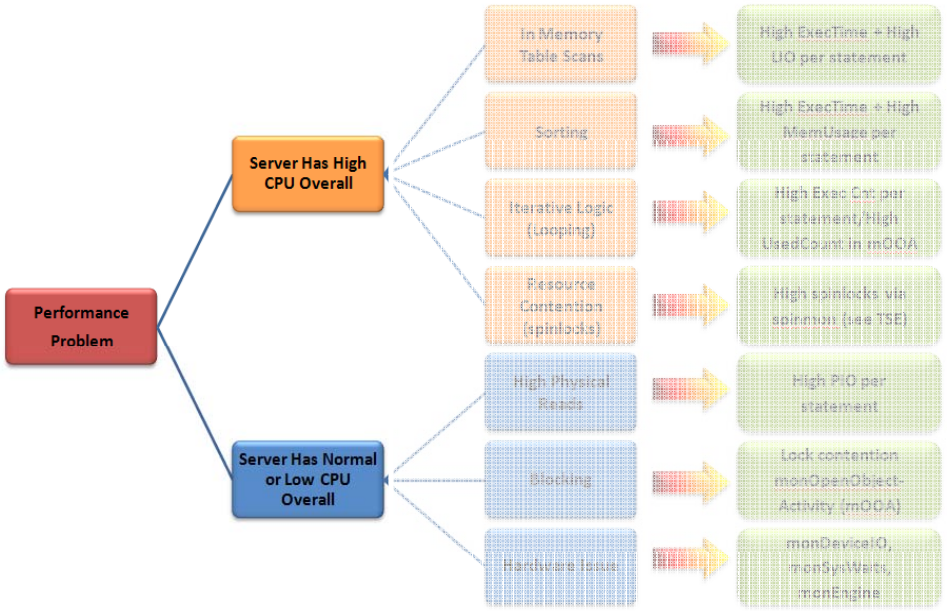
FAULT DETERMINATION WITH QP METRICS

You Notice I Didn't Discuss SQLText and PLANText pipes…..There is a better way…..

- **Use filters (sp_configure)**
  - ✓ enable metrics capture = DEFAULT
  - ✓ metrics lio max = DEFAULT
  - ✓ metrics pio max = DEFAULT
  - ✓ metrics elap max = 5000
  - ✓ metrics exec max = DEFAULT
- **Use login trigger**
  - ✓ "set metrics_capture on" for specific applications or representative number of users
  - ✓ Use with filters (above) to avoid chaff
  - ✓ Use with "capture missing statistics" to speed up missing/bad indexing
- **Post capture**
  - ✓ Fine tune filter with sp_metrics
  - ✓ Backup for regression comparison (sp_metrics)

ELAPSED VS. EXECUTION TIME

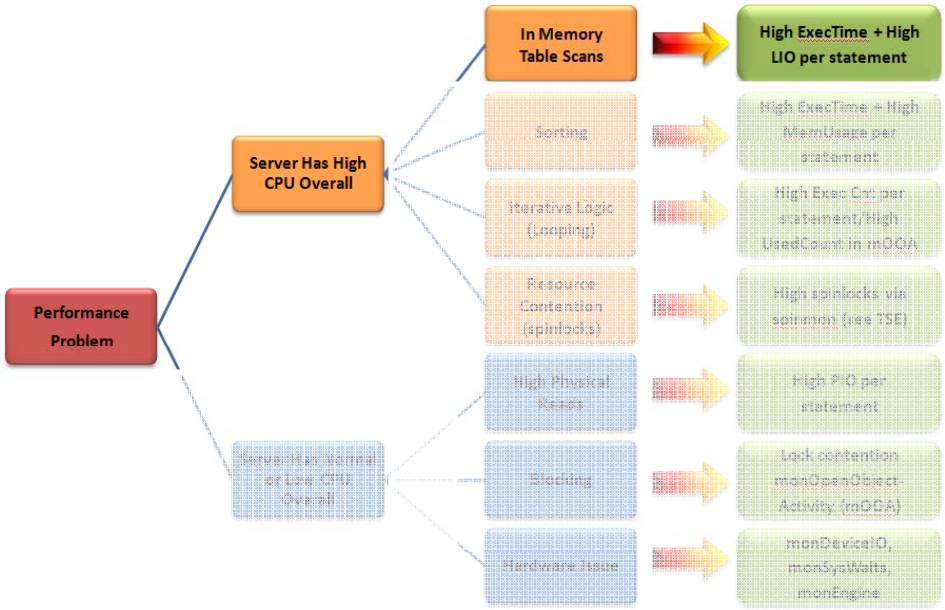
Step 1: Check CPU utilization



- **Check CPU ~100%**
  - ✓ Sp_sysmon
  - ✓ monEngine
  - ✓ Sybase Control Center
- **If High or Higher than Normal**
  - ✓ Problem is execution time related
- **If Normal → <89%**
  - ✓ Problem is elapsed time driven

POSSIBLE CAUSE: IN-MEMORY SCANS

In-Memory Table, Index or Large Range Scans

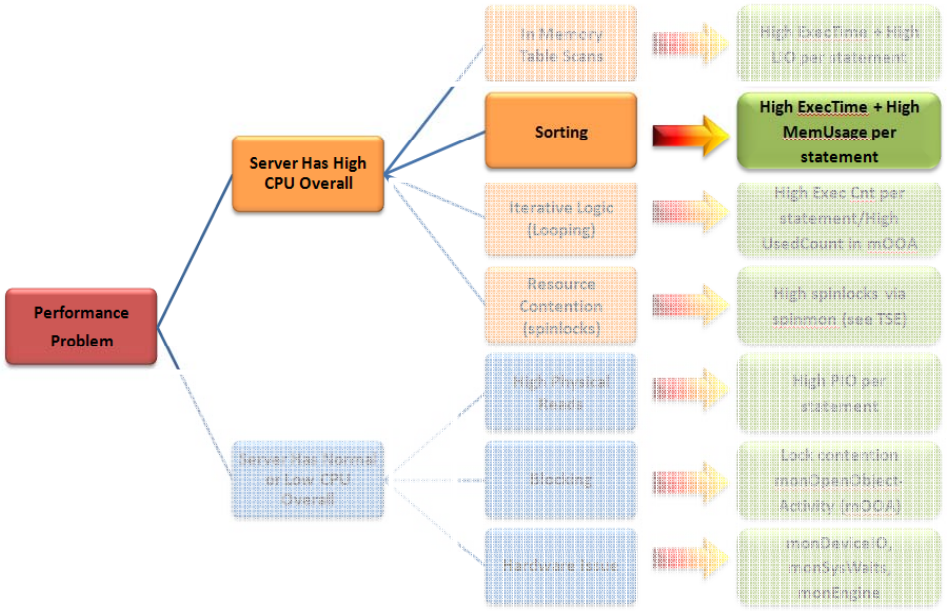


- **Key Symptoms**
  - ✓ high execution time compared to elapsed time
  - ✓ high LIO for the statement vs. rows returned on average
- **Quick validation**
  - ✓ monProcessActivity
  - ✓ monOpenObjectActivity shows high LIO and low PIO for particular objects
  - ✓ LIO disproportionately high
- **Root cause determination**
  - ✓ monCachedStatements
  - ✓ QP Metrics
  - ✓ monSysStatement

SYBASE | An SAP Company

POSSIBLE CAUSE: IN-MEMORY SORTING

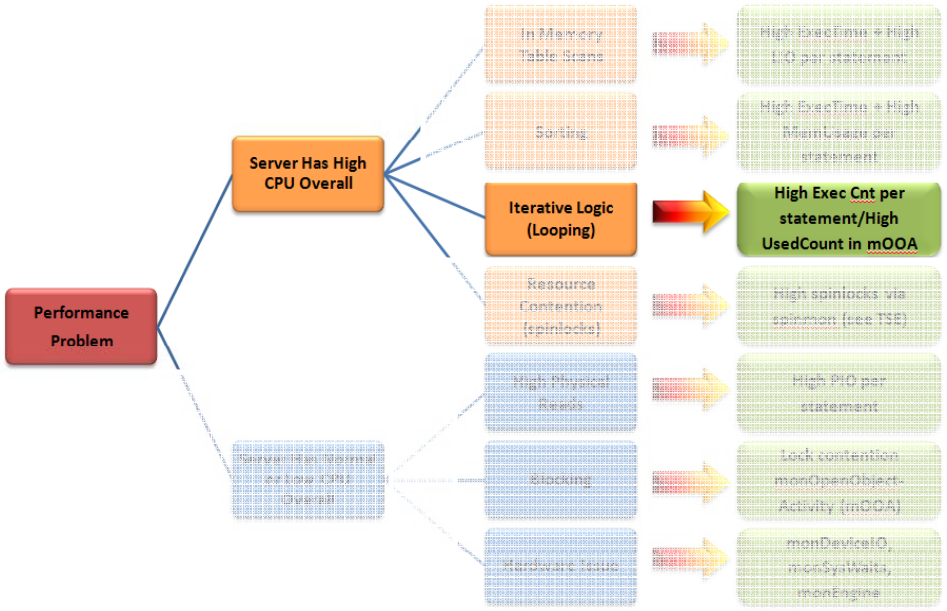
New to ASE 15.x → would have been low CPU/high IO in 12.5.x



- **Key Symptoms**
  - ✓ high execution time compared beyond LIO time (e.g. >1ms per LIO)
  - ✓ high memory usage per statement on average
- **Quick validation**
  - ✓ monOpenObjectActivity shows tablescans or high LIO on table
  - ✓ monProcedureCacheMemoryUsage HWM
  - ✓ monSysStatement/monCachedStatement memUsageKB
- **Root cause determination**
  - ✓ monCachedStatements
  - ✓ QP Metrics
  - ✓ monSysStatement

POSSIBLE CAUSE: ITERATIVE LOOPING

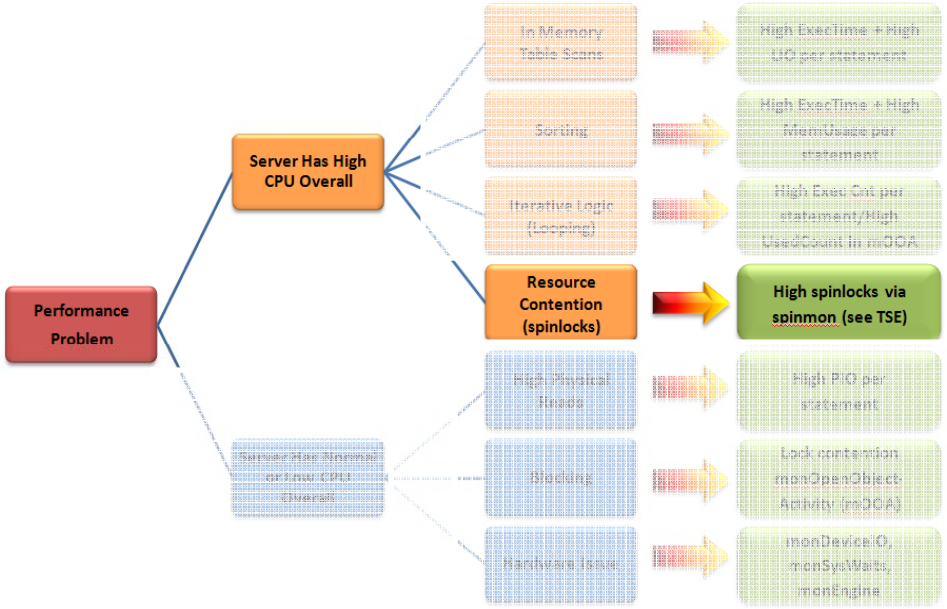
Common for Batch Processes as well as Reporting



- **Key Symptoms**
  - ✓ High aggregated execution time low LIO/avg exec time
  - ✓ high network waits
- **Quick validation**
  - ✓ monOpenObjectActivity shows high UsedCount
  - ✓ monProcessWaits shows high network wait times
  - ✓ monProcessActivity shows high Transactions
  - ✓ monSysStatement (aggregated) /monCachedStatement Use counts
- **Root cause determination**
  - ✓ monCachedStatements
  - ✓ QP Metrics
  - ✓ monSysStatement

POSSIBLE CAUSE: RESOURCE CONTENTION

In SMP, There Always Is Contention - Eliminate Everything Else First

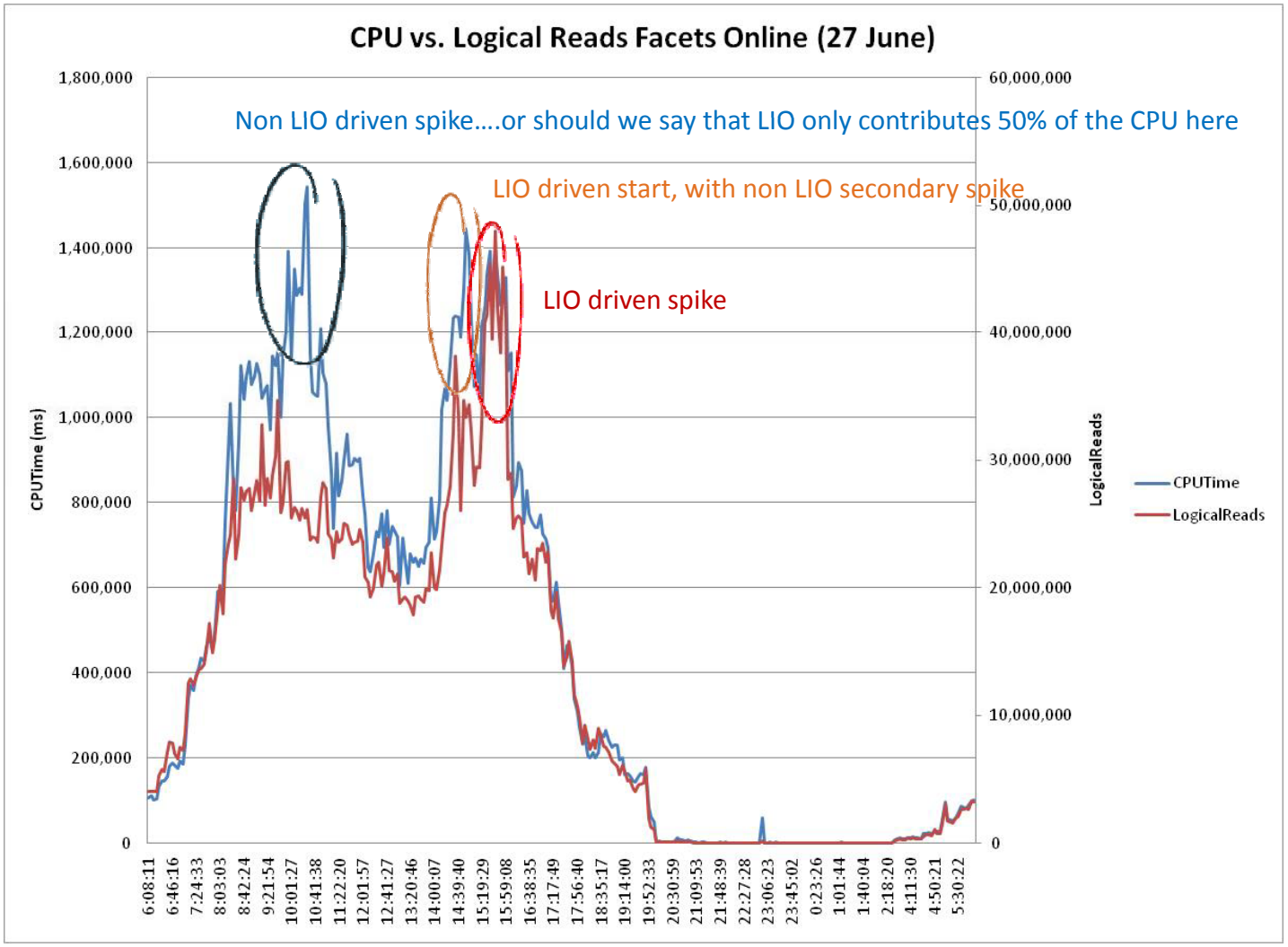


- **Key Symptoms**
  - ✓ High CPU without high LIO
  - ✓ High CPU without high memory
    - In-memory sorts
- **Quick validation**
  - ✓ Sp_sysmon
  - ✓ Spinmon (work with TSE)
- **Root cause determination**
  - ✓ monStatementCache
  - ✓ monProcedureCache
  - ✓ Spinmon
  - ✓ monProcessActivity
  - ✓ monTempdbActivity

SYBASE | An SAP Company

AN EXAMPLE

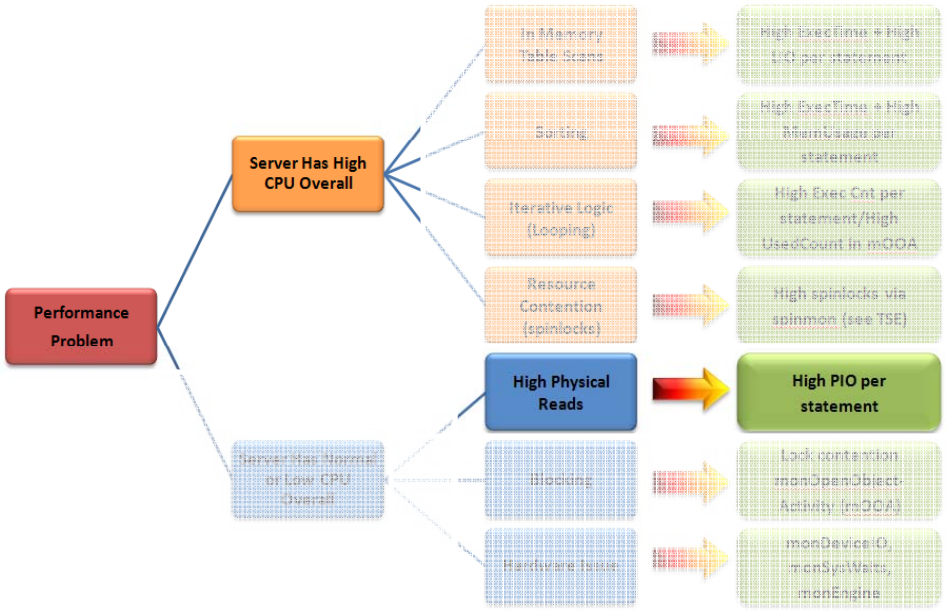
Composite Causes….as Seen from monProcessActivity



SYBASE | An SAP Company

POSSIBLE CAUSE: PHYSICAL READS

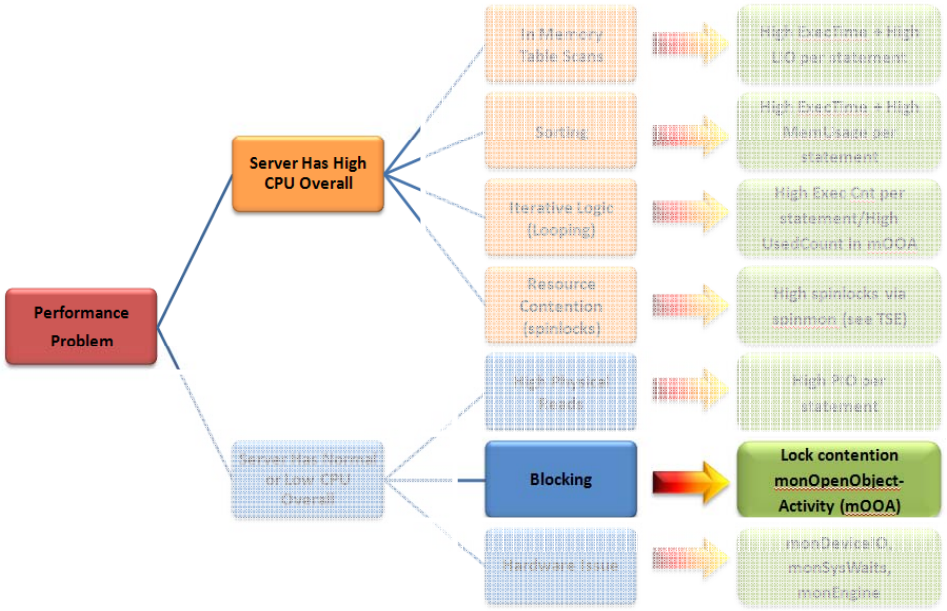
Disks Are The Slowest Device In The Computer….Any PIO Will Be Slow.



- **Key Symptoms**
  - ✓ High PIO in query
  - ✓ Long elapsed with low CPU
- **Quick validation**
  - ✓ monProcessWaits
  - ✓ monOpenObjectActivity
- **Root cause determination**
  - ✓ monSysStatement
  - ✓ monCachedStatement
  - ✓ QP Metrics
  - ✓ monCachedObject
  - ✓ monDeviceIO

POSSIBLE CAUSE: BLOCKING

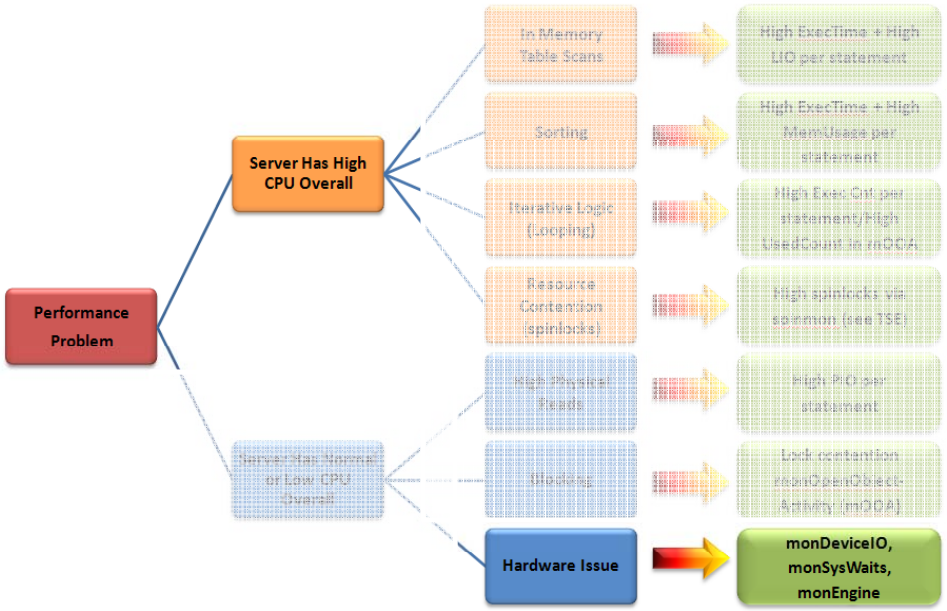
Application Design Problems….Or Failure to Use Datarows Locking



- **Key Symptoms**
  - ✓ High elapsed time with low CPU
  - ✓ Low LIO/PIO
- **Quick validation**
  - ✓ monProcessWaits
  - ✓ monOpenObjectActivity
- **Root cause determination**
  - ✓ monOpenObjectActivity
  - ✓ monLocks
  - ✓ monOpenDatabaseActivity
    - Log Semaphore waits

POSSIBLE CAUSE: HARDWARE ISSUE

The "It Isn't Our Problem Category"



- **Key Symptoms**
  - ✓ High elapsed time with any PIO level
  - ✓ High elapsed but low LIO,PIO and exec time
  - ✓ Low CPUTime/IOTime in monEngine vs. Elapsed time
- **Quick validation**
  - ✓ monProcessWaits
  - ✓ monEngine
- **Root cause determination**
  - ✓ monDeviceIO
  - ✓ monSysWaits/monProcessWaits
    - Aggregate network waits
  - ✓ sp_sysmon

RESOURCES & KPM

KEY MDA STATISTICS FOR OVERALL SERVER PERFORMANCE

RESOURCES: HW MEETS ASE

- **Traditionally, we think of HW resources as:**
  - ✓ Typical HW "kickable" things such as
    - CPU, Disk, Network, Memory
  - ✓ But there is an oft forgotten layer of OS abstraction on these resources
    - CPU timeslices, Disk IO Queues/schedulers, paging/swap
- **Normally, these are used as**
  - ✓ First layer of problem detection
    - i.e. high CPU usage compared to norms (baseline).
  - ✓ First layer of fault isolation
    - i.e. if users complain of slow response time
    - a process is consuming 100% of the CPU, we likely consider it the cause
    - but if it is only consuming 5% of the CPU, we assume it is not involved.
- **ASE as a multi-user server process, provides another layer**
  - ✓ Therefore we should have these as part of our application monitoring
  - ✓ However, ASE only sees resources, data objects, processes and command batches
    - The last three are application oriented.

