Squawk Java Virtual Machine Compiler Architecture

David Liu

Manager: Cristina Cifuentes

Presentation Outline

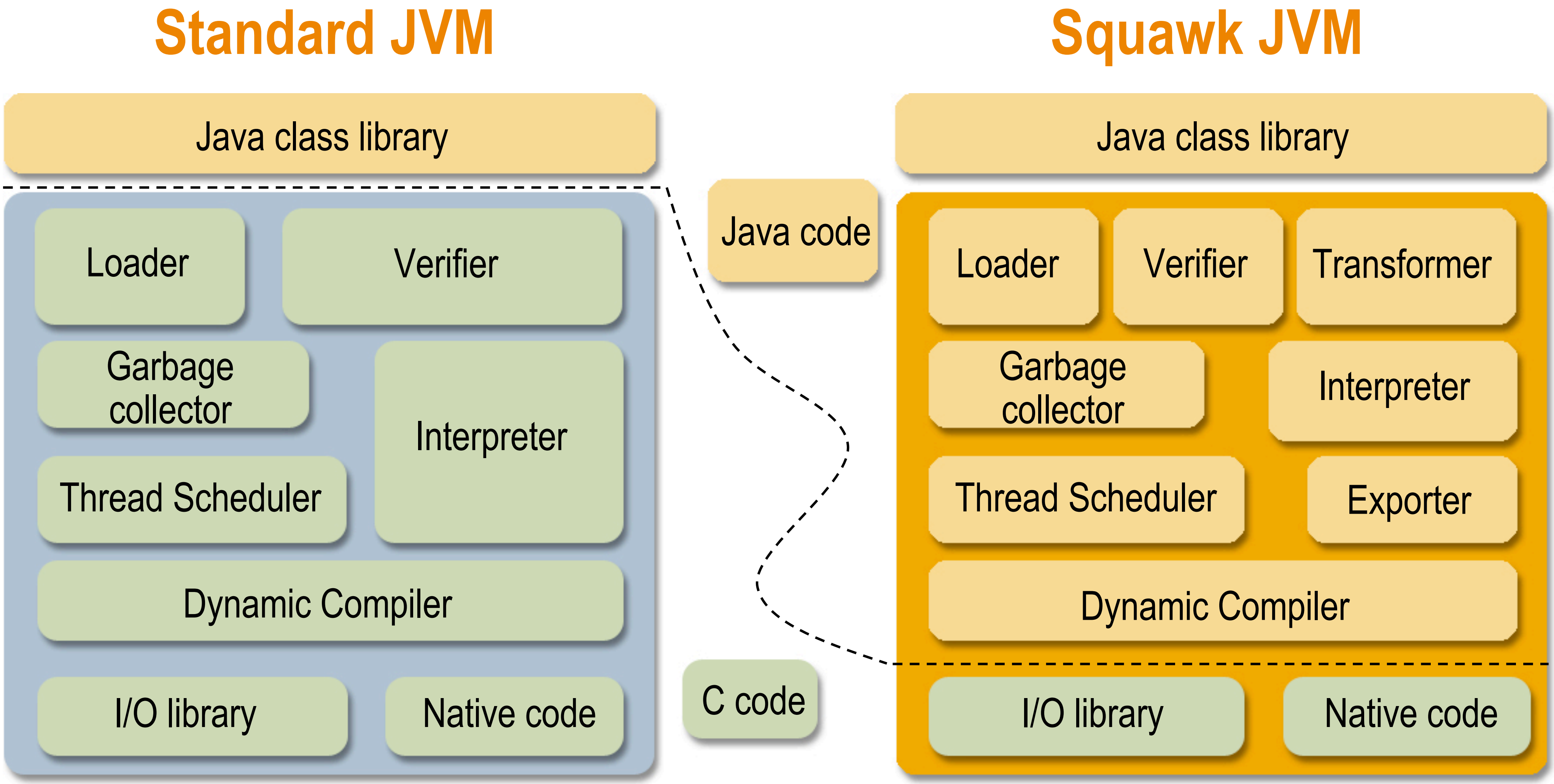
- What is Squawk?
- Standard JVM vs Squawk JVM
- Squawk architecture
- Compiler interface
- Multiple compilers - simple vs optimizing
- Execution of Squawk code
- Future work

What is Squawk?

