Java Virtual Machine (JVM) in the Settop Box

Taeho Oh

http://ohhara.sarang.net

ohhara@postech.edu

Contents (1)

- Introduction
- Portable Embedded Application Development
- Debugging Embedded Application
- Porting JVM
- Optimizing JVM
- Java Native Interface (JNI)

Contents (2)

- JPDA, JVMPI
- Resource Management
- References

Introduction

Introduction

- About the Author
- Preliminary Knowledge
- About Alticast
- JVM Overview
- Sun JVM Editions

About the Author (1)

- 1997 ~ 2000
  - Studied Computer Science and Engineering at Postech
- 2000 ~ 2003
  - Worked for Alticast
  - Developed JVM for Digital Broadcasting Settop Box
    - The JVM is used for SkyLife settop box.

About the Author (2)

- 2003 ~
  - Studying Electronic and Electrical Engineering at Postech

Preliminary Knowledge

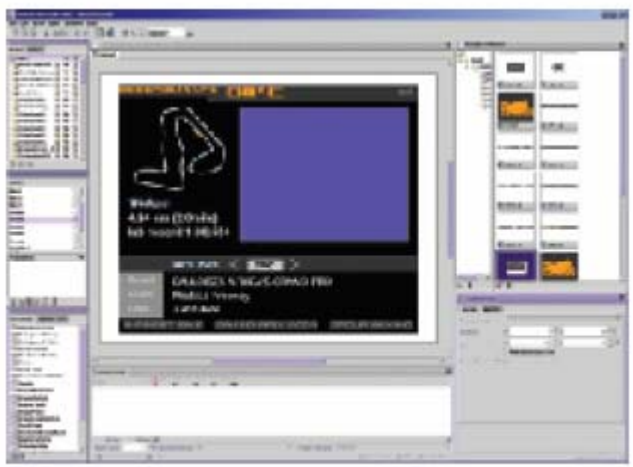
- C Language
- Java Language
- Java Native Interface (JNI)

About Alticast (1)

- Digital Broadcasting Total Solution

About Alticast (2)

- AltiComposer



AltiComposer™ in Action

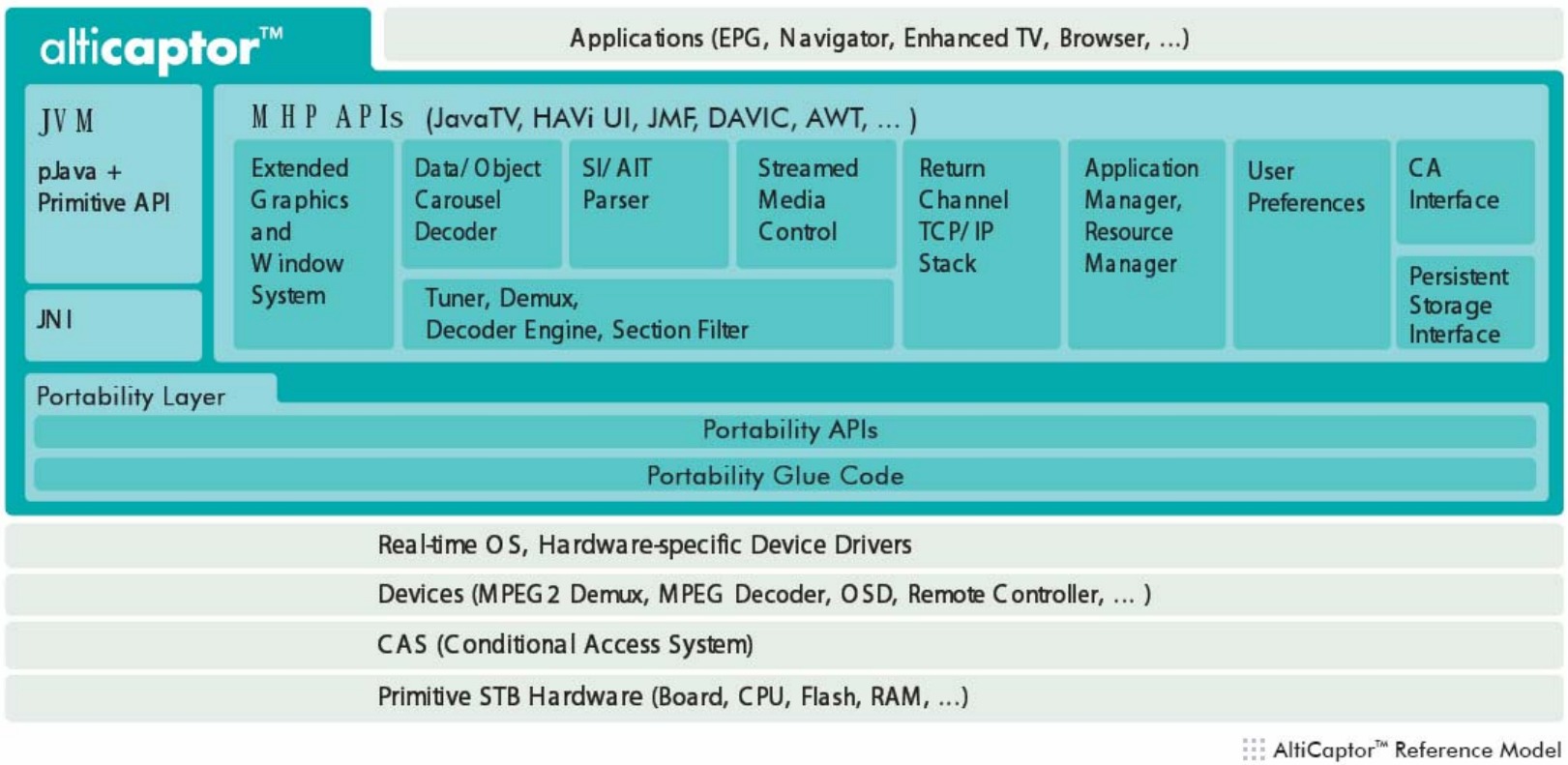
AltiEmulator™

About Alticast (3)

- DTV Applications

About Alticast (4)

- AltiCaptor

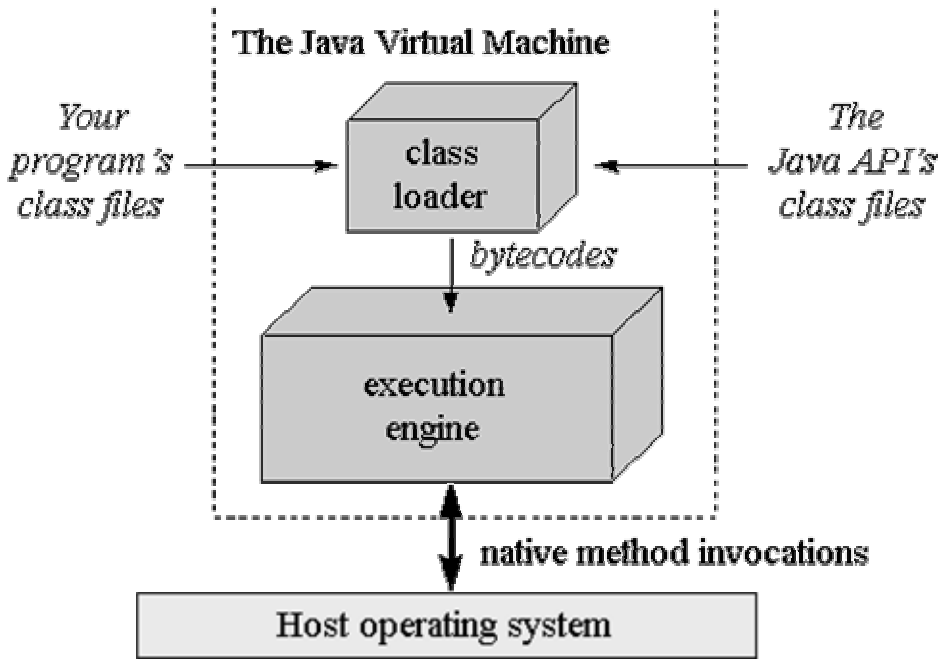


JVM Overview (1)

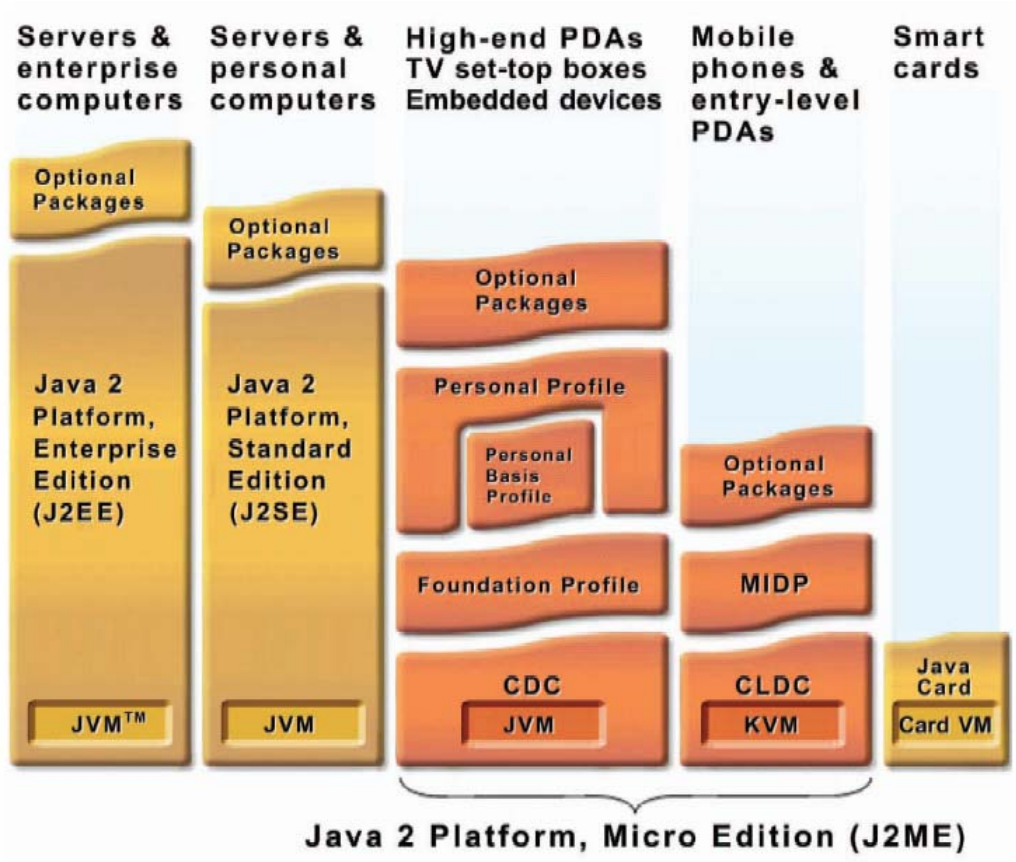
- Classloader
  - Read class file stream.
  - Verify the class file.
  - Resolve the class link.
  - Load into memory.
- Interpreter
  - Execute Java bytecode.

JVM Overview (2)

- Garbage Collector
  - Delete unused Java objects automatically.



Sun JVM Editions



Portable Embedded Application Development

Portable Embedded Application Development (1)

- External Library
- Wrapper API
- Byte Order
- Memory Alignment
- Evaluation Order
- char Type
- >> Operator

Portable Embedded Application Development (2)

- va_list
- Data Type Size
- 64bit Integer
- Floating Point Number
- Source Text Format
- C Compiler Options
- ANSI C (89)

Portable Embedded Application Development (3)

- OS Abstraction
- Portable Java Application

External Library

- Do not use external library
  - Even if it is ANSI C standard library.
    - ex. fopen

```
#include <stdio.h>
FILE *file_open(char *file) {
  return fopen(file, “r”);
}
```

Does not work properly if the platform doesn’t have fopen function (ex. filesystem is not available)

Wrapper API

- Make Wrapper API
  - If a specific API is really needed.

platform independent
(no modification is needed while porting)

platform dependent
(modification is needed platform by platform)

```
extern int em_printf(char *str);
int em_init(void) {
  return em_printf("Hello World₩n");
}
```

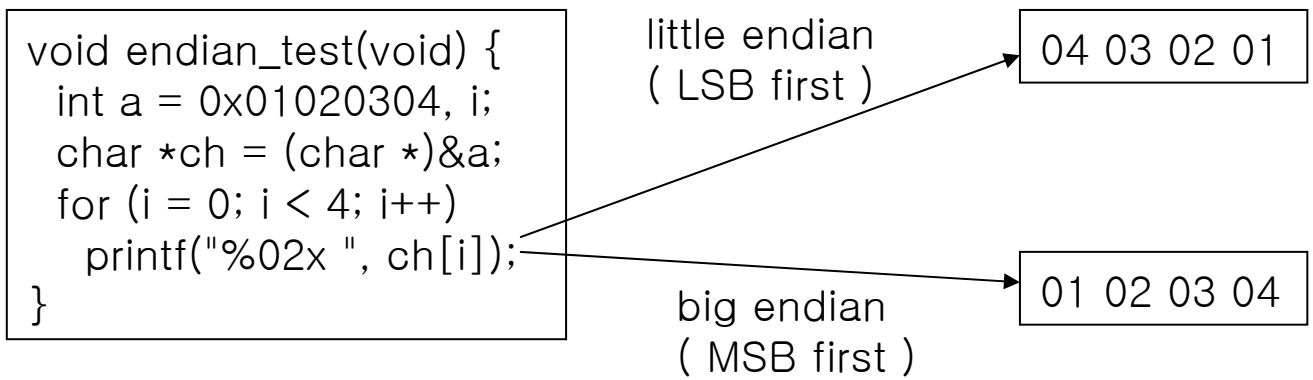
Wrapper API

```
extern int Print(char *str);
extern int em_printf(char *str);
extern int em_init(void);
int root() {
  return em_init();
}
int em_printf(char *str) {
  return Print(str);
}
```

Implementation of the wrapper API

Byte Order (1)

- Do not depend on byte order
  - If it is impossible, make a wrapper API.

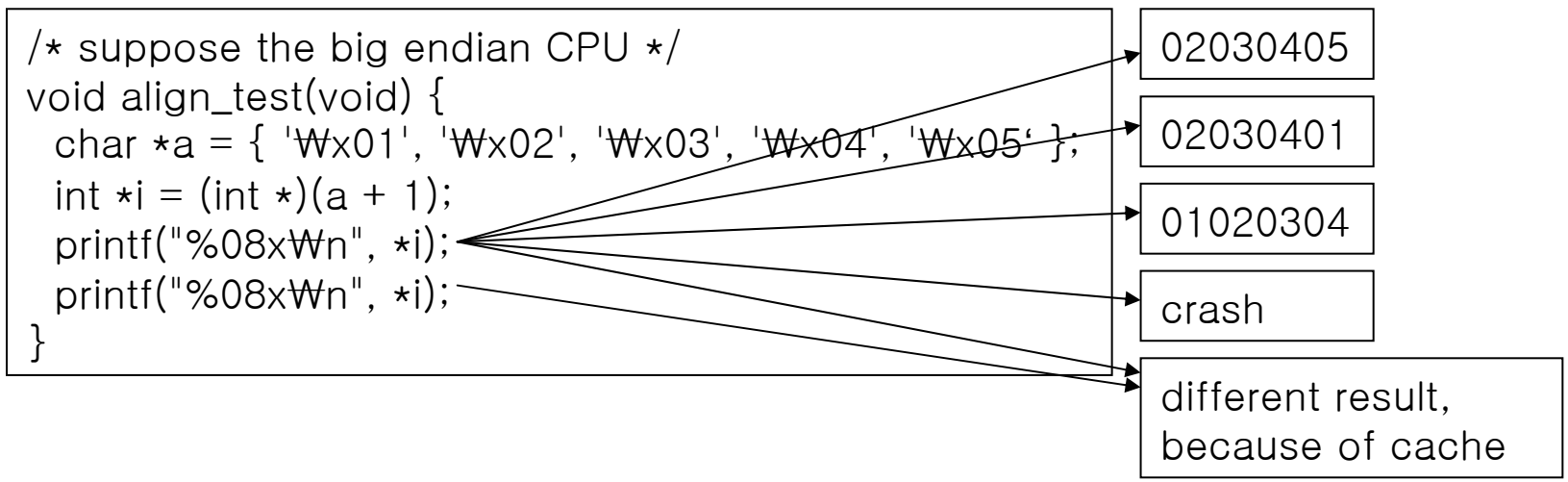


Byte Order (2)

- Floating point number and integer have a different byte order in some platforms
- 0x0102030405060708 is stored as 04 03 02 01 08 07 06 05 in some platforms

Memory Alignment (1)

- Do not access misaligned memory
  - The misaligned memory access may cause an unexpected result.

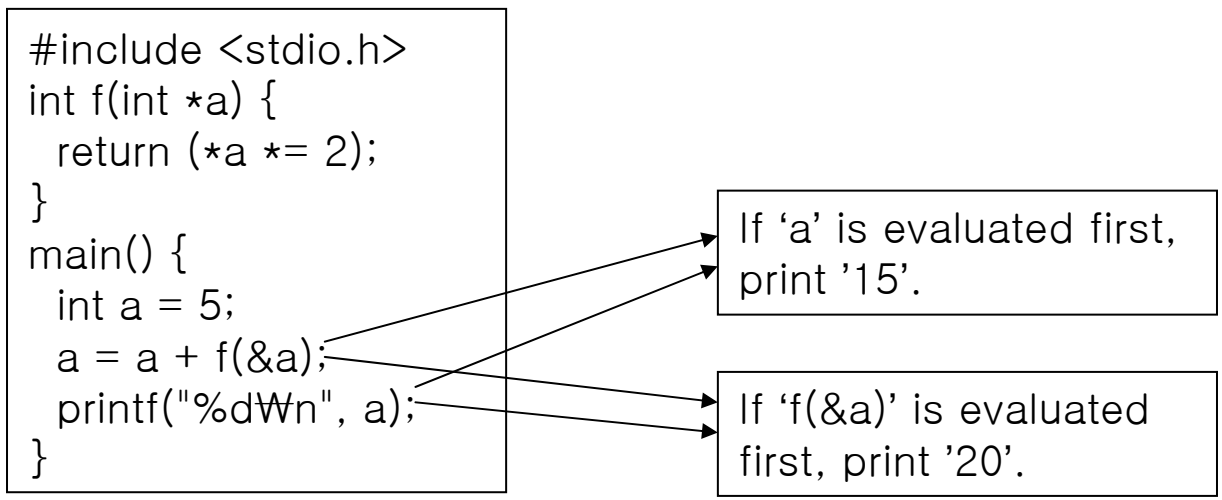


Memory Alignment (2)

- Floating point number and integer have a different memory alignment policy in some platforms
  - ex) 8byte alignment for 64bit integer, 4byte alignment for 64bit floating point number.