ASE RESOURCES (NON MEMORY)

How the MDA Tables Map to OS Resources

- **CPU**
  - ✓ Server-wide utilization and scheduling
    - monEngine, monSysWaits, monWorkload (15.5+) (all mostly ignored today)
  - ✓ Process detail utilization and scheduling
    - monProcessActivity, monProcessWaits
- **Disk IO**
  - ✓ Server-wide → monDeviceIO, monIOQueue (ignored today)
  - ✓ Process detail → monProcessActivity
    - There is an issue with this
      - Remember, that most data and index page modifications will be cached
      - Phys IO's include log writes, physical reads, a few physical writes
  - ✓ We will focus today on process detail here only
- **Network**
  - ✓ Server-wide → monNetworkIO; Process → monProcessNetIO
  - ✓ Usually not something watched as typically is not a problem
    - Partial cartesians and other bad queries that return a lot of data will have other more obvious symptoms
    - Normally, huge network consumption is due to bulk processing - which you can't change much
    - If there are networking issues, they will show up in WaitEvents more easily
  - ✓ We will ignore this today as often not an issue for applications

ASE RESOURCES (MEMORY)

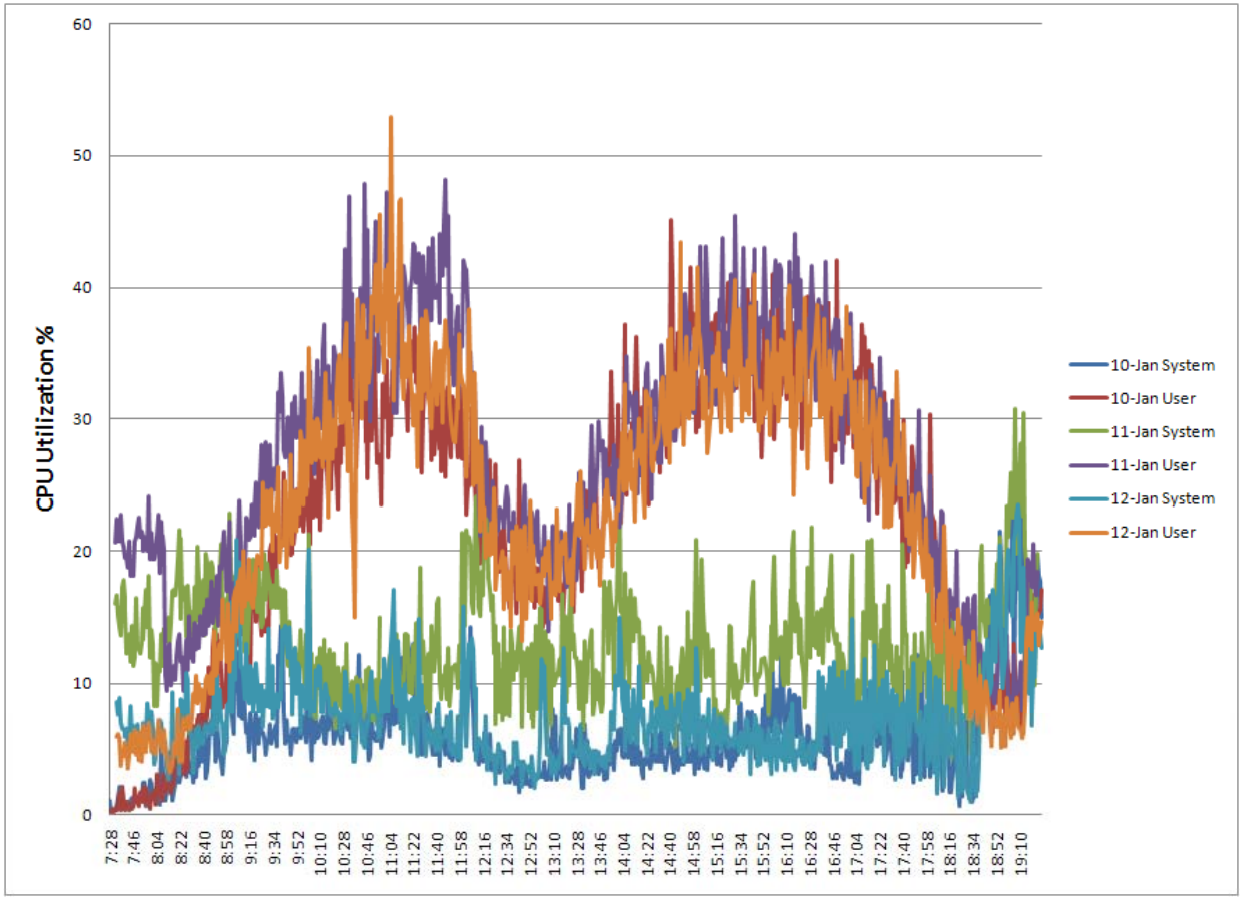
Analogous to Physical and Virtual Memory

- **Swap = tempdb**
  - ✓ Process detail → monProcessActivity
- **Physical Memory = caches**
  - ✓ Data caches
    - Pure object/data structure related caches
    - monDataCache, monCachePool, monCachedObject
    - All are important for application monitoring
  - ✓ Procedure cache
    - Shared procedure execution plans
    - Cached optimization statistics
    - Cached subquery results, sorting structures, etc.
    - monCachedProcedures, monProcedureCacheModuleUsage

WHY BASELINING MAKES SENSE

Bank

User vs. System CPU



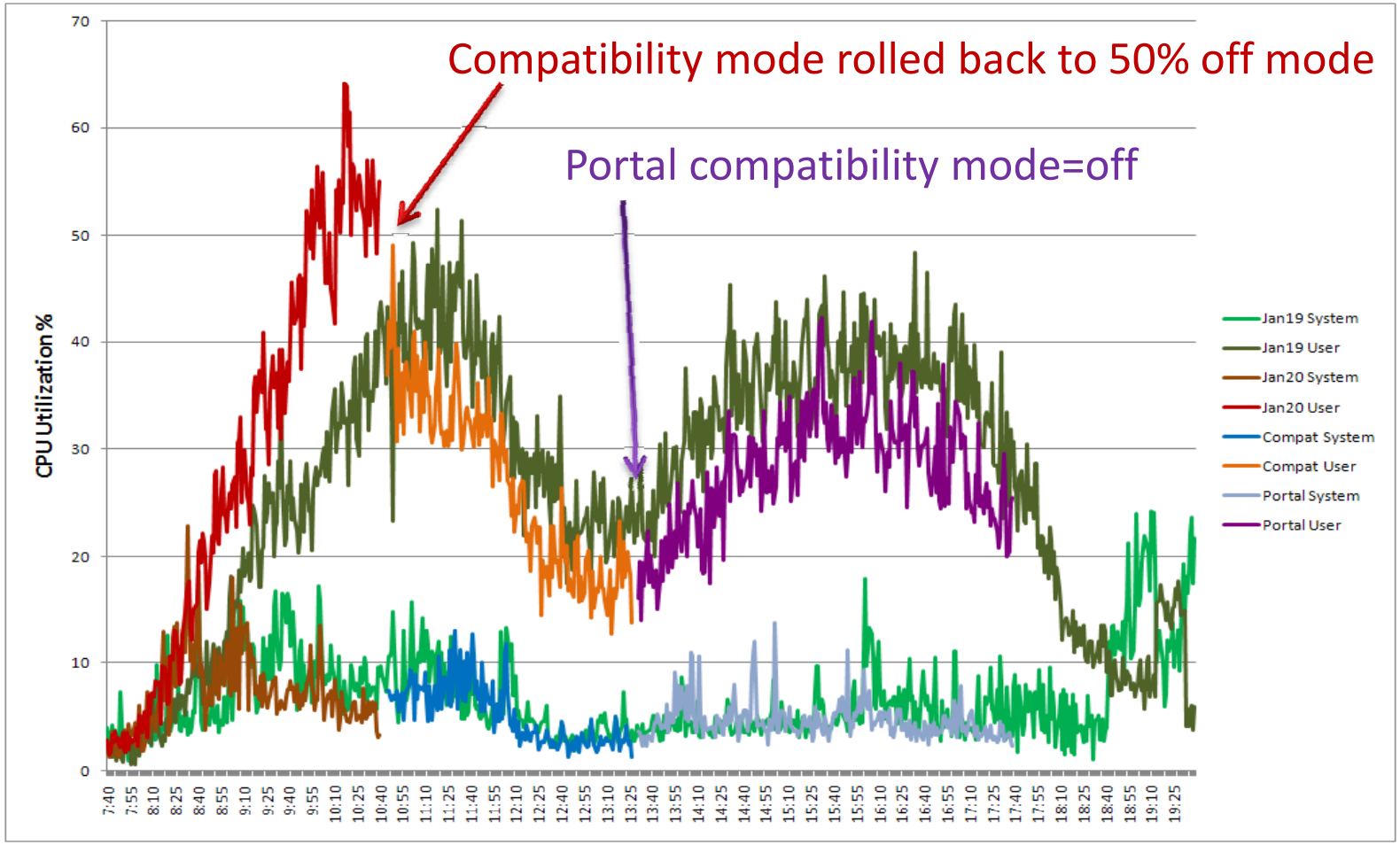
Note that differences in System CPU likely due to Physical IO's - early in the morning due to batch jobs running late - throughout day (e.g. 11 Jan) likely a cause for investigation

SYBASE | An SAP Company

…BUT DON'T JUMP TO CONCLUSIONS

Bank

Customer Claim That ASE 15 Optimizer Causing Issues…..



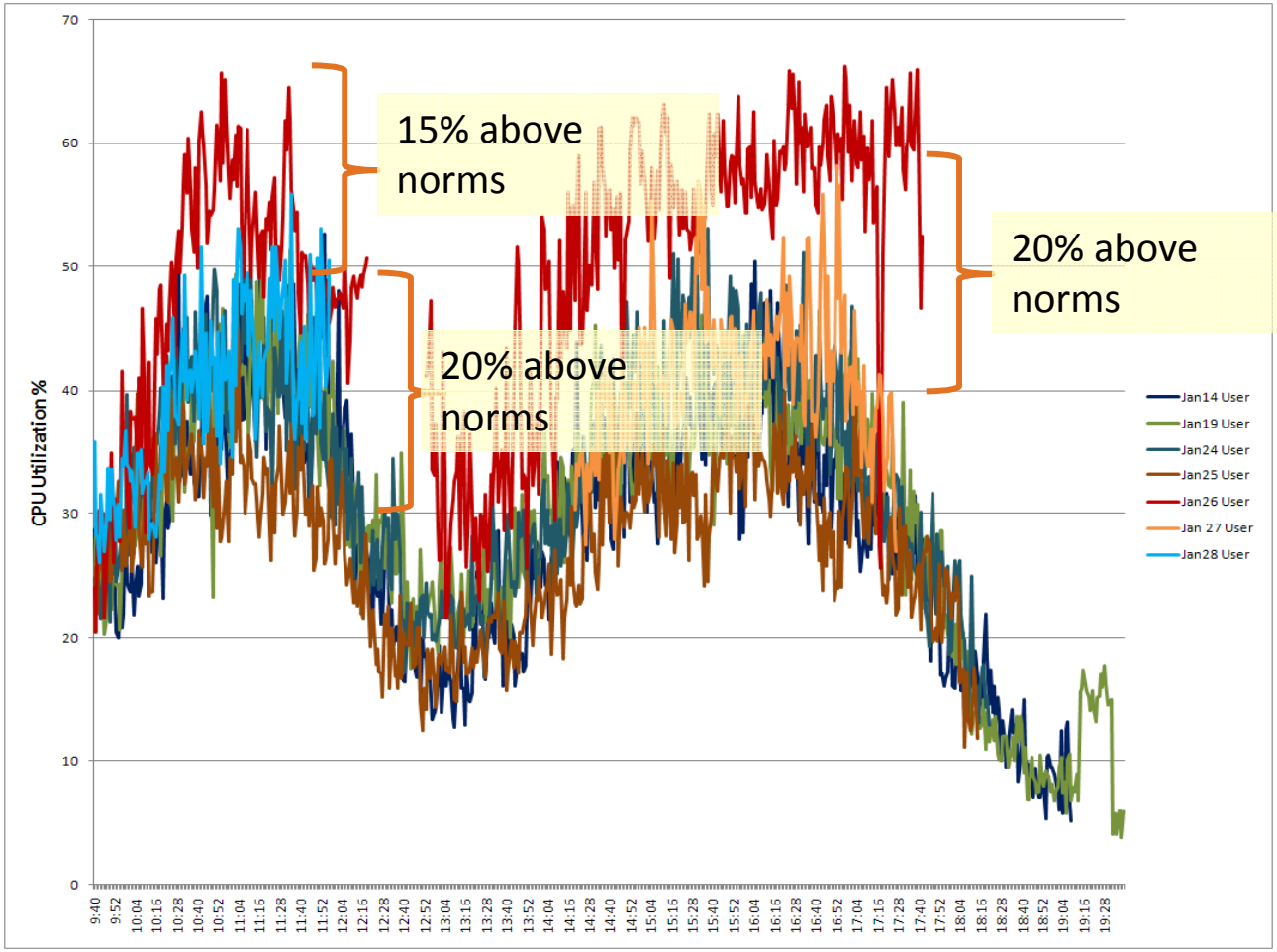
…turned out to be a difference in application workload (coincidence)

SYBASE | An SAP Company

IS IT A PROBLEM???

Seems to be…..

Bank



SYBASE | An SAP Company

BASELINE VS. SYSTEM MONITORING

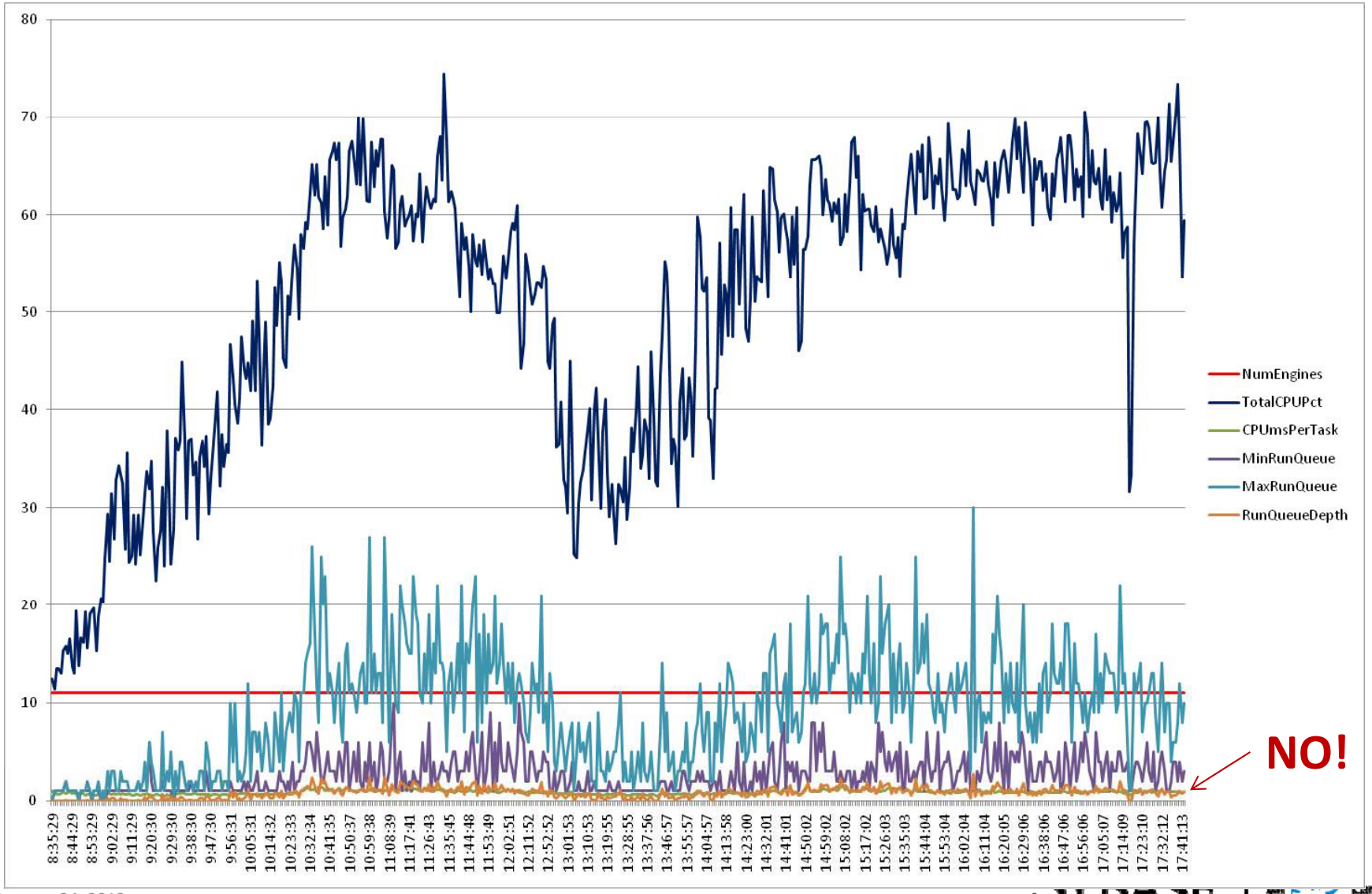
Why *REAL* Application Monitoring is Important….

- **Customer claimed issue on 20th , but not 26th**
  - ✓ Rationale was based on system monitoring in middle and DBMS tiers
    - Middle tier - tracking the depth of the application request queues (for DBMS connections in connection pool) and DBMS response times
    - DBMS tier - tracked blocking contention
  - ✓ Customer's assertion was that turning off compatibility mode changed query optimization and resulted in increased CPU utilization and contention.
  - ✓ Inexplicably for them, the 26th, had no blocking or issues
- **Deeper investigation proved otherwise**
  - ✓ In both cases, there were significant application workload differences
    - You will see this in some of the examples coming up
  - ✓ From the database tier, we can NOT tell what the difference was in a business sense
    - ….and depending on the gaps or polling frequency, the exact difference may be hard to spot except at cumulative statistics

DOES MY APP NEED MORE ENGINES??

Bank

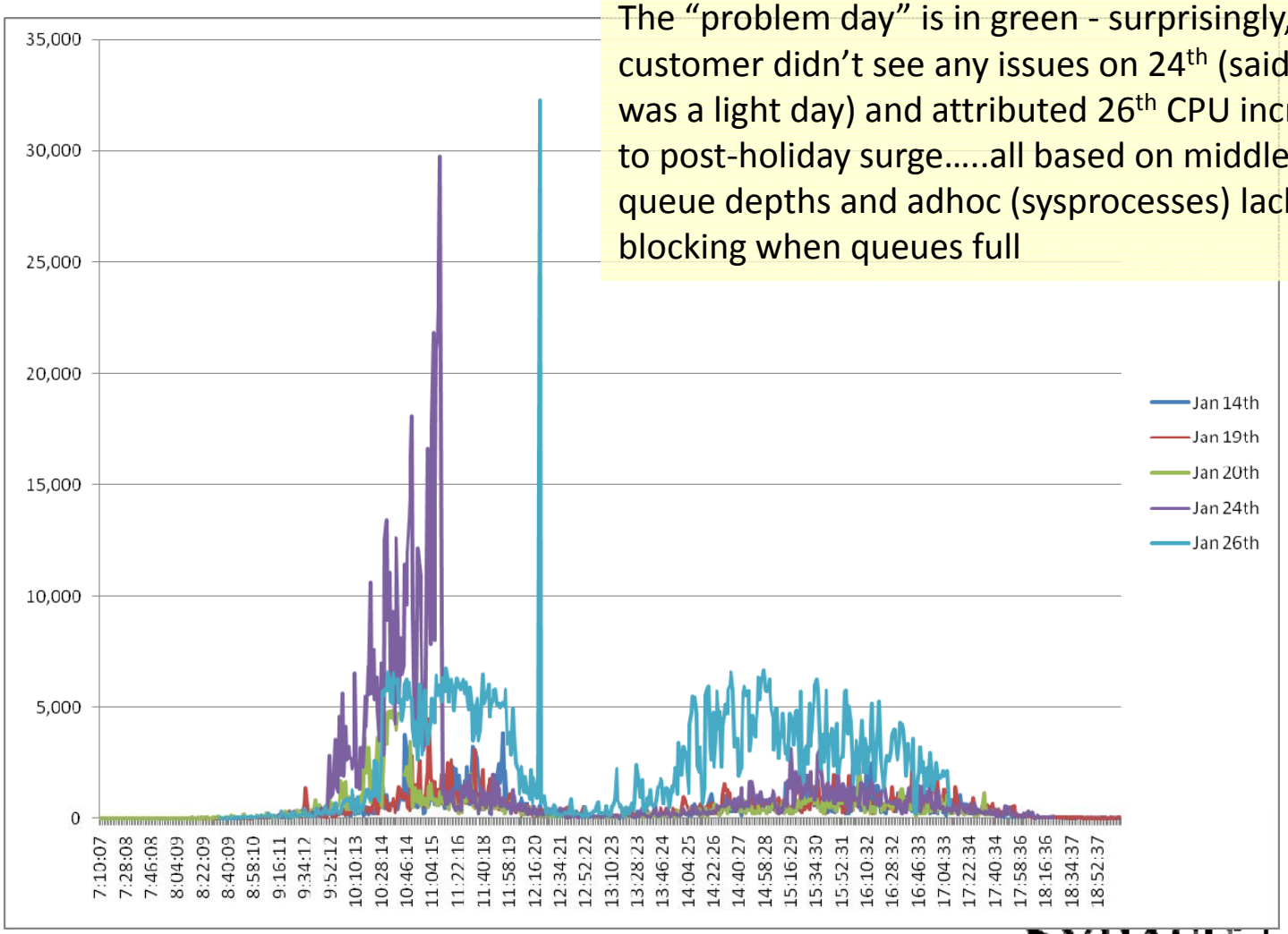
CPU Usage & Run Queue vs. Num Engines & Run Queue Depth



