OSSUMMIT

Design a Product-Ready JVM for Apache Harmony

Xiao-Feng Li, Pavel Ozhdikhin

Contributors: Mikhail Loenko, Vladimir Beliaev

OSSUMMIT

Agenda

- Harmony and DRLVM modularity
- Garbage Collectors in DRLVM
- JIT Compilers in DRLVM
- Engineering infrastructure

OSSUMMIT

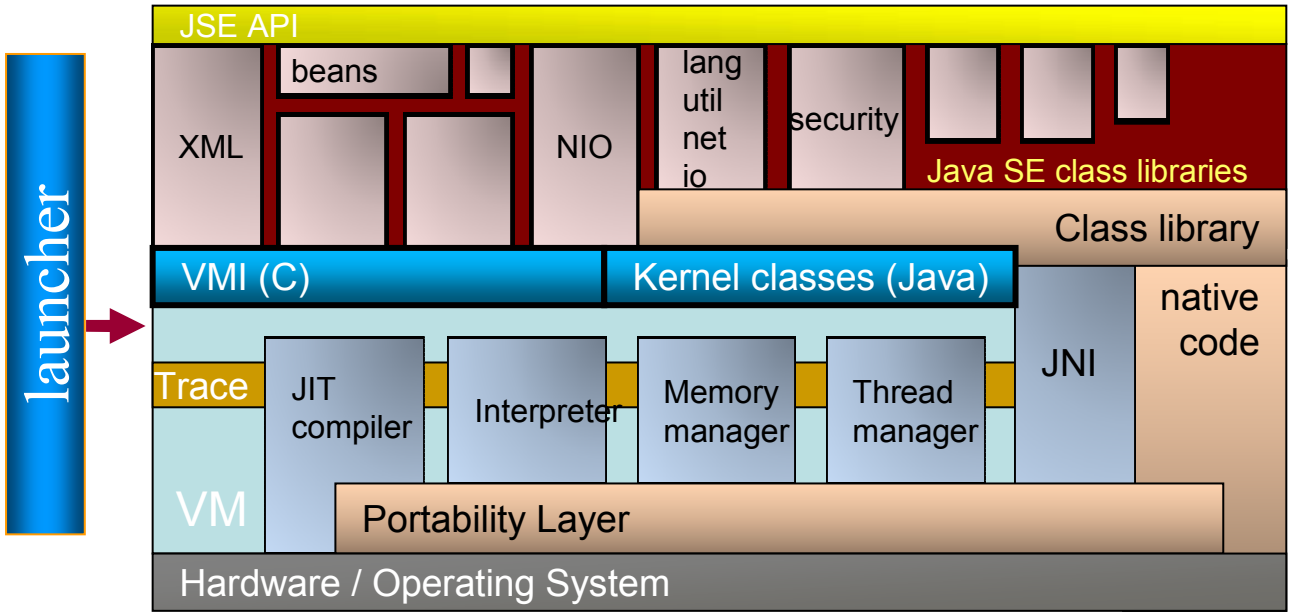
Apache Harmony

- Primary goal – full implementation of Java SE
  - Compatible class library
  - Competitive virtual machine
  - Full JDK toolset
- Founded in Apache Incubator, May 2005
- Became Apache Harmony Project, Oct 2006
- Facts today
  - 27 committers at the moment, 30 commits weekly
  - 250 messages weekly in mailing list
  - 150 downloads weekly

OSSUMMIT

Design Modularity

- Pluggability: easier for developers, researchers, and testers



OSSUMMIT

Harmony Status

- ~2.3 million LOC (Java 1.6m, C/C++ 0.7m)
- Components
  - API: 98% JDK5, 90% JDK6
  - VMs: JCHEVM, BootJVM, SableVM, DRLVM, evaluation version of BEA JRockit binary
  - Tools: javac, javah, jarsigner, keytool
- Platforms
  - Windows/Linux, 32bit/64bit
- Serious testing and engineering infrastructure

OSSUMMIT

Harmony DRLVM

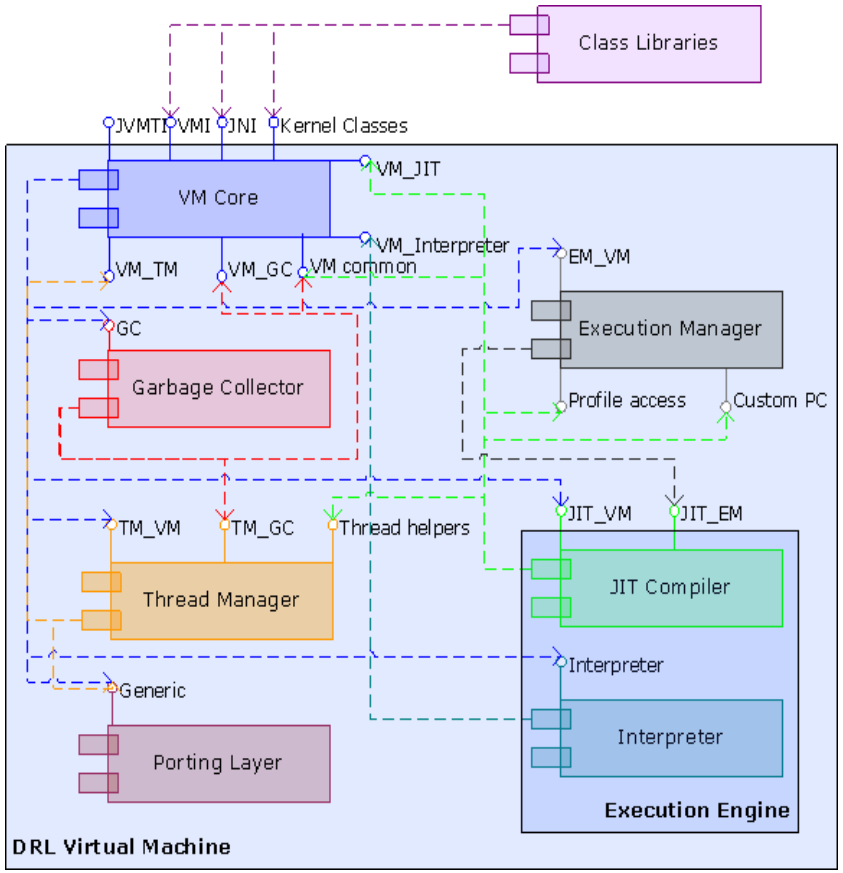
- The current default VM of Harmony
- Components
  - Two JIT compilers: fast and optimizing
  - Three GCs: simple/parallel/concurrent
  - Other features: optimized threading, JVMTI, class unloading, interpreter, etc.
- Targets
  - Robustness, performance, and flexibility
  - Server and desktop
  - Product-ready

OSSUMMIT

DRLVM Modularity Principles

- *Modularity:* Functionality is grouped into a limited number of coarse-grained modules with well defined interfaces.
- *Pluggability:* Module implementations can be replaced at compile time or run time. Multiple implementations of a given module are possible.
- *Consistency:* Interfaces are consistent across platforms.
- *Performance:* Interfaces fully enable implementation of modules optimized for specific target platforms.

