

# Trends in JVM Software Development

Jeremy Singer

# Overview

- 5 observations
- 5 implications
- Possible research directions

# (1) Impetus of open-source

- GNU classpath lists 16 open-source JVM projects
- Sun recently released HotSpot with GPL
- Growing number of research VMs
  - Jikes RVM (IBM)
  - Rotor (Microsoft)
  - Open Runtime Platform (Intel)
  - Moxie (?)

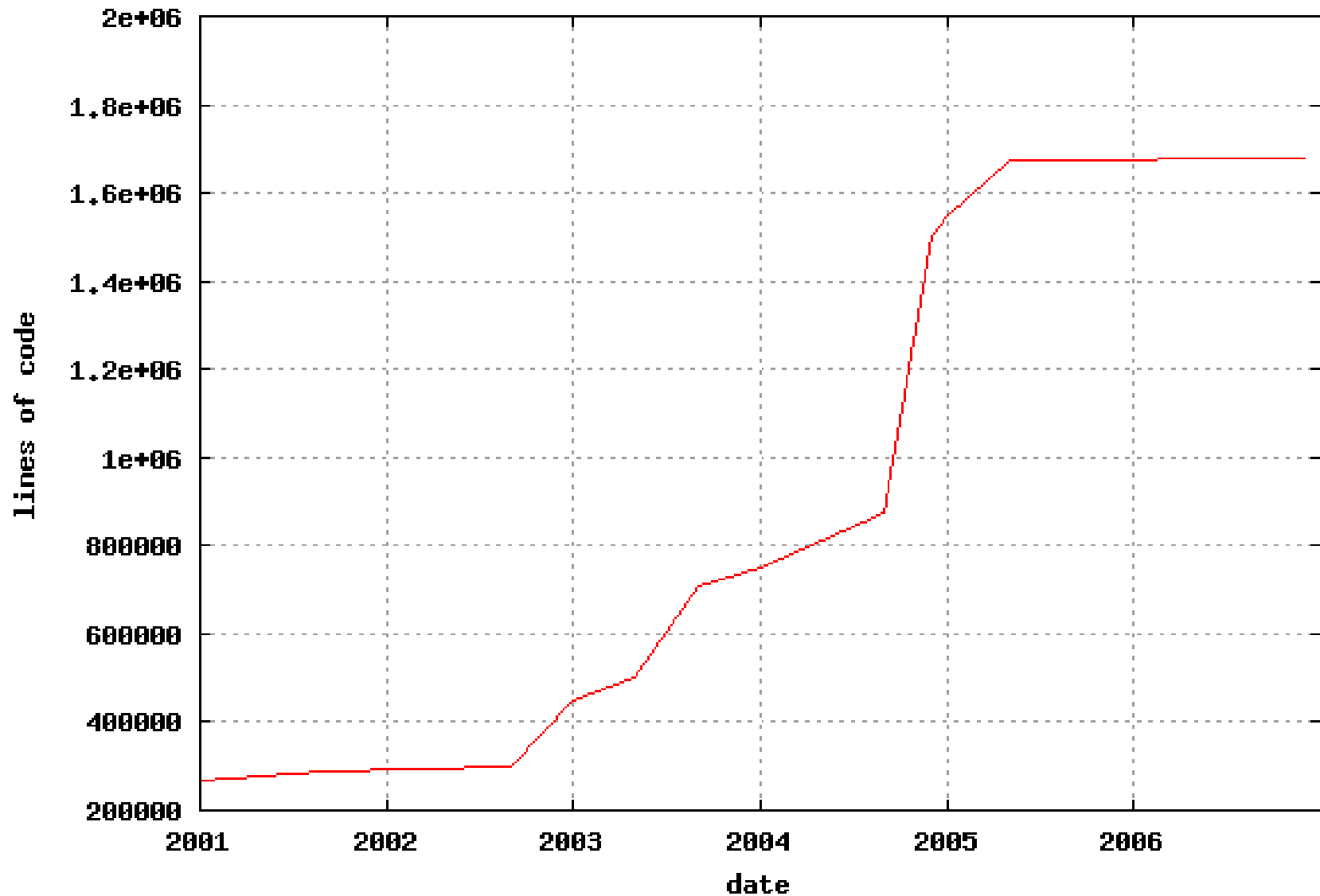
# Implication 1

- Increasing visibility of JVM systems source code
- Opportunities to
  - Inspect it
  - Improve it

## (2) Increasing size of JVMs

- Kaffe, early open source VM
- Has evolved
  - From simple interpreter
  - To JIT compiler
  - Supporting many recent Java features