MONITORING CONTENTION

monSysWaits - WaitEventID 150

Bank



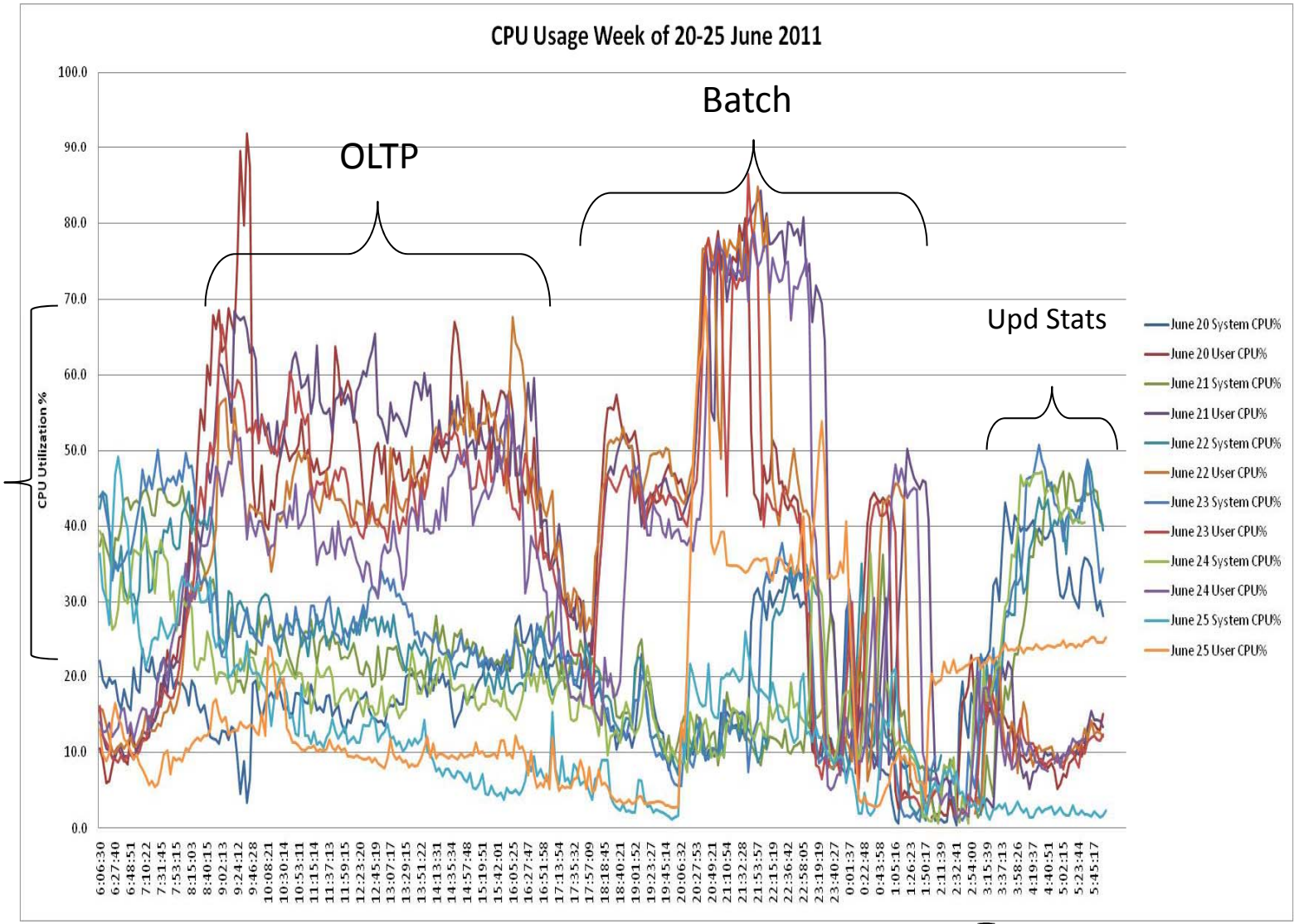
The "problem day" is in green - surprisingly, customer didn't see any issues on 24th (said it was a light day) and attributed 26th CPU increase to post-holiday surge…..all based on middle-tier queue depths and adhoc (sysprocesses) lack of blocking when queues full

SYBASE | An SAP Company

QUIZ #1: WHAT CAN WE SAY

Health

High CPU and high PIO suggests query issues (in fact, batch pegs to 100% for ~4 hours)



SYBASE | An SAP Company

A FINAL COMMENT ON CPU UTILIZATION

A Dose of Cold Reality

- **High CPU is *likely* NOT the cause of application performance**
  - ✓ It is a symptom
  - ✓ It also is most likely an indicator of query issues such as missing indexes
- **If you have high CPU, adding CPU's/engines won't help**
  - ✓ …unless it is run queue starvation (runnable processes/# of engines > 5)
  - ✓ It will probably make things worse
- **While high CPU may be driven by high LIO on some tables…**
  - ✓ …there may be other tables with high PhysicalReads (PagesRead)
  - ✓ Fixing the PhysicalReads (PagesRead) will have the bigger payback
    - PhysicalReads become LogicalReads anyhow, so it is a net decrease in both
    - Better indexing/query execution reduces cache over saturation due to table scans or other PhysicalRead side effects
    - An example of this later (Claims_Detail_Tbl slide)

APPLICATIONS & USERS

FINDING WHO CAUSED IT

MONPROCESS & MONPROCESSACTIVITY

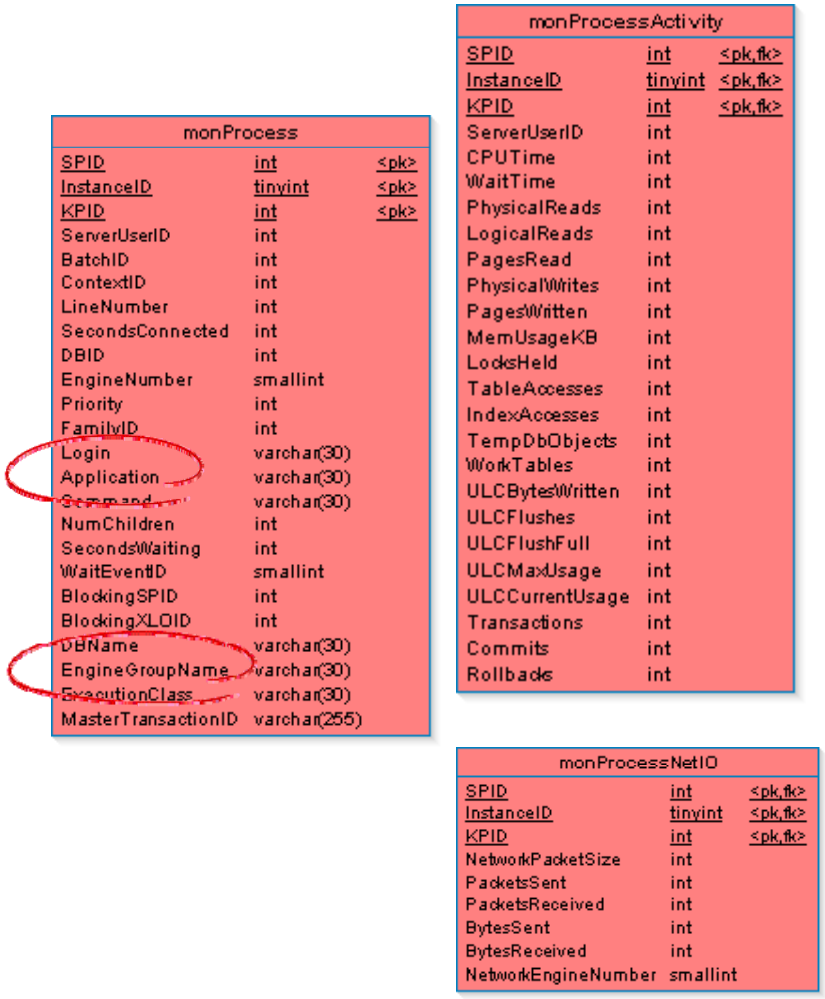
The Key to Application Monitoring

CAREFUL with Aggregation !!!!

- Both only track currently active SPID's
- If someone logs out, their data is not visible in the next sample
- If someone logs in, they may reuse the same SPID…so SPID+KPID are keys
  - Plus InstanceID in ASE/CE

Best Method for Aggregating

- Do the delta per time sample per SPID+KPID in monProcessActivity
- Join with monProcess to get Login, Application, Command, EngineGroup & ExecutionClass for that time sample
  - Command is transient except for system processes such as Checkpoint - need to figure out which SPID is Checkpoint vs. HK.
- Then aggregate grouping by Application, by Application + Login or whatever destired
  - Keep in mind that users that disconnected will not be in latest sample each time, consequently some things such as CPU time may not add up ala monEngine



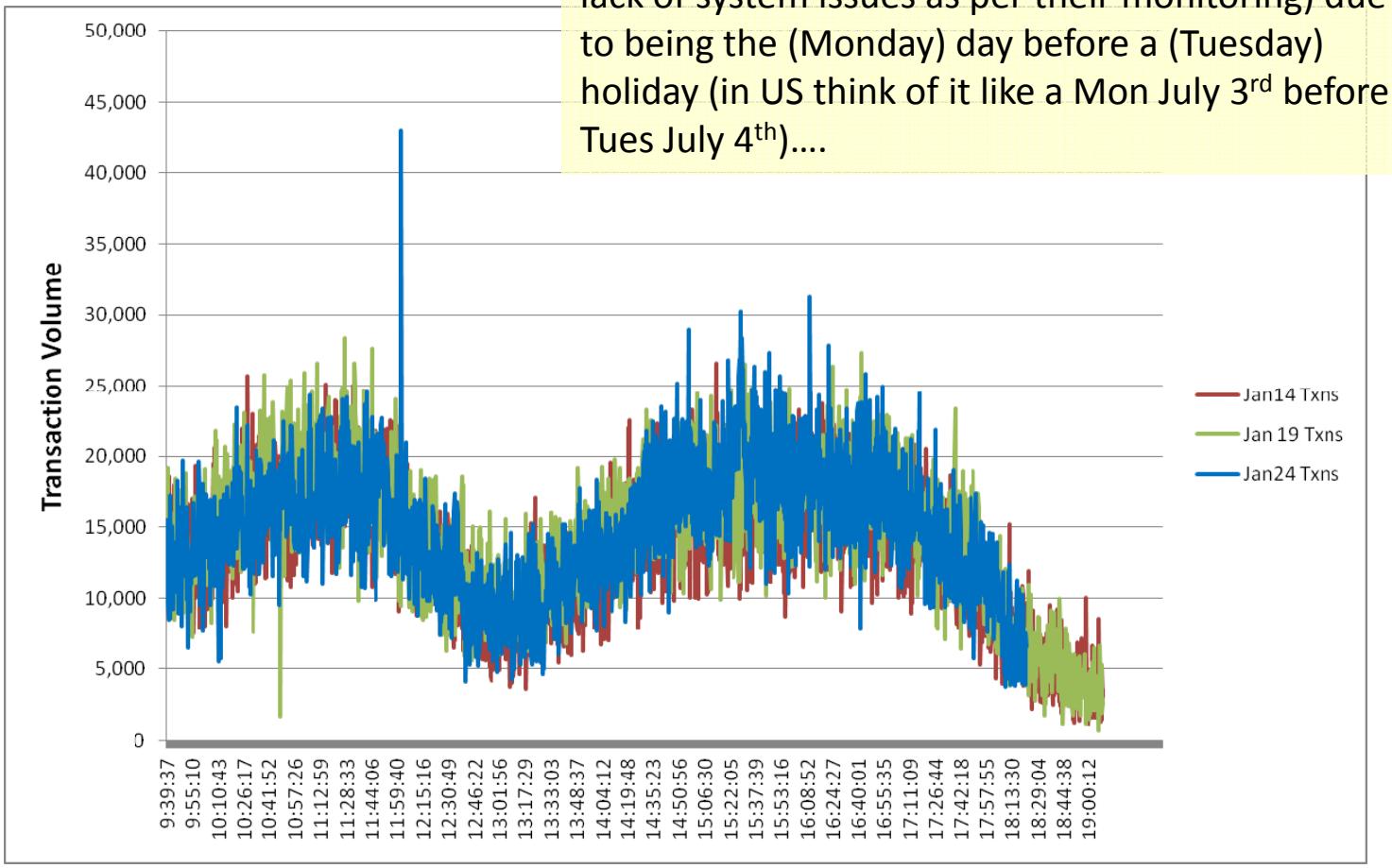
SYBASE | An SAP Company

BASELINING MORE THAN CPU….

…Comparing Transaction Volume over Time

Bank

Customer claimed 24th was a light day (based on lack of system issues as per their monitoring) due to being the (Monday) day before a (Tuesday) holiday (in US think of it like a Mon July 3rd before Tues July 4th)….

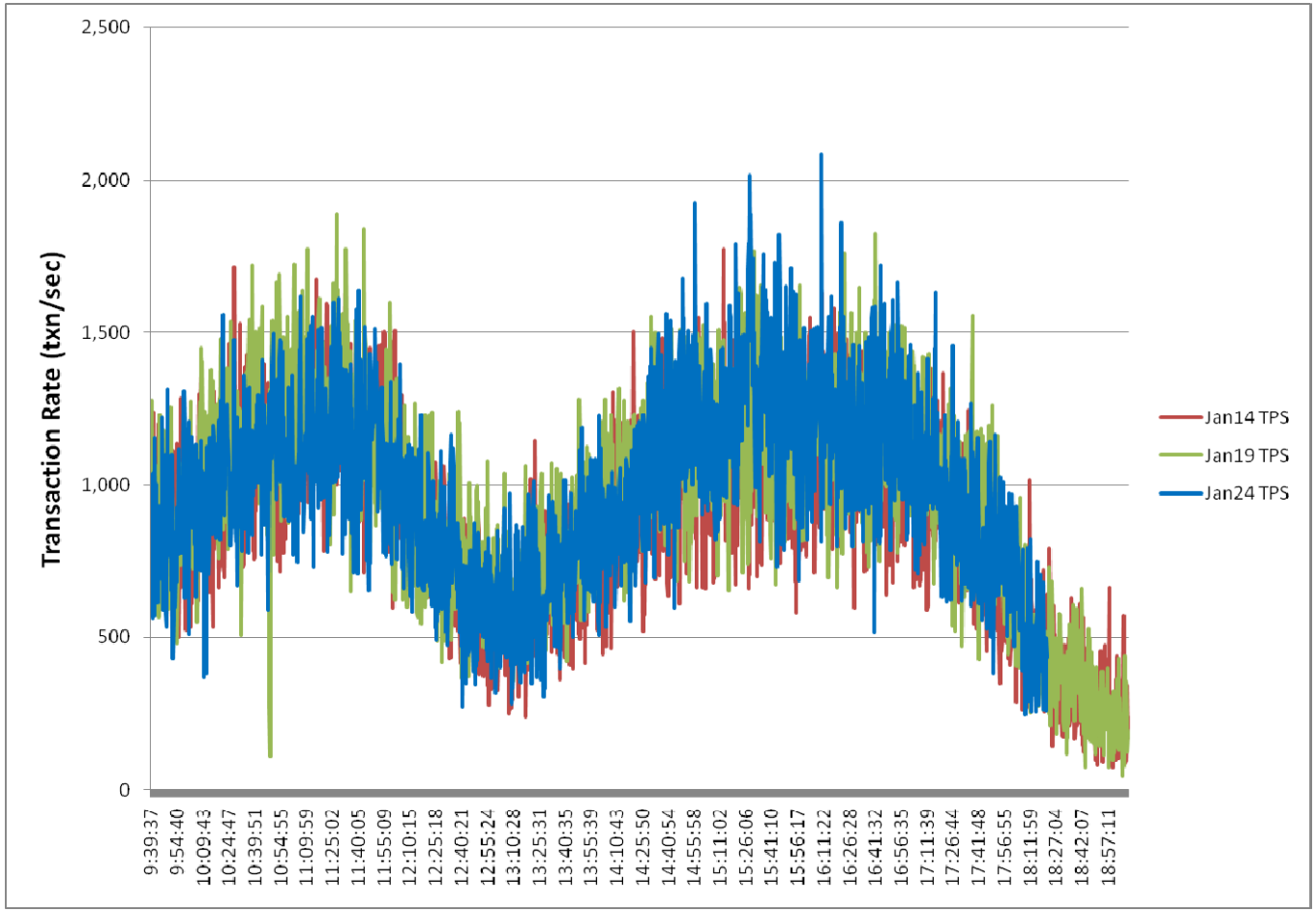


SYBASE | An SAP Company

BASELINING MORE THAN CPU….

…or if you prefer, Transaction Rate over Time

Bank

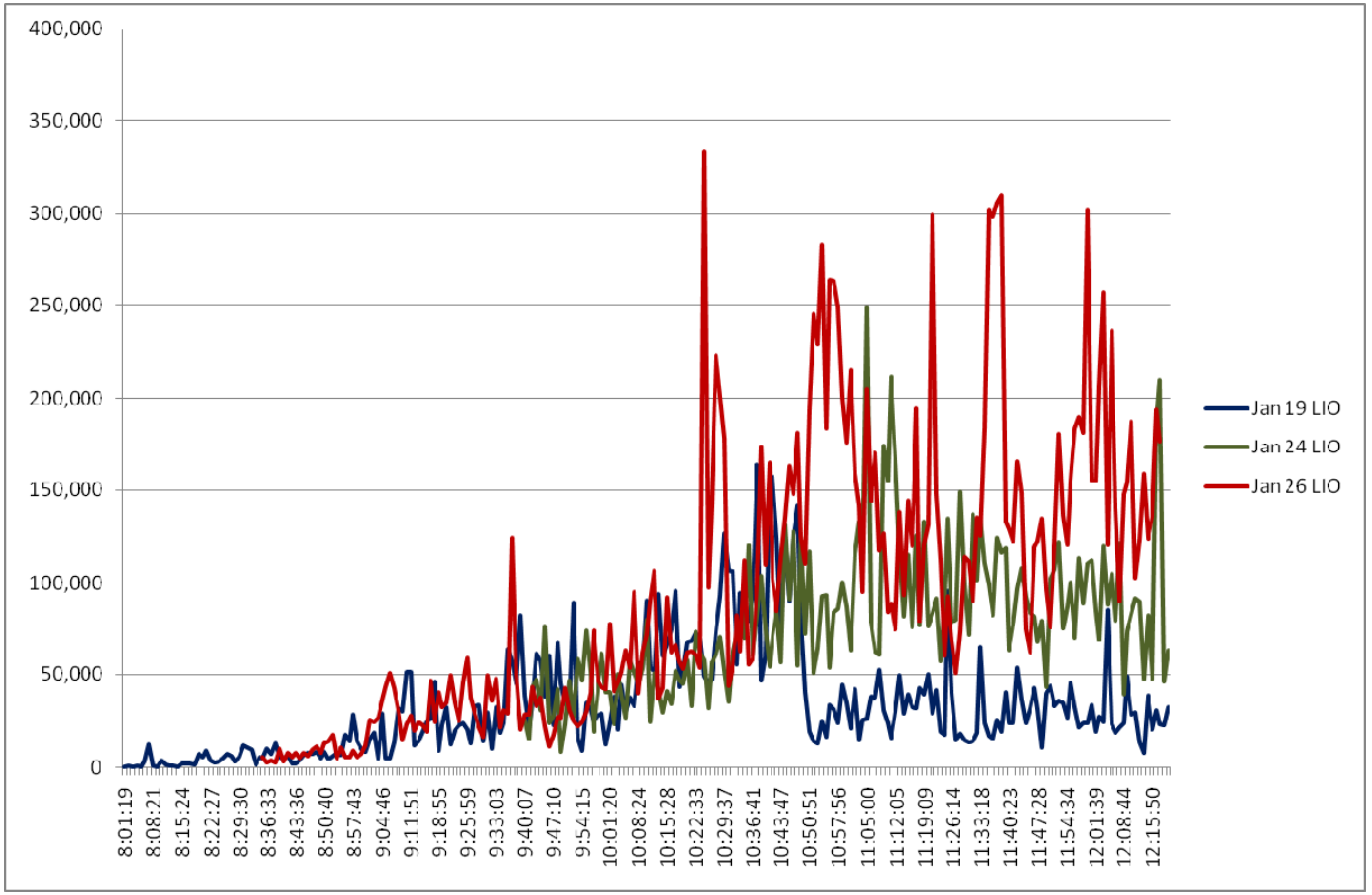


SYBASE | An SAP Company

BASELINING MORE THAN CPU….

