
Worldwide
Developers
Conference


Mac OS Runtime
for Java
Peri Frantz
MRJ Engineering
Manager

Mac OS Runtime for Java

MRJ Goals and Objectives

- Robust and stable
- Performance
- 100% Java — Always!
- The Macintosh Advantage
  - Bind and run applications and applets
  - Access the Mac OS APIs from Java
- Release at same time as Win/Solaris
  - MRJ 1.0 9 months behind JDK 1.0.2
  - MRJ 2.0 6 months behind JDK 1.1
  - MRJ 2.x in sync with JDK 1.2

Mac OS Runtime for Java

. . . and the JavaSoft JDKs

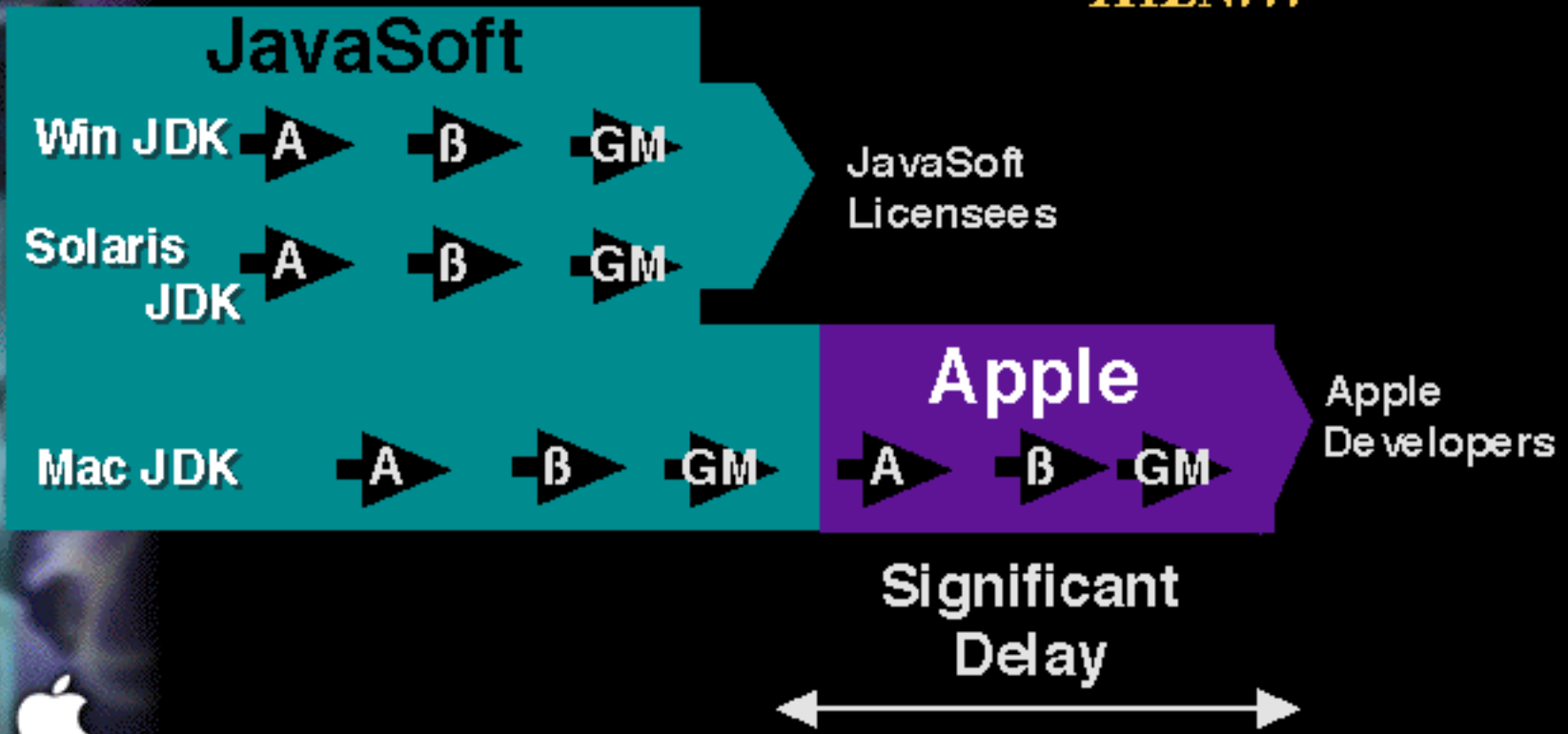
MRJ = JDK + Apple Extensions

Apple	JavaSoft
MRJ 1.0	JDK 1.0.2 +JManager, JBindery
MRJ 1.5	JDK 1.0.2 +JIT, MRJToolKit
MRJ 2.0	JDK 1.1 JDK 1.1.1 JDK 1.1.2 +JDirect

