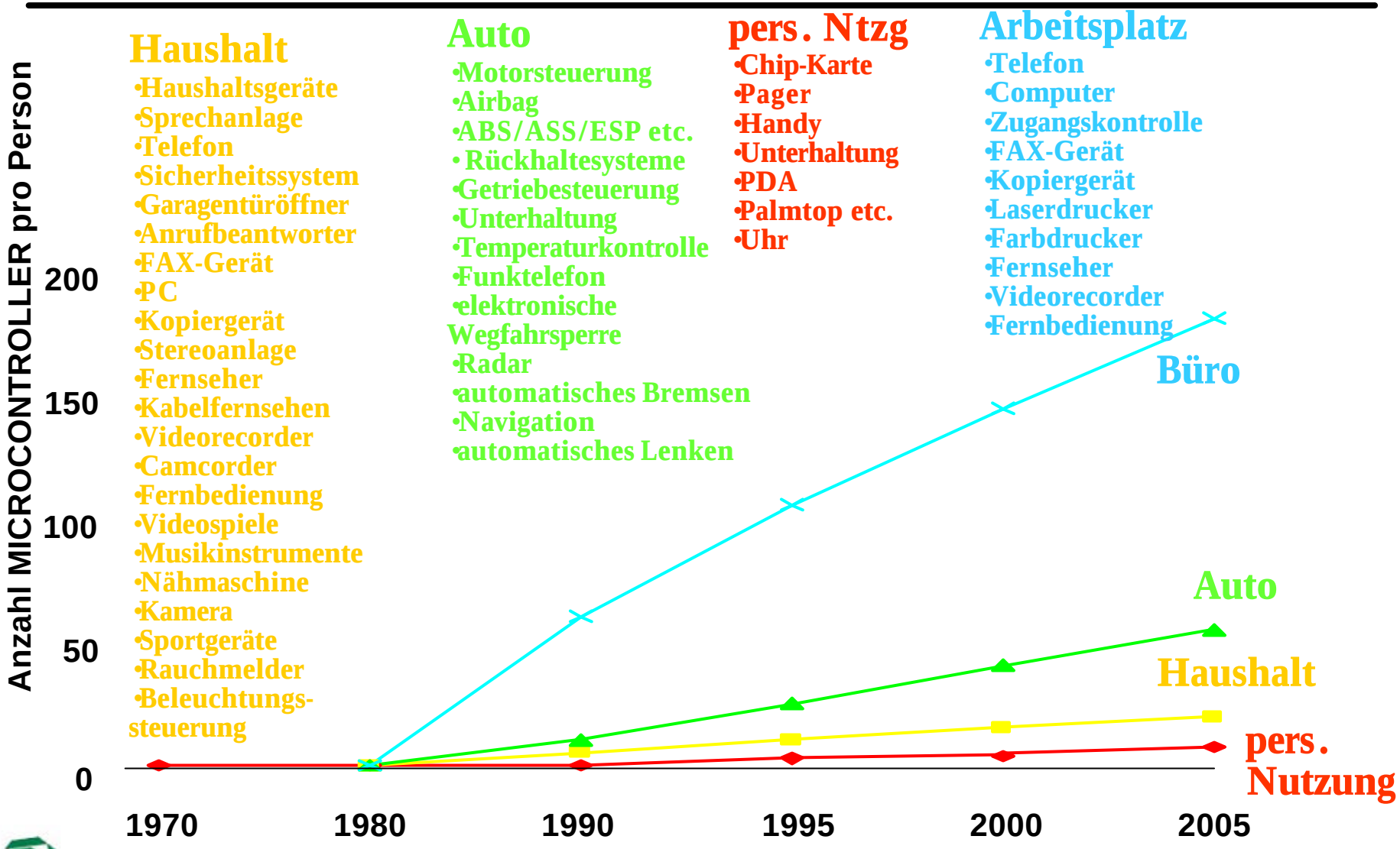
Java und eingebettete Systeme

Wolfgang Rosenstiel
Stephen Schmitt
Universität Tübingen
Arbeitsbereich “Technische Informatik”
und FZI-Forschungsbereich
“Systementwurf in der Mikroelektronik”