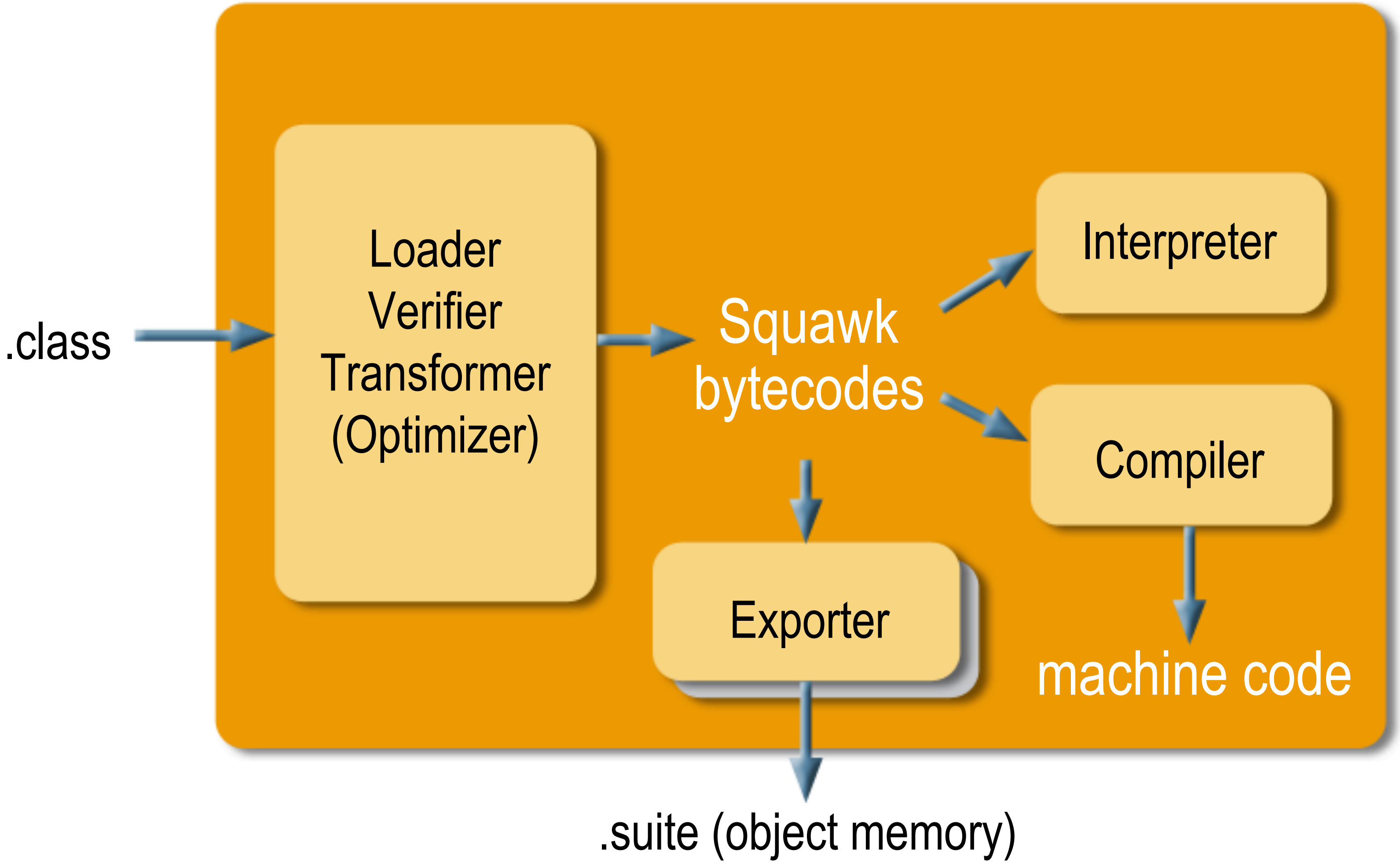
- Small, execute-in-place JVM
- Written in Java, not C
- Optimized for small devices
- Squawk architecture has an interpreter, compiler and jitter
- Current use: Sun Spot sensors platform
- Why write a JVM in Java?
  - Portability

- What performance can the Squawk JVM achieve?

