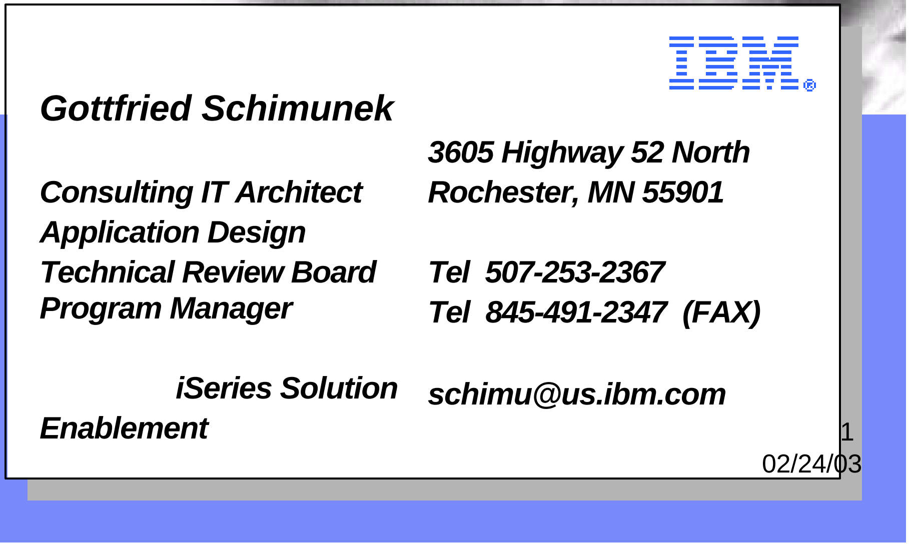
Session: SP01

# Performance analysis tools for WebSphere applications on iSeries

Gottfried Schimunek



**Gottfried Schimunek**

*Consulting IT Architect*
*Application Design*
*Technical Review Board*
*Program Manager*

*iSeries Solution Enablement*

*3605 Highway 52 North*
*Rochester, MN 55901*

*Tel 507-253-2367*
*Tel 845-491-2347 (FAX)*

*schimu@us.ibm.com*

This publication may refer to products that are not currently available in your country. IBM makes no commitment to make available any products referred to herein.

ITSO iSeries Technical Forum

# Acknowledgement

This presentation was produced with the help of Eric Barsness of the IBM WebSphere development lab in Rochester,MN

# Agenda

- **iSeries V5R2 Java™ Performance**
- Performance analysis tools
- **Key Tuning Tips and Techniques**
  - WebSphere
  - Java
  - Java Database Connectivity (JDBC)
- Resolving Common WebSphere and Java Problems

# *iSeries V5R2 Java Performance*

# V5R2 Java Performance

## *What's New?*

**Misc. Java/JVM improvements**

**Improvements to Java commands**

**Just In Time (JIT) compiler performance improvements**

**Support for JVM Profiler Interface (JVMPI)**

**Performance Explorer (PEX) changes for Java**

**POWER4 hardware allows greater scalability**

# Misc. Java/JVM Improvements

**JDK 1.4 support added (1.1.6 and 1.1.7 removed)**

**Java locking improvements**

- Lock deflation
- Pathlength improvements

**Object allocation pathlength improvements**

**Garbage Collector enhancements**

- Heap compaction allows the Java heap to shrink in size

**User classloader cache**

- Avoids bytecode verification if class already loaded by User classloader
- Avoids recreation of JVAPGMs, when class has already been loaded
- The os400.define.class.cache.file property enables this option

# Java Command Improvements

## CRTJVAPGM

- now multithreaded for quicker Direct Execution compiles

## ANZJVM

- New for V5R2
- Can be used to help resolve object leaks

# The JIT Compiler has become...

## Stronger

- As a general rule of thumb, JIT outperforms Direct Execution  by 15% in V5R2, once the system "warms up"

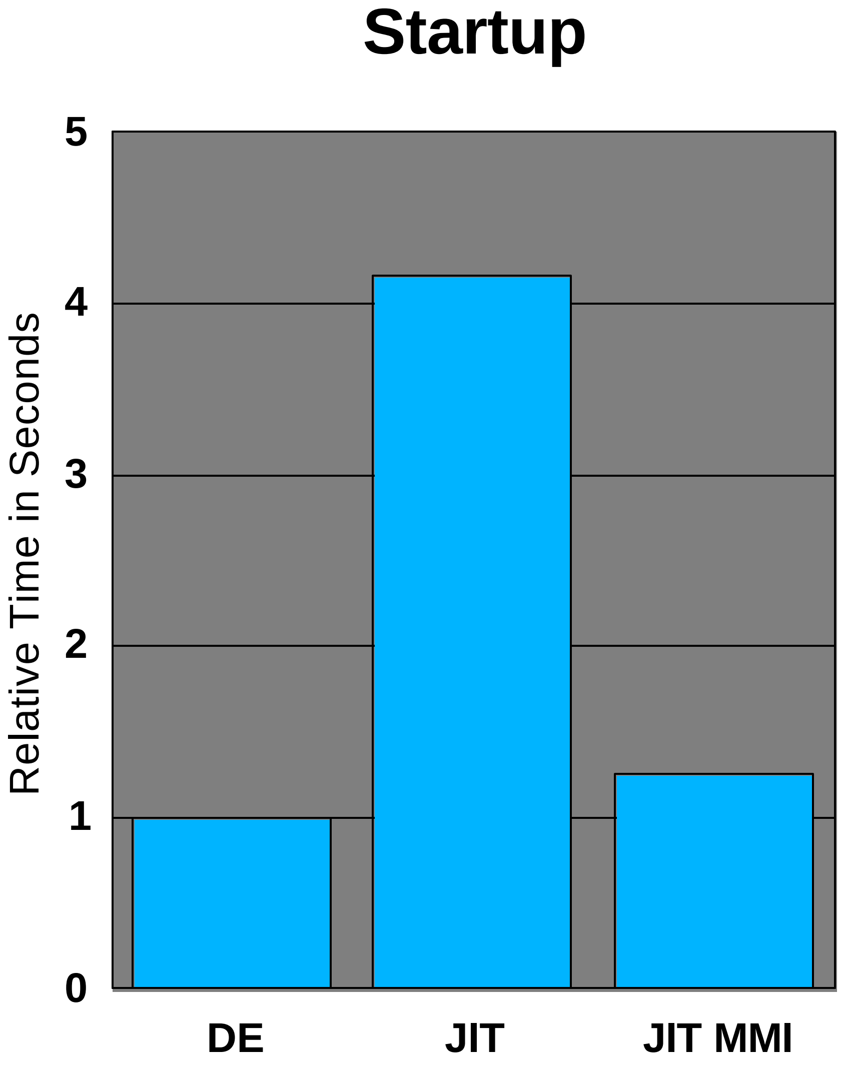
## Smarter

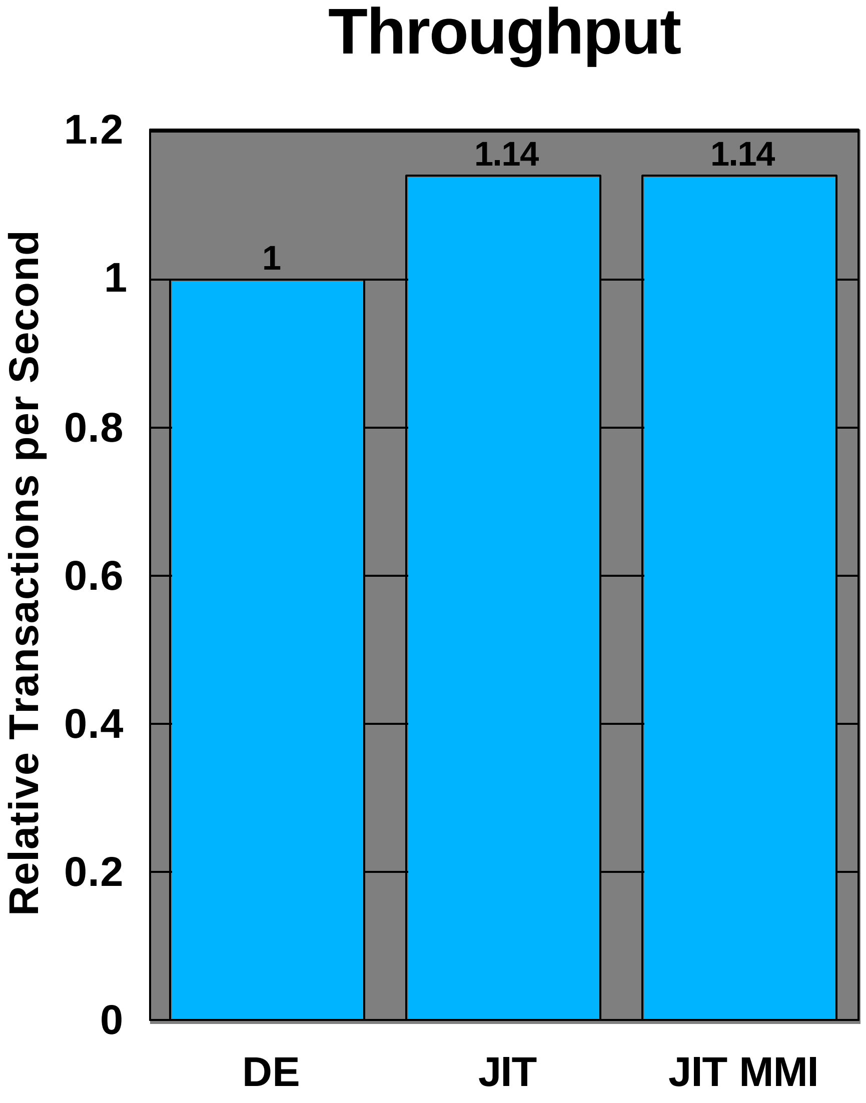
- Register allocation looks at entire method to minimize load and store operations.
- Model dependent code generation
  - Resulting JIT code will be different for a POWER4 iSeries then a sStar iSeries

## More flexible

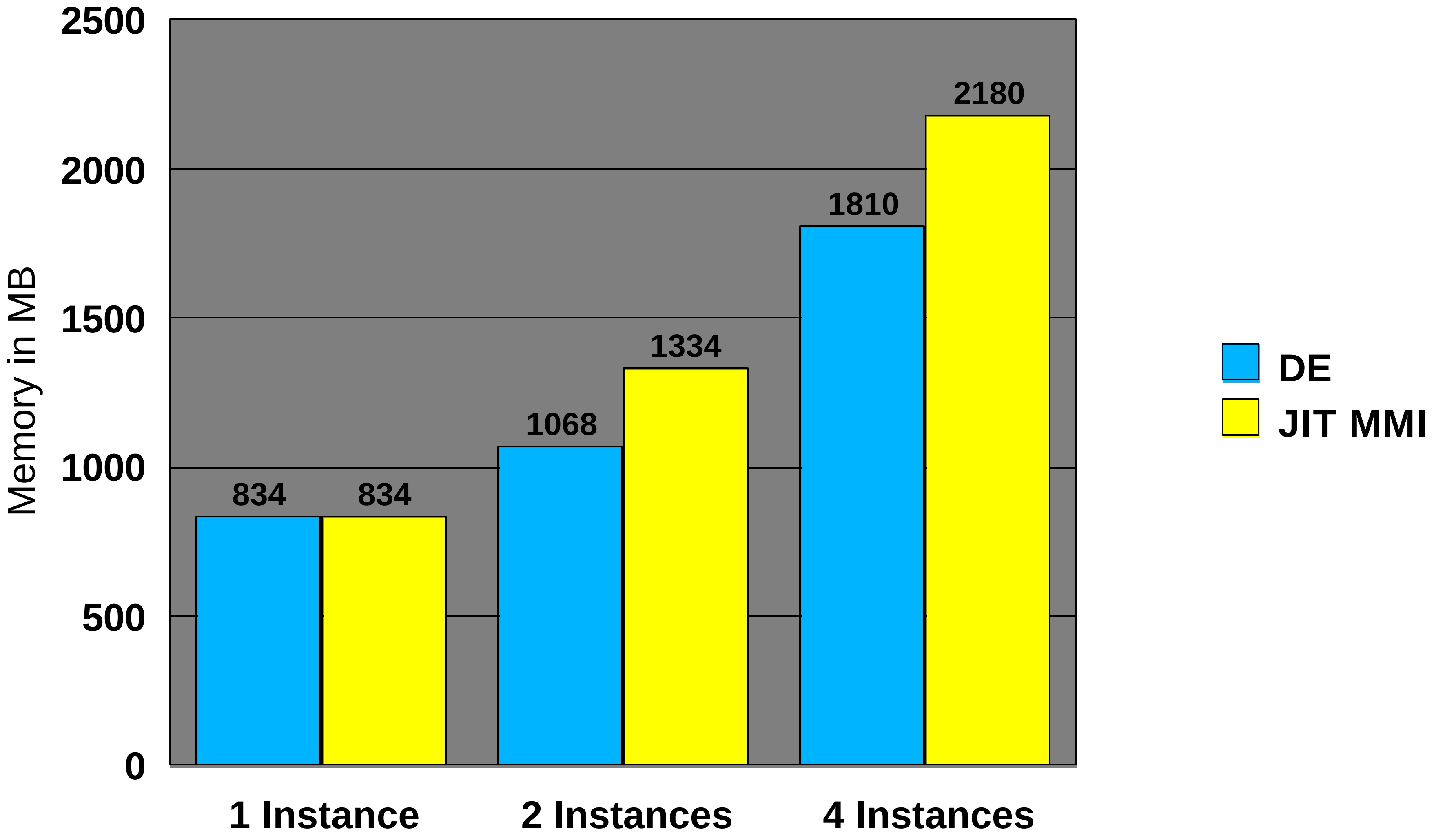
- A new execution mode, Mixed Mode Interpreter (MMI) is introduced
  - Quicker startup time then standard JIT (but still worse than fully DE'd code)
    - Java class is interpreted until the JVM determines that it is a common path within the code, then will run the JIT
    - Code executed only on startup will probably be interpreted
  - Can be tuned with the property **os400.jit.mmi.threshold**