Mac OS Runtime for Java

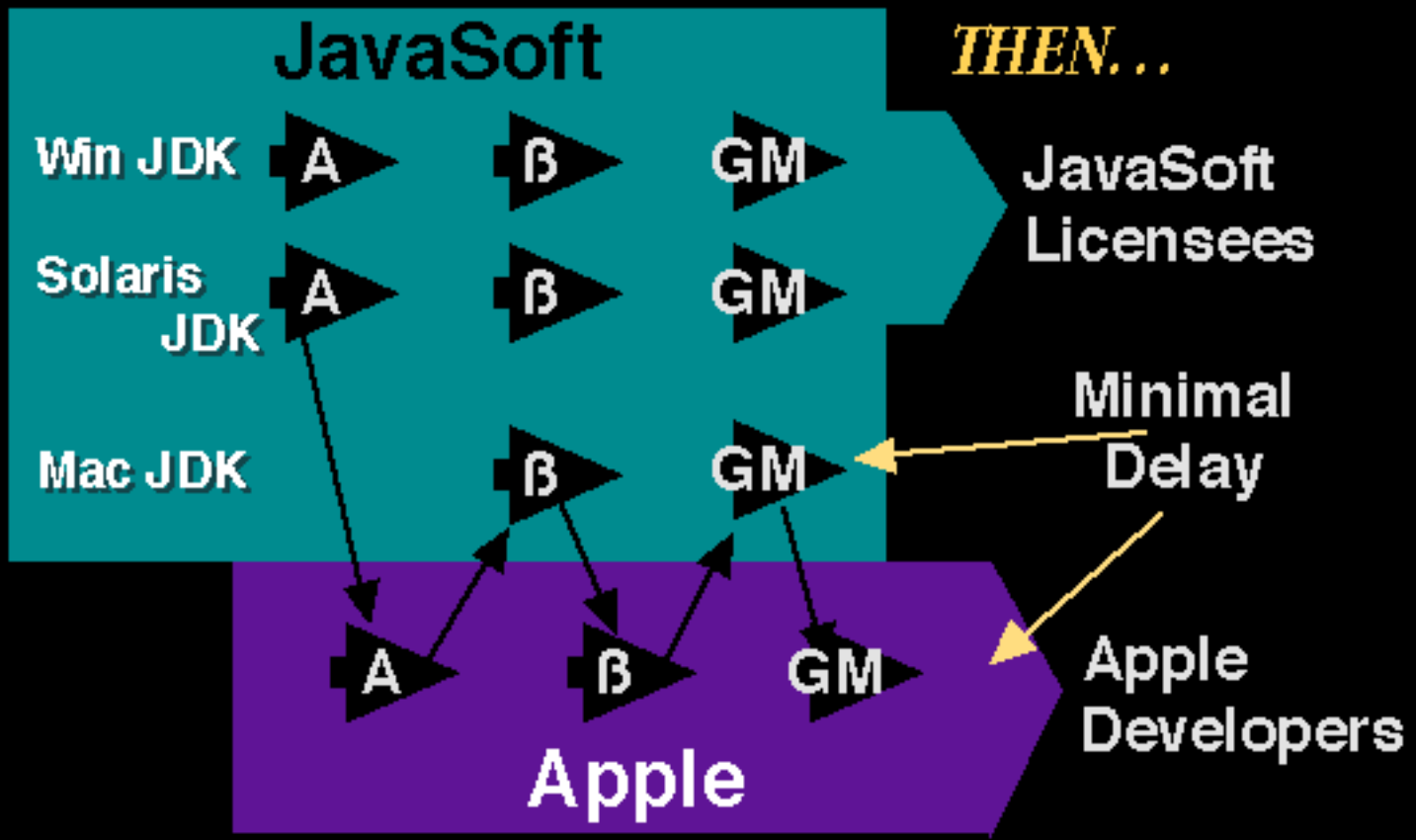
The Apple — JavaSoft Relationship

THEN...



Mac OS Runtime for Java

The Apple—JavaSoft Relationship




MRJ and
Applications
Steve Zellers
MRJ 1.0/1.5 Tech Lead

Cross Platform Java on the Mac

Extending Java for the Mac

- JBindery
  - Build and run Java applications
- Virtual File System (VFS)
  - Package Java applications
- MRJToolKit
  - Extend Java applications
  - Mac-specific functionality
  - File Utils • Application Utils • MenuUtils

Cross Platform Java on the Mac

JBindery

- Encapsulates the "/usr/local/java/bin/java" command line arguments
  - "main" class
  - "main" parameters
  - classpath
  - properties
- Produces a double clickable application or "AppTag"

Cross Platform Java on the Mac

Virtual File System (VFS)

- Flattens a folder hierarchy into a read-only .zip file
- classpath can point "into" this hierarchy
- Auxiliary files (images, html) can be accessed like any other file

MRJ 1.x Zip

MRJ 2.0 JAR

Cross Platform Java on the Mac

MRJ ToolKit

- MRJ 1.5 and later only
- File Utils: Mac File system extras
  - set/get type/creator
  - findFolder
  - findApplication