# Kaffe source code growth



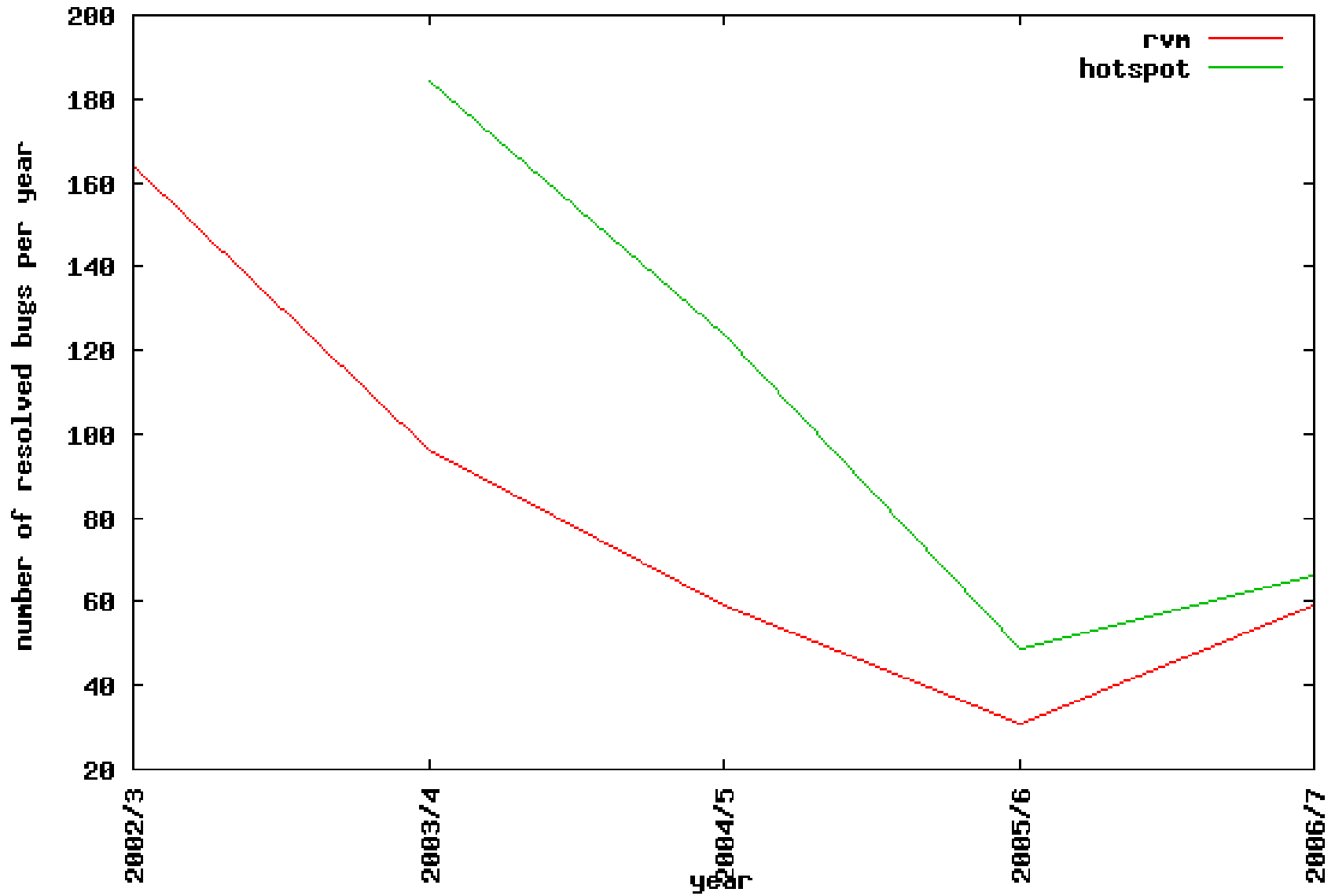
# Implication 2

- Static complexity of JVMs has increased by an *order of magnitude* in *five years*
- Affects system
  - Comprehensibility
  - Maintainability
  - Extensibility

### (3) Empirical study of maintainability

- Query bug databases of open source code bases
- See how many bugs are fixed per year
  - Jikes RVM (on [sourceforge.net](http://sourceforge.net))
  - HotSpot (on [bugs.sun.com](http://bugs.sun.com))

# Bugs fixed per year



# Implication 3

- Reached a turning point
- Due to increasing static complexity, maintainability is becoming a problem
- Forecast is for worse

# (4) X-ray of adaptive runtime

- Lots of adaptive services operating
  - Invisible to user at runtime
  - Invisible to application programmer at compile time
- Example JVM services
  - Garbage collection
  - JIT compilation

