OVM
lessons learned

Jan Vitek

Jason Baker, Antonio Cunei, Chapman Flack, Filip Pizlo, Marek Prochazka,
Krista Grothoff, Christian Grothoff, Andrey Madan, Gergana Markova, Jeremy Manson,
Krzystof Palacz, Jacques Thomas, Hiroshi Yamauchi

Purdue University

David Holmes

DLTeCH

DARPA Program Composition for Embedded Systems (PCES)

NSF/HDCP - Assured Software Composition for Real-Time Systems

PURDUE
UNIVERSITY

(ς^3)

Darpa's Goal: Fly Boeing's UAV

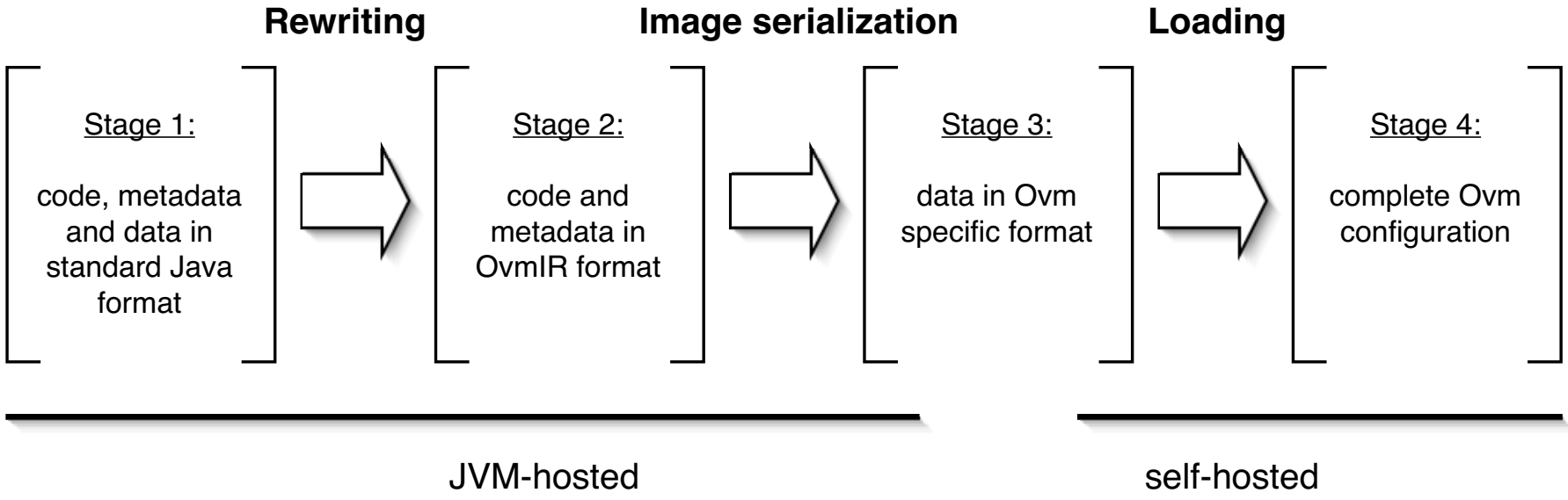
- Our mission:
  implement a Real-time Specification for Java compliant VM
- Only other RTSJVM was an interpreter & proprietary
- Target is avionics software for the Boeing/Insitu ScanEagle UAV

A Configurable Open VM

- A clean-room implementation
- *Internal project goal:*
  open source framework for language runtime systems
- A Java-in-Java VM
- 150KLoc of Java, 15Kloc of C code
- GNU classpath libraries + our own RTSJ implementation

Build Process

- Bootstrapped under Hotspot
- Configuration and partial evaluation
- Generate an executable image (data+code)
- IR-spec + interpreter generation



