JAVA DAYS ´98

Project HotSpot

Robert Holt
Java software
Sun Microsystems Inc.

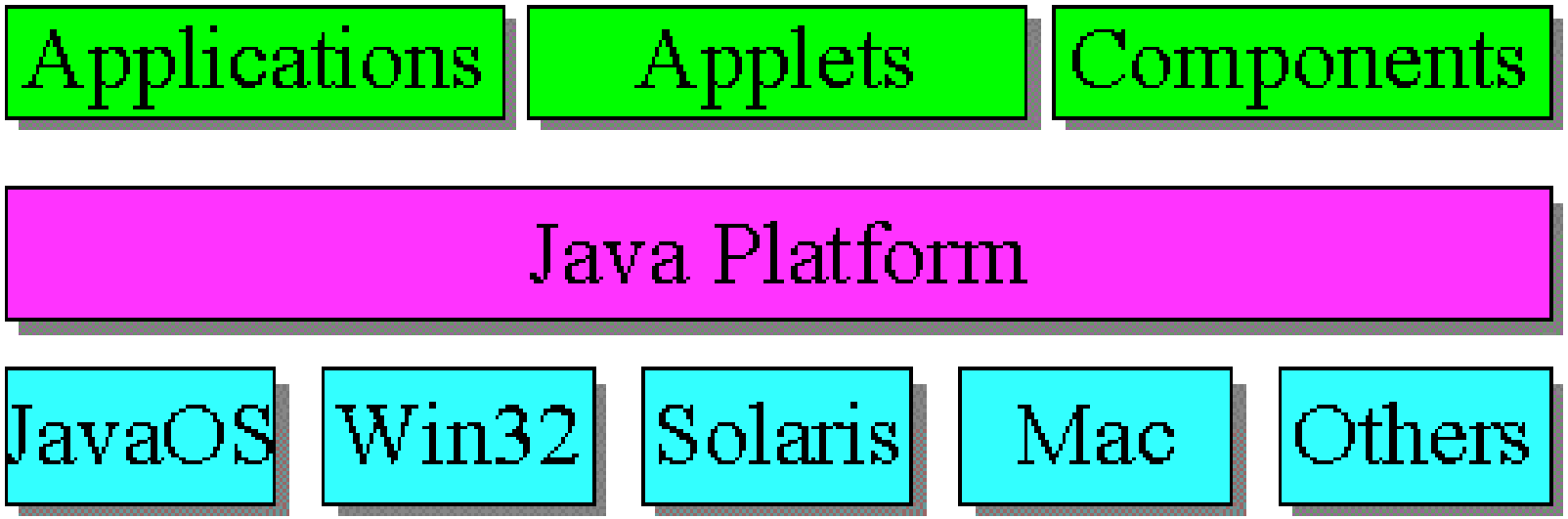
Sun microsystems

Agenda

- **Java architecture & why**
- **VM Genealogy / Evolution**
- **HotSpot**
  - **HotSpot architecture**
  - **Perceived problems and solutions**
  - **HotSpot technologies**

