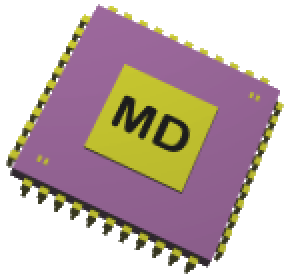
Java Virtual Machines für ressourcenkritische eingebettete Systeme und Smartcards

Frank Golatowski · Hagen Ploog · Ralf Kraudelt
Tino Rachui · Olaf Hagendorf · Dirk Timmermann



Universität Rostock
Fachbereich Elektrotechnik und Informationstechnik
Institut für Angewandte Mikroelektronik und Datentechnik

Überblick

- **Java für Embedded Systems**
- **Java auf Smartcards**
  - Sprachumgebung
  - JavaCard Runtime Environment
- **Die Java Virtuelle Maschine**
  - Aufbau
  - Funktionsweise
- **Implementierung**
  - Merkmale

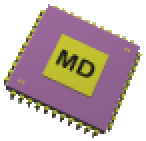


Java für Embedded Systems

- **PC-basiertes System miniJava**
  - 80386 und höher
  - Basis: JVM Kaffe von Transvirtual

- **mikrocontroller-basiert**
  - Testplattform für Smartcard und Prozessor ⇒ JavaCard-Umgebung
  - zu Testzwecken compilierbar mit jedem ANSI-C-Compiler

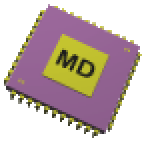




Java auf Smartcards Hardware

- **Mikroprozessoren**
  - 8-32-bit-Prozessor mit 3,5-5 MHz Taktfrequenz
- **Speicher**
  - ROM (16 - 32 KByte)
  - RAM (0,5 - 1 KByte)
  - EEPROM (8 - 16 KByte), NVRAM
- **Ein- und Ausgabe**
  - serielle Schnittstelle





Java auf Smartcards Besonderheiten

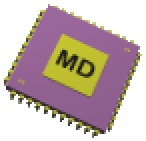
- **Einschränkungen der Sprache**
  - keine Strings, Chars, Fließkommazahlen, Threads
  - keine <clinit>-Methoden, (java.*),mehrdimensionalen Arrays
- **Erweiterungen der Klassenbibliothek**
  - neue Packages (javacard.* , javacardx.*)
- **Einschränkungen der JVM**
  - nicht alle Bytecodes verwendet
  - Stack: 127 Byte vs. 64K Worte
  - Instanzmethoden: 127 Byte vs. 65535 Byte
  - Felder einer Klasse: max. 255 Byte
  - Anzahl Array-Elemente: 32767 Byte vs. Max_long



Java auf Smartcards Besonderheiten

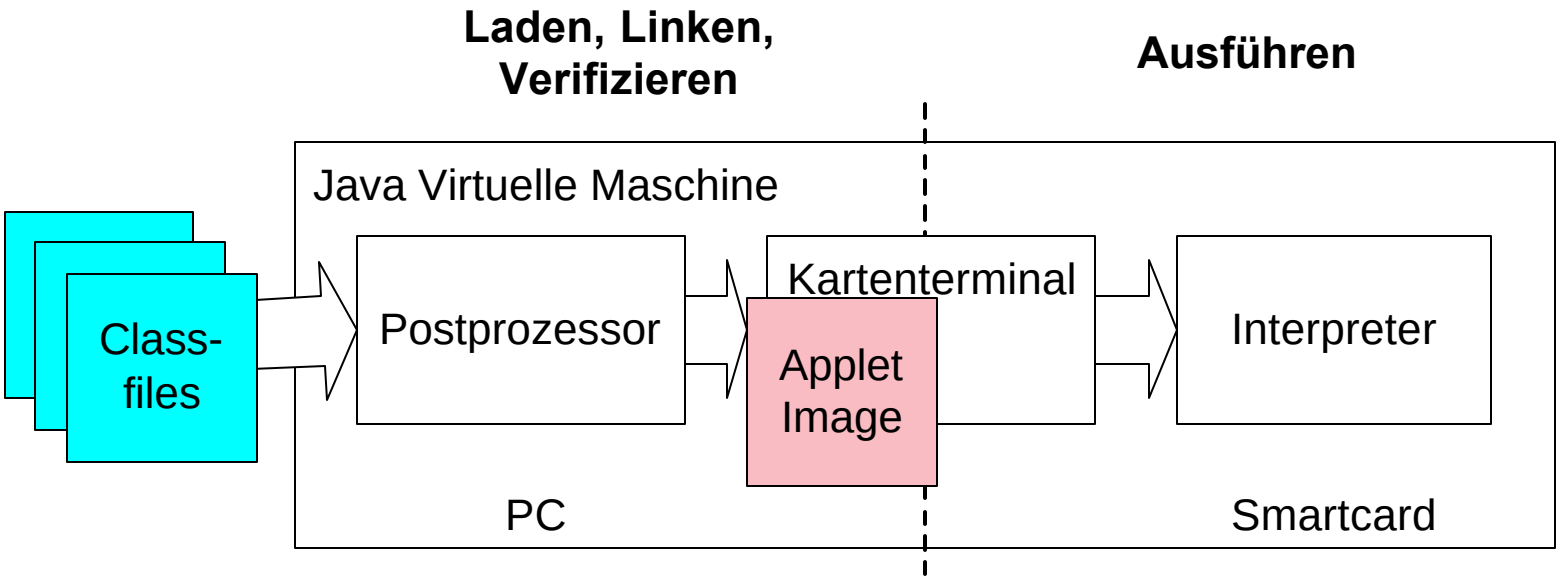
- **JavaCard Runtime Environment**
  - kein Garbage Collector ⇒keine Freigabe von mit NEW reserviertem Speicher
  - Instance sharing
    - Kommunikation mit CAD über APDU
    - Exception Behandlung
  - Atomare Operationen, Transaktionen
  - Persistente und transiente Objekte