Gliederung

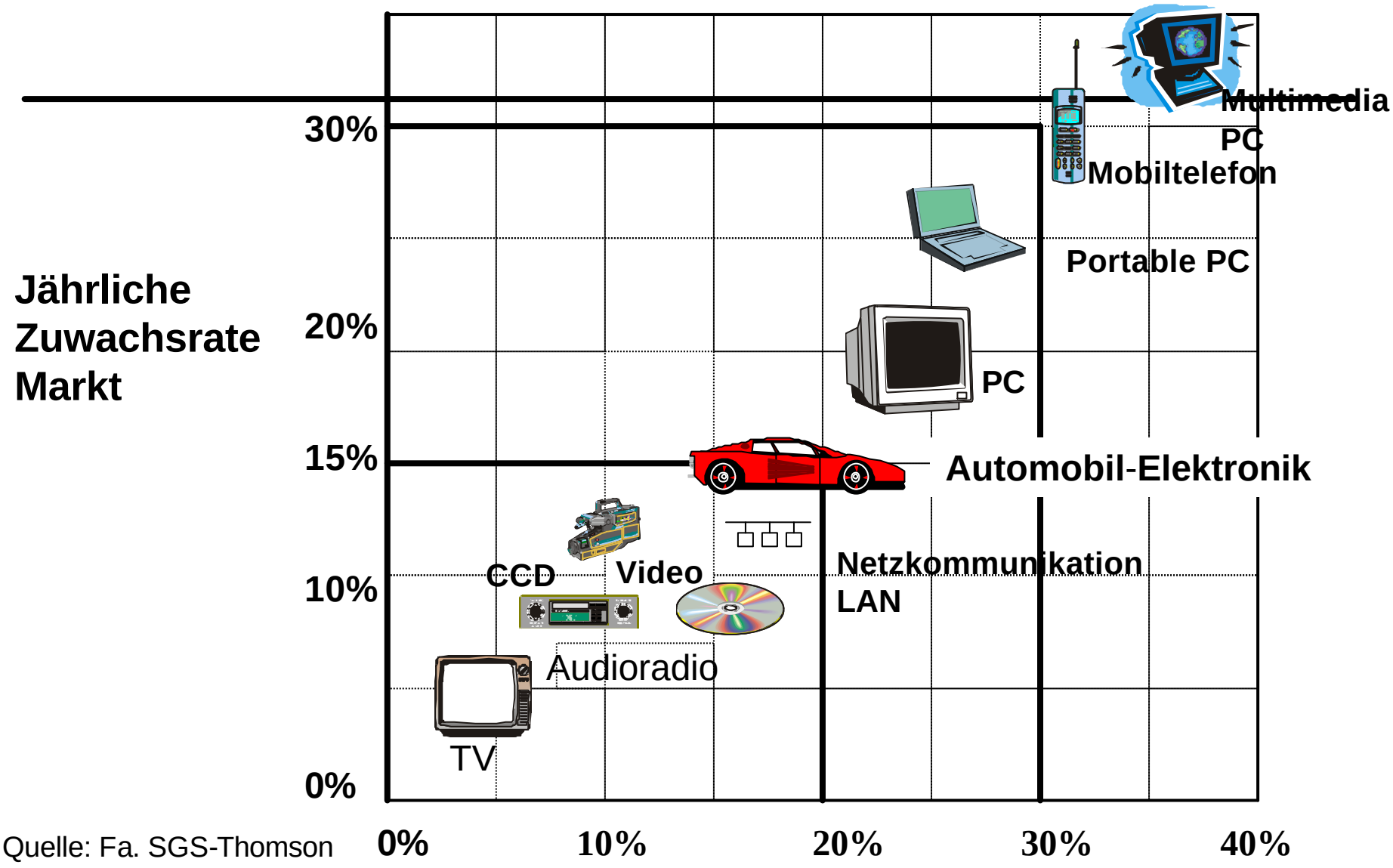
1. Motivation
2. Java in eingebetteten Systemen:
   Probleme und Lösungsmöglichkeiten
3. Java für verteilte eingebettete
   Systeme
4. Internetgestützte Steuerung und Kontrolle
   eingebetteter Systeme mit Java
5. Java2-Plattform und eingebettete
   Systeme
6. Zusammenfassung

„Elektronik-Explosion“



Anforderungen des Marktes

- **Günstige Produkte**
- **Schnelle Markteinführung**
- **Konfigurierbare Produkte**
- **Sichere Softwareentwicklung**



