Full-scale Java™ Platform Diagnostics for Production Environments

Flavio Bergamaschi—IBM Hursley Lab, UK

Trent Gray-Donald—IBM Ottawa Lab, Canada

Chris Bailey—IBM Hursley Lab, UK

TS-3881

Goal

What's in it for you?

Learn about IBM's RAS technology and hear about a new toolkit that will help you quickly identify and resolve problems all the way up the Java™ platform stack, including in your own code.

Agenda

Introduction

Review of the IBM Reliability, Availability and Serviceability (RAS), and Problem Determination (PD) Technology

- Functionalities in 1.4.2
- Current PD Tools

The new IBM RAS and PD Technology

- Improvements in 5.0
- Roadmap for 6.0
- PD Tooling Roadmap

Dump Toolkit and Framework for Java (DTFJ)

- Overview and Architecture
- How to get started
- DTFJ View

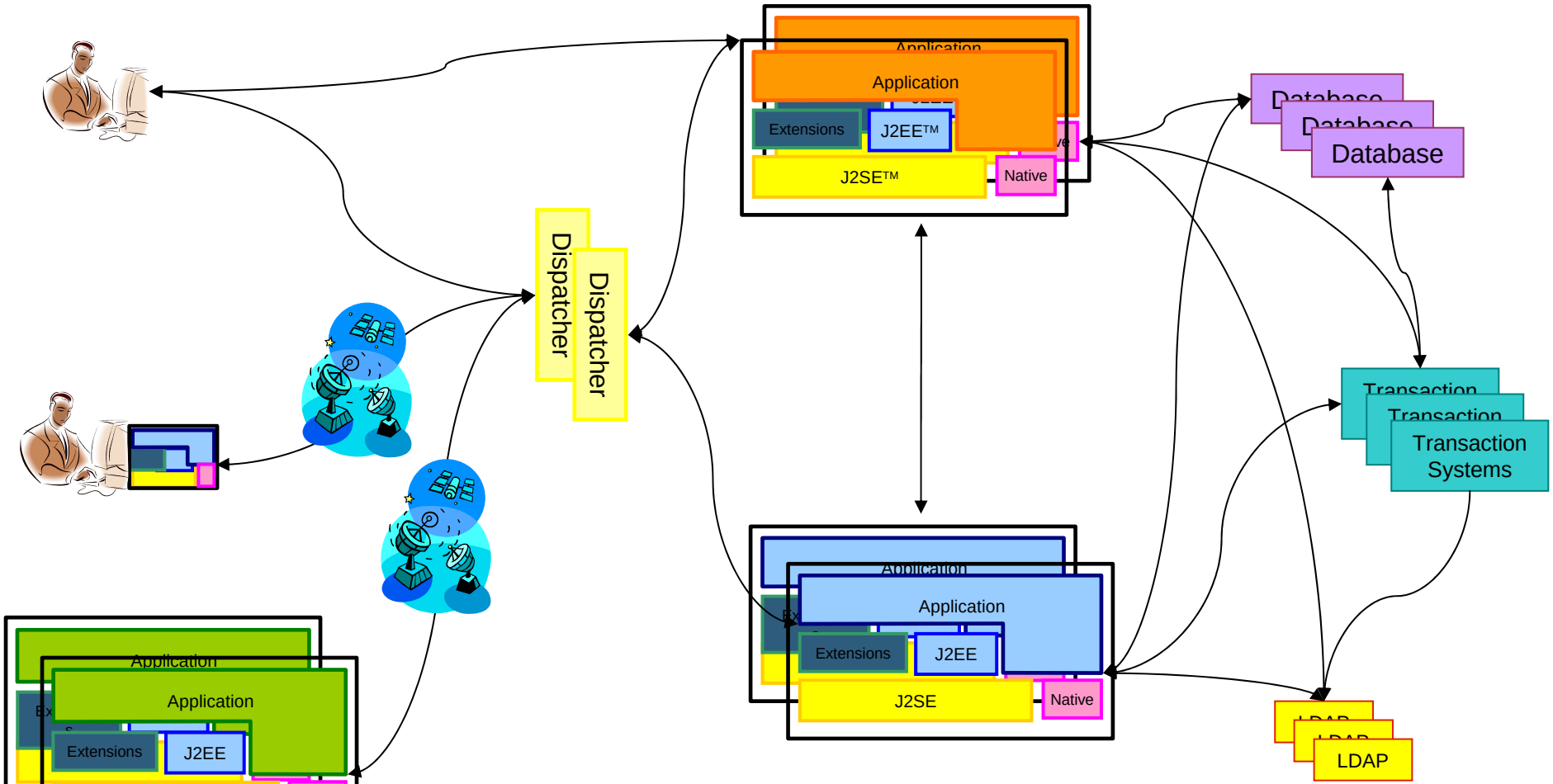
Demo

INTRODUCTION

Why Do We Want to Solve Problems Quickly?