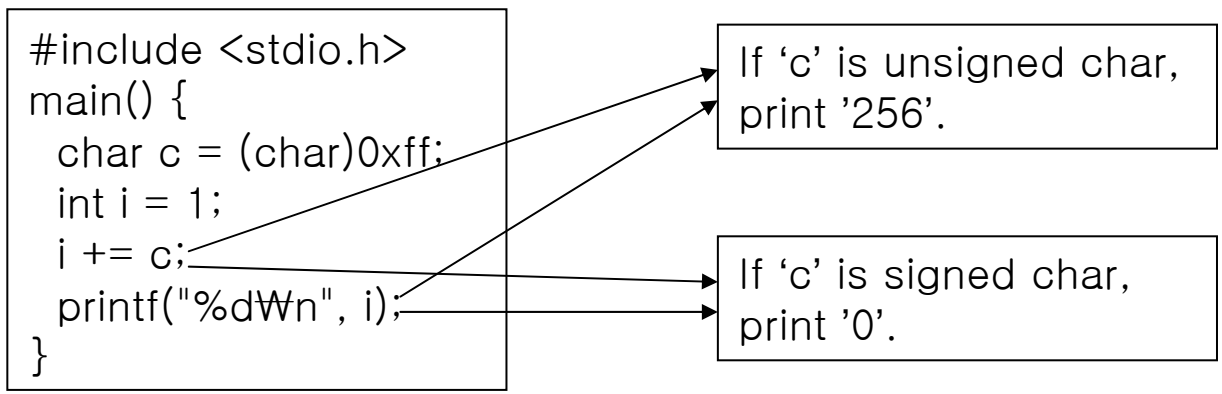
Evaluation Order

- Do not depend on evaluation order



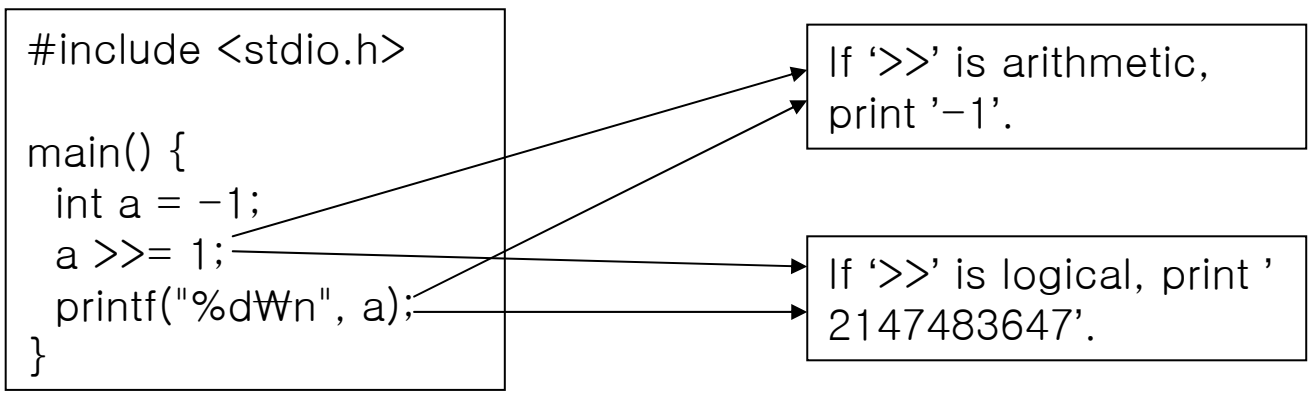
char Type

- Do not assume char type is signed (or unsigned)



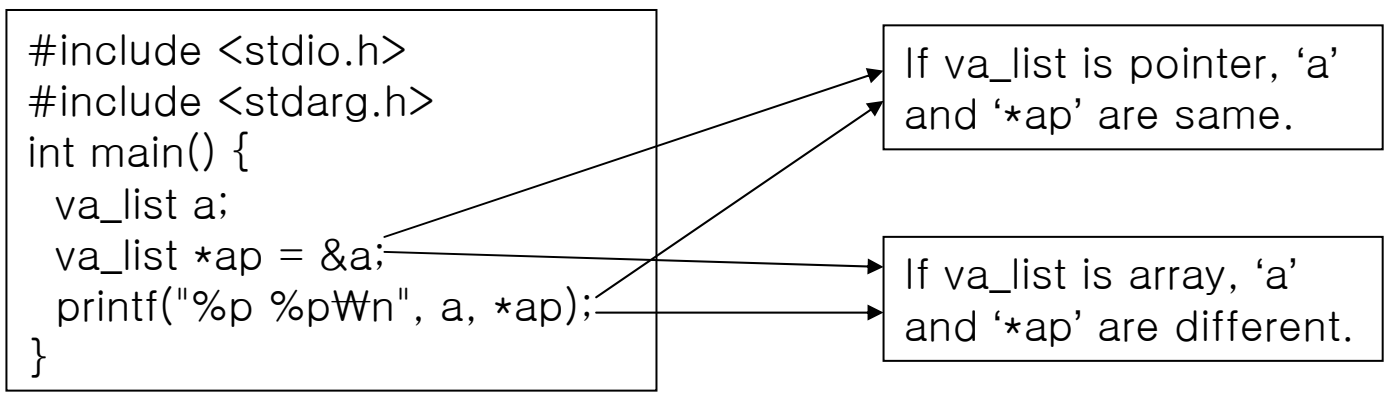
>> operator

- Do not assume >> is arithmetic (or logical)



va_list (1)

- Do not assume va_list is pointer (or array)



va_list (2)

- How to copy va_list?
  - If it is a pointer
    - va_list a, b; a = b;
  - If it is an array
    - va_list a, b; *a = *b;
  - See va_copy macro
    - va_copy is ANSI C (99) macro.

Data Type Size

- char, short, int, long, float, double, pointer size is platform dependent

Typical data type size (bit)

char	short	int	long	float	double	pointer
8	16	32	32	32	64	32

64bit Integer (1)

- In some compilers, 64bit integer is supported
  - In Microsoft Visual C++, __int64
  - In gcc, long long
- In some compilers, 64bit integer is not supported
  - Need to implement 64bit integer struct by combining two 32bit integers.

64bit Integer (2)

- 64bit Integer API from Alticast

ac_ll_shr(a, b) ac_ll_ushr(a, b) ac_ll_shl(a, b) ac_ll_ushl(a, b) ac_ll_or(a, b) ac_ll_uor(a, b) ac_ll_xor(a, b) ac_ll_uxor(a, b) ac_ll_and(a, b) ac_ll_uand(a, b) ac_ll_neg(a) ac_ll_uneg(a) ac_ll_not(a) ac_ll_unot(a)	ac_ll_add(a, b) ac_ll_uadd(a, b) ac_ll_sub(a, b) ac_ll_usub(a, b) ac_ll_mul(a, b) ac_ll_umul(a, b) ac_ll_div(a, b) ac_ll_udiv(a, b) ac_ll_mod(a, b) ac_ll_umod(a, b) ac_ll_eq(a, b) ac_ll_ueq(a, b) ac_ll_ne(a, b) ac_ll_une(a, b)	ac_ll_ge(a, b) ac_ll_uge(a, b) ac_ll_le(a, b) ac_ll_ule(a, b) ac_ll_lt(a, b) ac_ll_ult(a, b) ac_ll_gt(a, b) ac_ll_ugt(a, b) ac_ll_ll2ull(a) ac_ll_ull2ll(a) ac_ll_int2ll(a) ac_ll_ll2int(a) ac_ll_int2ull(a) ac_ll_ull2int(a)	ac_ll_uint2ull(a) ac_ll_ull2uint(a) ac_ll_uint2ll(a) ac_ll_ll2uint(a) ac_ll_ll2double(a) ac_ll_double2ll(a)

Floating Point Number

- Even if floating point number data size are same, the result of calculation can be slightly different
  - because internal floating point number data size are different for error correction.

Source Text Format

- Unix text file format is recommended
  - DOS text file format may cause an error in some compilers.
- The source code file should be ended with End-Of-Line (EOL) character
  - If not, it can cause an error in some compilers.

C Compiler Options

- Add warning options to reduce mistakes
- Microsoft Visual C++

  /W3 /WX
- gcc

```
-pedantic -W -Wall -Wshadow -Wpointer-arith -Wcast-align
-Waggregate-return -Wstrict-prototypes -Wmissing-prototypes
-Wmissing-declarations -Wnested-externs -Werror -Wno-unused
```

ANSI C (89)

- ANSI C (89) is more portable than assembly, ANSI C (99), C++
- Following ANSI C (89) strictly is very tough
  - However, should try to.
- JAVA is portable but needs JVM

OS Abstraction

- OS Abstraction API from Alticast

ac_t_create ac_t_delete ac_t_sleep ac_t_suspend ac_t_resume ac_t_setPriority ac_t_getPriority ac_t_self ac_t_comp ac_tsd_create ac_tsd_delete ac_tsd_set ac_tsd_get	ac_tm_set ac_tm_get ac_tm_setMillis ac_tm_getMillis ac_cv_create ac_cv_delete ac_cv_wait ac_cv_signal ac_cv_broadcast ac_q_create ac_q_delete ac_q_receive ac_q_send	ac_sm_create ac_sm_delete ac_sm_wait ac_sm_signal ac_mu_create ac_mu_delete ac_mu_lock ac_mu_unlock ac_mem_get ac_mem_release

Portable Java Application (1)

- Do not assume the external class implementation will not change in the future
  - Extending from external class may not be portable.
  - Serializing external class object may not be portable.

Portable Java Application (2)

Value.java (External class)

```
public class Value {
  double v;
  public void setValue(double v) {
    this.v = v;
  }
  public void setValue(int v) {
    setValue((double)v);
  }
  public double getValue() {
    return v;
  }
}
```

UValue.java (User class)

```
public class UValue extends Value {
  public void setValue(double v) {
    this.v = v * 2;
  }
  public static void main(String[] args) {
    UValue uv = new UValue();
    uv.setValue(10);
    System.out.println(uv.getValue());
  }
}
```

“uv.getValue()” returns “20.0”

If “setValue((double)y));” is changed to “this.v = (double)v;”, “uv.getValue()” will return “10.0”.

Portable Java Application (3)

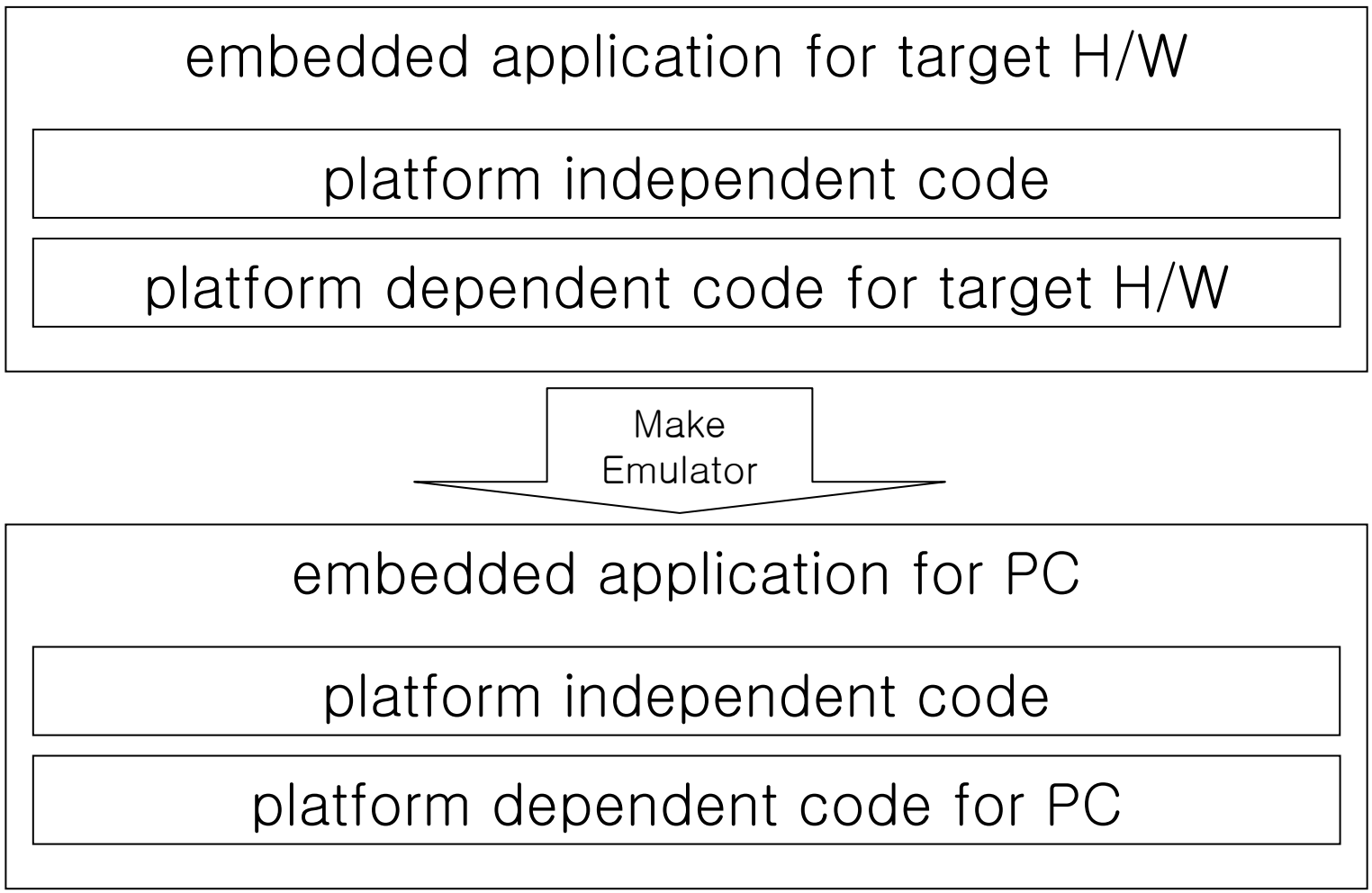
- System class can be GCed (Garbage Collected) in some JVM
  - A class static variable can become 0 (or null) long time later even if it is set by some value.
  - To prevent the class is GCed, create a dummy java object of the class.

Debugging Embedded Application

Debugging Embedded Application

- Emulator
- Serial I/O
- Remote Debugging

Emulator (1)


embedded application for target H/W
platform independent code
platform dependent code for target H/W
Make
Emulator
embedded application for PC
platform independent code
platform dependent code for PC

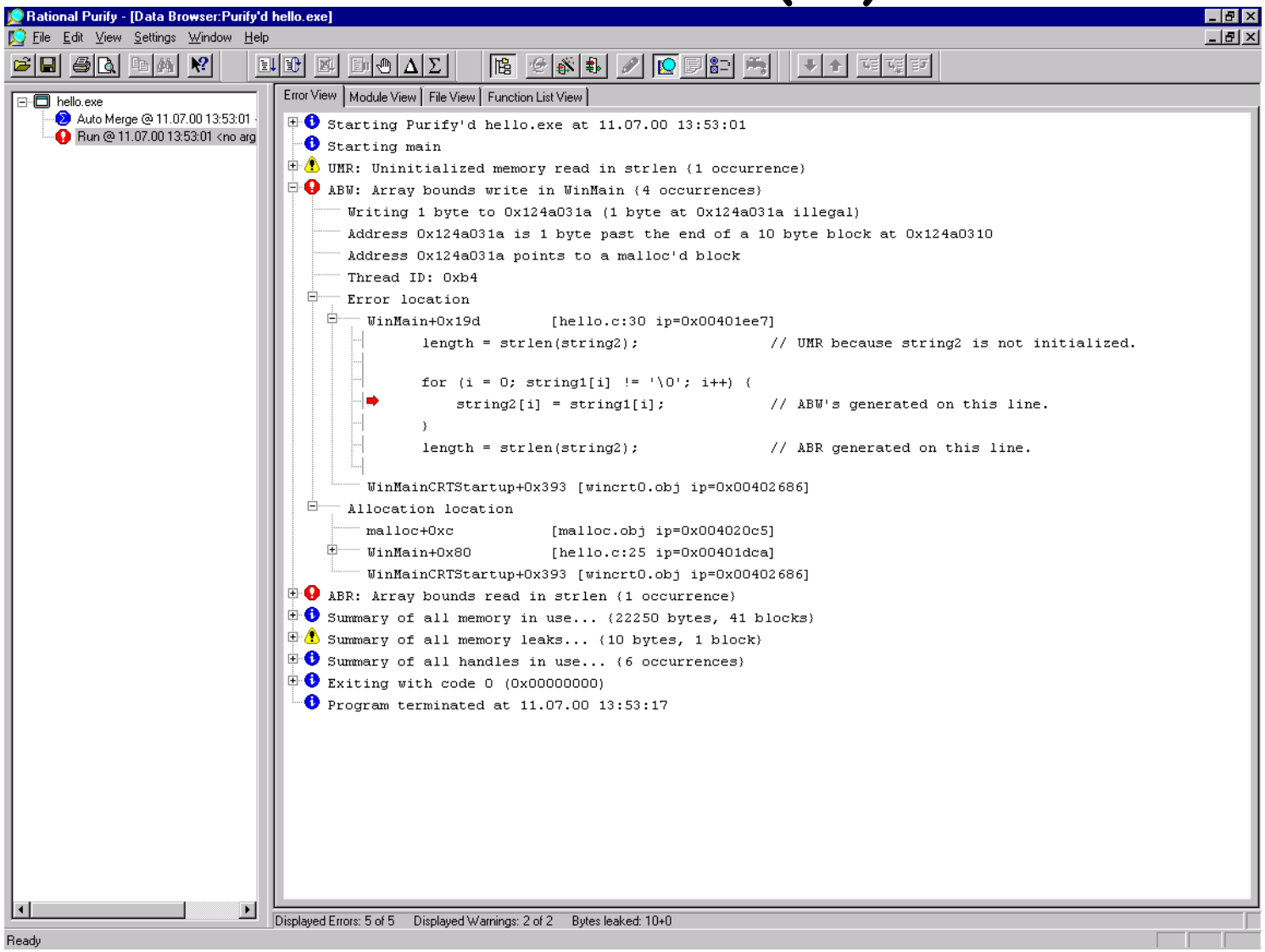
Emulator (2)

