



SAP on i

Getting the Most from Your SAP Java Application Server on IBM i

Kolby Hoelzle
hoelzle@us.ibm.com

Agenda

- **An Overview of Java**
- **Architecture of the IBM i Classic JVM**
- **Analyzing JVM Performance**
- **Tuning the JVM**
- **Recognizing the Need to Tune the JVM**

Introduction

- **This presentation will provide:**
 - An overview of Java and the architecture of the IBM i Classic JVM
 - A methodology for analyzing Java performance
 - Tips and techniques for tuning the JVM in a SAP Netweaver environment

- **This presentation will not provide:**
 - Instruction on how to tune the OS or the application
 - Details of the IBM Technology for Java (IT4J) JVM



| SAP on i

The Good, The Bad, and The Ugly -- An Overview of Java

Three Primary Components Make Up Java

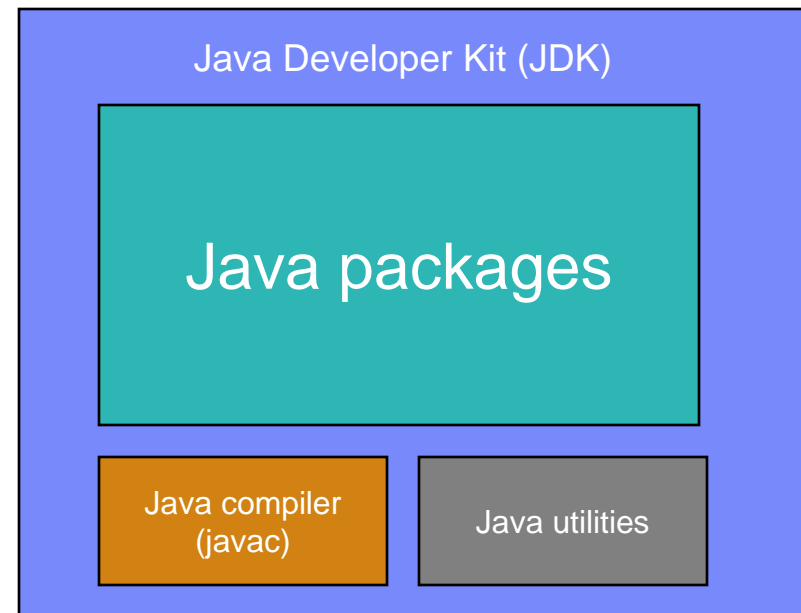
- **Java language**
- **Java Developer Kit (JDK)**
- **Java Virtual Machine (JVM)**

Java Language

- **Object oriented**
- **Syntax based on C++**
- **Compiles into platform independent bytecodes called a class**
- **Similar classes can be grouped together in a container called a package**

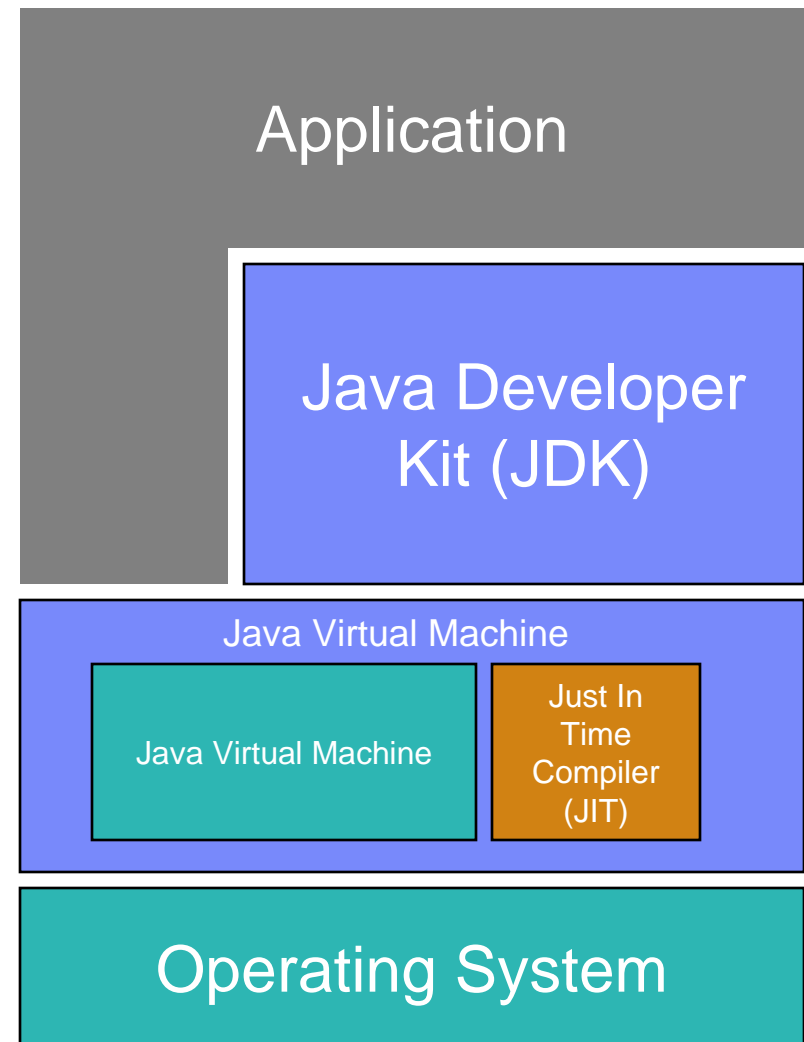
Java Developer Kit (JDK)

- **Contains the Java run time packages**
- **New JDK versions released regularly**
- **Each JDK is backward compatible**
- **Programs are compiled to a specific JDK level**