Cross Platform Java on the Mac

MRJ ToolKit

- Application Utils: “High Level User Events”
  - handleAbout()
  - handleOpen(File)
  - handleQuit()
- MenuUtils: Set command keys for menu items


ImageViewerDropon
Demo

Hosting Java in a Macintosh Application

JManager

- High-level APIs in a shared library
- Gives applications the ability to host Java content
- Interface, documentation and samples in the SDK

Hosting Java in a Macintosh Application

JMSession

- Abstraction for Java runtime environment
- Handles memory allocation in temp memory
- `JMOpenSession()` to create `JMCloseSession()` to dispose
- Feed it time with `JMIdle()`

Hosting Java in a Macintosh Application

JMAWTContext

- Dispatcher for Macintosh events
- Requests “frames” from the embedding application through callbacks
- Requests “MenuIDs” for menus created by AWT
- Reports exceptions

Hosting Java in a Macintosh Application

JMFrame

- An imaging substrate
- Embedding application supplies callbacks
- Respond to requests

```
fSetupPort()
fRestorePort()
```

- Give it events

```
JMFrameClick()
JMFrameUpdate()
```


MRJ Native Methods
Patrick Beard
Java Runtime Specialist

Native Methods

Reasons for use

- Provides direct access to hardware/OS
- Improves performance
- Adapts legacy software
- Implemented in C or C++

Native Methods

Drawbacks to use

- Reduces portability
- Decreases robustness
- Significant implementation effort
- Implemented in C or C++

Native Methods

Implementation Strategies

- Stubs (JDK 1.0.2)
  - Tied to specific VM, javah, <interpreter.h>
- JRI (Navigator, MRJ 1.0 and beyond)
  - Insulates VM details, rudimentary API
- JNI (JDK 1.1, MRJ 2.0 and beyond)
  - Extends JRI
- JDirect
  - Direct access to C shared libraries

JRI Native Methods

The Environment: JRIEnv

- One per Java thread
- Explicit exception handling
- Struct of JRI entry points
  - `GetClassID()`
  - `GetMethodID()`
  - `GetFieldID()`

JRI Native Methods

Member Signatures

```
methodSig   ::= '(' argSig* ')' resultSig
argSig      ::= classSig | arraySig |
                simpleSig
resultSig   ::= argSig | 'V'
classSig    ::= 'L' className ';'
arraySig    ::= '[' argSig
simpleSig   ::= 'B' | 'Z' | 'S' | 'C' |
                'I' | 'F' | 'J' | 'D'
fieldSig    ::= argSig
```

JRI Native Methods

Native Class Example

```
// Lightweight Synchronization Example

package com.acme;
class AtomicAdder {
    public native static int
    atomicAdd(int[] numbers, int offset);
}
```

JRI Native Methods

Registering Native Methods

```
// Using C++ JRI Bindings

JRIEnv* env = JRI_GetCurrentEnv();
JRIClassID classID =
    env->FindClass("com/acme/AtomicAdder");
char* methodNames[] =
    { "atomicAdd([II)I", NULL };
void* methodProcs[] =
    { &myAtomicAdd, NULL };
env->RegisterNatives(classID, methodNames,
    methodProcs);
```

JRI Native Methods

Unregistering Native Methods

```
// Removes all JRI registered
// native methods for a class

JRIClassID classID =
    env->FindClass("com/acme/AtomicAdder");
env->UnregisterNatives(classID);
```

Direct Native Methods

JDirect Advantages

- Direct access to shared libraries
  - CFM-PPC and CFM-68K
  - Win32 prototype
- Rapid prototyping
  - No stubs / no C code required
  - Java-only specification
- Simple parameter marshalling

Direct Native Methods

Accessing JDirect

```
// NativeObject is a special interface
import com.apple.NativeObject;
public class Printf implements NativeObject {
    static String[] kNativeLibraryNames = {
        "StdCLib"
    };
    native static int
        printf(byte[] format, int x, double y);
}
```