- Debugging in the emulator is much easier than in the target H/W
  - Many powerful debugging tools
    - MS Visual Studio Debugger
    - Rational PurifyPlus
    - Numega DevPartner
  - Short roundtrip test time
  - Doesn't need target H/W

Emulator (3)

- Rational Purify Can Detect
  - Uninitialized memory access
  - Array out of bounds read/write
  - Memory leak
  - Invalid memory free
  - Double free
  - Freed memory read/write

Emulator (4)


Rational Purify - [Data Browser:Purify'd hello.exe]
File Edit View Settings Window Help
hello.exe
Auto Merge @ 11.07.00 13:53:01
Run @ 11.07.00 13:53:01 <no arg
Error View
Module View
File View
Function List View
Starting Purify'd hello.exe at 11.07.00 13:53:01
Starting main
UMR: Uninitialized memory read in strlen (1 occurrence)
ABW: Array bounds write in WinMain (4 occurrences)
Writing 1 byte to 0x124a031a (1 byte at 0x124a031a illegal)
Address 0x124a031a is 1 byte past the end of a 10 byte block at 0x124a0310
Address 0x124a031a points to a malloc'd block
Thread ID: 0xb4
Error location
WinMain+0x19d [hello.c:30 ip=0x00401ee7]
length = strlen(string2); // UMR because string2 is not initialized.
for (i = 0; string1[i] != '\0'; i++) {
string2[i] = string1[i]; // ABW's generated on this line.
}
length = strlen(string2); // ABR generated on this line.
WinMainCRTStartup+0x393 [wincrt0.obj ip=0x00402686]
Allocation location
malloc+0xc [malloc.obj ip=0x004020c5]
WinMain+0x80 [hello.c:25 ip=0x00401dca]
WinMainCRTStartup+0x393 [wincrt0.obj ip=0x00402686]
ABR: Array bounds read in strlen (1 occurrence)
Summary of all memory in use... (22250 bytes, 41 blocks)
Summary of all memory leaks... (10 bytes, 1 block)
Summary of all handles in use... (6 occurrences)
Exiting with code 0 (0x00000000)
Program terminated at 11.07.00 13:53:17
Displayed Errors: 5 of 5
Displayed Warnings: 2 of 2
Bytes leaked: 10+0
Ready

Emulator (5)

- Debugging in the emulator is very good BUT
  - Can't debug platform dependent code for target H/W.
  - If platform independent code has platform dependent code accidentally, it may not be easy to debug.
  - There is something difficult to emulate.

Serial I/O

- Primitive but powerful debugging tool in the target H/W
- Needs to be
  - Stable
  - Non Buffered
- Structured debug message system is needed like syslog

Remote Debugging

- MultiICE, OpenICE, etc
  - In Circuit Emulator for ARM
  - Expensive
- MS eMbedded Visual C++
- gdb

Porting JVM PersonalJava 3.1

Porting JVM
Personal Java 3.1 (1)

- Overview
- Time
- Memory Allocation
- IO
- Startup
- Thread
- Monitor

Porting JVM
Personal Java 3.1 (2)

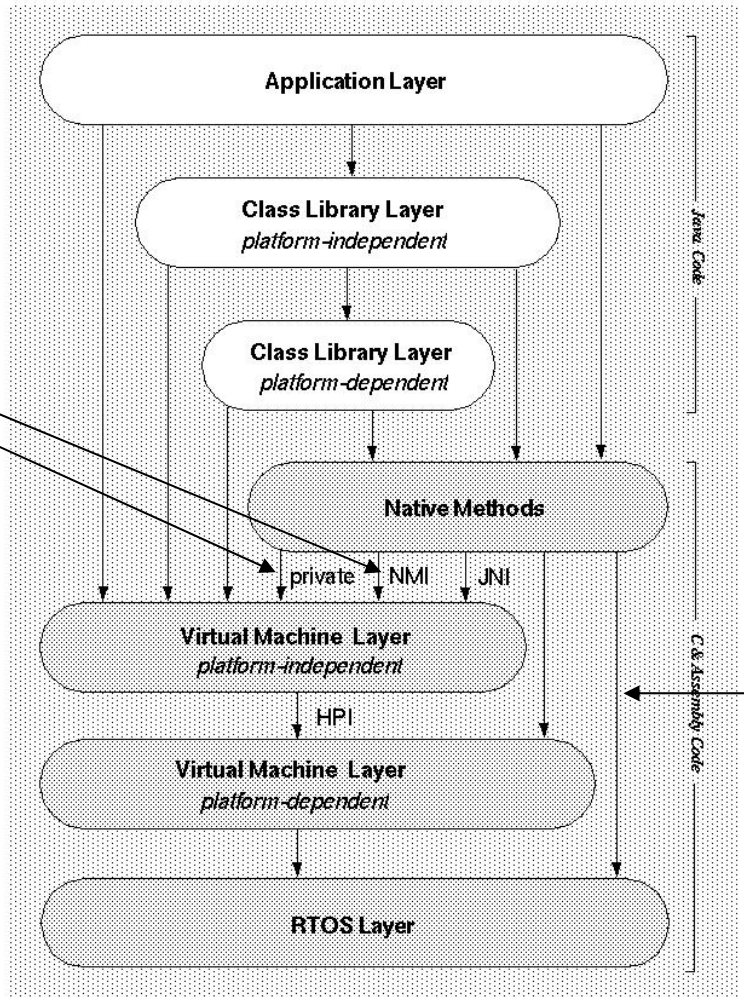
- Dynamic Linking
- Termination
- Miscellaneous
- Truffle, Network
- Test
- The Problems of Personal Java 3.1

Overview (1)

- Derived from JDK 1.1.8 virtual machine
- Java 2 security is integrated
- Not supported anymore
  - See CVM (CDC PBP)

Overview (2)

If JVM is changed, it will cause a problem.



PJAE Architecture

If RTOS is changed, it will cause a problem.

Time

- long sysGetMilliTicks(void);
  - Not needed.
  - Use sysTimeMillis, instead.
- int64_t sysTimeMillis(void);
  - Use ac_tm_getMillis.

Memory Allocation (1)

- void *sysMalloc(size_t size);
- void *sysRealloc(void *ptr, size_t size);
- void sysFree(void *ptr);
- void *sysCalloc(size_t nelem, size_t elsize);
  - Use ac_mem_get and Doug Lea's malloc.

Memory Allocation (2)

- void *sysMapMem(size_t requestedsize, size_t *mappedsize);
- void *sysUnmapMem(void *requestedaddress, size_t requestedsize, size_t *unmappedsize);
- void *sysCommitMem(void *requestedaddress, size_t requestedsize, size_t *unmappedsize);
- void *sysUncommitMem(void *requestedaddress, size_t requestedsize, size_t *unmappedsize);
  - Not needed.
  - Use sysMalloc, instead.

Memory Allocation (3)

- void *sysAllocBlock(size_t block, void **allocHead);
- void sysFreeBlock(void *allocHead);
  - Not needed.
  - Only for page mode.

IO (1)

- int sysAccess(const char *pFile, int perm);
- int sysStat(const char *path, struct stat *sbuf);
- int sysRename(const char *srcName, const char *dstName);
- int sysUnlink(const char *file);
- int sysMkdir(const char *path, int mode);
- int sysRmdir(const char *path);
- DIR *sysOpenDir(const char *path);
- int sysCloseDir(DIR *dp);
- struct dirent *sysReadDir(DIR *dp);
- int sysIsAbsolute(const char *path);
- int sysCanonicalPath(char *path, char *result, int result_len);
  - Create memory based Unix style file system.

IO (2)

- int sysOpenFD(Classjava_io_FileDescriptor *fd, const char *name, int openMode, int filePerm);
- int sysCloseFD(Classjava_io_FileDescriptor *fd);
- long sysSeekFD(Classjava_io_FileDescriptor *fd, long offset, int whence);
- size_t sysReadFD(Classjava_io_FileDescriptor *fd, void *buf, unsigned int nBytes);
- size_t sysWriteFD(Classjava_io_FileDescriptor *fd, const void *buf, unsigned int nBytes);
- size_t sysSyncFD(Classjava_io_FileDescriptor *fd);
- int sysAvailableFD(Classjava_io_FileDescriptor *fd, long *bytes);
- void sysInitFD(Classjava_io_FileDescriptor *fdobj, int fd);
  - Create memory based Unix style file system.

Startup

- void sysGetDefaultJavaVMInitArgs(void *args_);
- int sysInitializeJavaVM(void *ee_, void *args_);
  - Use solaris port source code.
- int sysFinalizeJavaVM(void *ee_);
  - Not needed.
  - JVM runs infinitely.
- void sysAttachThreadLock(void);
- void sysAttachThreadUnlock(void);
  - Not needed.
  - Don’t attach native thread to JVM by using JNI.

Thread (1)

- Use native thread (not green thread)
  - To communicate with non-java thread.
- int sysThreadBootstrap(sys_thread_t **ptid, void *cookie);
- void sysThreadInitializeSystemThreads(void);
  - Use solaris port source code.

Thread (2)

- int sysThreadCreate(long stack_size, uint_t flags, void (*start)(void *), sys_thread_t **ptid, void *cookie);
- void sysThreadExit(void);
- sys_thread_t *sysThreadSelf(void);
- void sysThreadYield(void);
- int sysThreadSuspend(sys_thread_t *tid);
- int sysThreadResume(sys_thread_t *tid);
- int sysThreadSetPriority(sys_thread_t *tid, int priority);
- int sysThreadGetPriority(sys_thread_t *tid, int priority);
  - Use ac_t_* API from Alticast portability layer.

Thread (3)

- void *sysThreadStackPointer(sys_thread_t *tid);
- stackp_t sysThreadStackBase(sys_thread_t *tid);
- void sysThreadSetStackBase(sys_thread_t *tid, stackp_t sp);
  - Difficult to port.
  - Set thread stack pointer as stack top when thread starts.
  - For garbage collection.

Thread (4)

- int sysThreadSingle(void);
- void sysThreadMulti(void);
- int sysThreadEnumerateOver(int (*)(sys_thread_t *, void *), void *arg);
- void sysThreadInit(sys_thread_t *tid, stackp_t stack);
- void *sysThreadGetBackPtr(sys_thread_t *t);
- int sysThreadAlloc(sys_thread_t **ptid, stackp_t stack_base, void *cookie);
- int sysThreadFree(sys_thread_t *tid);
  - Use solaris port source code.

Thread (5)

- void sysThreadInterrupt(sys_thread_t *tid);
  - Set interrupted bit in the sys_thread_t struct.
- int sysThreadIsInterrupted(sys_thread_t *tid, int ClearInterrupted);
  - Check interrupted bit in the sys_thread_t struct.
- void sysThreadPostException(sys_thread_t *tid, void *exc);
  - Set exception in the sys_thread_t struct.

Thread (6)

- int sysThreadCheckStack(void);
  - Use solaris port source code.
  - Use wider red zone than it in the solaris port source code.
- void sysThreadDumpInfo(sys_thread_t *tid);
  - Not needed.
  - Only for debug.

Monitor (1)

- size_t sysMonitorSizeof(void);
- int sysMonitorInit(sys_mon_t *mid);
- int sysMonitorDestroy(sys_mon_t *mid);
  - Use solaris port source code.
- bool_t sysMonitorEntered(sys_mon_t *mid);
- void sysMonitorDumpInfo(sys_mon_t *mid);
  - Not needed.
  - Only for debug.

Monitor (2)

- int sysMonitorEnter(sys_mon_t *mid);
- int sysMonitorExit(sys_mon_t *mid);
  - Use ac_mu_* API from Alticast portability layer.
  - Check interrupted bit periodically.
    - Polling

Monitor (3)

- int sysMonitorNotify(sys_mon_t *mid);
- int sysMonitorNotifyAll(sys_mon_t *mid);
- int sysMonitorWait(sys_thread_t *mid, sys_mon_t *millis, int64_t clear);
  - Use ac_cv_* API from Alticast portability layer.
  - Check interrupted bit periodically.

Monitor (4)

- sysCacheLockInit
- sysCacheLock
- sysCacheLocked
- sysCacheUnlock
  - Use solaris port source code.

Dynamic Linking (1)

- char *sysInitializeLinker(void);
  - Not needed.
  - Linker doesn't need to be initialized.
- int sysAddDLSegment(char *function);
- void sysBuildLibName(char *buf, int buflen, char *prefix, char *name);
  - Not needed.
  - The portable platform doesn't support dynamic library loading.

Dynamic Linking (2)

- long sysDynamicLink(char *symbol_name);
  - Create a big table of the function pointer and name string. And search the function pointer by its name string in the table. It is possible because application can't have native method and the portable platform doesn't support dynamic library loading.
- int sysBuildFunName(char *buf, int buflen, struct methodblock *mb, int encodingIndex);
  - Use solaris port source code.

Dynamic Linking (3)

- long *sysInvokeNative(JNIEnv_ *env, void *address, long *optop, char *sig, int argSize, void *staticRef);
  - Strictly, it's impossible to implement this in C language.
    - Because C language can't express calling the function whose number of arguments is not decided statically. So it should be implemented in assembly language.

Dynamic Linking (4)

- If the called native function always has 20 or less arguments, it can be implemented as follows in C language.

```
/* in the case of jint */
 sp[0].i = (*(jint (*)(jint, jint, jint, jint, jint, jint, jint, jint, jint, jint, jint, jint,
jint, jint, jint, jint, jint, jint, jint, jint))code)(args[0].i, args[1].i, args[2].i,
args[3].i, args[4].i, args[5].i, args[6].i, args[7].i, args[8].i, args[9].i,
args[10].i, args[11].i, args[12].i, args[13].i, args[14].i, args[15].i,
args[16].i, args[17].i, args[18].i, args[19].i);
```

It's not really portable. In some platforms, 32bit integer and 64bit integer have different alignment in the function call. In addition, some platforms passes floating point number as register and integer as stack.

Termination

- void sysExit(int status);
- int sysAtexit(void (*func)(void));
- void sysAbort(void);
  - Not needed.
  - JVM runs infinitely.

Miscellaneous

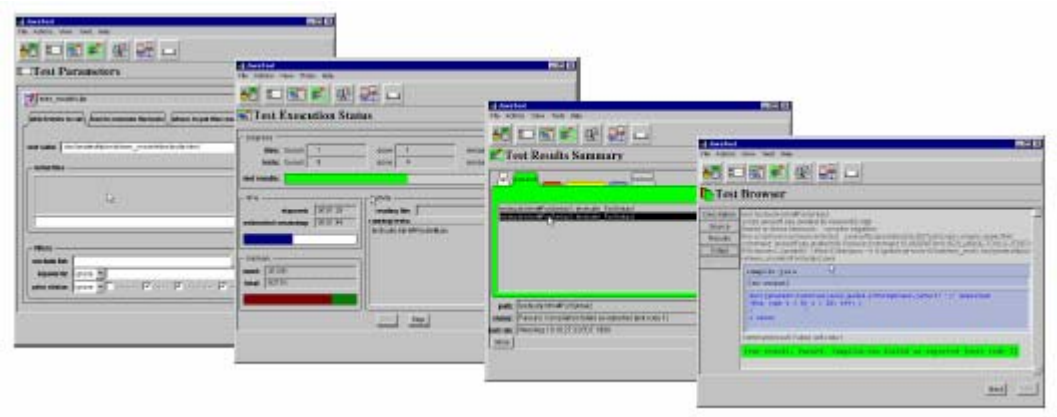
- sysCheckException(exception);
  - Use solaris port source code.
- sysStricmp(const char *s1, const char *s2);
  - Implement it in C. (trivial)
- cpe_t **sysGetBootClassPath(void);
- cpe_t **sysGetClassPath(void);
  - Use solaris port source code.

Truffle, Network

- Truffle
  - Modify it to make truffle more portable.
  - Use ac_g_* API from Alticast portability layer.
- Network
  - Use solaris port source code.
  - Use ac_socket_* API from Alticast portability layer.