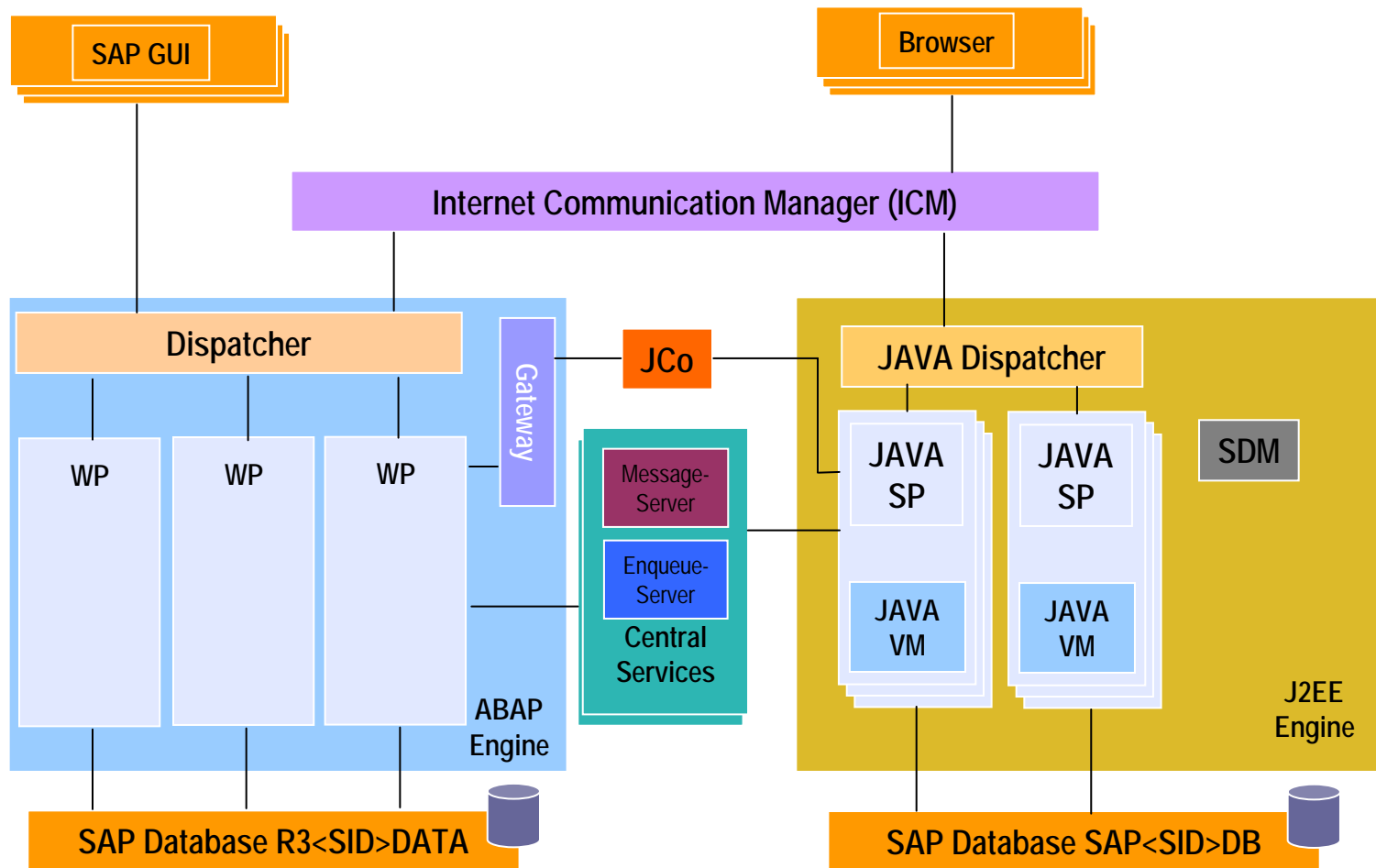


Java Virtual Machine (JVM)

- **Java classes execute on a JVM**
- **JVM runs on operating system**
- **JVMs are platform dependent**
- **Many aspects of JVMs are not standardized**



NetWeaver 7.0 Application Server Architecture



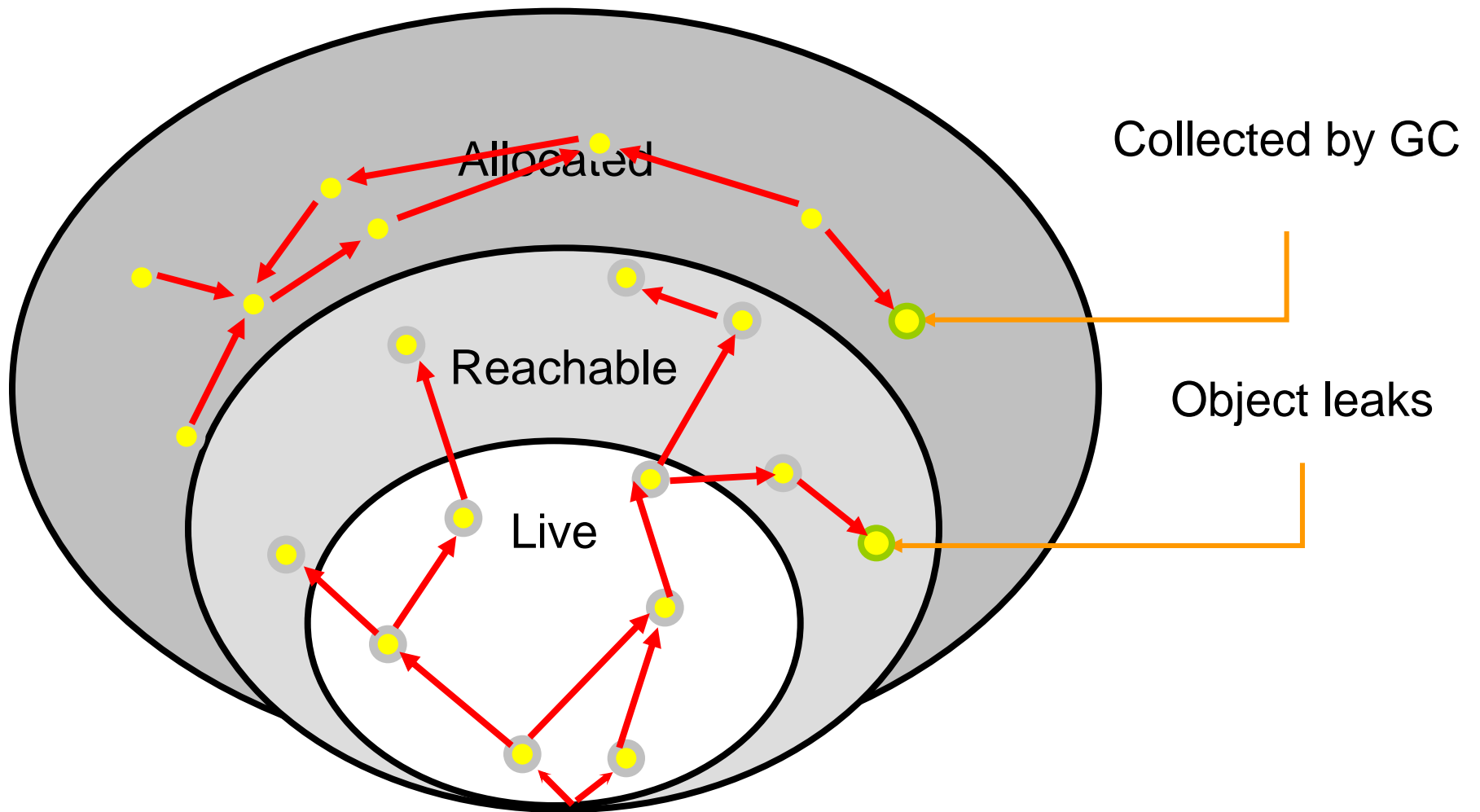
Java Memory Management

- **Typically called garbage collection**
- **OS allocates memory to JVM for use as heap**
- **JVM allocates and de-allocates heap memory for Java objects in an application**
- **JVM utilizes a mechanism called the garbage collector to free unused memory**