# JIT Startup vs. Throughput (for a large Java application)





# JIT Memory Consumption (for a large Java application)



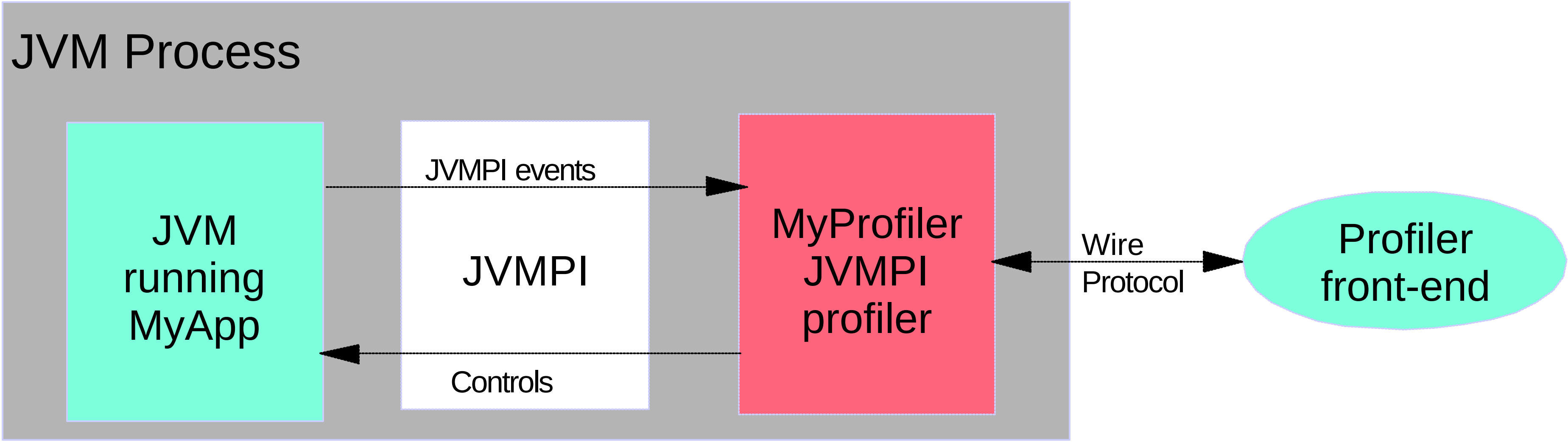
# JVMPI - Overview

## Java Virtual Machine Profiler Interface (JVMPI) is new for V5R2 on iSeries

- JVMPI by itself is not exciting to the end user
- JVMPI is an API that can be used to build profilers

## Instructions on how to invoke the profiler

- In QSH,  **java -XrunMyProfiler MyApp**, where MyProfiler is the profiler (J-Probe, Optimizeit, JProf, etc.
- From CL, **JAVA CLASS(MyApp) PROP((os400.xrun.option MyProfiler))**



# JVMPI - Profile Agent sends controls to JVM

**The profile agent sends a request to the JVM through a JVMPI defined interface**

- The profile agent wants to be notified asynchronously when an event happens
  - EnableEvent(), DisableEvent(), RequestEvent(), etc
- The profile agent wants the JVM to execute something
  - DisableGC(), EnableGC(), RunGC(), etc
- The profile agent wants the JVM to perform an action
  - SuspendThread(), ResumeThread(), etc
- The profile agent wants information about the JVM
  - GetCurrentThreadCpuTime(), GetThreadStatus(), GetThreadLocalStorage, etc

# JVMPI - JVM sends events to the profile agent

## A list of events that the JVM can send to the profile agent

- method enter and exit
- object alloc, move, and free
- heap arena create and delete
- GC start and finish
- JNI global reference alloc and free
- JNI weak global reference alloc and free
- compiled method load and unload
- thread start and end
- class file data ready for instrumentation
- class load and unload
- contended Java monitor wait to enter , entered, and exit
- contended raw monitor wait to enter, entered, and exit
- Java monitor wait and waited
- monitor dump
- heap dump
- object dump
- request to dump or reset profiling data
- Java virtual machine initialization and shutdown

# Performance Explorer (PEX) Changes

## Database tables have been changed

- Some old queries will need to be rewritten (if you run them manually)
- Must use new PTDV to access V5R2 data

## New or updated event types

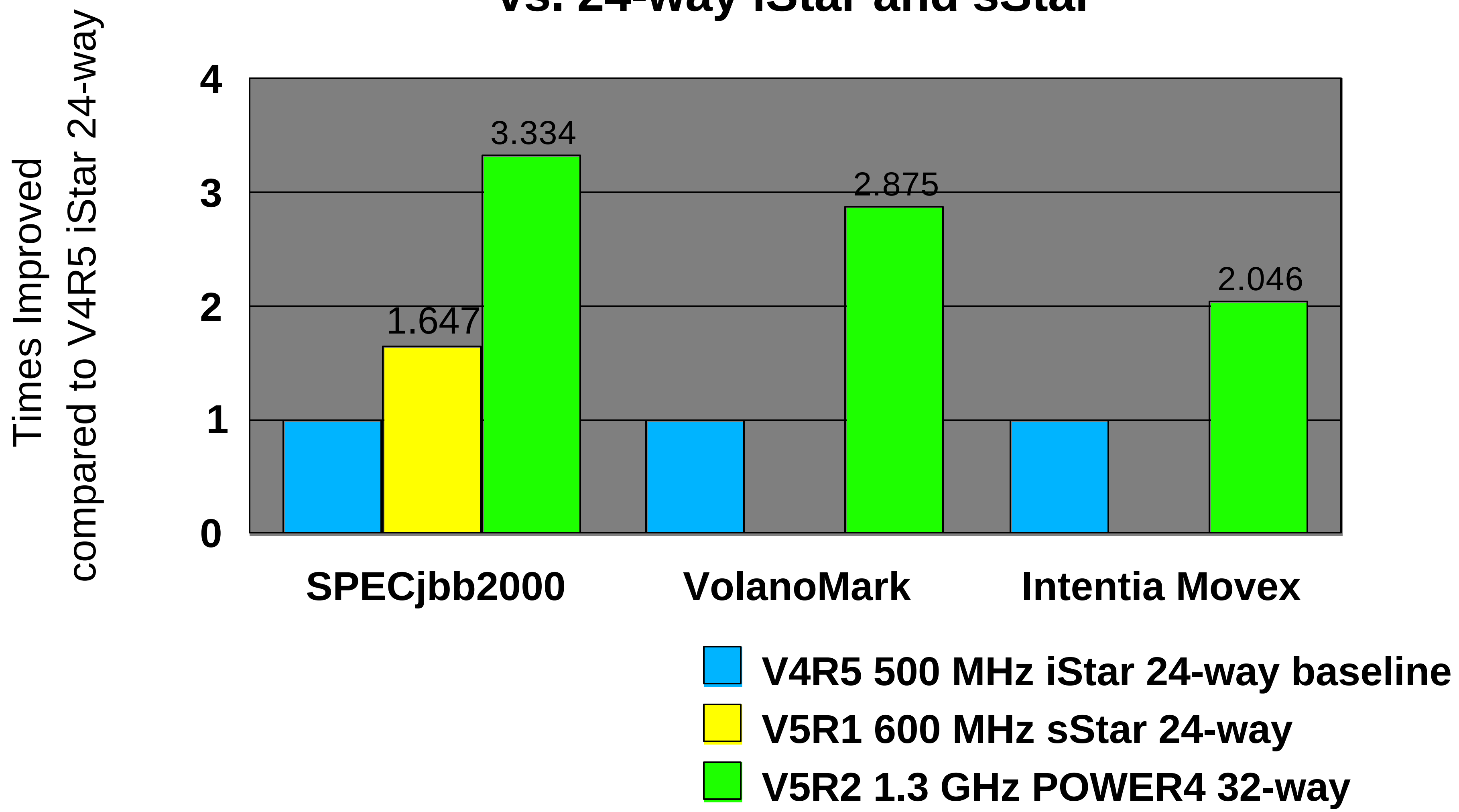
- Native JDBC <-> CLI events added (*DBSVRCNN, *DBSRVREQ)
- Lock / unlock events use *LCKSTR and *UNLCK
- Thread create, start events report parent, child thread id
- Class load/unload events contain class loader and optimization level
- Wait/notify/notifyAll events contain class information

## Filtering support has been added for Java methods

## Many Java events now contain five levels of stack information

# POWER4 Hardware

## POWER4 32-way Scalability
vs. 24-way iStar and sStar



# *WebSphere and Java Related Performance Tools*

# Performance Tools

**verboseGC**

**DMPJVM**

**ANZJVM**

**Performance Explorer (PEX)**

**Performance Trace Data Visualizer (PTDV)**

**WebSphere Resource Analyzer**

# Java and WebSphere Related Performance Tools

## *verboseGC*

**Simple way to monitor GC behavior, check for object leaks**

**Enable with -verboseGC option on Java command line, or via WAS GUI**

**Example output**

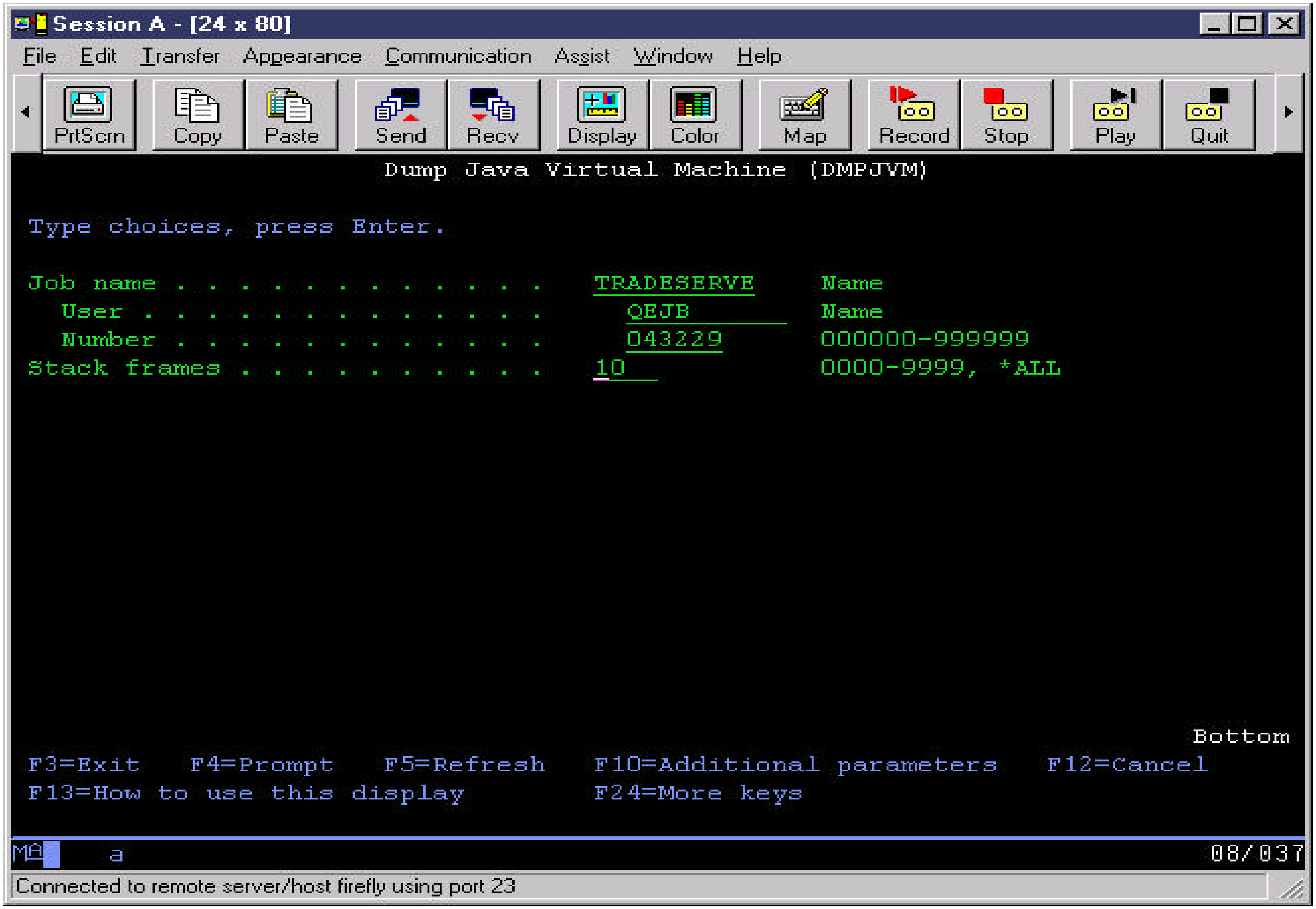
```
GC 5: starting collection, threshold allocation reached.
GC 5: live objects 31739457; collected objects 33663346; collected(KB) 4177772.
GC 5: queued for finalization 0; total soft references 622; cleared soft references 5.
GC 5: current heap(KB) 9066464; current threshold(KB) 2097152.
GC 5: collect (milliseconds) 9232.
GC 5: current cycle allocation(KB) 950219; previous cycle allocation(KB) 4194338.
GC 5: total weak references 3987; cleared weak references 0.
GC 5: total final references 118763; cleared final references 2267.
GC 5: total phantom references 0; cleared phantom references 0.
GC 5: total old soft references 0; cleared old soft references 0.
GC 5: total JNI global weak references 0; cleared JNI global weak references 0.
```