DRLVM Modules



- Other modules
  - Class loader
  - Class verifier
- Advantages
  - Easy for developing and testing
  - Easy to use third-party components
  - Portability

OSSUMMIT

Modularity with Performance

- Cross-module (shared lib) procedural call has extra penalties
  - Indirect call overhead, and cannot be inlined
- “Runtime helpers” to solve the problem
  - Frequently accessed procedures written in Java
  - JIT inlines the helpers
  - Helpers use “magics” for pointer arithmetics and compiler intrinsics

Helper: Bump-Pointer Allocation

```
@Inline
public static Address alloc(int objSize, int allocationHandle)
{
    Address TLS_BASE = VMHelper.getTlsBaseAddress();

    Address allocator_addr = TLS_BASE.plus(TLS_GC_OFFSET);
    Address allocator = allocator_addr.loadAddress();
    Address free_addr = allocator.plus(0);
    Address free = free_addr.loadAddress();
    Address ceiling = allocator.plus(4).loadAddress();

    Address new_free = free.plus(objSize);
    if (new_free.LE(ceiling)) {
        free_addr.store(new_free);
        free.store(allocationHandle);
        return free;
    }
    return VMHelper.newResolved (objSize, allocationHandle);
}
```

OSSUMMIT

Agenda

- Harmony and DRLVM modularity
- Garbage Collectors in DRLVM
- JIT Compilers in DRLVM
- Engineering infrastructure

OSSUMMIT

DRLVM GC Design Goals

- Product-quality GC in robustness, performance, and flexibility
  - Robustness: modularity and code quality
  - Performance: scalability and throughput
  - Flexibility: configurability and extensibility

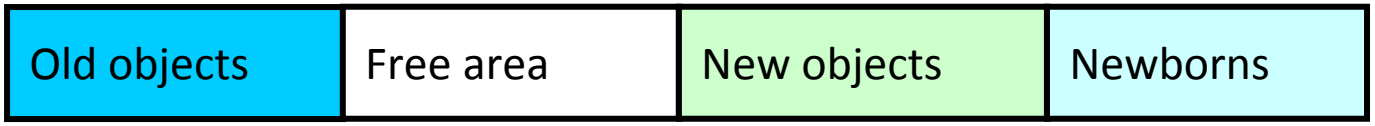
OSSUMMIT

DRLVM GC Current Status

- GCv4.1
  - Copying collector with compaction fallback
  - Simple, sequential, non-generational
- GCv5
  - Copying collector with compaction fallback
  - Parallel, generational (optional)
- Tick
  - On-the-fly mark-sweep-compact
  - Concurrent, parallel, generational (optional)