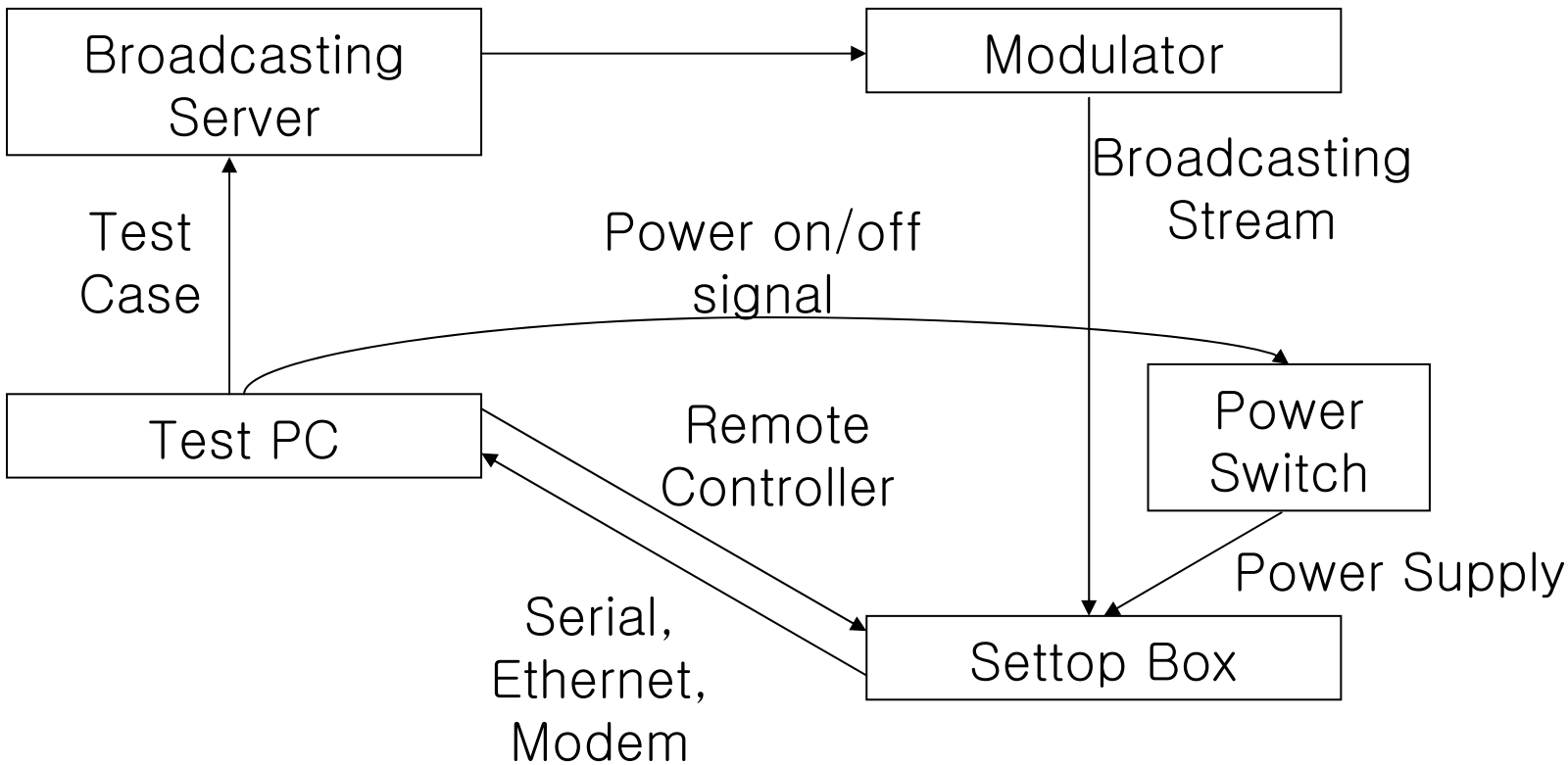
Test (1)

- Technology Compatibility Kit (TCK)
  - Sophisticated JVM compatibility test tool from Sun Microsystems.



Test (2)

- How to use TCK in the Settop Box?



The Problems of Personal Java 3.1 (1)

- Too old and too buggy
  - It's derived from JDK 1.1.8.
  - Not supported anymore.
    - See CDC PBP.
- Porting guide is not well documented
- The portable code assumes the platform supports ANSI C standard library and POSIX

The Problems of Personal Java 3.1 (2)

- The garbage collection has hard to fix bugs (or non portable codes)
  - Scan native stack to find alive java object.
  - Suspend threads unsafe way.

Porting JVM
CVM 1.0.1

Porting JVM
CVM 1.0.1 (1)

- defs.h
- doubleword.h
- endianness.h
- float.h
- globals.h
- int.h
- io.h

Porting JVM
CVM 1.0.1 (2)

- jni.h
- linker.h
- net.h
- path.h
- sync.h
- system.h
- threads.h

Porting JVM
CVM 1.0.1 (3)

- time.h

defs.h (1)

- Primitive Type Definition
- ANSI Header File Location
- CVM Port Header File Location
- Lock Optimization Definition

defs.h (2)

- CVMfloat32
- CVMfloat64
- CVMInt8
- CVMInt16
- CVMInt32
- CVMInt64
- CVMSize
- CVMUint8
- CVMUint16
- CVMUint32
- CVMUint64

defs.h (3)

- CVM_HDR_ANSI_ASSERT_H
- CVM_HDR_ANSI_CTYPE_H
- CVM_HDR_ANSI_ERRNO_H
- CVM_HDR_ANSI_LIMITS_H
- CVM_HDR_ANSI_SETJMP_H
- CVM_HDR_ANSI_STDARG_H
- CVM_HDR_ANSI_STDDEF_H
- CVM_HDR_ANSI_STDIO_H
- CVM_HDR_ANSI_STDLIB_H
- CVM_HDR_ANSI_STRING_H
- CVM_HDR_ANSI_TIME_H

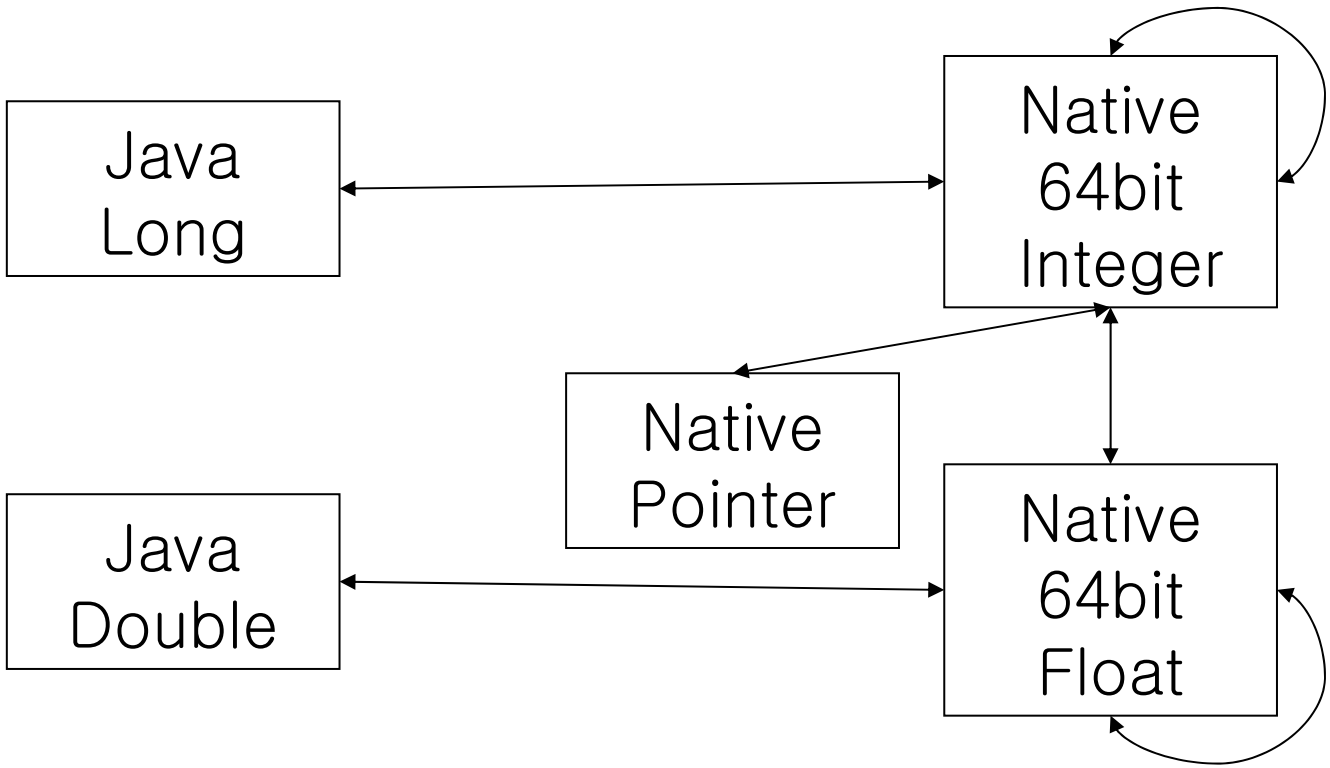
defs.h (4)

- CVM_HDR_DOUBLEWORD_H
- CVM_HDR_ENDIANNESS_H
- CVM_HDR_FLOAT_H
- CVM_HDR_GLOBALS_H
- CVM_HDR_INT_H
- CVM_HDR_IO_H
- CVM_HDR_JNI_H
- CVM_HDR_LINKER_H
- CVM_HDR_NET_H
- CVM_HDR_PATH_H
- CVM_HDR_SYNC_H
- CVM_HDR_THREADS_H
- CVM_HDR_TIME_H

defs.h (5)

- CVM_ADV_ATOMIC_CMPANDSWAP
- CVM_HAVE_PLATFORM_SPECIFIC_MICROLOCK
- CVM_ADV_ATOMIC_SWAP
- CVM_ADV_MUTEX_SET_OWNER
- CVM_ADV_SCHEDLOCK

doubleword.h (1)



doubleword.h (2)

- void CVMdouble2Jvm(CVMUint32 location[2], CVMJavaDouble val);
- CVMJavaDouble CVMjvm2Double(const CVMUint32 location[2]);
- CVMJavaLong CVMjvm2Long(const CVMUint32 location[2]);
- void CVMlong2Jvm(CVMUint32 location[2], CVMJavaLong val);
- void CVMmemCopy64(CVMUint32 to[2], const CVMUint32 from[2]);

doubleword.h (3)

- CVMJavaLong CVMdouble2Long(CVMJavaDouble val);
- CVMJavaLong CVMdouble2LongBits(CVMJavaDouble val);
- CVMJavaLong CVMint2Long(CVMJavaInt val);
- CVMJavaDouble CVMlong2Double(CVMJavaLong val);
- CVMJavaFloat CVMlong2Float(CVMJavaLong val);
- CVMJavaInt CVMlong2Int(CVMJavaLong val);
- void *CVMlong2VoidPtr(CVMJavaLong val);
- CVMJavaDouble CVMlongBits2Double(CVMJavaLong val);
- CVMJavaLong CVMvoidPtr2Long(void * val);

doubleword.h (4)

- CVMJavaLong CVMlongAdd(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongAnd(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongDiv(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongMul(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongOr(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongRem(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongSub(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongXor(CVMJavaLong op1, CVMJavaLong op2);

doubleword.h (5)

- CVMInt32 CVMlongCompare(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongConstZero();
- CVMJavaLong CVMlongConstOne();
- CVMInt32 CVMlongEq(CVMJavaLong op1, CVMJavaLong op2);
- CVMInt32 CVMlongEqz(CVMJavaLong op);
- CVMInt32 CVMlongGe(CVMJavaLong op1, CVMJavaLong op2);
- CVMInt32 CVMlongGez(CVMJavaLong op);
- CVMInt32 CVMlongGt(CVMJavaLong op1, CVMJavaLong op2);

doubleword.h (6)

- CVMInt32 CVMlongLe(CVMJavaLong op1, CVMJavaLong op2);
- CVMInt32 CVMlongLt(CVMJavaLong op1, CVMJavaLong op2);
- CVMInt32 CVMlongLtz(CVMJavaLong op);
- CVMInt32 CVMlongNe(CVMJavaLong op1, CVMJavaLong op2);
- CVMJavaLong CVMlongNeg(CVMJavaLong op);
- CVMJavaLong CVMlongNot(CVMJavaLong op);
- CVMJavaLong CVMlongShl(CVMJavaLong op1, CVMJavaInt op2);
- CVMJavaLong CVMlongShr(CVMJavaLong op1, CVMJavaInt op2);
- CVMJavaLong CVMlongUshr(CVMJavaLong op1, CVMJavaInt op2);

doubleword.h (7)

- CVMJavaFloat CVMdouble2Float(CVMJavaDouble val);
- CVMJavaInt CVMdouble2Int(CVMJavaDouble val);
- CVMJavaDouble CVMdoubleAdd(CVMJavaDouble op1, CVMJavaDouble op2);
- CVMInt32 CVMdoubleCompare(CVMJavaDouble op1, CVMJavaDouble op2, CVMInt32 direction);
- CVMJavaDouble CVMdoubleConstOne();
- CVMJavaDouble CVMdoubleConstZero();

doubleword.h (8)

- CVMJavaDouble CVMdoubleDiv(CVMJavaDouble op1, CVMJavaDouble op2);
- CVMJavaDouble CVMdoubleMul(CVMJavaDouble op1, CVMJavaDouble op2);
- CVMJavaDouble CVMdoubleNeg(CVMJavaDouble op);
- CVMJavaDouble CVMdoubleRem(CVMJavaDouble op1, CVMJavaDouble op2);
- CVMJavaDouble CVMdoubleSub(CVMJavaDouble op1, CVMJavaDouble op2);
- CVMJavaDouble CVMint2Double(CVMJavaInt val);

endianness.h

- CVM_DOUBLE_ENDIANNESS
  - Floating point number endianness
- CVM_ENDIANNESS
  - Integer endianness

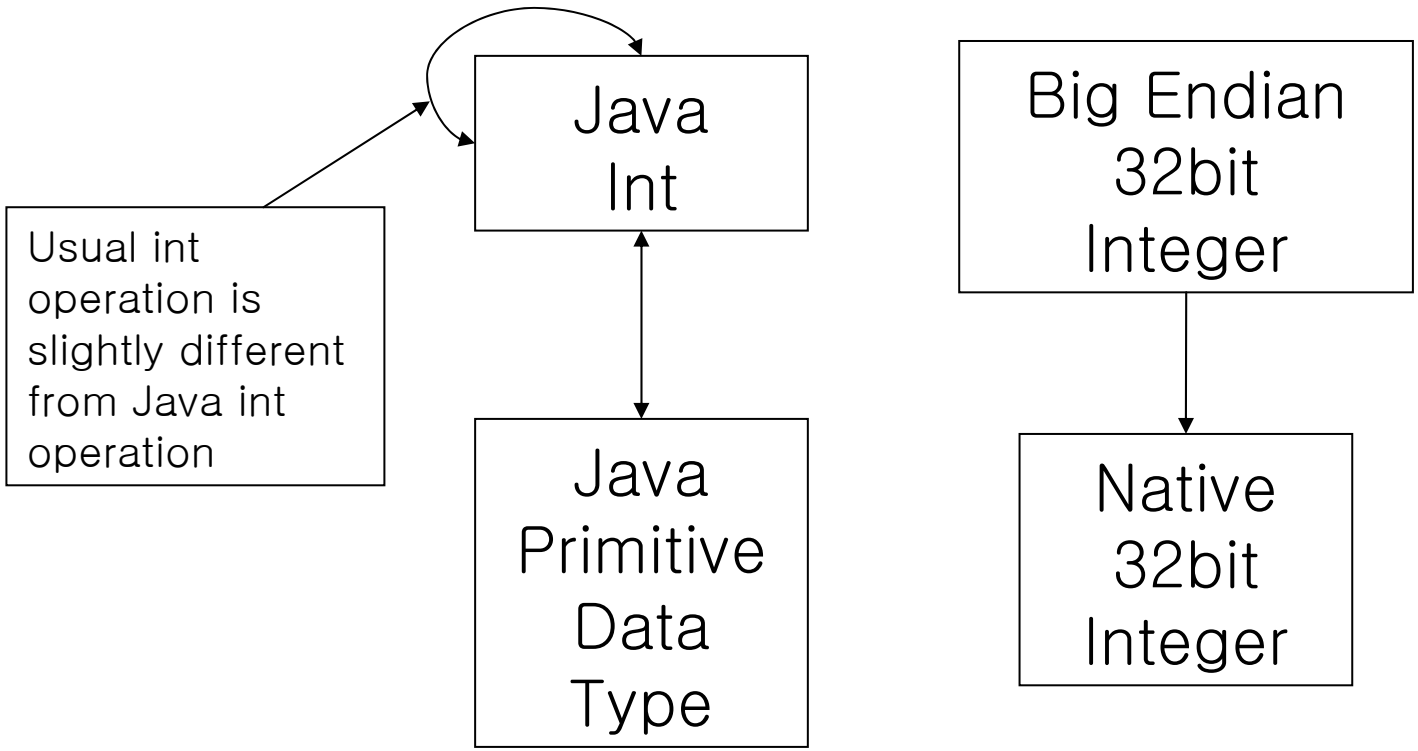
float.h

- CVMJavaDouble CVMfloat2Double(CVMJavaFloat op);
- CVMJavaInt CVMfloat2Int(CVMJavaFloat op);
- CVMJavaLong CVMfloat2Long(CVMJavaFloat op);
- CVMJavaFloat CVMfloatAdd(CVMJavaFloat op1, CVMJavaFloat op2);
- CVMInt32 CVMfloatCompare(CVMJavaFloat op1, CVMJavaFloat op2, CVMInt32 direction);
- CVMJavaFloat CVMfloatDiv(CVMJavaFloat op1, CVMJavaFloat op2);
- CVMJavaFloat CVMfloatMul(CVMJavaFloat op1, CVMJavaFloat op2);
- CVMJavaFloat CVMfloatNeg(CVMJavaFloat op1, CVMJavaFloat op2);
- CVMJavaFloat CVMfloatRem(CVMJavaFloat op1, CVMJavaFloat op2);
- CVMJavaFloat CVMfloatSub(CVMJavaFloat op1, CVMJavaFloat op2);

globals.h

- void CVMinitStaticState();
- void CVMdestroyStaticState();
  - Per address space static state. (Usually not used)
- struct CVMTargetGlobalState {};
- void CVMinitVMTargetGlobalState(CVMTargetGlobalState *);
- void CVMdestroyVMTargetGlobalState(CVMTargetGlobalState *);
  - Per VM global state. (Usually not used)
- const CVMProperties *CVMGetProperties(void);
  - Get Properties. (classpath, home dir, etc)

int.h (1)


Java
Int
Usual int
operation is
slightly different
from Java int
operation
Java
Primitive
Data
Type
Big Endian
32bit
Integer
Native
32bit
Integer

int.h (2)

- CVMJavaInt CVMintAdd(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintSub(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintMul(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintDiv(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintRem(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintAND(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintOr(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintXor(CVMJavaInt op1, CVMJavaInt op2);
- CVMJavaInt CVMintNeg(CVMJavaInt op);
- CVMJavaInt CVMintUshr(CVMJavaInt op1, CVMJavaInt num);
- CVMJavaInt CVMintShl(CVMJavaInt op1, CVMJavaInt num);
- CVMJavaInt CVMintShr(CVMJavaInt op1, CVMJavaInt num);

int.h (3)