Quelle: Fa. SGS-Thomson

Entwicklung der Elektronik in der Autoindustrie

- Durchschnittskosten für die Gesamtfertigung
  - pro Auto
    $7.000/Auto
- geschätzte Anzahl gefertigter Autos 50 Mio.
- Inhalt der elektrischen und elektronischen Teile 30%

- $ 7. 000/Auto x 50 Mill. Autos x 30%

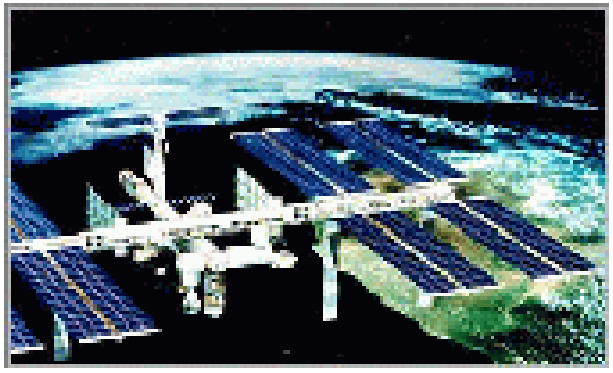
  = ca. 100 Milliarden $
- jährliches Wachstum 2000 - 2005 = 26%

Eingebettete Systeme - und wo sie zu finden sind

Automobilindustrie

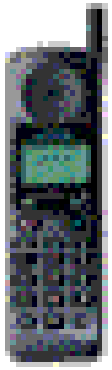


Luft- und Raumfahrt



industrielle Automation





Telekommu-nikation

Haushalt



Eingebettete Systeme - und deren Anforderungen

Echtzeit

techn. Randbed.

Sicherheit

Fehlertoleranz

Qualität

Bedienbarkeit

Umwelteinflüsse



Time-to-Market

Entwurfskosten

Turn-around-Zeiten

Produktionskosten

sonstige Randbed.