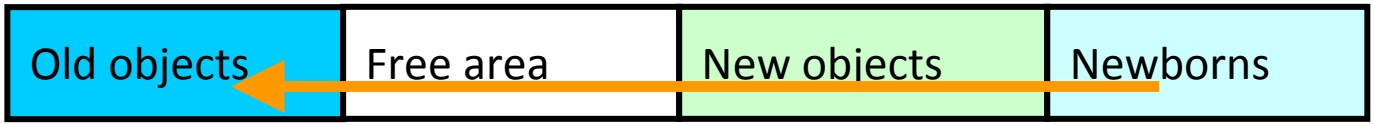
OSSUMMIT

GCv4.1: Heap Layout

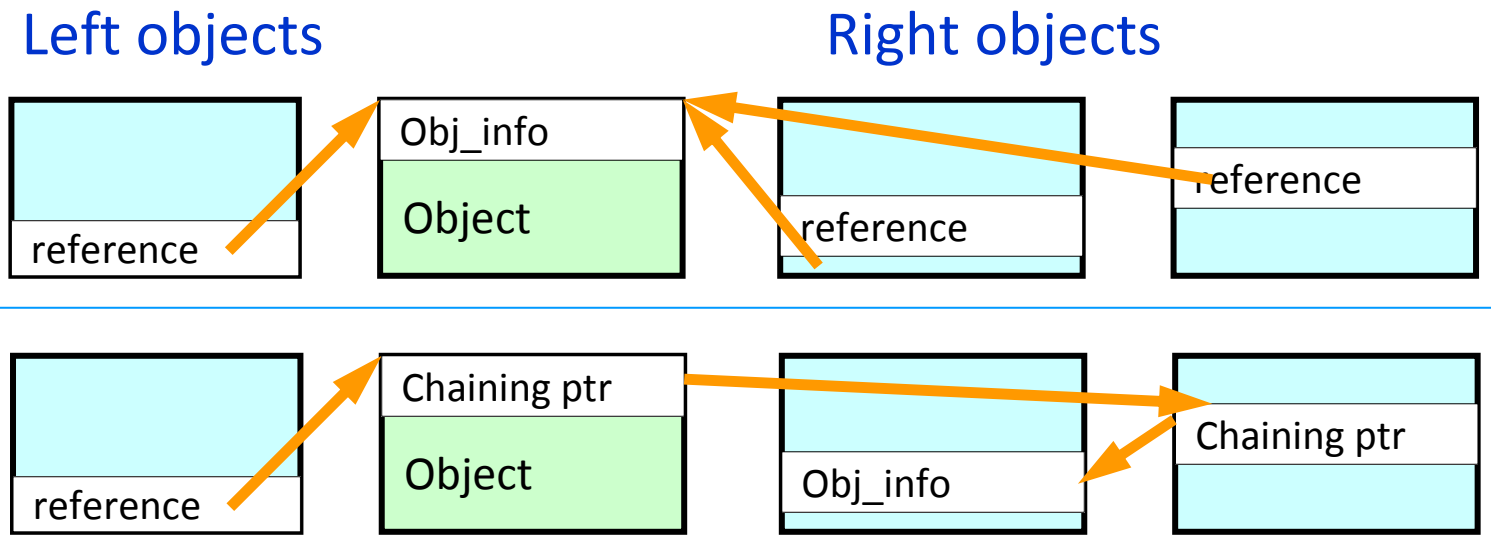


- During a collection, new objects are copied to free area

- If free area is inadequate, GC changes to compaction algorithm



GCv4.1: Ref-Chain Compactor

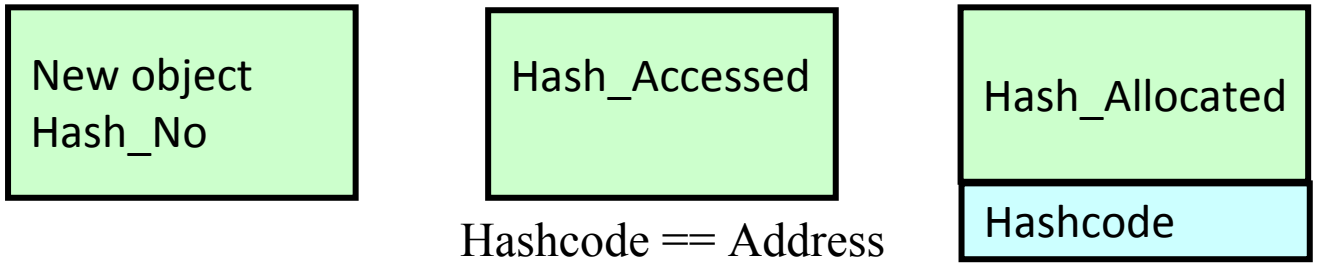


- Chain the references to an object, so that they can be updated once the object's new location is known. No extra space needed, two heap passes

OSSUMMIT

GCv4.1: Hashcode

- 3 hashcode states
  - Hash_No, Hash_Accessed, Hash_Allocated



- Assumptions
  - Most objects not accessed for hashcode
  - Most objects accessed for hashcode before they survive once collection (moved)

OSSUMMIT

GCv4.1: Characteristics

- Good performance
- Simple, easy to learn

- But the algorithm is not parallel
  - Cannot leverage multiple cores
- Has no generational support
- We developed GCv5

OSSUMMIT

GCv5: Heap Layout

- Minor collection: young objects are copied to free area, large object space are mark-swept

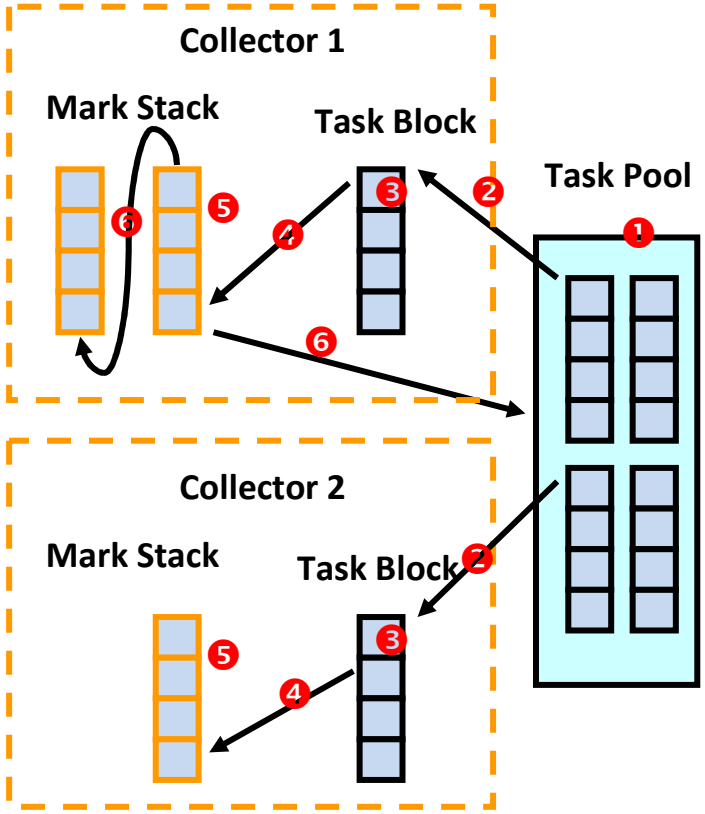
- Major collection: compact non-LOS and LOS

OSSUMMIT

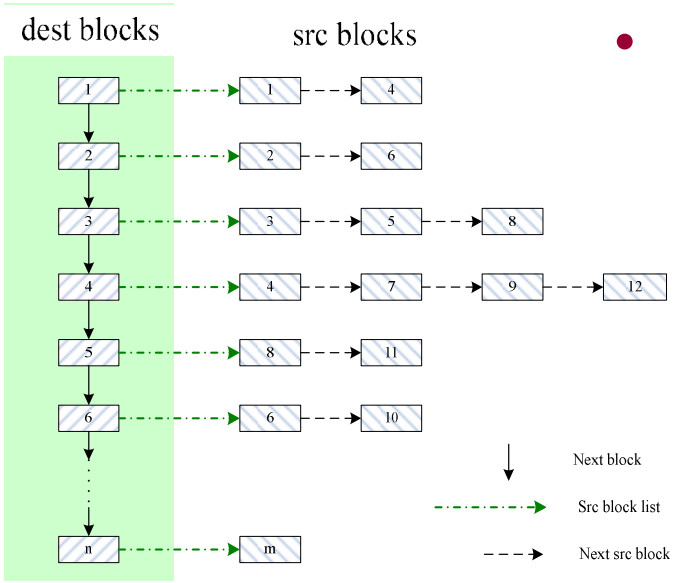
Parallel Marking& Copying

- Parallel pool-sharing marking:

1. Shared Pool for task sharing
2. Collector grabs task block from pool
3. One reference is a task
4. Pop one task from task block, push into mark stack
5. Scan object in mark stack in DFS order
6. If stack is full, grow into another mark stack, put the full one into pool
7. If stack is empty, take another task from task block

