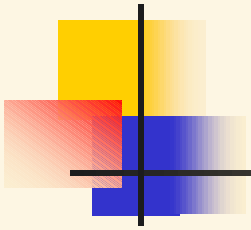
Development of Java™ Applications for Embedded Devices

Aldo Eisma

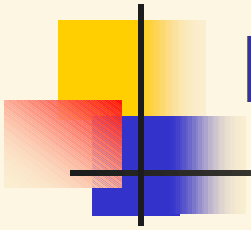

Object Technology International Inc.

Aldo Eisma@oti.com



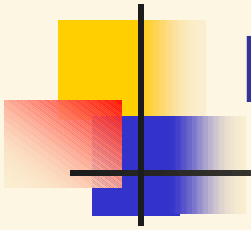
Introduction

- Examples of embedded systems
- Why Java?
- What is OTI’s role and history?



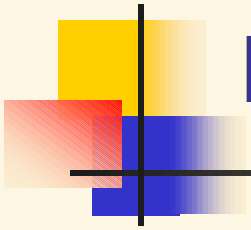
Mobile == Embedded

- Limited resources (CPU, memory, power)
- Limited outputs: small screens
- Limited input: keypad, or really tiny keyboards
- Limited connectivity (low bandwidth, expensive)



Requirements

- Smart
- Flexible / Personalizable
- Small, yet fast
- Dynamically configurable



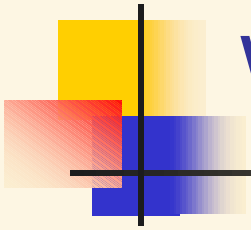
Examples of mobile devices

Resource constraint devices:

- PDAs (like Palm handhelds)
- Cell phones, pagers (including WAP)
- Automotive appliances (radio, navigation)

Very similar in their limitations:

- Industrial automation (process control)
- Set-top boxes (web-tv)

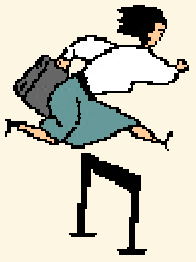


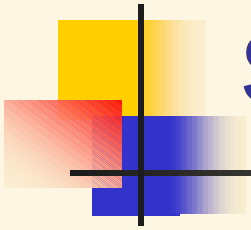
Why Java for Embedded Systems?

- Higher level of abstraction
- Java is relatively easier to master than C++
- Java is relatively secure
- Java supports dynamic loading of new classes
- Supporting object and thread creation at runtime
- Designed to support component integration & reuse
- Technologies developed with careful consideration
- Language and platform support application portability
- Support for distributed applications
- Java provides well-defined execution semantics

Java 1 Shortcomings for Embedded

- Java 1 has no concern for embedded platform issues
- “One size fits all” does not apply for embedded
  - Classes.zip of 10MB?
- Problems with Real-time aspects
- Development of embedded Java applications is not without hurdles...

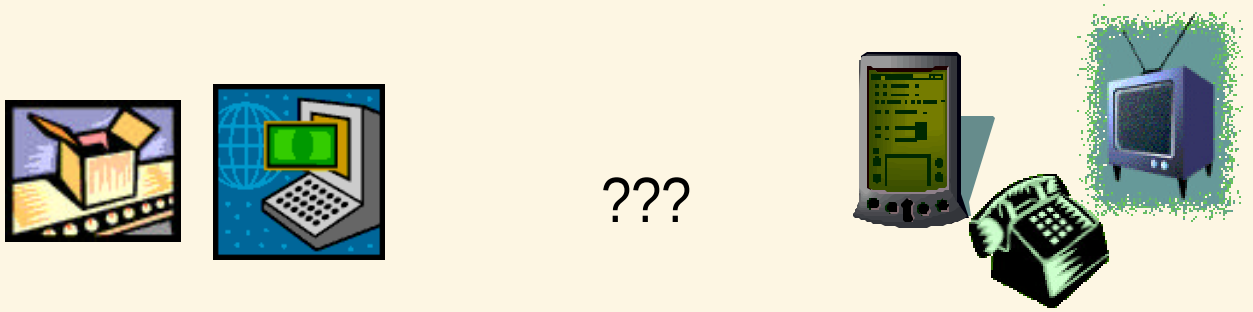




Sun’s answer: Embedded Java

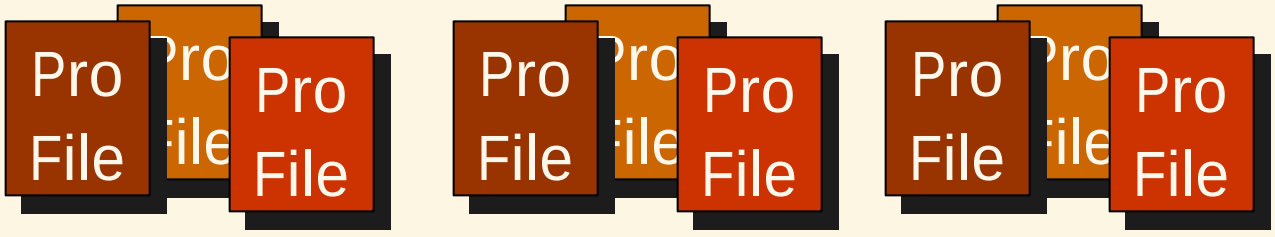
- Embedded Java
  - Concept: arbitrary subset of JVM and classlibs
  - Licensing: may not expose any public API
  - “ROMizing” tool
- Personal Java
  - Small spaces??? 2MB ROM/2MB RAM
  - To be replaced with future Personal Java Profile
- JavaCard
- Jini

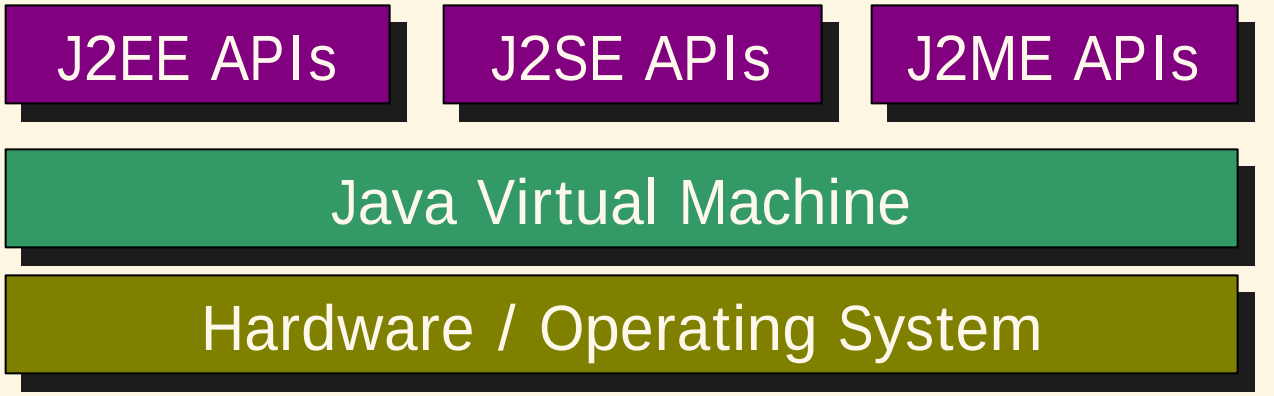
Java™ 2 Platform

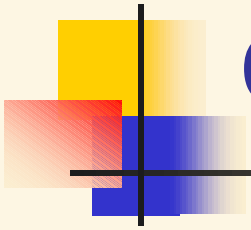


Profiles

Configu-
rations





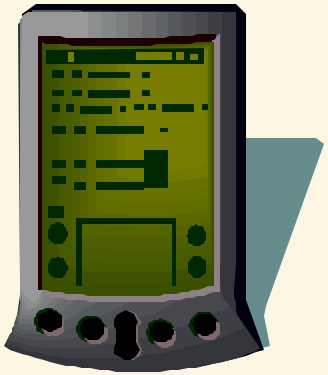


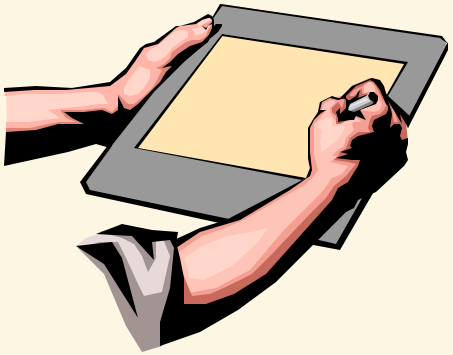
Configuration: CLDC

- Connected Limited Device Configuration
- JSR-000030
- **Quick Overview:**
  - Memory: 128K-256K total (<= 256K ROM / <= 256K RAM)
  - Limited power (battery)
  - Some type of connectivity (maybe < 9600Bps)
  - No or limited UI

CLDC Profile: MID

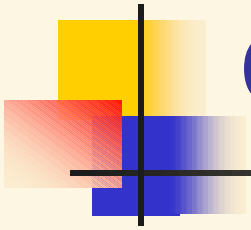
- MID (Mobile Information Device) Profile
- Based on CLDC
- JSR-000037
- See: http://java.sun.com (search for jsr-000037)



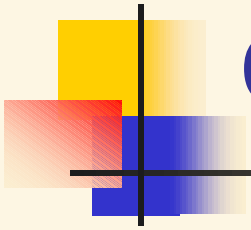
CLDC Profile: MID (continued)

- **Potential APIs**
  - Display toolkit for limited size displays
  - UI input (pen, buttons, keyboard, etc)
  - Persistent data storage
  - Messaging (SMS, e-mail, etc)
  - Networking (datagram and connection oriented)
  - Security
  - Integrity of device and network from rogue apps
  - End-to-end security and data integrity
  - Wireless telephony



Configurations: CLDC/MID (cnt'd)

- **Sourced from**
  - J2ME CLDC
  - Java Phone
  - Java Telephony API
  - Mobile Network Computer Reference
  - . . .