# DMPJVM

**DMPJVM (Dump Java Virtual Machine) is a standard OS/400 command**

- Dump information about the JVM for a specified job
  - The classpath
  - Heap information
  - Garbage collection information
  - Thread information
  - Class loader list
  - Current Object list
- Use DMPJVM to debug problems "on the fly"
  - Real time lock information to detect deadlocks
  - Garbage collection statistics to detect memory leaks
  - Simple debug
- May need to do **CHGJOB JOB(*) DFTWAIT(300)** before running DMPJVM on busy Java jobs to avoid timeout.

# DMPJVM


Session A - [24 x 80]
File Edit Transfer Appearance Communication Assist Window Help
PrtScrn Copy Paste Send Recv Display Color Map Record Stop Play Quit

```
                   Dump Java Virtual Machine (DMPJVM)

Type choices, press Enter.

Job name . . . . . . . . . . . . .   TRADESERVE    Name
  User . . . . . . . . . . . . . .     QEJB        Name
  Number . . . . . . . . . . . . .     043229      000000-999999
Stack frames . . . . . . . . . . .   10            0000-9999, *ALL

                                                                         Bottom
F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display        F24=More keys
MA        a                                                            08/037
```

Connected to remote server/host firefly using port 23

# DMPJVM - Spool File Data

## Dump information about the JVM for a specified job

- The JDK level:  java.version=1.3
- The classpath used for the JVM
- Heap information
  - Garbage collector parameters
    - Initial size: 1048576 K
    - Max size: 240000000 K
  - Current values
    - Heap size: 5645408 K
    - JIT heap size: 494896 K
    - JVM heap size: 716600 K
  - Garbage collection information
    - Garbage collections: 427
    - Last GC cycle time: 779 ms

# DMPJVM - Spool File Data (continued)

- Thread information
  - Thread:  00000002 Thread-0
  - TDE:  B004300003235000
  - Thread priority: 5
  - Thread status: Waiting
  - Wait object: com/ibm/ejs/sm/server/ManagedServer
  - Thread group: main
  - Runnable: java/lang/Thread
  - Stack:
    - java/lang/Object.wait()V+1 (Object.java:420)
    - com/ibm/ws/runtime/Server.awaitShutdown()V+35 (Server.java:1687)
    - com/ibm/ejs/sm/server/ManagedServer.main([Ljava/lang/String;)V+38 (ManagedServer.java:172)
    - com/ibm/ws/bootstrap/WSLauncher.main([Ljava/lang/String;)V+713 (WSLauncher.java:158)
    - com/ibm/ws/bootstrap/WSLauncher.main([Ljava/lang/String;)V+713 (WSLauncher.java:158)
  - Locks:
    - None

# DMPJVM - Spool File Data (continued)

- Class loader list / Current Object list

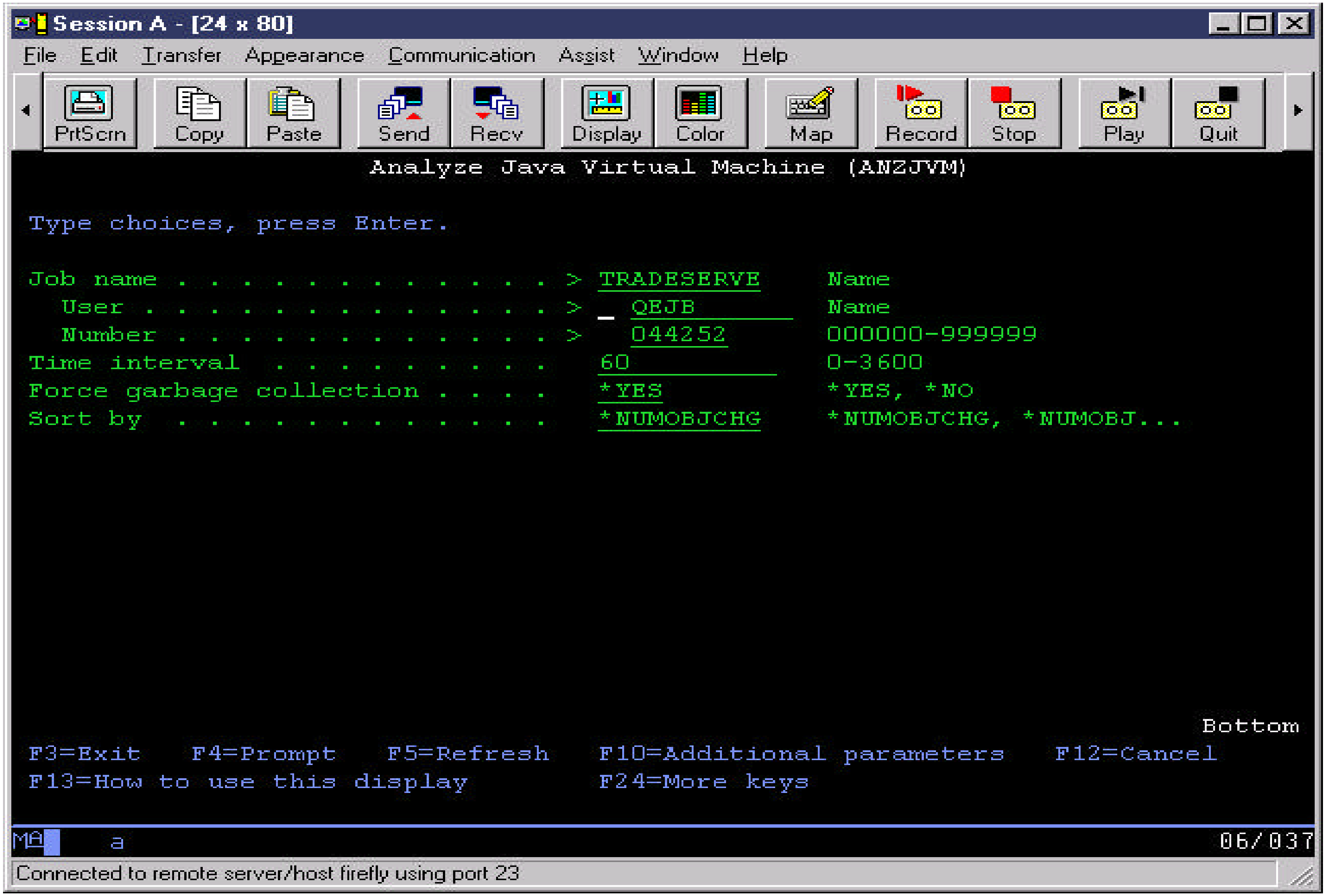
```
0 Default class loader
Loader     Objects      Class name
------     -------      ----------
0          15623710     [C
0          756654       java/util/Vector
0          318020       java/util/Stack
0          14293745     java/lang/String
0          635734       [Ljava/util/Hashtable$Entry;
0          1192987      [Ljava/lang/Object;
0          498741       java/util/Hashtable
0          5            java/lang/ThreadGroup
0          6088         java/lang/Class
```

# ANZJVM

**ANZJVM (Analyze Java Virtual Machine) is a standard OS/400 command**

- New in V5R2
- Dumps a report diagnosing the differences in the JVM over specified amount of time
  - Can automatically run the garbage collector before each snapshot
  - Groups statistics by object type
  - Sorts the report by either the number of allocated objects difference (for a leak of a lot of small objects) or the size of the allocated objects difference(for a slow leak of large objects)
- Generates a spool file with results
- Use ANZJVM to debug problems on a live JVM
  - Its main use is to debug object leaks

# ANZJVM


Session A - [24 x 80]
File Edit Transfer Appearance Communication Assist Window Help
PrtScrn Copy Paste Send Recv Display Color Map Record Stop Play Quit

```
                     Analyze Java Virtual Machine (ANZJVM)

 Type choices, press Enter.

 Job name . . . . . . . . . . . . > TRADESERVE    Name
   User . . . . . . . . . . . . . > _ QEJB        Name
   Number . . . . . . . . . . . . >   044252      000000-999999
 Time interval  . . . . . . . . .   60            0-3600
 Force garbage collection . . . .   *YES          *YES, *NO
 Sort by  . . . . . . . . . . . .   *NUMOBJCHG    *NUMOBJCHG, *NUMOBJ...

                                                                         Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
 F13=How to use this display        F24=More keys

MA     a                                                               06/037
Connected to remote server/host firefly using port 23
```

# ANZJVM - Spool File Data (subset)

```
Mon Sep 23 13:42:09  2002
Job: 044252/QEJB/TRADESERVE
Interval: 60 (SEC)
Total garbage collection cycles prior to running: 0
Total garbage collection cycles after running: 0
GC forced: YES
TIME OF FORCED GC: Wed Dec 31 17:59:59  1969
TIME OF FORCED GC: Wed Dec 31 17:59:59  1969
.........................................................................
. Class loader information                                              .
.........................................................................
0 Default class loader
1 com/ibm/ws/classloader/CompoundClassLoader
2 com/ibm/ws/bootstrap/ExtClassLoader
3 sun/misc/Launcher$AppClassLoader
4 sun/misc/Launcher$ExtClassLoader
.........................................................................
. GC heap information                                                   .
.........................................................................
Loader
|   Number of pass one objects in the GC heap
|   |         Number of pass two objects in the GC heap
|   |         |         Change in the number of objects in the GC heap
|   |         |         |         Pass one object size (K)
|   |         |         |         |       Pass two object size (K)
|   |         |         |         |       |       Change in object size (K)
|   |         |         |         |       |       |       In global registry
|   |         |         |         |       |       |       |   Class name
0   592340    593326    986       26062   26106   44      NO  java/lang/String
0   1042837   1043791   954       147246  147303  57      YES [C
0   341041    341269    228       13641   13650   9       NO  java/lang/StringBuffer
0   22457     22561     104       28060   28110   50      YES [B
2   145       248       103       7       12      5       NO  com/ibm/ejs/util/am/_Alar
```

# Performance Explorer (PEX) and Java

**The two most important uses of PEX with Java are**

- Collecting Trace Profile data to determine where CPU time is spent
- Collect Java specific event data

**PEX is part of OS/400**

**Performance Tools/400 (PT1) provides a capability to print reports for review and manual interpretation.  Required for creating Trace Profile report.**

**IBM makes a no-charge tool available at AlphaWorks named PTDV (Performance Trace Data Visualizer). PTDV can analyze and visualize Java specific events in a PEX trace.**

- http://www.alphaworks.ibm.com/tech/ptdv

# PEX Trace Profile

## To create PEX definition for Trace Profile:

- pre-V5R2: ADDPEXDFN DFN(TPROF5) TYPE(*TRACE) JOB(*ALL) TASK(*ALL) MAXSTG(100000) INTERVAL(5) TRCTYPE(*SLTEVT) SLTEVT(*YES) BASEVT((*PMCO))
- V5R2: ADDPEXDFN DFN(TPROF5) TYPE(*PROFILE) PRFTYPE(*JOB) JOB(*ALL) TASK(*ALL) MAXSTG(100000) INTERVAL(5)

## To collect data:

- STRPEX SSNID(mytprof) DFN(TPROF5)
- ENDPEX SSNID(*SELECT) - wait until you have around 100,000 or more events

## To create the report:

- PRTPEXRPT MBR(mytprof) TYPE(*PROFILE) PROFILEOPT(*SAMPLECOUNT *PROCEDURE)

# PEX Trace Profile - example output (subset)

```
                                                    Performance Explorer Report                                    9/23/02 15:22:3
                                                        Profile Information                                               Page   5
Library . . : QPEXSPEC
Member. . . : SJAS0908A
Description : *BLANK
                           Histogram     Hit   Hit   Cum   Start             Map    Stmt   Name
                                         Cnt     %     %   Addr              Flag   Nbr
                                                                                                                                 -
                                  **     4927   3.5   3.5 FFFFFFFFFE9D9BE0   ++    0001E0  JAVADEEP/javaxresolveinterfacebla
                                   *     3060   2.2   5.7 FFFFFFFFB39D3250   ++    005E30  JAVAGC/sweepReuseSegment__27JavaThreadS
                                                                                           stemGCCollectorFP6JavaVMUlT2Uc
                                   *     2881   2.1   7.8 FFFFFFFFFE9C75F0   ++    000030  JAVABLA/markOldNewGCStore__FP10JavaObje
                                                                                           tPP10JavaObjectUc
                                         2159   1.6   9.4 FFFFFFFFB17539B0   ++    002650  QUMUGA/pSpinWait__11QuMutexGateFv
                                         1339   1.0  10.3 FFFFFFFFB39CDB04   ++    0006E4  JAVAGC/markGrayCollector__6JavaGCFP10Ja
                                                                                           aObjectP10JavaThread
                                         1313   0.9  11.3 FFFFFFFFFEAEDED0   ++    0007C0  JVAOBJLK/javalockmonitorenterweak
                                         1231   0.9  12.2 17A377697F01CD10   ==         0  QSQROUTX/XTPROCES
```

# Java Related PEX Trace Events

## PEX will create a record for particular events

- Method entry/exit
- Java Object Creates and Deletes
- Java Locks
- WebSphere Events
- etc.