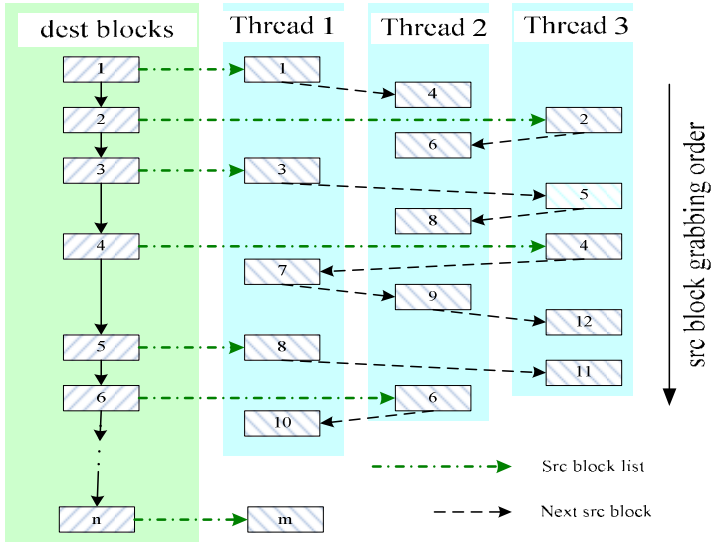


GCv5: Parallel Compactor



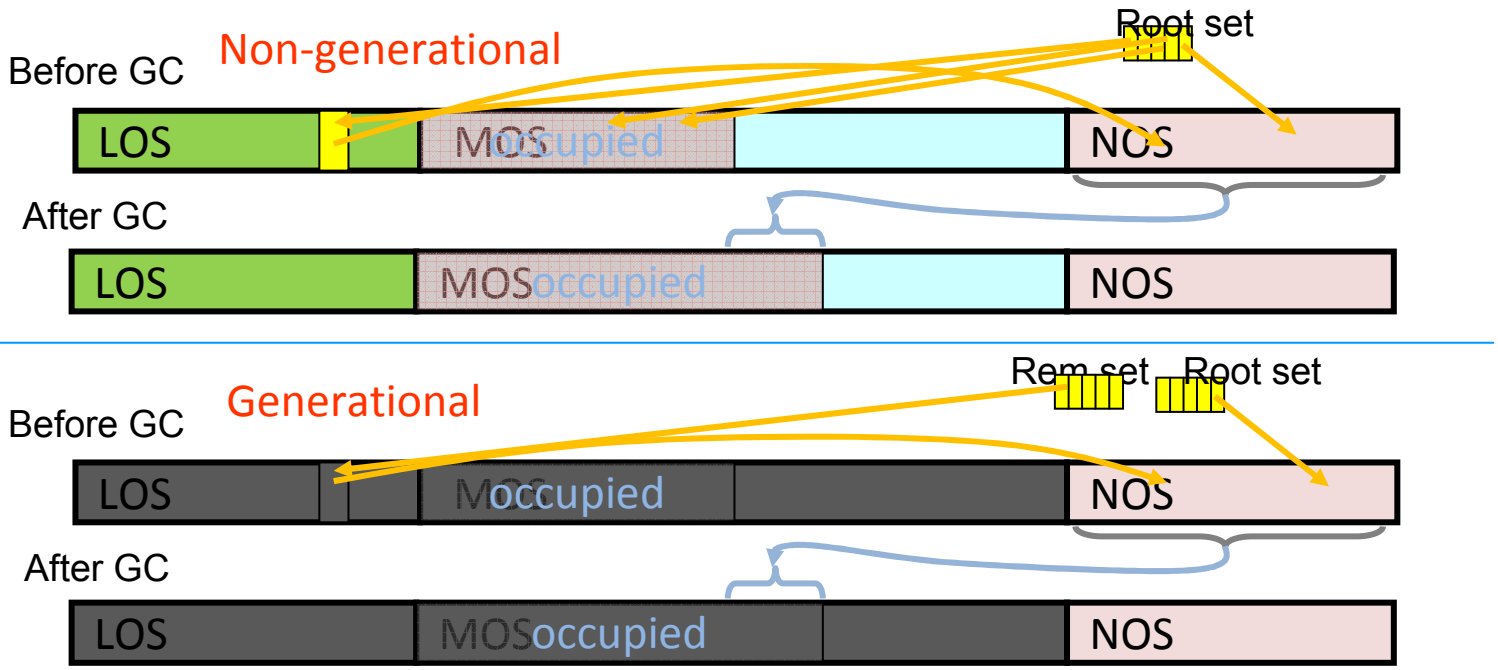
1. Maintain a src-block list for each dest block
2. Grab a src-block from the list atomically in parallel for compaction

- Two parallel compaction algorithms
  - Here shows Parallel LISP2 Compactor



OSSUMMIT

GCv5: Generational Collection



- Assumptions
  - Most objects die young
  - Remember set do not keep lots of floating garbage

OSSUMMIT

GCv5: Runtime Adaptations

- Runtime adaptation for maximal GC throughput
  - Select major or minor collections
  - Adjust the boundaries between spaces
  - Switch between generational and non-generational collections (off by default)
  - Tune finalization speed

OSSUMMIT

GCv5: Characteristics

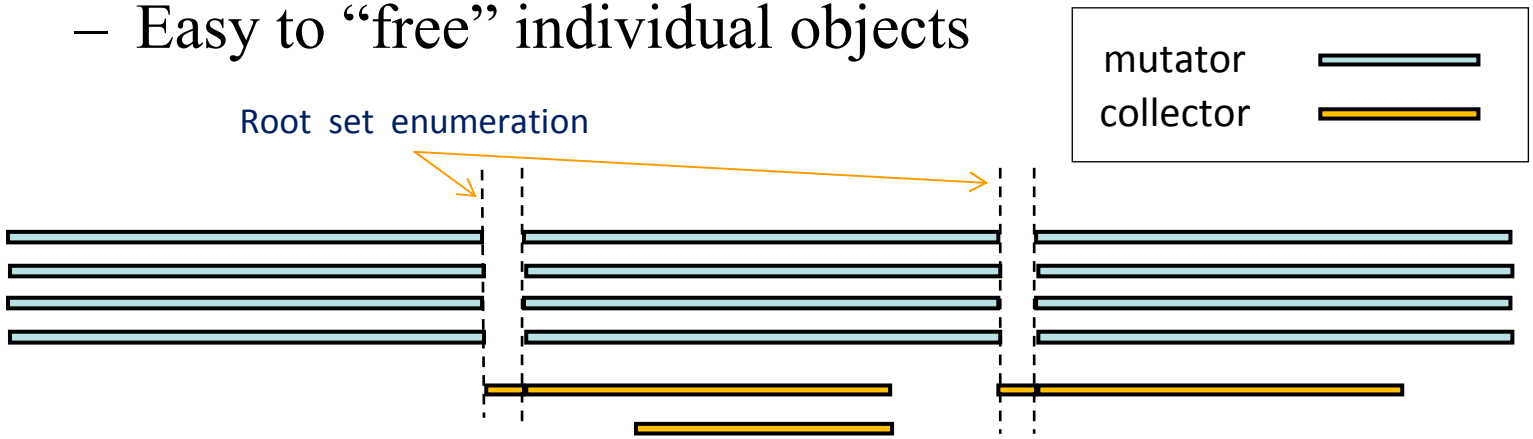
- Good performance
- Scalable on multiple cores
- Runtime adaptations

- But pause time in major collection is high
- Not support conservative collection
- We developed Tick

