Research Plan

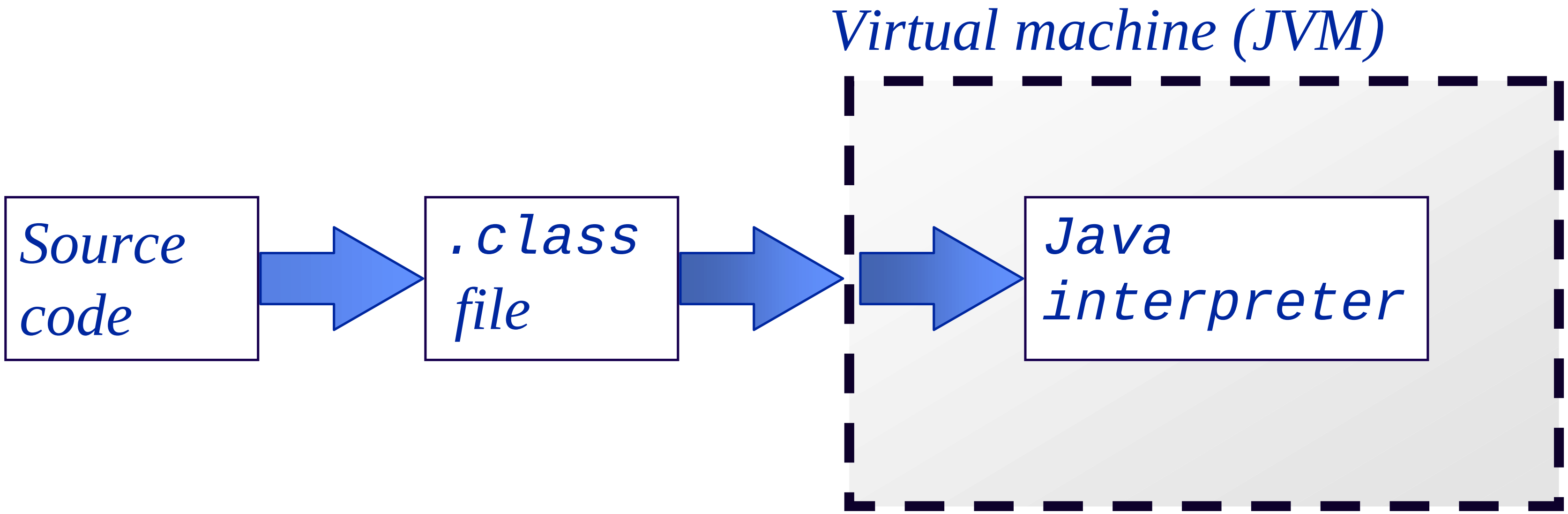
Tim Harris

14th October 1997

© 1997 APM Ltd

Initial JVM implementations

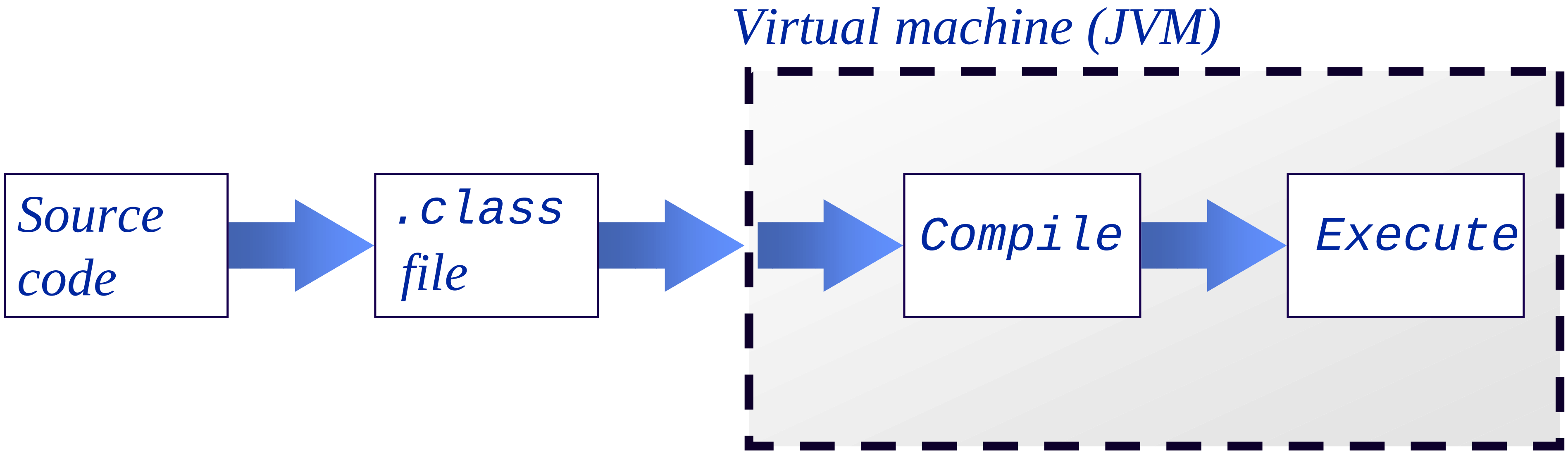
- The JVM loads `.class` files and uses an interpreter to execute the bytecode that they contain.