Configuration/Profile Candidates

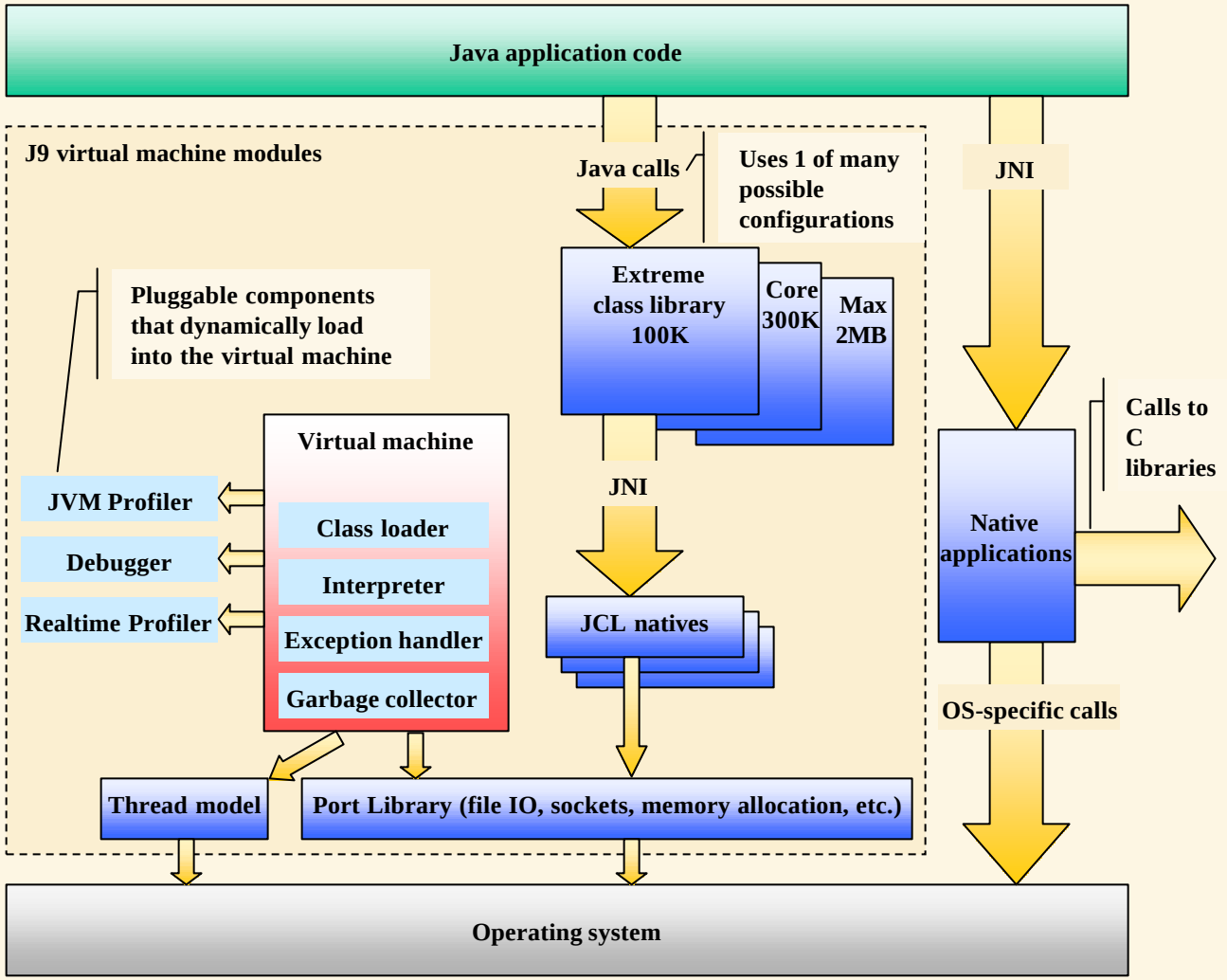
- Sun “products” (see http://java.sun.com/products):
  - Personal Java, Java TV API, KVM, Java Phone, Java Media Framework, JTAPI
- Java Specification Requests (JSR):
  - Personal Profile (#62), Building Automation (#60)
- Mobile Network Computer Reference (http://www.oadg.or.jp/activity/mncrs)
- Mobile Station Application Execution Environment (http://www.etsi.org/smg/smg4/smg4.htm)
- ECTF (http://www.scsa.org/refspecs.htm)

IBM/OTI Offering

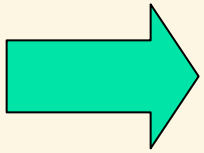
- VisualAge Micro Edition
- Modular JVM (J9) with separate modules for:
  - Real-time profiling, debugging, gc, jni, ...
- Classlibs: Extreme (100K), Core (300K), Max (2MB)
- Much of VM code shared among platforms
- Porting done in specially architected port layer
- More information:

- http://www.embedded.oti.com
- http://www.embeddedsystems.com/issues/february2000/49/feb2000-49.htm

IBM/OTI Offering (continued)



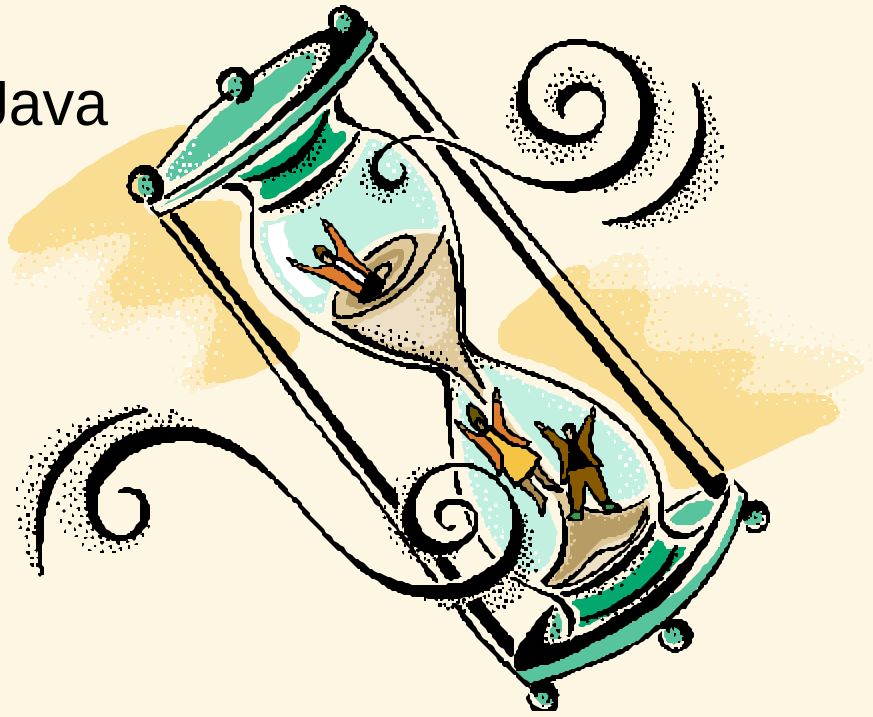
Various Issues in more detail

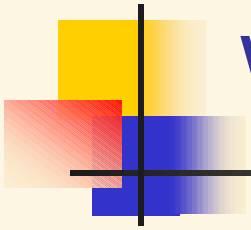


- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues

Real-Time Java

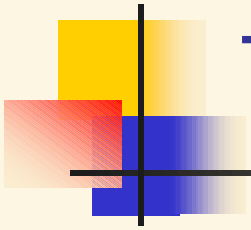
- What is real-time?
- The Real-Time Java Specification for Java
- Goals
- Modifications needed to Java
- Timeline
- Suggested readings





What is real-time?

- Real-time requires consistent timing behavior more than absolute speed
- Excellent overall performance always required



The Real-Time Specification for Java

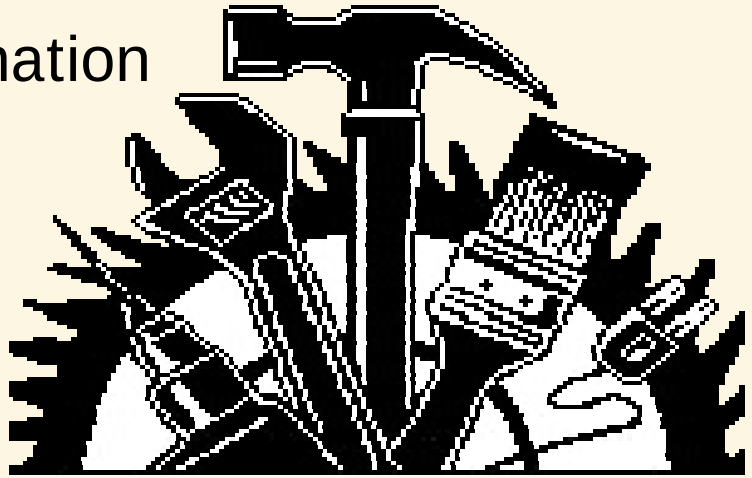
- The RTSJ provides modifications to the language and the VM specifications
  - Working group started mid 1998
  - The Real-Time Specification for Java Expert Group (RTJEG), convened under the Java Community Process and JSR-000001 in March 1999
  - The Public Review of the RTSJ closed 14 Feb 2000
- Following up on the work of the Requirements Group for Real-time Extensions For the Java™ Platform, National Institute of Standards and Technology

RTSJ: Guiding principles

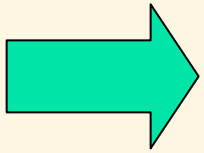
- Applicability to particular Java environments
- Backward compatibility
- WORA
- Predictable execution
- Current practice now, support advanced features later
- No syntactic extensions
- Allow variation in implementation decisions

RTSJ: Seven modified areas

- Thread scheduling and dispatching
- Memory management
- Synchronization and resource sharing
- Asynchronous event handling
- Asynchronous transfer of control
- Asynchronous thread termination
- Physical memory access



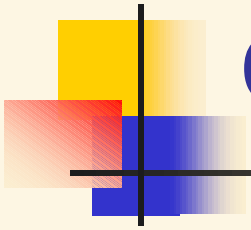
Various Issues in more detail



- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues

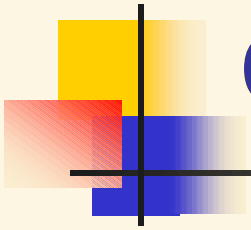
Garbage Collection

- What is it?
- What do we need for embedded systems?