OSSUMMIT

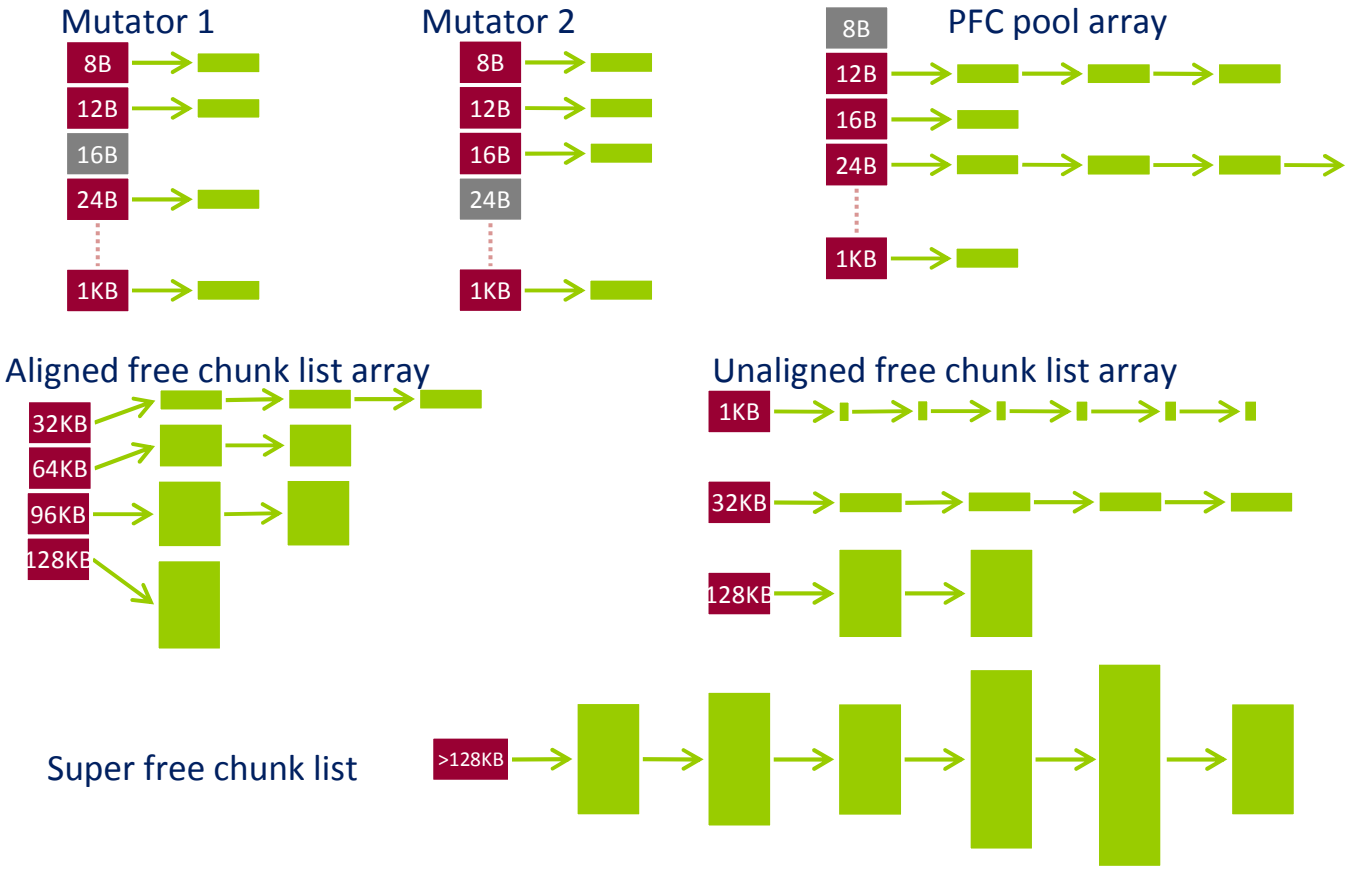
Tick: Overview

- Tick is on-the-fly mark-sweep GC
  - Non-moving, conservative GC possible
  - Easy to “free” individual objects



- Heap layout
  - Partition Sweep-Space into chunks
  - Each chunk only holds objects of certain size

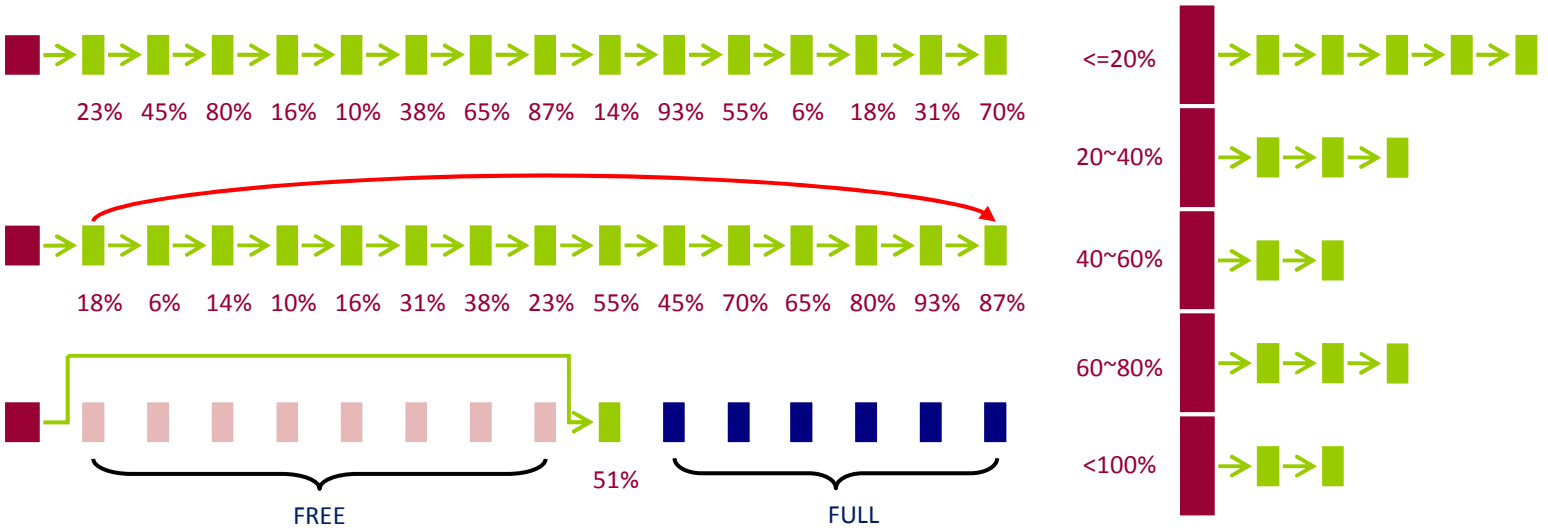
Tick: Heap Layout



OSSUMMIT

Tick: Compactor

- Mark-sweep has fragmentation problem
- Compact heap optionally when appropriate
  - Fast compaction algorithm



OSSUMMIT

Tick: Characteristics

- Short collection pause time
  - Target is at ms level
  - Tradeoff with GC throughput
- Parallel and adaptive collection
- Working models
  - Concurrent or stop-the-world
  - Standalone or in generational GC

OSSUMMIT

GC Modularity

- External modularity
  - Different GC implementations can be specified in command line
  - Achieved by implementing the required GC interfaces in include/open/gc.h
- Internal modularity
  - GCv5 and Tick are implemented on the same code base
  - Achieved by coordinating different space collection algorithms
  - Shared GC verbose, GC verifier, etc.