Java Garbage Collection

- **Objects collected when they are unreachable by the application**
 - An object with no references to it is unreachable
- **Garbage collector is part of the JVM**
- **JVM determines when garbage collector runs**
 - Behavior can be influenced by configuration
- **Garbage collector has overhead**
- **Many GC algorithms exist**

Illustration of Garbage Collection



Comparing Java to C/C++

Java

- Executables run on VM
- Memory automatically managed by JVM
- No direct memory access
- Static and dynamic optimization

C/C++

- Executables run on OS
- Memory managed by application
- Direct memory access
- Static optimization only



| SAP on i

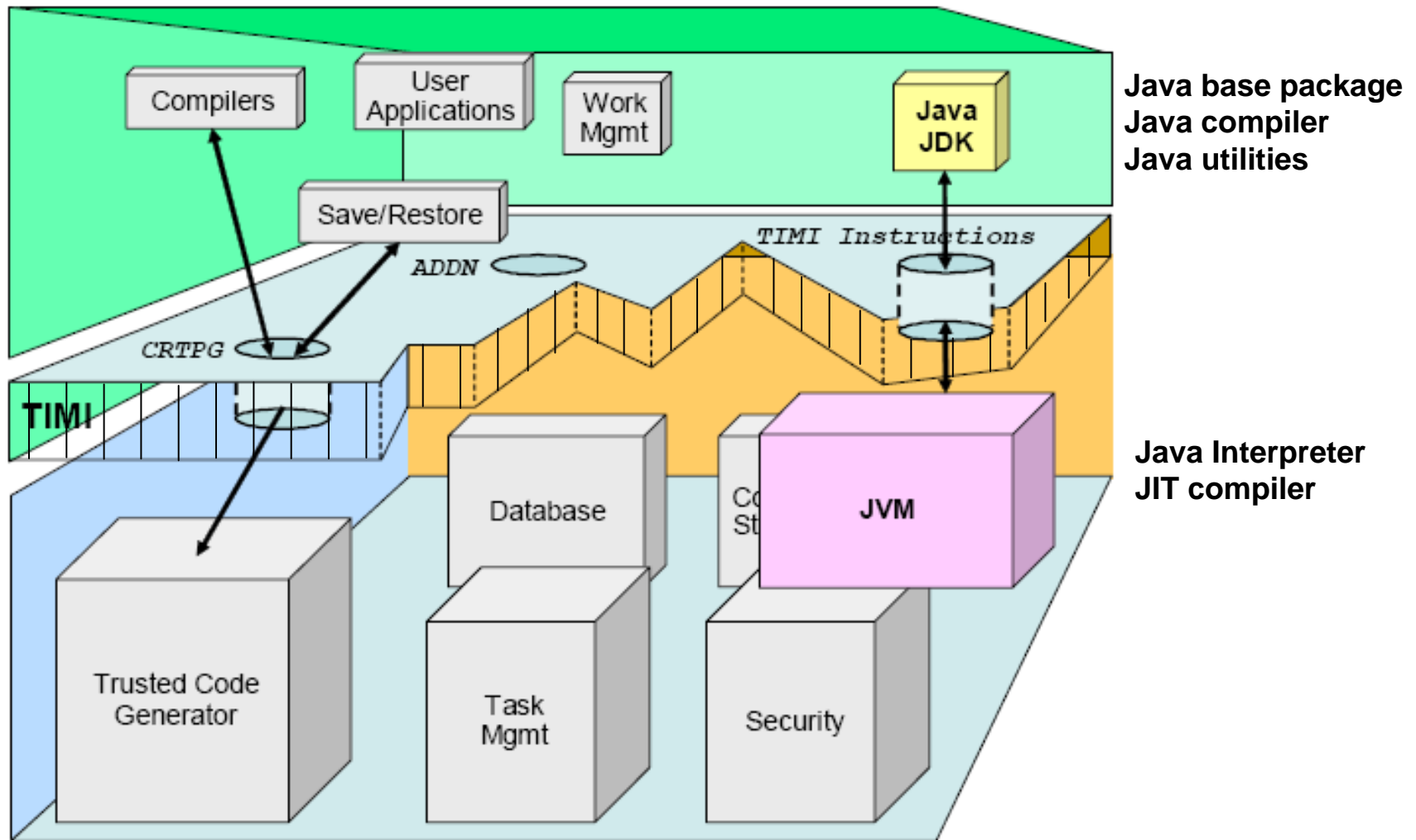
Living in a Virtual World

-- Architecture of the IBM i Classic JVM

The IBM i Classic JVM

- **Classic JVM implemented in SLIC**
- **Majority of work performed below the Machine Interface (MI) layer**
- **Java Development Kit (JDK) located above MI layer**
- **Java first supported with V4R2**
- **Licensed program 57xx-JV1**

Classic JDK and JVM



The IBM i Classic JVM Garbage Collector

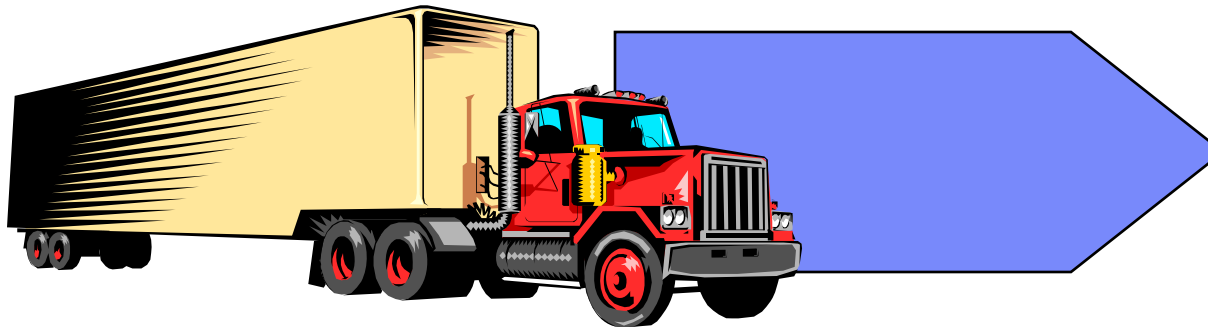
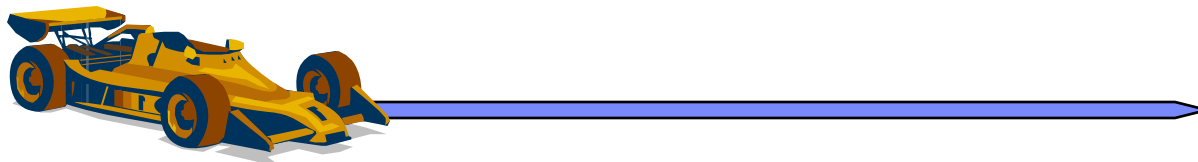
- **Asynchronous:** runs in the background
- **Concurrent:** runs at the same time as the other Java application threads
- **Multi-threaded:** utilizes multiple threads, but only one GC cycle can run at a time