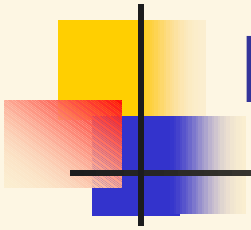
GC: What is it?

- Heap storage for objects is reclaimed by an automatic storage management system known as the *garbage collector*
- Fewer programmer errors, and can be fast and predictable
- Different algorithms with different properties
- Algorithm not specified by Java virtual machine spec
- Critical component in (real-time) embedded



Current Java memory management

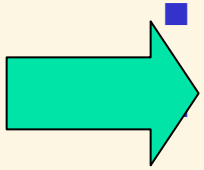
- GC is allowed to:
  - Be disruptive
  - Preempt all threads
  - No latency guarantees
- Limited control:
  - System.gc()
  - System.runFinalization()



Embedded system requirements

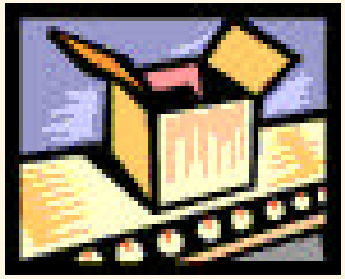
- Characteristics of run-time environment
  - Interpreted, JIT or compiled
- Hardware characteristics and limitations
  - Footprint, memory types
- Application requirements
  - latency
  - forward progress consistent with the rate at which the application allocates memory (paced)
- Behavior in critical memory situations
  - Fragmentation not acceptable
  - Graceful degradation

Various Issues in more detail



- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues

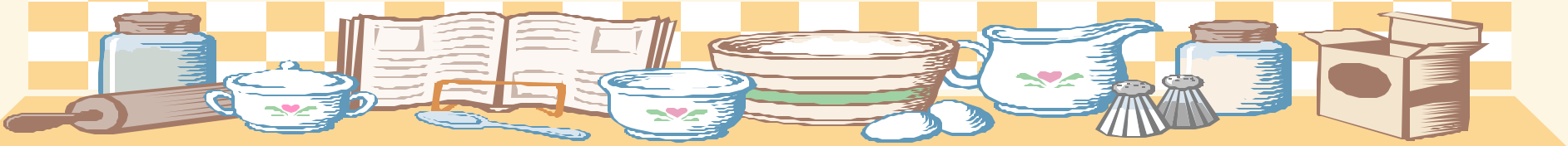
Java Application packaging



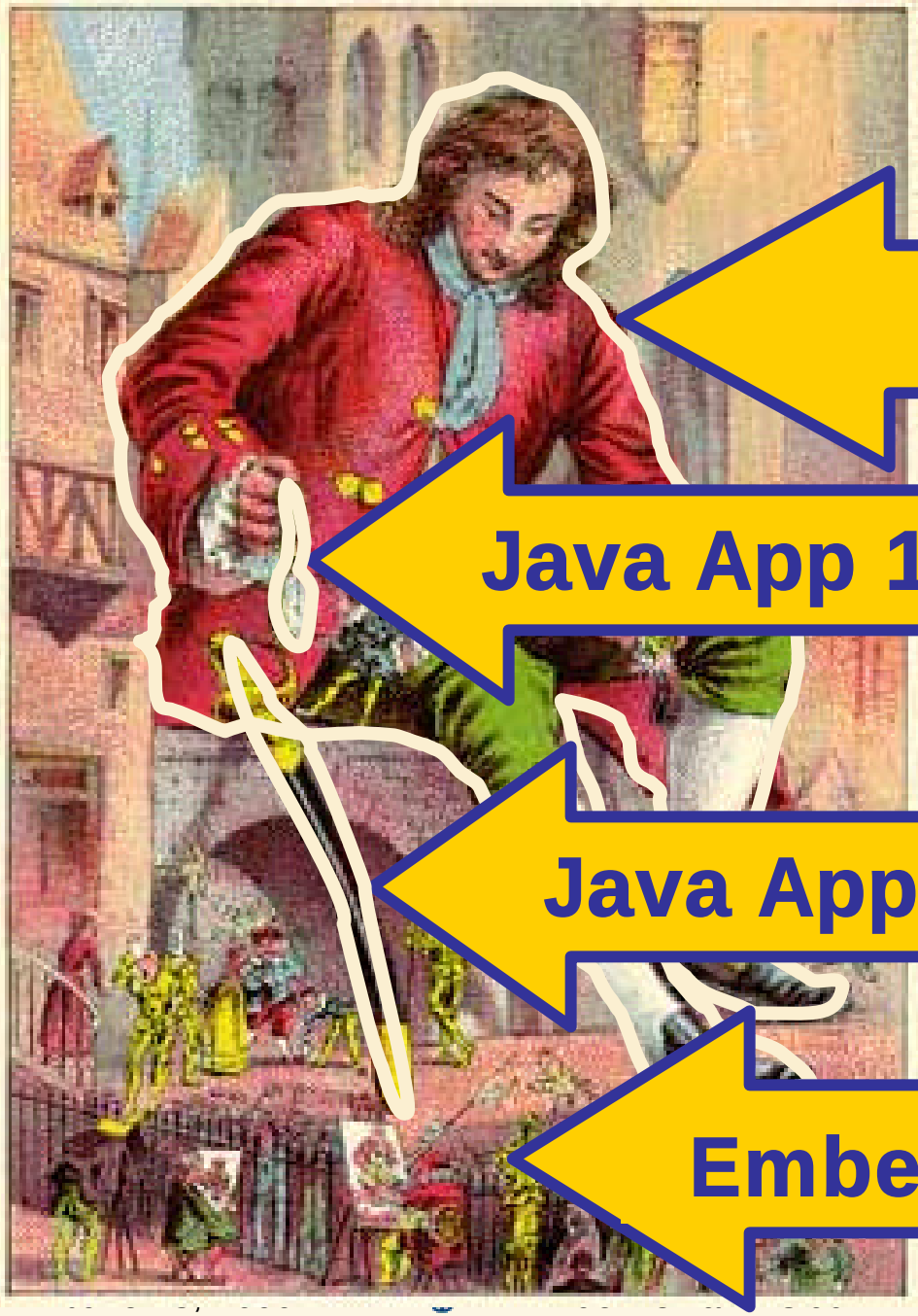
Goal is to make apps smaller before shipping them

The basic recipe

- Take a Java program
- Analyze it
- Select code that is actually used
- Optimize and compress this code
- Determine dependencies on others
- Make code run on embedded device



The Problem...



