JAVA DAYS ´99



Java™ HotSpot™ Virtual Machine

Bill Pataky
Senior Product Manager
Sun Microsystems

Sun microsystems

Agenda



- Overview
- HotSpot client vs. server
- Roadmap
- Technology
- Questions and answers

© 1999 Sun Microsystems GmbH

Sun microsystems



HotSpot™ Overview

© Sun Microsystems GmbH

Sun microsystems

HotSpot™ Overview



- Highest performance solution available
- State of the art JVM
  - Engineered for reliability
  - Accurate, pause-less garbage collection
  - Ultra-fast object allocation
  - Fast thread synchronization
- Dynamic optimizing compiler

© 1999 Sun Microsystems GmbH

Sun microsystems



Client vs. Server

© Sun Microsystems GmbH

Sun microsystems

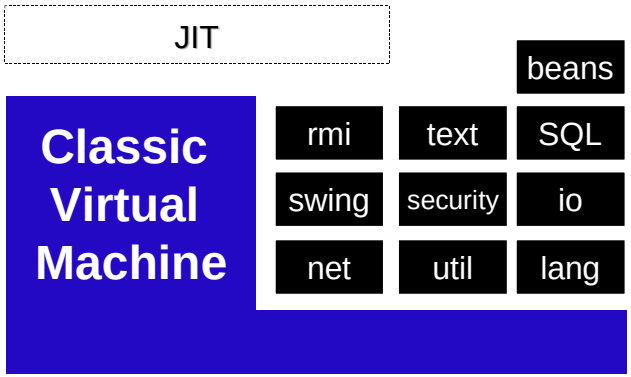
Client vs. Server



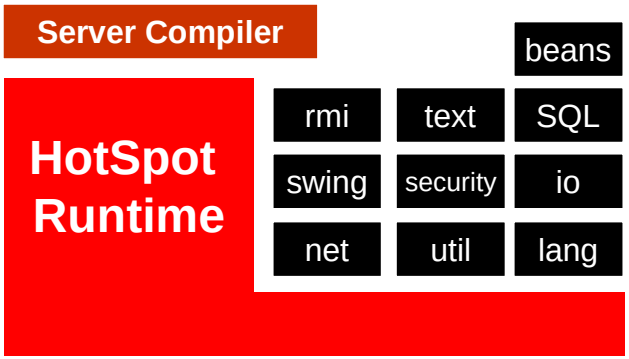
- Java™ HotSpot™ Server VM
  - HotSpot 1.0, 2.0
  - Flat out performance
- Java HotSpot Client VM
  - Part of Java 2 SE 1.3 (Kestrel)
  - Small memory footprint
  - Fast startup time
  - Fast interactive response

© 1999 Sun Microsystems GmbH

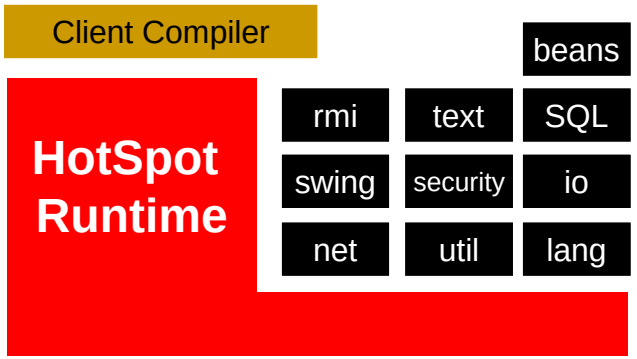
Java™ Runtime Environments



Java 2 SE 1.2
Classic VM



HotSpot 1.0
Java HotSpot™ Server VM



Java 2 SE 1.3
Java HotSpot™ Client VM

© 1999 Sun Microsystems GmbH



HotSpot™ Roadmap

© Sun Microsystems GmbH

Sun microsystems

Roadmap



- HotSpot™ 1.0
  - Windows FCS April 1999
  - Solaris/SPARC FCS June 1999
- HotSpot 2.0
  - Beta October 1999
  - FCS Q1 2000
- Java™ 2 SE 1.3
  - Beta August 1999
  - FCS Q1 1999