Bank

Watching for the Abnormal Increase…

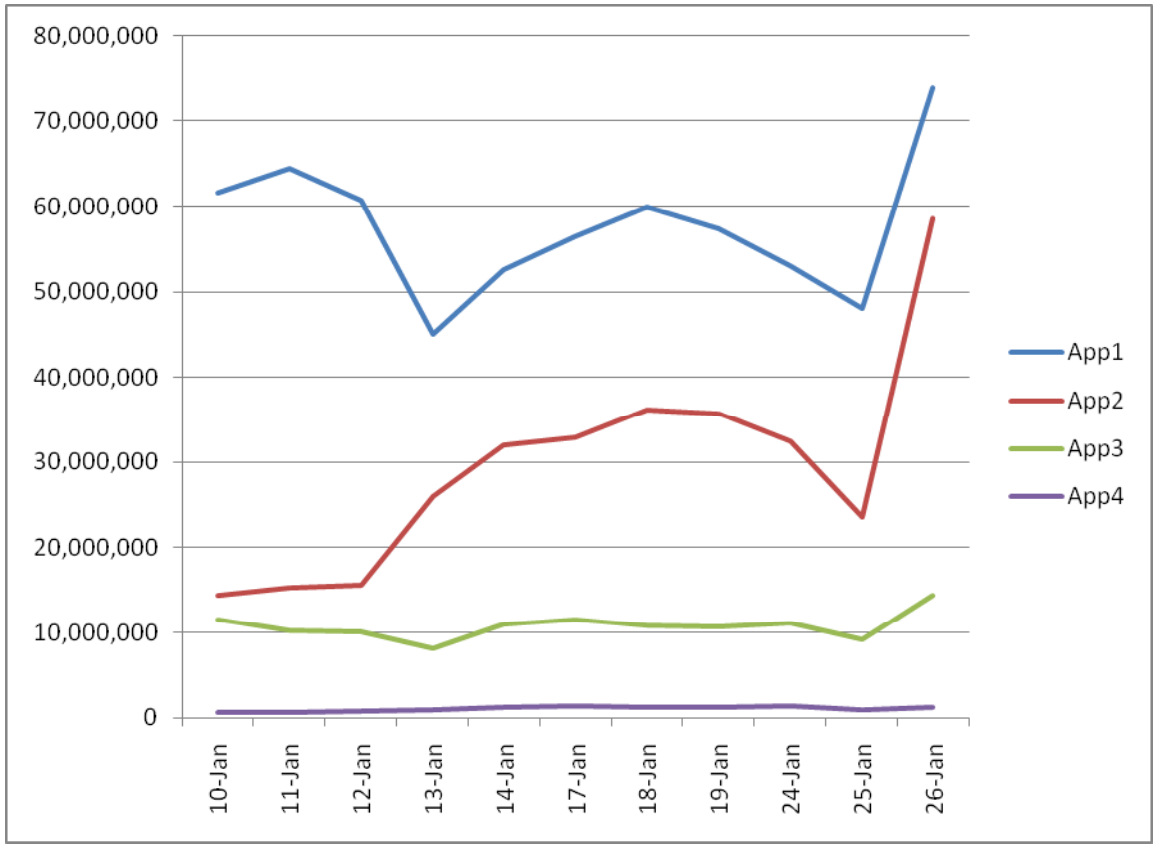


SYBASE | An SAP Company

CPU TIME

Trying to Get More Information on the Jan 26th Issue

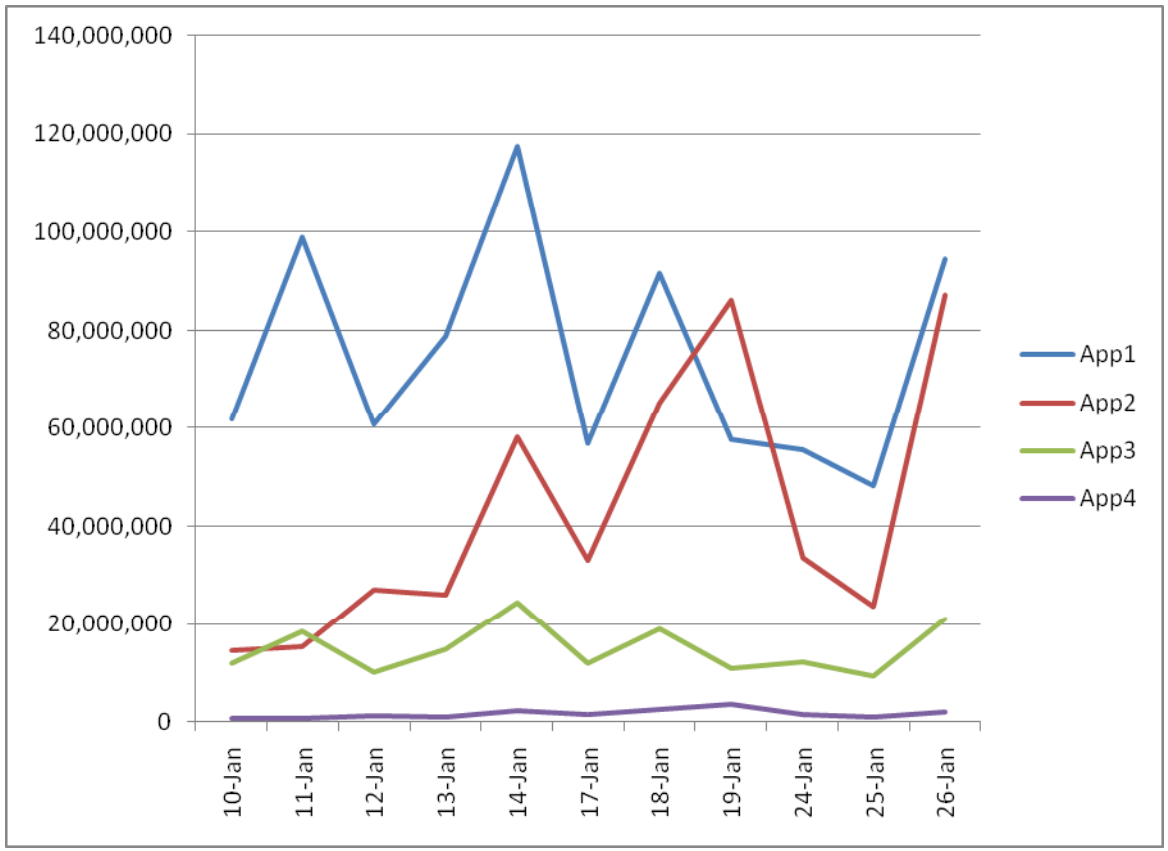
Bank



SYBASE | An SAP Company

TRANSACTIONS

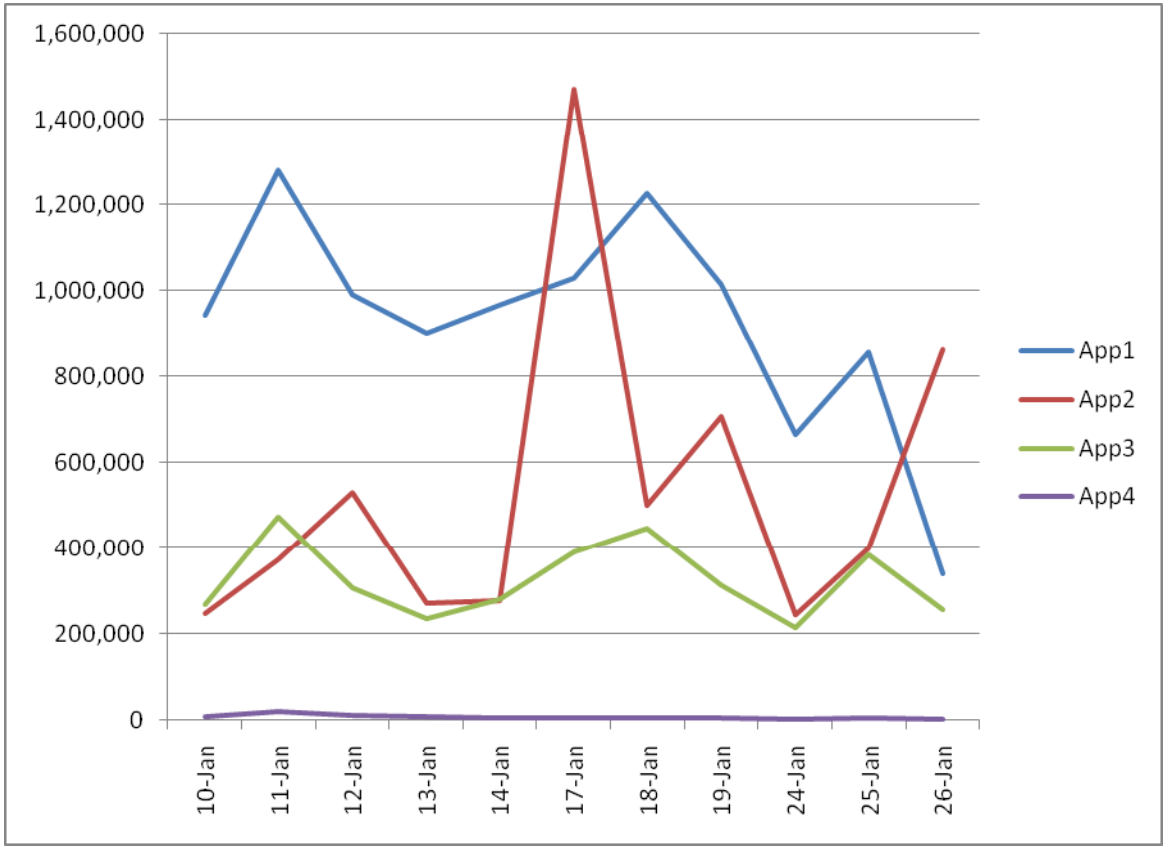
Trying to Get More Information on the Jan 26th Issue



PHYSICAL READS

Trying to Get More Information on the Jan 26th Issue

Bank

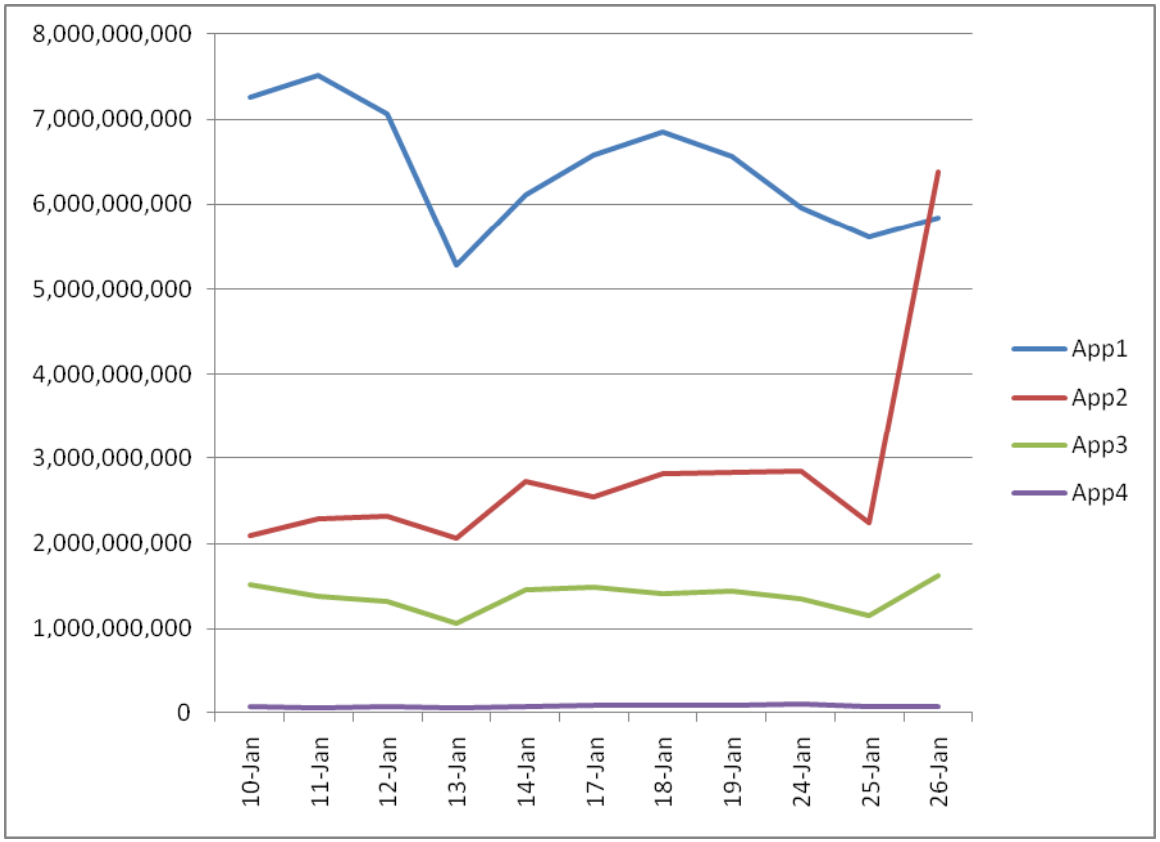


Notice the swap in volume between App1 & App2 - if we had only aggregated to the server level, we never would have spotted this as 26th would have matched 25th ….shows up in tempdb as well…one reason why relying solely on sp_sysmon results in DBA's missing obvious problems

SYBASE | An SAP Company

LOGICAL READS

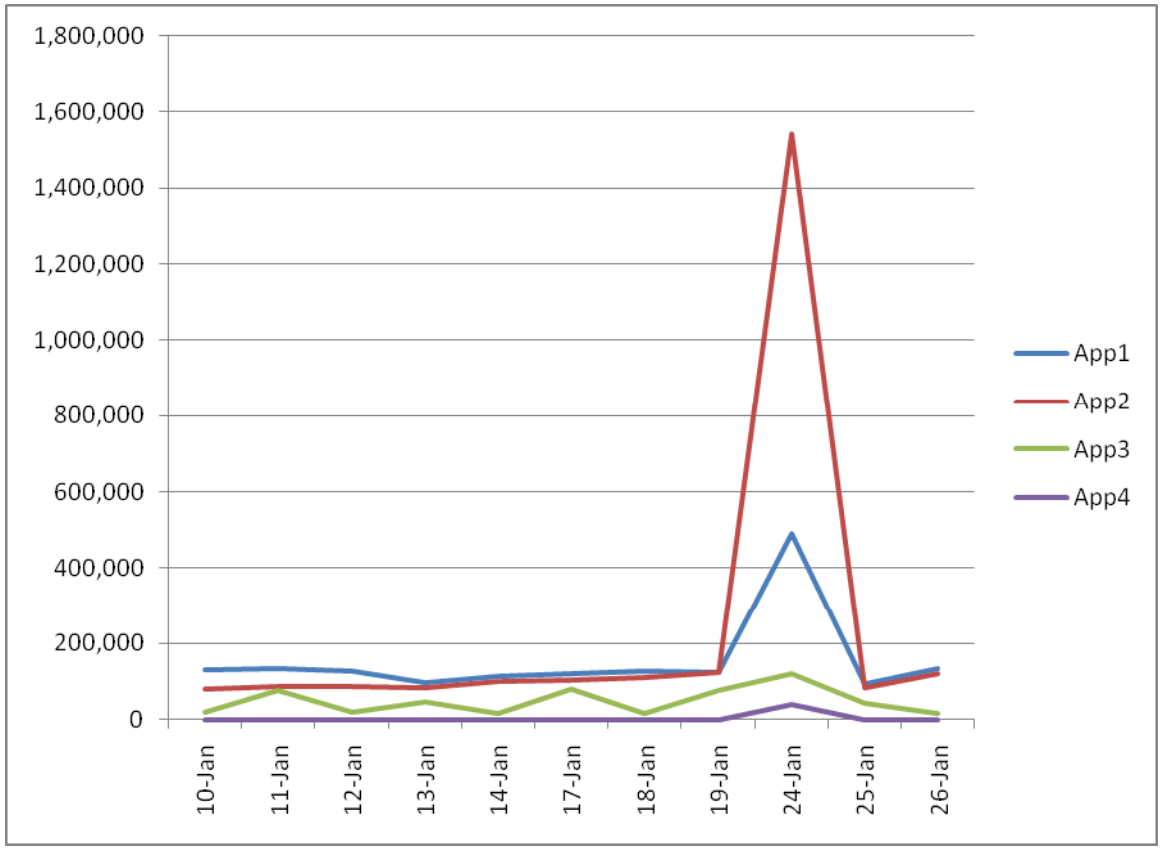
Trying to Get More Information on the Jan 26th Issue



PHYSICAL WRITES

Trying to Get More Information on the Jan 26th Issue

Bank



SYBASE | An SAP Company

TEMPDB TEMP TABLES

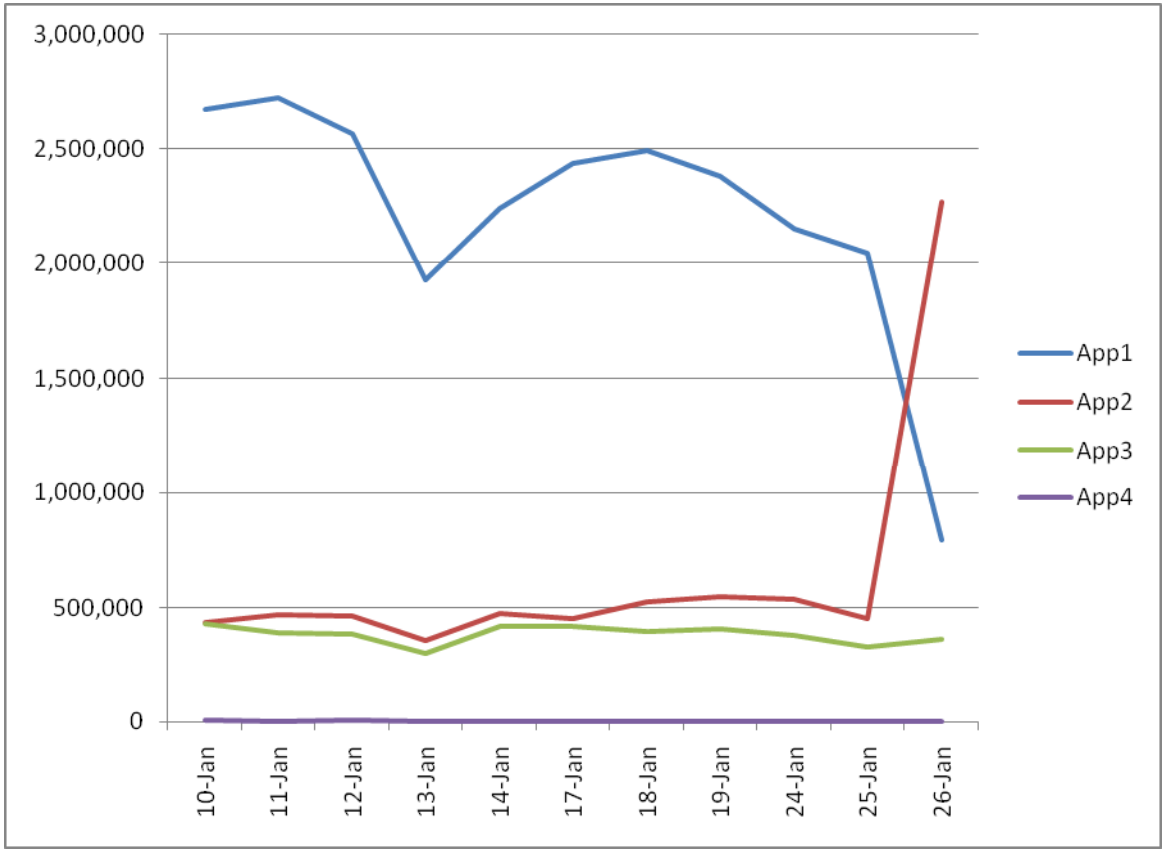
Trying to Get More Information on the Jan 26th Issue

Bank

SYBASE | An SAP Company

TEMPDB WORK TABLES

Trying to Get More Information on the Jan 26th Issue

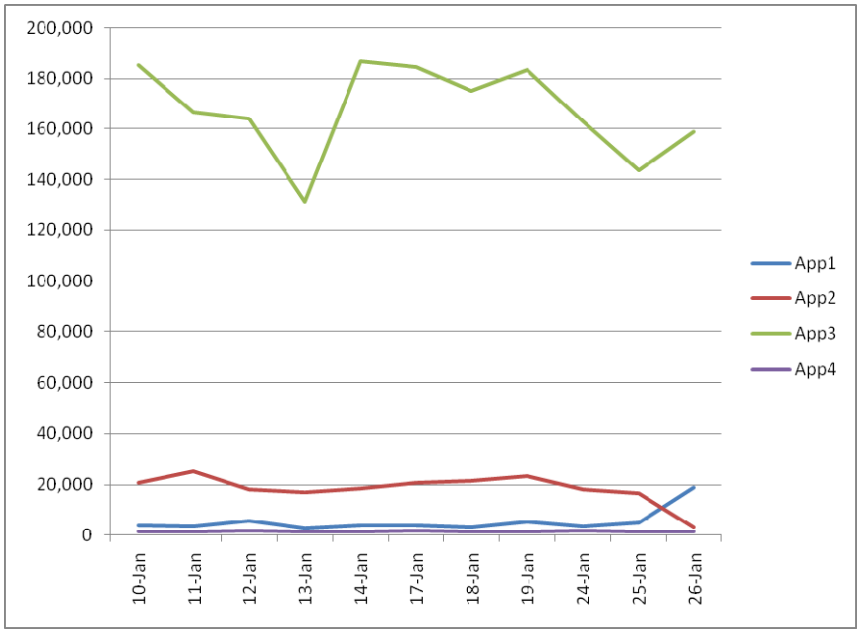


…here's our swap again

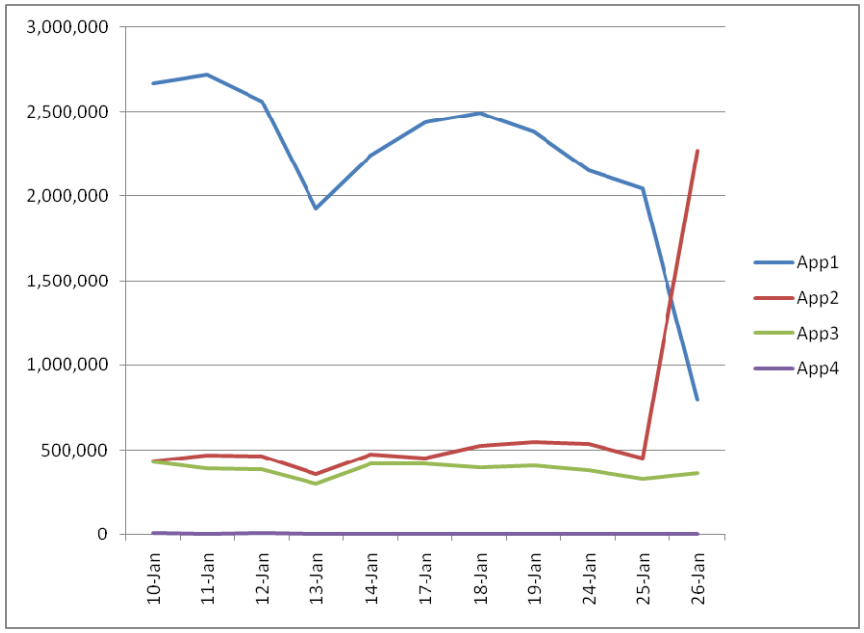
TEMP VS. WORK TABLES

Did You Notice App1 & App3 in Each Case???

Tempdb Objects



Work Tables



SYBASE | An SAP Company

SO, WHAT CAN WE OBSERVE??

Conclusions, Suspicions and Just Plain "Needs More Investigation"…

- **Archive process or batch job runs on Monday's**
  - ✓ Cause of spike in Physical Writes
- **App1 seems to have a lot of PhysicalReads as a norm**
  - ✓ App2 has spikes of Physical Reads
- **App2 seemed show steady increase in workload**
  - ✓ CPU utilization and transaction volume for 14-19 vs. 10-12
  - ✓ Logical reads up about 50% as well (2M → 3M)
  - ✓ But tempdb utilization remained flat
- **App1 has high tempdb usage - but mainly due to work tables**
  - ✓ Join or subquery processing - either in adhoc queries or stored procs
- **App3 has high tempdb usage - but mainly due to temp tables**
  - ✓ Probably due to high stored proc execution
- **Spike on 26th…**
  - ✓ Although both App1 & App2 saw increase in CPU and Transactions, App2 had a significant increase in LogicalReads and in Tempdb work tables
  - ✓ Suspicion is that a lot higher read activity is occuring
    - ….we don't know where yet (e.g. which tables)
    - ….we don't know why yet (e.g. reports, bad queries, proc looping, etc.)