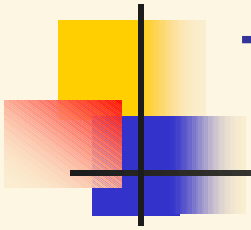
Java VM

Java App 1

Gulliver in Lilliputland

Java App 2

Embedded Land

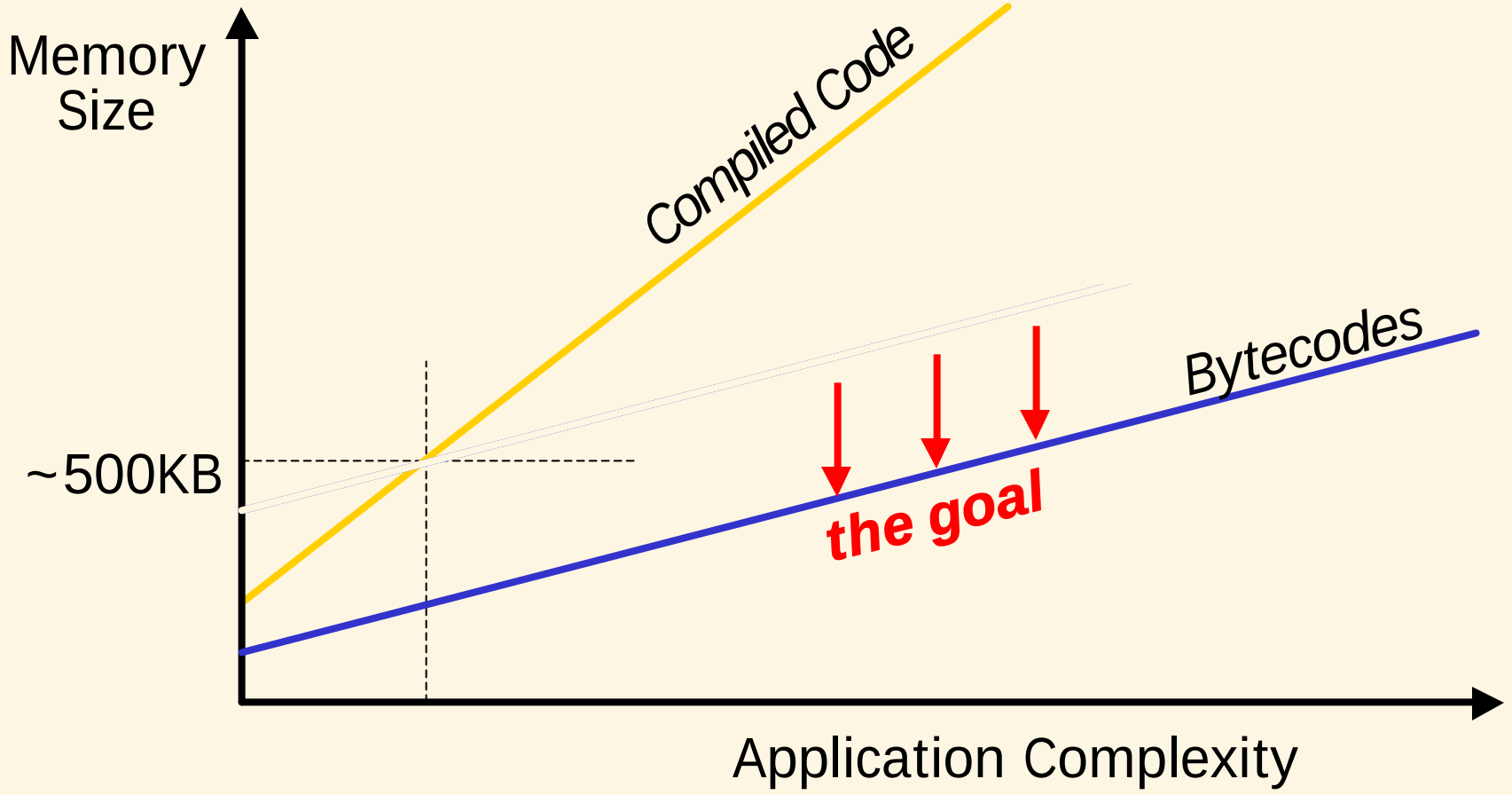


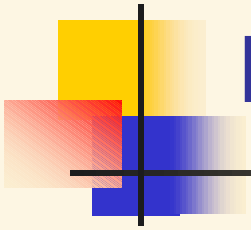
The VAME Smart Linker Credo

640K ought to be enough for anybody

Bill Gates, 1981

Break-even point for Bytecodes

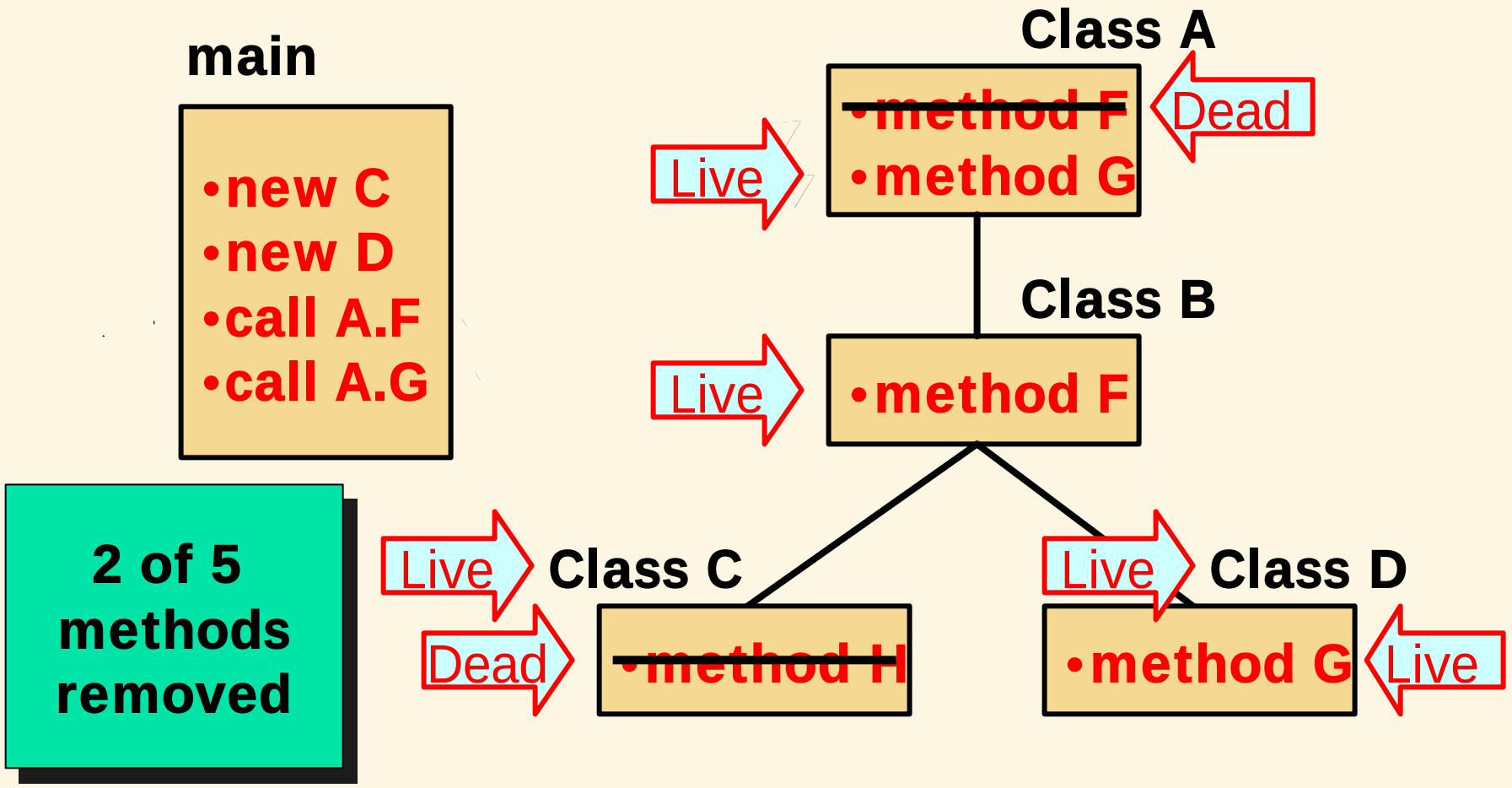




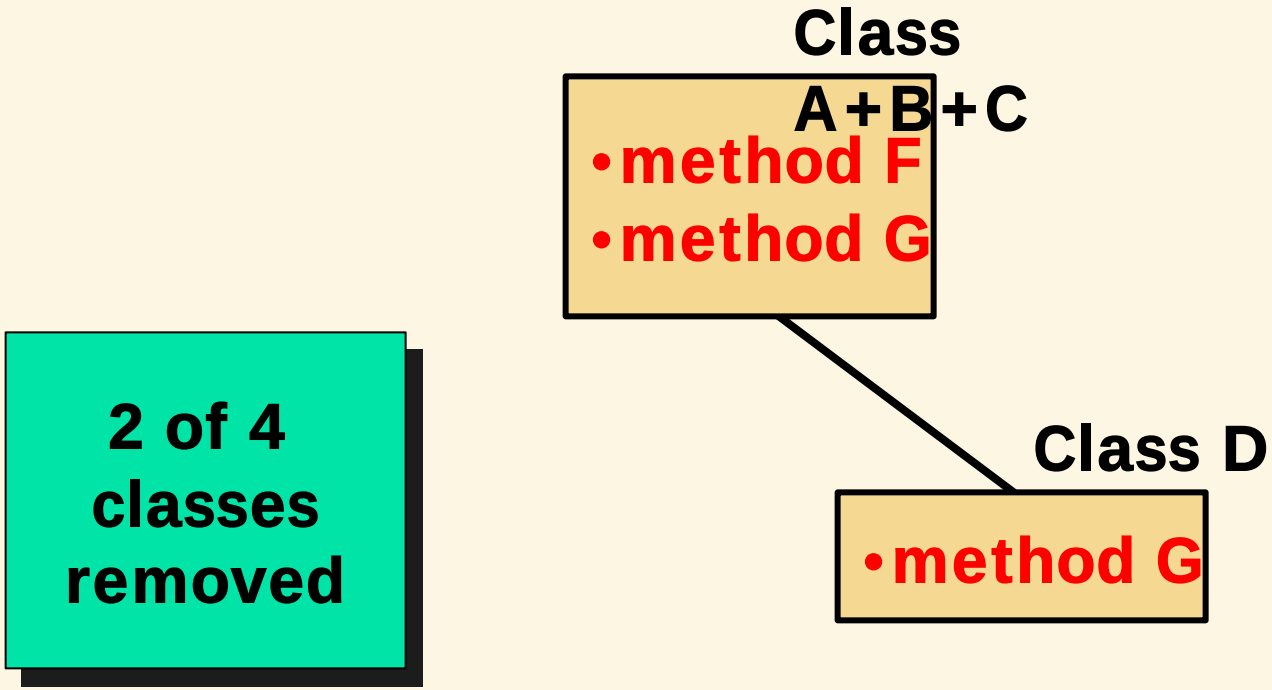
Run Big Things on Tiny Devices?

1. Moore's Law (i.e., just wait a few years...)
2. Reduce the size of JVM (i.e., smart coding)
3. Invent more compact class file format
4. Remove unused artifacts
5. Perform Class Hierarchy Transformations
6. Rename long names and 'intern' constants
7. All of the above...

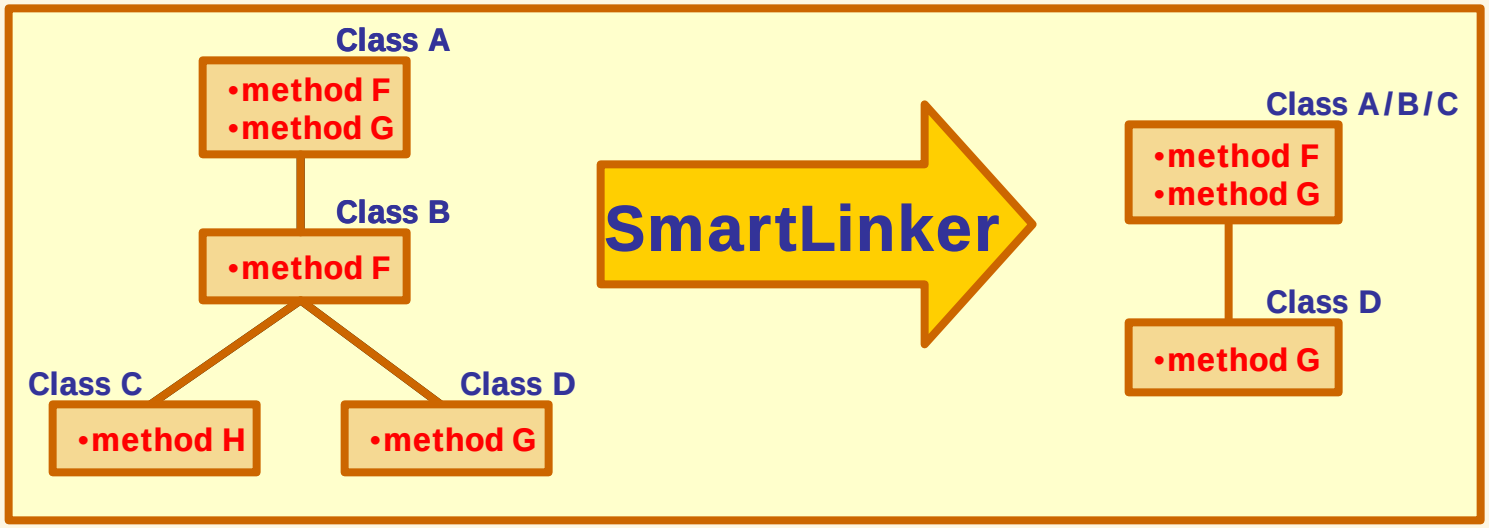
Removal of Unused Artifacts



Class Hierarchy Transformations

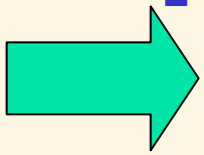


Sample Savings Summary

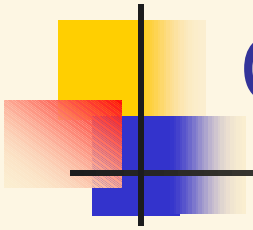


original ZIP	3660 bytes	100%
dead methods	1733 bytes	47%
transformations	1184 bytes	32%
name compression	1134 bytes	30%