© 1997 APM Ltd

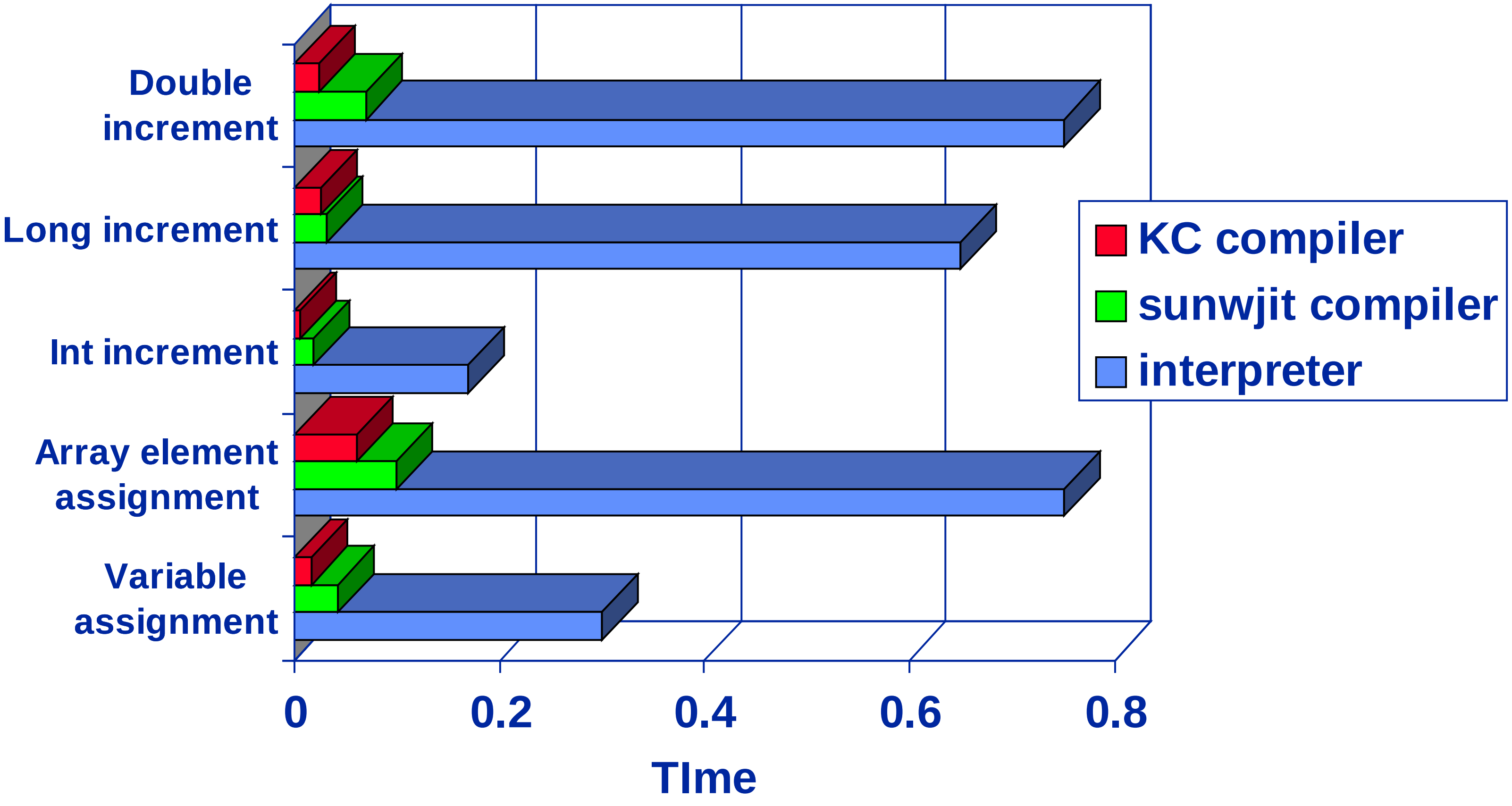
Adding a JIT compiler

- Native code is generated within the JVM.
- The `.class` file format is not changed.



© 1997 APM Ltd

Benchmark results



© 1997 APM Ltd

Pros and cons

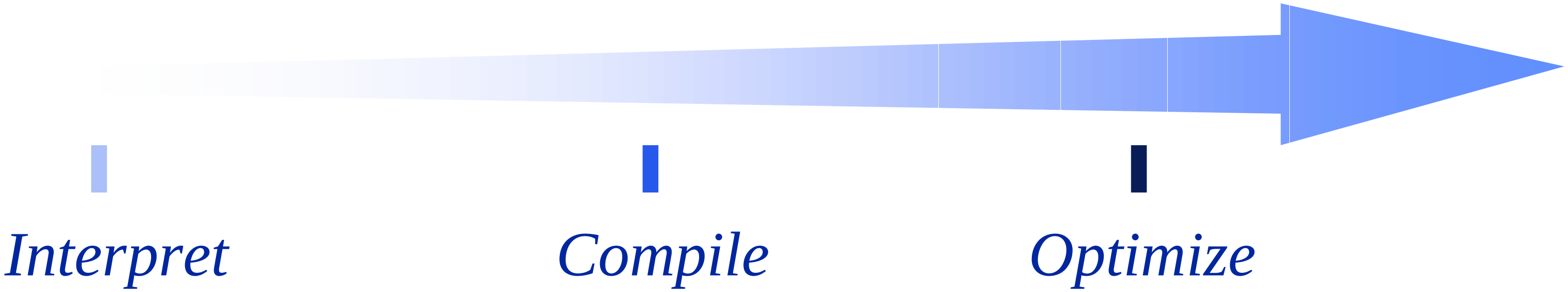
- ✓ Performance can be 40x better:
  - Code is optimized for the processor.
  - Processor registers can be used for storage.
  - Virtual-method lookup can be simplified.
  - Redundant run-time checks can be removed.
- ✓ `.class` file machine independence preserved.

- ✗ Compilation introduces pauses:
  - What if the application is dealing with continuous media streams?
- ✗ Programmer cannot control when/if methods should be compiled.

© 1997 APM Ltd

Should everything be compiled?

- Programmer could choose run-time strategy:



Interpret *Compile* *Optimize*

  - Compile and then re-optimize with feedback directed results?
- When should compilation occur?
  - Class at a time? Method at a time?
- These choices could only really be hints.

© 1997 APM Ltd

How can QoS requirements be expressed?

- What would ‘n processing every k seconds’ mean?
  - Wide variation in JVM performance.
  - Which thread is ‘charged’ for compiling a library method?
- QoS requirements for access to memory/IO devices as well as processing?
  - eg control over garbage collection strategy.

© 1997 APM Ltd