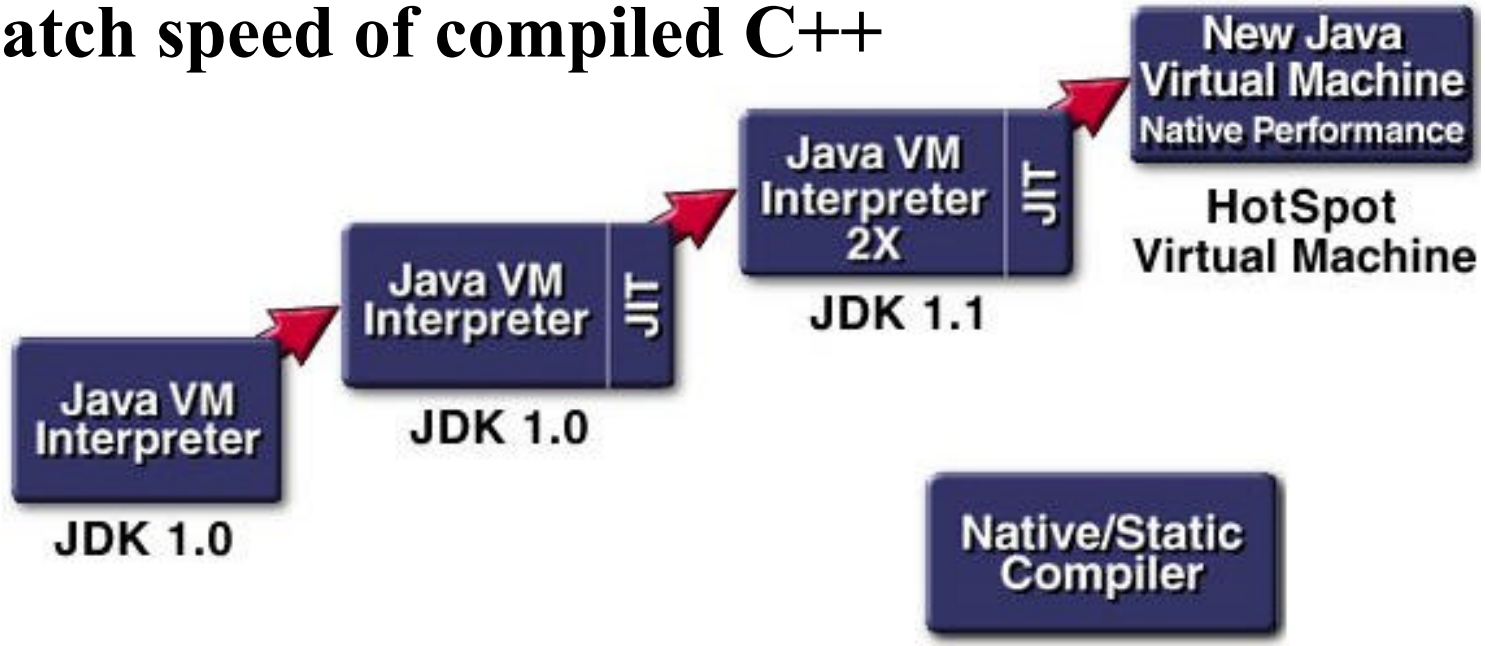
Java Architecture

- **Provide *the* universal platform for Internet, Intranet, & Application Server software development and deployment**