- It means a lower Total Cost of Ownership (TCO) for you and your customers!
- The projected software warranty cost for the software industry in 2006 is tens of billion dollars!
- 4 out of 5 IT dollars spent on operations, maintenance, and minor enhancements
- Problem determination and resolution has become a daunting task as more of today's solutions involve complex collections of products and applications deployed in heterogeneous environments
- Developing and deploying new solutions get delayed by maintenance of diverse existing systems
- Typically 20% of problems experienced are new defects and 80% are non-defects
- 25–50% of time is spent in problem determination and resolution
- The skills needed to do manual cross-product problem determination are scarce and expensive

Introduction
Typical Production Environment



Introduction
Business Challenges

- Multi-vendor Distributed Environment (Software and Hardware)
- Most of the problems are only encountered in a production environment and not in test
- Cannot afford any performance loss when capturing diagnostic data
- Requires automatic diagnostic data collection and analysis
- Must avoid deploying a “custom-instrumented” Java platform

Introduction
Business Challenges

- There is an urgent need to reduce the TCO of Java platform-based solutions
- There is an urgent need to provide the developers and the clients with the reliability, availability and serviceability (RAS), and problem determination (PD) infrastructure and tools that will help to identify and correct problems or undesired behaviour in the Java platform solutions stack

Review of IBM RAS/PD Technology

Agenda

Introduction

Review of the IBM Reliability, Availability and Serviceability (RAS), and Problem Determination (PD) Technology

- Functionalities in 1.4.2
- Current PD Tools

The new IBM RAS and PD Technology

- Improvements in 5.0
- Roadmap for 6.0
- PD Tooling Roadmap

Dump Toolkit and Framework for Java (DTFJ)

- Overview and Architecture
- How to get started
- DTFJ View

Demo

IBM RAS/PD Technology Functionalities in 1.4.2

- JVM™ Software Trace Facility
  - Componentized high performance trace engine
  - Comprehensive set of trace in all the JVM software components
  - Tracing to thread local buffers
  - Trigger Tracing
  - Back-stack tracing
- First Failure Data Capture—FFDC
  - “Flight Recorder” tracing
  - System dump on all platforms
  - JVM software-initiated system dumps
  - Componentized Cross-platform dump formatter
- Improved FFDC for memory leak detection and analysis
  - Enhanced Java code heap dump
  - Enhanced native memory traces

IBM RAS/PD Technology Functionalities in 1.4.2

- Component Level Diagnostics
- JVM Software RAS Interface
  - External interface to the JVM software trace facility and Component level diagnostics
  - Trace configuration/control on-demand
  - Diagnostic reports/requests on-demand
  - Error Injection
- Light weight JVM Software Monitoring
  - Monitoring of JVM software health indicators
  - Monitoring of JVM software behavior
- JVM Software Monitoring Interface
  - External interface to light weight JVM software monitoring
  - Monitoring configuration/control on demand

IBM RAS/PD Technology Functionalities in 1.4.2

- Cross Platform dump formatter
  - Dump analysis tool to analyze crash dumps
  - Looks inside the JVM software
  - Looks at state of Java code heap
  - Inspects loaded classes and methods
  - Traces through the stack of the JVM software threads
  - Checks lock states, etc
- Improved FFDC for memory leak detection and analysis
  - Enhanced Java code heap dump (example correlation of multiple heap dumps)
  - Enhanced native memory traces
  - Automatic triggering of data capture ahead of total memory exhaustion
- Management and control of diagnostic data capture of IBM VM for the Java platform
  - No JVM restart required
  - Fine control of JVM software internal tracing
  - Java code heap dump, thread dump, etc
- Management control of platform diagnostic data capture
  - Request OS statistics and diagnostic data on current health of JVM software