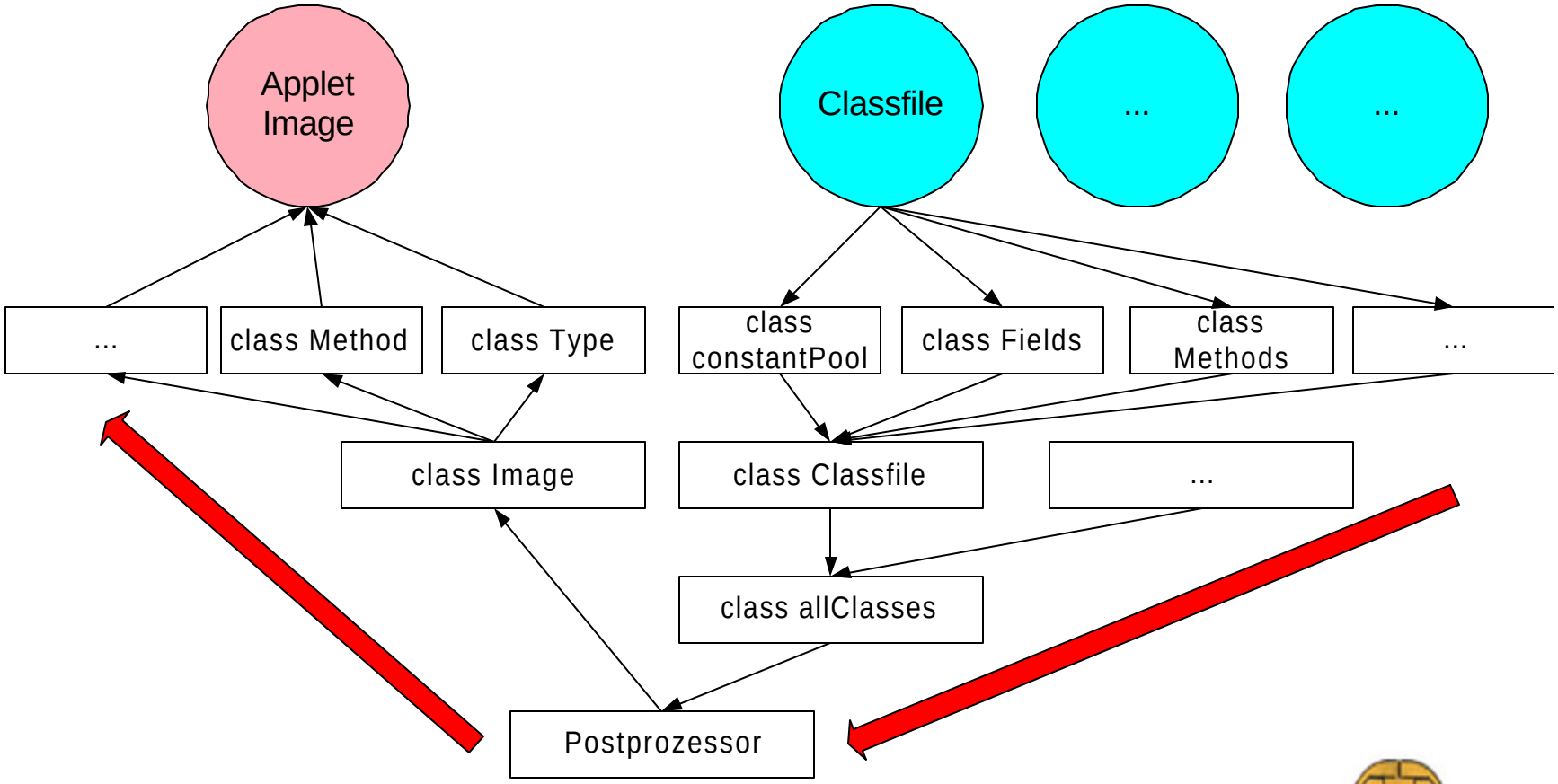
Virtuelle Maschine Aufbau

- **Die geteilte Java Virtuelle Maschine**



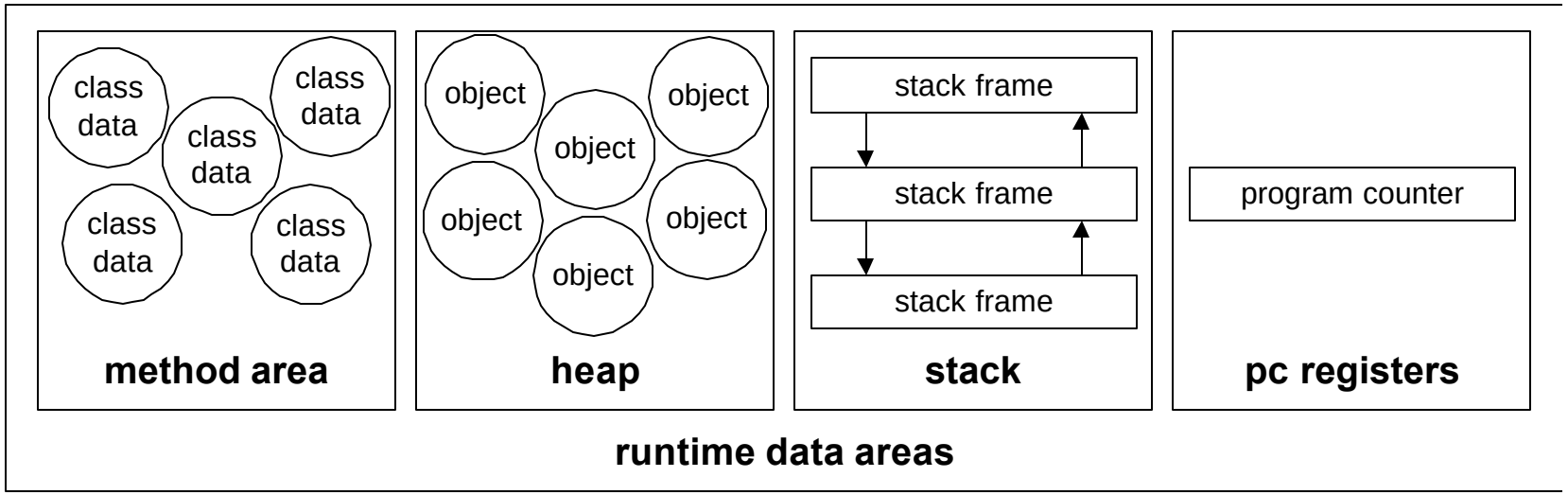
Implementierung Aufbau

- **Arbeitsweise des Postprozessors**



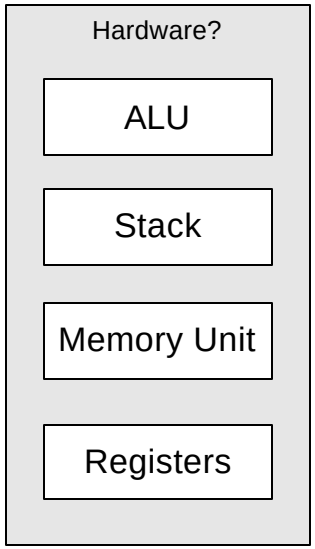
Virtuelle Maschine Aufbau

- Laufzeitbereiche der Virtuellen Maschine