QUIZ #2: TABLE ACCESS VS. INDEX ACCESS

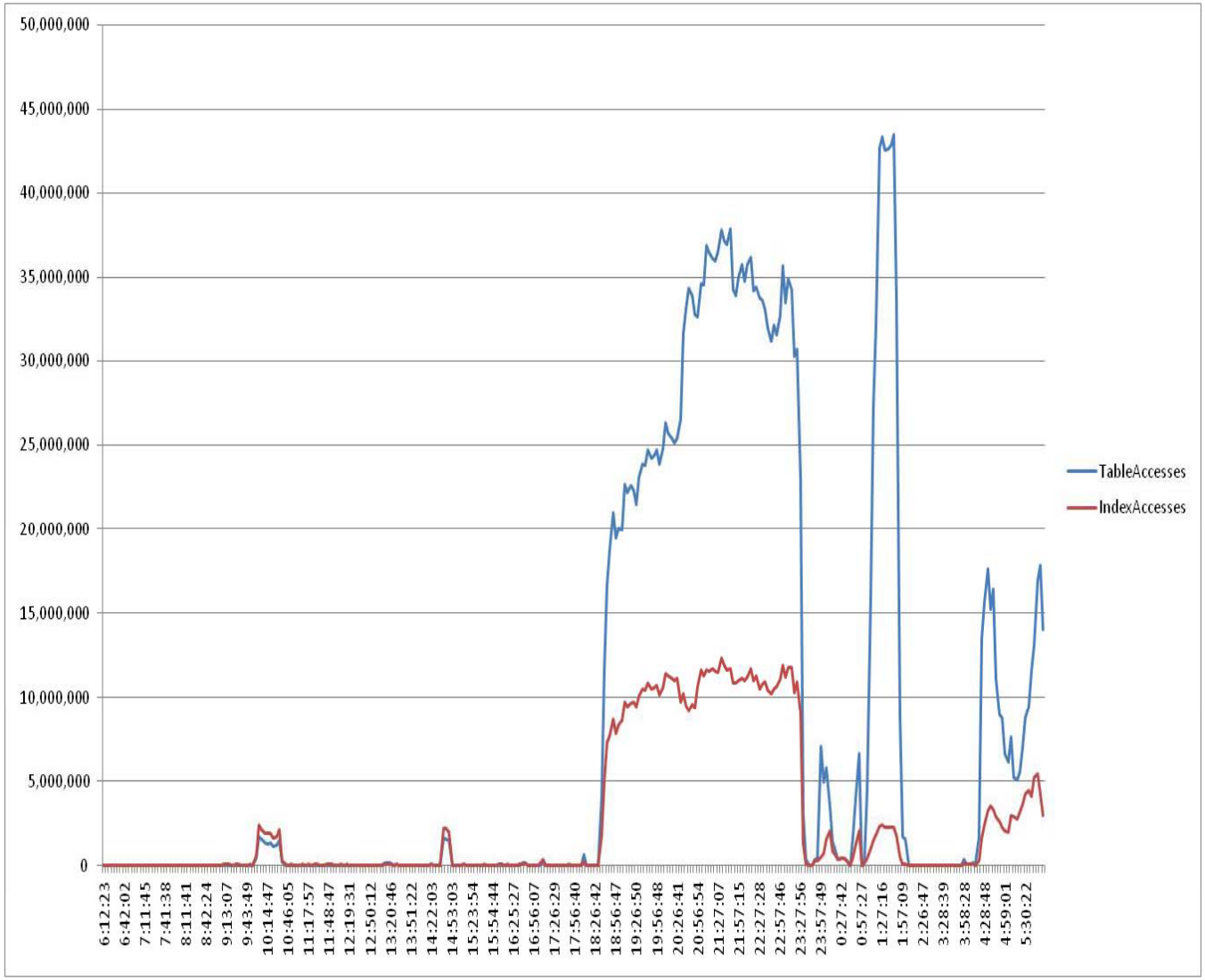
A Simple Test…..

- **Consider a typical join between two tables**
  - ✓ Inner table has 4 levels of indexing
  - ✓ For every outer row, we then hit 4 index pages to find each data page
  - ✓ Exceptions:
    - Min()/max() to find starting point for scan
    - Partial index usage/reformatting
- **Consider the typical DML (e.g. an insert)**
  - ✓ Traverse the pkey index to find the data page (4 Index IO's + 1 Data IO)
  - ✓ Traverse each "unsafe" index tree to modify the index key value and insert the new page # & rid (4 index IO's x number of indices)
- **Some Exceptions**
  - ✓ **Heap inserts (distorts table accesses)**
  - ✓ Index covering (distorts 4:1 ratio)
  - ✓ Range scans (distorts 4:1 ratio)

IS THIS THEN A GOOD THING???

Batch Processing

Health



SYBASE | An SAP Company

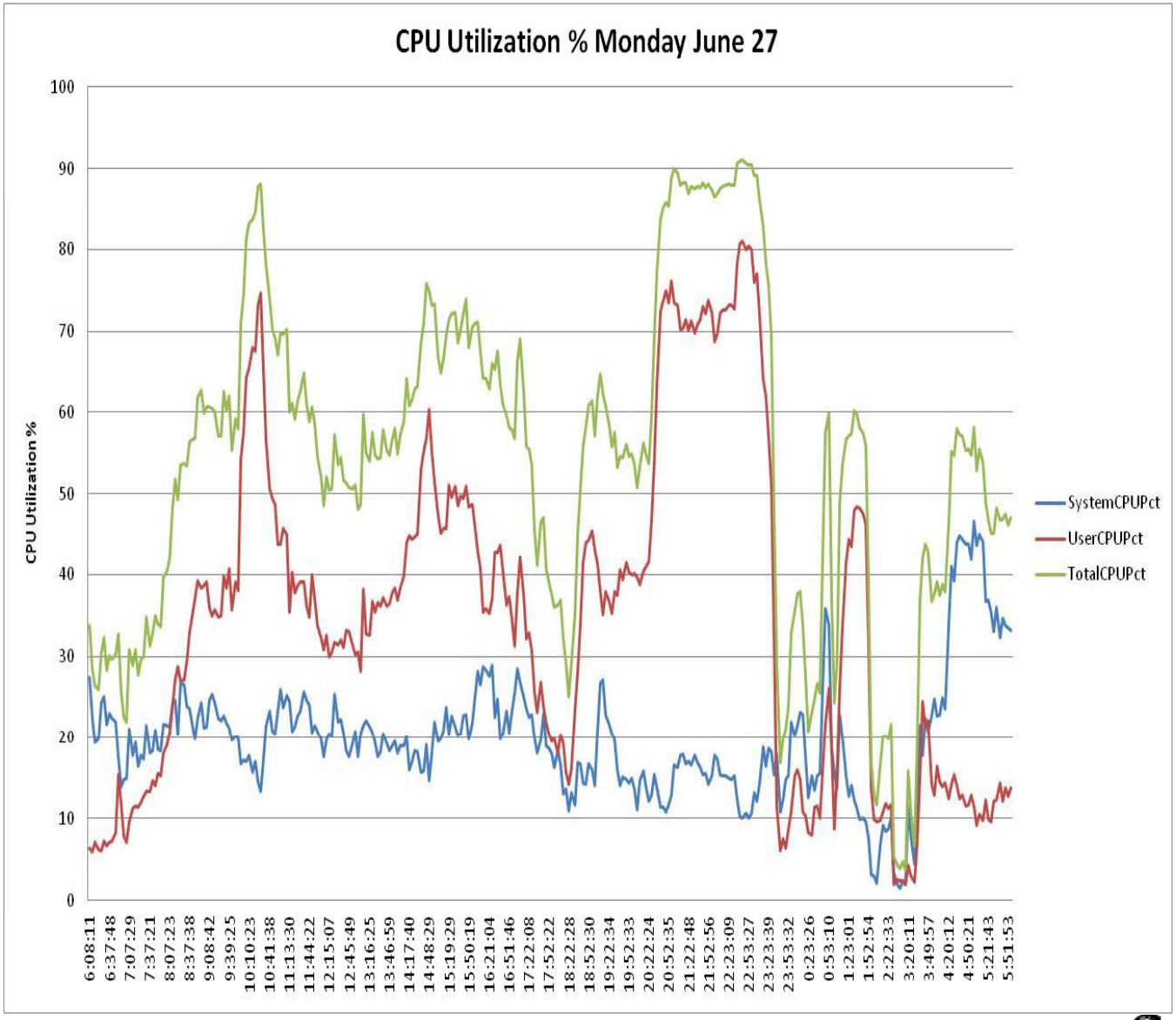
NOW LET'S DIG A BIT DEEPER

Finding the Root Cause

- **Baselining Applications is a good start**
  - ✓ Helps you to identify when something is abnormal
  - ✓ Provides an application on which to focus more detailed investigation
- **The next step is to drill down**
  - ✓ Is it a particular login (often shared among 100's of connections)?
  - ✓ Is it a single user or just a complete swamp of the system?

LET'S TAKE A TYPICAL DAY

Health



SYBASE | An SAP Company

THE TOP CPU USERS BY APPLICATION

Health

Application	Num SPIDs	CPU Time	Physical Reads	Logical Reads	Physical Writes	Table Accesses	Index Accesses	TempDb Objects	Work Tables	Trans-actions	Mbytes Sent
Batch App	1,268	278,176,400	45,186,584	8,348,010,501	8,441,952	3,426,984,517	1,180,546,805	5,293,556	15,553,236	50,718,284	209,193
Online App	6,502	139,461,600	43,882,272	3,862,716,095	3,348,953	1,812,983,476	526,140,478	807,532	11,847,547	7,770,953	257,414
Unnamed App(s)	6,623	60,178,500	5,606,470	2,034,589,868	1,821,049	975,774,437	382,863,391	159,199	3,676,678	4,165,564	76,841
Main Process 1	1,964	56,151,200	8,172,356	3,336,723,825	660,530	1,410,413,824	506,244,144	2,664,453	11,761,582	51,427,521	6,485
Common App	32	28,104,300	13,025,015	3,202,384,924	5,524,927	911,248,948	415,837,761	14	767,365	3,134,841	2,116
isql	167	13,010,700	28,129,609	1,799,195,226	49,563	107,542,966	17,738,625	39	683	762	0
SQR	141	9,998,000	387,459	672,490,978	12,556	407,435,270	46,013,204	0	398,501	13,184	276
HK CHORES	1	4,356,600	1,297	8,495,066	1,649,314	141,303	135,532	0	0	0	0
OmniServer	57	4,095,300	111,770	62,237,808	89,544	15,010,154	15,743,297	0	969	8,333	2,950
Someother App	4	2,797,400	17,052	7,582,848	24	5,307,483	1,070,191	487	19	1,987	2
CHECKPOINT SLEEP	1	1,890,700	264	2,392,322	30,869,299	126,472	79,048	0	0	34,261	0
Another App	1,607	1,739,800	2,993	67,070,648	4	50,224,444	7,829,391	0	0	0	97
HK WASH	1	1,561,400	0	14,349	19,826,080	0	0	0	0	0	0

Unless you have a lot of time, focus on the heavy hitters (top 5 or 10)…

The question will always be whether the resource consumption is just unfortunate due to the number of users or not. A good clue in this case is the LIO, TableAccesses, etc.

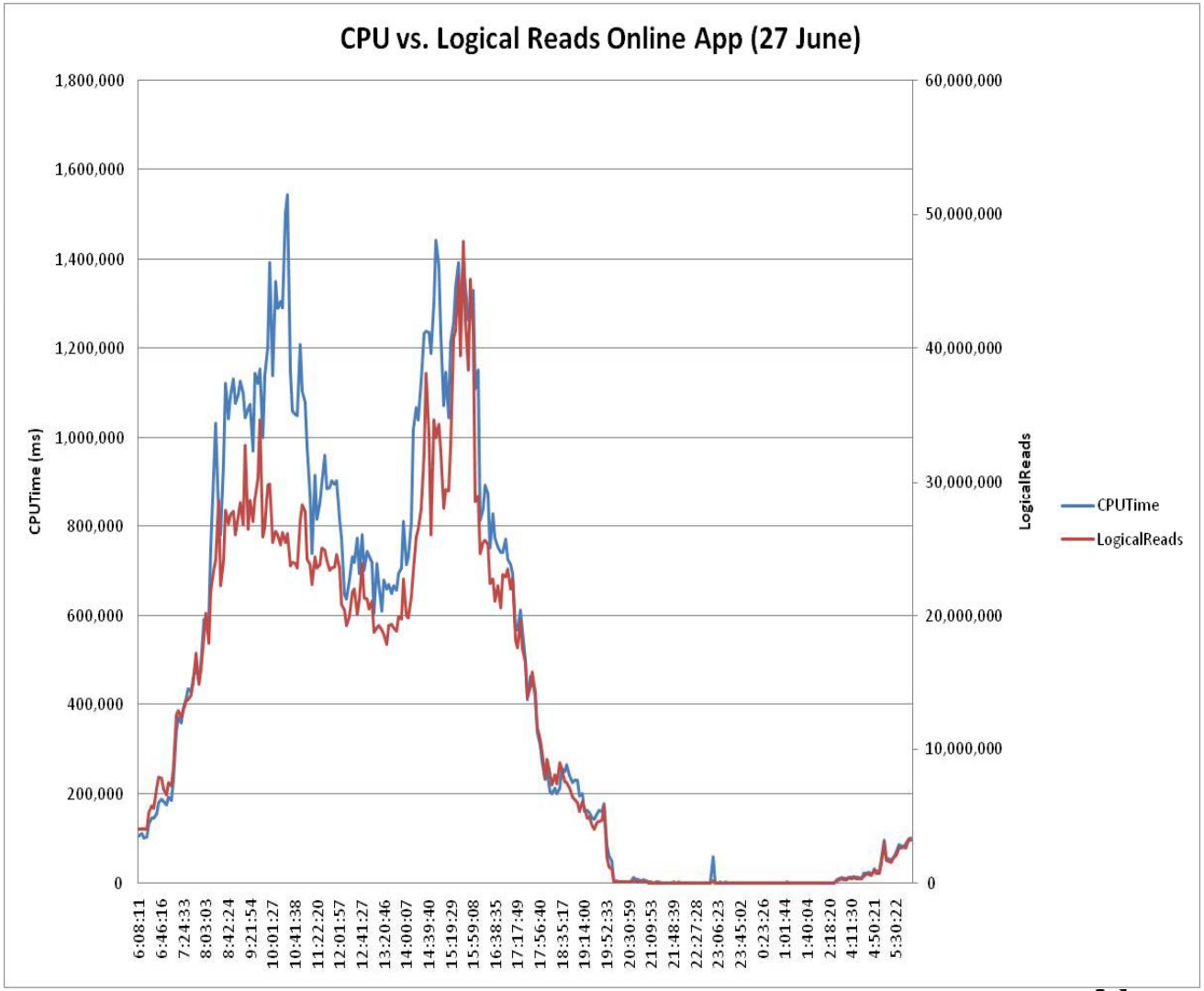
PAUSE FOR RATIONAL THOUGHT

Remember The Lesson From Earlier??

- **What drives CPU???**
  - ✓ LogicalReads (scanning and comparing data values in memory)
    - Bad queries, missing indexes, tablescans, etc.
    - Joins (e.g. a lot of joined rows or Cartesian products)
  - ✓ System Requirements (processing PhysicalReads, Network)
  - ✓ Application Logic (e.g. tight loops)
  - ✓ System Contention (blocking, spinlock contention, etc.)
- **We already know**
  - ✓ #2 isn't the cause as User CPU is much higher than system CPU
  - ✓ #3 & #4 are not as likely as #1
    - If looping, the number of txns might be astronomically higher (later)
    - If contention - e.g. blocking, the user cpu per SPID would be lower as it is sleeping while awaiting the lock
    - But we can't completely rule it out yet….

ONLINE FIRST…AN ALL TOO OFTEN PICTURE

Health



SYBASE | An SAP Company

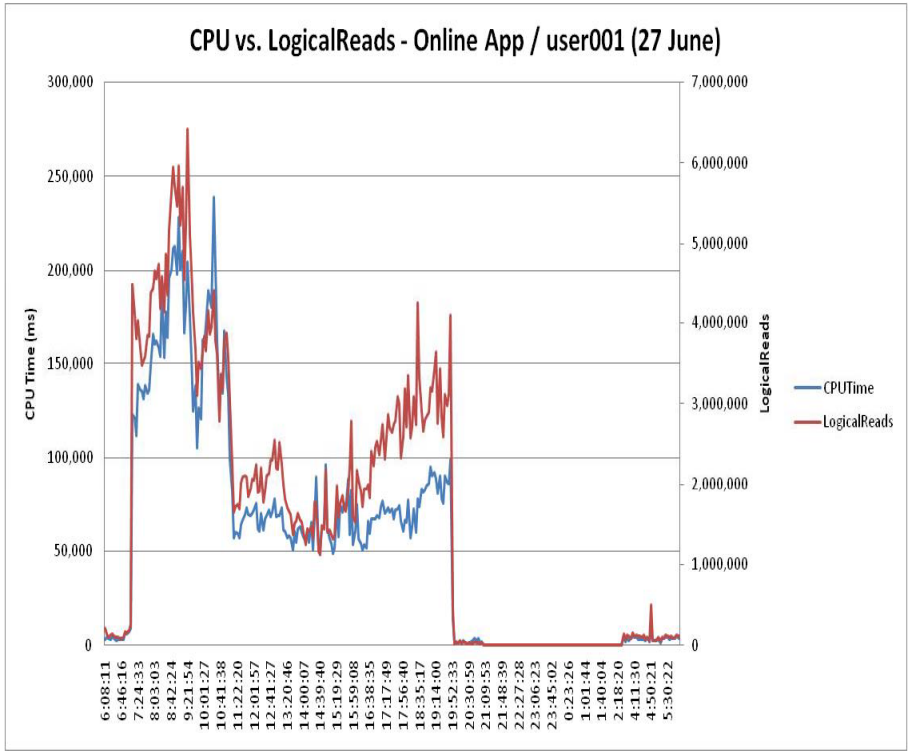
TOP ONLINE LOGINS & CPU USAGE

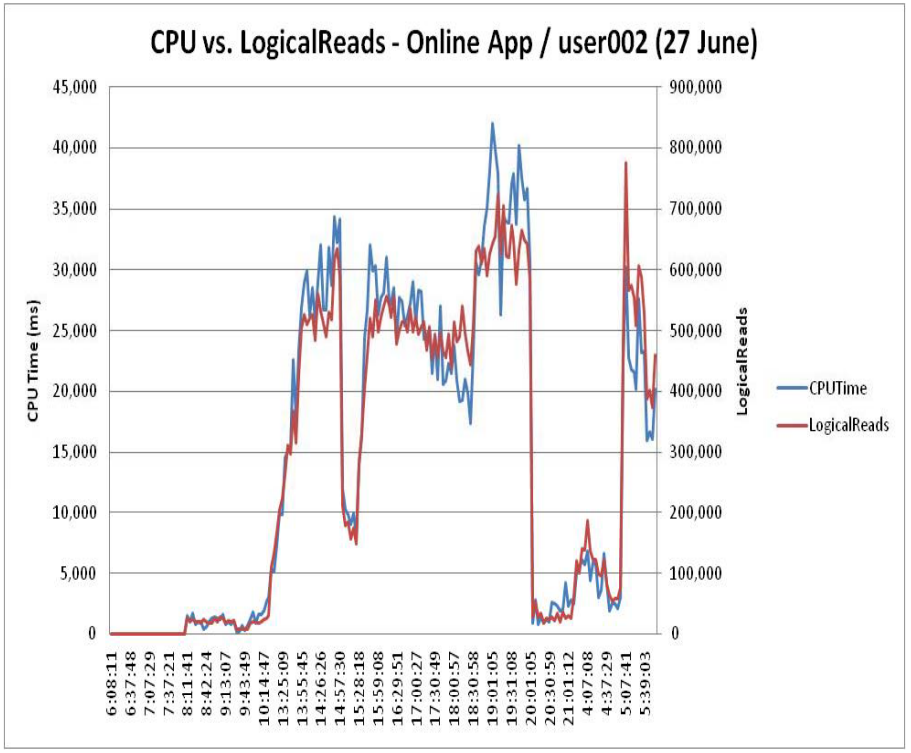
Login Name	Application	Num Spids	CPU Time	Physical Reads	Logical Reads	Physical Writes	Table Accesses	Index Accesses	TempDb Objects	Work Tables	Transactions
user001	Online App	20,240	17,134,300	4,030,711	502,141,353	133,049	263,846,621	53,430,946	142,065	1,652,938	1,022,516
user002	Online App	2,753	2,865,000	335,199	56,009,499	20,538	13,309,467	11,820,354	0	351,855	18,650
user003	Online App	228	463,400	55,784	13,841,034	11,932	6,410,238	1,969,054	13,114	26,210	93,710
user004	Online App	320	443,800	58,938	11,725,898	3,670	6,154,194	1,399,906	2,726	35,836	21,050
user005	Online App	282	431,200	94,202	9,982,600	2,000	5,434,460	1,177,286	566	34,122	6,494
user006	Online App	129	416,500	50,670	24,150,241	9,655	13,511,083	3,869,355	417	23,495	9,166
user007	Online App	139	403,100	28,982	9,181,661	2,466	3,117,682	1,617,146	2,655	20,209	20,408
user008	Online App	118	398,700	30,856	24,242,050	4,891	13,885,446	3,777,742	303	21,665	5,477
user009	Online App	85	396,700	3,548,113	20,345,446	618	19,779,044	3,425	9	11	60
user010	Online App	168	395,100	57,036	11,091,123	2,752	5,933,283	1,304,508	3,538	25,589	27,164
user011	Online App	124	381,800	31,348	8,399,175	4,020	2,324,360	1,735,376	952	8,806	9,621
user012	Online App	159	377,400	197,226	10,941,467	7,368	6,162,460	1,208,597	4,220	28,103	33,061
user013	Online App	167	366,600	44,299	9,407,562	2,935	5,007,985	948,675	485	33,698	6,806
user014	Online App	379	350,500	29,847	8,178,359	7,409	4,528,184	780,238	23	41,508	4,807
user015	Online App	268	349,200	39,597	6,938,306	1,909	4,092,065	710,051	1,049	15,817	9,377

Kind of what we are afraid of - most of the users all come in with a common set of logins with the predominant workload using 2 of them. The danger in trusting aggregation here is that the worst offender only shows 500LIO, 4PIO and 17ms CPU per transaction. You want to keep drilling down to find cause of CPU and then find cause of that…

WHAT IS DRIVING CPU (ONLINE)???