IBM RAS/PD Technology Functionalities in 1.4.2

- First/Second Failure Data Capture:
  - OOM: heapdump, verbose GC
  - Crash: system dump, java dump
  - Hang: deadlock detector in javadump
  - Performance verbose GC, JVM software trace, method trace
  - Exceptions method trace

IBM RAS/PD Technology
Current PD Tools

- Verbose GC:
  - PMAT, GC Collector, GC Analyzer
- Heapdump
  - HeapAnalyzer, HeapRoots, FindRoots, MDD4J
- Javadump
  - ThreadAnalyzer
- System dump
  - JFormat/JDmpView

New IBM RAS/PD Technology

Agenda

- Introduction
- Review of the IBM Reliability, Availability and Serviceability (RAS), and Problem Determination (PD) Technology
  - Functionalities in 1.4.2
  - Current PD Tools
- The new IBM RAS and PD Technology
  - Improvements in 5.0
  - Roadmap for 6.0
  - PD Tooling Roadmap
- Dump Toolkit and Framework for Java (DTFJ)
  - Overview and Architecture
  - How to get started
  - DTFJ View
- Demo

IBM RAS/PD Technology Improvements in Java 5.0 Platform

- Re-engineering of Trace Engine:
  - "Flight Recorder" for First Failure Data Capture (FFDC)
    - Traced to in-memory buffer
    - Traces key VM trace points on per thread basis
    - Separate buffer for GC data
    - Buffers dumped on error scenarios or user request
  - Method Trace Improvements
    - Trace methods entry/exit with parameters
    - Trace works with JIT on or off

IBM RAS/PD Technology Improvements in Java 5.0 Platform

- Re-engineering of Dump Facilities:
  - Extended range of dump triggers
    - From 3 triggers to 14
    - Dump events extended by the use of filters
  - User defined dump labels
    - Ability to include: time, date, pid, uid, jre info
  - Ability to configure number of dumps generated
  - Ability to execute tool on dump event

IBM RAS/PD Technology Improvements in Java 5.0 Platform

- Re-engineering of JIT PD options
  - Consolidation of options into -Xjit:
  - Ability to set compilation levels for methods
  - Ability to trace compilation of individual methods
    - Does not require a debug library

IBM RAS/PD Technology Improvements in Java 5.0 Platform

- JNI Code Validator (-Xcheck:jni)
  - Enables addition checks on Java Native Interface (JNI) code at runtime:
    - Provides errors, warnings and advice level statements
    - Configurable according to level of information required
    - Checks for 20 commonly occurring JNI coding mistakes
  - Includes -Xcheck:jni:trace option
    - Outputs details of every JNI native and function call

IBM RAS/PD Technology Improvements in Java 5.0 Platform

- Introduction of the Dump Toolkit and Framework for Java (DTFJ) API
  - A Java technology API for accessing JVM software data from system dumps
  - Removes need for tools writers to:
    - understand system dump formats
    - understand VM implementation specifics
- System Dump Analytic Tool
  - DTFJView

IBM RAS/PD Technology Roadmap for Java 6.0 Platform

- Enhance dump agents to collect "Must Gather" data
- Create javadump for more exception types
- Allow trigger trace to support dynamic options
- Allow time triggered/delayed dumps
- Extend DTFJ to act on more data formats and live processes
  - javadump, heapdump, verbose GC, GC trace

Diagnostic Tooling Roadmap

- Deployment of DTFJ as a tooling API
- Extend DTFJ API for Applications
- Creation of Scenario-based Tooling
- Integration with IBM Support Assistant

DTFJ—Dump Toolkit and Framework for Java

Agenda

Introduction

Review of the pre-Java 5 IBM Reliability, Availability and Serviceability (RAS), and Problem Determination (PD) Technology

- Functionalities in 1.4.2
- Current PD Tools

The new Java 5 IBM RAS and PD Technology

- Improvements in 5.0
- Roadmap for 6.0
- PD Tooling Roadmap

Dump Toolkit and Framework for Java (DTFJ)