Standard JVM vs Squawk JVM



The Squawk Architecture



Compiler Interface

- Stack-based, machine-independent instruction interface
- Allows multiple compiler implementations for Squawk
- Two types of compilers will be used by Squawk - simple and optimizing (collaborative research)

Squawk bytecode	Bytecode deconstructor	Compiler interface

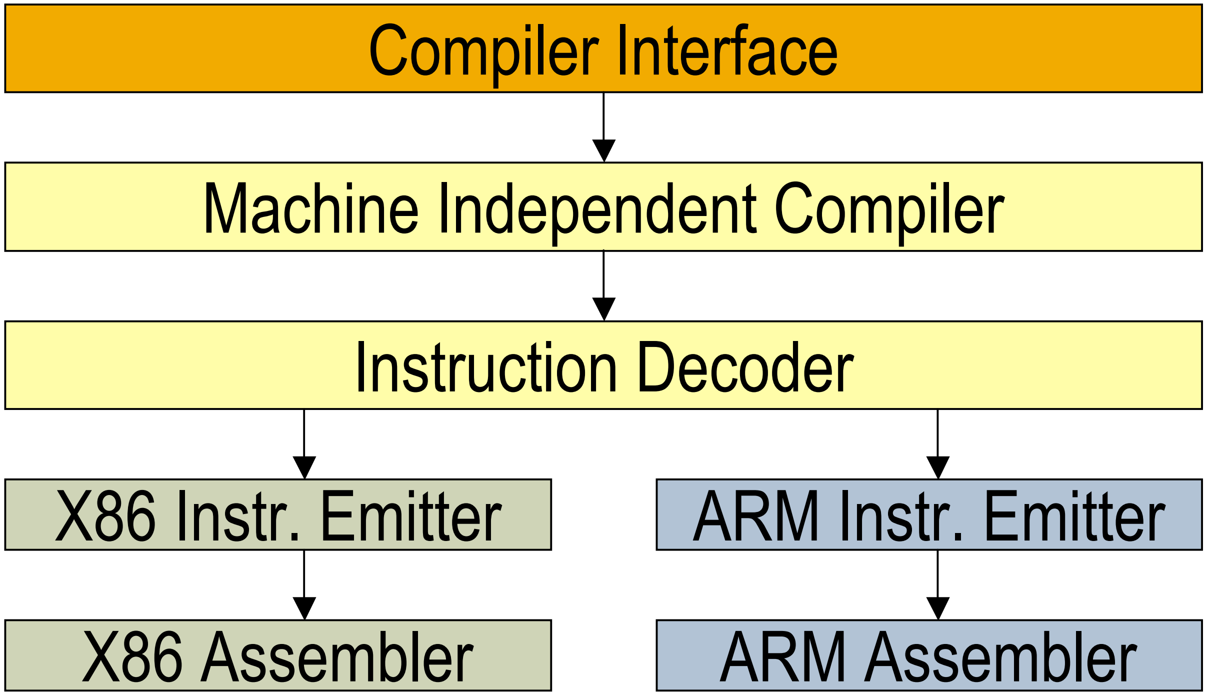
const_char 1004	→	c.literal(1004);
		c.push();

const_byte 32	→	c.literal(32);
		c.push();

ushr_i	→	c.pop(INT);
		c.pop(INT);
		c.ushr();
		c.push();

Simple Compiler

- Light-weight, single pass, non-optimizing compiler
- Designed for fast code compilation
- Currently supported back-ends - x86, ARM
- Will be used for jitting classes loaded at run time