- CVMJavaFloat CVMint2Float(CVMJavaInt val);
- CVMJavaByte CVMint2Byte(CVMJavaInt val);
- CVMJavaChar CVMint2Char(CVMJavaInt val);
- CVMJavaShort CVMint2Short(CVMJavaInt val);
- CVMUint16 CVMgetUint16(CVMconst CVMUint8 *ptr);
- CVMUint32 CVMgetUint32(CVMconst CVMUint8 *ptr);
- CVMInt16 CVMgetInt16(CVMconst CVMUint8 *ptr);
- CVMInt32 CVMgetInt32(CVMconst CVMUint8 *ptr);
- CVMInt32 CVMgetAlignedInt32(constCVMUint8 *ptr);

io.h (1)

- File System
  - Use memory based Unix style file system to make it portable in AltiCaptor.

io.h (2)

- CVMInt32 CVMioGetLastErrorString(char *buf, int len);
- char *CVMioReturnLastErrorString();
  - Not needed.
  - Only for debug.
- char *CVMioNativePath(char *path);
- CVMInt32 CVMioFileType(const char *path);
- CVMInt32 CVMioOpen(const char *name, CVMInt32 openMode, CVMInt32 filePerm);
- CVMInt32 CVMioClose(CVMInt32 fd);

io.h (3)

- CVMInt64 CVMioSeek(CVMInt32 fd, CVMInt64 offset, CVMInt32 whence);
- CVMInt32 CVMioSetLength(CVMInt32 fd, CVMInt64 length);
- CVMInt32 CVMioSync(CVMInt32 fd);
- CVMInt32 CVMioAvailable(CVMInt32 fd, CVMInt64 *bytes);
- size_t CVMioRead(CVMInt32 fd, void *buf, CVMUint32 nBytes);
- size_t CVMioWrite(CVMInt32 fd, const void *buf, CVMUint32 nBytes);
- CVMInt32 CVMioFileSizeFD(CVMInt32 fd, CVMInt64 *size);

jni.h

- CVMInt32 CVMjniInvokeNative(void *env, void *nativeCode, CVMUint32 *args, CVMUint32 *terseSig, CVMInt32 argsSize, void *classObject, CVMJNIReturnValue *returnValue);
  - Similar to sysInvokeNative of Personal Java 3.1.

linker.h

- void * CVMdynlinkOpen(const void *absolutePathName);
- void * CVMdynlinkSym(void *dsoHandle, const void *name);
- void * CVMdynlinkClose(void *dsoHandle);
- CVMBool CVMdynlinkBuildLibName(void *holder, int holderlen, void *pname, void *fname);
  - Similar to dynamic linking of Personal Java 3.1.

net.h (1)

- Socket
  - Use ac_socket_* APIs from Alticast portability layer.
  - There is no gethostbyaddr and gethostbyname. Why?

net.h (2)

- CVMInt32 CVMnetSocketClose(CVMInt32 fd);
- CVMInt32 CVMnetSocketShutdown(CVMInt32 fd, CVMInt32 howto);
- CVMInt32 CVMnetSocketAvailable(CVMInt32 fd, CVMInt32 *pbytes);
- CVMInt32 CVMnetConnect(CVMInt32 fd, struct sockaddr *him, CVMInt32 len);
- CVMInt32 CVMnetAccept(CVMInt32 fd, struct sockaddr *him, CVMInt32 *len);
- CVMInt32 CVMnetSendTo(CVMInt32 fd, char *buf, CVMInt32 len, CVMInt32 flags, struct sockaddr *to, CVMInt32 tolen);
- CVMInt32 CVMnetRecvFrom(CVMInt32 fd, char *buf, CVMInt32 nBytes, CVMInt32 flags, struct sockaddr *from, CVMInt32 *fromlen);
- CVMInt32 CVMnetListen(CVMInt32 fd, CVMInt32 count);

net.h (3)

- CVMInt32 CVMnetRecv(CVMInt32 fd, char *buf, CVMInt32 nBytes, CVMInt32 flags);
- CVMInt32 CVMnetSend(CVMInt32 fd, char *buf, CVMInt32 nBytes, CVMInt32 flags);
- CVMInt32 CVMnetTimeout(CVMInt32 fd, CVMInt32 timeout);
- CVMInt32 CVMnetSocket(CVMInt32 domain, CVMInt32 type, CVMInt32 protocol);
- CVMInt32 CVMnetSetSockOpt(CVMInt32 fd, CVMInt32 type, CVMInt32 dir, const void *arg, CVMInt32 argSize);
- CVMInt32 CVMnetGetSockOpt(CVMInt32 fd, CVMInt32 proto, CVMInt32 flag, void *in_addr, CVMInt32 *inSize);
- CVMInt32 CVMnetGetSockName(CVMInt32 fd, struct sockaddr *lclAddr, CVMInt32 *lclSize);
- CVMInt32 CVMnetGetHostName(char *hostname, CVMInt32 maxlen);
- CVMInt32 CVMnetBind(CVMInt32 fd, struct sockaddr *bindAddr, CVMInt32 size);

path.h (1)

- File Path Definition
  - Use Unix style path in AltiCaptor.

path.h (2)

- CVM_PATH_CLASSFILEEXT
- CVM_PATH_CLASSPATH_SEPARATOR
- CVM_PATH_CURDIR
- CVM_PATH_LOCAL_DIR_SEPARATOR
- CVM_PATH_MAXLEN
- int CVMcanonicalize(char* path, const char* out, int len);

sync.h (1)

- CVMBool CVMmutexInit(CVMMutex *m);
- void CVMmutexDestroy(CVMMutex *m);
- CVMBool CVMmutexTryLock(CVMMutex *m);
- void CVMmutexLock(CVMMutex *m);
- void CVMmutexUnlock(CVMMutex *m);
- CVMBool CVMcondvarInit(CVMCondVar *c, CVMMutex *m);
- void CVMcondvarDestroy(CVMCondVar *c);
- CVMBool CVMcondvarWait(CVMCondVar* c, CVMMutex *m, CVMJavaLong millis);
- void CVMcondvarNotify(CVMCondVar *c);
- void CVMcondvarNotifyAll(CVMCondVar *c);

sync.h (2)

- void CVMschedLock(void);
- void CVMschedUnlock(void);
- CVMint32 CVMatomicCompareAndSwap(volatile CVMUint32 *addr, CVMUint32 new, CVMUint32 old);
- CVMint32 CVMatomicSwap(volatile CVMUint32 *addr, CVMUint32 new);
- CVMUint32 CVMatomicIncrement(CVMUint32 *addr);
- CVMUint32 CVMatomicDecrement(CVMUint32 *addr);
- CVMBool CVMmicrolockInit(CVMMicroLock *m);
- void CVMmicrolockDestroy(CVMMicroLock *m);
- void CVMmicrolockLock(CVMMicroLock *m);
- void CVMmicrolockUnlock(CVMMicroLock *m);
- void CVMmutexSetOwner(CVMThreadId *self, CVMMutex *m, CVMThreadID *ti);
- CVM_FASTLOCK_TYPE
- CVM_MICROLOCK_TYPE

system.h

- void CVMhalt(CVMInt32 status);
- void CVMSystemPanic(const char *msg);
  - Not needed.
  - JVM runs infinitely.

threads.h (1)

- CVMInt32 CVMthreadCreate(CVMThreadID *thread, CVMSize stackSize, CVMInt16 priority, void (*func)(void *), void *arg);
- void CVMthreadYield(void);
- void CVMthreadSetPriority(CVMThreadID *thread, CVMInt32 prio);
- void CVMthreadSuspend(CVMThreadID *thread);
- void CVMthreadResume(CVMThreadID *thread);

threads.h (2)

- void CVMthreadAttach(CVMThreadID *self, CVMBool orphan);
- void CVMthreadDetach(CVMThreadID *self);
- CVMThreadID * CVMthreadSelf(void);
- void CVMthreadInterruptWait(CVMThreadID *thread);
- CVMBool CVMthreadIsInterrupted(CVMThreadID *thread, CVMBool clearInterrupted);
- CVMBool CVMthreadStackCheck(CVMThreadID *self, CVMUint32 redZone);

time.h

- CVMInt64 CVMtimeMillis(void);
  - Use ac_tm_getMillis.

JVM Optimization

JVM Optimization

- Synchronization
- Interpreter
- Class Files
- Garbage Collection

Synchronization (1)

- Few contentions in Java Synchronization
- CVM Fast Lock
  - 1) Try to lock with lightweight lock.
  - 2) If no contention occurred, go ahead.
    - No system call occurred. Fast.
  - 3) If contention occurred, lightweight lock will be converted to heavyweight lock and the set the mutex owner.

Synchronization (2)

- If “atomic compare and swap” and “atomic swap” are available
  - Intel x86, PowerPC, MIPS(R4000)

```
#define CVM_ADV_MUTEX_SET_OWNER
#define CVM_ADV_ATOMIC_CMPANDSWAP
#define CVM_ADV_ATOMIC_SWAP
#undef CVM_HAVE_PLATFORM_SPECIFIC_MICROLOCK

/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_NONE */
#define CVM_FASTLOCK_TYPE CVM_FASTLOCK_ATOMICOPS
/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_MICROLOCK */

#define CVM_MICROLOCK_TYPE CVM_MICROLOCK_DEFAULT
/* #define CVM_MICROLOCK_TYPE CVM_MICROLOCK_SCHEDLOCK */
```

Synchronization (3)

- If “atomic swap” is available
  - arm

```
#define CVM_ADV_MUTEX_SET_OWNER
#undef CVM_ADV_ATOMIC_CMPANDSWAP
#define CVM_ADV_ATOMIC_SWAP
#define CVM_HAVE_PLATFORM_SPECIFIC_MICROLOCK

/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_NONE */
/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_ATOMICOPS */
#define CVM_FASTLOCK_TYPE CVM_FASTLOCK_MICROLOCK

#define CVM_MICROLOCK_TYPE CVM_MICROLOCK_DEFAULT
/* #define CVM_MICROLOCK_TYPE CVM_MICROLOCK_SCHEDLOCK */
```

Synchronization (4)

- If “scheduler lock” is available
  - mips(r3000), ST

```
#define CVM_ADV_MUTEX_SET_OWNER
#undef CVM_ADV_ATOMIC_CMPANDSWAP
#undef CVM_ADV_ATOMIC_SWAP
#undef CVM_HAVE_PLATFORM_SPECIFIC_MICROLOCK

/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_NONE */
/* #define CVM_FASTLOCK_TYPE CVM_FASTLOCK_ATOMICOPS */
#define CVM_FASTLOCK_TYPE CVM_FASTLOCK_MICROLOCK

/* #define CVM_MICROLOCK_TYPE CVM_MICROLOCK_DEFAULT */
#define CVM_MICROLOCK_TYPE CVM_MICROLOCK_SCHEDLOCK
```

Interpreter (1)

- C interpreter
  - Portable.
  - Slow.
- Assembly interpreter
  - Not portable.
  - Faster than C interpreter.

Interpreter (2)

- Just-In-Time(JIT) Compiler
  - Not portable.
  - Faster than assembly interpreter.
  - Consumes a lot of ram.
  - Takes long time to execute at first because of compiling the java bytecodes.

Interpreter (3)

- Ahead-Of-Time(AOT) Compiler
  - Not portable.
  - Compiles java class files and creates native binary image.
  - JVM binary size becomes bigger.
    - Consumes rom (flash memory).
    - Native code is bigger than Java bytecode.
  - GCJ.

Interpreter (4)

- Hotspot
  - Not portable.
  - Consumes less ram than JIT.
  - Doesn't takes long time to execute at first because interpreting java bytecodes at first.
    - Analyzes the hotspot and compile it.

Interpreter (5)

- H/W Acceleration
  - Executes part of java bytecodes with H/W.
  - ARM, Nazomi.

Class Files (1)

- Romize Class Files
  - Short class loading time.
  - No extra ram is needed to load the class.
  - Redundant, useless Strings are removed.
  - Optimize bytecode.
    - custom bytecode, bytecode inlining, etc.
  - JVM binary size becomes bigger.
    - Consumes rom (flash memory).
  - Romizing widely used classes is recommended.

Class Files (2)

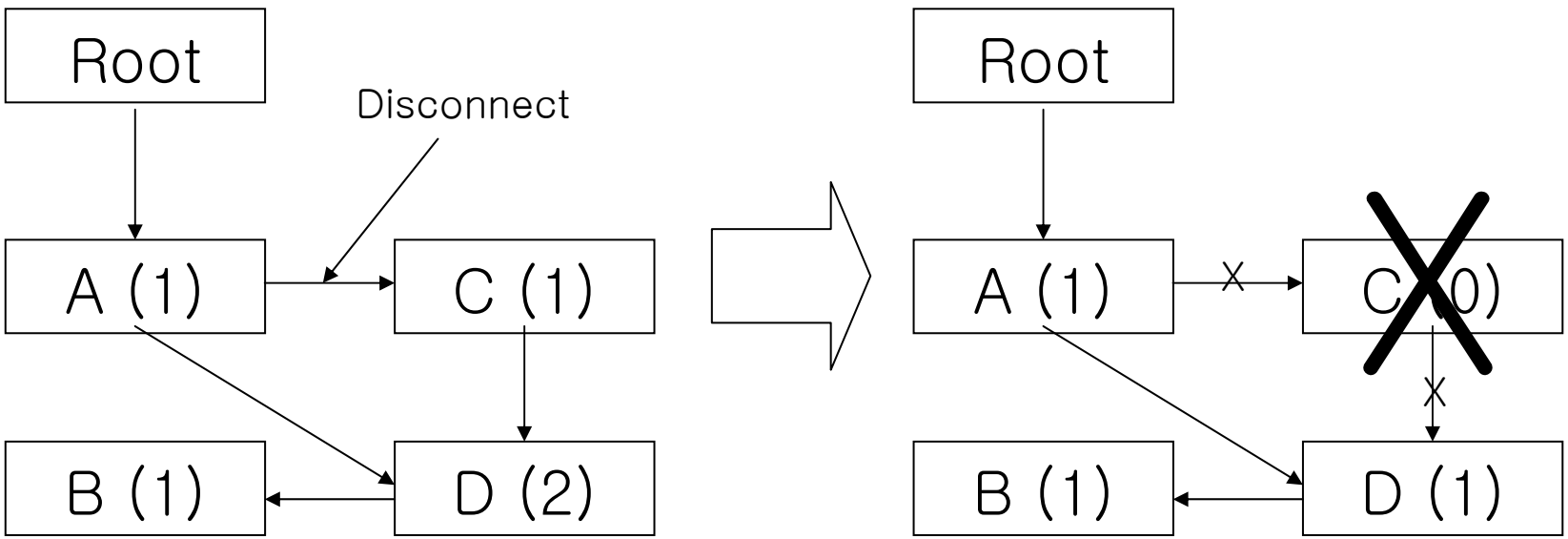
- Zip Class Files
  - Long class loading time.
    - Unzip and load.
  - Extra ram is needed to load the class.
  - Smaller JVM binary size than romized class files.
  - Zipping rarely used classes is recommended.

Class Files (3)

- Obfuscate Class Files
  - Make class files impossible to decompile.
  - Strings becomes shorter.
    - Reduces class files size.
    - Can cause a problem in JNI or reflect.

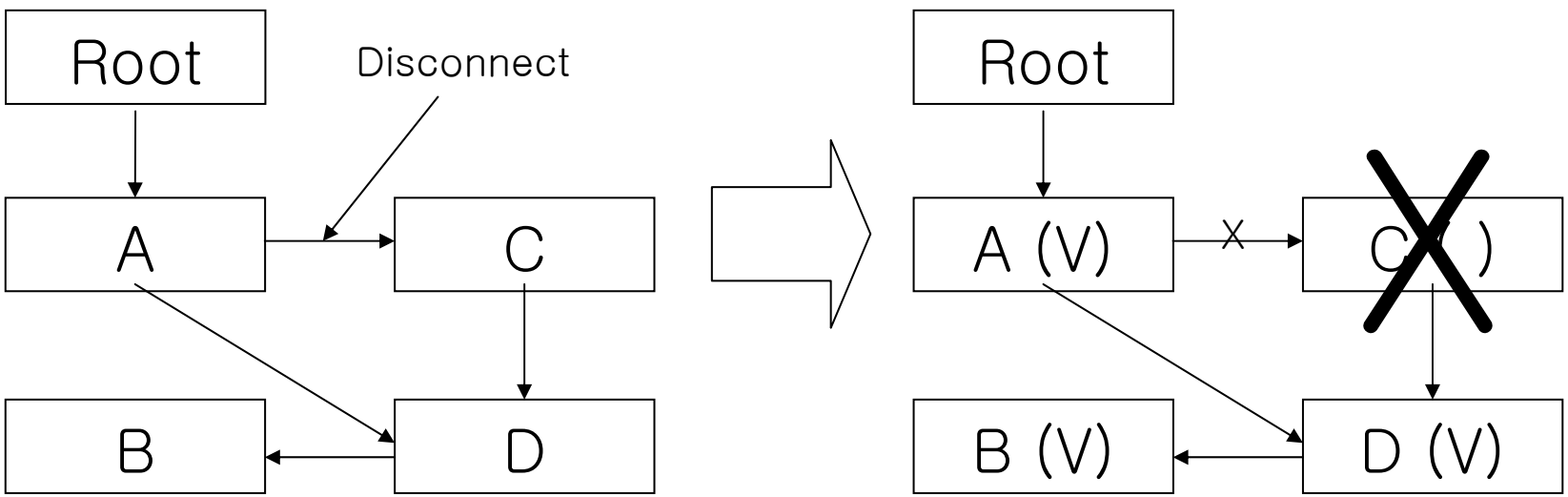
Garbage Collection (1)

- Reference Count
  - If there is a circular reference, no way to collect the garbage.



Garbage Collection (2)

- Mark and Sweep
  - Mark all referenced objects from root object and sweep unmarked (unreferenced) objects.



Garbage Collection (3)

  - Slow when there are many objects.
- Generational
  - Exploits recently created objects are reclaimed soon.
  - Newly created objects are young generation.
  - Long time alive objects are old generation.
  - Only young generation objects are reclaimed.
  - Need to track the reference from old generation objects to young generation objects.