- Overview and Architecture
- How to get started
- DTFJ View

Demo

Dump Toolkit and Framework for Java (DTFJ)—Motivation

- Lots of different RAS artifacts, no common tooling to introspect
  - System dumps (core files, MiniDumps)
  - Heap dumps (.phd files)
  - javacore files (javacore…txt files)
- Many good reasons for different file formats
- Historical tooling too JVM software-specific
  - jformat, jcore, kca

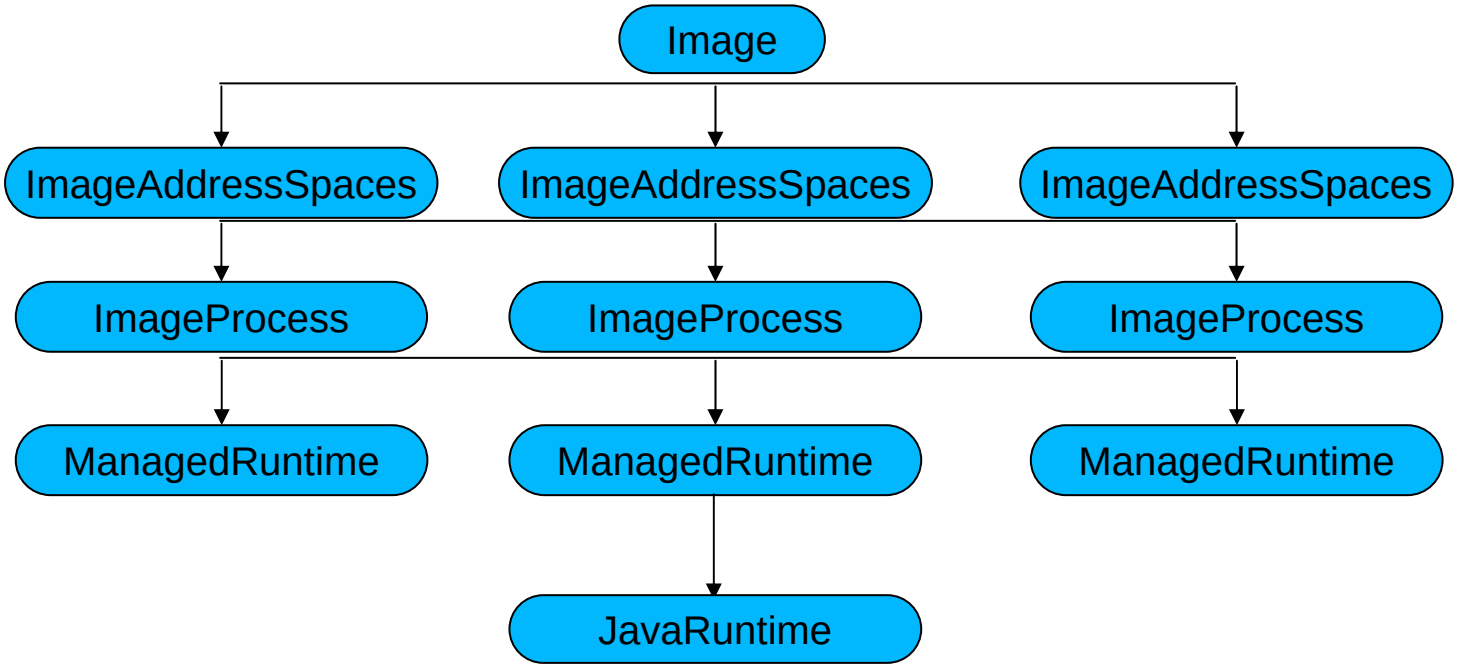
DTFJ—Overview

- Java technology API, not a fixed purpose tool
- Layered interface independent of runtime implementation
  - Cross Platform
  - Cross VM
  - Language neutral
- Base extensions understand and introspect on IBM VM data structures
  - Heaps
  - Objects
  - Threads
  - Monitors…