Eingebettete Systeme - Gemeinsamkeiten von Waschmaschine und Sojourner



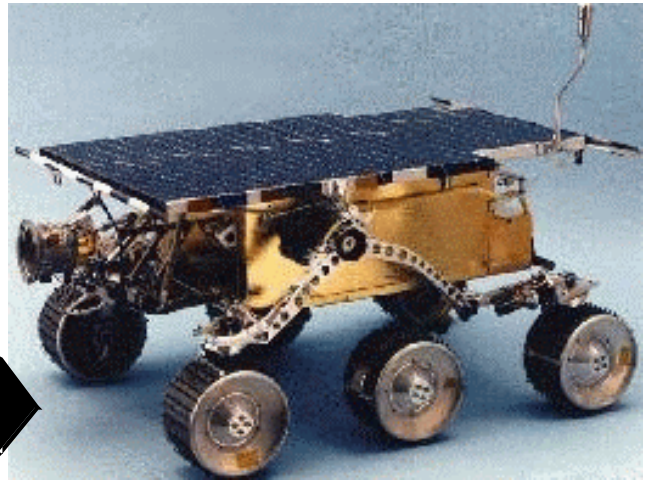
Schnittstellen

M

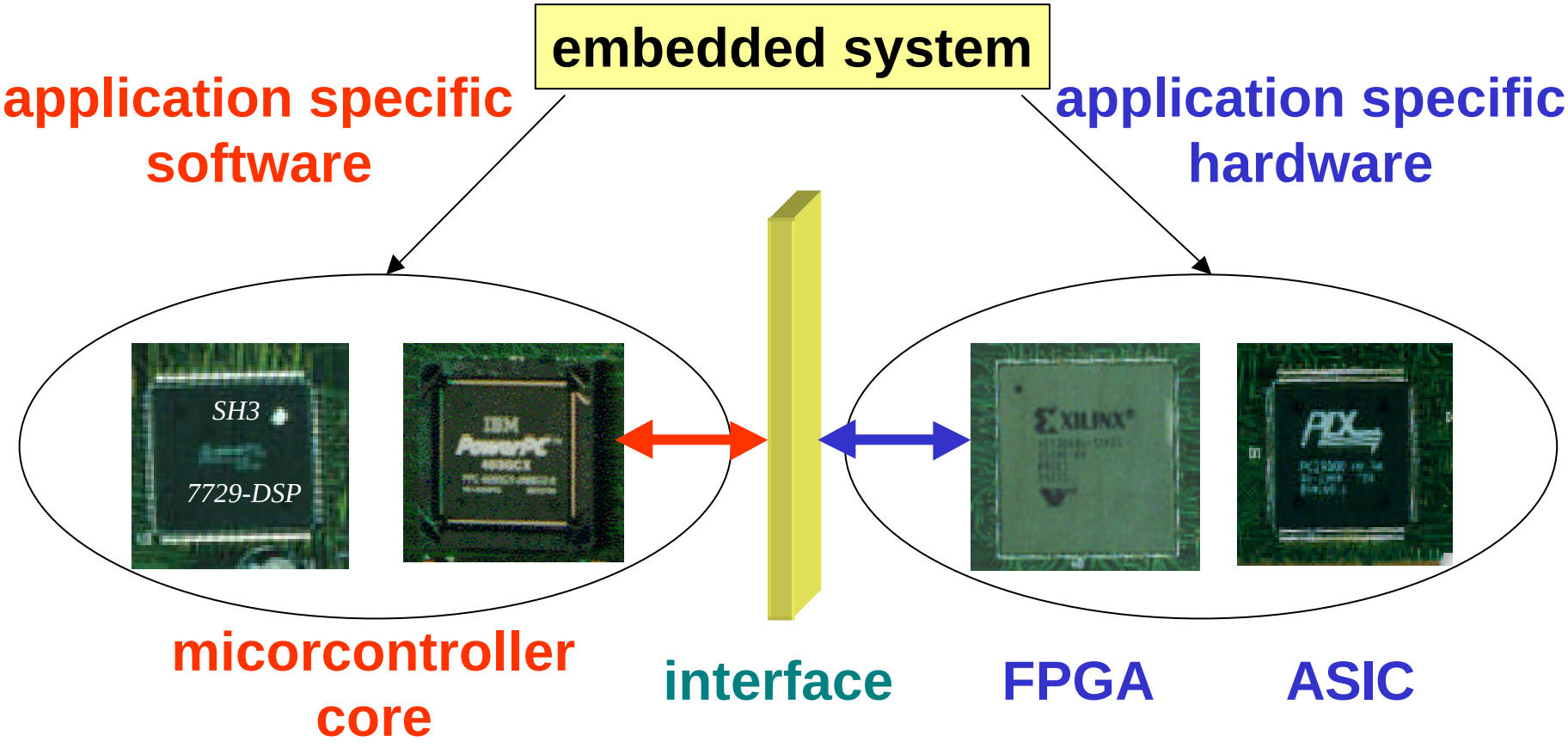




dware

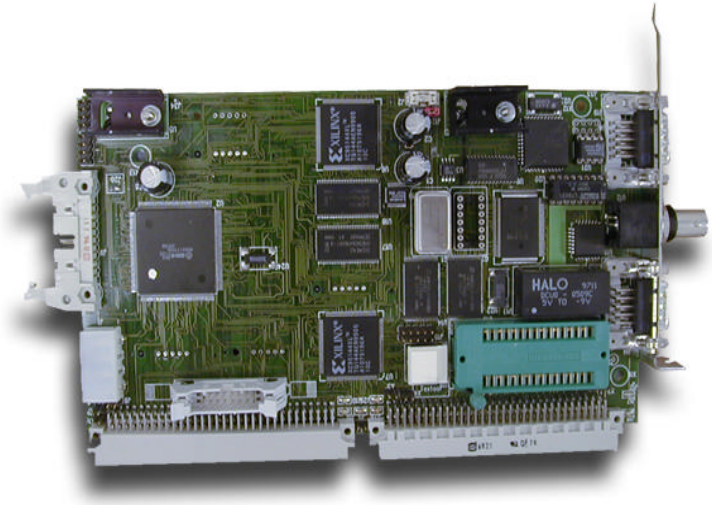