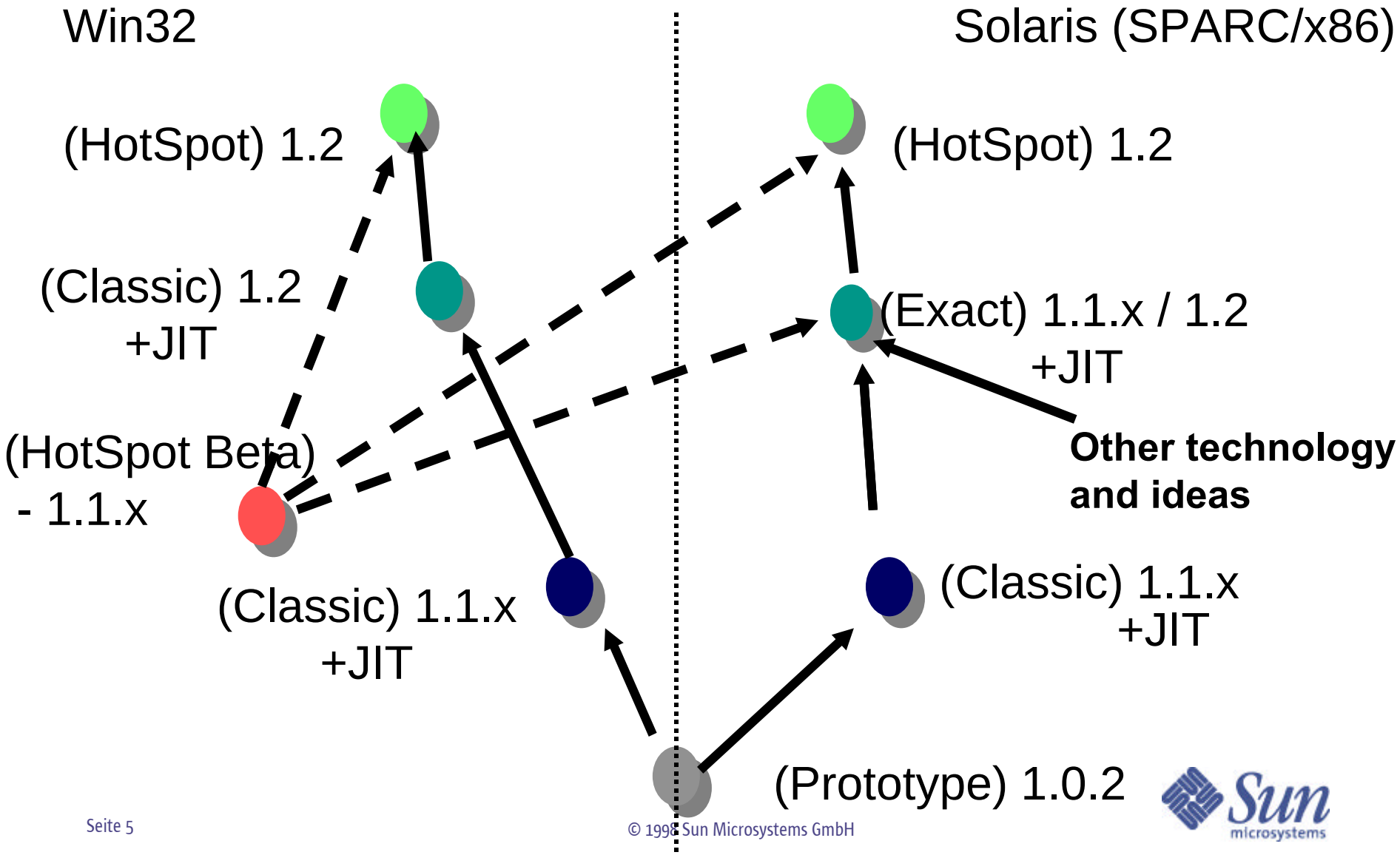


Java Performance Evolution and Objectives

- Enable good object-oriented code
- Eliminate portability vs. performance trade-off
- match speed of compiled C++



VM Genealogy / Evolution



Prototype VM

- **Engineering project**
  - **everything interpreted**
  - **conservative garbage collector**
  - **HotJava 1.0.2**
  - **HUGELY popular when put onto web !!**
    - **(we love Duke!)**

Classic VM

- **Productised VM**
  - **better performance**
  - **- still conservative garbage collection**
  - **more APIs**
    - **(Beans, JDBC, RMI, rework of AWT)**
  - **Add Symantec JIT - greatly increased performance**

Exact VM

- Base for next Solaris production VM
- highly reengineered VM
- both 1.1.x and 1.2.x
- Includes some HotSpot technology
  - some inlining (and de-inlining (deoptimization)), core VM alg has some HotSpot technology,
  - Generational Garbage Collection (x4-5)
- add JIT = fast, scaleable, production VM