| SAP on i

Analyzing JVM Performance

How Do You Define Performance?

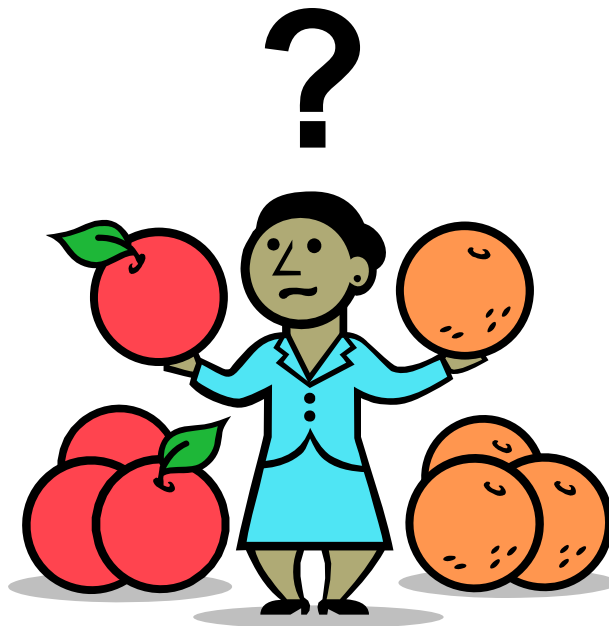


How Do You Measure Performance?

- **Standard of measurement (metrics)**
 - Challenge: picking a metric that provides useful information
- **Valid starting point for comparison (baseline)**
 - Challenge: keeping relative comparisons... well relative



Are we comparing apples to oranges when we should be comparing apples to apples?



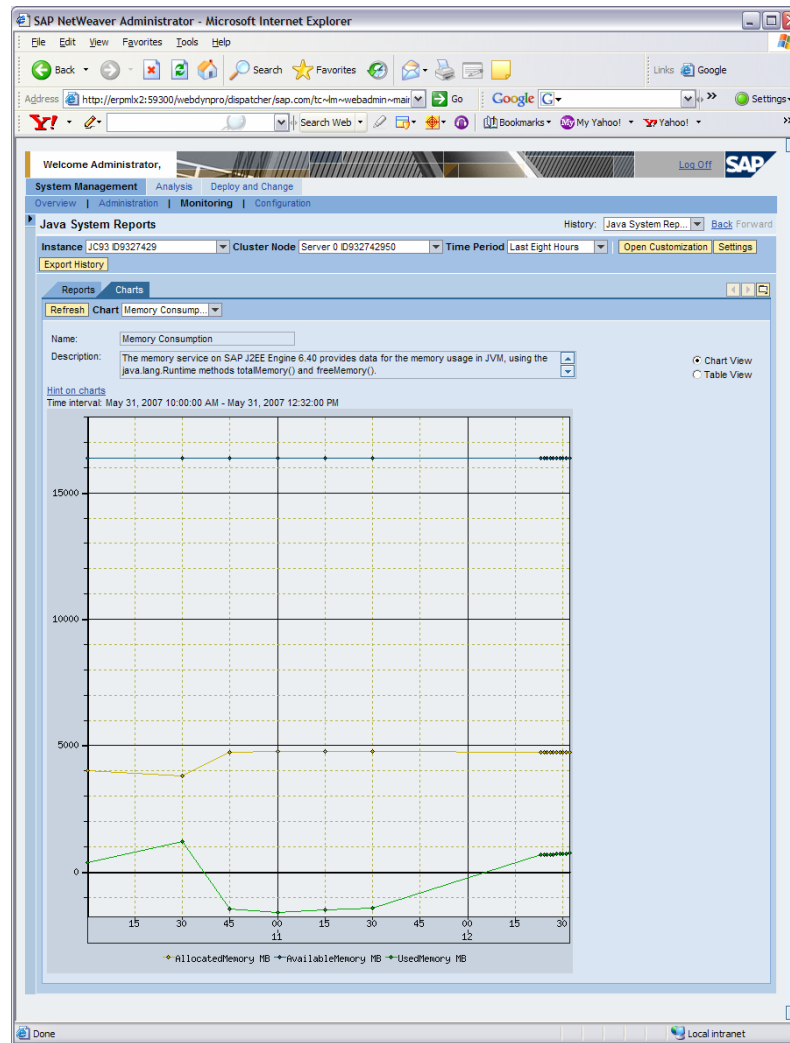
What Do You Look At For Performance?

- **Need to consider the entire stack**
 - Hardware
 - Operating system
 - Java virtual machine
 - Java application server
 - J2EE applications
 - Network