## Some events (method entry/exit) require special "hooks"

- If running Direct Execution, hooks can automatically be inserted, with the ENBPFRCOL parameter on the CRTJVAPGM command
- If running JIT, hooks can automatically be inserted with the Java property os400.enbpfrcol set to '1'.

## Can generate large amounts of data

# Performance Explorer Filtering - Overview

**New in V5R2**

**Filtering allows you to reduce the amount of PEX data collected**

**For Java, you can reduce the Java method entry/exit events in the PEX collection**

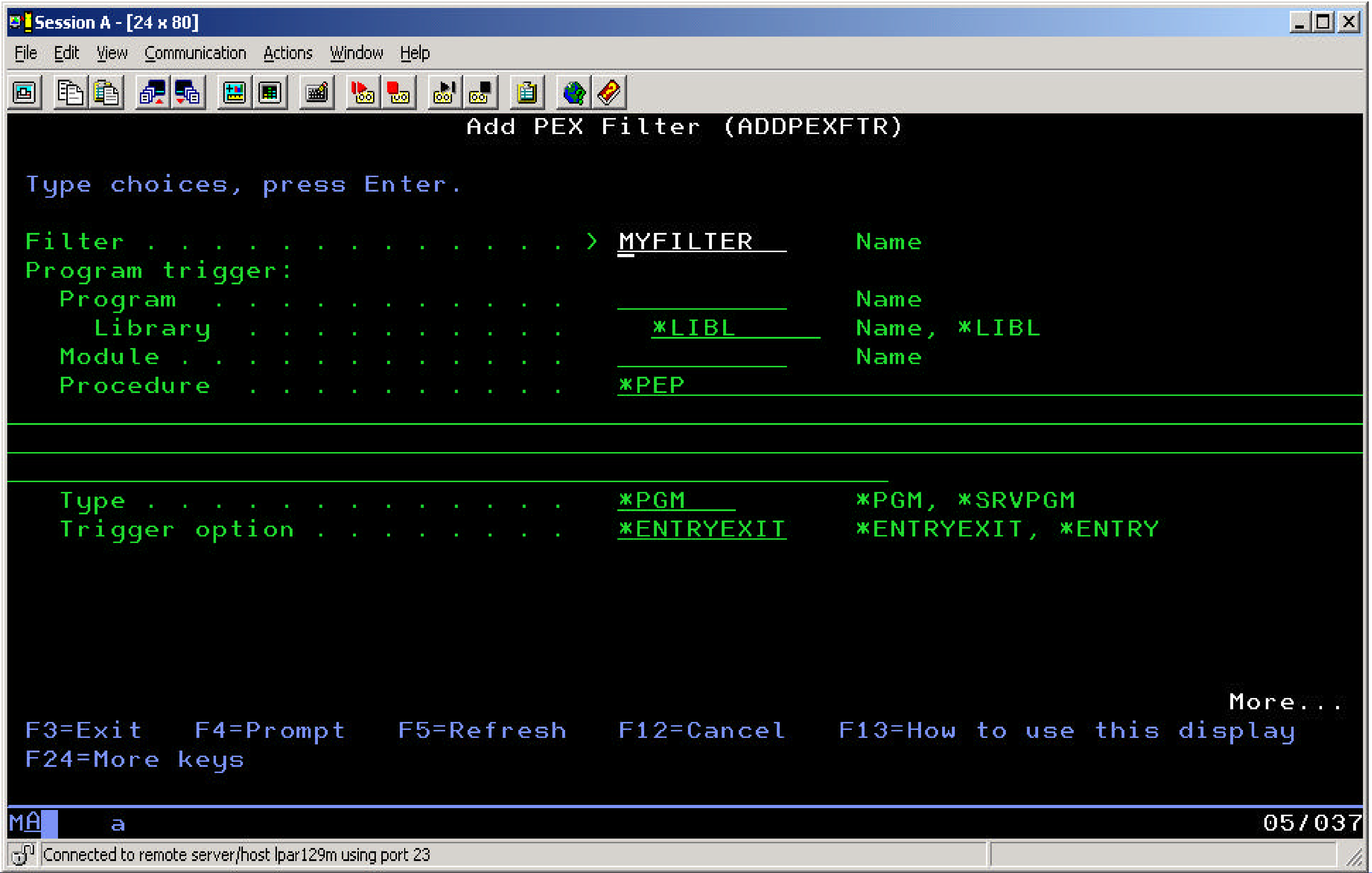
- Methods from one or more packages
- Methods from one or more classes
- Individual methods

**Simplifies analysis by only containing information relevant to the Java methods in question**

- or can exclude methods

**Triggers not available for Java at this time**

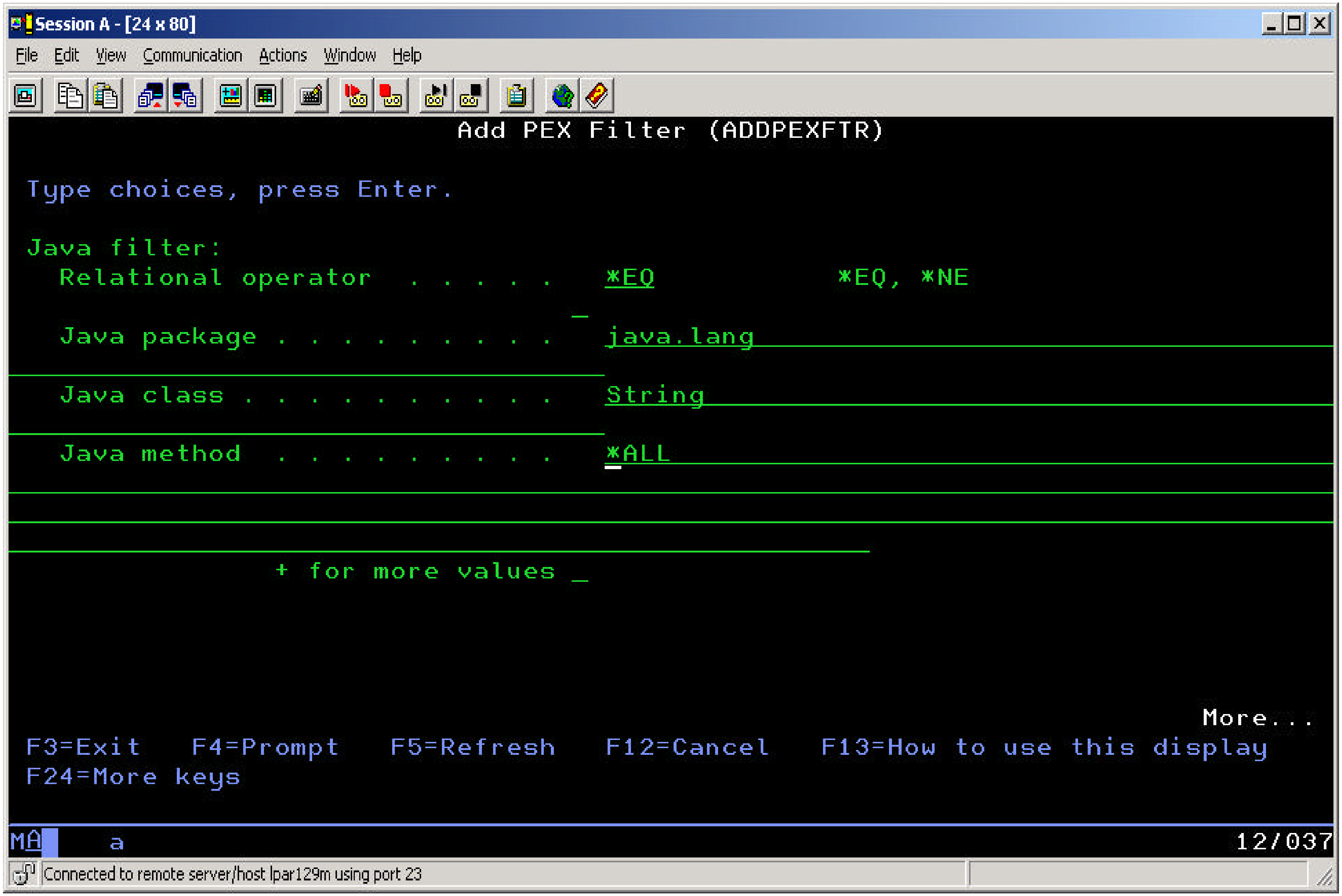
# Performance Explorer Filtering - ADDPEXFTR



```
                         Add PEX Filter (ADDPEXFTR)

 Type choices, press Enter.

 Filter . . . . . . . . . . . . . > MYFILTER      Name
 Program trigger:
   Program  . . . . . . . . . . .                 Name
     Library  . . . . . . . . . .     *LIBL       Name, *LIBL
   Module . . . . . . . . . . . .                 Name
   Procedure  . . . . . . . . . .   *PEP



   Type . . . . . . . . . . . . .   *PGM          *PGM, *SRVPGM
   Trigger option . . . . . . . .   *ENTRYEXIT    *ENTRYEXIT, *ENTRY








                                                                         More...
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys

MA     a                                                                 05/037
```

Connected to remote server/host lpar129m using port 23

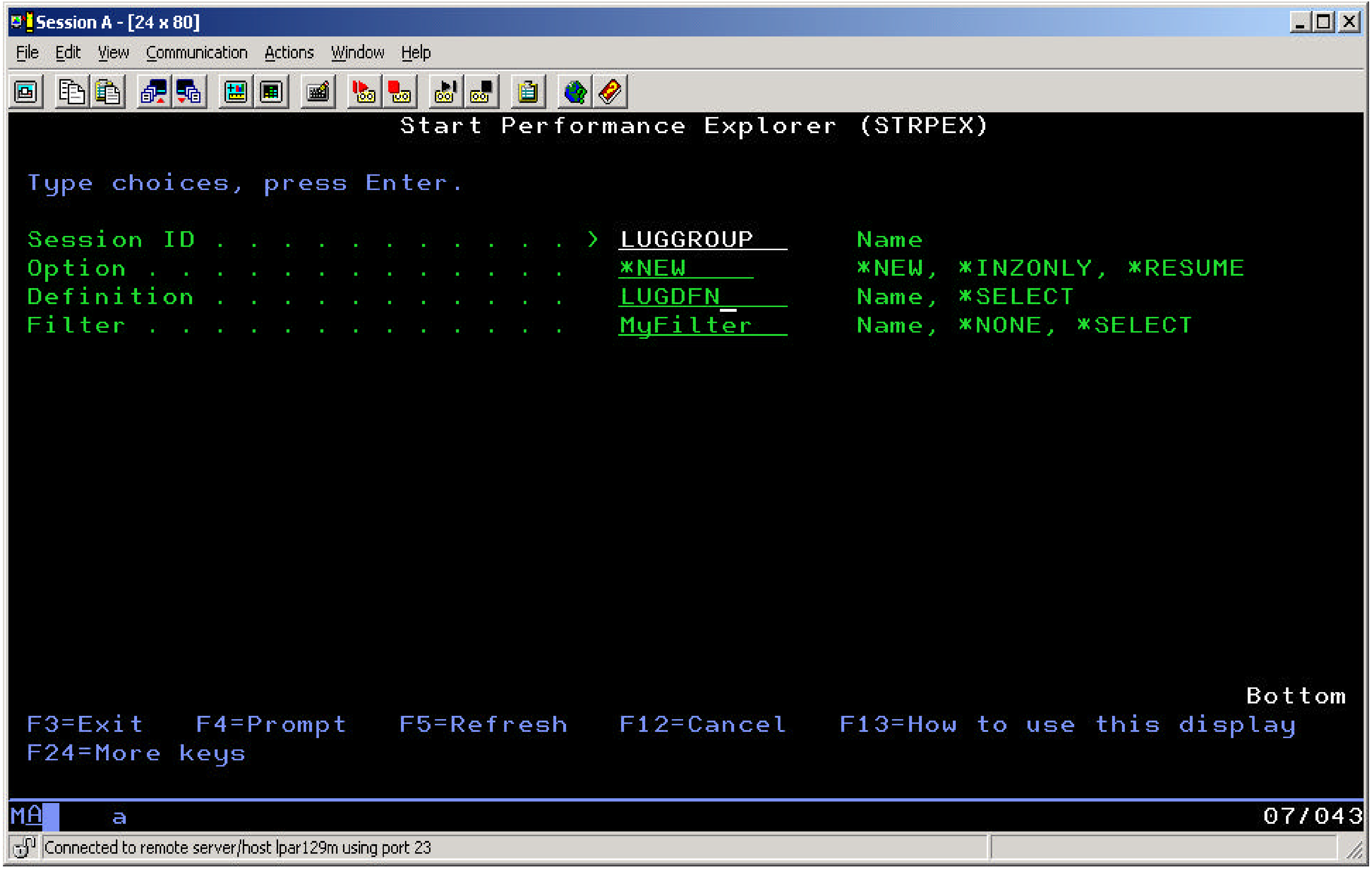
# Performance Explorer Filtering - ADDPEXFTR