Health

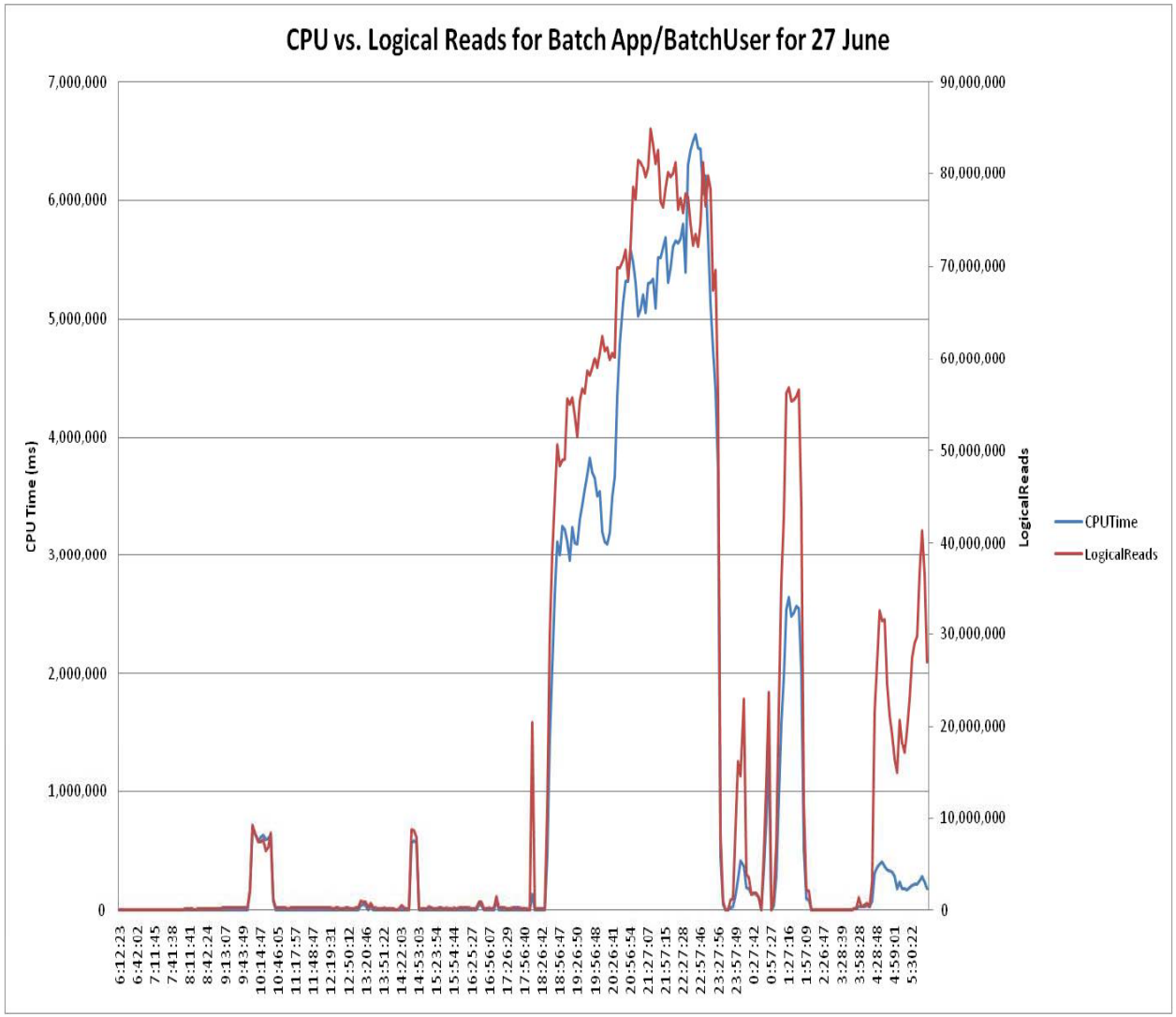




SYBASE | An SAP Company

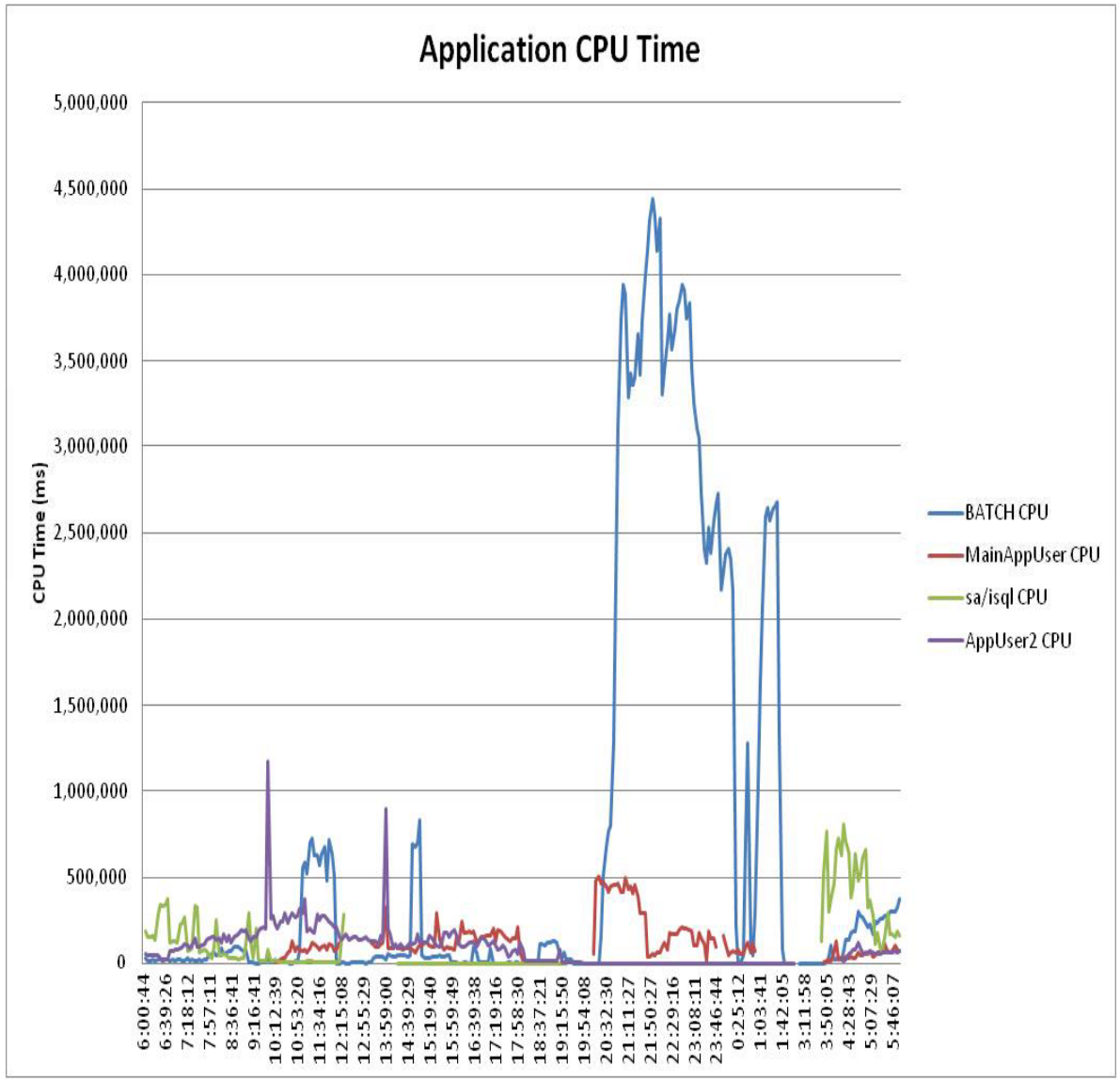
…AND A REPEAT (BATCH)

Health



SYBASE | An SAP Company

ANOTHER EXAMPLE (CPU BY APP/LOGIN)

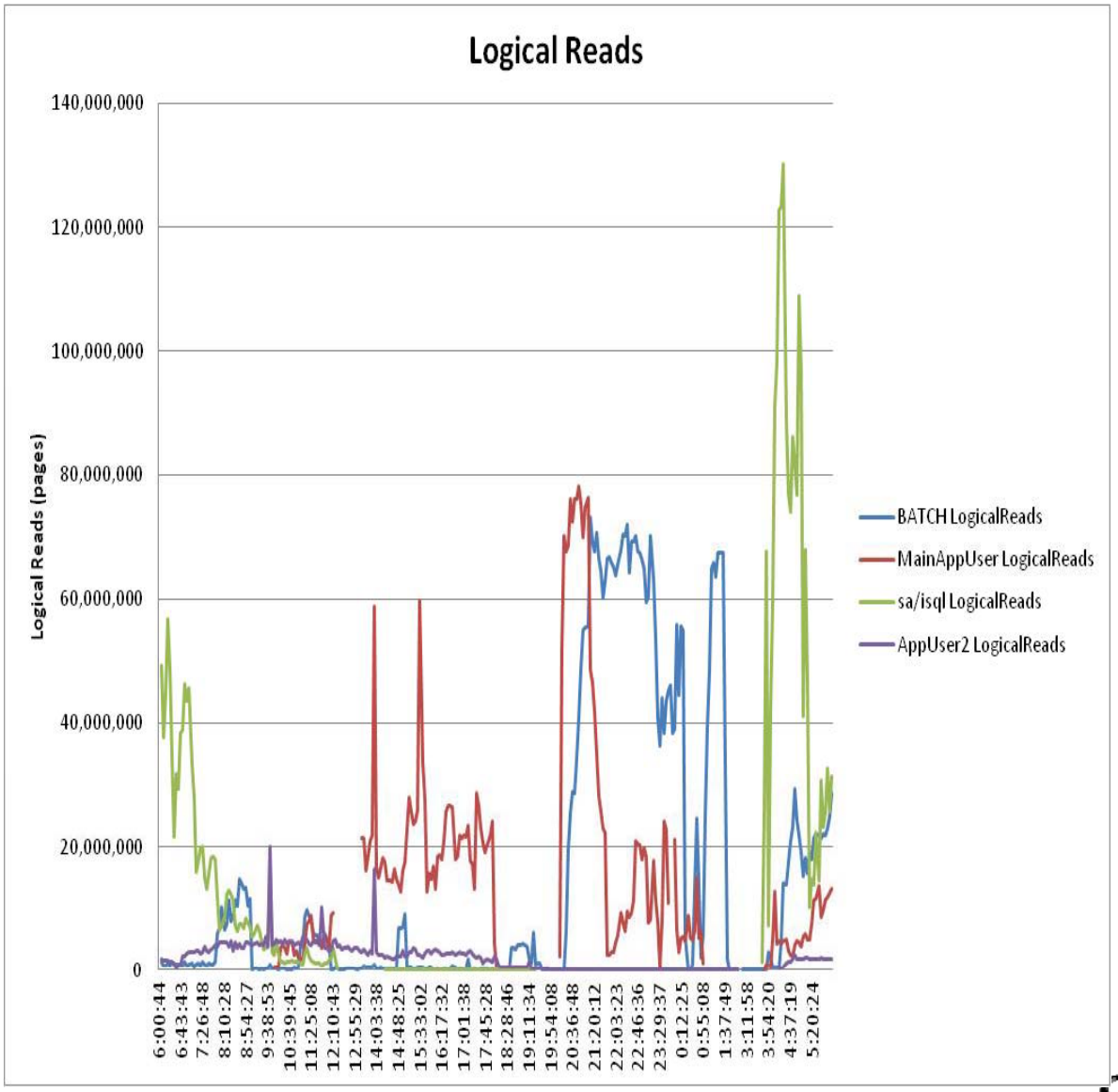


Health

SYBASE | An SAP Company

…..LIO BY APP/LOGIN….

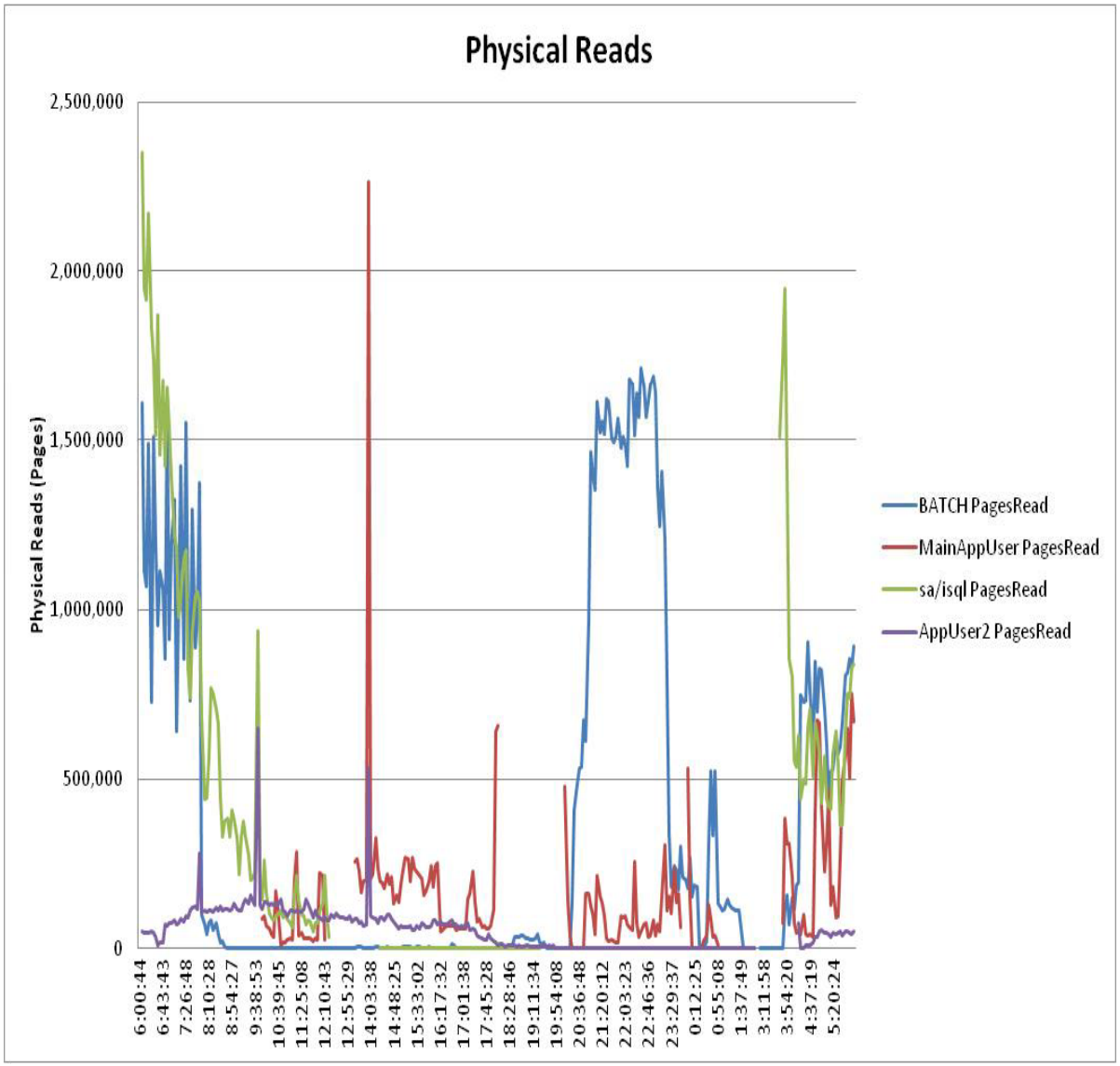
Health



SYBASE | An SAP Company

…..PHYSICAL READS BY APP/LOGIN

Health



SYBASE | An SAP Company

WAITEVENTS: AGGREGATED BY APP/LOGIN

WaitEventID	Description	WaitsTotal	WaitTimeTotal	msPerWait	% Waits	% WaitTime
250	waiting for incoming network data	84,046,503	5,335,886,000	63.4	40.2%	93.0%
251	waiting for network send to complete	71,838,685	118,921,200	1.6	34.4%	2.1%
29	waiting for regular buffer read to complete	19,297,502	136,818,600	7	9.2%	2.4%
150	waiting for a lock	9,637,820	74,080,800	7.6	4.6%	1.3%
52	waiting for i/o on MASS initated by another task	8,161,844	39,476,200	4.8	3.9%	0.7%
214	waiting on run queue after yield	4,505,339	427,000	0	2.2%	0.0%
55	wait for i/o to finish after writing last log page	4,414,327	5,891,800	1.3	2.1%	0.1%
36	waiting for MASS to finish writing before changing	3,430,326	4,855,700	1.4	1.6%	0.1%
51	waiting for last i/o on MASS to complete	1,151,936	4,965,100	4.3	0.6%	0.1%
124	wait for mass read to finish when getting page	1,041,433	6,422,700	6.1	0.5%	0.1%
54	waiting for write of the last log page to complete	983,762	2,065,300	2	0.5%	0.0%
272	waiting for lock on ULC	186,562	2,159,200	11.5	0.1%	0.0%
41	wait to acquire latch	157,179	916,500	5.8	0.1%	0.0%
53	waiting for MASS to finish changing to start i/o	140,471	2,900	0	0.1%	0.0%
31	waiting for buf write to complete before writing	19,363	30,200	1.5	0.0%	0.0%
37	wait for MASS to finish changing before changing	585	100	0.1	0.0%	0.0%
35	waiting for buffer validation to complete	262	2,400	9.1	0.0%	0.0%
215	waiting on run queue after sleep	8	5,685,300	710662.5	0.0%	0.1%
280	wait for access to a memory manager semaphore	3	0	0	0.0%	0.0%
	Totals	**209,013,910**	**5,738,607,000**			

Aggregated by joining monProcess/monProcessWaits

CPUTime was ~200,000,000ms so total elapsed is 5.9B ms (or CPU is 4% of total)

SOME THOUGHTS ABOUT MONITORING

Best Practices To Consider…Thus Far….

- **Regular Monitoring…not Before/After Snapshots**
  - ✓ Use 5-10 minute sample intervals
  - ✓ <5 minute samples is overkill and is just taxing the system
- **Use Enough Samples to get a Baseline**
  - ✓ Remember, though, you may have different baselines for different days of the week or peak periods - i.e. don't assume every payday is a bad day just because it spikes over day previous - baseline paydays
- **Use at least 2-3 tiers of aggregation**
  - ✓ **E.g. 1) Login; 2) Application; 3) Server-Wide**
  - ✓ Compute aggregates and store for later retrieval/redisplay
- **Plan Workload Separation Using Data**
  - ✓ E.g. Multiple tempdbs (by application), Engine Groups, etc.
- **Consider a repository using Sybase IQ**
  - ✓ May only need a DT license if only accessed by DBA's
  - ✓ 20 samples per hour, 10 hour active window, 2000 concurrent users
    - 400,000+ rows per day
    - 8M rows per month, ~100M rows per year

OBJECT ACTIVITY

APP PROFILES + FINDING WHAT IS CAUSING THE LIO'S

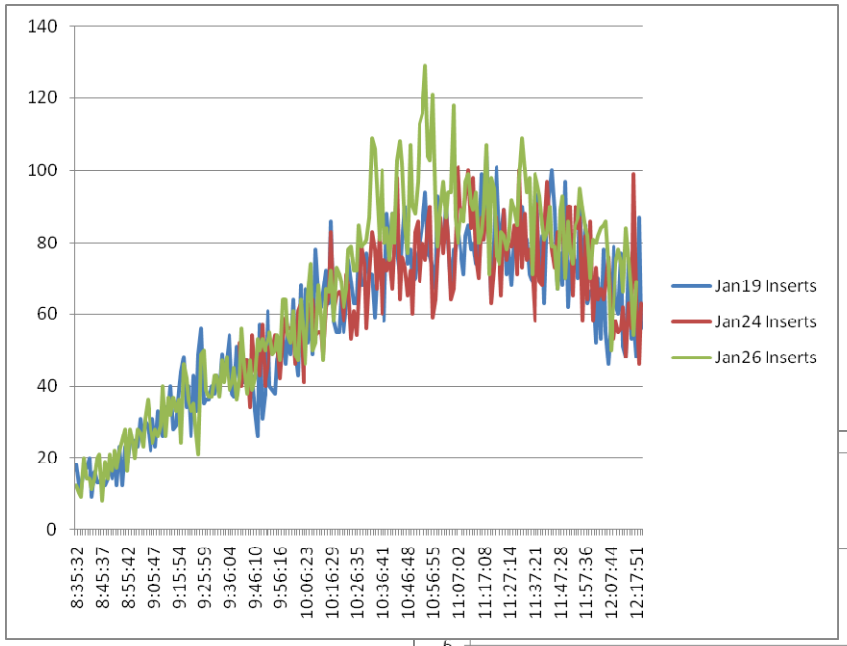
A QUICK ROADMAP SO FAR

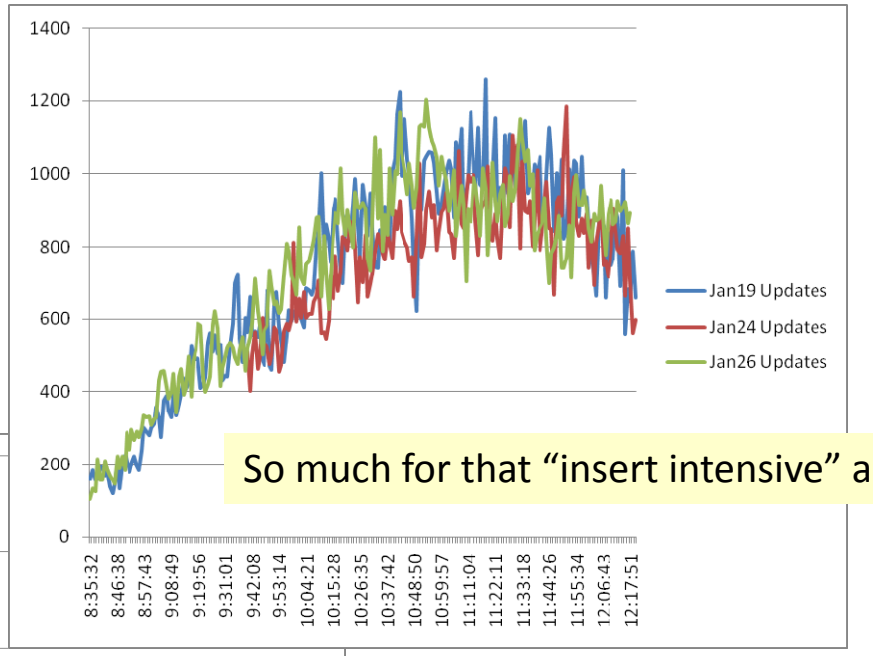
- **We used CPU via monEngine to determine**
  - ✓ When problems were happening
  - ✓ Baselining performance
- **We then use monProcess & monProcessActivity to**
  - ✓ Isolate the problematic applications and users
  - ✓ Determine if LIO is probable driver of CPU
- **Now we are going to use monOpenObjectActivity**
  - ✓ Profile/baseline the application (hot tables, etc.)
  - ✓ Find tables/indexes associated with problems

BASELINING OPEN OBJECTS

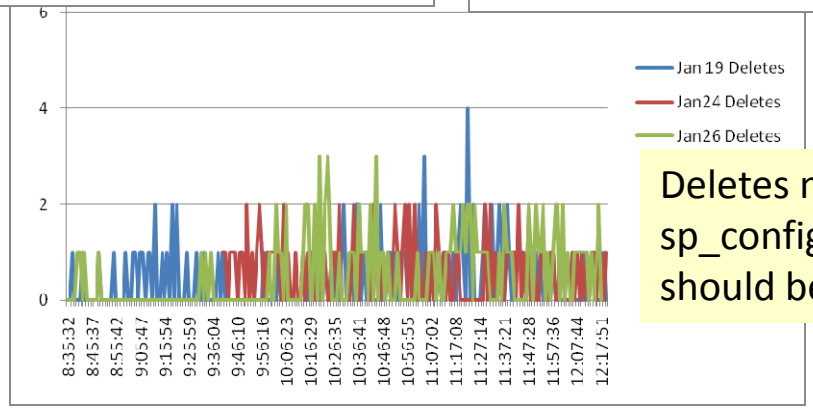
Comparing Inserts, Updates, Deletes aggregated across all objects