Direct Native Methods

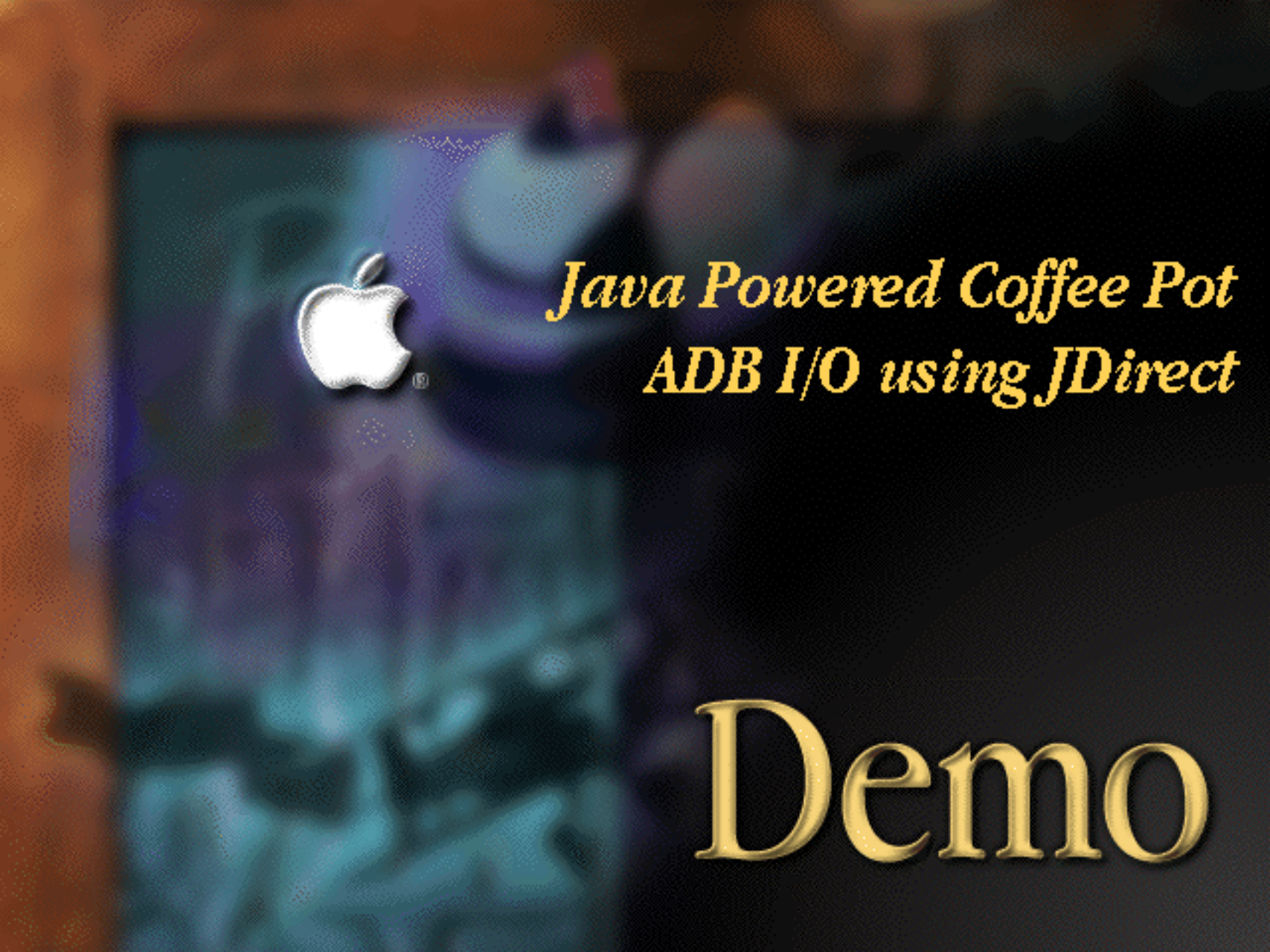
JDirect Parameter Marshalling

- Arrays passed as pointer to first element
- Objects passed as pointer to first field
- Strings must be converted manually
  - Convert to `byte[]` for ASCII
  - Pass `char[]` for Unicode

Direct Native Methods

JDirect in Action

- Callback functions
- Interrupt/IOCompletion routines
- MRJ Uses JDirect internally
  - Access to Sound Manager
  - MRJ Toolkit