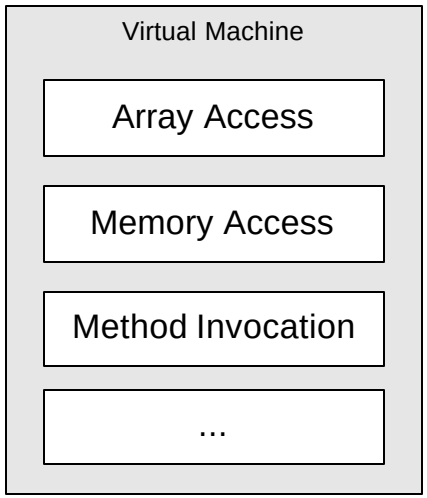


Implementierung Modularisierung

- **hardwareorientiert**

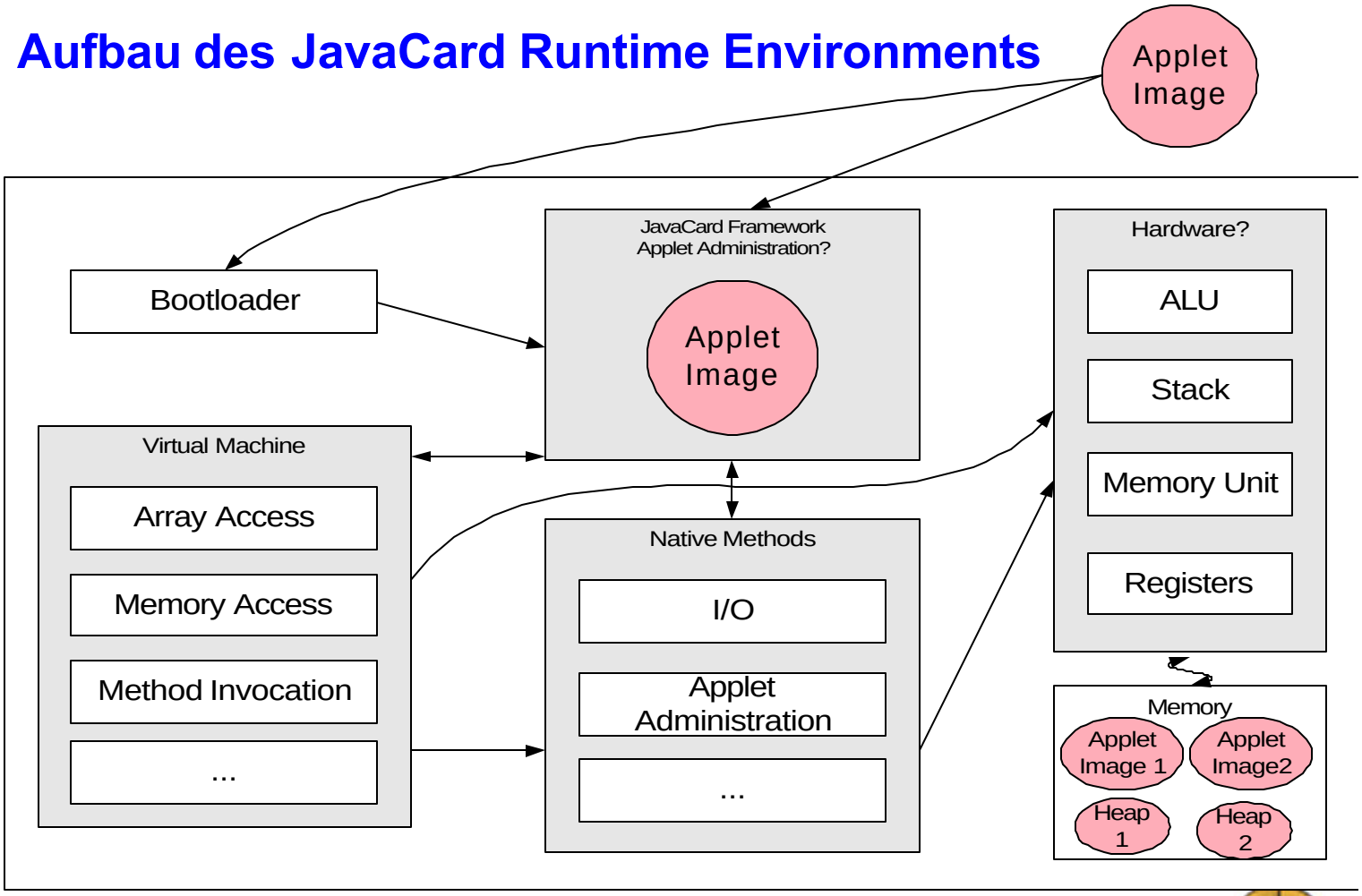


- **bytecodeorientiert**



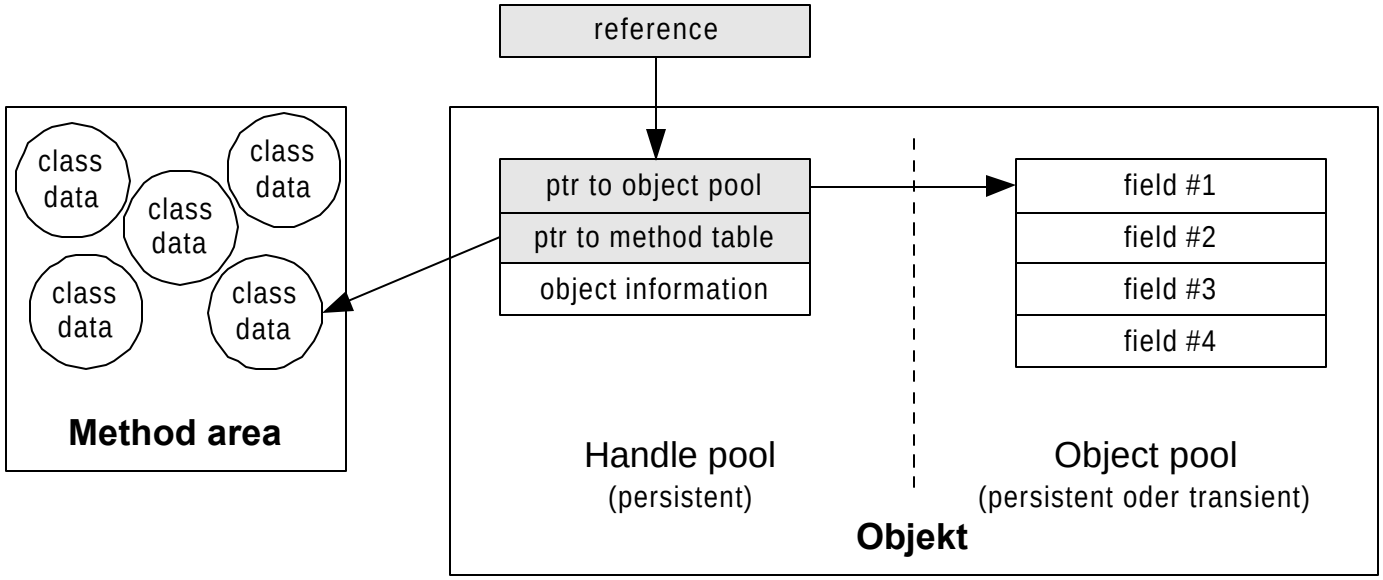
Implementierung Merkmale

- **Aufbau des JavaCard Runtime Environments**



Virtuelle Maschine Aufbau

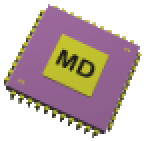
- **Aufbau eines Objekts auf dem Heap**



Implementierung Merkmale

- **Postprozessor**
  - C++
- **JavaCard Runtime Environment**
  - C, Umstellung auf Java für Java-Prozessor
  - läuft on board und off board
  - kein Betriebssystem
  - keine zusätzlichen Bytecodes
- **Sicherheit**
  - kein direkter Hardwarezugriff durch Applet, nur über API
  - zusätzlicher Schutz von API-Aufrufen (z. B. Speicherzugriff) durch Hardware
  - keine Stack- und Bytecodeüberprüfung zur Laufzeit

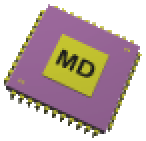




Ergebnisse der Implementierung

- Statt „Case“-Schleife ➔ Modularisierung
- Modularisierung in Bezug auf µP-Komponenten und Bytecodegruppen
- 14 KByte ROM
- Kosten der Modularisierung 3 KByte