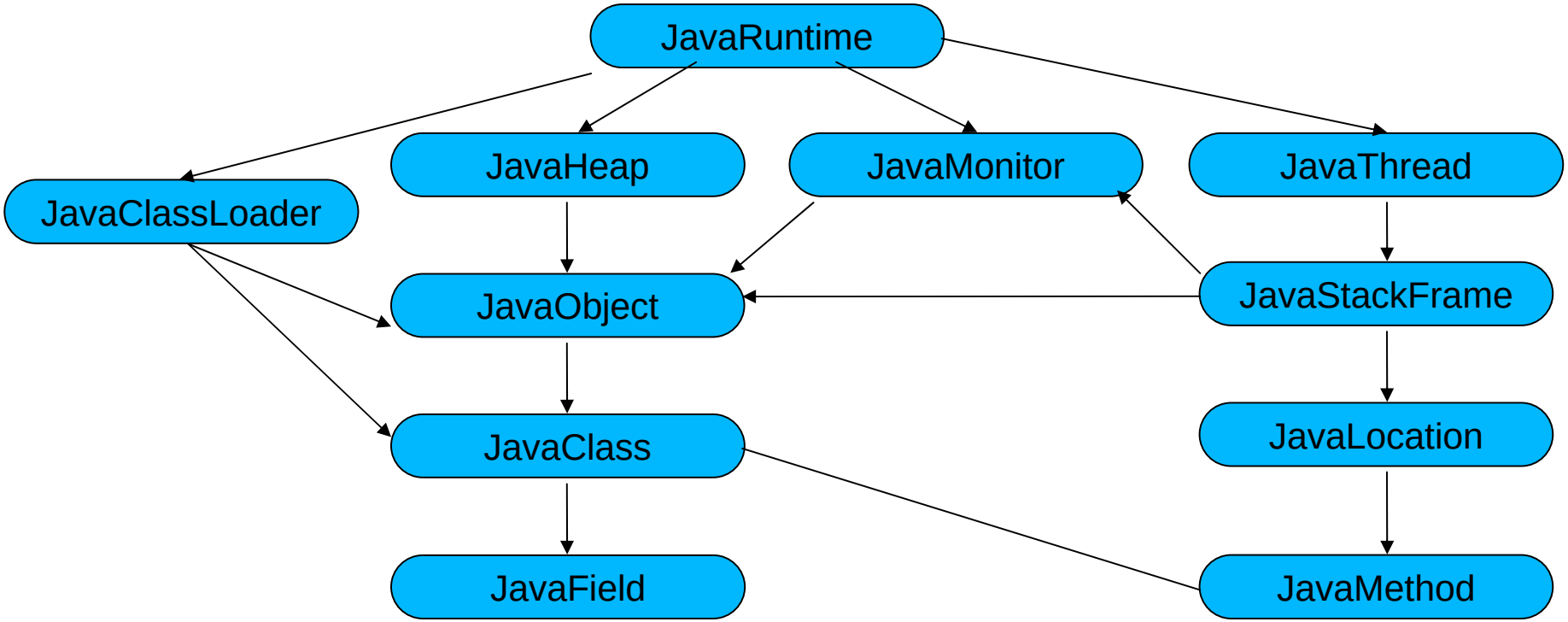
What Is DTFJ Like?

- A set of Java technology APIs, influenced by
  - JVM Tool Interface
  - Reflection
- Loosely based on a hierarchical view of a JVM software process
  - High-level objects contain iterators to examine increasingly specific components