Java Powered Coffee Pot
ADB I/O using JDirect
Demo

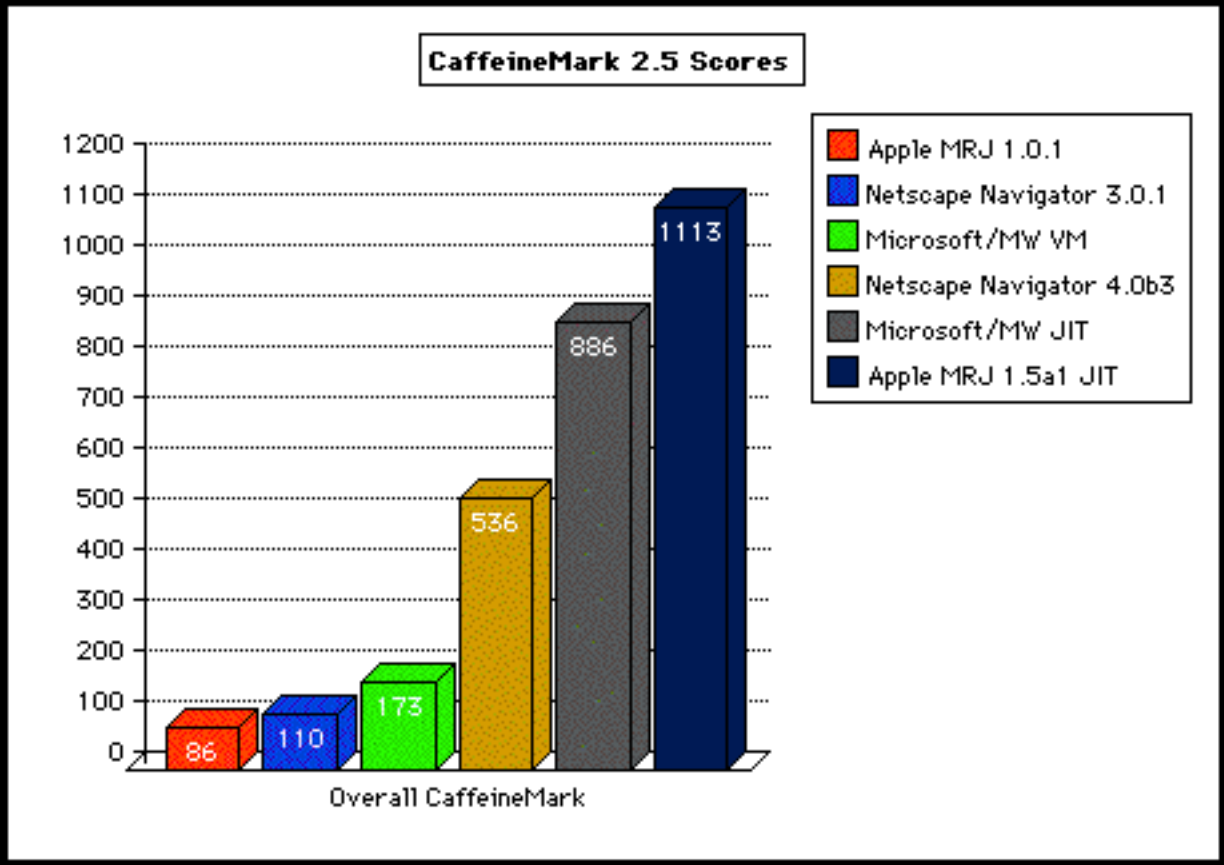

MRJ Futures
Peter Steinauer
MRJ 2.0 Project Lead

MRJ 1.5

Release: Early Summer 1997

- MRJToolkit
- Graphics enhancements
- JITC
  - PPC JITC in 1.5
  - 68K JITC in 2.0

MRJ Performance



MRJ 2.0

Release: Fall 1997

- JDK 1.1.x reference release
- AWT enhancements
- Macintosh enhancements

MRJ 2.0

JDK 1.1.x Reference Release

- Reflection / Serialization
- Security
- JDBC (database access)
- RMI (Remote Method Invocation)
- Internationalization / Localization
- JavaBeans
- JNI (Java Native Interface)

MRJ 2.0

AWT Enhancements

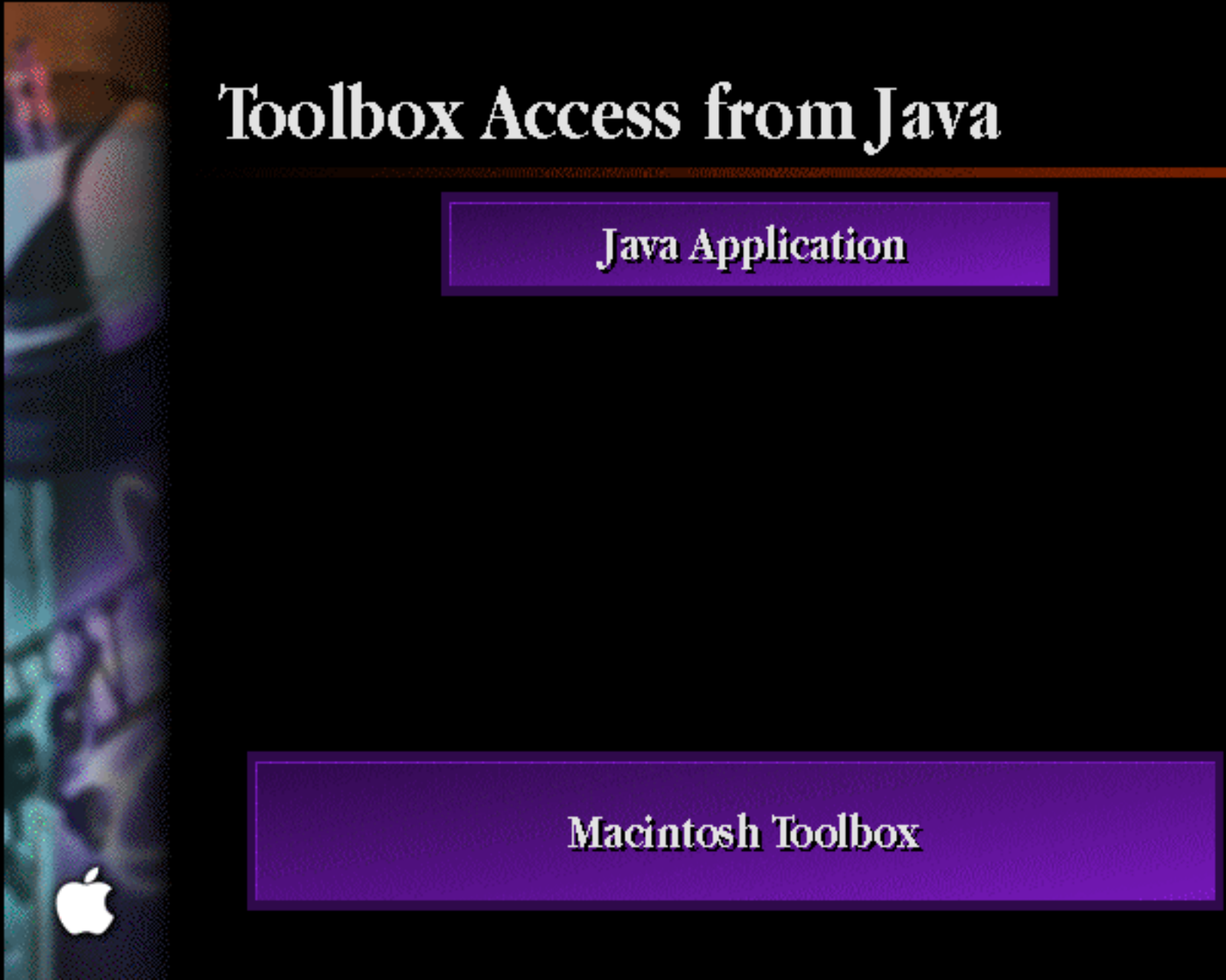
- Data transfer (Clipboard)
- Printing
- Delegation event model
- Lightweight peers