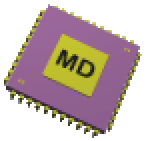




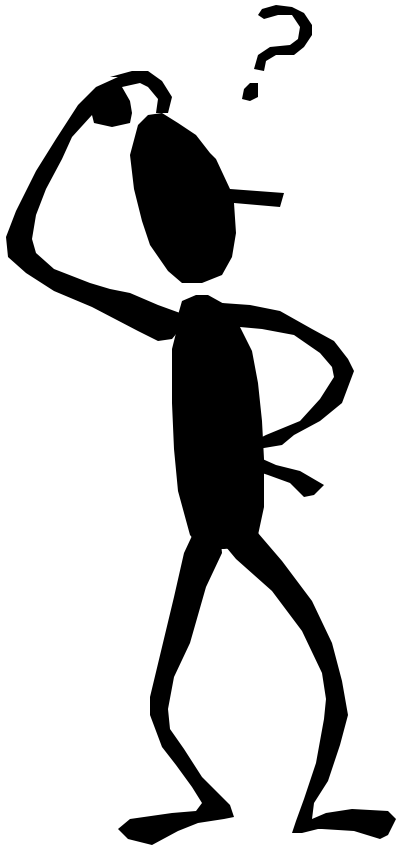
Zusammenfassung

- **3 verschiedene JVM für ressourcenkritische Systeme**
  - 80x86, 8051, VHDL
- **Hier: Konzentration auf 8051-Implementierung**
- **Ziel: Entwicklung eines Java-Prozessors**



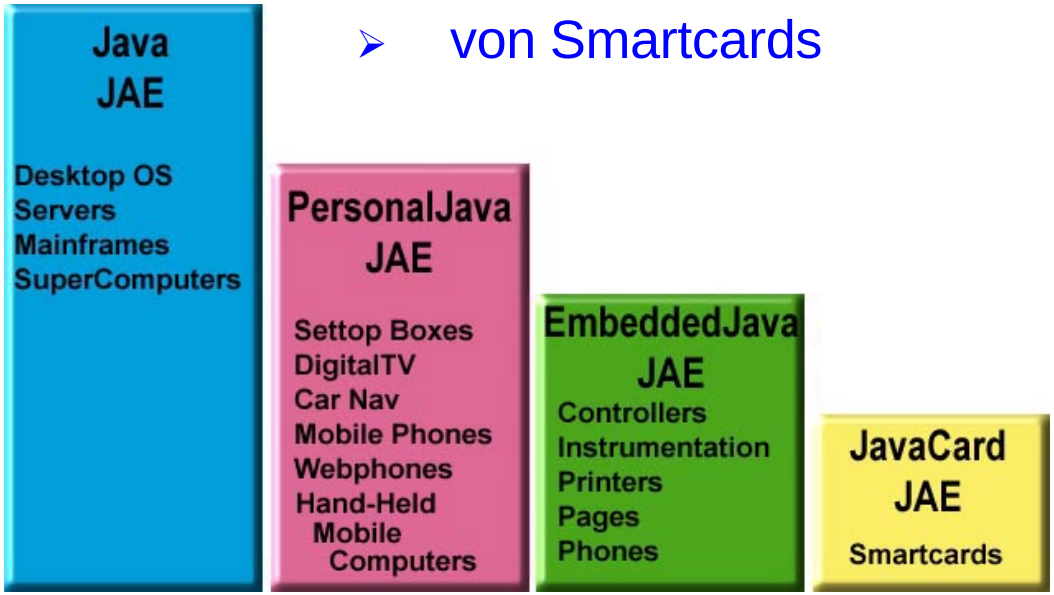


Fragen?