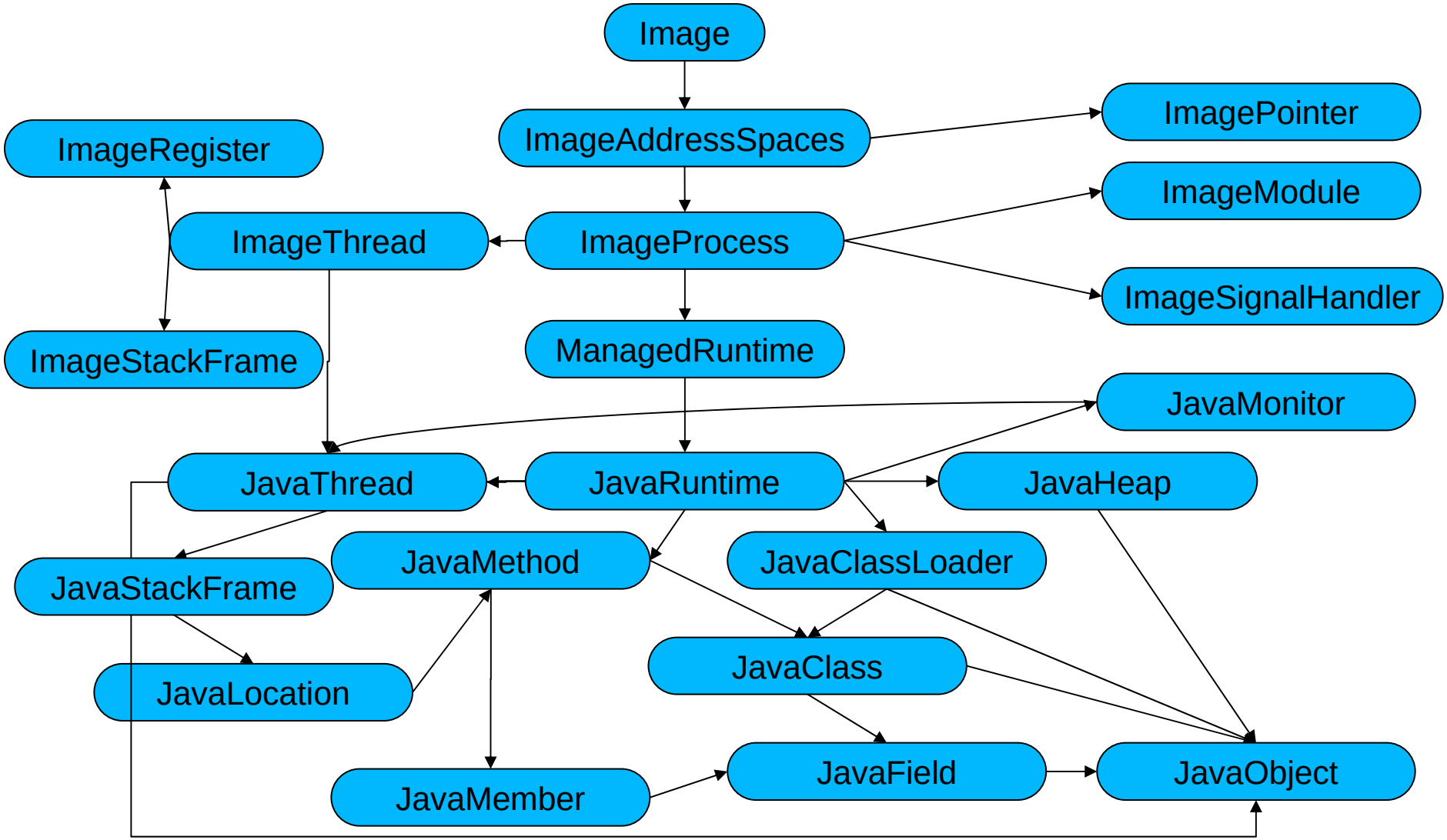
The DTFJ View of a JVM Software Process



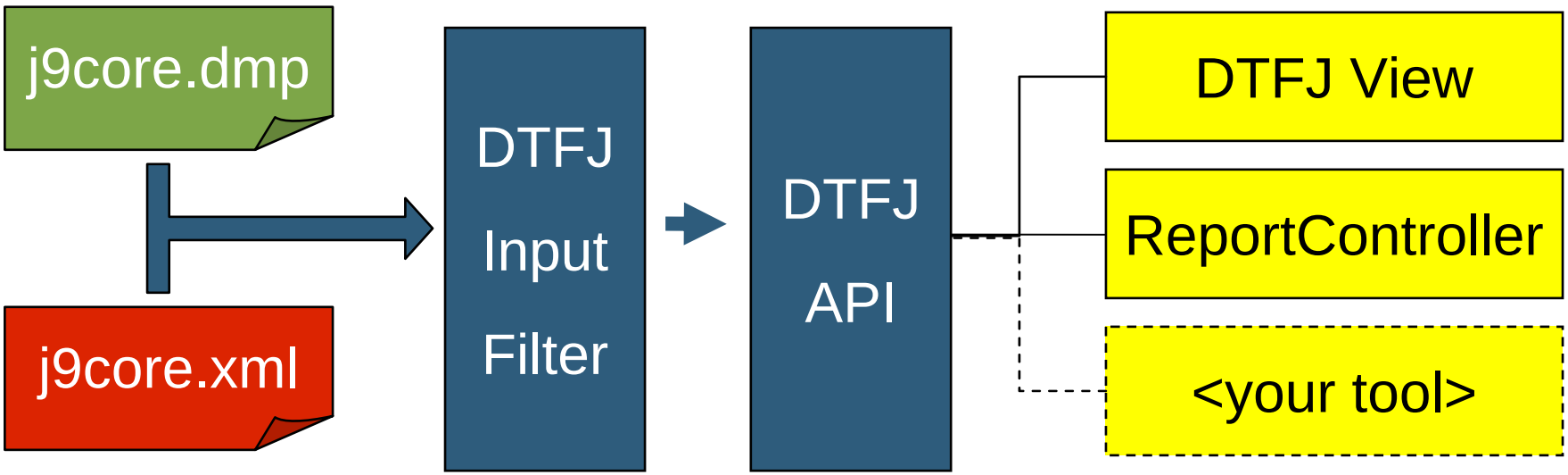
The DTFJ View of a JVM Software Process



The DTFJ View of a JVM Software Process



Dump Toolkit and Framework for Java (DTFJ)



How to Get Started With DTFJ?

- Get an Image Object
  - There is a ImageFactory class to do this
    - Different Image implementations
- Everything subsequent is based on iterators
  - Image.getAddressSpaces()
  - AddressSpace.getProcesses()
  - etc…

DTFJ—Base Tooling

- Build diagnostic tools capable of inspecting a process image
- Extract information about system
  - Physical memory, CPU type and count, system type
- Extract process information
  - Threads, libraries and symbols, command line, environment
  - Native stack traces and register contents
- Extract Java VM information
  - Class loaders, threads, monitors, heaps, objects, trace buffers
  - Java thread state: stack trace, priority, native thread, j.l.Thread
  - Monitor state: owner, waiters
  - Class info: class loader, name, inheritance, fields, methods, etc
  - Object info: class, size, hashcode, id, field values

DTFJ—Extended Tooling

- Application specific tooling
  - You write extensions for your app
- Middleware tooling
  - Web Applications server
  - Cache Servers
- Layered approach allows proper separation of concerns

DTFJ Packages

- com.ibm.dtfj.image
  - Entities which are common to process images, with or without managed runtimes
- com.ibm.dtfj.runtime
  - Generic managed runtime entity
- com.ibm.dtfj.java
  - Entities which apply to the Java technology-managed runtime

Example: Count Objects by Class

```
public static void main(String[] args) throws Exception {
     Image theImage = (Image) ImageFactory.getInstance(args[0]);