Garbage Collection (4)

- CVM 1.0.1
  - Supports generational garbage collection.
  - Supports pluggable garbage collection algorithm.

Java Native Interface
(JNI)

Java Native Interface
(JNI)

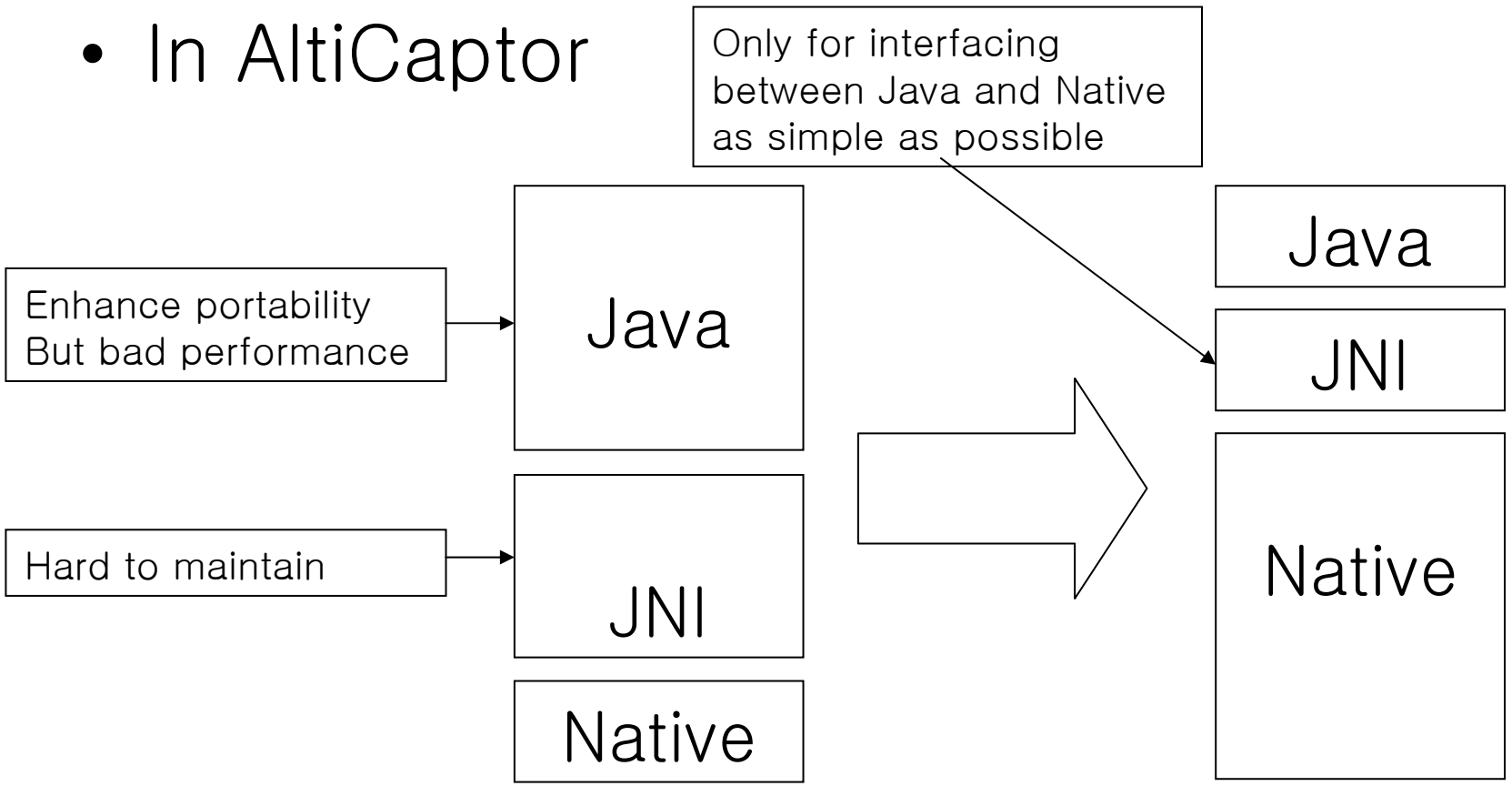
- What is JNI?
- Exception Handling
- Garbage Collection
- Reference Cache
- Thread Context
- Array Access
- Miscellaneous

What is JNI? (1)

- Standard Interface for JVM and Native Code
  - There are also non standard alternatives.
    - NMI, CNI, and so on.
  - Very portable but not efficient.
- With JNI
  - Call native function from Java method.
  - Almost everything Java method can do can be done in the native method.

What is JNI? (2)

- In AltiCaptor



What is JNI? (3)

- Book
  - “Java Native Interface”
    - Author : Liang
    - Publisher : Addison Wesley

Exception Handling (1)

```
JNIEXPORT int JNICALL
Java_ExceptionTest_getHashCode(JNIEnv *env, jobject obj)
{
    jclass objClass;
    jmethodID hashCodeMid;

    objClass = (*env)->FindClass(env, "com/alticast/lang/Object");
    hashCodeMid = (*env)->GetMethodID(env, objClass, "hashCode", "()I");

    return (*env)->CallIntMethod(env, obj, hashCodeMid);
}
```

Can be null if exception occurred.
Not easy to be sure unexpected exception (StackOverflowError, OutOfMemory, etc) will not occur.

Exception Handling (2)

```
JNIEXPORT int JNICALL
Java_ExceptionTest_getHashCode(JNIEnv *env, jobject obj) {
    jclass objClass;
    jmethodID hashCodeMid;
    objClass = (*env)->FindClass(env, "com/alticast/lang/Object");
    if (objClass == 0)
        return 0;
    hashCodeMid = (*env)->GetMethodID(env, objClass, "hashCode", "()I");
    if (hashCodeMid == 0)
        return 0;
    return (*env)->CallIntMethod(env, obj, hashCodeMid);
}
```

If exception occurred, return to java code as soon as possible.

Garbage Collection (1)

```
JNIEXPORT int JNICALL
Java_GCTest_getHashCode(JNIEnv *env, jobject obj) {
    jclass objClass;
    jmethodID hashCodeMid;
    objClass = (*env)->FindClass(env, "com/alticast/lang/Object");
    hashCodeMid = (*env)->GetMethodID(env, objClass, "hashCode", "()I");
    return (*env)->CallIntMethod(env, obj, hashCodeMid);
}
```

If GC occurred here, will "objClass" be GCed? -> No

Garbage Collection (2)

- Who has the reference to “objClass”?
  - Java stack allocated before calling Java_GCTest_getHashCode native function.
  - All JNI functions returning object reference (jclass, jobject, jarray, jstring) add the reference to java stack before returning the JNI function.

Garbage Collection (3)

- When is the reference to “objClass” deleted?
  - When the reference is deleted from the java stack by hand by calling DeleteLocalRef JNI function.
  - When returning from Java_GCTest_getHashCode native function.
    - Java stack is deleted when returning.

Garbage Collection (4)

- How to prevent “objClass” from being GCed even after returning from Java_GCTest_getHashCode?
  - Use NewGlobalRef JNI function.
    - If not calling DeleteGlobalRef, “objClass” will not GCed forever.

Garbage Collection (5)

```
JNIEXPORT int JNICALL
Java_GCTest_getHashCode(JNIEnv *env, jobject obj) {
    int i;
    jclass objClass;
    jmethodID hashCodeMid;
    for (I = 0; I < 10; i++)
    objClass = (*env)->FindClass(env, “com/alticast/lang/Object”);
    hashCodeMid = (*env)->GetMethodID(env, objClass, “hashCode”, “()I”);
    return (*env)->CallIntMethod(env, obj, hashCodeMid);
}
```

All “objClass” reference is alive in the java stack even if only one “objClass” is accessible. Therefore, all “objClass” will not be GCed before returning from Java_GCTest_getHashCode.

Reference Cache (1)

- Getting jclass, jfieldID, jmethodID takes long time without caching
  - FindClass, GetFieldID, GetMethodID is slow.

Reference Cache (2)

```
public class FidCache {
    private int a = 15;
    public native void getA();
    public static void main(String[] args) {
        FidCache fc = new FidCache();
        System.out.println(fc.getA());
    }
}
```

"objClass", "a_fid" become garbage after returning from Java_FidCache_getA.

Not always returns FidCache class. The care must be taken.

```
JNIEXPORT int JNICALL
Java_FidCache_getA(JNIEnv *env, jobject obj) {
    static jclass objClass = 0;
    static jfieldID a_fid = 0;
    if (objClass == 0) {
        objClass = (*env)->GetObjectClass(env, obj);
        a_fid = (*env)->GetFieldID(env, objClass, "a", "I");
    }
    return (*env)->GetIntField(env, obj, a_fid);
}
```

Reference Cache (3)

```
public class FidCacheChangeA extends FidCache {
    private int a = 20;
    public static void main(String[] args) {
        FidCacheChangeA fcca = new FidCacheChangeA();
        System.out.println(fcca.getA());
    }
}
```


Print “20”.
The value of the private field “a” is changed.
It may or may not be intentional but it is very confusing.

Reference Cache (4)

```
public class FidCache {
  static { initJNI(); }
  private native static initJNI();
  private int a = 15;
  public native void getA();
  public static void
   main(String[] args) {
    FidCache fc = new FidCache();
    System.out.println(fc.getA());
  }
}
```

Java_FidCache_initJNI is called when FidCache class is loaded.

```
static jfieldID _fid_FidCache_a;
JNIEXPORT void JNICALL
Java_FidCache_initJNI(JNIEnv *env,
  jclass clazz) {
  _fid_FidCache_a =
    (*env)->GetFieldID(env, clazz, "a", "I");
}
JNIEXPORT int JNICALL
Java_FidCache_getA(JNIEnv *env,
  jobject obj) {
  return
    (*env)->GetIntField(env, obj, a_fid);
}
```

Even if "obj" is not FidCache instance, returns FidCache.a.

Reference Cache (5)

```
public class HardFidCache {
  static { initJNI(); }
  private native static initJNI();
  public native void getA(FidCache obj);
  public static void main(String[] args) {
    FidCache fc = new FidCache();
    HardFidCache hfc = new HardFidCache();
      System.out.println(hfc.getA(fc));
    }
}
```

Reference Cache (6)

```
static jfieldID _fid_FidCache_a;
static jclass _class_FidCache;
JNIEXPORT void JNICALL
Java_HardFidCache_initJNI(JNIEnv *env, jclass clazz) {
    if (_class_FidCache == 0 || _fid_FidCache_a == 0) {
        jclass local_class;
        local_class = (*env)->FindClass(env, “FidCache”);
        if (local_class == 0) return;
        _fid_FidCache_a = (*env)->GetFieldID(env, local_class, “a”, “I”);
        if (_fid_FidCache_a == 0) return;
        _fid_FidCache = (*env)->NewGlobalRef(env, local_class);
        if (_fid_FidCache == 0) return;
        (*env)->DeleteLocalRef(env, local_class);
    }
}
JNIEXPORT int JNICALL
Java_HardFidCache_getA(JNIEnv *env, jobject obj, jobject obj_HardFidCache) {
    if (obj_HardFidCache == 0) return 0;
    return (*env)->GetIntField(env, obj_HardFidCache, _fid_FidCache_a);
}
```

Reference Cache (7)

- No Class Finalizer
  - No way to call DeleteGlobalRef for the NewGlobalRef when class initializer is called.
  - Potential memory leaks.
  - Use NewWeakGlobalRef.
    - Too complex to use.

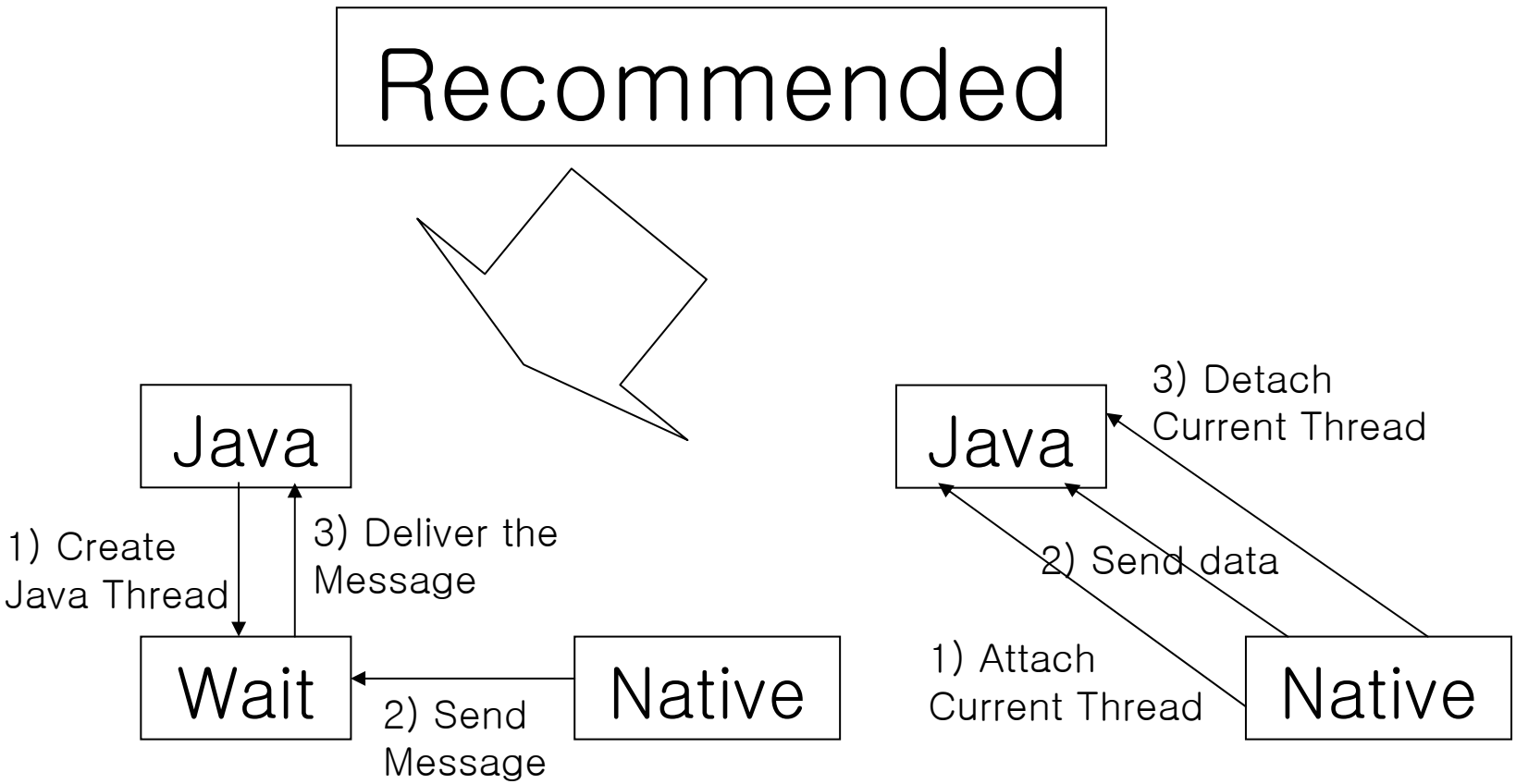
Thread Context (1)

- Only Java thread can use JNI
- If the native thread want to send data to Java thread
  - 1) Create new Java thread.
  - 2) Call native method which waits for the messages from native thread by using message queue.
  - 3) Get the message and call Java method with the message.

Thread Context (2)

- Or use AttachCurrentThread, DetachCurrentThread.
  - The Java thread model and native thread model must be same.
    - It can be difficult in some cases.
  - Not recommended.

Thread Context (3)



Array Access (1)

- (Get,Release)(Type)ArrayElements
  - Buffer copy. Slow.
  - If not call “Release”, memory will leak.
  - Undo is possible.
- (Get,Release)(Type)ArrayCritical
  - Direct access to java heap. Fast.
  - If not call “Release”, java heap fragmentation may occur.

Array Access (2)

- (Get,Set)(Type)ArrayRegion
  - Buffer copy. Slow.
  - Not need to call “Release”.
- String
  - Unicode is faster than UTF.
    - JVM uses Unicode String representation.

Miscellaneous (1)

- Do not forget to call ReleaseXXX after calling GetXXX
  - If forget, it may cause memory leak.
- Native memory allocated in the native method is not GCed
  - Need to free manually.

Miscellaneous (2)

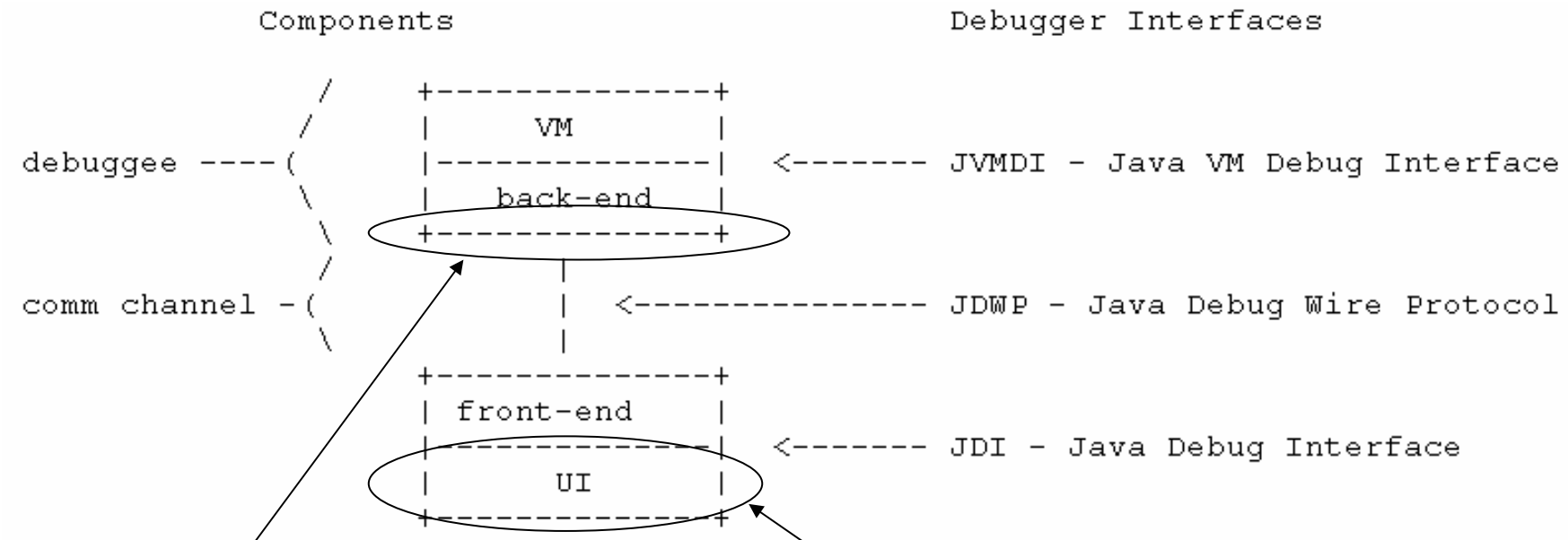
- Check the argument passed from java method to native method carefully
  - If not, it can cause whole JVM crash.