OSSUMMIT

Agenda

- Harmony and DRLVM modularity
- Garbage Collectors in DRLVM
- JIT Compilers in DRLVM
- Engineering infrastructure

OSSUMMIT

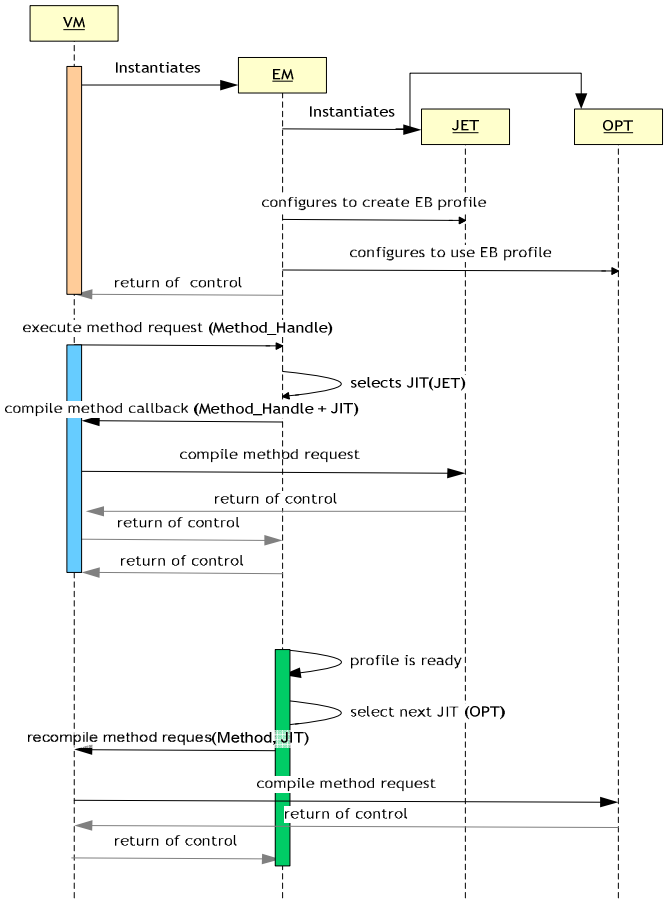
DRLVM execution engines

- The Execution Manager
- Jitrino compilers:
  - Jitrino.JET
  - Jitrino.OPT
    - Optimizations
    - Pipeline Management Framework
    - Internal profiler

OSSUMIT

The Execution Manager

- Keeps a registry for all execution engines and profile collectors available at run time
- Selects an execution engine to compile a method by a VM request according to the configuration file
- Coordinates profile collection and use between various execution engines
- Supports asynchronous recompilation in a separate thread to utilize multi-core



OSSUMMIT

Dynamic profilers

- **EB_PROFILER**
  Entry/backedge profile. Collects 2 values for each method:
  - number of times a method has been called (entry counter)
  - number of loop interactions (backedge counter) performed in a method

- **EDGE_PROFILER**
  Edge profile. Collects 2 types of values for each method:
  - Number of times a method has been called
  - Number of times every branch in a method has been taken

- **VALUE_PROFILER**
  Value profile. Collects up to N the most frequent values for each registered profiling site in a method. Uses advanced Top-N-Value algorithm.