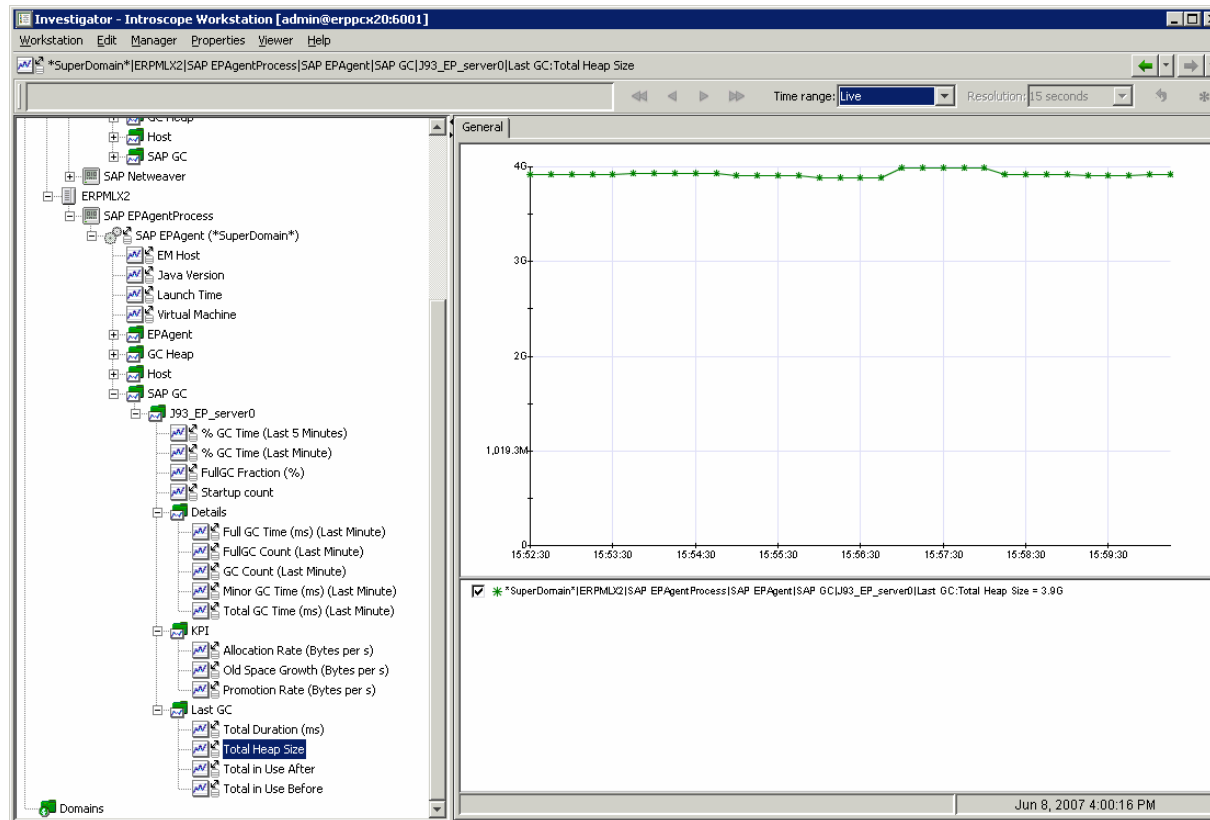
So You Still Think the Problem Is the JVM

- **Taking a closer look**
 - Netweaver administrator
 - Wily Introscope (Solution Manager Diagnostics)
 - Log files
 - Third party tools

Monitoring the Java Heap with NetWeaver Administrator



Monitor and Analyze Java with Wily Introscope



GUIs Are Nice, But Let's Look at What Really Goes On

- **System property `verbose:gc` prints trace data for the garbage collector**
- **The `verbose:gc` output can be analyzed using any text editor or viewer**
- **A pattern matching tool such as `grep` is also very helpful, but not necessarily required**

Understanding Garbage Collector Output

- Cycle number
- Collection start time
- Reason for collection
- Number of objects on the heap
- Number of objects collected
- Size of objects collected
- Number and types of references
- Cycle time
- Size of the heap
- Amount heap has grown since garbage collection cycle started
- Cycle ending time

Garbage Collector Output

```

                                /QOpenSys/usr/bin/-st
GC 2: collection ending 03/16/06 16:32:03
GC 3: starting collection, threshold allocation reached.
GC 3: collection starting 03/16/06 16:38:55
GC 3: live objects 3579509; collected objects 19880628; collected(KB) 2454780.
GC 3: queued for finalization 0; total soft references 16845; cleared soft references 0.
GC 3: current heap(KB) 3082080; current threshold(KB) 2621410.
GC 3: collect (milliseconds) 3203.
GC 3: current cycle allocation(KB) 67506; previous cycle allocation(KB) 2621463.
GC 3: total weak references 21666; cleared weak references 2041.
GC 3: total final references 25708; cleared final references 21764.
GC 3: total phantom references 0; cleared phantom references 0.
GC 3: total JNI global weak references 0; cleared JNI global weak references 0.
GC 3: collection ending 03/16/06 16:33:59
GC 4: starting collection, threshold allocation reached.
GC 4: collection starting 03/16/06 16:40:26
GC 4: live objects 4459928; collected objects 21621938; collected(KB) 2536347.
GC 4: queued for finalization 0; total soft references 22412; cleared soft references 0.

===> _____
_____
_____
F3=Exit      F6=Print    F9=Retrieve  F11=Truncate/Wrap
F10=Clear   F17=Top     F20=Dotcom   F21=CL command entry

```

Taking a Look at JVM Behavior

- **Reason for collection**
- **Size of the heap**
- **Cycle time**
- **Percentage of time garbage collector is running**

Look at why the garbage collector is running

```
grep "starting collection" std_server0.out
```

```
GC 1: starting collection, reason external thread attached.  
Login :GC 2: starting collection, threshold allocation reached.  
GC 3: starting collection, threshold allocation reached.  
GC 4: starting collection, threshold allocation reached.  
GC 5: starting collection, threshold allocation reached.  
GC 6: starting collection, threshold allocation reached.  
GC 7: starting collection, threshold allocation reached.  
GC 8: starting collection, threshold allocation reached.  
GC 9: starting collection, threshold allocation reached.  
GC 10: starting collection, threshold allocation reached.  
GC 11: starting collection, threshold allocation reached.  
GC 12: starting collection, threshold allocation reached.  
GC 13: starting collection, threshold allocation reached.  
GC 14: starting collection, threshold allocation reached.  
. .
```

Look at the behavior of the heap

```
grep "current heap" std_server0.out
```

```
GC 1: current heap(KB) 4160; current threshold(KB) 2621440.  
GC 2: current heap(KB) 2555808; current threshold(KB) 2621440.  
GC 3: current heap(KB) 3082080; current threshold(KB) 2621440.  
GC 4: current heap(KB) 3696576; current threshold(KB) 2621440.  
GC 5: current heap(KB) 3731168; current threshold(KB) 2621440.  
GC 6: current heap(KB) 3879744; current threshold(KB) 2621440.  
GC 7: current heap(KB) 3877440; current threshold(KB) 2621440.  
GC 8: current heap(KB) 3904224; current threshold(KB) 2621440.  
GC 9: current heap(KB) 3898208; current threshold(KB) 2621440.  
GC 10: current heap(KB) 3903328; current threshold(KB) 2621440.  
GC 11: current heap(KB) 3902880; current threshold(KB) 2621440.  
GC 12: current heap(KB) 3893504; current threshold(KB) 2621440.  
GC 13: current heap(KB) 3833120; current threshold(KB) 2621440.  
GC 14: current heap(KB) 3815200; current threshold(KB) 2621440.  
GC 15: current heap(KB) 3853696; current threshold(KB) 2621440.  
GC 16: current heap(KB) 3893408; current threshold(KB) 2621440.  
.
```