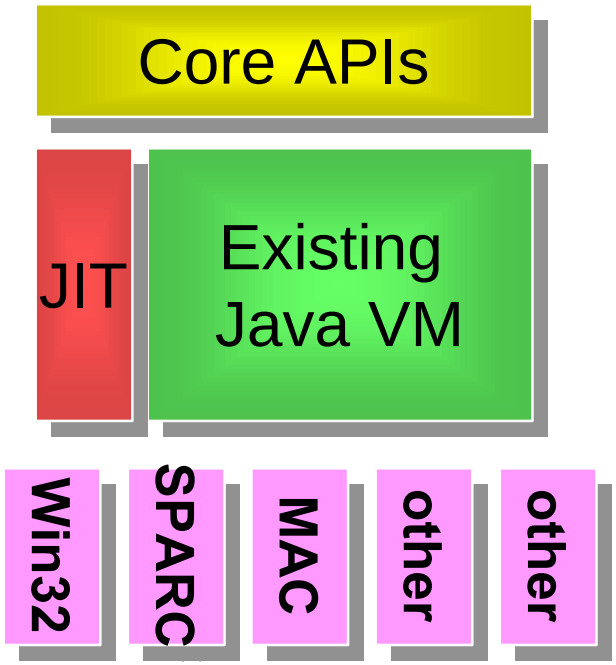
Sun microsystems

HotSpot Core VM

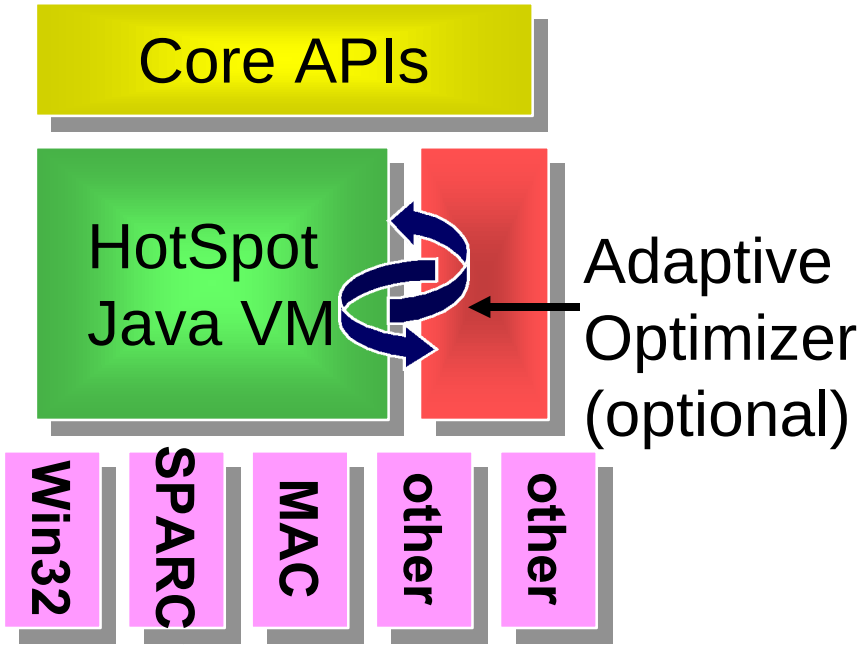
- New, faster object allocation/deallocation
- New, faster core interpreter algorithm
- Generational Garbage Collection (exact) (x4-5)
- Faster thread synchronization