Design Methodik

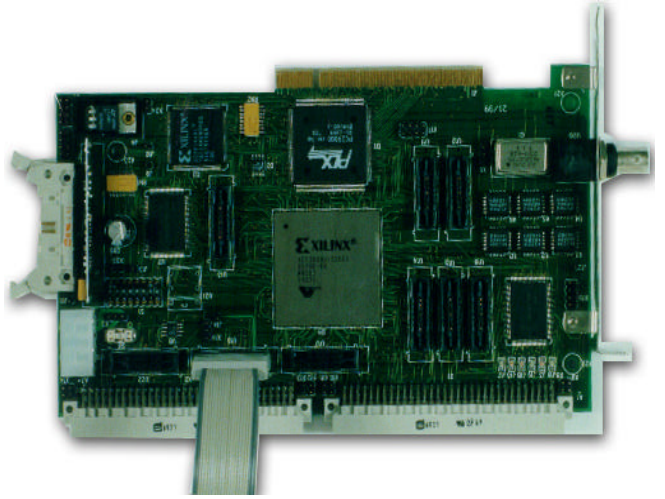


SPYDER-System: Second Generation

- **PCI & LAN-based add-on boards (1999)**
  - **SPYDER-CORE-P2/SH3-7709A/7729-DSP (Hitachi)**
  - **SPYDER-Virtex-X2/XCV300..1000 (XILINX)**