Session A - [24 x 80]
File Edit View Communication Actions Window Help

```
                          Add PEX Filter (ADDPEXFTR)

 Type choices, press Enter.

 Java filter:
   Relational operator  . . . . .   *EQ           *EQ, *NE
                                  _
   Java package . . . . . . . . .   java.lang

   Java class . . . . . . . . . .   String

   Java method  . . . . . . . . .   *ALL



                + for more values _




                                                                       More...
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys

MA      a                                                             12/037
```

Connected to remote server/host lpar129m using port 23

# Performance Explorer Filtering - STRPEX



```
                       Start Performance Explorer (STRPEX)

 Type choices, press Enter.

 Session ID . . . . . . . . . . . > LUGGROUP      Name
 Option . . . . . . . . . . . . .   *NEW          *NEW, *INZONLY, *RESUME
 Definition . . . . . . . . . . .   LUGDFN        Name, *SELECT
 Filter . . . . . . . . . . . . .   MyFilter      Name, *NONE, *SELECT

                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
MA     a                                                               07/043
```

Connected to remote server/host lpar129m using port 23

# Five levels of stack in Java PEX data

```
SELECT count(*), varchar(n.qjvnam, 30), P.QPRPNM FROM qaypetidx i,
qaypejva j, qaypejvci c, qaypejvni n, qaypeproci p WHERE i.qrecn =
j.qrecn and j.qjvpca = c.QJVCKY and C.QJVCNI = N.QJVNMO and
j.QJVSkey3 = p.qprkey and varchar(n.qjvnam, 512) like '%TimeZone%'
GROUP BY varchar(n.qjvnam, 30), p.qprpnm
```

```
  COUNT ( * )   VARCHAR                         Procedure name
          148   java/util/SimpleTimeZone        java-util-Calendar-getInstance()Ljava-util-Calendar;
          363   java/util/SimpleTimeZone        LE_Create_Thread2__FP12crtth_parm_t
********  End of data  ********
  COUNT ( * )   VARCHAR                         Procedure name
          148   java/util/SimpleTimeZone        java-util-GregorianCalendar-<init>()V
          363   java/util/SimpleTimeZone        startThread__FPv
********  End of data  ********
  COUNT ( * )   VARCHAR                         Procedure name
          363   java/util/SimpleTimeZone        #cfmir
          148   java/util/SimpleTimeZone        java-util-TimeZone-getDefault()Ljava-util-TimeZone;
********  End of data  ********
  COUNT ( * )   VARCHAR                         Procedure name
          148   java/util/SimpleTimeZone        java-util-SimpleTimeZone-clone()Ljava-lang-Object;
          363   java/util/SimpleTimeZone        javaattachthread
********  End of data  ********
  COUNT ( * )   VARCHAR                         Procedure name
          148   java/util/SimpleTimeZone        javadetointerpreter
          363   java/util/SimpleTimeZone        startThread__27JavaThreadSystemGCCollectorFv
********  End of data  ********
```

# Performance Trace Data Visualizer

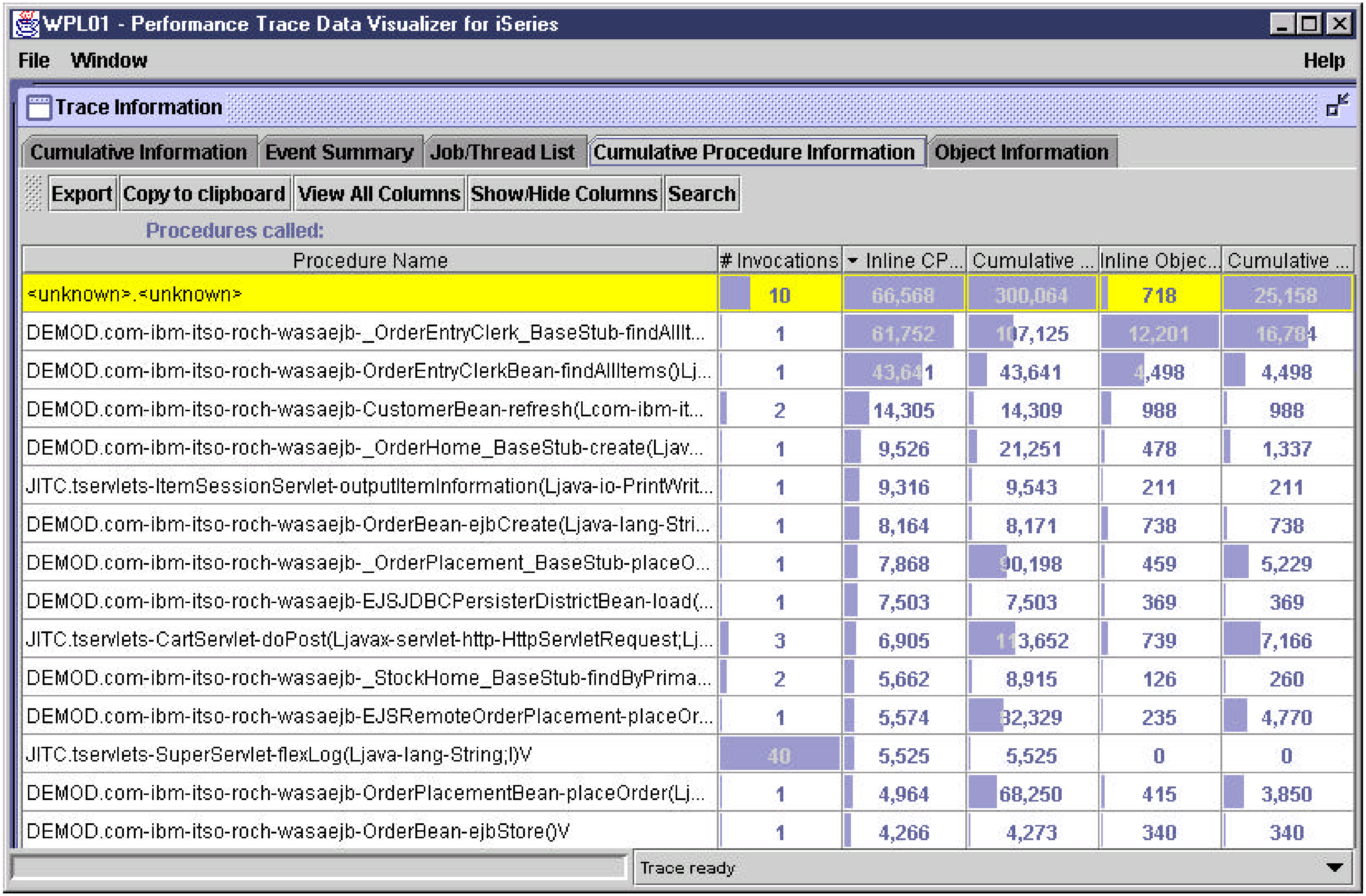
## Performance Trace Data Visualizer

- Tool for visualizing PEX trace data
- Designed for working with Java programs, but will work for all ILE languages
- Runs in client-server mode, with data and logic residing on the iSeries and presentation on the PC
- Originally an internal Java performance team tool, but is now externally available with limited support
  - http://www.alphaworks.ibm.com/tech/ptdv

## Used for Low-level analysis of problems:

- Path length
- Java Object leaks
- Excessive exception handling
- Excessive Java locking
- Functional problems/program understanding

# PTDV Screen Shot



Procedures called:

| Procedure Name | # Invocations | Inline CP... | Cumulative ... | Inline Objec... | Cumulative ... |
|---|---|---|---|---|---|
| <unknown>.<unknown> | 10 | 66,568 | 300,064 | 718 | 25,158 |
| DEMOD.com-ibm-itso-roch-wasaejb-_OrderEntryClerk_BaseStub-findAllIt... | 1 | 61,752 | 107,125 | 12,201 | 16,784 |
| DEMOD.com-ibm-itso-roch-wasaejb-OrderEntryClerkBean-findAllItems()Lj... | 1 | 43,641 | 43,641 | 4,498 | 4,498 |
| DEMOD.com-ibm-itso-roch-wasaejb-CustomerBean-refresh(Lcom-ibm-it... | 2 | 14,305 | 14,309 | 988 | 988 |
| DEMOD.com-ibm-itso-roch-wasaejb-_OrderHome_BaseStub-create(Ljav... | 1 | 9,526 | 21,251 | 478 | 1,337 |
| JITC.tservlets-ItemSessionServlet-outputItemInformation(Ljava-io-PrintWrit... | 1 | 9,316 | 9,543 | 211 | 211 |
| DEMOD.com-ibm-itso-roch-wasaejb-OrderBean-ejbCreate(Ljava-lang-Stri... | 1 | 8,164 | 8,171 | 738 | 738 |
| DEMOD.com-ibm-itso-roch-wasaejb-_OrderPlacement_BaseStub-placeO... | 1 | 7,868 | 90,198 | 459 | 5,229 |
| DEMOD.com-ibm-itso-roch-wasaejb-EJSJDBCPersisterDistrictBean-load(... | 1 | 7,503 | 7,503 | 369 | 369 |
| JITC.tservlets-CartServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Lj... | 3 | 6,905 | 113,652 | 739 | 7,166 |
| DEMOD.com-ibm-itso-roch-wasaejb-_StockHome_BaseStub-findByPrima... | 2 | 5,662 | 8,915 | 126 | 260 |
| DEMOD.com-ibm-itso-roch-wasaejb-EJSRemoteOrderPlacement-placeOr... | 1 | 5,574 | 82,329 | 235 | 4,770 |
| JITC.tservlets-SuperServlet-flexLog(Ljava-lang-String;I)V | 40 | 5,525 | 5,525 | 0 | 0 |
| DEMOD.com-ibm-itso-roch-wasaejb-OrderPlacementBean-placeOrder(Lj... | 1 | 4,964 | 68,250 | 415 | 3,850 |
| DEMOD.com-ibm-itso-roch-wasaejb-OrderBean-ejbStore()V | 1 | 4,266 | 4,273 | 340 | 340 |

Trace ready

# PTDV Detailed data

**Allows you to view all running jobs and threads and see which are doing the most work**

**Shows you a call trace for each thread, and shows amount of time, and cycles used by each method call**

**Summarizes information at trace, job, thread, and method level**

**Detailed information on objects -- e.g. number of creates, locking behavior, lifetime**

# Resource Analyzer - Overview

**Resource Analyzer is a GUI performance monitor for WebSphere Application Server**

- Available for WebSphere Application Server, Advanced Edition Version 4.0.x

**Once the user turns on data collection**

- Data is collected continuously by the application server
- The data is retrieved, as needed, by Resource Analyzer

**The Resource Analyzer provides access to a wide range of performance data for two kinds of resources**

- Application resources (for example, enterprise beans and Servlets)
- WebSphere run-time resources (for example, Java Virtual Machine (JVM) memory, application server thread pools, and database connection pools)

# Resource Analyzer is used for the following analysis:

- Monitor real-time performance, such as response times for servlet requests or enterprise bean methods
- Detect trends by analyzing logs of data over time
- Determine the efficiency of a configuration of resources (such as the amount of allocated memory, the size of database connection pools, and the size of a cache for enterprise bean objects)
- Gauge the load on application servers and the average wait time for clients

# Resource Analyzer can monitor the following information

**EJB information - Number of active beans, method statistics, cache and pool information**

**Database connection pools - Number of connections, average wait time, number of threads, number of times connection used**

**System (IMS) - Number of physical connections, Number of connection handles**

**JVM run time - Total memory available to JVM, amount of free memory**

**Servlet session manager - Total number of HTTP sessions, average time to perform request, average concurrent active HTTP sessions**

**Thread pools - Object Request Broker (ORB) pool, web container pools thread information**

**Transaction manager - Average number of active transactions, duration of transactions, number of methods per transaction**

**Web applications - Number of loaded Servlets, average response time for requests, number of requests for the Servlet**

# Notes: The Analyzer collects and reports performance data for the following resource categories:

**Enterprise beans**. Data for this category reports load values, response times, and life cycle activities for enterprise beans. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. It reports information for enterprise bean methods, which are the remote interfaces used by an enterprise bean. Examples include the number of times a method was called and the average response time for the method. It also reports information on the size and usage of a cache of bean objects (enterprise bean object pools). Examples include the number of calls attempting to retrieve an object from a pool and the number of times an object was found available in the pool.