Platform Architecture with HotSpot

- **Classic VM**



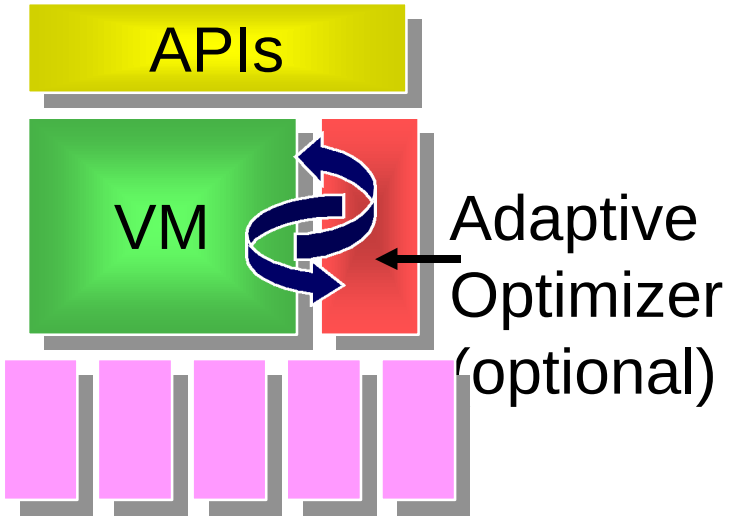
With HotSpot



Faster Object Allocation

- Object heap doesn't need to be contiguous
- Handless Objects
  - Better performance
- Two machine-word object headers
  - rather than 3 (except for arrays)
  - 8% savings on total object memory usage

Faster Core Interpreter



- **Core Interpreter**
  - **Coded in C++**
  - **learns about code compiles code that is used most**
  - **Can (if required) be cached between runs or stored in ROM**
- **Partial machine code translation ("snippets")**
  - **Translates frequently used byte-code sequences into native code "snippets"**

Generational GC

- **Fully accurate garbage collector**
  - **Reliable reclamation of object memory**
  - **Objects are relocatable, allowing memory compaction**
- **Generational Copying Collection**
  - **Object "nursery" (over 95% of Java objects are temporary)**
  - **Adaptive "promotion" algorithm**
  - **Mark-Compact "Old Object" collector**

Faster Thread Syncronisation

- 2-50x performance gain for synchronized threads!
- Patented process for synchronization of threads - uses as little as 4 CPU cycles!
- Zero space overhead for uncontended thread synchronization

HotSpot Core VM Summary

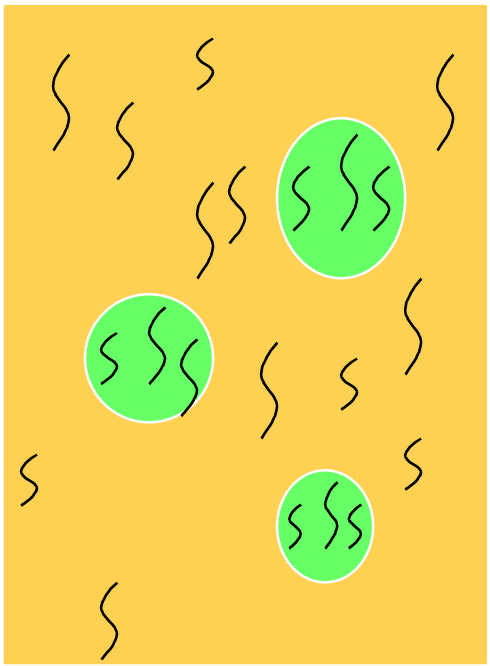
- Source code available to all existing Java licensees - no charge
- 50% performance gain (or more!)
- Roughly same footprint as existing VM
  - Can be targeted for small footprint environments

HotSpot Adaptive Optimizer / Compiler

- Fully adaptive compiler
- Inlining virtual methods
- Incremental ("pauseless") Garbage Collection
- "Soft Real Time" performance
- C-quality optimzer and code generator

HotSpot Adaptive Optimizer/Compiler

- Run in interpreted mode first to detect critical "hot spots" in a program
- Optimize only critical pieces
  - Inlinine virtual methods
    - Dynamic deoptimization
  - Programs start faster (compilation is "spread out")
  - Lower memory usage
  - On-the-fly adjustmant for best results fully adaptive compiler