Various Issues in more detail

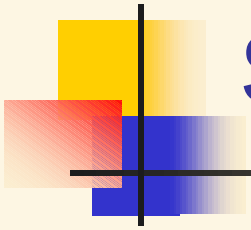


- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues



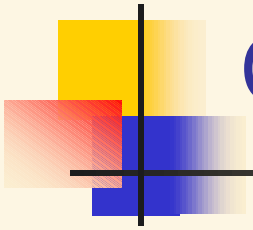
OSGi (Open Services Gateway Initiative)

- Founded in March 1999
  - Open specs for delivery of services to home gateways, over wide-area networks
- IBM founding member of OSGi
  - Part of board of directors
  - Provides technical lead
- Applications are distributed in the form of “bundles”
- The code management layer for embedded devices, not just home, but also car and eMobile devices.
- More information: http://www.osgi.org



SMF (Service Management Framework)

- IBM’s implementation of OSGi standards
- Implements OSGi Release 1.0 level:
  - Framework
  - HttpService
  - LogService
  - Device Access
- Publicly released in May 2000

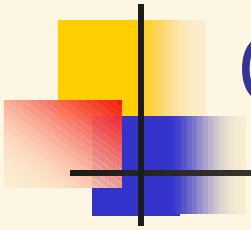


OTI's Component Architecture

- Light-weight component model
- Supports version management
- Component life-cycle support:
  - Loading components
  - Unloading components
  - Dependency model
- Target attributes:
  - resource-constrained embedded products
  - Client-server applications
  - Low speed (wireless) connections

Bundles vs. OTI Components

Bundles	OTI Components
No resource constraints	Has resource constraints
Run on standard JVM	Needs special packaging
Loaded from Internet	Restricted, from server
No compatibility model	Has compatibility model
Cached on file system	Stored in flash memory
Configuration remotely	Localized configuration
Set of core services	No core services



Components and Resource Math

- Embedded systems have limited resources:
  - Amount of memory used
  - CPU time used
  - Number of files
  - Number of threads
  - Number of sockets
  - Restricted access to certain API
    - E.g., no access to Java reflection

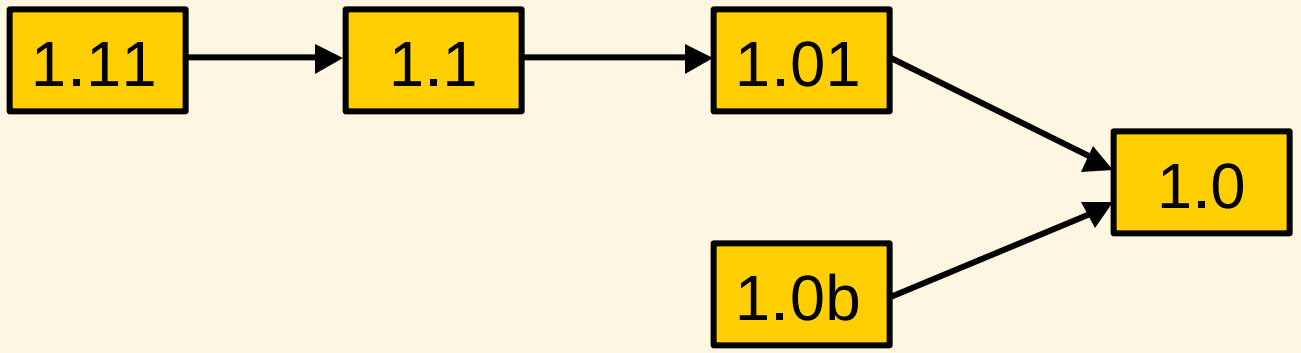
What's a Component?

- Stored in the form of a **jxe** (special sort of JAR)
- Component Description:

```
Name=Hangman Game
UniqueID=45727250726F6E66,39393034303961
CompatibilityID=45727250726F6E66,39393034303961
ICControllerClassName=com.acme.HangmanTextUI
RAMSize=5620
ROMSize=1240
PrerequisiteID=4261644943736F,39393034303961
```

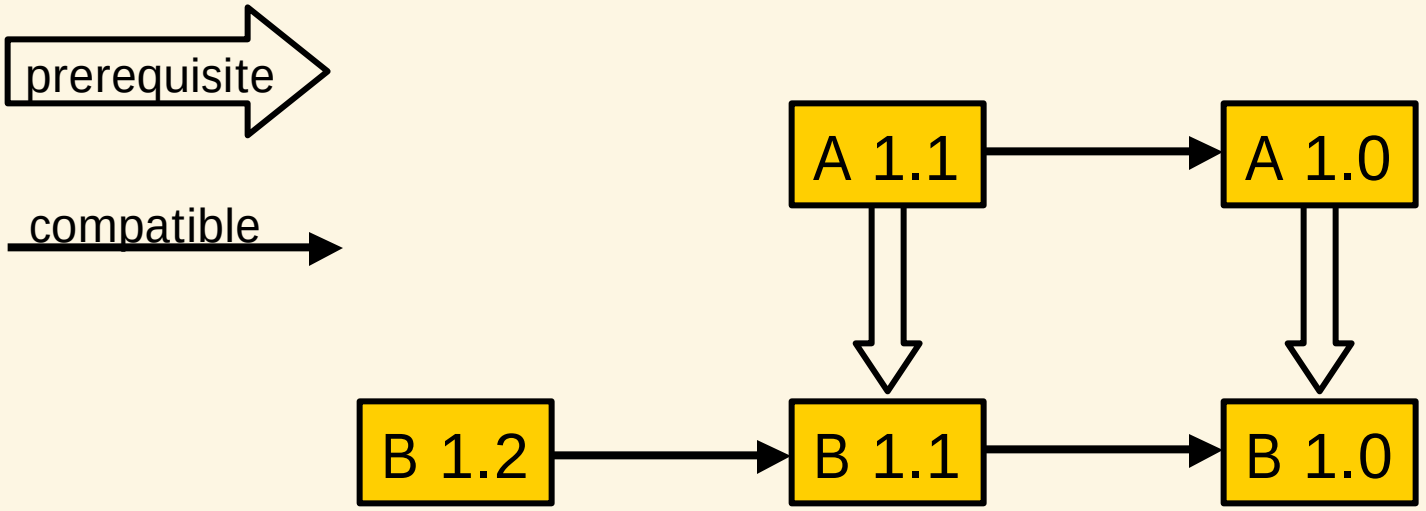
Component Compatibility Model

- Version 1.1 is compatible with 1.0 if 1.1 implements at least all of the API that 1.0 implements
- Relationship is transitive, but not reflexive

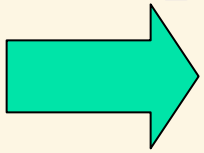


Component Prequisite Model

- Component can depend on existence of other components (or *compatible* version)



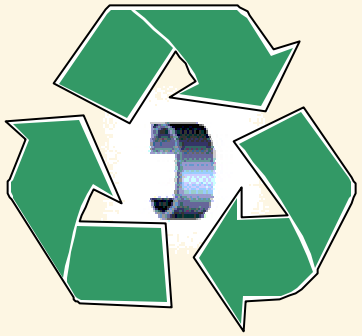
Various Issues in more detail

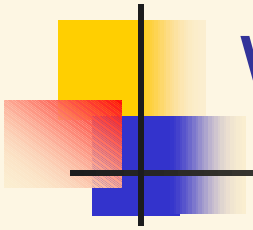


- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues

Java Native Interface

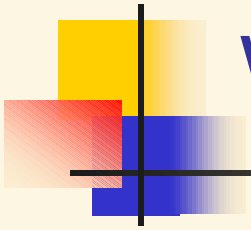
- What is JNI? Why JNI?
- The evolution of JNI?
- JNI basics
- Patterns for native interfacing
- Suggested reading





What is Java Native Interface?

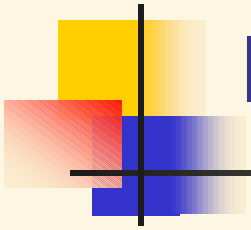
- Write Java *native methods* to integrate a Java application with code written in C or C++
- Use the *invocation interface* to integrate Java code into an existing application
- Defined in “The Java Native Interface Programmer’s Guide and Specification”



Why JNI?