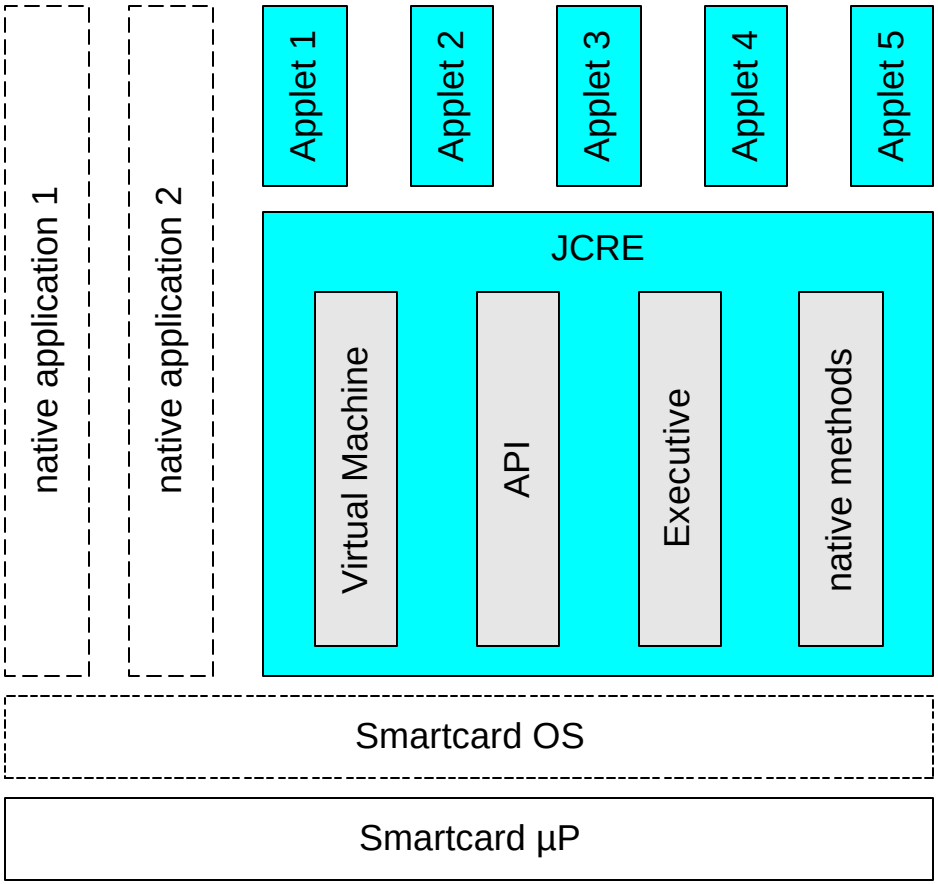


Einführung Java

- **Java als Programmiersprache**
  - des World Wide Web,
  - verteilter Anwendungen,
  - von Embedded Systems,
  - von Smartcards