Method Inlining

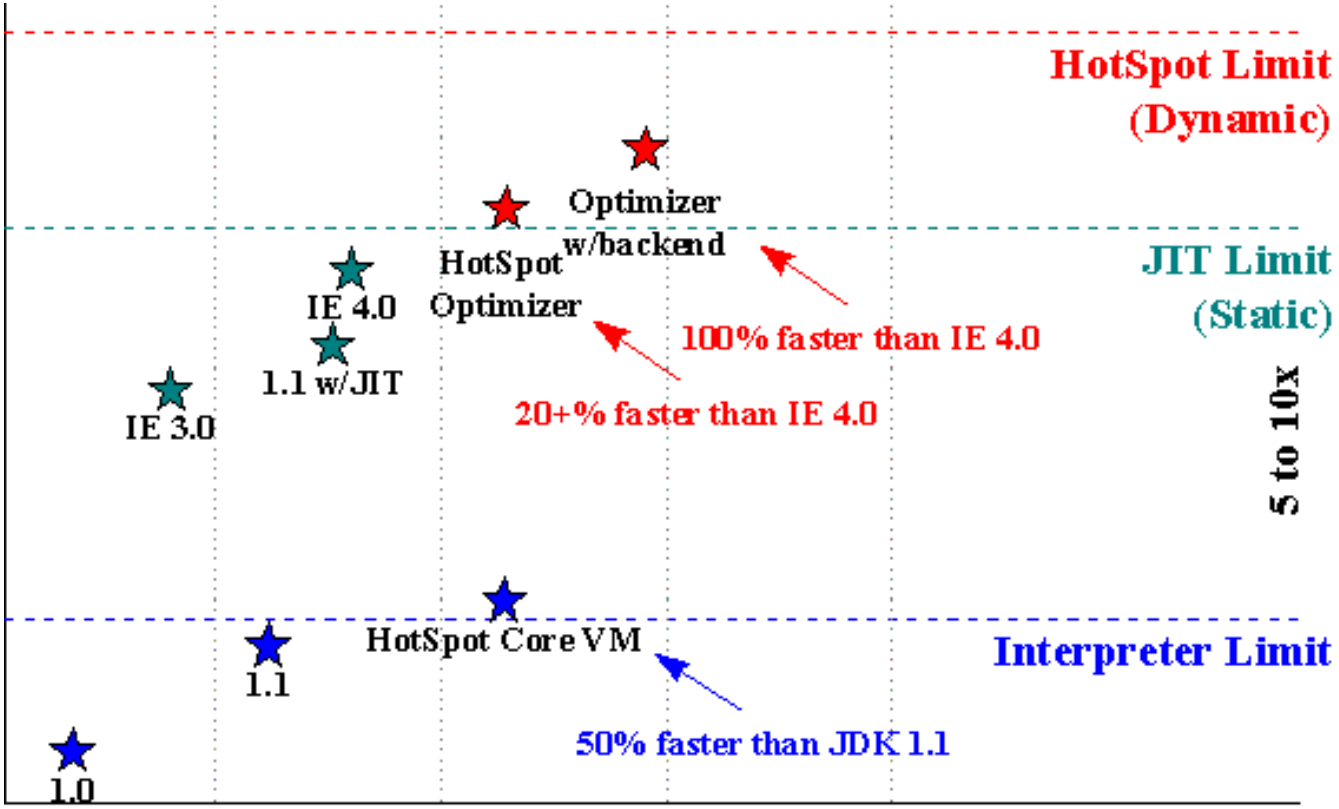
- Dynamic inlining as opposed to static
- Produces larger blocks of code for optimization
- Eliminates overhead of multiple method invocations
- Designed to handle dynamic loading of new Java code (via de-optimization backout)

Incremental Pauseless Garbage Collection

- Collects only from "old object" section
  - "Train" algorithm that garbage collects in millisecond increments
  - Relocate groups of tightly "coupled objects into regions of adjacent memory