**Database connection pools**. Data for this category reports usage information about connection pools for a database. Examples are the average size of the connection pool (number of connections), the average number of threads waiting for a connection, the average wait time in milliseconds for a connection, and the average time the connection was in use.

**J2C Connectors**. Data for this category reports usage information about the J2EE (Java 2 Enterprise Edition) Connector Architecture that enables enterprise beans to connect and interact with procedural back-end systems, such as Customer Information Control System (CICS), and Information Management System (IMS). Examples are the number of managed connections (physical connections) and the total number of connections (connection handles).

**JVM run time**. Data for this category reports memory used by a process as reported by the JVM. Examples are the total memory available and the amount of free memory for the JVM.

JVMPI run time. In addition, the Resource Analyzer makes use of a Java Virtual Machine Profiler Interface (JVMPI) to enable a more comprehensive performance analysis. This profiling tool enables the collection of information about the Java Virtual Machine (JVM) that runs the application server. See Enabling JVMPI data reporting.

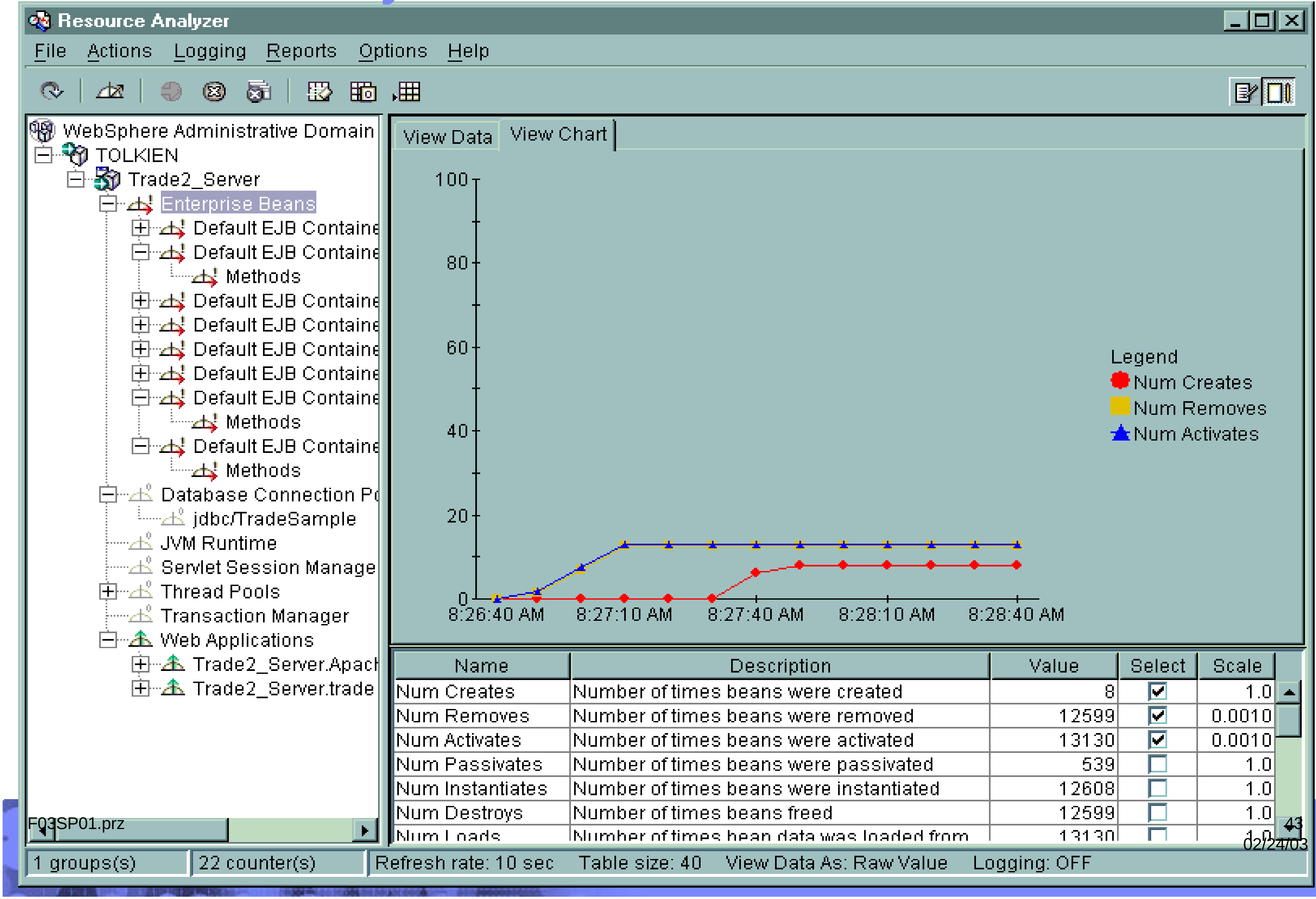
**Servlet session manager**. Data for this category reports usage information for HTTP sessions. Examples include the total number of sessions being accessed, the average amount of time it takes for a session to perform a request, and the average number of concurrently active HTTP sessions.

**Thread pools**. Data for this category reports information about the pool of Object Request Broker (ORB) threads that an application server uses to process remote methods and the Web container pools that are used to process HTTP requests coming into the application server. Examples include the number of threads created and destroyed, the maximum number of pooled threads allowed, and the average number of active threads in the pool.

**Transaction manager**. Data for this category reports transaction information for the container. Examples include the average number of active transactions, the average duration of transactions, and the average number of methods per transaction.

**Web applications**. Data for this category reports information for the selected server. Examples include the number of loaded servlets, the average response time for completed requests, and the number of requests for the servlet.

# Resource Analyzer - Screen Shot



| Name | Description | Value | Select | Scale |
|---|---|---|---|---|
| Num Creates | Number of times beans were created | 8 | ☑ | 1.0 |
| Num Removes | Number of times beans were removed | 12599 | ☑ | 0.0010 |
| Num Activates | Number of times beans were activated | 13130 | ☑ | 0.0010 |
| Num Passivates | Number of times beans were passivated | 539 | ☐ | 1.0 |
| Num Instantiates | Number of times beans were instantiated | 12608 | ☐ | 1.0 |
| Num Destroys | Number of times beans freed | 12599 | ☐ | 1.0 |
| Num Loads | Number of times bean data was loaded from | 13130 | ☐ | 1.0 |

1 groups(s) | 22 counter(s) | Refresh rate: 10 sec Table size: 40 View Data As: Raw Value Logging: OFF

# *WebSphere Tuning Tips and Techniques*

# WebSphere Tuning Tips and Techniques - Topics

## Use the latest version of WebSphere

## Setting initial GC size

## Configuring queues

- HTTP Server
- Web Container
- Data Source

## Configuring caches

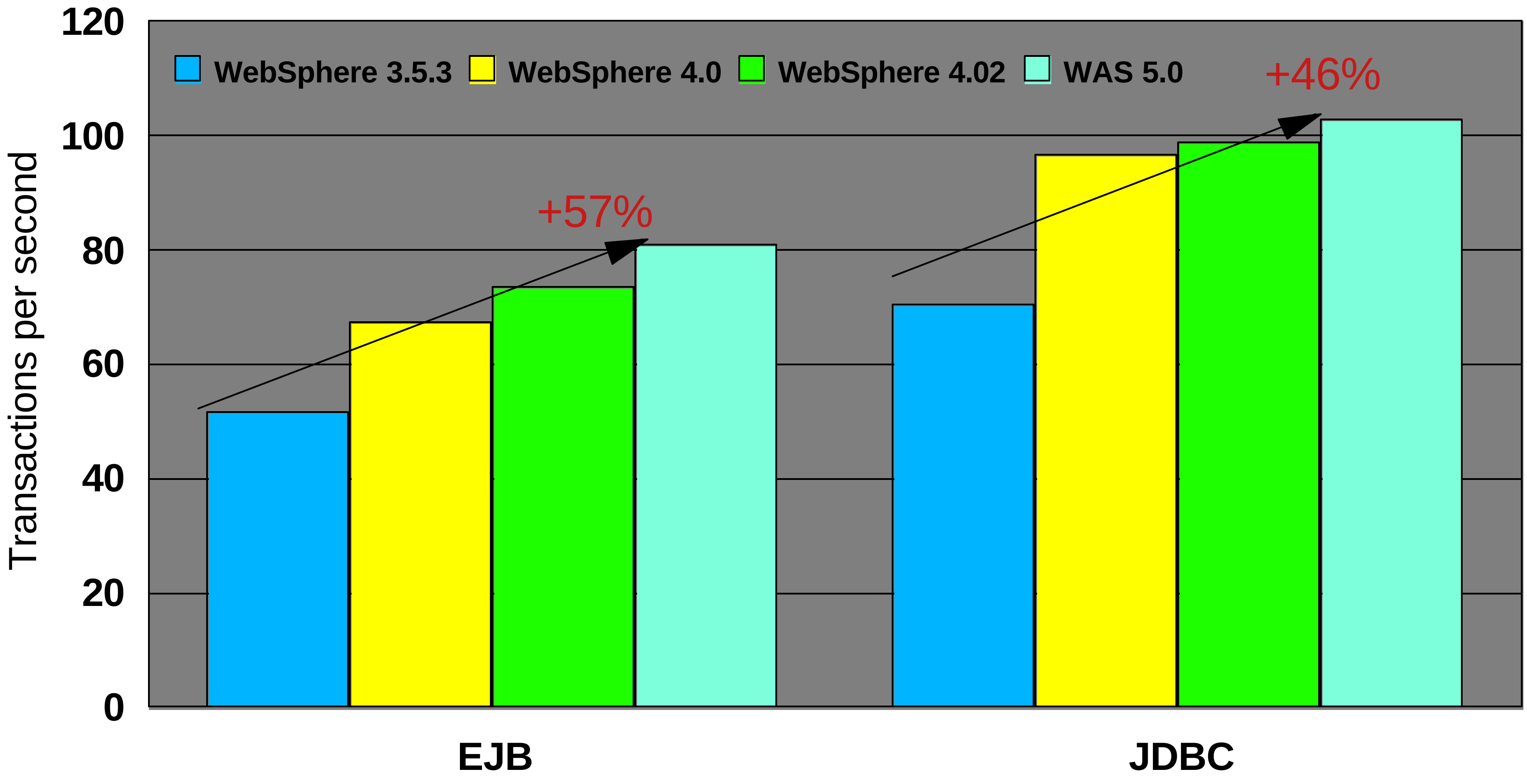
- Prepared Statement
- EJB

## Datasources and JDBC

# Use the latest version of WebSphere

**WebSphere Advanced 5.0 provides better performance than 4.0.x and 3.5.x**

**Load the latest WebSphere fix pack**



170-2385 252 MHz 1way

# Notes: Use the latest version of WebSphere

. Results were measured on a 170/2385 system
. Trade2 JDBC and Trade2 EJB benchmarks
. WebSphere 3.0.2, 3.5.0, 3.5.1, and 3.5.2 were on a V4R5 system
. WebSphere 3.5.3 was measured on both V4R5 and V5R1
. WebSphere 4.0 AE was measured on V5R1
. WebSphere 4.0.3 AE on V5R2 was estimated via measurements with WAS 4.0.2 with software enhancements to be included with WAS 4.0.3
. The IBM HTTP Server (powered by Apache) was used starting with the V5R2 measurements
. * - Results are projected from Trade2.7, which is 20% heavier then Trade 2.5
Notes/Disclaimers:
WebSphere Application Server Trade2 Results