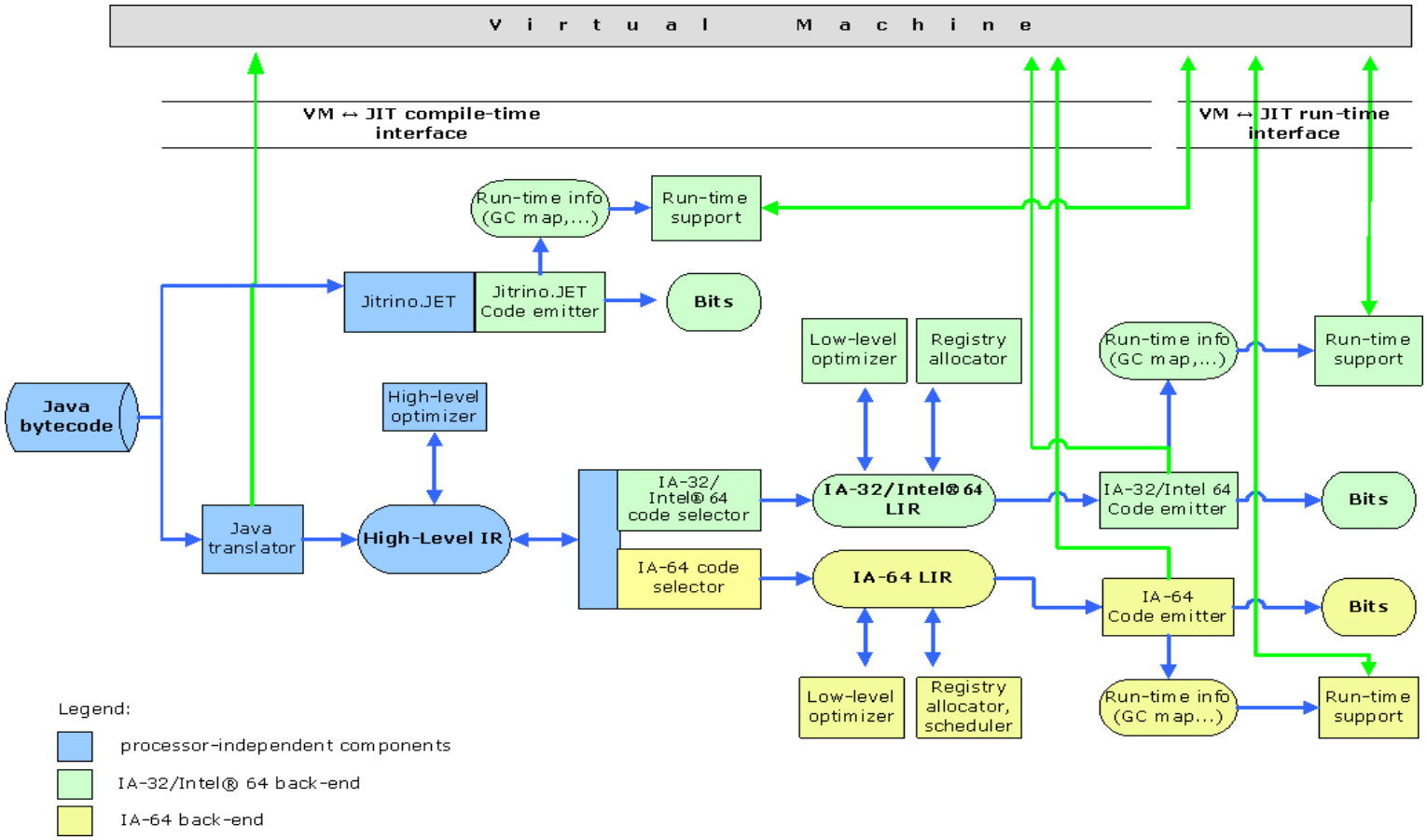
OSSUMMIT

Jitrino compilers

- Jitrino.JET
  - baseline compiler for IA32/Intel64 platforms
- Jitrino.OPT
  - Optimizing compiler for IA32/Intel64/IPF platforms

OSSUMIT

Jitrino Architecture



OSSUMMIT

Jitrino.JET – baseline compiler

- Simple: no internal representation, just 2 passes over bytecode
- Small: ~500K code, ~14K NSLOCs
- Fast: Compilation speed ~ 10-20K methods per second (1.5Ghz laptop)
- Supports JVMTI, VMMagic and can easily be modified to support new features
- Produces more then 10 times faster code than the interpreter (and ~2 times slower than the code made by Jitrino.OPT)

Jitrino.JET: log sample

Java method:

```
public static int max(int x, int y) {
  return x > y ? x : y;
}
```

Prologue: <store all callee-save registers in use>

```
;;   0) ILOAD_0
;;   1) ILOAD_1
;;   2) IF_ICMPLE        ->9<-
0x03EB00B6  cmp ebx, esi
0x03EB00B8  jle dword 0x11
;;   5) ILOAD_0
;;   6) GOTO             ->10<-
0x03EB00BE  mov [ebp+0xffffff14], ebx
0x03EB00C4  jmp 0xb
;;   9) ILOAD_1
0x03EB00C9  mov [ebp+0xffffff14], esi
;;  10) IRETURN
0x03EB00CF  mov eax, [ebp+0xffffff14]
```

Epilogue: <restore all callee-save register in use>

OSSUMMIT

Jitrino.OPT – optimizing compiler

- The fast, aggressively optimizing compiler
- Pluggability facilitated by the Pipeline Management Framework
- Features:
  - High- and low-level intermediate representations
    - Most optimizations run at the platform-independent high level
  - Supports edge and value profiles
  - A flexible logging system enables tracing of major Jitrino activities, including detailed IR dumps during compilation

OSSUMMIT

Jitrino.OPT optimizations

- Guarded devirtualization
- Global Code Motion
- Escape Analysis based optimizations:
  - Synchronization elimination
  - Scalar replacement
- Array initialization/copying optimizations
- Array bounds check elimination

…and many other most known optimizations

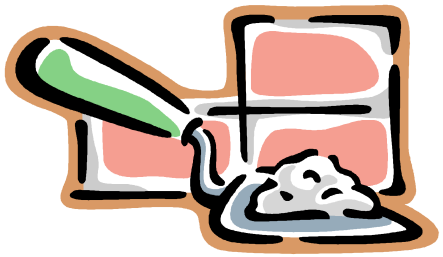
OSSUMMIT

Advanced optimizations

- VM Magics and helper inlining
  - Allow developers to write performance critical code in Java using address arithmetic and low-level compiler intrinsics.
- Value profile guided devirtualization
  - Effectively de-virtualize not only virtual but also interface and abstract calls
- Lazy exceptions
  - Create exception objects on demand, i.e. only if it's actually used in the exception handler

OSSUMMIT

Pipeline Management Framework



PMF - the JIT pluggability vehicle

- PMF features:
  - Standard interface for the pipeline steps (IR transformers)
  - Nested pipelines
  - Full control over the pipeline steps and their options through the Java property mechanism
  - Rich control over the logging based on JIT instances, pipelines, class and method filters
- PMF details:
  - http://harmony.apache.org/subcomponents/drlvm/JIT_PMF.html

Jitrino.OPT internal profiler

The internal profiler (iprof) in the Jitrino.OPT compiler can instrument the code so that per-method counters of the instructions executed at run time will be dumped.

- To use iprof you need to create the iprof.cfg configuration file with the profiler's configuration and specify the following option:
  -XX:jit.arg.codegen.iprof=on

- An example of the iprof output:

Method name	Insts	ByteCodeSize	MaxBBExec	HottestBBNum	…
java/lang/Thread.<clinit>	7	13	1	2	…
java/lang/Object.<init>	6445	1	6445	2	…
java/lang/Thread.<init>	2440	257	24	0	…
…	…	…	…	…	…

OSSUMMIT

JIT Resources

- Execution Manager:
  http://harmony.apache.org/subcomponents/drlvm/EM.html
- Jitrino JIT Compiler:
  http://harmony.apache.org/subcomponents/drlvm/JIT.html
- Pipeline Management Framework and Jitrino logging system:
  http://harmony.apache.org/subcomponents/drlvm/JIT_PMF.html
- Jitrino.OPT internal profiler:
  http://harmony.apache.org/subcomponents/drlvm/internal_profiler.html
- Harmony performance reports:
  http://harmony.apache.org/performance.html

OSSUMMIT

Agenda

- Harmony and DRLVM modularity
- Garbage Collectors in DRLVM
- JIT Compilers in DRLVM
- Engineering infrastructure

OSSUMMIT

Engineering Infrastructure

- Tasks
  - Maintain code integrity
  - Build and publish snapshots
  - Pre-release testing
  - Maintain and benefit from modular architecture
  - Obtain and publish test results
  - Define Milestone schedule/criteria

OSSUMMIT

Harmony Quality Base

- Classlib test suite: 23K tests
- DRLVM test suites: 10K tests
- Stress, reliability test suites: ~300 tests
- Unit tests for 3rd party apps
  - Eclipse: 40K tests, Geronimo: 600 tests, etc.
- Application automated test scenarios
  - App servers, client and GUI apps

- Open Challenging Problems:

OSSUMMIT

Quality Engineering

- Pre-commit testing
  - Committer tests for 0.5 ~ 2 hours
- Code integrity testing
  - Activates after any commit, fully automated
    - Challenge: how to minimize traffic pressure to Apache infrastructure
- Snapshot testing
  - Up to 48 hours testing cycle, more tests
  - Build snapshots for developers
- Pre-milestone testing
  - Produce “stable builds”, go through all tests