MRJ 2.0

Macintosh Enhancements

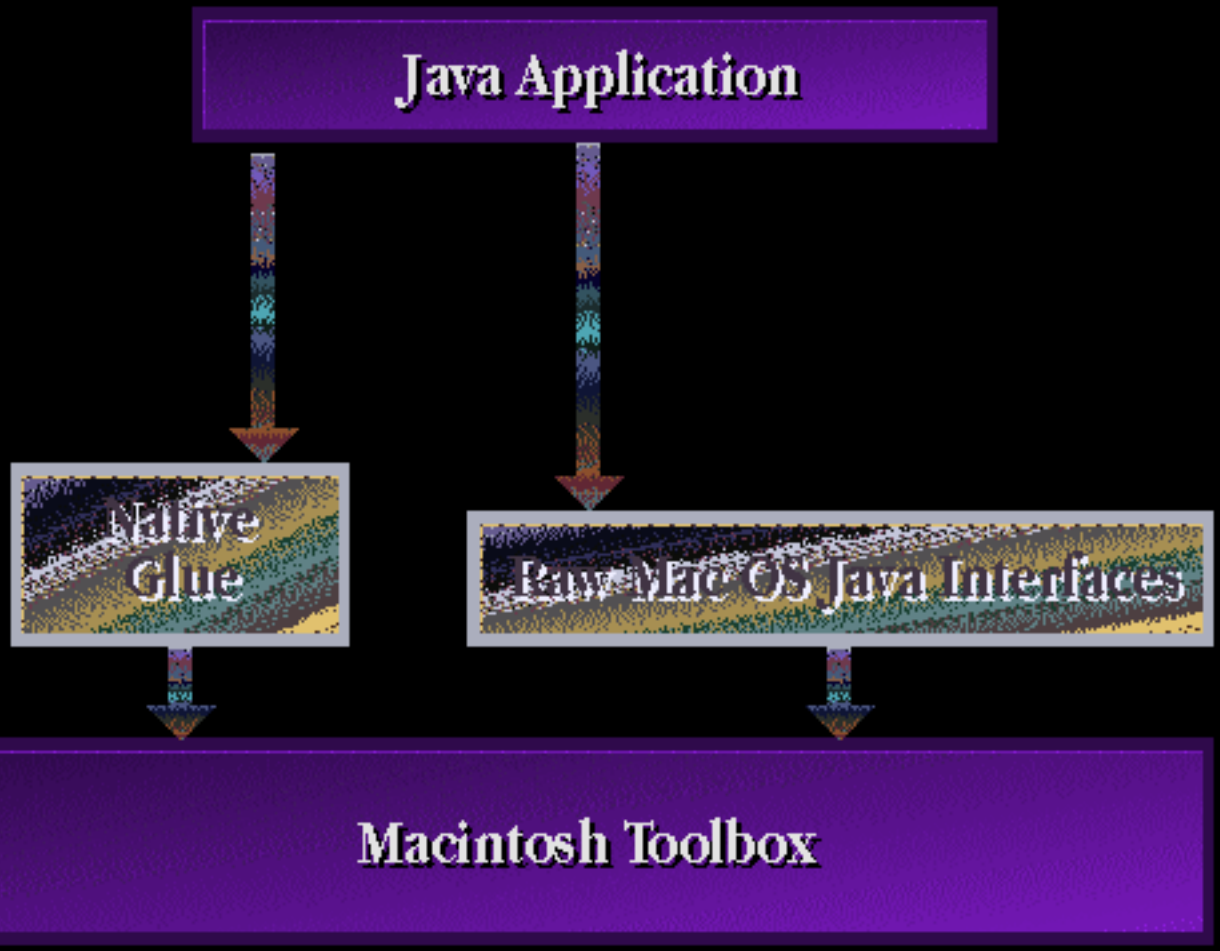
- javac / javah / javadoc droplets
- MRJToolkit enhancements
- JManager to support Unicode
- Macintosh HI improvements
- Java bindings for the Macintosh Toolbox