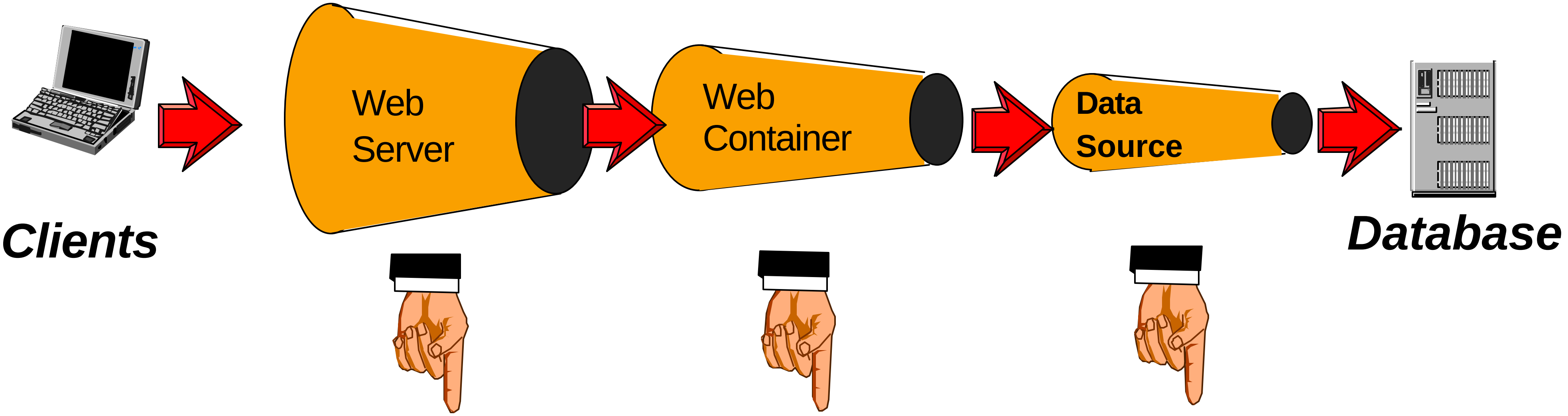
# Setting initial GC size (for WebSphere applications)

**Set the *initial* GC size to:**

- 64MB for 1-2 way systems
- 256MB for 4-8 way systems
- 512MB for 12 way and above
- ***Rule of Thumb: set initial GC size to 64MB per processor, increase as needed***

**This is twice the recommended initial GC size for non-WebSphere applications.**

# Configuring Queues



*HTTP Server* ***Administration - Use the performance tuner to change*** **Max Active Threads**

***Use the Admin Console to change*** **Web Container Max Threads** ***(formerly known as Max Connections)***

***Use the Admin Console to change*** **Maximum Connections Pool Size**

# Configuring Queues (cont.)

## WebSphere Queues

- Bigger doesn't necessarily mean better performance
- Testing is the premier way to tune the queues
- Tune from Back To Front
  - Data Source
  - Web Container
  - HTTP Server
- Where able, restrict pool from dynamically growing

## Guidelines

- Set HTTP threads somewhat higher than the maximum application concurrency
- Set the Servlets queue size to a lower value
- Set the DataSource queue size to an even lower value

## Excellent Whitepaper:

- http://www.ibm.com/**software**/webservers/appserv/3steps_perf_tuning.pdf

# Prepared Statement Cache

## When a database query is made, there are two phases

- Prepare phase - Parse the SQL text and put in a format the database understands
- Execute phase - Execute the query

## WebSphere and DataSources

- WebSphere handles statement caching for you with a Prepared Statement Cache
  - After closing a PreparedStatement allocated through DataSource, WebSphere will keep it open under the covers in the Prepared Statement Cache
  - One prepared statement cache per DataSource
    - Two separate connections using the same DataSource will use the same prepared statement cache
    - One connection cannot use a prepared statement from a different connection

Rule of thumb: Set statementCacheSize to:

stmtCacheSize = (number of stmts per connection) * (max number of connections)

# EJB Cache

**Tune the cache settings (absolute and preferred limits, size, cleanup interval) to avoid passivation of objects.**

**To estimate the required value for the absolute limit property,**

**multiply the number of entity beans active in any given transaction by the total number of concurrent transactions expected.**

**Then add the number of active session bean instances.**

**Tip: remove stateful Session beans when finished with them**

- Instances of stateful session beans have affinity to specific clients.
- Reduce Container cleanup by explicitly removing after use:

```
mySessionBean.remove();
```

# *Java Tuning Tips and Techniques*

# Java Tuning Tips and Techniques - Topics

**System level optimization**

**Initial heap size setting**

**Execution mode**

# System Level Optimization

## Set MAXACT for the *BASE pool

- The maximum number of threads that can use the processor concurrently. If the activity level is too low, the threads may transition to the ineligible condition. If the activity level is too high, excessive page faulting may occur.
- It is important to increase this value for systems that are executing a large number of threads. If set too low, may slowdown or even hang the system. Note. The value applies to number of **threads** not jobs
- Increasing this value will reduce or eliminate thread transitions into the ineligible state.
  - Initial choice of value should be (arbitrarily) high, then as implementation proceeds, monitor, and decrease the value if necessary.
  - Consider a separate pool if non-Java work in *BASE
- Can set and monitor via WRKSYSSTS ASTLVL(*INTERMED)

## Apply the latest Java PTFs

- Performance improvements may be included

# Initial Heap Size Parameter

## Tuning the Initial Heap Size is the way to tune the Garbage Collector (GC) on iSeries

## When starting the JVM, the initial heap size is specified

- -Xms option on Java command line (Ex:  Java -Xms256m MyClass)
- Initial Heap Size should be called "garbage collection threshold" on the iSeries
  - Everytime the JVM creates objects of this size, it will run the GC.

## This value should be tuned for each application

- If the value is too high
  - The heap will grow too large, resulting in a higher cache miss ratio and increased paging
  - The JVM would need a larger amount of memory to run
  - The GC would have to collect more objects and scan more memory every time it runs
- If the value is too low
  - The GC will be kicked off too often, resulting in a lot of CPU cycles dedicated to the GC

# Initial Heap Size Parameter (continued)

## Measure GC efficiency by measuring the time spent performing GC

- Get PEX trace profile, and print the report using *PROGRAM.
  - Look for JAVAGC program
  - Rule of thumb is less then 10 percent of total CPU time should be spent in JAVAGC.
- <or> Run with the -verbosegc option on the JVM command line
  - Will dump out how long the GC ran during each collection

## Rule of thumb for Initial Heap Size (Java programs)

| Processors | Initial Heap Size |
|---|---|
| 1 | 32 MB |
| 2 | 64 MB |
| 4 | 256 MB |
| 8 or more | 512 MB |

# Execution Modes

## Direct Execution (DE)

- Java classes or jar files are compiled into hidden, static programs

## Just in time compilation (JIT)

- Performs all compiles dynamically as the classes are used

## Mixed-Mode Interpreter (MMI) execution

- Uses Direct Execution or Interpreter until a method is deemed a common path
- JIT compile is then performed on the method

| Characteristic | Direct execution (DE) | Just-in-Time (JIT) Compiler | Mixed-Mode Interpreter (MMI) |
|---|---|---|---|
| Execution speed | In V5R2, DE is slower then JIT. Primarily due to optimizations can not extend past the class | Faster for most programs and dynamic environments, once the application warms up | Just about same as JIT compiler |
| Bring up speed | Fast if the code is pre-compiled; slow if is not pre-compiled | Slower then pre-compiled DE, due to compilations being performed on all methods dynamically | Somewhere between JIT and DE, due to some (but not all) methods being JIT compiled |
| Ease of use | More user management | Invisible to user | Invisible to user |

# *JDBC Tuning Tips and Techniques*

# JDBC Tuning Tips and Techniques - Topics

## What is JDBC

## Which driver to use with iSeries

## Misc. Tips and Techniques

- in Backup section

# What is JDBC?

## JDBC is Java's call-level interface for SQL data access

- Based on X/Open SQL CLI (Call Level Interface)

## DBMS-independent interface

- Generic SQL database access framework which provides a uniform interface on top of a variety of different database connectivity modules
- Allows programmers to write to a single database interface
- Enables DBMS-independent Java application development tools and products
- Allows database connectivity vendors to provide a variety of different connectivity solutions

# Which JDBC Driver to use?

| | Toolbox Driver | Developer Kit (Native) Driver |
|---|---|---|
| Driver Type | Network enabled,<br>Type 4 | Direct access<br>Type 2 |
| DB server | QZDASOINIT<br>same as ODBC | QSQSRVR<br>Call Level Interface |
| Statement caching | *SQLPKG | System-wide statement cache |
| Recommended Usage | Use when database does not reside on same machine as client | Use when database resides on same machine as client, for better performance |

# *Summary*

More time available?
Goto - Resolving Common Java and WebSphere Performance Problems

# Java and WebSphere Performance *Summary* - key points

The JVM continues to be improved each release.

Starting in V5R2, running with the JIT will result in best performance.

Tools like ANZJVM continue to be added and improved to help resolve performance problems. They are being added for good reason, so consider where they may help you in the future.

PEX database files have changed for V5R2. You may need to update "manual" queries and will need the latest version of PTDV from alphaWorks.

Five levels of stack in V5R2 Java PEX event data can be very useful to get "quick and dirty" information.

Move to WebSphere Application Server 5.0 (or Express) for best WAS performance

iSeries continues to demonstrate leadership in industry standard benchmarks.

# *More Questions ?*

# *Backup*
# *JDBC Tips and Techniques*

# Code Examples for Selecting JDBC Driver

## Native Driver for server Java to DB2 UDB for 400

- Register/Load the Driver

```
java.sql.DriverManager.registerDriver(new
  com.ibm.db2.jdbc.app.DB2Driver());
```

- Connect

```
Connection c = DriverManager.getConnection("jdbc:db2://mySystem",p);
```

## Toolbox Driver for client Java to DB2 UDB for AS/400

- Register/Load the Driver

```
java.sql.DriverManager.registerDriver(new
  com.ibm.as400.access.AS400JDBCDriver());
```

- Set the properties

```
Properties p = new Properties();
p.put("extended dynamic", "true");
```

- Connect

```
Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

# JDBC Database Subsystem Settings

## Increase the number of QSQSRVR initial jobs (Native JDBC)

- Use a value equal to the approximate number of expected concurrent transactions, plus something for the Application Server.
- Used for Native JDBC, default 5
- CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQSRVR)
- Increased number slightly increases overhead
- Ensure maximum number of uses is at default (200)

## Increase the number of QZDASOINIT initial jobs (Toolbox)

- Used for Toolbox JDBC, default 1
- CHGPJE SBSD(QSYS/QSERVER) PGM(QIWS/QZDASOINIT)
- Same rules as above

# Tips for Improving JDBC Performance

## Utilize PreparedStatement objects

- Especially for repetitive execution of SQL Statement (allows for re-use)
- Cache prepared Statements for subsequent operations
- Statement caching built-into WAS using DataSource object

## Ensure you close all open statements, when the application has finished processing them

## Reuse and pool database connections

- Creating and setting up a connection is expensive.
- Database connection pooling is built-in to WAS 3.0 or later

# Tips for Improving JDBC Performance (continued)

## Select only columns needed for application

- Do not specify "SELECT *", unless all columns are needed

## Specify the ordinal number of the column instead of column name

- Column name must be resolved before processing
- Column number will not have any extra processing

## Use blocked operations

- When retrieving a large result set
- When inserting multiple rows within a database table

## Create character columns as UNICODE (CCSID 13488), if allowable

- No conversion is needed between Java and database data

# Tips for Improving JDBC Performance (continued)

## Avoid the use of packed and zoned decimal

- Best to use primitive types (int and double)
- Minimizes object creates

## Minimize the use of getString(), especially on objects that do not need to be treated as strings

- Performs an object instantiation

## Use the appropriate commitment control level

- Use the minimum acceptable to ensure data integrity of application
- Higher levels require more processing and locking

## Utilize stored procedures

- Allows Java to call embedded SQL programs

# Tips for Improving JDBC Performance (continued)

Use Batch Updates (part of JDBC 2.0 specification)

Sends a set of updates to DB to be executed at the same time.

Example:

```
dbConn.setAutoCommit(false);

Statement stmt = dbConn.createStatement();
stmt.addBatch("INSERT INTO test VALUES (10,'Text 1',1,2");
stmt.addBatch("INSERT INTO test VALUES (11,'Text 2',3,1");
stmt.addBatch("INSERT INTO test VALUES (12,'Text 3',2,2");

int[] updCnt = stmt.executeBatch();
dbConn.commit();
```

# *WebSphere Tips and Techniques*

# DataSources and JDBC

## Reuse DataSources for JDBC connections

- A DataSource is obtained through a JNDI naming lookup
- Obtain the DataSource in the Servlet.init() method and cache it

## Use JDBC connection pooling

- Avoids acquiring and closing JDBC connections

## Release JDBC resources when done

- Failure to do this can cause long waits for connections
- Ensure code is structured to close and release under all conditions (even in exceptions and error conditions)

# DataSource and JDBC Example

## Using JDBC the right way:

```
          public class GoodJDBCServlet extends HttpServlet {

            private javax.sql.DataSource ds = null;   // For Caching the DataSource

            // Get the DataSource (Exception Handling removed for clarity)
            public void init(ServletConfig config) throws ServletException {
              super.init(config);
              Context ctx = new InitialContext();
              // Store the DataSource for use by every instance
              ds = (javax.sql.DataSource)ctx.lookup("jdbc/SAMPLE");
              ctx.close();
            }