Lessons

- JavaInJava
  - anecdotal evidence of lower bug rates
  - same optimizing compiler for VM & user code
  - fewer cross-language calls

```
public Oop updateReference(MovingGC oop) {
  int sz = oop.getBluepqrint().getVariableSize(oop);
  if (sz >= blockSize) {
      movedBytes += sz;
      VM_Word off = VM_Address.fromObject(oop).diff(heapBase);
      int idx = off.asInt() >>> blockShift;
      block.pin(idx);
      return oop;
  } else {
      VM_Address newLoc = getHeapMem(sz, false);
      Mem.the().cpy(newLoc, VM_Address.fromObject(oop), sz);
      oop.markAsForwarded(newLoc);
      return newLoc.asOop();
  }
```

Ovm Configurations

execution engine	aot / jit / interp	
static analysis	off / CHA / RTA	
fast locks	on / off	
memcopy	fast / bounded-latency	
I/O system	SIGIOSocketsPollingOther (Profiling) / SIGIOSockectsStallingFilesPolling	SelectSocketsPollingOther SelectSocketsStallingFiles-PollingOther
threading	java / realtime / profiling	pip / time preemptive
transactions	on / off/ profile	
object/mem models	AllCopy:B-M-F-H	MostlyCopyWB:B-M-F-H
	MostlyCopySplitRegions:B-Mf-F-H	JMTk:B-M-J-H
	MostlyCopyWB:B-Mf-F-H	MostlyCopy:B-M-F-H
	MostlyCopyRegions:B-M-F-H	SimpleSemiSpace:B-M-F-H
	MostlyCopyingRegions-B_Mf_F_H	minimalMM-B_0M
	MostlyCopyingSC-B_M_F_H	minimalMM-B_M
	minimalMM-B_M_J_H	

Lessons

- Configuration mechanisms
  - ☑ Interfaces and inheritance are not sufficient (we have 3371 classes and ~450 interfaces)
  - ☑ AOP should be revisited
  - ☑ Component systems such as Jiazzi, Scala...
  - ☑ We rolled our own...

Lessons

- Configuration mechanisms, example transactions:
  - Implementing a form of transactional memory in Ovm takes about ~1200 lines code.
  - Changes to the sources of the VM, ~40 lines in 34 different places, e.g.:

```
void runThread(OVMThread t) throws PragmaNoPollcheck{
  boolean aborting =
   Transaction.the().preRunThreadHook(thisThread, t);
  setCurrentThread(t);
  Processor.getCurrentProcessor().run(t.getContext());
  ...
  Transaction.the().postRunThreadHook(aborting);
```

Lessons

```
boolean aborting =
 Transaction.the().preRunThreadHook(thisThread, t);
```

- Generated C code

```
jboolean _stack_2 =
 S3Transaction_preRunThreadHook(e.roots->vals[97]),
                                _stack_0, _stack_1);
```

- Stitcher specification

```
# Select an implementation of the transactional API described in the
# Preemptible Atomic Region paper. EmptyTransaction gives the
# default behavior. S3Transaction is the real thing.
s3.services.transactions.Transaction \
                s3.services.transactions.S3Transaction
```

Lessons

- Domains
  - Separation is necessary
  - one Executive and possibly multiple User domains
  - Each domain can have it's memory manager, scheduler, class libraries, and even object model
  - opaque types
  - cross domain accesses are reflective
  - enforced by the type system -- requires Object *not* to be builtin
  - special handling of exceptions crossing boundaries

Lessons

- GCC as a backend
  - ☑ offload low-level optimizations
  - ☑ cross-platform portability
  - ☑ using C++ exceptions is suboptimal
  - ☑ inlining can lead to bloat and long compile times
  - ☑ No precise GC ... but working on it.

Lessons

- Cooperative Scheduling
  - ☑ OS-independent
  - ☑ Priority inversion avoidance (PIP/PCE) supported in a portable fashion and optimized by the compiler
  - ☑ but, we had to implement our own non-blocking I/O

```
# 291 "./s3/util/queues/Queue.java"
static jboolean Queue_isEmpty(queues_Queue * ovm_this){
        _pc0:;
        if (CHECK_EVENTS())  signalEvent();
```