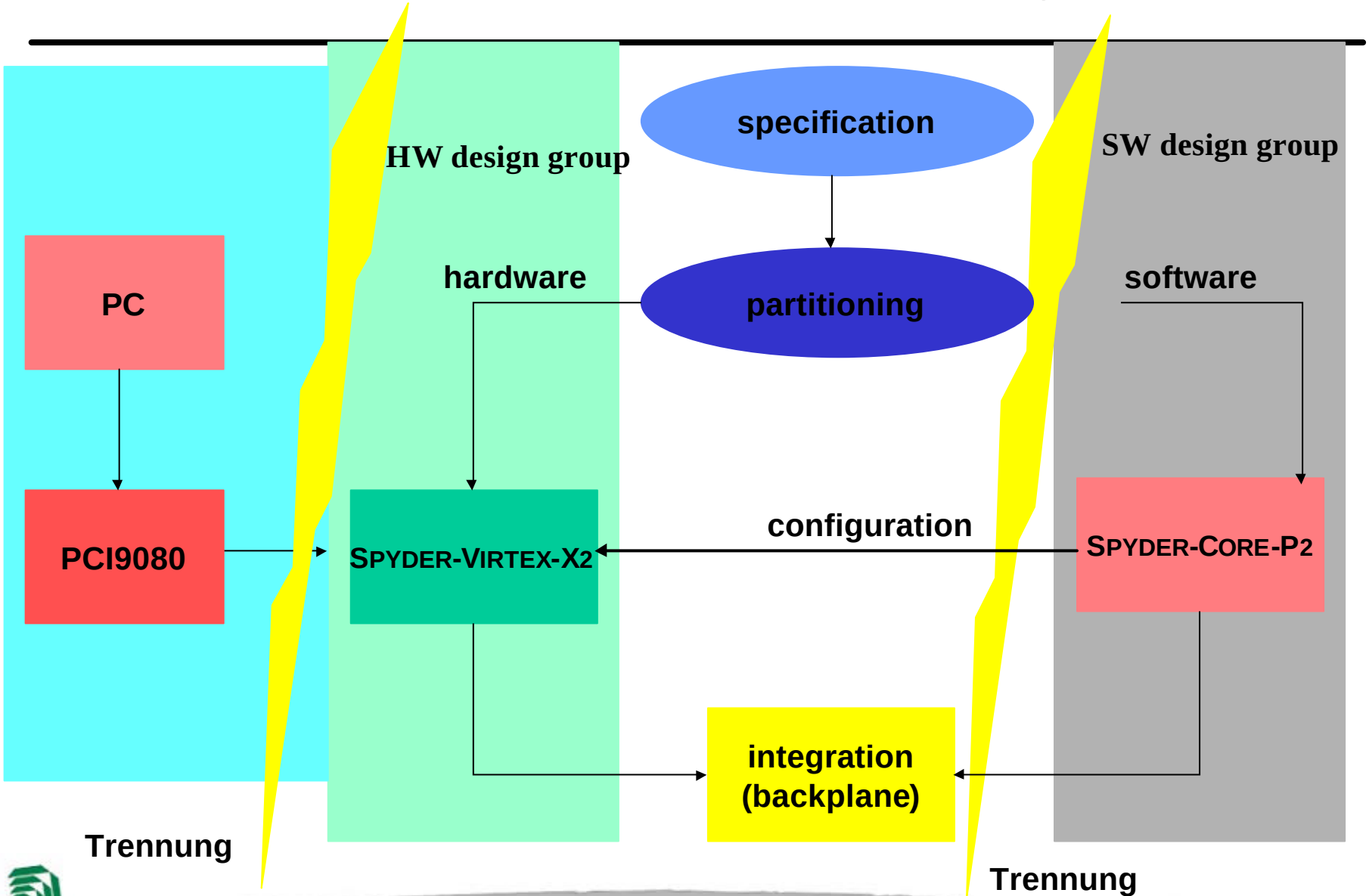


SPYDER-CORE-P2/SH3-7709/7729-DSP

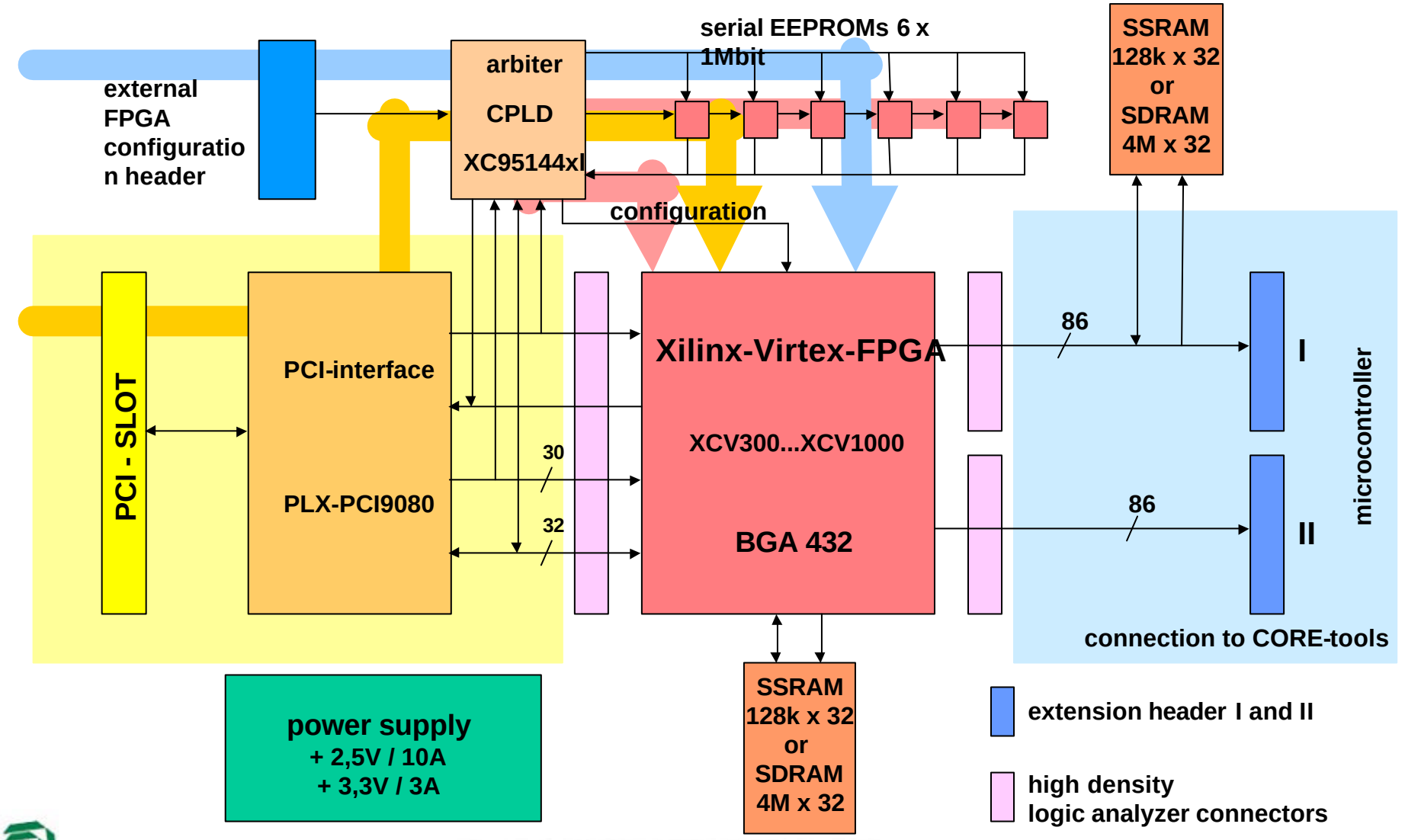