     ImageAddressSpace currentAddressSpace = ((ImageAddressSpace)theImage.getAddressSpaces().next());
     ImageProcess currentProcess = currentAddressSpace.getCurrentProcess();
     JavaRuntime currentRuntime = (JavaRuntime) currentProcess.getRuntimes().next();
     Map<String,Long> objectCountMap = new HashMap<String,Long>();
     Iterator allHeaps = currentRuntime.getHeaps();
     while(allHeaps.hasNext()) {
          countObjects((JavaHeap)allHeaps.next(),objectCountMap);
     }
     for (String objectClassName : objectCountMap.keySet()) {
          System.out.println(objectClassName + " occurs " + objectCountMap.get(objectClassName));
     }
}
```

Example: Count Objects by Class

```
private static void countObjects(JavaHeap currentHeap, Map<String, Long> objectCountMap)
     throws Exception{

     Iterator currentHeapObjects = currentHeap.getObjects();

     while(currentHeapObjects.hasNext()) {

          JavaObject currentObject = (JavaObject)currentHeapObjects.next();

          String objectClassName = currentObject.getJavaClass().getName();

          long objectCount = 0;

          if (objectCountMap.containsKey(objectClassName)) {

               objectCount = objectCountMap.get(objectClassName);

          }

          objectCountMap.put(objectClassName, objectCount + 1);

     }

}
```

DTFJ View

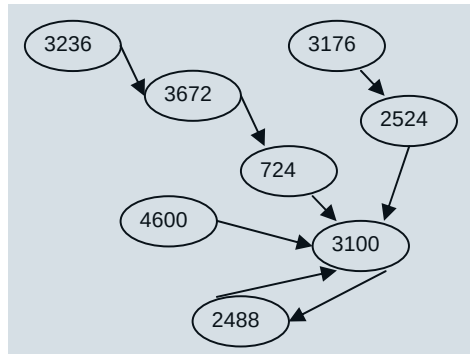
- A command-line prompt-driven tool designed to analyze core dump files using the DTFJ API
  - Look at the contents of objects
  - Read arbitrary sections of memory
  - Get a lot of other information from a core dump file
  - Analyze problems
    - e.g. detect and display deadlock information
- An extensible analysis API
- Driven by a GDB like command set