JVMDI, JVMPI

JPDA, JVMPI

- Java Platform Debugger Architecture (JPDA)
- Java Virtual Machine Profiler Interface (JVMPI)

Java Platform Debugger Architecture (JPDA) (1)



Port here.
Usually based on socket.

Many third party debuggers using JDI are available.
JDB, NetBeans IDE, JSwat, JDebugTool, JBuilder, WebSphere Studio, BugSeeker, and so on.

Java Platform Debugger Architecture (JPDA) (2)

- Java Source Level Debug
- Easy to Port to Embedded Target
  - If socket is available.
- Sun JDK Provides “Socket Connection” and “Shared Memory Connection”

Java Platform Debugger Architecture (JPDA) (3)

Wait for debugger connection

```
$ javac -g HelloWorld.java

$ java -Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8000 HelloWorld
```

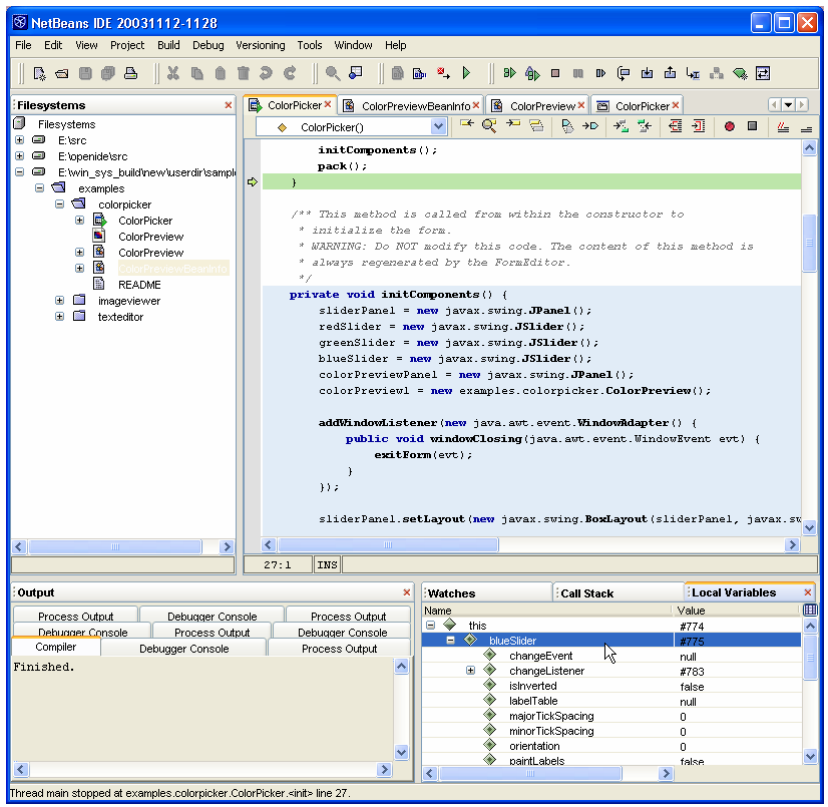
Connect to debugee

```
$ jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=8000
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
>
VM Started: No frames on the current call stack

main[1]
```

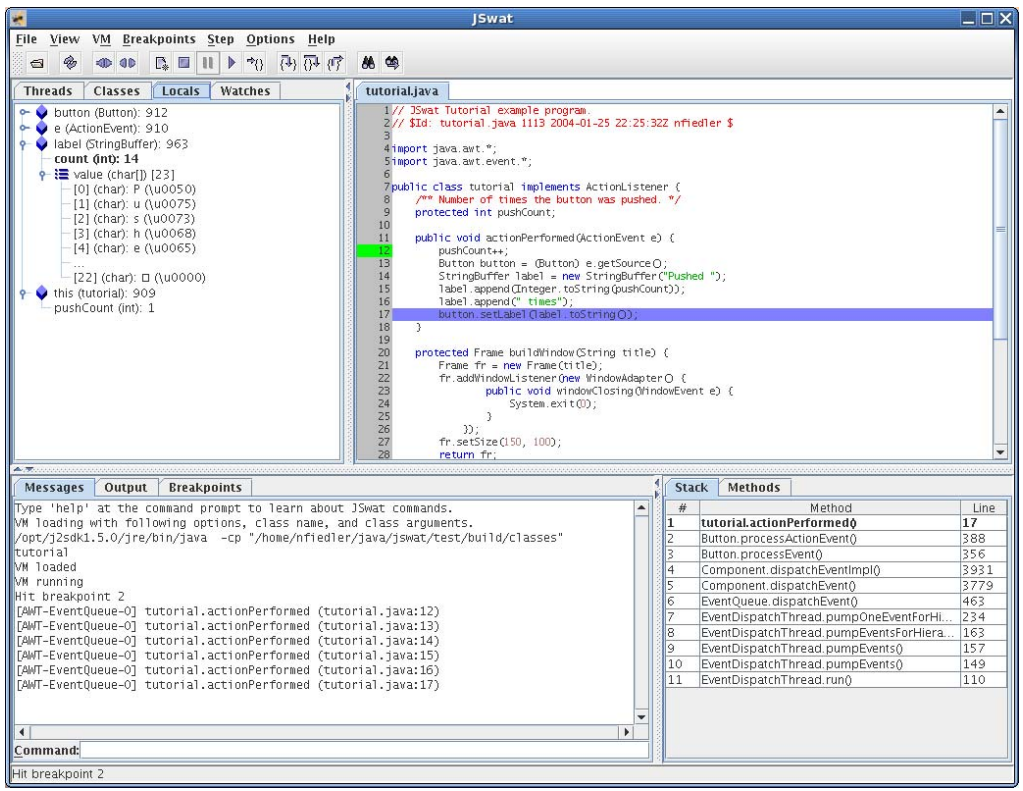
Java Platform Debugger Architecture (JPDA) (4)

- NetBeans IDE Screenshot

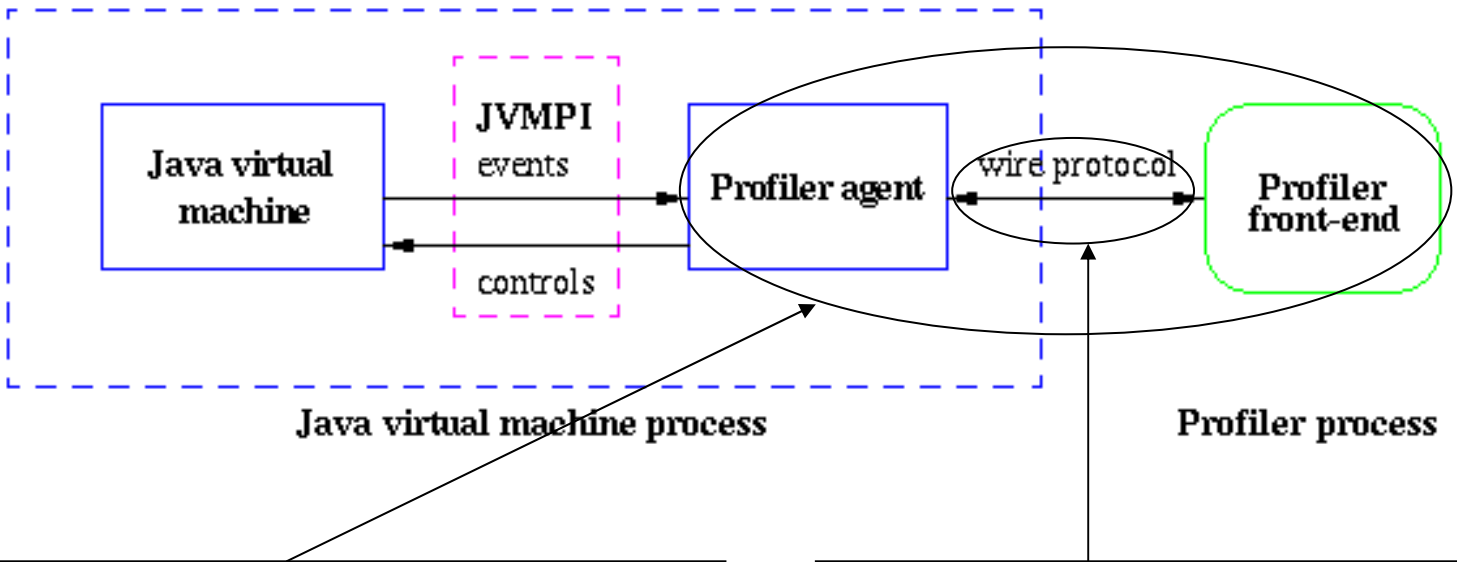


Java Platform Debugger Architecture (JPDA) (5)

- JSwat Screenshot



Java Virtual Machine Profiler Interface (JVMPI) (1)



Because there is no standard wire protocol, the profiler should handle the large potion. Absolutely, JVM platform dependent.

No standard wire protocol is available.
Not easy to use third party profiler.

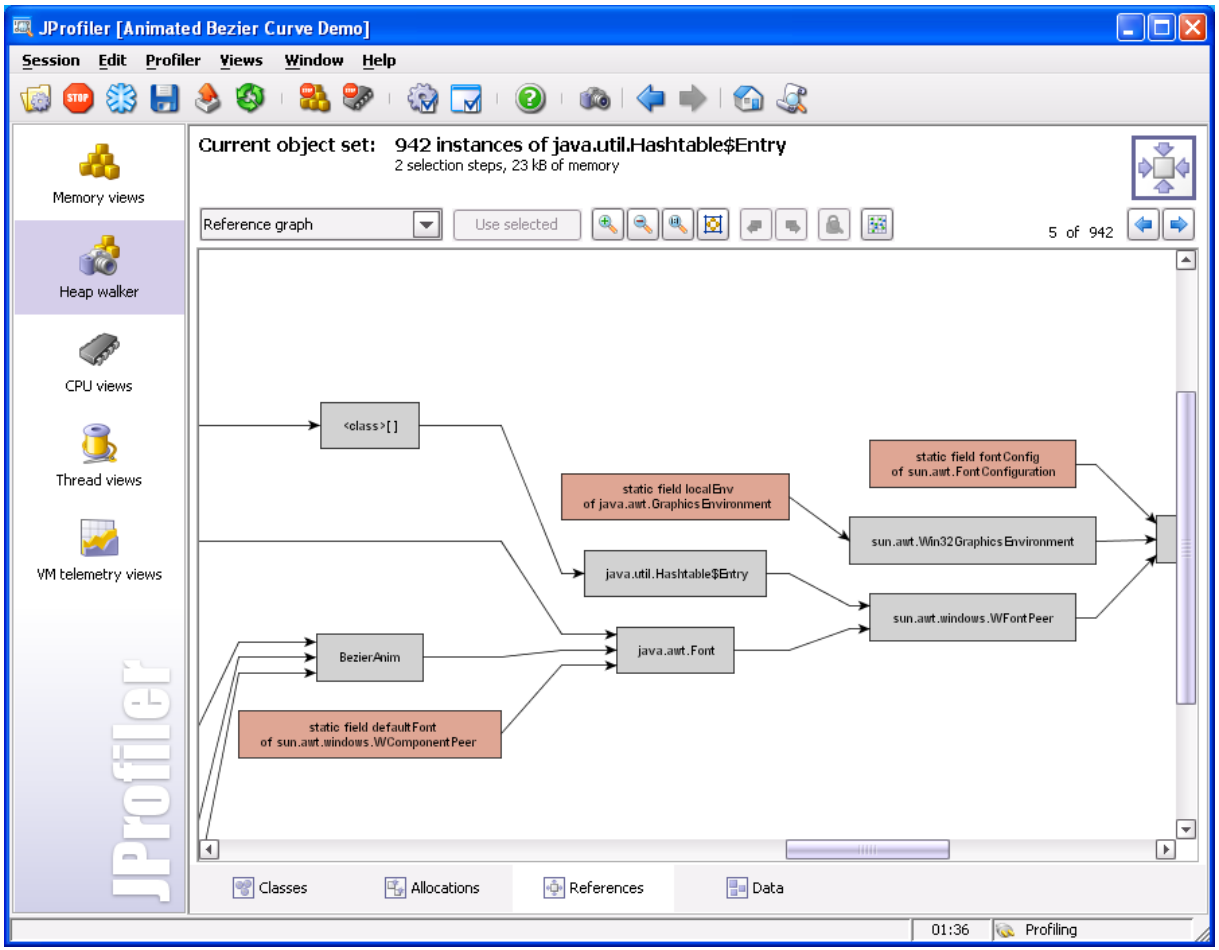
Java Virtual Machine Profiler Interface (JVMPI) (2)

- Several third party profiler available
  - But usually, JVM platform dependent and source code is not available.
  - Not easy to use in the target H/W.
  - hprof, JProfiler, EJP, Optimizeit, Quantify, YourKit Java Profiler, JProbe, jProf, and so on.

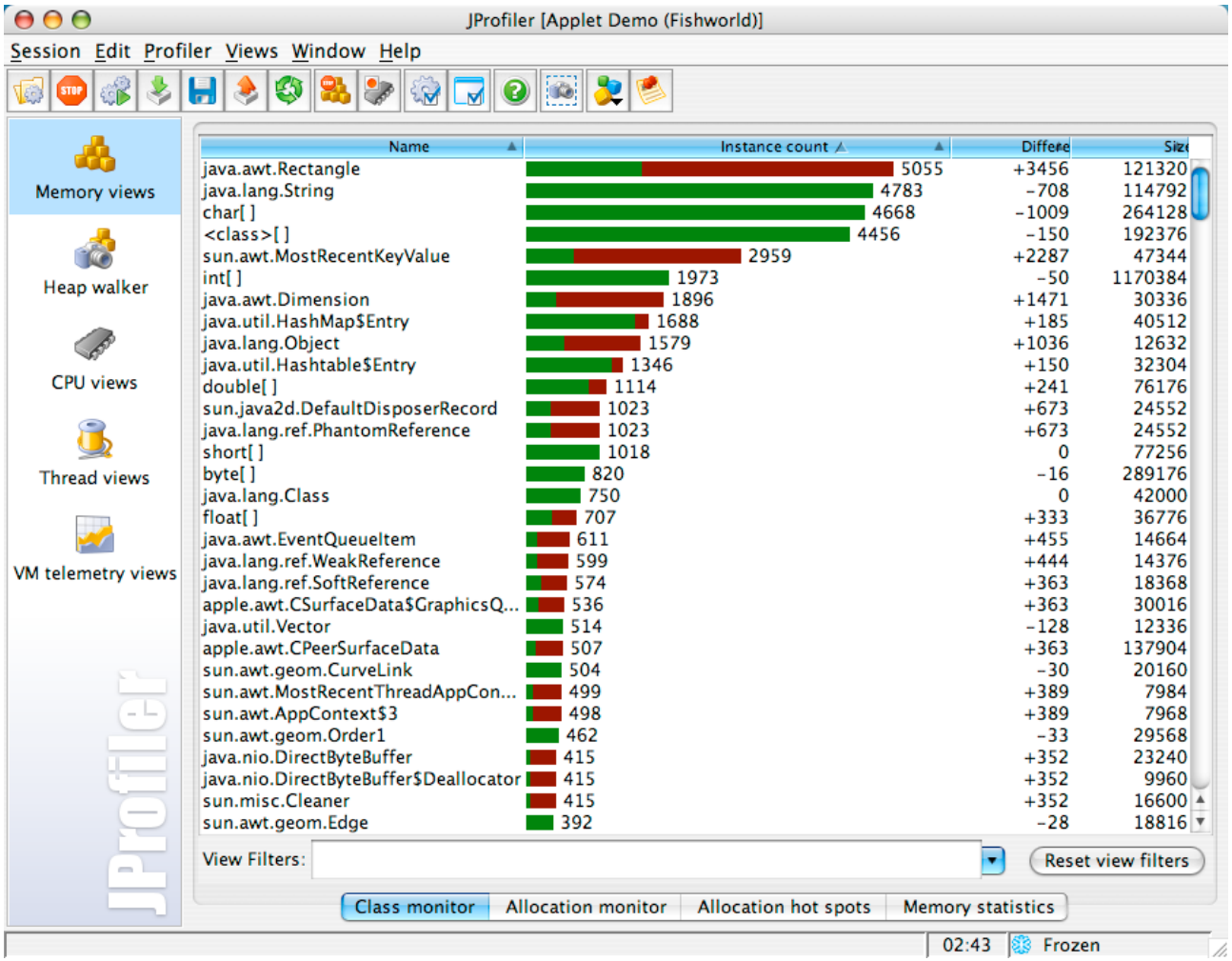
Java Virtual Machine Profiler Interface (JVMPI) (3)

- JProfiler
  - Invocation Tree
  - Memory Monitor
  - Heap Walker - References
  - Call Graph
  - Allocation Hotspots
  - Thread History
  - Automatic Deadlock Detection
  - Telemetry - Heap Usage
  - Monitor usage history