Toolbox Access from Java

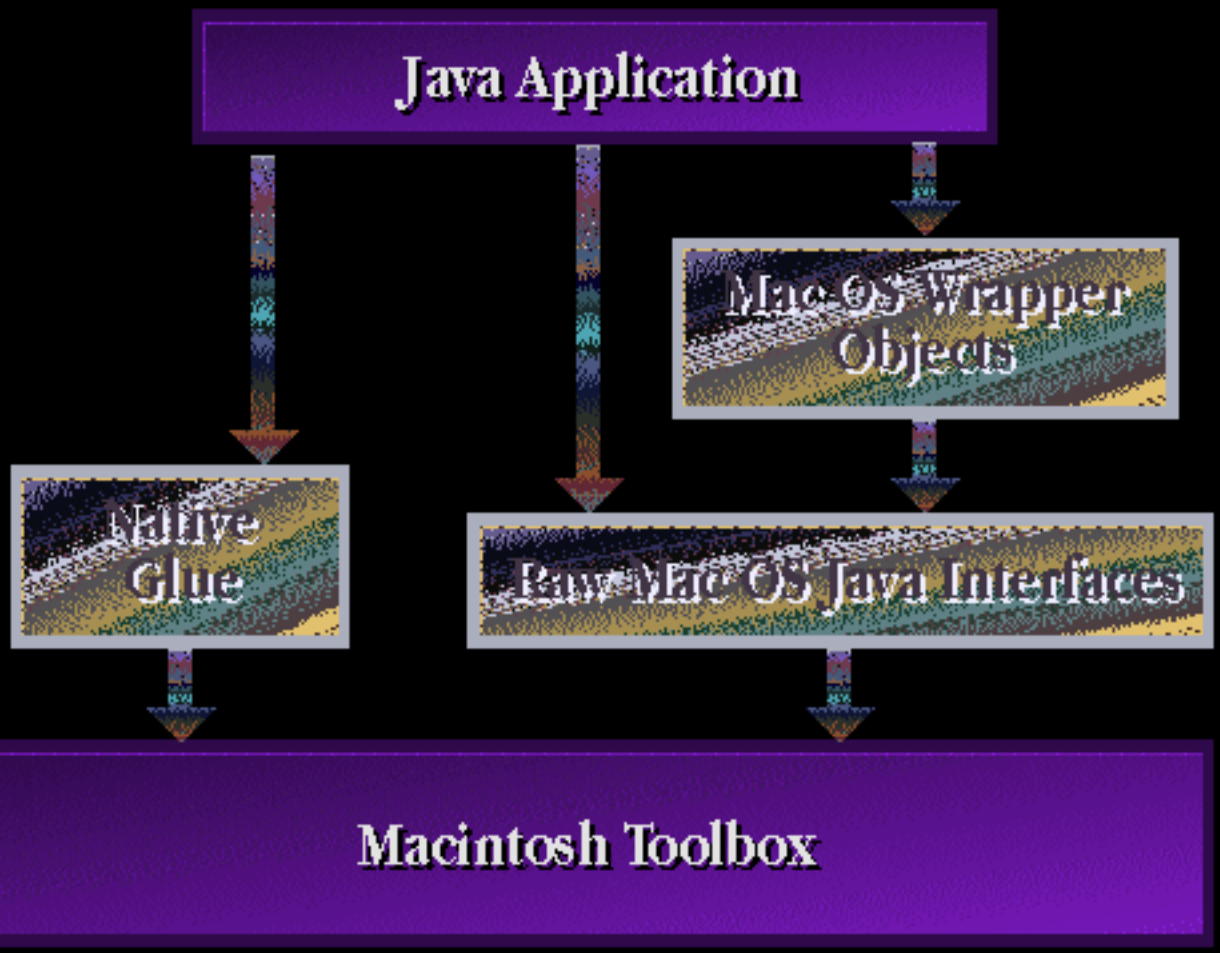


Toolbox Access from Java

Toolbox Access from Java

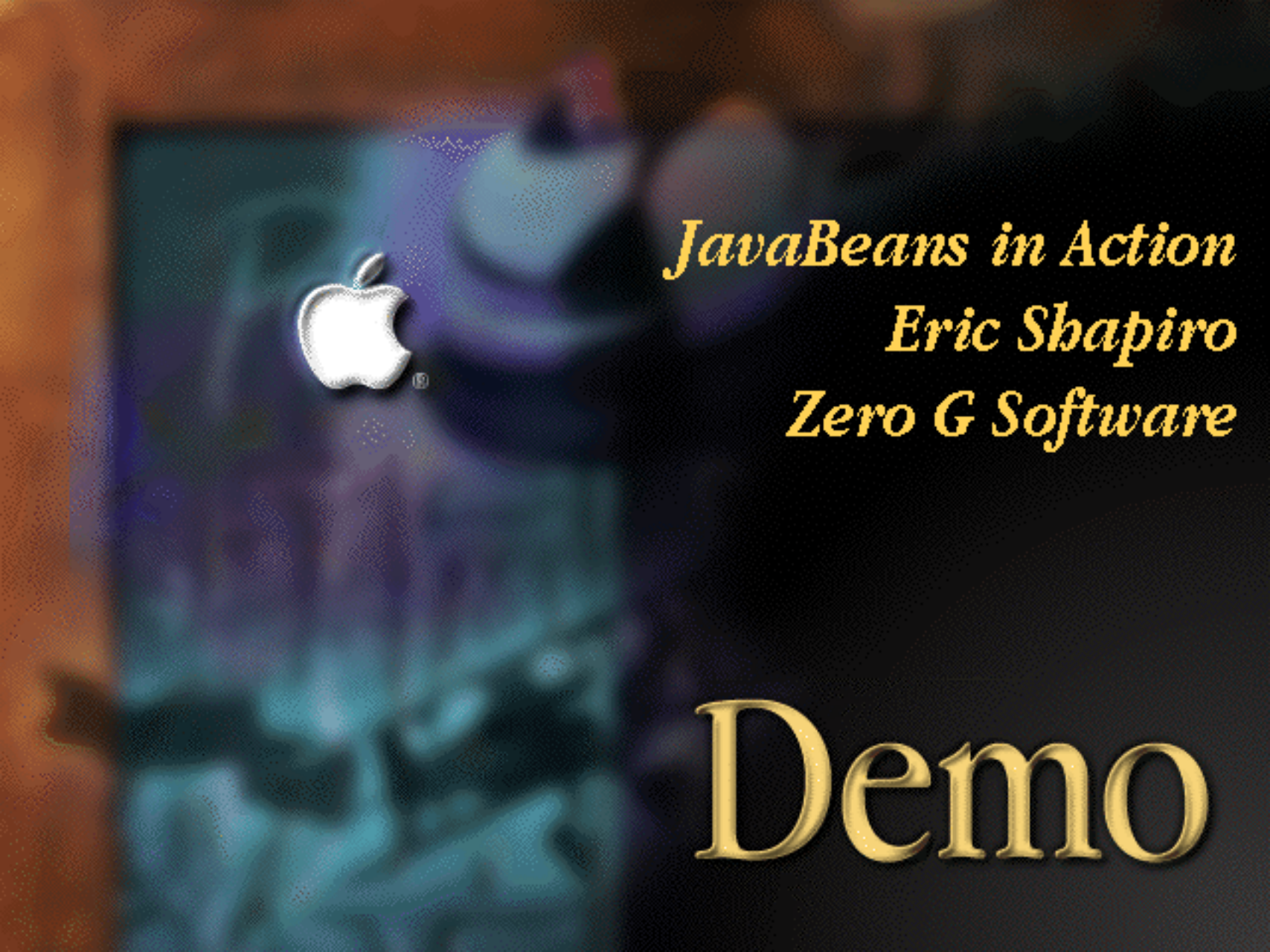


Toolbox Access from Java



Toolbox Access from Java

	Pro	Con
Pure AWT	Truly Cross Platform	No Access to Mac OS
Native Glue	Full Access to Toolbox	Requires Glue
Raw Mac OS Interfaces	Full Access to Toolbox	Not Object Based
Mac OS Wrapper Objects	Object Based, Stable	Platform Dependent


JavaBeans in Action
Eric Shapiro
Zero G Software
Demo

More About MRJ

How to Contact Us

MRJ Feedback Session	Friday, 3:10, Hall J4
Developer Discussion List	mrj-dev@adr.apple.com
Reports Bugs & Feedback	mrj_feedback@apple.com
Website	http://applejava.apple.com/
Evangelism: Shaan Pruden	pruden@apple.com


Worldwide
Developers
Conference


Worldwide
Developers
Conference