DTFJ View—Command Set

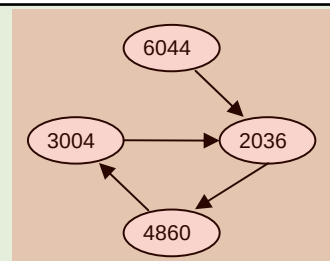
- `hexdump`
- `info [cls|thread|ls]`
- `x/[j|x|gd|wd|k]`
- `deadlock`
- `help`
- `Heap dumps, trace, synonyms, execute`

DTFJ View—Deadlock Analysis

```
> deadlock

      deadlocks for runtime #0 - "J2RE 1.5.0 IBM J9 2.3 Windows XP x86-32  (JIT enabled) J9VM - 20051004_03470_lHdSMR JIT  - 20051004_1800_r8
      GC   - 20050930_AA"

      deadlock branch(es):

       3176 (0x8bcec0) => 2524 (0x8bcea0) => 3100 (0x8bcd80)

       3236 (0x8bcf00) => 3672 (0x8bcee0) => 724 (0x8bcdc0) => 3100 (0x8bcd80)

       4600 (0x8bcda0) => 3100 (0x8bcd80)

      deadlock loop:

       2488 (0x8bcd60) => 3100 (0x8bcd80) => 2488 (0x8bcd60)
```




```
      deadlock branch(es):

       6044 (0x8bd060) => 2036 (0x8bd040)

      deadlock loop:

       2036 (0x8bd040) => 4860 (0x8bd080) => 3004 (0x8bd0a0) => 2036 (0x8bd040)
```



LIVE DEMO

Summary

- 1.4.2
  - FFDC dumps for OutOfMemory, etc..
- 5.0
  - FFDC Trace
  - New Dump triggers
- Java 6.0 Platform
  - Easier ‘Must Gather’
  - Focus on automation
- Dump Toolkit and Framework for Java

Q&A

Flavio Bergamaschi—IBM Hursley Lab, UK

Trent Gray-Donald—IBM Ottawa Lab, Canada

Chris Bailey—IBM Hursley Lab, UK

Resources

- http://www.ibm.com/developerworks/java/jdk
- http://www.alphaworks.ibm.com/java
- http://www.ibm.com/support

Contact info:

Flavio@uk.ibm.com
Trent_Gray-Donald@ca.ibm.com
Baileyc@uk.ibm.com

© Copyright IBM Corporation 2006. All rights reserved.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, the e-business logo and other IBM products and services are trademarks or registered trademarks of the International Business Machines Corporation, 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.

All other trademarks, company, products or service names may be trademarks, registered trademarks or service marks of others

Disclaimer: NOTICE – BUSINESS VALUE INFORMATION IS PROVIDED TO YOU 'AS IS' WITH THE UNDERSTANDING THAT THERE ARE NO REPRESENTATIONS OR WARRANTIES OF ANY KIND EITHER EXPRESS OR IMPLIED. IBM DISCLAIMS ALL WARRANTIES INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IBM DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE, VALIDITY, ACCURACY OR RELIABILITY OF THE BUSINESS BENEFITS SHOWN.. IN NO EVENT SHALL IBM BE LIABLE FOR ANY DAMAGES, INCLUDING THOSE ARISING AS A RESULT OF IBM'S NEGLIGENCE.WHETHER THOSE DAMAGES ARE DIRECT, CONSEQUENTIAL, INCIDENTAL, OR SPECIAL, FLOWING FROM YOUR USE OF OR INABILITY TO USE THE INFORMATION PROVIDED HEREWITH OR RESULTS EVEN IF IBM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE ULTIMATE RESPONSIBILITY FOR ACHIEVING THE CALCULATED RESULTS REMAINS WITH YOU.

Full-scale Java™ Platform Diagnostics for Production Environments

Flavio Bergamaschi—IBM Hursley Lab, UK

Trent Gray-Donald—IBM Ottawa Lab, Canada

Chris Bailey—IBM Hursley lab, UK

TS-3881