Look at how long it takes for a garbage collection cycle to complete

```
grep "collect (mill" std_server0.out
```

```
GC 1: collect (milliseconds) 24.  
GC 2: collect (milliseconds) 3552.  
GC 3: collect (milliseconds) 3203.  
GC 4: collect (milliseconds) 4420.  
GC 5: collect (milliseconds) 4445.  
GC 6: collect (milliseconds) 8914.  
GC 7: collect (milliseconds) 5045.  
GC 8: collect (milliseconds) 6538.  
GC 9: collect (milliseconds) 5037.  
GC 10: collect (milliseconds) 5170.  
GC 11: collect (milliseconds) 4407.  
GC 12: collect (milliseconds) 4413.  
GC 13: collect (milliseconds) 4290.  
GC 14: collect (milliseconds) 5017.
```

```
.  
. .  
. . .
```

Some fluctuations may occur, but this generally isn't a problem

Estimate percentage of time garbage collector is running

- **Estimate average cycle time**
- **Estimate approximate number of cycles per minute**
- **Multiple number of cycles by average cycle time get number of seconds per minute garbage collector runs**

Estimate estimate average cycle time

grep "collect (mill)" std_server0.out

```
      /QOpenSys/usr/bin/-sh

GC 41: collect (milliseconds) 5093.
GC 42: collect (milliseconds) 5375.
GC 43: collect (milliseconds) 5407.
GC 44: collect (milliseconds) 5238.
GC 45: collect (milliseconds) 5653.
GC 46: collect (milliseconds) 4560.
GC 47: collect (milliseconds) 5699.
GC 48: collect (milliseconds) 4795.
GC 49: collect (milliseconds) 5000.
GC 50: collect (milliseconds) 4630.
GC 51: collect (milliseconds) 5400.
GC 52: collect (milliseconds) 5206.
GC 53: collect (milliseconds) 4823.
GC 54: collect (milliseconds) 4952.
GC 55: collect (milliseconds) 4575.
GC 56: collect (milliseconds) 4859.
GC 57: collect (milliseconds) 5029.

===> _____

F3=Exit      F6=Print    F9=Retrieve  F11=Truncate/Wrap
F13=Clear   F17=top     F18=Bottom  F21=CL command entry
```

Estimate cycles per minute

grep "collection starting" std_server0.out

```
/QOpenSys/usr/bin/-sh
GC 41: collection starting 03/16/06 17:13:15
GC 42: collection starting 03/16/06 17:13:45
GC 43: collection starting 03/16/06 17:14:12
GC 44: collection starting 03/16/06 17:14:44
GC 45: collection starting 03/16/06 17:15:15
GC 46: collection starting 03/16/06 17:15:42
GC 47: collection starting 03/16/06 17:16:10
GC 48: collection starting 03/16/06 17:16:38
GC 49: collection starting 03/16/06 17:17:06
GC 50: collection starting 03/16/06 17:17:34
GC 51: collection starting 03/16/06 17:18:01
GC 52: collection starting 03/16/06 17:18:29
GC 53: collection starting 03/16/06 17:18:55
GC 54: collection starting 03/16/06 17:19:23
GC 55: collection starting 03/16/06 17:19:50
GC 56: collection starting 03/16/06 17:20:21
GC 57: collection starting 03/16/06 17:20:48
===> _____
F3=Exit      F6=Print    F9=Retrieve  F11=Truncate/Wrap
F13=Clear   F17=Top    F18=Bottom   F21=CL command entry
```



SAP on i

Knobs, Levers, and Dials -- Tuning the JVM

Let's Get Something Straight...

No amount of tuning will fully compensate for insufficient HW resources

Configuration and Tuning – Well It Depends On...

- **Performance goals**
- **Type of application**
- **Expectations**
- **Currency of hardware**
- **Amount of available hardware resources**
- **Concurrent workloads**