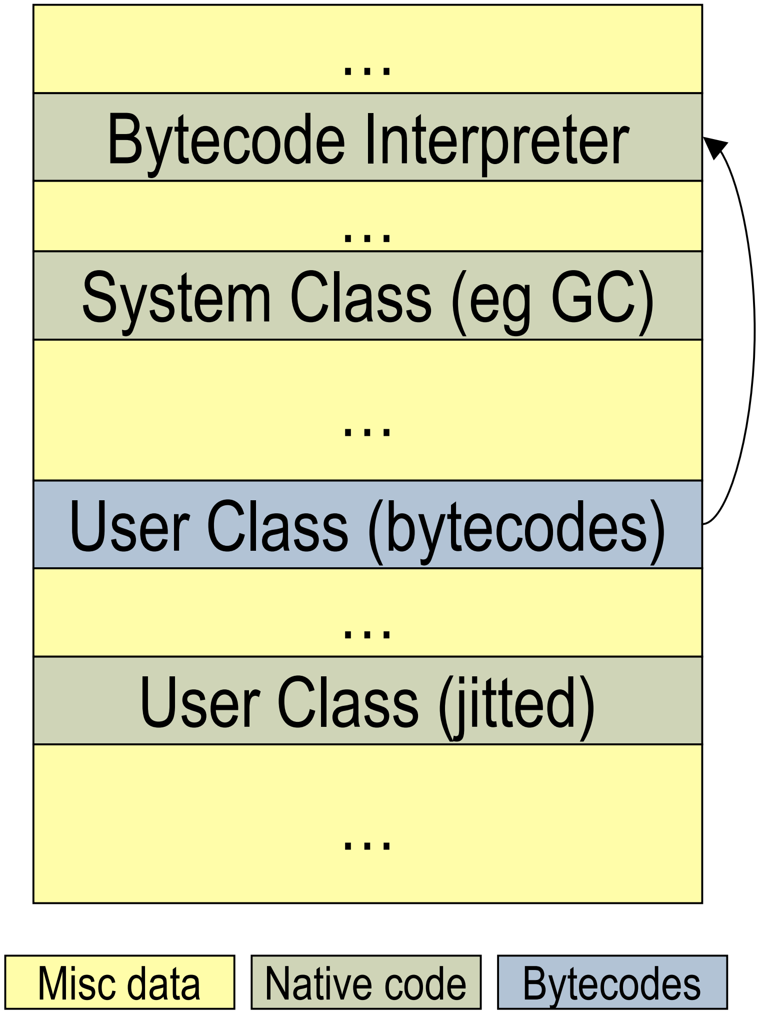
Optimizing Compiler

- Collaborative research with Usyd - Dr Bernhard Scholz
- Aim to test a new instruction selection technique - Partitioned Boolean Quadratic Programming as applied to general purpose machines
- Will be an optimizing compiler:
  - Constant folding
  - Dead code elimination
  - Partial Redundancy Elimination
  - Etc
- Will be used for:
  - Ahead-of-time compilation of system code (interpreter, libraries)
  - Potential jitting of frequently executed code for better performance

Squawk Bytecode Execution

- Currently using a Java interpreter macro-expanded and converted to C
- Future interpreter will be generated through the compiler interface
- Future bytecode execution modes:
  - > Interpreted
  - > Ahead-of-time compiled
  - > Just-in-time compiled
- Mix of bytecodes and compiled code in system memory

System Memory Layout



...

Bytecode Interpreter

...

System Class (eg GC)

...

User Class (bytecodes)

...

User Class (jitted)

...

Misc data | Native code | Bytecodes

Future Work

- Completion of Squawk compiler integration
  - Testing and debugging the interpreter
  - Compiling classes ahead-of-time
- Other compiler back-end implementations as needed
- Integration with Scholz's optimizing compiler framework
- Runtime profiling for better jitting
- Running benchmarks with standard JVMs vs Squawk using an optimizing compiler

Summary

- Squawk is a small, execute-in-place CLDC 1.0 compliant Java Virtual Machine
- Written in Java, not C
- Compiler interface for multiple compiler support
- Simple compiler for jitting classes at run time
- Optimizing compiler for ahead-of-time compilation
- Squawk code executed using mix of interpretation, ahead-of-time and just-in-time compilation
- Future work - compiler integration, benchmarking