Performance Evolution

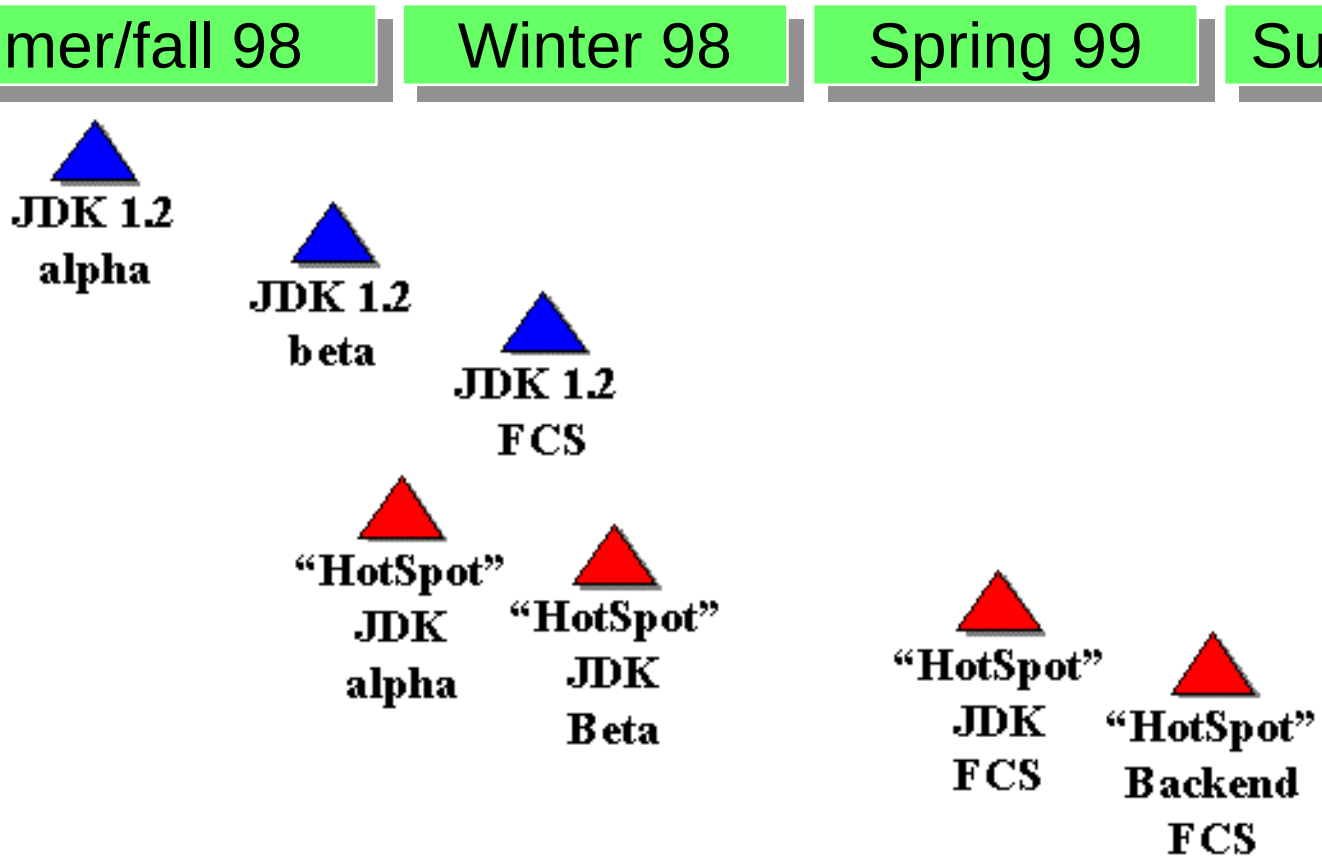
- **HotSpot at beginning of performance curve**



HotSpot Optimizer effects

- Programs start faster as less compilation is performed up-front when compared to JIT
- Compilation is spread over time, making compilation pauses shorter and less noticeable
- Compiles only performance critical code, saving time as well as consumed memory
- Delaying compilation provides time to gather information to perform better optimizations.

JDK Roadmap



Q & A

Sun microsystems