            // Get a pooled connection
            public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
              try { // Get connection and execute Query
                Connection conn = ds.getConnection(USERID,PASSWORD);
                PreparedStatement pStmt = conn.prepareStatement("select * from SCHEMA.SOMETABLE");
                ResultSet rs = pStmt.executeQuery();
                  ...
              } finally { // Always close both the preparedStatement and the Connection
                if (pStmt!=null) pStmt.close(); // Note: Wrap this statement in try..catch
                if (conn!=null) conn.close();   // Note: Wrap this statement in try..catch
              }
            }
          }
```

# *Resolving Common Java and WebSphere Performance Problems*

# Resolving Common Problems

Object Leaks

High CPU consumption

Inactive CPU

Excessive Object Creates

Excessive Exception Processing

Excessive Locking

Preparing Statements in WebSphere Applications

# Object Leaks

## Symptoms

- Increasing paging rates as application runs
- Degrading response times

## Verification

- Examine -verboseGC output for number of active objects and heap size
- Collect *GBGCOLSWEEP PEX events and perform manual queries
- Run DMPJVM multiple times and compare number of objects and heap size

## Debug

- Use ANZJVM to determine which objects may be leaking
- Collect *OBJCRT PEX events and perform manual queries for object type and 5 levels of stack
- Collect *JVAENTRY, *JVAEXIT, and *OBJCRT PEX events to use with PTDV for object type and larger stack sizes

# High CPU Consumption

## Symptoms

- System CPU utilization too high

## Verification

- Use WRKACTJOB to monitor JVM CPU utilization

## Debug

- Collect PEX Trace Profile data and view printed report for "inline" CPU consumption. Improve performance of listed methods if they are in your application.
- Collect *JVAENTRY, *JVAEXIT PEX events to use with PTDV for inline and cumulative CPU analysis.  Improve the performance of methods using the most CPU, or recode to avoid calling expensive methods.

# Inactive CPU

## Symptoms

- As the amount of input is increased, the throughput stays about the same
- The CPU seems to max out well below 100%

## Debug

- Use WRKSYSSTS to verify MAXACT is set high enough to avoid thread transitions to Ineligible state
- If running WebSphere, attempt to tune the WebSphere queues outlined earlier
- Look at thread stacks reported by DMPJVM for waiting threads
- Collect *LCKSTR PEX events for manual queries to determine objects being locked on and 5 levels of stack
- Collect *JVAENTRY, *JVAEXIT, *LCKSTR, *UNLCK PEX events for use with PTDV to determine objects being locked on, length of locks, and larger stack sizes
- Collect *THDWAIT PEX events for manual queries
- Collect *JVAENTRY, *JVAEXIT, *THDWAIT, *THDNFY, *THDNFYALL PEX events for use with PTDV

# Excessive Object Creates

## Symptoms

- Higher CPU than expected in JVM

## Verification

- Check if PEX Trace Profile shows > 10% of time in JAVAGC

## Debug

- Collect *OBJCRT PEX events for manual queries to determine most popular objects created and 5 levels of stack. Try to reduce object creates through object re-use or other code modifications.
- Collect *JVAENTRY, *JVAEXIT, *OBJCRT PEX events for use with PTDV to determine most popular objects created and more levels of stack.

# Excessive Exception Processing

## Symptoms

- Higher CPU than expected in JVM

## Debug

- Collect *OBJCRT PEX events for manual queries to determine objects created with names like "XYZException" and 5 levels of stack.
- Collect *JVAENTRY, *JVAEXIT, *OBJCRT PEX events for use with PTDV to determine exception objects created and additional levels of stack information.

# Excessive Locking

## Symptoms

- High CPU in OS/400 spinWait routines
- CPU not scaling with load

## Detailed Debug

- Look at thread stacks reported by DMPJVM for waiting threads
- Collect *LCKSTR PEX events for manual queries to determine objects being locked on and 5 levels of stack
- Collect *JVAENTRY, *JVAEXIT, *LCKSTR, *UNLCK PEX events for use with PTDV to determine objects being locked on, length of locks, and larger stack sizes
- Collect *THDWAIT PEX events for manual queries
- Collect *JVAENTRY, *JVAEXIT, *THDWAIT, *THDNFY, *THDNFYALL PEX events for use with PTDV

# Preparing Statements in WebSphere Applications

## Symptoms

- CPU in JVM greater than expected
- CPU increases too much when additional load added

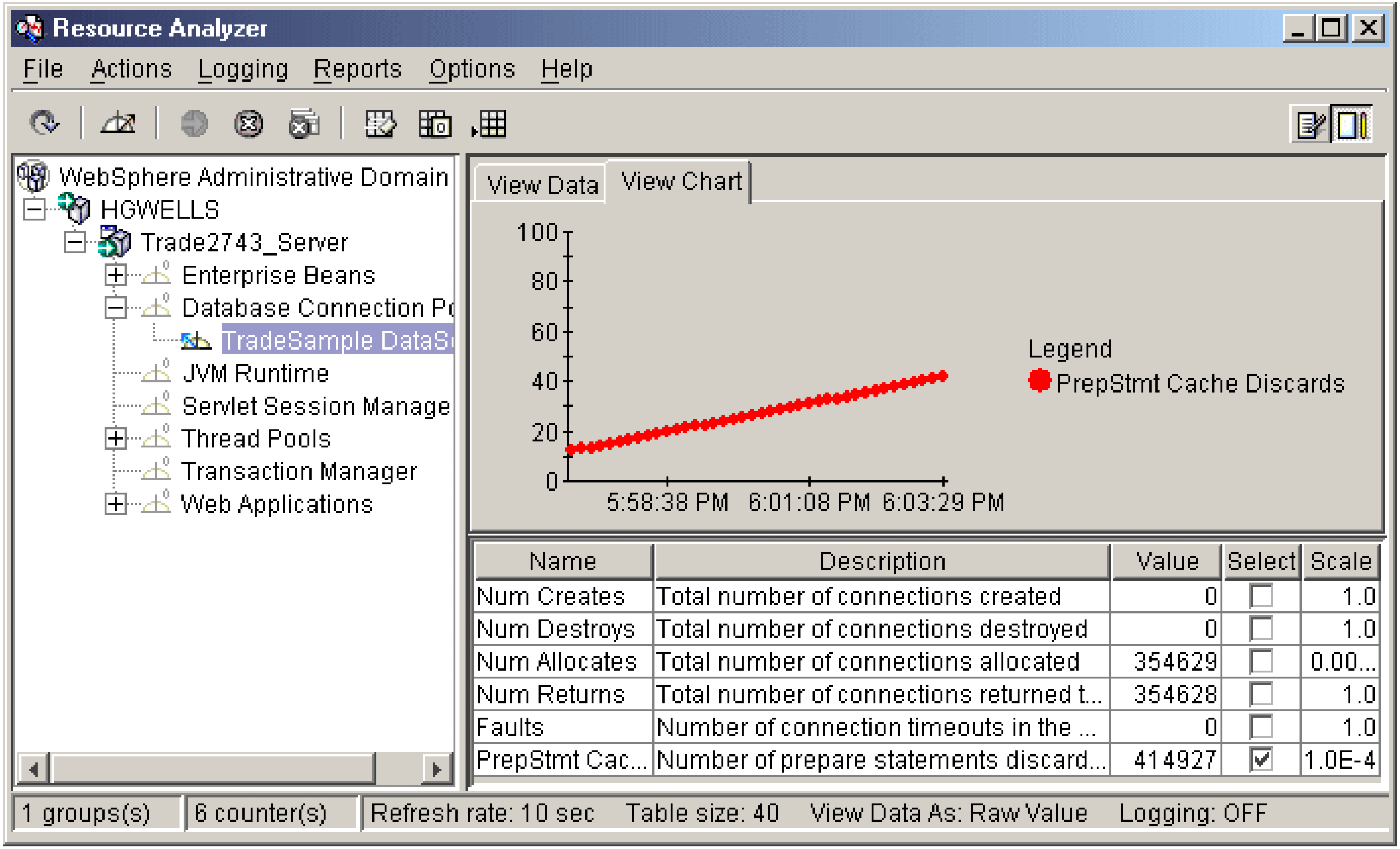
## Verification

- Routines doing parsing show up high in PEX trace profiles
- Resource Analyzer Prepared Statement Cache monitoring shows discards

## Debug

- Increase size of cache until discards are gone.  Allow room for growth if load may increase.

# Tuning WebSphere Prepared Statement Cache - Using Resource Analyzer to Detect Problem



| Name | Description | Value | Select | Scale |
|---|---|---|---|---|
| Num Creates | Total number of connections created | 0 | □ | 1.0 |
| Num Destroys | Total number of connections destroyed | 0 | □ | 1.0 |
| Num Allocates | Total number of connections allocated | 354629 | □ | 0.00... |
| Num Returns | Total number of connections returned t... | 354628 | □ | 1.0 |
| Faults | Number of connection timeouts in the ... | 0 | □ | 1.0 |
| PrepStmt Cac... | Number of prepare statements discard... | 414927 | ☑ | 1.0E-4 |

1 groups(s) | 6 counter(s) | Refresh rate: 10 sec Table size: 40 View Data As: Raw Value Logging: OFF

# end-to-end Technical Support - the Key Links

**How to engage Technical Sales Support**

- ✓ External Support:
  - ✓ http://www.ibm.com/support
- ✓ Sizing Tool - http://www.ibm.com/servlet/EstimatorServlet

INTEGRATED APPLICATION SERVERS

IBM eserver iSeries

NEW TOOLS TO MANAGE E-BUSINESS, INNOVATIVE TECHNOLOGY, APPLICATION FLEXIBILITY.

http://www-1.ibm.com/servers/eserver/iseries/education/key.html

http://publib.boulder.ibm.com/pubs/html/as400/infocenter.html

IBM eserver iSeries University

http://www.ibm.com/servers/eserver/iseries/education/

Redbooks

http://www.redbooks.ibm.com

IBM eserver iSeries Technology Center

http://www.ibm.com/servers/eserver/iseries/service/itc/educ.htm

IBM®

Partners In Education

*Educating at the speed of change*

IBM eserver iSeries Support

Served by Lotus® Domino™ with iSeries® power

http://www.ibm.com/eserver/iseries/support

http://www-1.ibm.com/servers/eserver/iseries/education/pie/

# WebSphere Application Server Performance *References*

## WebSphere Application Server for iSeries™ Performance Considerations

- **WebSphere 5.0 and WebSphere 5.0 Express**
  - http://www-1.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/performancews50.html
- **WebSphere 4.0**
  - http://www-1.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/performanceAE40.html
- **WebSphere 3.5.x**
  - http://www-1.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/performanceAE35.html

## iSeries Performance Capabilities Reference

- V5R2
  - **http://ca-web.rchland.ibm.com/perform/perfguideup/V5R21PerfGuide/V5R21PerfGuide.pdf**
  - http://ca-web.rchland.ibm.com/perform/perfguideup/v5r2perfguide/v5r2perfguide.pdf
- V5R1
  - http://ca-web.rchland.ibm.com/perform/perfguideup/v5r1perfguide/v5r1perfguide.pdf

## WebSphere Application Server - Performance Home Page

- http://www-3.ibm.com/software/webservers/appserv/performance.html

# References

## WebSphere

- iSeries
  - http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/
- iSeries WebSphere Performance White Paper
  - http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/PerformanceConsiderations.html
- PartnerWorld for Developers
  - http://www.iseries.ibm.com/developer/websphere
- IBM WebSphere
  - http://www.software.ibm.com/webservers/
  - http://www.software.ibm.com/webservers/appserv
- WebSphere White Papers
  - http://www.ibm.com/software/webservers/appserv/whitepapers.html

## iSeries Performance Tools

- http://www.ibm.com/servers/eserver/iseries/perfmgmt/

## Redbooks/Redpapers/Redpieces

- http://www.redbooks.ibm.com/

# References

## iSeries 400

- http://www.ibm.com/as400/infocenter

## iSeries 400 Web Programming Technology

- http://www.iseries.ibm.com/ebusiness/

## WebSphere

- http://www.iseries.ibm.com/websphere
- http://www.iseries.ibm.com/developer/websphere
- http://www.software.ibm.com/webservers/
- http://www.software.ibm.com/webservers/appserv

## VisualAge for Java

- http://www.software.ibm.com/ad/vajava

## Java

- http://www.iseries.ibm.com/developer/java

# Trademarks and Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AS/400
AS/400e
e (logo) business
IBM

IBM(logo)
iSeries
OS/400
WebSphere

Lotus, Freelance Graphics, and Word Pro are registered trademarks of Lotus Development Corporation and/or IBM Corporation.
Domino is a trademark of Lotus Development Corporation and/or IBM Corporation.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.
SPECjbb®2000, SPECjAppServer®2001, SPECweb®99, and SPECweb®99_SSL are registered trademarks of the Standard Performance Evaluation Corporation (SPEC).
Other company, product and service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.