- Directly access hardware and embedded operating system
- Integrate with other embedded software components
- Use native code for performance critical functions

- JNI breaks WORA
- Tiny VM configurations do not support JNI
  - Connected, Limited Device Configuration
  - KVM, J9 for PalmOS



Basics

- Write a Java class with a native method

```
public void native drawLine(int x1, int y1, int x2, int y2);
```

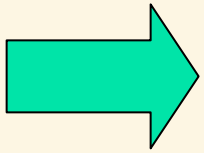
- Write the corresponding C code

```
JNIEXPORT void JNICALL Java_MyClass_drawLine(JNIEnv *env, jobject obj,
                                             jint x1, jint y1, jint x2, jint y2) {
    WinDrawLine(x1, y1, x2, y2);
}
```

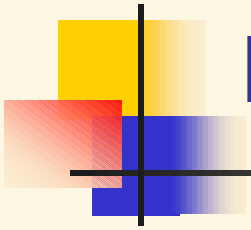
- Compile and link
- Run

All within a minute…

Various Issues in more detail



- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues

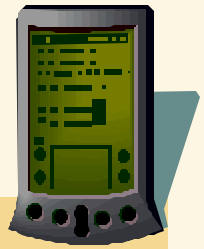


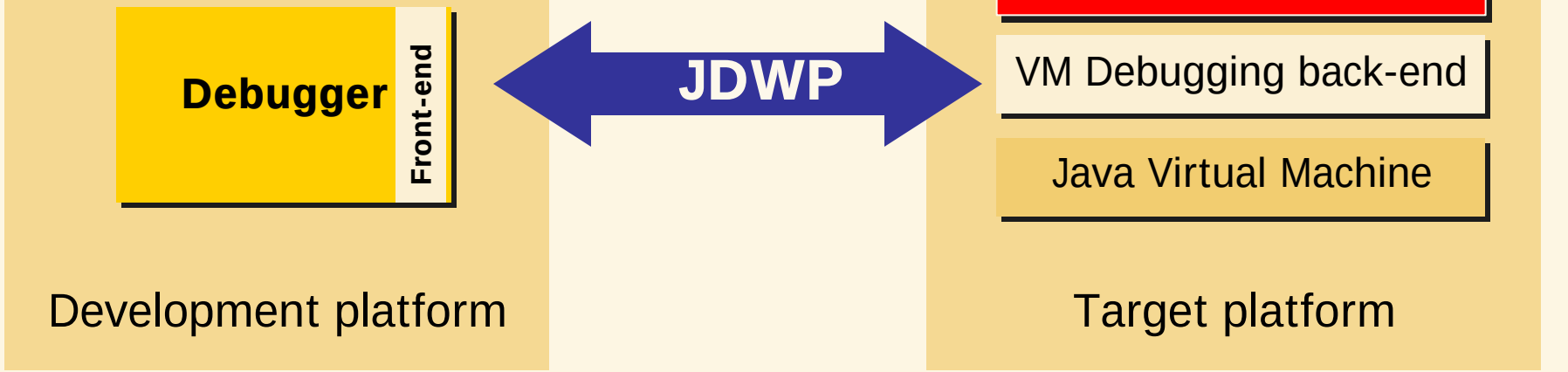
Debugging and profiling

- Traditionally, the debugger and the target Java VM execute on the same machine
- Acceptable model for desktop machines
- Much harder (impossible?) for embedded systems
  - Debugger usually too big to run on the target
  - Not always a UI
  - Lots of setup problems
- Solution?
  - Use an emulator (if it produces the same bug!)
  - Debug remotely (when problem is device specific)

Remote debugging architecture

- Java Platform Debugger Architecture (JPDA)
- Specified by Sun

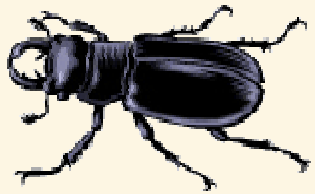




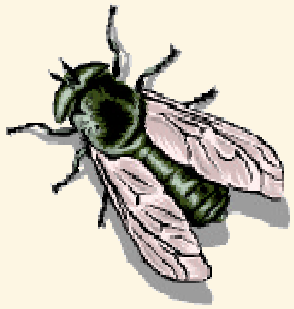
Remote debugging acronyms

- Java Platform Debugger Architecture (JPDA):

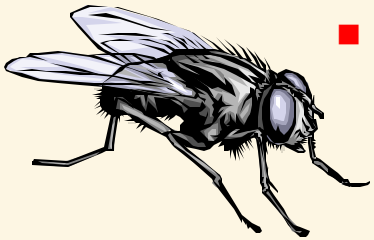
  - VM back-end
    - Java VM Debugger Interface (JVMDI)



  - Debugger front-end
    - Java Debug Interface (JDI)

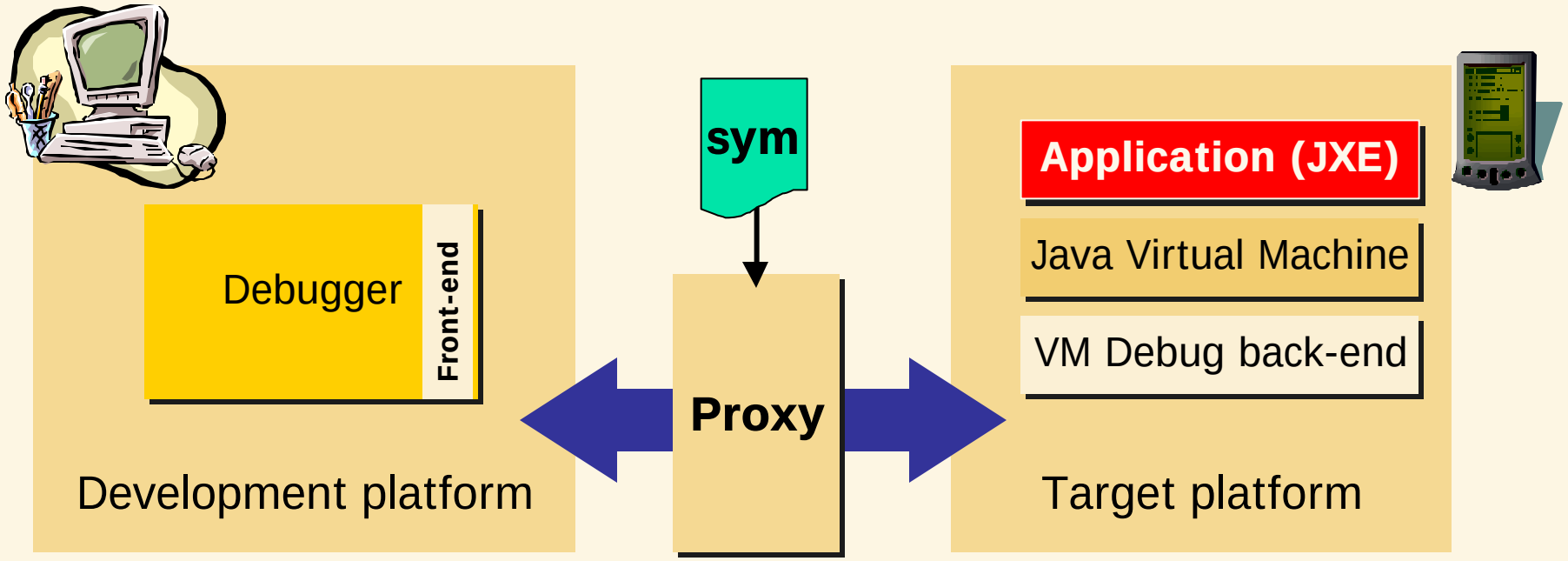


  - Link in-between
    - Java Debug Wire Protocol (JDWP)

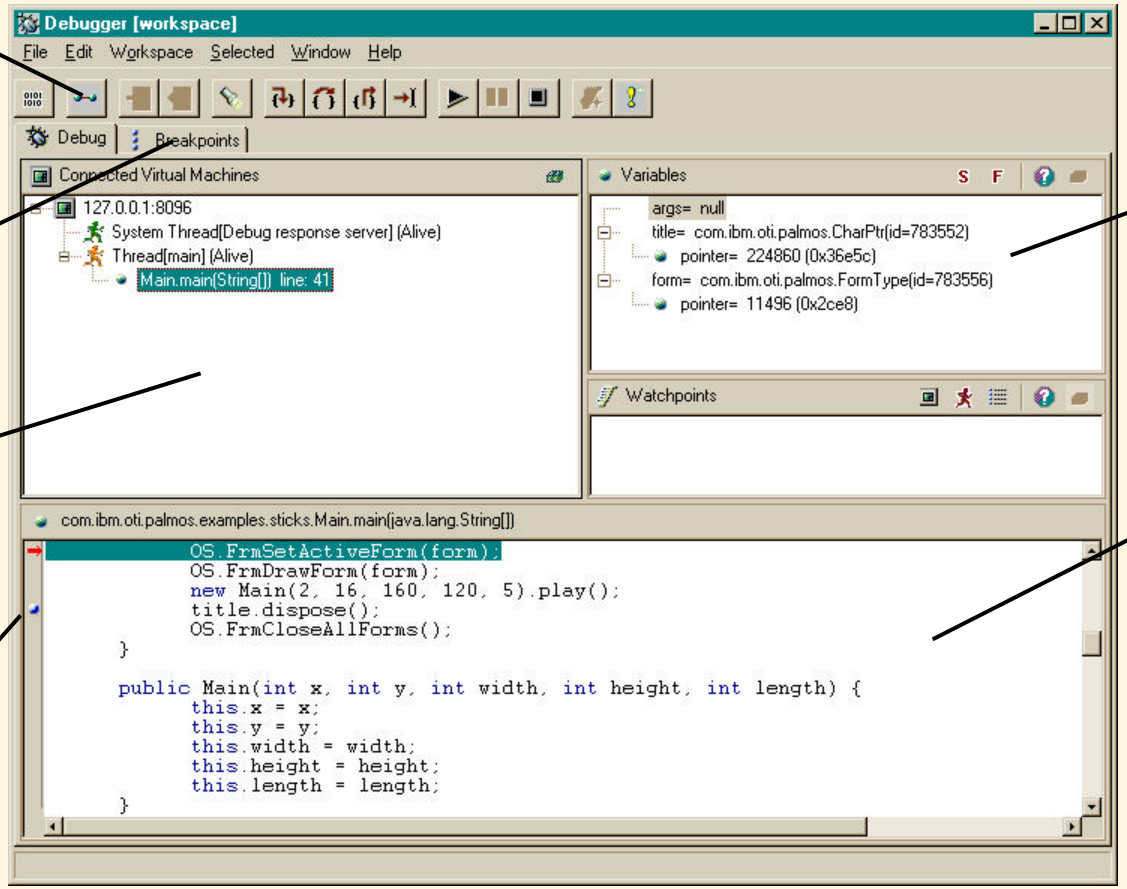


Remote symbolic debugging

- Symbolic debug info takes lots of space
- Cannot be stored at embedded device
- Saved in **sym** file; generated during packaging