OSSUMMIT

What we offer

- For Harmony developers:
  - Up-to-date Code Base Health Indicators
  - Snapshots accompanied by testing results
    - Windows/Linux on IA32/Intel64
- For Harmony Users:
  - More stable snapshots, and
  - Milestone “Stable builds” every 2~3 months,
  - No official Harmony releases available yet
- For Harmony Testers:
  - Build and Test Infrastructure 2.0 – easy add any test suite to the testing cycle using test suite adaptors

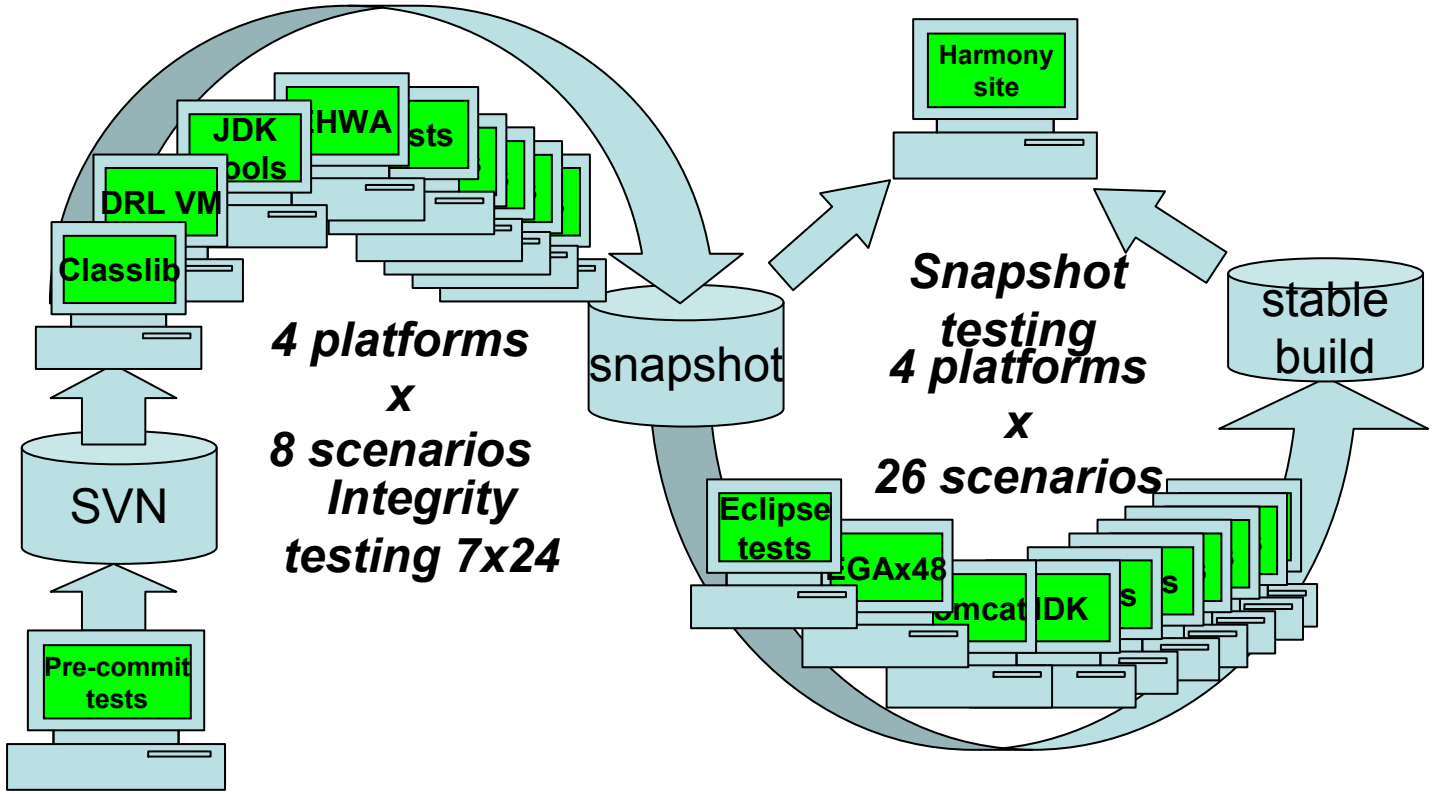
Harmony Code Integrity status on Fri Sep 28 05:20:03 2007 GMT

Default mode debug build

	Windows x86 32 bit	Linux x86 32 bit	Windows x86 64 bit	Linux x86 64 bit	Linux IA64
Classlib build	2007-09-25	2007-09-25	2007-09-25	2007-09-25	2007-09-25
DRLVM build	2007-09-26	2007-09-26	2007-09-26	2007-09-26	2007-09-26
Federated HDK build	2007-09-26	2007-09-26	2007-09-26	2007-09-26	N/A
Classlib Swing/AWT tests	2007-09-26	2007-08-31 error	2007-09-26 error	2007-08-26 error	N/A
Classlib tests	2007-09-26 since 2007-09-25	2007-09-26 error	2007-09-26 since 2007-09-26	2007-09-26	N/A
DRLVM regression tests	2007-09-26	2007-09-26	2007-09-26	2007-09-26	N/A
DRLVM tests	2007-09-26 since 2007-09-25	2007-09-26 since 2007-09-25	2007-09-26	2007-09-26	2007-09-26 error
Eclipse Hello World Application	2007-09-26	2007-09-26	N/A	2007-09-26	N/A
JDKTools tests	2007-09-26	2007-09-26 since 2007-09-24	2007-09-26	2007-09-26	N/A

OSSUMMIT

Harmony Testing Process



OSSUMIT

Research Exploiting Modularity

- Moxie JVM project (http://moxie.sourceforge.net/)
  - A JVM written in Java uses Jitrino.OPT
- Java STM (Software Transactional Memory) in DRLVM
  - JIT recognizes atomic constructs and injects calls to the STM library into the code
- Dataflow Java based on DRLVM
  - To introduce CSP (communicating sequential process) into Java programming

OSSUMMIT

Workloads Example: EIOffice

- Evermore EIOffice (http://www.evermoresw.com)
  - Large scale office suite written in pure Java
  - Extensively uses Swing/AWT
- EIOffice with Harmony (http://eio-harmony.sf.net)
  - First release: v0.02

Demo

OSSUMMIT

Summary

- We develop DRLVM as a product-ready JVM for Apache Harmony
- DRLVM benefits from its modularity design in
  - Developing, researching, and testing
- DRLVM has sophisticated GC and JIT implementations
  - Still under heavy development to enhance
  - Real time GC, huge heap support, etc.

OSSUMMIT

Thanks!
&
Questions?