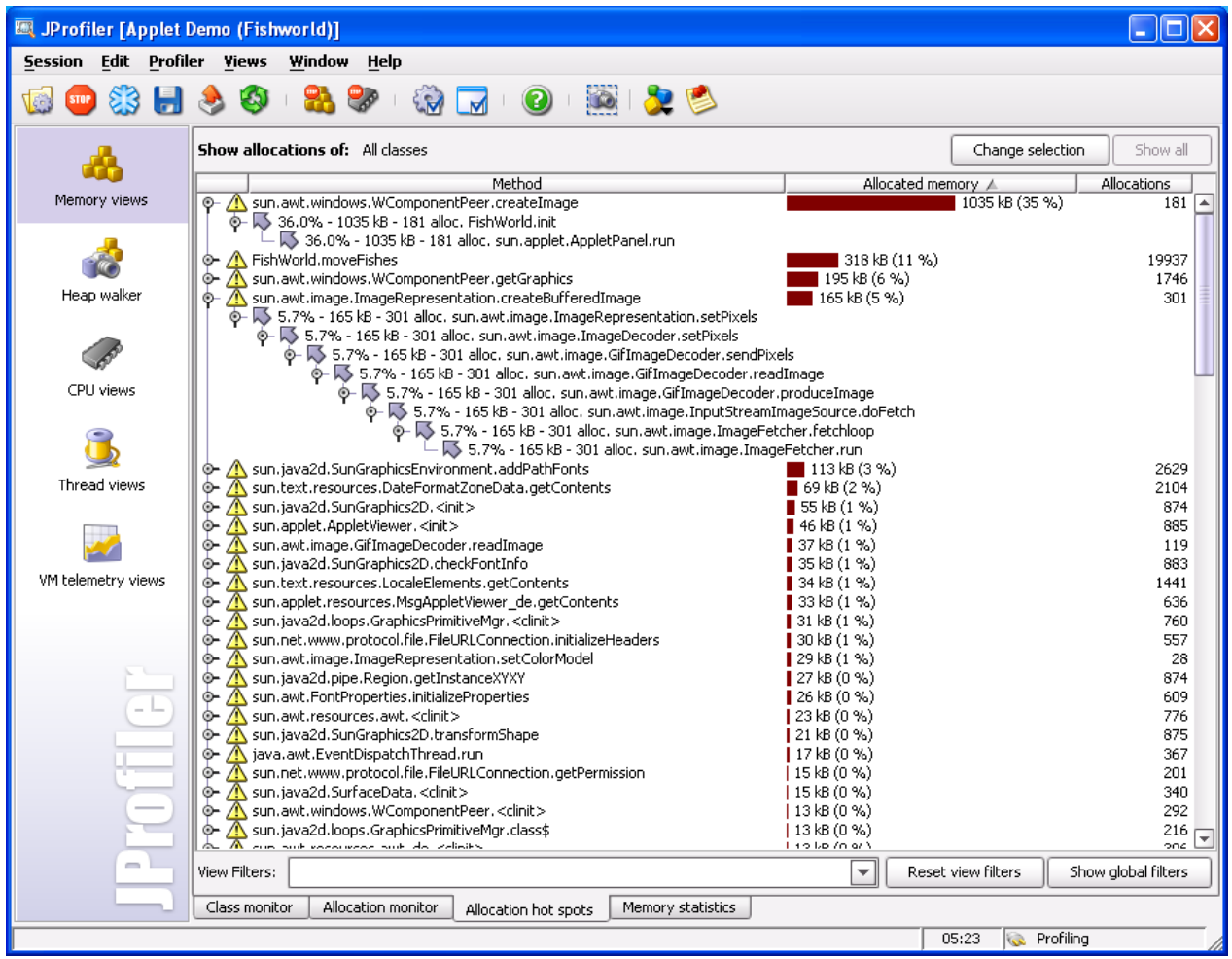
Java Virtual Machine Profiler Interface (JVMPI) (4)



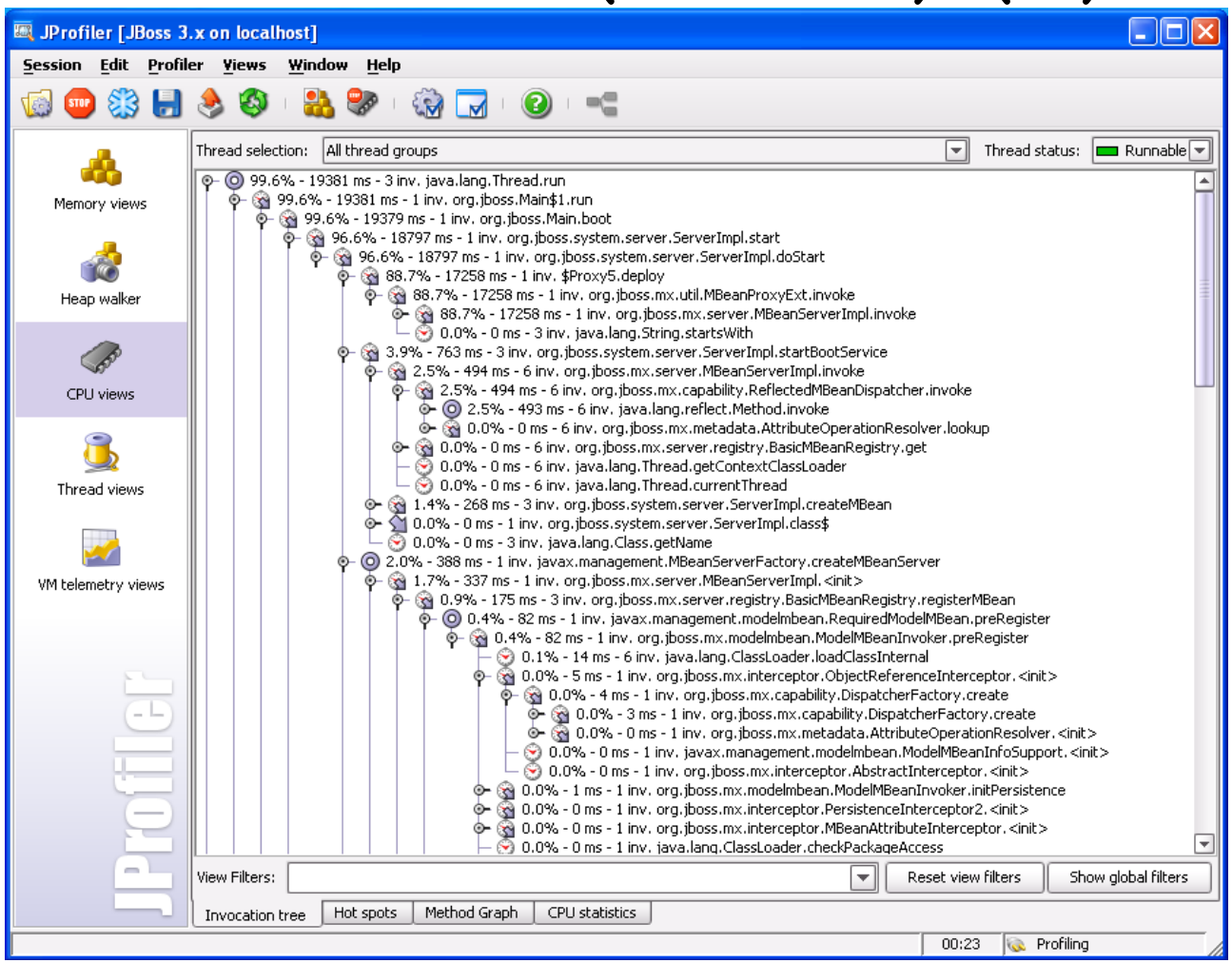
Java Virtual Machine Profiler Interface (JVMPI) (5)



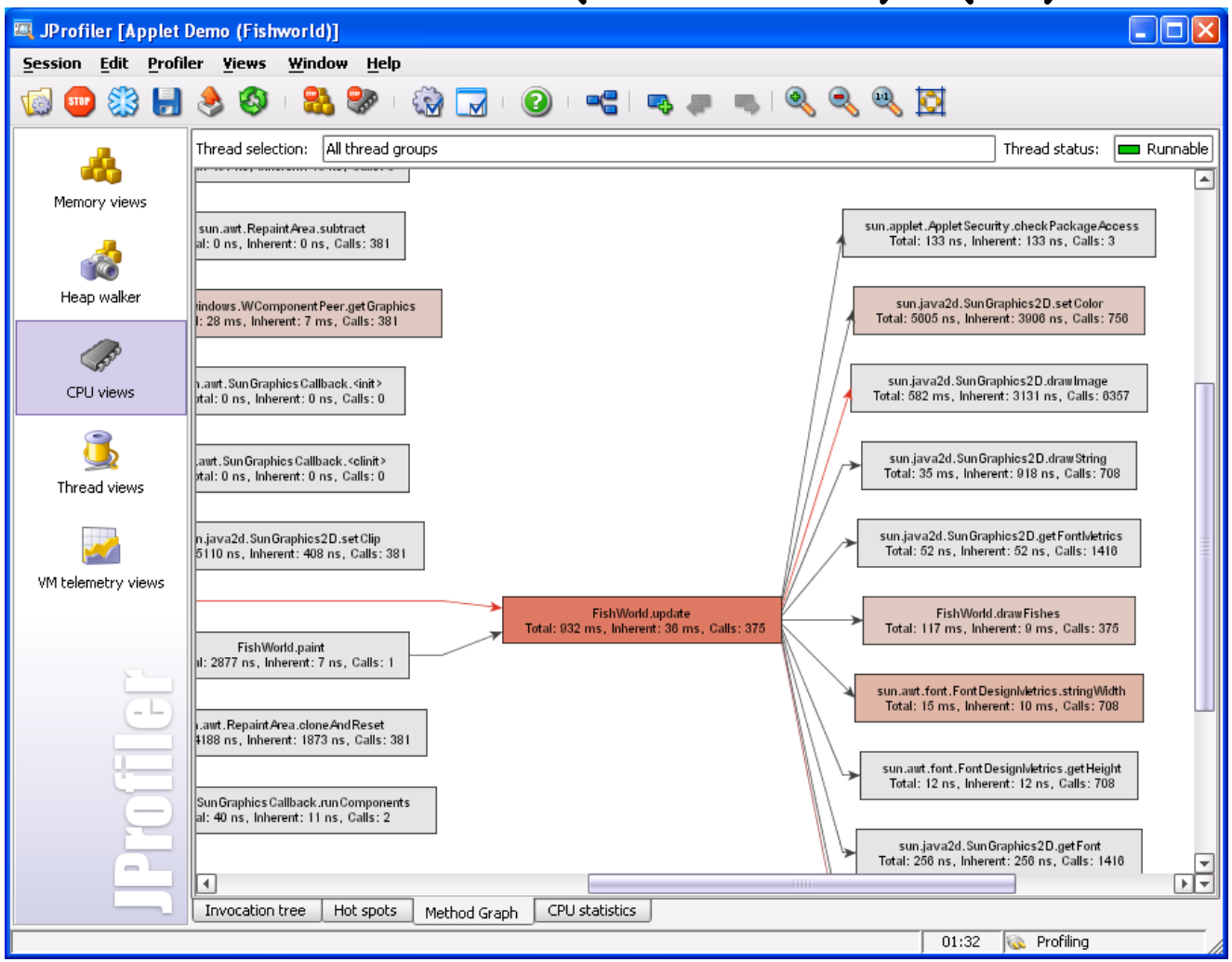
Java Virtual Machine Profiler Interface (JVMPI) (6)



Java Virtual Machine Profiler Interface (JVMPI) (7)



Java Virtual Machine Profiler Interface (JVMPI) (8)



Resource Management

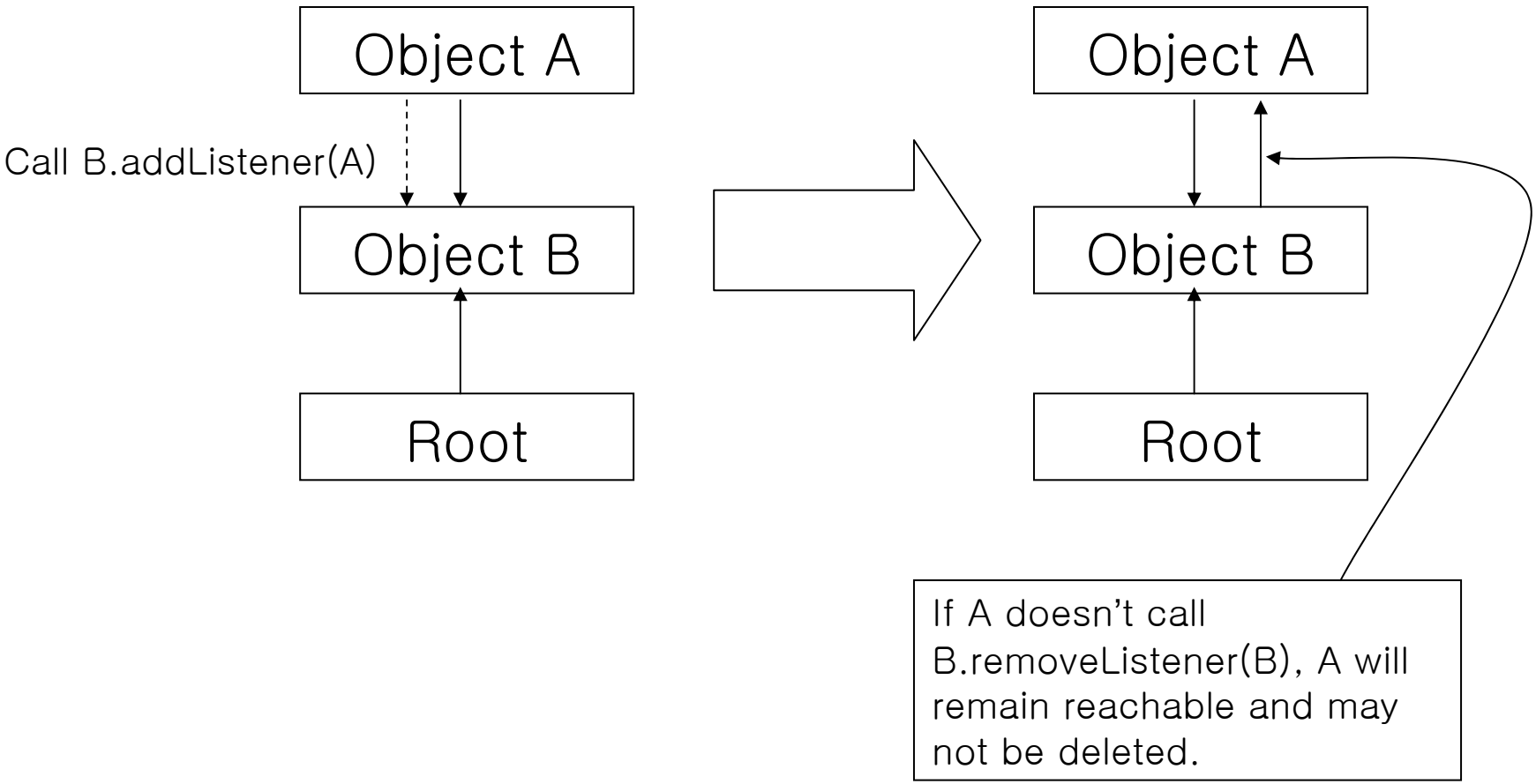
Resource Management

- Java Memory Leak
- Java Heap and Native Heap
- Thread Kill
- Application Context
- Resource Manager

Java Memory Leak (1)

- Many Java programmers believe
  - Not need to pay attention to memory leak in Java because the garbage collector handles it.
    - It's completely wrong.
- JVM thinks the reachable Java objects are not garbage
  - Therefore, to delete unused Java objects, they must not be reachable.

Java Memory Leak (2)



Java Heap and Native Heap (1)

- Java Heap
  - Java “new” allocates Java object in the Java heap.
  - Divided into handle pool and data pool for the compaction.
  - Java objects are deleted by GC.

Java Heap and Native Heap (2)

- Native Heap
  - Native “malloc” allocates memory in the native heap.
  - “free” should be called for deleting the allocated memory.

Java Heap and Native Heap (3)

- In AltiCaptor
  - If only small free native heap is available, call GC.
    - GC can increase the free native heap.
  - It is also possible to use Java heap directly from native.
    - By using internal private API.
    - The great care must be taken because Java heap data can be moved by the compaction.

Thread Kill (1)

- How to kill a thread by force?
  - According to JVM specification, there is no way to kill the thread by force.
- Make the thread do nothing by
  - Setting the priority to the lowest.
  - Removing all permission of the thread.
  - But still alive and consumes memory.

Thread Kill (2)

- Thread.interrupt()?
  - If the thread is waiting in the “monitorenter” bytecode, can’t be interrupted.
- Thread.stop()?
  - Deprecated.

Thread Kill (3)

- Modify JVM
  - Create special “kill” method to kill thread.
  - The “kill” method throws the special Exception to the target thread.
  - The JVM handles the special Exception.
    - Awake from “monitorenter”.
  - Throws the Exception periodically until the thread die.

Application Context

- How to translate current directory “.” into absolute directory?
  - 1) Load application classes with the special class loader.
    - Different application is loaded by different class loader.
  - 2) Dump the java stack and search for the special class loader.
  - 3) The class loader will have the application context and have the current directory information.

Resource Manager (1)

- How to kill the application by force without resource leak?
  - 1) When the application calls “addXXXListener” system class method, the system class method registers to the resource manager.
  - 2) Exit all monitor which was entered by the application.

Resource Manager (2)

- 3) Kill all threads blocked in the application.
- 4) The resource manager calls “removeXXXListener” registered by the application.

References (1)

- Alticast
  - http://www.alticast.com
  - http://alticast.com/downloads/Solution%20Overview.pdf
- Sun Microsystems
  - http://www.sun.com
  - http://java.sun.com/j2me/j2me-ds.pdf
  - http://java.sun.com/j2se/1.4.2/docs/guide/jpda
  - http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi

References (2)

- ej-technologies
  - http://www.ej-technologies.com
  - http://www.ej-technologies.com/products/jprofiler/overview.html

References (3)

- Embedded System FAQ by Taeho Oh
  - http://ohhara.sarang.net/history/info/embedded_system_faq.txt