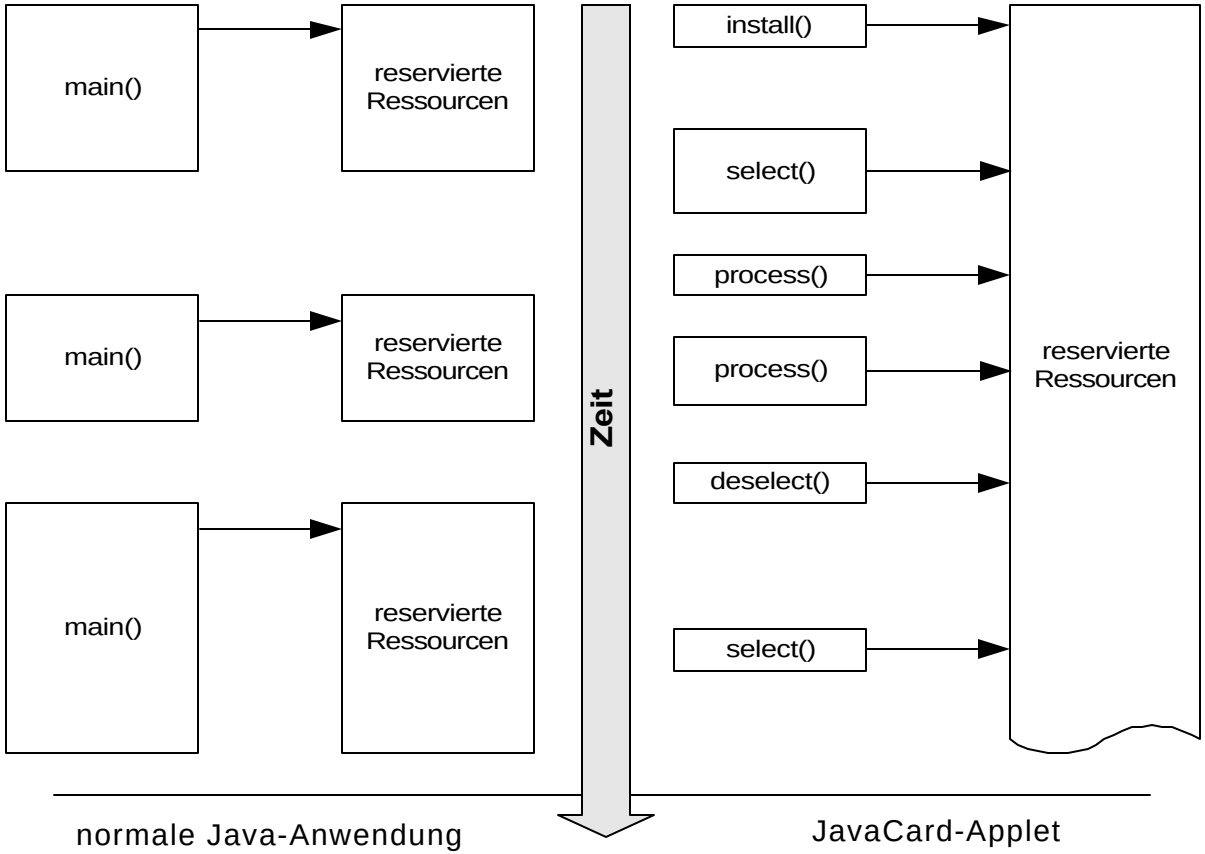


Java auf Smartcards Runtime Environment



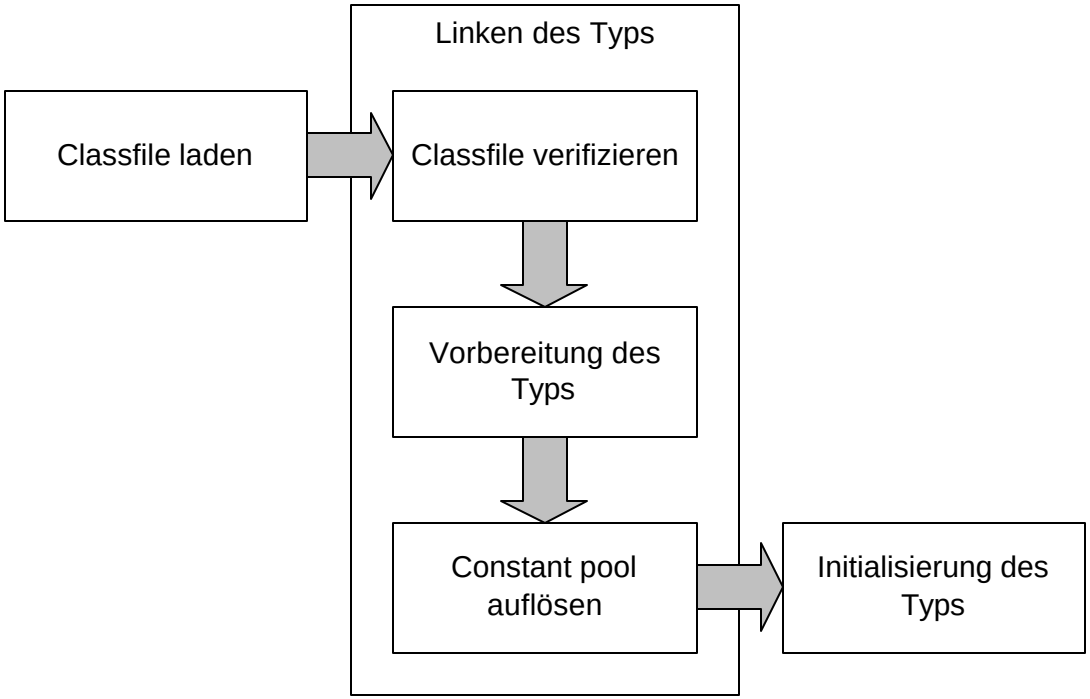
Java auf Smartcards Runtime Environment

- main() vs. Applet



Virtuelle Maschine Funktionsweise

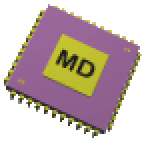
- **Funktionsweise des Postprozessors**
  - kein dynamisches Nachladen auf die Smartcard (Sicherheit, Performance)
  - API-Typen nicht nachladen, da schon auf Karte vorhanden



Virtuelle Maschine Funktionsweise

- **Linken**
  - Beziehungen zu anderen Typen auflösen = Nachladen
  - Verifikation
    - strukturelle Konsistenz, Bytecode
  - Vorbereitung
    - Klassenvariablen initialisieren (erst auf Karte)
    - z.B. Methodentabelle aufbauen
  - Constant Pool auflösen
    - symbolische durch direkte Referenzen ersetzen
    - Verkleinern und Umordnen des Constant Pools
- **Zusammenfassen aller Typen im Applet Image**





Virtuelle Maschine Funktionsweise

- **Constant-Pool-Auflösung durch Postprozessor**