... and possibly even other things could have an impact

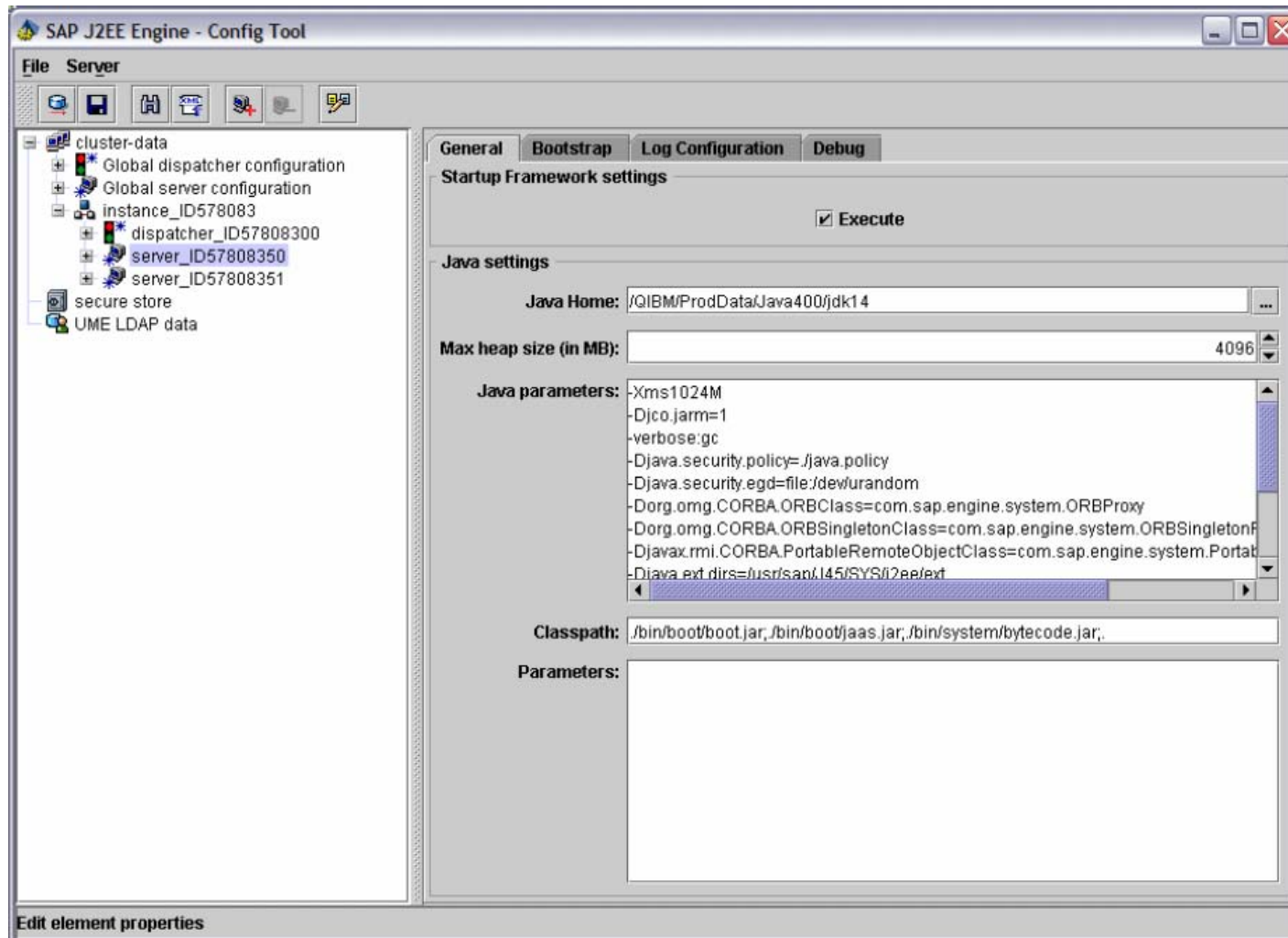
Levers, Knobs, and Dials

- **Hardware resources**
 - Processors
 - Memory
 - Disk
 - ...
- **Application configuration**
 - Number of SAP server nodes
 - Number of application threads per server node
 - ...
- **Operating system settings**
 - JVM configuration
 - Memory pools
 - Performance adjuster
 - Max active
 - ...

JVM Levers, Knobs, and Dials

- **Levers - set and forget**
 - `java.compiler=jitc` and `os400.run.mode=jitc`
 - Disable explicit gc (for some JVMs)
 - Enable class verification cache
- **Knobs and dials – settings vary by workload**
 - Initial heap size (`Xms`)
 - Maximum heap size (`Xmx`)

Making It Happen with the Config Tool



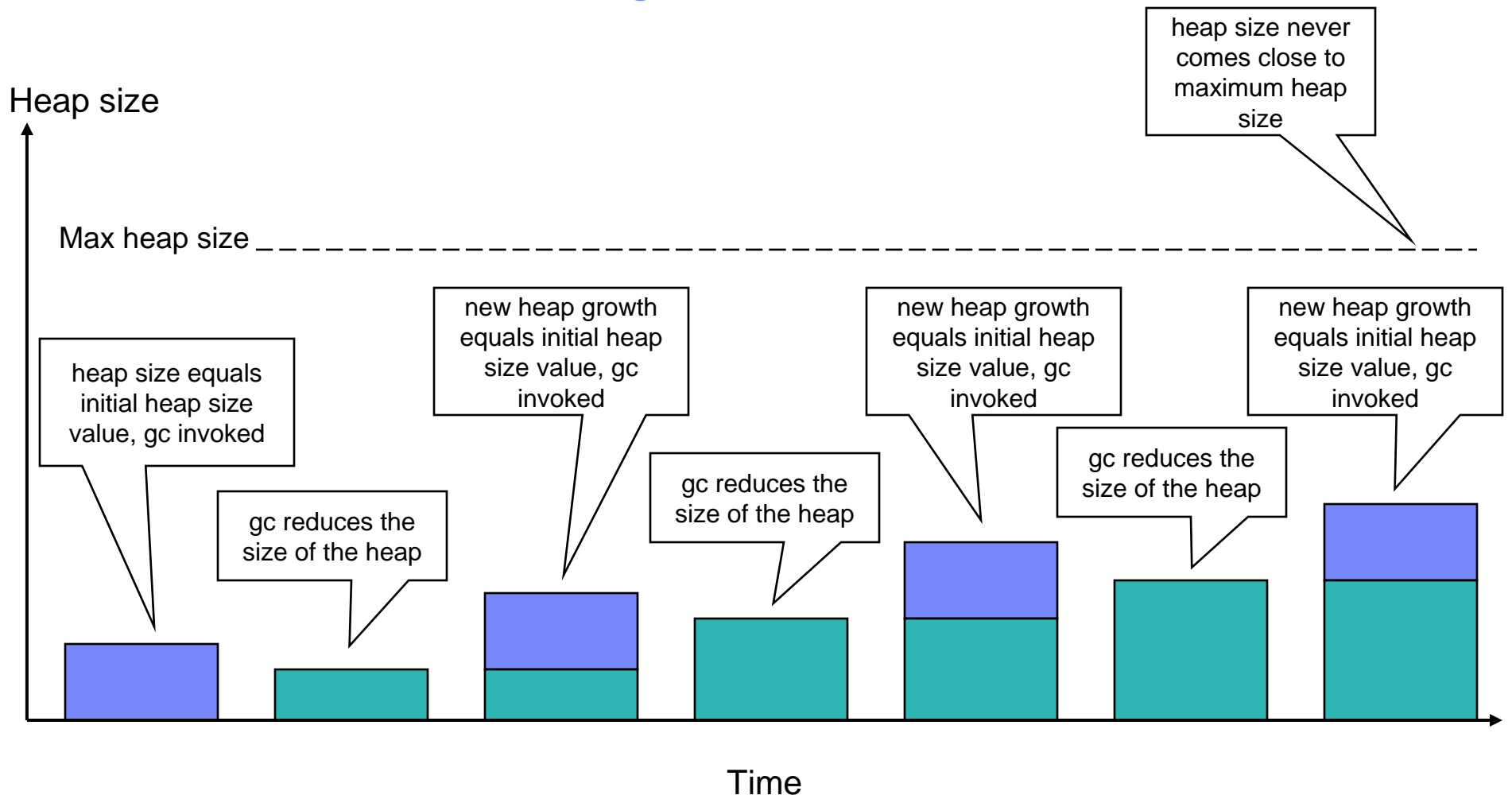
Adjusting the Knobs

- **Maximum heap size should not be used for tuning, but as a limit for heap growth in case something goes wrong with the application**
- **Rule of thumb is to set maximum heap size to the available physical memory and then leave it**
- **It is important the maximum heap size is set sufficiently high as it will have a negative impact on the GC if the value is too low.**

Dialing In JVM Performance

- **Initial heap size determines frequency of garbage collection cycles**
- **Value indicates the amount of heap growth before a new cycle starts**
- **Goal is to balance frequency of cycles with duration of cycles**

Behavior of the Garbage Collector



Tuning Guidelines

- **Ensure minimum HW requirements are met**
- **Set parameters as directed by SAP**
 - Complete other configuration
- **Collect and analyze when system is under load**
- **Start with a reasonable heap size and work your way up**
- **Increase initial heap size until the current GC cycle can finish before next one starts**