VAME Remote debugger in action



Connect to (another) remote VM

Tab to inspect current set of breakpoints

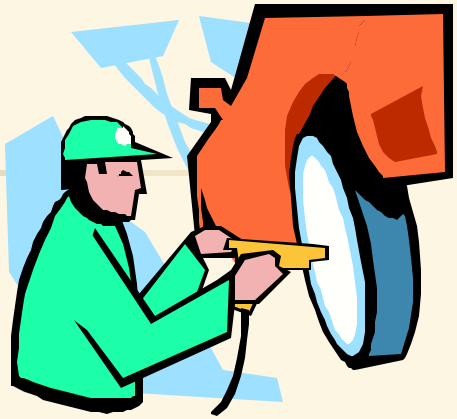
List of running threads

A breakpoint

Fields and variables visible from this method

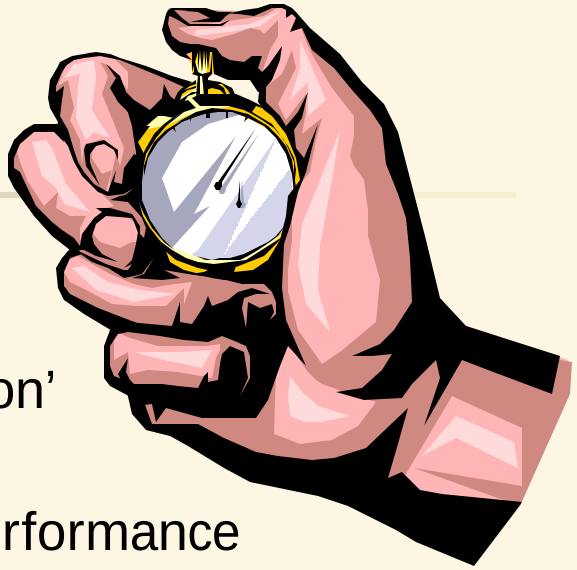
Source of the active method. The info is obtained from the .sym file generated during packaging.

Hot Code Replacement



- The idea:
  - VM runs (on a remote device)
  - Current application suspended, but not exited
  - Replace already loaded class, totally or partially
  - Continue execution with new class
- JDWP does not support this
  - The J9 VM **can** be told to reload a class
  - New class to be loaded with standard classloader
- OTI suggests additions to JDWP/JDI to allow H.C.R. over the debugging channel (TCP/IP)

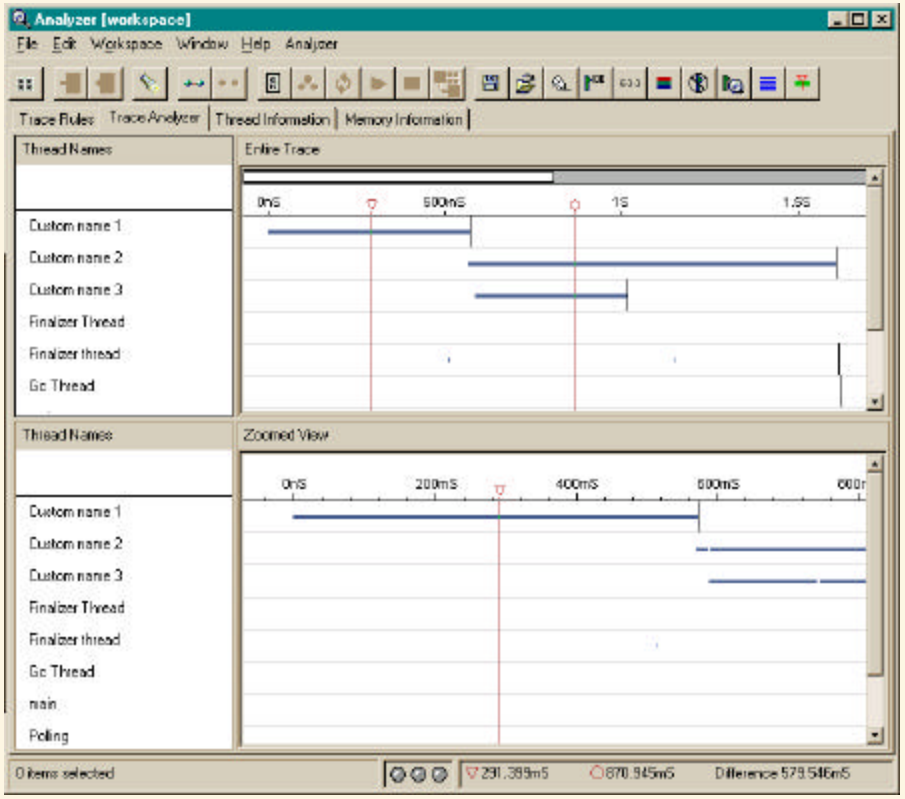
Real-Time Profiling



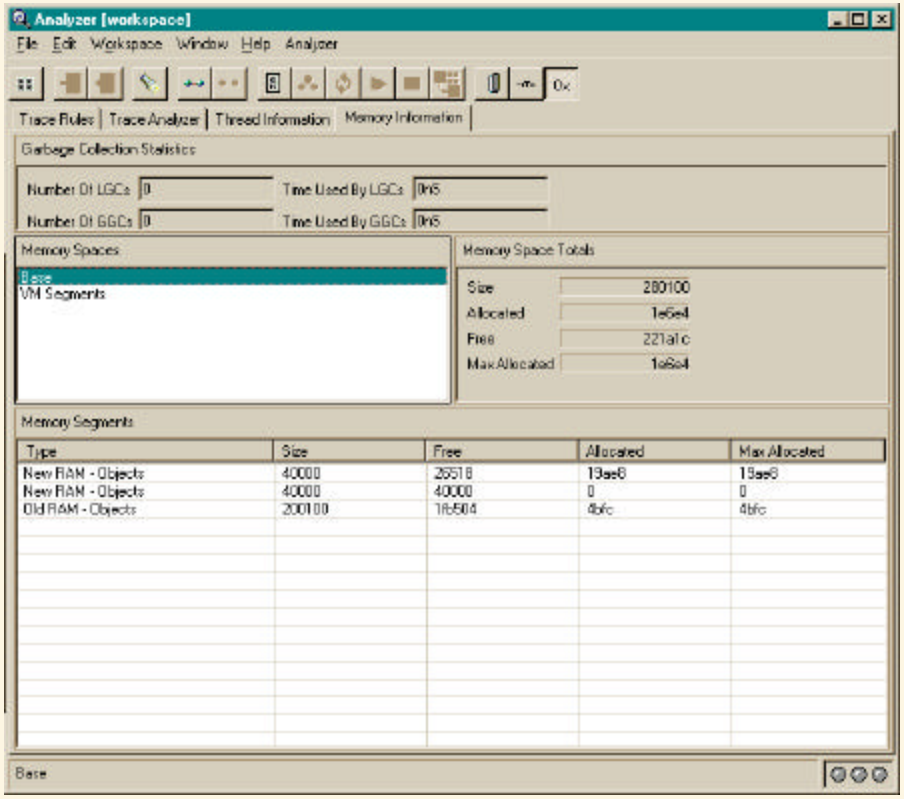
- Embedded devices are slow, profiling is often essential (e.g., to improve instant-on' response time or general behavior)
- If you know what resources are used, performance improvements can be made by changing the code
- Information collected remotely, sent over the wire to host
- The VAME remote analyzer shows detailed information of execution on remote device:
  - Thread state changes (and time spent where)
  - Memory usage (heap, #object handles)
  - JNI calls
  - Garbage collection

Real-Time Profiling

- After running 3 threads “simultaneously”

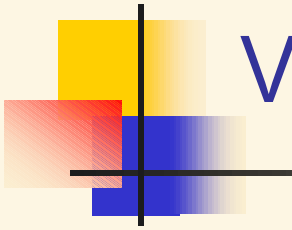


Trace analyzer

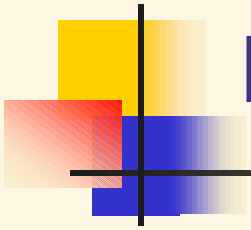


Memory analyzer

Various Issues in more detail



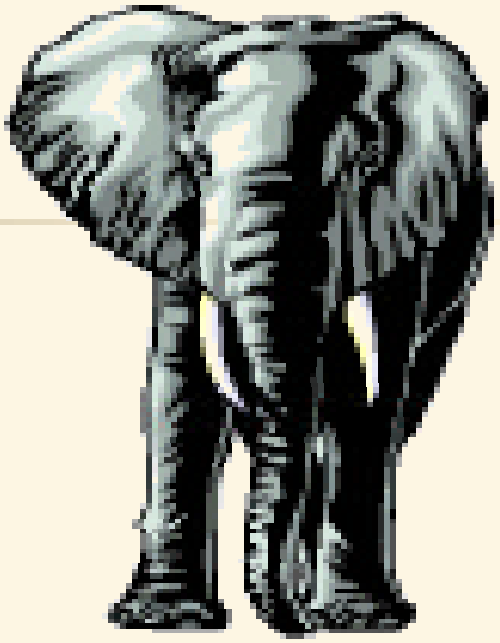
- Real-time issues with Java on embedded systems
- Garbage Collection
- Java Application packaging
- Components
- JNI
- Debugging and Profiling in embedded Java
- Presentation and UI issues



Presentation and UI issues

- UI Issues on small devices
  - Once again, limited memory space
  - Interaction with the embedded system
    - No mouse, no keyboard?
    - Low performance system
    - Usually a small screen
- UI definitions depends on the profile (J2ME)

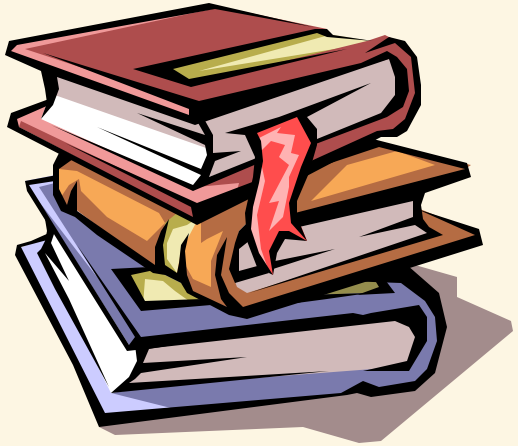
Small, Medium, Large?