Bank





So much for that “insert intensive” app



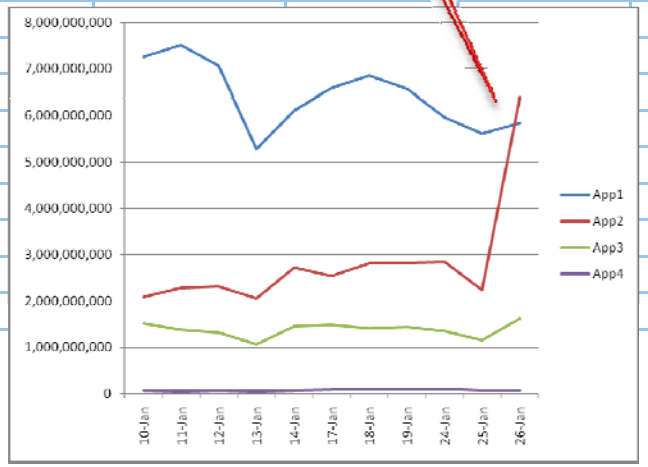
Deletes must be via batch…so sp_config ‘enable housekeeper GC’ should be set to ‘4’ at a minimum

SYBASE | An SAP Company

COMPARING OPEN OBJECTS

Comparing FSI baseline (Jan 19th) to problem day (Jan 26th) sorted by Logical Reads

DBName	ObjectName	IndexID	Diff LIO Reads	Diff PIO Reads	Diff PIO Writes	Diff Rows Inserted	Diff Rows Updated	Diff Rows Deleted	Diff Used Count
db1	ix1_quote	2	110,075,846	926	8	0	0	-93	44,650,574
db1	quote_details	0	73,589,160	-2,590	1,360	8,841	727	230	0
db2	auth_params	0	61,853,439	0	0	0	0	0	0
db1	quote	0	45,095,052	-1,853	7,722	4,899	33,324	-93	0
db3	quote_mod	0	35,853,360	-2,965	0	0	0	0	0
db4	www_registry	0	14,503,334	218	0	0	0	0	0
db2	original_quote	0	9,502,177	0	0	0	0	0	0
db5	aggregate_rmp	0	6,471,317	0	0	0	0	0	0
db2	process_ssp	0	5,715,933	0	0	0	0	0	0
db2	ix1_tableD	1	5,208,446	-2	7	-64	0	-62	1,768,811
db5	ix1_tableE	5	4,327,117	-2,099	-1,814	0	0	-3,429	1,547,988
db2	tableD	0	4,149,878	167	7	-64	0	-62	0
db4	tableA	0	3,034,923	-1,326	-156	-54	0	-54	0
db2	tableB	0	2,624,399	599	-3	-235	0	-234	0
db5	ix1_tableF	1	1,622,545	-382	-403	-1,208	0	500	384,521
db2	quote_idc	0	1,584,369	0	0	0	0	0	0
db1	ix1_quote_efg					0	0	-93	432,519
db2	remote_pro					-14	0	-10	0
db1	ix2_quotebyquote					-956	0	3	335,440
db1	ix1_quote_abc_efg					0	0	56,730	104,241
db4	www_line_auth					-1,345	0	-1,345	0
db2	ix1_scrub					0	0	0	535,289
db2	scrub					0	0	0	0
db2	ix1_Active_Emp					0	0	0	531,604
db2	Active_Emp					0	0	0	0



SYBASE | An SAP Company

…A BIT OF A CLOSER LOOK

Comparing the "quote" table…..19Jan vs. 26Jan

19-Jan	SampleDuration=12h00m26s											
Object Name	Index ID	Logical Reads	Physical Reads	APF Reads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	Operations	Lock Requests	Lock Waits	Used Count	LastUsedDate
quote	0	24,475,547	2,007,230	220,795	115,266	35,113	366	447,358	3,987,202	890,644	28	0	
quote_ix1	2	61,256,243	35,803	15,692	2,372	0	366	0	0			16,258,693	19-Jan-2011 19:39
quote_ix2	3	334,301	3,454	3	17,911	43,569	8,822	0	0			4,302	19-Jan-2011 19:39
quote_ix3	4	2,163,971	3,181	164,269	27,928	55,243	20,496	0	0			251	19-Jan-2011 18:57
quote_ix4	5	2,004,840	10,715	0	106,029	267,897	233,150	0	0			0	
quote_ix5	6	466,124	2,488	0	25,491	75,511	40,764	0	0			0	
quote_ix6	7	142,461	1,452	0	5,992	35,113	366	0	0			0	
quote_ix7	8	1,732,295	8,864	138,200	72,854	218,241	183,494	0	0			14,597	19-Jan-2011 19:39
quote_ix8	9	888,949	8,925	28,335	67,971	125,168	90,421	0	0			0	

26-Jan	SampleDuration=9h07m49s											
Object Name	Index ID	Logical Reads	Physical Reads	APF Reads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	Operations	Lock Requests	Lock Waits	Used Count	LastUsedDate
quote	0	69,657,132	171,949	189,621	111,103	37,878	262	450,667	4,678,401	1,648,764	154	0	25-Jan-2011 22:58
quote_ix1	2	173,732,122	3,577	4,349	1,887	0	262	0	0			63,351,417	26-Jan-2011 17:42
quote_ix2	3	332,873	2,386	3	16,085	45,913	8,297	0	0			23,629	26-Jan-2011 17:42
quote_ix3	4	2,466,180	67	6,761	28,379	59,570	21,954	0	0			187	26-Jan-2011 17:40
quote_ix4	5	2,056,652	9,342	0	99,761	275,658	238,042	0	0			5	26-Jan-2011 17:03
quote_ix5	6	450,687	827	0	18,883	74,959	37,343	0	0			0	26-Jan-2011 4:08
quote_ix6	7	154,159	364	0	4,642	37,878	262	0	0			0	
quote_ix7	8	1,781,512	7,347	144,649	67,537	227,193	189,574	0	0			12,563	26-Jan-2011 17:42
quote_ix8	9	948,351	7,716	24,838	62,614	134,204	96,588	0	0			0	

DML appears to be the same (although we might need to normalize for 12h vs. 9h) ….**big diff is the UsedCount on quote_idx1** - not covered queries as 45M increase in index access is nearly same as 45M increase in table LIOs…and coupled with sky rocketing worktables (earlier graphs) means joins/complex queries) ….plus a change in quote_idx2 used count but not a big diff in LIO…relative overall usage is low enough though (comparatively) that if we normalize the 9h vs. 12h a slight diff in LIO is possible - not a major factor though…

DID YOU NOTICE???

Txn speed degraded by maintaining volatile indexes that are never used.

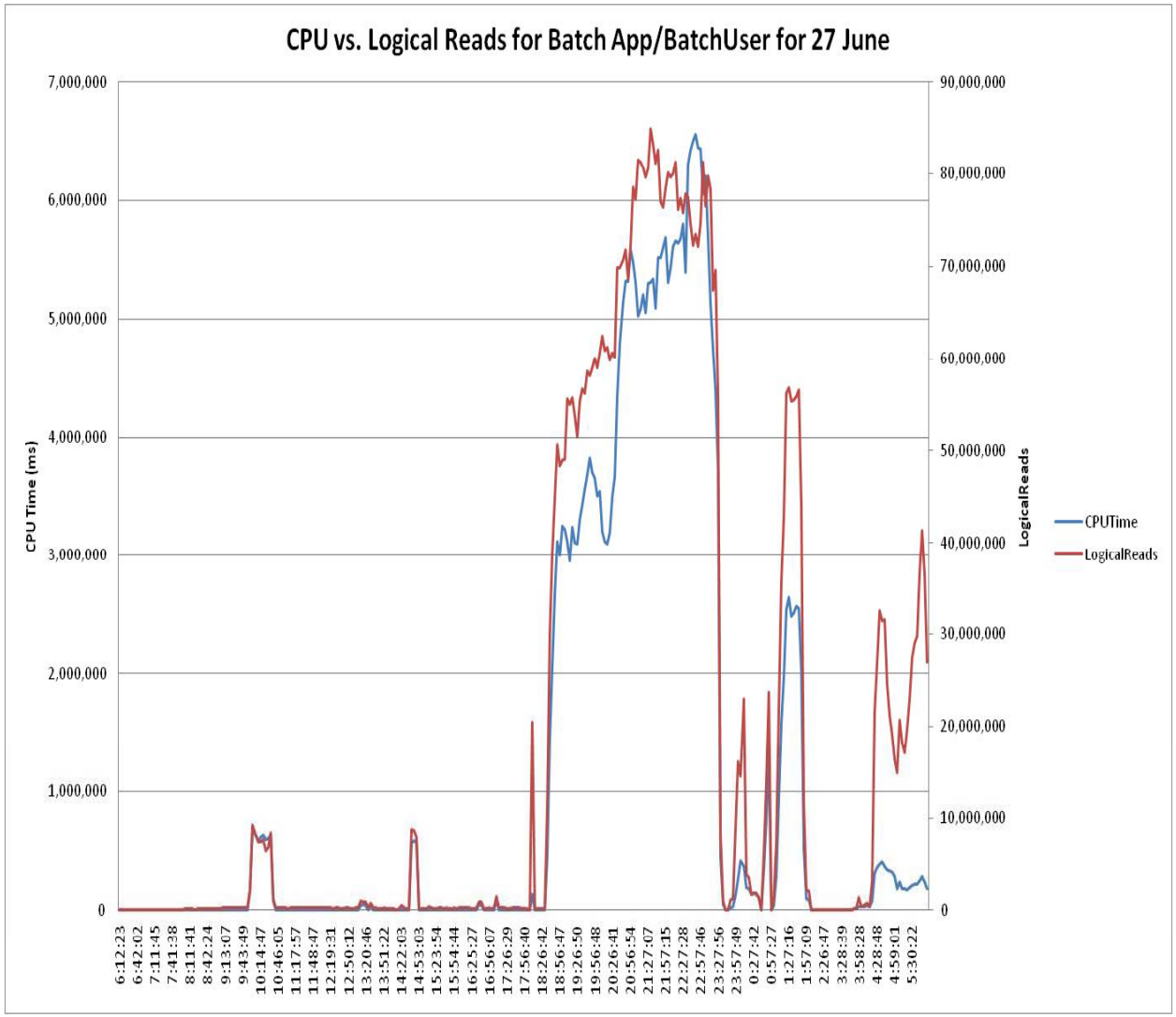
19-Jan	SampleDuration=12h00m26s												
Object Name	Index ID	Logical Reads	Physical Reads	APF Reads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	Operations	Lock Requests	Lock Waits	Used Count	LastUsedDate
quote	0	24,475,547	2,007,230	220,795	115,266	35,113	366	447,358	3,987,202	890,644	28	0	
quote_ix1	2	61,256,243	35,803	15,692	2,372	0	366	0	0			16,258,693	19-Jan-2011 19:39
quote_ix2	3	334,301	3,454	3	17,911	43,569	8,822	0	0			4,302	19-Jan-2011 19:39
quote_ix3	4	2,163,971	3,181	164,269	27,928	55,243	20,496	0	0			251	19-Jan-2011 18:57
quote_ix4	5	2,004,840	10,715	0	106,029	267,897	233,150	0	0			0	
quote_ix5	6	466,124	2,488	0	25,491	75,511	40,764	0	0			0	
quote_ix6	7	142,461	1,452	0	5,992	35,113	366	0	0			0	
quote_ix7	8	1,732,295	8,864	138,200	72,854	218,241	183,494	0	0			14,597	19-Jan-2011 19:39
quote_ix8	9	888,949	8,925	28,335	67,971	125,168	90,421	0	0			0	

26-Jan	SampleDuration=9h07m49s												
Object Name	Index ID	Logical Reads	Physical Reads	APF Reads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	Operations	Lock Requests	Lock Waits	Used Count	LastUsedDate
quote	0	69,657,132	171,949	189,621	111,103	37,878	262	450,667	4,678,401	1,648,764	154	0	25-Jan-2011 22:58
quote_ix1	2	173,732,122	3,577	4,349	1,887	0	262	0	0			63,351,417	26-Jan-2011 17:42
quote_ix2	3	332,873	2,386	3	16,085	45,913	8,297	0	0			23,629	26-Jan-2011 17:42
quote_ix3	4	2,466,180	67	6,761	28,379	59,570	21,954	0	0			187	26-Jan-2011 17:40
quote_ix4	5	2,056,652	9,342	0	99,761	275,658	238,042	0	0			5	26-Jan-2011 17:03
quote_ix5	6	450,687	827	0	18,883	74,959	37,343	0	0			0	26-Jan-2011 4:08
quote_ix6	7	154,159	364	0	4,642	37,878	262	0	0			0	
quote_ix7	8	1,781,512	7,347	144,649	67,537	227,193	189,574	0	0			12,563	26-Jan-2011 17:42
quote_ix8	9	948,351	7,716	24,838	62,614	134,204	96,588	0	0			0	

Half of the indexes are never used (except one has low usage on 26th) - and 3 of the 4 are volatile compared to the updates - do we really need these indexes??? (**NO**) Is an index that is only used 5 times in 9 hours on the odd date useful??? Depends on whether it might use a different index if there and what a table scan costs….something to investigate

REMEMBER ME???

Health



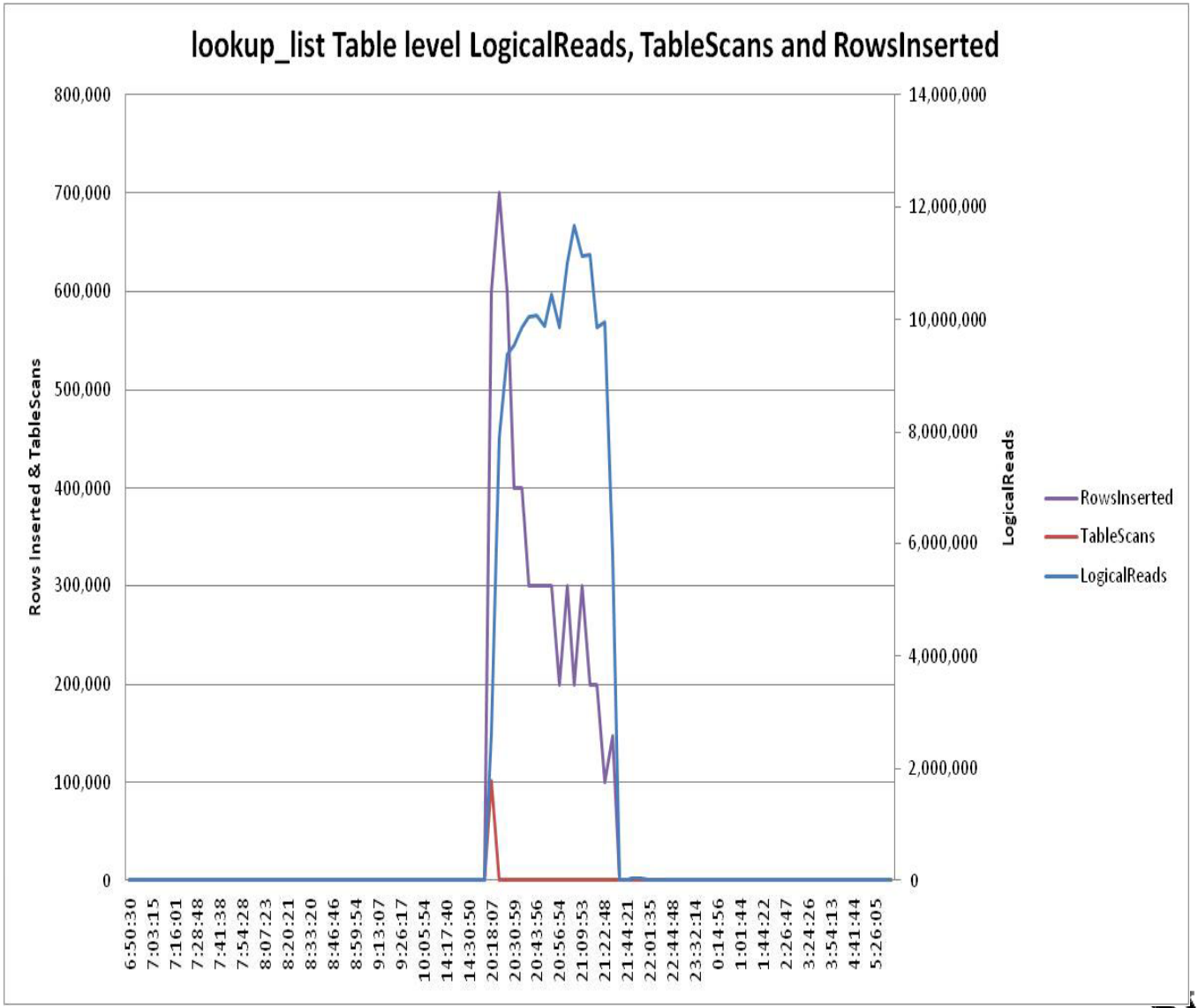
SYBASE | An SAP Company

THE TOP LIO OBJECTS

DBName	TableName	Index ID	Logical Reads	Logical PerSec	Physical Reads	APFReads	Physical Writes	Rows Inserted	Rows Deleted	Rows Updated	UsedCount
Lookup	Lookup_list	2	518,826,902	29,118	3	0	104,918	5,547,883	0	0	159,658,672
Appdb	Note_text	0	233,879,797	13,126	772	21,445	3,011	6,911	0	0	6,911
Appdb	App_definitions	0	194,455,526	10,913	34,126	111,080	0	0	0	0	0
Lookup	Lookup_list	0	160,244,892	8,993	3	0	30,282	5,547,883	0	0	102,335
Appdb	Payments	0	126,623,592	7,106	272,706	2,987,731	0	0	0	0	0
Appdb	Table1	0	117,456,511	6,592	14,833	52,407	354	5,649	0	0	1
Appdb	Suppliers	0	54,555,524	3,061	1,831	1,467	0	0	0	0	0
Lookup	Customer_Group	0	42,051,344	2,360	1,109	0	0	0	0	0	10,224
Appdb	Limits_Liabilities	1	38,811,787	2,178	55	0	53	334	0	0	19,405,541
Appdb	Entity_Info	4	37,069,351	2,080	4,480	0	39,530	4,627,974	4,628,017	0	0
Appdb	Table2	0	36,410,176	2,043	1,043,937	1,984,473	159,553	419,189	17,717	44,855	419,358
Appdb	Table3	1	35,800,669	2,009	45	34,855	29	22	0	0	11,932,138
Appdb	Eligibility	0	30,568,139	1,715	2,850,076	9,862,403	11	26	14	0	26
Appdb	Entity_Info	3	28,429,942	1,595	25,503	0	109,295	4,627,908	4,627,987	0	226,739
Appdb	Provider	2	27,992,291	1,571	1,586	1	0	0	0	0	16,080,756
Appdb	Entity_Info	2	27,812,237	1,560	9,769	10	4	0	0	0	9,275,528

MONOPENOBJECTACTIVITY OVER TIME

Health



SYBASE | An SAP Company

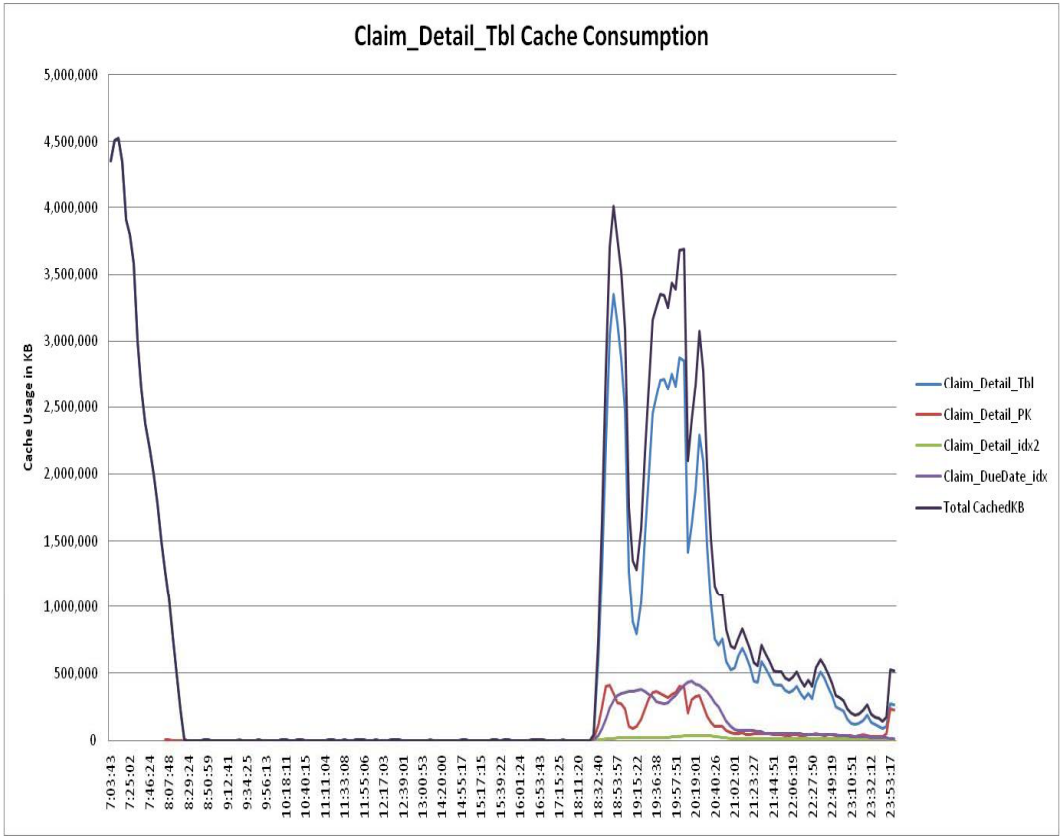
WHAT TO DO, WHAT TO DO

Reducing LIO and CPU for Batch Processing

- **Three tables causing issues**
- **Two of the tables (yellow) have obvious indexing issues**
  - ✓ The table (indid=0) appears in the top LIO list long before any index
  - ✓ Table pages read are at least 10x of an index reads just based on top n
    - Reality is much worse than that
- **The lookup_list table is an interesting challenge**
  - ✓ The table has about 5,000,000 rows in it
  - ✓ Yes, the 102,335 table scans are an obvious bad thing
  - ✓ But the ~160 million queries that used an index in 5 hours is a bit much
    - That's ~9,000 queries a second
      - remember our 5,000 webservice calls/sec
    - Sniff….sniff…I smell a loop somewhere….and it is stinkin'

PHYSREADS, CACHING & LIO

Health



The price of 800K probable table scans (0 RowsInserted into PK index rules out heap inserts plus the APF's using Large IO's show scan behavior) that cause 2M PhysReads (5.5M PagesRead)

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

SYBASE | An SAP Company

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

STORED PROCEDURES

ISOLATING APPLICATION ISSUES

WHERE DO WE GO FROM HERE

Tracing the Cause of the LIO Spike

- **We now know….**
  - ✓ Seems to be centered on several tables with one key table (quotes)
  - ✓ It could be a runaway join executed a few times…..
  - ✓ ….or a single or a few queries executed a ton of times more often
- **A Runaway Join**
  - ✓ Would show high CPU time than previous executions (and higher LIO)
- **A Big Jump in Executions**
  - ✓ …would show a large increase in certain statements executed more often
- **Of course, it could be a mix….**
- **…but we need to look at statement metrics to find out**
  - ✓ App leverages stored procs heavily, so we have better traceability if it is a single statement (would also work with Statement Cache)
  - ✓ If it was adhoc queries, we would just see a lot of statements but no way to associate if queries are same (without monSysSQLText)