© 1999 Sun Microsystems GmbH

Sun microsystems



HotSpot™ Technology

© Sun Microsystems GmbH

Sun microsystems

HotSpot™ Technology



- Adaptive optimization
- Memory management
- Thread synchronization

© 1999 Sun Microsystems GmbH

Adaptive Optimization



- Only compile hotspots
  - Dynamically profile
  - Still interpret when sensible
- Compile fewer methods
  - More aggressive optimizations
  - Spread out compilation pauses
  - Preserves memory
- Dynamic de-optimization

© 1999 Sun Microsystems GmbH

Sun microsystems

Aggressive Inlining



- Bigger basic blocks
- Inlining of virtual methods
  - Class loading can invalidate code
  - Dependencies must be tracked

```
class A {
  int foo() { return 3; }
}
```


© 1999 Sun Microsystems GmbH

Sun microsystems

Dynamic Deoptimization



- Permits backing out of optimization decisions

```
class B extends A {
  int foo() { return 6; }
}
```

- Transforms compiled activations into interpreted activations

© 1999 Sun Microsystems GmbH

Sun microsystems

Memory Management Problems



- Use of time
  - Scalability to large heaps
  - Annoying pauses
- Use of space
  - Memory leaks
  - Fragmentation
  - Handles
- Programming style
  - Use of free lists

© 1999 Sun Microsystems GmbH

Sun microsystems

HotSpot™ Memory Management

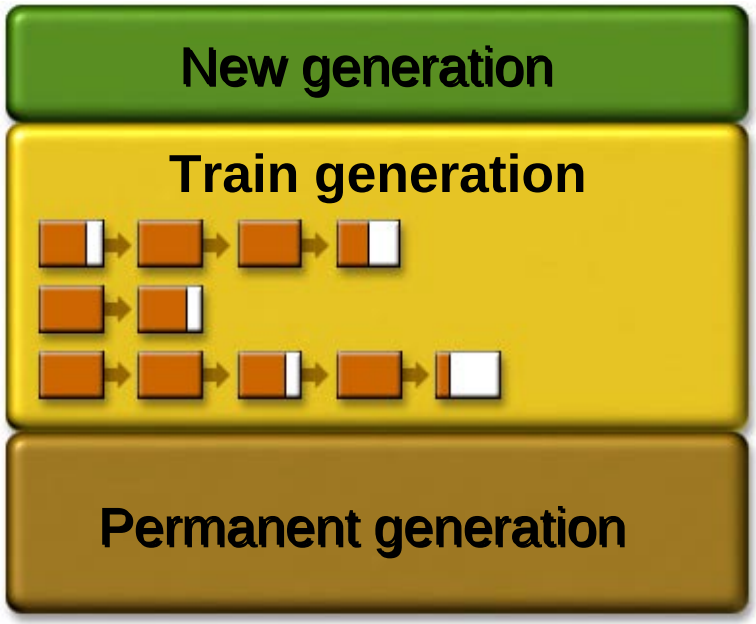


- Accurate garbage collection
- Fast allocation

- Collection scales to large heaps
  - Generational
  - Incremental

© 1999 Sun Microsystems GmbH

Three Collectors at Work



- Copying collector
  - Efficient since objects die young
- Train algorithm copying collector
  - Incremental collection of old objects
- Mark-compact collector
  - Relocate objects in place
- Typical pauses <10ms
- Software write barrier

© 1999 Sun Microsystems GmbH

Thread Synchronization



- Typically a major bottleneck in Java™ applications
- Common cases fast
- Allocate monitor on stack
- No increase in memory footprint
- Scales to large shared-memory multiprocessor systems

© 1999 Sun Microsystems GmbH

HotSpot™ Summary



- Fast
- Scalable
- Reliable
- Compliant
- Available on http://java.sun.com

© 1999 Sun Microsystems GmbH



Questions and Answers

© Sun Microsystems GmbH

Sun microsystems


Sun
microsystems