Fine Tuning

- **Tune GC so there is a balance between frequency of cycles and duration**
- **Each GC cycle consumes CPU**
- **Too many GC cycles consumes CPU unnecessarily**
- **Too few GC cycles may lead to increased paging, very long cycle times and possible abnormal heap growth**
- **Balance frequency of GC cycles with duration**
 - Rule of thumb: Garbage collector should run 50% or less of the time

Even More Fine Tuning

- **If desired the GC can be fine tuned even further by utilizing the Performance Tools LPP (5722PT1).**
 - This level of fine tuning is usually not necessary for most customer environments
- **Ideally, the GC should consume less than 15% of the CPU, preferably around 10%**
- **The initial heap size is adjusted to achieve this**

Remember Performance Tuning Is On-going

- **GC tuning is highly dependent on the application and the workload**
- **All workloads change over time**
 - Company growth
 - Usage patterns change
 - Unexpected use of application (always some of this)

Factors Influencing Workload ... Other Than the Workload

- **New or modified hardware**
- **Application fixes**
- **OS fixes**
- **Application upgrades**
- **OS upgrades**
- **Additional workload added to the partition**
- **Network**



| SAP on i

When Things Go Wrong -- Recognizing the Need for Tuning

Bad Garbage Collector!

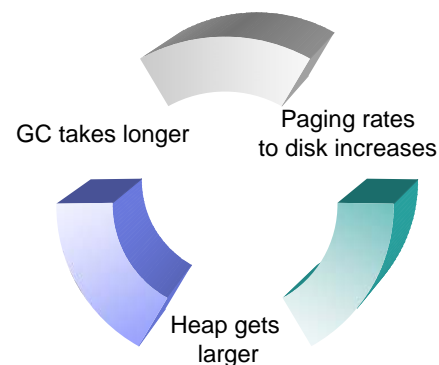
- **Most JVM performance problems are due to the GC**
- **... and most GC problems are usually caused by one of two things:**
- **Initial heap size set to low**
- **Not enough available memory**
 - not enough physical memory
 - maximum heap size set incorrectly
 - other application consuming memory

Symptoms – Initial Heap Setting Too Low

- **Garbage collector can't keep up**
 - Stacked garbage collector cycles
- **Inconsistent heap size and heap growth**
- **Huge fluctuations in cycle times**

Symptoms – Not Enough Memory

- **GC invoked for reasons other than threshold allocation reached**
- **Very rapid heap growth**
- **Extremely long cycle times**



GC Invoked for Reasons Other Than Threshold Allocation Reached

grep "starting collection" std_server0.out

```

/Q0penSys/usr/bin/-sr
GC 5: starting collection, reason stop the world collection.
GC 5: starting collection, reason stop the world collection.
GC 5: starting collection, reason stop the world collection.
GC 5: starting collection, reason stop the world collection.
GC 5: starting collection, reason stop the world collection.
GC 6: starting collection, threshold allocation reached.
GC 7: starting collection, threshold allocation reached.
GC 8: starting collection, threshold allocation reached.
GC 9: starting collection, threshold allocation reached.
GC 10: starting collection, threshold allocation reached.
GC 11: starting collection, threshold allocation reached.
GC 11: starting collection, threshold allocation reached.
GC 11: starting collection, threshold allocation reached.
GC 11: starting collection, threshold allocation reached.
GC 11: starting collection, maximum allocation reached.
GC 11: starting collection, maximum allocation reached.
GC 11: starting collection, maximum allocation reached.
===>
-----
F3=Exit      F6=Print    F9=Retrieve  F11=Truncate/Wrap
F10=Clear   F17=Top    F20=Dotcm   F21=CL command entry
```

Abnormal Heap Growth

grep "current heap" std_server0.out

```

/QOpenSys/usr/bin/-sh
>
$
>
$
> grep "current heap" std_server0.out
GC 1: current heap(KB) 4192; current threshold(KB) 2621440.
GC 2: current heap(KB) 2710496; current threshold(KB) 2621440.
GC 3: current heap(KB) 3324064; current threshold(KB) 2621440.
GC 4: current heap(KB) 4863840; current threshold(KB) 2621440.
GC 5: current heap(KB) 4841568; current threshold(KB) 2621440.
GC 6: current heap(KB) 4867360; current threshold(KB) 2621440.
GC 7: current heap(KB) 4847008; current threshold(KB) 2621440.
GC 8: current heap(KB) 4847008; current threshold(KB) 2621440.
GC 9: current heap(KB) 4884512; current threshold(KB) 2621440.
GC 10: current heap(KB) 16683904; current threshold(KB) 2621440.
GC 11: current heap(KB) 15976632; current threshold(KB) 2621440.
$
===>

```

F3=Exit	F6=Print	F9=Retrieve	F11=Truncate/Wrap
F10=Clear	F17=Top	F20=Dotcom	F21=CL command entry

Large Fluctuations and Extremely Long Cycle Times

grep "collect (mill)" std_server0.out

```
/QOpenSys/usr/bin/-sh
>
$
>
$
> grep "collect (mill)" std_server0.out
GC 1: collect (milliseconds) 45.
GC 2: collect (milliseconds) 5630.
GC 3: collect (milliseconds) 136147.
GC 4: collect (milliseconds) 258727.
GC 5: collect (milliseconds) 60421.
GC 6: collect (milliseconds) 66036.
GC 7: collect (milliseconds) 8860.
GC 8: collect (milliseconds) 5791.
GC 9: collect (milliseconds) 23359.
GC 10: collect (milliseconds) 1986438.
GC 11: collect (milliseconds) 5190231
$
===> _____
_____
_____
F3=Exit      F6=Print    F9=Retrieve  F11=Truncate/Wrap
F13=Clear    F17=Top    F18=Bottom   F21=CL command entry
```

And Finally Remember That ...

- **Keeping a system tuned is ongoing, not a one time task**
- **As your environment and workload change so must your configuration and tuning**
- **Identifying the need to tune your system early on will help you avoid potential problems in the future**
- **No amount of tuning will fully compensate for insufficient HW resources**



| SAP on i

Resources

References

- **SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”**
- **“SAP Netweaver Java on IBM i5/OS”, Redpiece, November 2006**
- **“Implementing SAP applications on the IBM System i platform with IBM i5/OS”, Redbook, June 2006**
- **“Tuning garbage collection for Java and WebSphere running classic JVM on i5/OS”, Whitepaper, February 2006**
- **“Basic Java Performance for iSeries”, Whitepaper, 2003**



SAP on i

Questions?