DEBUGGING PROCS: MONSYSSTATEMENT

- **One of the most useful tables in MDA**
  - ✓ Set the pipesize high (10,000+)
  - ✓ poll frequently (every 3-5 seconds) to avoid loosing too many events
- **Understand the execution order**
  - ✓ For a given SPID+KPID (& InstanceID)
    - BatchID
    - ContextID
    - LineNumber
  - ✓ …or cheat via (SPID+KPID)
    - BatchID
    - StatementNumber
- **Problem Proc Lines are easy**
  - ✓ Use DBID + ProcedureID + LineNumber
  - ✓ You can even use PlanID if you suspect that different proc plans are getting bad plans.
- **Watch the clock drift getting reset**
  - ✓ Is EndTime < StartTime???

monSysStatement		
SPID	int	<fk1>
InstanceID	tinyint	<fk1,fk2>
KPID	int	<fk1>
DBID	int	
ProcedureID	int	
PlanID	int	
BatchID	int	
ContextID	int	
LineNumber	int	
CpuTime	int	
WaitTime	int	
MemUsageKB	int	
PhysicalReads	int	
LogicalReads	int	
PagesModified	int	
PacketsSent	int	
PacketsReceived	int	
NetworkPacketSize	int	
PlansAltered	int	
RowsAffected	int	
ErrorStatus	int	
HashKey	int	<fk2>
SsqlId	int	<fk2>
ProcNestLevel	int	
StatementNumber	int	
DBName	varchar(30)	
StartTime	datetime	
EndTime	datetime	

MONSYSSTATEMENT & CPU TIME

Interpreting CPU Time for Stored Procedure Lines in MonSysStatement

- **monSysStatement CPUTime is derived**
  - ✓ ElapsedTime - WaitTime
  - ✓ Clock drift can make this larger than reality or negative (so watch for -2B)
  - ✓ WaitTime is defined as amount of time that the particular statement waited on a defined WaitEvent
    - E.g. Waited on disk I/O, network I/O, etc.
- **Impact on Stored Procs**
  - ✓ When one stored proc calls another
  - ✓ Calling statement has elapsed time, but no WaitTime
    - It is the sub-proc statements that incur WaitTime as disk I/O etc. are required
  - ✓ Result is that the calling statement will report CPUTime as same as ElapsedTime
    - Mixed in due to this is any CPU time spent recompiling that procedure line as well as any procedure lines in proc lines nested in the subproc calls
- **Spotting Sub-Procedure Executions/Recompilation**
  - ✓ Look for statements with high CPUTime, 0 WaitTime, RowsAffected=Exec Count and <1000 LogicalReads on average
    - LogicalReads will happen as ASE re-reads sysprocedures to recompile
  - ✓ MemUsageKB on calling line is recompilation overhead
  - ✓ If MemUsageKB is high and CPUTime not justified by nested statement elapsed time, some portion of CPUTime may be due to recompilation

TOP PROC LINES BY TOTAL CPU

19-Jan											
DBName	ObjectName	Line Number	Num Execs	Elapsed avg	Elapsed tot	CpuTime avg	CpuTime tot	Physical Reads tot	Logical Reads avg	Logical Reads tot	Rows Affected tot
db1	proc1	57	13	12,783	166,190	8,599	111,787	6,388	2,477,579	32,208,528	0
db2	proc2	29	411	374	153,923	106	43,566	28,804	56,892	23,382,650	4,310
db3	proc3	55	53,883	1	102,352	1	92,305	0	421	22,728,015	53,883
db1	proc1	51	11	7,791	85,709	3,691	40,602	5,784	1,608,890	17,697,798	0
db4	proc4	517	9,372	10	93,779	9	92,770	0	1,095	10,264,694	9,372
db4	proc4	540	9,353	10	99,840	10	98,606	0	1,086	10,157,358	9,353
db5	proc5	762	1,391	25	35,549	23	32,544	3	6,070	8,443,892	1,351,442
db1	delete_trigger1	45	11	2,047	22,518	737	8,116	1,767	449,468	4,944,155	0
db2	proc6	32	26	935	24,334	297	7,727	2,528	145,566	3,784,736	598
db6	proc7	62	3,051	48	148,821	10	31,854	17,663	1,091	3,329,580	107,129
db1	delete_trigger2	155	14	681	9,546	424	5,936	465	221,789	3,105,051	0
db6	proc8	468	5,205	127	663,386	7	41,418	34,768	566	2,946,627	121,224
db5	proc5	1,325	1,195	19	22,904	6	7,599	878	2,299	2,748,147	254

26-Jan											
DBName	ObjectName	Line Number	Num Execs	Elapsed avg	Elapsed tot	CpuTime avg	CpuTime tot	Physical Reads tot	Logical Reads avg	Logical Reads tot	Rows Affected tot
db3	proc3	55	184,863	2	389,965	1	350,981	0	454	84,078,413	184,861
db1	proc13	60	19,889	5	115,731	4	92,449	12	3,678	73,152,376	103
db5	proc5	1,325	3,163	38	200,341	24	123,985	2,821	9,402	48,543,741	1,072
db5	proc5	762	5,143	32	165,092	28	145,429	0	6,386	32,847,339	5,190,790
db3	proc9	3,276	989,173	0	66,150	0	53,915	0	16	16,815,567	989,152
db3	proc9	1,054	164,854	0	48,830	0	37,800	69	79	13,166,654	158,915
db3	proc3	43	184,801	0	117,736	0	97,225	0	69	12,844,383	184,800
db2	proc10	49	7	6,719	47,034	6,633	46,431	0	1,747,962	12,235,739	420
db6	proc11	103	15,216	10	166,570	4	69,082	3,970	787	11,977,292	0
db6	proc12	1,320	289,083	0	37,172	0	30,107	0	36	10,517,355	289,075
db4	proc5	710	4,815	14	71,001	10	48,671	0	2,109	10,158,535	2,604,915
db3	proc9	1,065	128,598	0	36,735	0	28,486	481	75	9,767,596	111,238
db3	proc9	922	996,841	0	40,734	0	29,957	0	8	8,971,272	996,810

PROC LINES BY EXEC COUNT DIFF

DBName	ObjectName	LineNumber	24th Execs	26th Execs	Execs Diff
db1	stored_proc_176	4,569	35,783	442,784	407,001
db1	stored_proc_176	4,571	32,507	402,568	370,061
db1	stored_proc_176	4,594	32,471	402,422	369,951
db1	stored_proc_176	…(8)			
db1	stored_proc_6178	347	16,921	93,944	77,023
db1	stored_proc_176	4,587	6,475	80,008	73,533
db2	strored_proc_1838	2,201	331,614	388,107	56,493
db1	stored_proc_176	737	2,758	50,426	47,668
db1	stored_proc_6175	1,244	2,862	50,202	47,340
db1	stored_proc_6175	1,747	2,160	48,887	46,727
db1	stored_proc_6178	404	4,184	49,757	45,573
db1	stored_proc_6178	352	7,186	52,253	45,067
db1	stored_proc_6178	530	7,177	52,240	45,063
db1	stored_proc_6178	524	7,195	52,246	45,051
db1	stored_proc_176	837	3,682	48,345	44,663
db1	stored_proc_176	…(5)			
db1	stored_proc_6175	1,248	2,426	41,986	39,560
db1	stored_proc_6175	…(22)			
db1	stored_proc_6178	215	11,056	49,670	38,614
db1	stored_proc_176	891	2,974	40,657	37,683
db2	stored_proc_9456	66	2,954	40,636	37,682
db1	stored_proc_176	861	3,011	40,673	37,662
db1	stored_proc_176	…(4)			
db2	stored_proc_9456	60	2,976	40,606	37,630
db2	stored_proc_9456	68	3,038	40,656	37,618
db1	stored_proc_176	882	3,008	40,622	37,614
db1	stored_proc_176	912	3,003	40,580	37,577
db1	stored_proc_176	1,146	3,045	40,591	37,546
db2	stored_proc_9456	58	2,989	40,527	37,538
db1	stored_proc_176	3,657	2,969	40,491	37,522
db1	stored_proc_176	…(35)			

TRACKING DOWN PROBLEM LINES

How To Ignore Sub-Proc Execution Time

26-Jan									
DBName	ObjectName	Line Number	Num Execs	CpuTime avg	CpuTime tot	LogicalReads avg	LogicalReads tot	MemUsageKB tot	RowsAffected tot
db1	stored_proc_G08742	932	382	6,141	2,345,891	331	126,698	25,952	382
db1	stored_proc_G06176	6,052	4,394	259	1,138,406	22	99,662	329,238	4,394
db1	stored_proc_G06176	5,570	4,097	147	602,882	43	178,314	496,106	4,097
db1	stored_proc_G06176	4,740	1,619	273	443,176	65	105,456	193,292	1,619
db1	stored_proc_G10772	1,439	88	1,882	165,637	332	29,243	6,478	88
db1	stored_proc_G06168	1,206	282	516	145,664	132	37,247	19,516	282
db2	stored_proc_S17351	55	74,624	1	139,494	454	33,947,003	0	74,623
db3	stored_proc_G14375	762	1,520	27	41,842	6,349	9,651,460	49,458	1,526,504
db1	stored_proc_G06176	5,160	292	140	40,916	60	17,747	35,126	292
db1	stored_proc_I04729	47	7,968	5	40,603	15	123,402	137,756	7,799
db2	stored_proc_S17351	43	74,668	0	39,252	69	5,187,084	0	74,667
db3	stored_proc_G14375	1,325	1,530	24	36,994	9,196	14,071,138	0	296
db1	stored_proc_S16177	60	7,796	4	36,807	3,678	28,680,138	251,060	37
db1	stored_proc_G07201	103	6,629	5	33,278	797	5,287,868	0	0
db3	stored_proc_G14375	1,604	1,554	20	32,440	346	537,691	0	0
db2	stored_proc_G01838	1,077	47,590	0	30,157	76	3,649,951	0	23,550
db4	stored_proc_S13648	85	8,797	3	28,717	278	2,447,667	0	8,774
db5	stored_proc_S11230	338	326	84	27,696	129	42,181	18,188	0
db1	stored_proc_D01378	282	4,984	5	27,395	16	81,422	0	5,100
db1	stored_proc_G06530	340	2,650	10	26,654	45	119,926	48,904	2,650

Highlighted lines are those from bad queries…not only high LogicalReads, but also some sorting/reformatting based on MemUsageKB

COMMENTS ABOUT GRAYED OUT LINES

Examples of identifying sub-proc calls…and isolating nested issue…..

- **All of the grayed out lines are sub-proc calls**
  - ✓ Hence CPUTime is actually ElapsedTime (as discussed earlier)
- **db1..stored_proc_G08742 line 932**
  - ✓ 3x increase in execution counts, 15x increase in elapsed time
  - ✓ Call Stack:
    - …calls stored_proc_G06172
    - …which calls stored_proc_G06174 at line 1545
    - …which calls stored_proc_G06176 at line 553
- **db1.. stored_proc_G06176**
  - ✓ Lines 4740, 5570 and 6052 with significant increases in execution count
  - ✓ All three lines plus line 5160 call stored_proc_G06177 (next slide)
- **db1.. stored_proc_G10772**
  - ✓ Line 1439 calls stored_proc_G06172
- **db1.. stored_proc_G06168**
  - ✓ Line 1206 calls stored_proc_G06170

All of these are grayed out on previous slide

STORED_PROC_G06177

Lines By Total CPUTime

Line Number	Num Execs	Elapsed avg	Elapsed max	Elapsed tot	CpuTime avg	CpuTime max	CpuTime tot	WaitTime avg	WaitTime max	WaitTime tot	Logical Reads tot	Mem UsageKB tot	Packets Sent tot	Rows Affected avg
1,040	3,422	79	1,080	272,598	79	1,080	271,716	0	200	200	0	424,138	43	1
7,255	4,809	85	1,914	410,266	8	96	40,817	76	1,900	367,086	1	0	0	0
7,278	2,750	67	2,120	186,983	6	100	18,977	60	2,120	167,316	0	345,624	0	1
825	2,817	3	1,310	9,631	3	1,310	9,116	0	0	0	0	360,386	24	0
996	3,476	1	183	4,013	0	43	2,862	0	183	832	724,144	0	0	0
546	10,565	0	76	2,054	0	76	1,865	0	0	0	443,473	0	39	0
696	7,775	0	250	2,084	0	30	1,567	0	86	86	368,770	0	37	0
734	7,811	0	657	2,109	0	73	1,297	0	0	0	327,439	0	28	0
7,351	10,409	0	363	1,525	0	43	968	0	100	100	93,663	11,131,166	0	0
505	10,581	0	2,547	3,853	0	90	828	0	150	388	75,575	0	37	0
608	10,616	0	503	4,078	0	56	803	0	500	3,218	66,346	0	29	0
7,384	9,869	0	1,240	3,325	0	40	803	0	1,200	2,435	98,690	0	0	0
7,246	7,577	0	623	3,441	0	60	757	0	623	2,641	52,822	81,710	0	0
650	7,775	0	284	2,142	0	100	663	0	283	1,201	65,800	0	37	0
1,012	3,477	0	34	669	0	16	571	0	33	33	114,414	1,587,036	0	15

STORED_PROC_G06177

Sifting the Wheat from the Chaff

- **Line 1040 calls stored_proc_G05405**
  - ✓ In which, lines 1050 and 1073 have high elapsed but no other metrics of any note
  - ✓ Both lines are if clauses with nested query using PKey
  - ✓ Elapsed could be due to clock drift as only 3 rows (of 1,000's of execs) have elapsed > 1ms
- **Lines 7255 & 7278 are queries**
  - ✓ Biggest issue is the 'WaitTime'
  - ✓ Line 7255 does an update
    - Which invokes an update trigger
  - ✓ Line 7278 does an insert
    - Which invokes insert trigger
- **Lines 7351 and 1012 are queries with a lot of sorting**
  - ✓ Looks like reformatting due to no index or merge join

LINES 7255 & 7278 WAITS

SPID	KPID	PlanID	BatchID	ContextID	LineNumber	CpuTime	WaitTime	MemUsageKB	StatementNumber	StartTime	EndTime
1,882	429,785,620	8,810,573	33,171	369	7,255	0	1,283	0	31,528	10:14:10	10:14:11
1,737	428,802,670	8,961,923	23,363	378	7,255	73	1,300	0	32,987	10:29:56	10:29:57
362	427,689,747	8,961,845	19,836	184	7,255	0	1,700	0	9,914	10:29:56	10:29:58
362	427,689,747	8,965,706	19,836	454	7,255	13	1,900	0	38,302	10:30:26	10:30:28
1,243	444,466,075	9,132,212	40,374	159	7,255	0	1,376	0	8,071	10:45:17	10:45:19
2,130	426,705,121	9,475,784	6,581	468	7,255	50	1,100	0	40,312	11:14:40	11:14:41
793	429,458,028	9,493,702	57,994	232	7,278	0	1,360	122	13,597	11:16:21	11:16:23
1,679	427,821,047	9,654,535	58,557	113	7,278	76	1,100	136	4,945	11:30:47	11:30:48
1,679	427,821,047	9,669,899	58,919	110	7,278	16	1,500	136	4,940	11:32:05	11:32:06
1,347	429,654,672	9,766,225	42,073	166	7,278	6	1,400	140	8,324	11:40:06	11:40:07
319	426,902,081	9,780,813	40,597	402	7,255	46	1,100	0	33,773	11:41:39	11:41:40
190	428,671,565	9,781,857	27,957	127	7,278	50	1,100	136	6,161	11:41:39	11:41:40
859	428,999,290	9,792,884	52,431	164	7,278	0	1,786	136	8,496	11:42:45	11:42:47
362	427,689,747	9,813,353	26,535	111	7,278	0	1,453	138	4,773	11:44:17	11:44:18
859	428,999,290	9,835,728	53,883	385	7,255	0	1,293	0	34,019	11:46:59	11:47:01
2,107	426,639,880	9,836,001	21,410	378	7,255	0	1,286	0	31,699	11:46:59	11:47:01
362	427,689,747	9,836,892	27,099	444	7,255	0	1,396	0	38,047	11:47:10	11:47:11
330	427,753,726	9,877,845	51,891	368	7,255	0	1,090	0	31,092	11:51:25	11:51:26
1,836	428,474,683	9,919,779	37,170	127	7,278	0	2,120	122	6,211	11:55:22	11:55:24
437	427,951,538	10,000,488	122,886	187	7,255	40	1,100	0	10,566	12:02:06	12:02:07

We are only sampling ProcessWaitEvents every 15 seconds - so we will not have a per statement sampling of what the waits were. However, we have 4 SPIDS and batches fairly close proximity time wise, so we can narrowly scope to a 15 second sample period containing the above shaded statements and see the results.

SPID 362 10:30 & 11:42

WaitEventID	WaitClass	WaitEvent	Waits	WaitTimeSec	msPerWait	delta_sec	delta_hms
283	waiting on another thread	Waiting for Log writer to complete	337	34.4	102	107	0:01:01
250	waiting for input from the network	waiting for incoming network data	300	28.5	95	107	0:01:01
150	waiting to take a lock	waiting for a lock	190	15.3	80	107	0:01:01
36	waiting for memory or a buffer	waiting for MASS to finish writing before changing	38	0	0	107	0:01:01
29	waiting for a disk read to complete	waiting for regular buffer read to complete	34	2.2	64	107	0:01:01
214	waiting to be scheduled	waiting on run queue after yield	30	0.3	10	107	0:01:01
31	waiting for a disk write to complete	waiting for buf write to complete before writing	16	0.1	6	107	0:01:01
251	waiting to output to the network	waiting for network send to complete	14	0	0	107	0:01:01
272	waiting for internal system event	waiting for lock on ULC	4	0.2	50	107	0:01:01
41	waiting for internal system event	wait to acquire latch	2	0	0	107	0:01:01

WaitEventID	WaitClass	WaitEvent	Waits	WaitTimeSec	msPerWait	delta_sec	delta_hms
250	waiting for input from the network	waiting for incoming network data	566	66.8	118	184	0:03:03
283	waiting on another thread	Waiting for Log writer to complete	378	35.1	92	184	0:03:03
150	waiting to take a lock	waiting for a lock	241	23.2	96	184	0:03:03
29	waiting for a disk read to complete	waiting for regular buffer read to complete	58	2	34	184	0:03:03
36	waiting for memory or a buffer	waiting for MASS to finish writing before changing	54	0.1	1	184	0:03:03
214	waiting to be scheduled	waiting on run queue after yield	50	1.7	34	184	0:03:03
171	waiting to output to the network	waiting for CTLIB event to complete	38	0.2	5	184	0:03:03
251	waiting to output to the network	waiting for network send to complete	18	0	0	184	0:03:03
41	waiting for internal system event	wait to acquire latch	8	0.2	25	184	0:03:03
272	waiting for internal system event	waiting for lock on ULC	7	1.2	171	184	0:03:03

SPIDS 859 & 1659

WaitEventID	WaitClass	WaitEvent	Waits	WaitTimeSec	msPerWait	delta_secs	delta_hms
250	waiting for input from the network	waiting for incoming network data	1572	163.3	103	277	0:04:04
283	waiting on another thread	Waiting for Log writer to complete	976	44.4	45	277	0:04:04
150	waiting to take a lock	waiting for a lock	367	22.8	62	277	0:04:04
29	waiting for a disk read to complete	waiting for regular buffer read to complete	163	1.5	9	277	0:04:04
36	waiting for memory or a buffer	waiting for MASS to finish writing before changing	141	0.2	1	277	0:04:04
214	waiting to be scheduled	waiting on run queue after yield	92	1.5	16	277	0:04:04
251	waiting to output to the network	waiting for network send to complete	59	0	0	277	0:04:04
WaitEventID	WaitClass	WaitEvent	Waits	WaitTimeSec	msPerWait	delta_secs	delta_hms
124	waiting for internal system event	wait for mass read to finish when getting page	7	0.1	14	277	0:04:04
250	waiting for input from the network	waiting for incoming network data	646	71.3	110	167	0:02:02
283	waiting on another thread	Waiting for Log writer to complete	464	31.1	67	167	0:02:02
150	waiting to take a lock	waiting for a lock	239	15.4	64	167	0:02:02
29	waiting for a disk read to complete	waiting for regular buffer read to complete	128	6.1	47	167	0:02:02
36	waiting for memory or a buffer	waiting for MASS to finish writing before changing	75	0.2	2	167	0:02:02
214	waiting to be scheduled	waiting on run queue after yield	33	0	0	167	0:02:02
251	waiting to output to the network	waiting for network send to complete	32	0	0	167	0:02:02
272	waiting for internal system event	waiting for lock on ULC	7	0.3	42	167	0:02:02
41	waiting for internal system event	wait to acquire latch	2	0.2	100	167	0:02:02
52	waiting for a disk write to complete	waiting for i/o on MASS initated by another task	0	0	0	167	0:02:02

COMMENTS ON WAIT EVENTS

Summarizing

- **WaitEvent 250 is 'Awaiting Command'**
  - ✓ Ignorable
- **In all 4 cases, the primary wait causes were:**
  - ✓ Waiting for the log writer thread (70% of the issue)
    - Due to log device speed
  - ✓ Lock contention (25% of the issue)
    - Lockwait totals from 10:29→10:32 and 11:42→11:47
      - Sequence_table1 103
      - KeyTxnTable 3
      - Sequence_table2 2
  - ✓ Physical Disk Read (<5%)
  - ✓ All others less than 1%

SUMMARY OF INVESTIGATION

What is Wrong with Stored_Proc_G06177

- **A few bad queries to investigate**
  - ✓ Lines 7351 and 1012
- **Log device seems slow**
  - ✓ Either natively (and it was)
  - ✓ …or Async Log Service enabled with too few engines (likely)
    - Was enabled due to log semaphore contention - which was more likely caused by the slow log device
    - While ALS eliminated the log semaphore contention, it merely masked the issue by hiding the waits in areas sp_sysmon doesn't show (MDA does)
- **Overall**
  - ✓ We have 6 other query lines to look at in 4 other stored procs
    - Though-out the two weeks, we really only noticed 10-12 proc lines that needed to be looked at for optimization
    - Point being - a small amount of effort by a few developers could have potentially huge performance improvements
  - ✓ ….but fixing the log device is key to a lot more than this
  - ✓ Lack of business monitoring and detailed application monitoring led DBA staff and developer's an a 6 months- 1 year long debacle due to application workload changes

SUMMARY FOR TODAY

Key Points to Remember

- **Business Level Monitoring is Key**
- **Application Level Monitoring**
  - ✓ Start by baselining resources and workload
    - Server wide as well as for top 3-4 key applications
    - Cpu, LIO's, PIO's, tempdb, transactions
  - ✓ Key tables
    - monEngine, monProcess, monProcessActivity
    - monOpenObjectActivity, monSysStatement, monCachedStatements
  - ✓ Supporting tables
    - monProcessWaits, monLocks, monDeviceIO
- **Proper Analysis requires a repository DB & stored procs**
  - ✓ Don't expect answers to jump out at you - **thinking is required!!!**


SYBASE®
An SAP Company