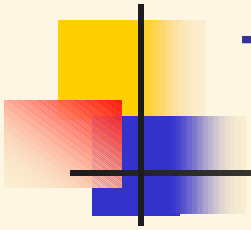


- AWT?
  - Standard AWT: 500K+200K
- Swing?
  - Even bigger, more complicated
  - Not enough interaction with native capabilities
- All this is too much! Remember: JCL = 100K !
- What do we want?
  - Small UI library
  - Better portability

Many players in the UI area

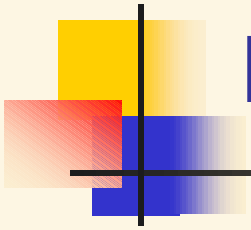
- Espial (www.espialgroup.com)
- Sun, with Truffle (http://java.sun.com/products/personaljava/truffle/)
- IBM, with MicroView (http://www.embedded.oti.com/about/microvue.htm)
- Bonita (www.bonitasoftware.com)
- ...





Two different approaches

- Using the native look-and-feel of the device
  - Better performance and integration
  - Example: MicroView
- Bringing the look-and-feel with the toolkit
  - Whatever platform you use, the look-and-feel can be the same
  - Example: Truffle

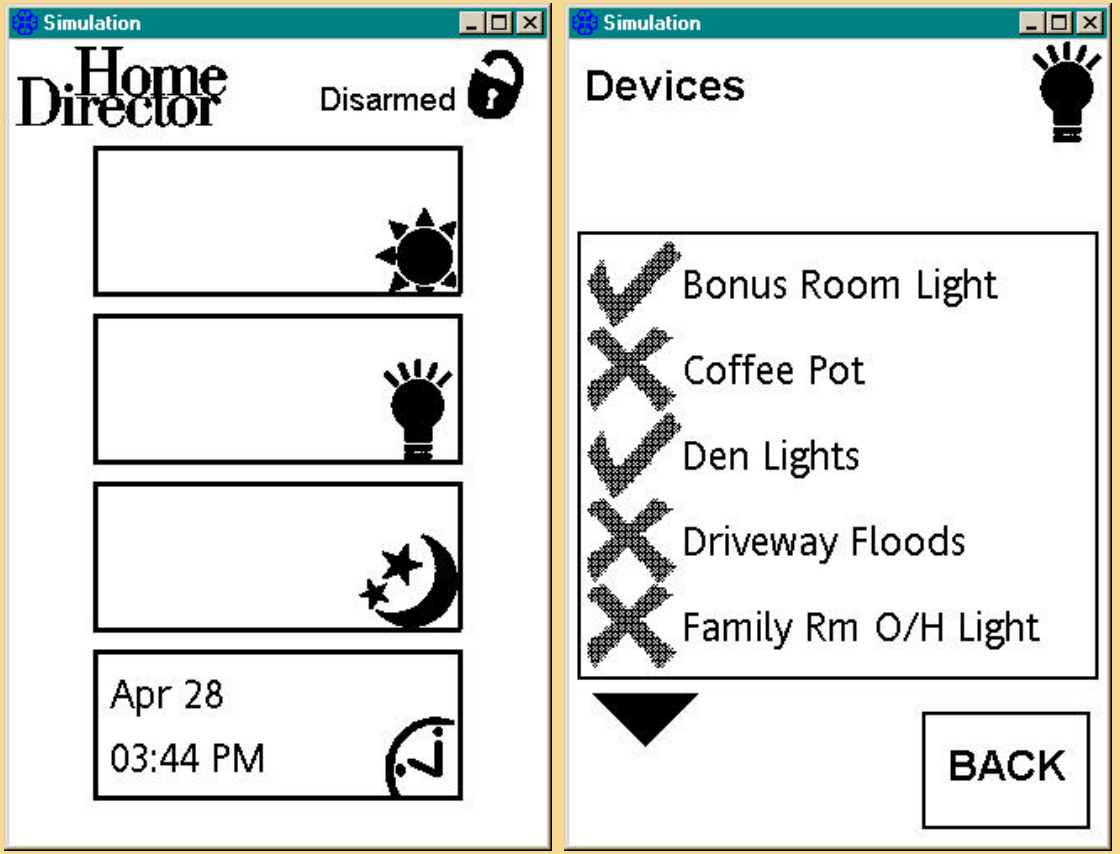


Microview

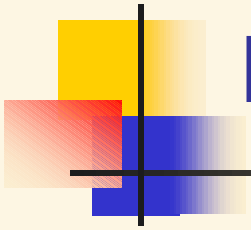
- IBM's UI framework for embedded
  - With small memory footprint
    - Less than 200K
  - Designed to make portability easier
  - Based on the MVC paradigm
    - Model (notified at every update)
    - View (bitmaps, labels, lists...)
    - Controller (process input events)

Microview Examples

Dialer



HomeDirector demo

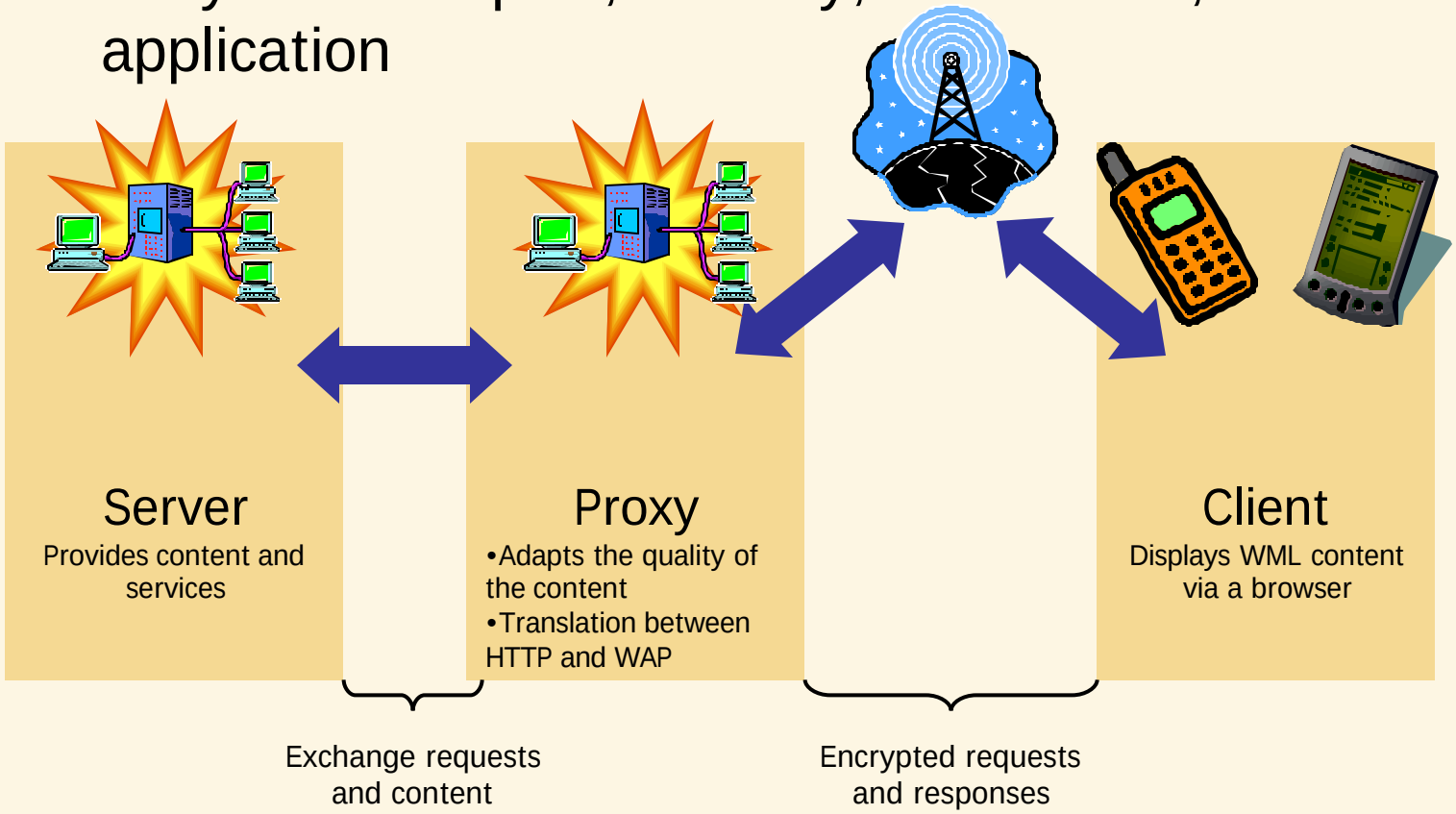


Presentation

- Sometimes, *content* is most important
- Wireless Application Protocol (WAP)
  - Is a set of protocols enabling content broadcast to a wide range of mobile devices
    - Mobile phones, PDAs, Etc.
- WAP content displayed via a browser
  - Content is in WML (kind of lightweight HTML)
  - WMLScript provided client-side interactivity
  - A Java WAP-Browser can be as small as 64-128K
- More info about WAP:
  - The WAP Forum (www.wapforum.org)

WAP

- 5 layers: transport, security, transaction, session and application



Questions?