# Visualized adaptive runtime services



- baseline compiler
- optimizing compiler
- garbage collector
- non-VARS code

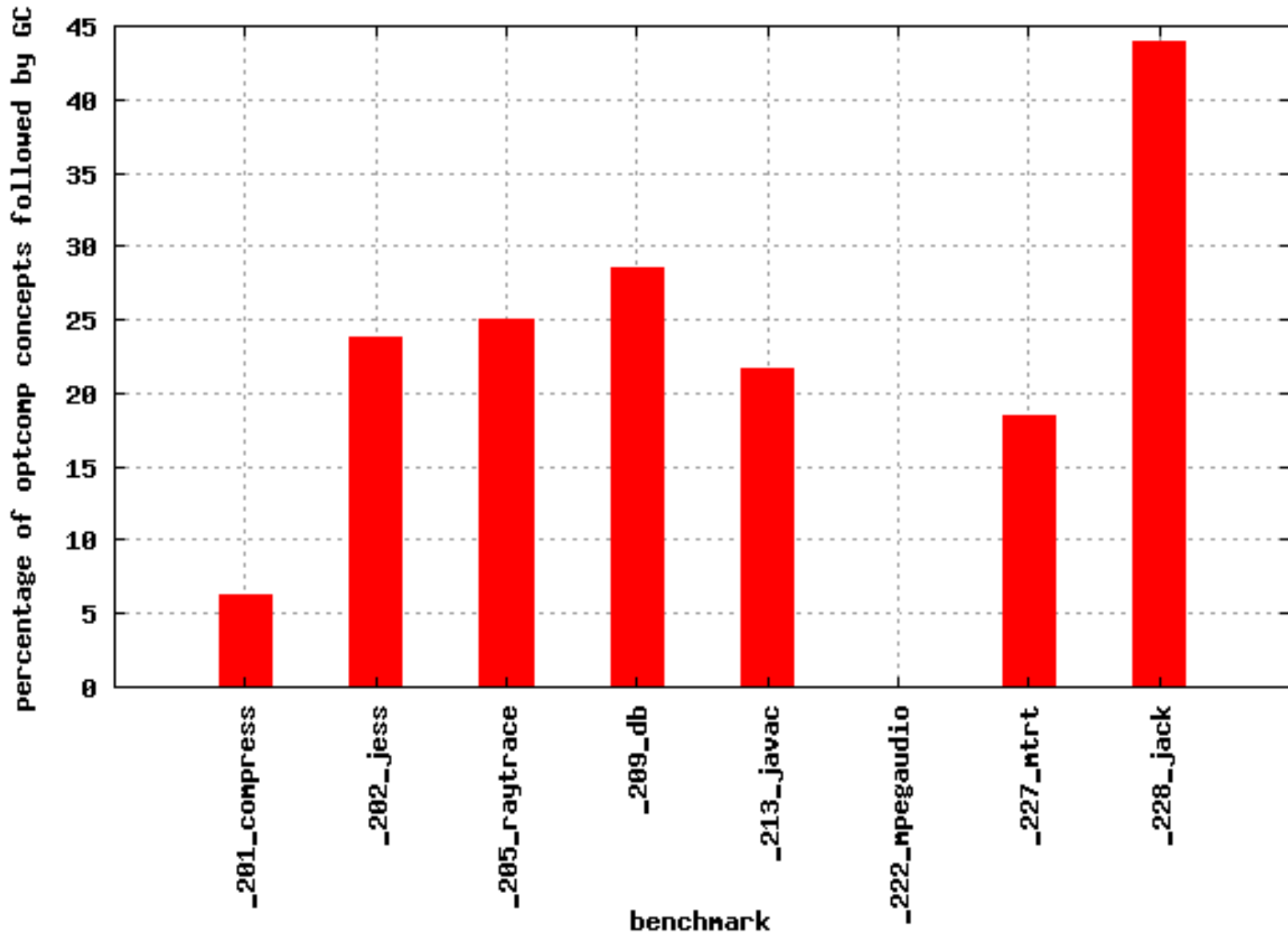
# Implication 4

- Dynamic complexity
- Many distinct services operating
  - Independently
  - Throughout execution time
  - On demand
  - Take up significant execution time

# (5) Service Interactions

- Some of these services may interact with one another
  - Symbiotic relationships (good)
  - Disruptive relationships (bad)
- Jikes RVM example – OptComp and GC

# OptComp / GC interactions



# Implication 5

- Need to manage interactions properly
- Avoid disruptive interactions
- Take advantage of symbiotic interactions
- Requires effective whole-system scheduler
- Reduce dynamic complexity
- Decrease execution time(!)

# Conclusion

- JVMs are becoming increasingly complex
  - Statically
  - Dynamically
- This complexity must be
  - Quantified
  - Managed
  - Visualized