SPYDER-Virtex-X2/XCV300..1000

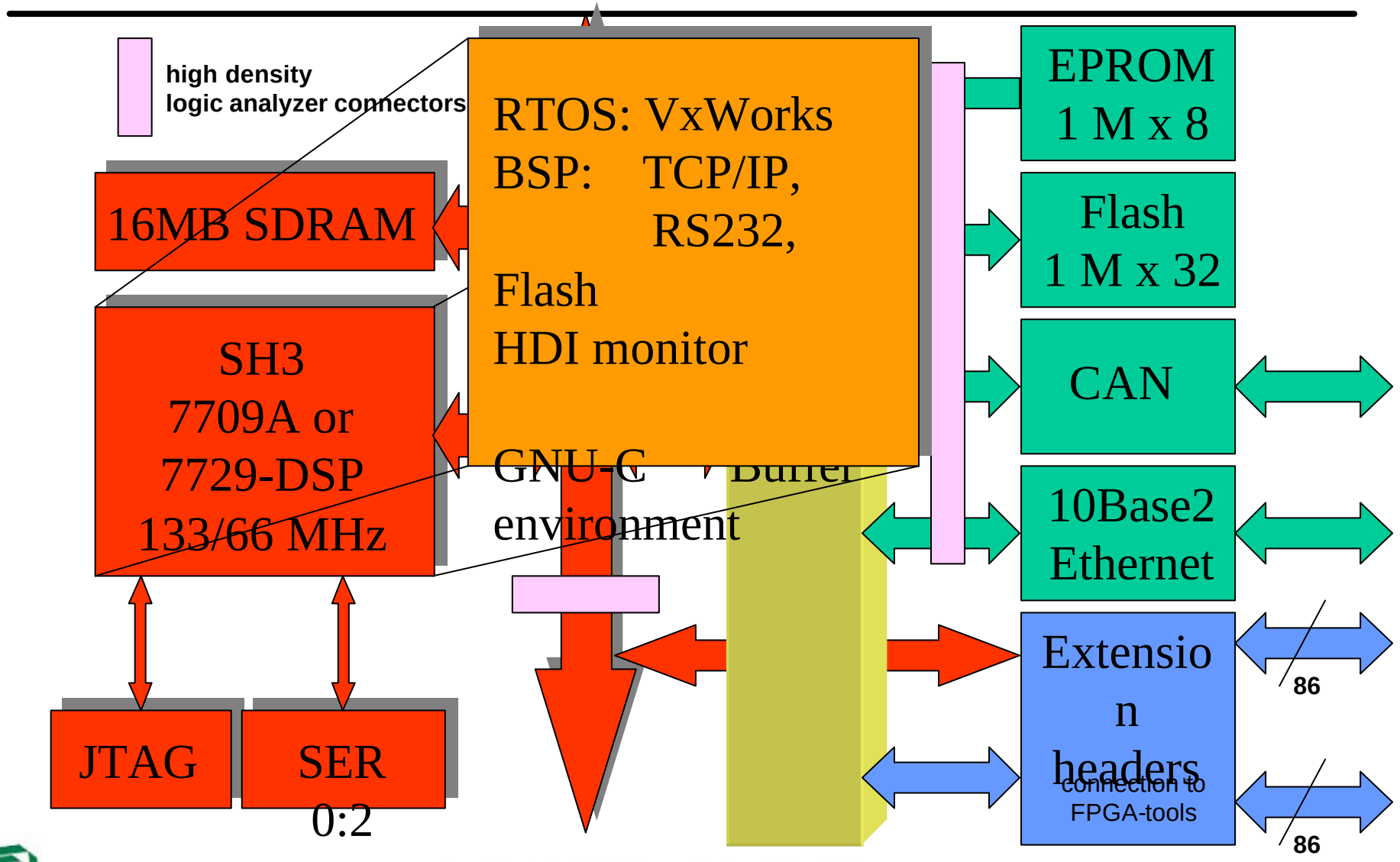
SPYDER-System: Paralleler Design Flow



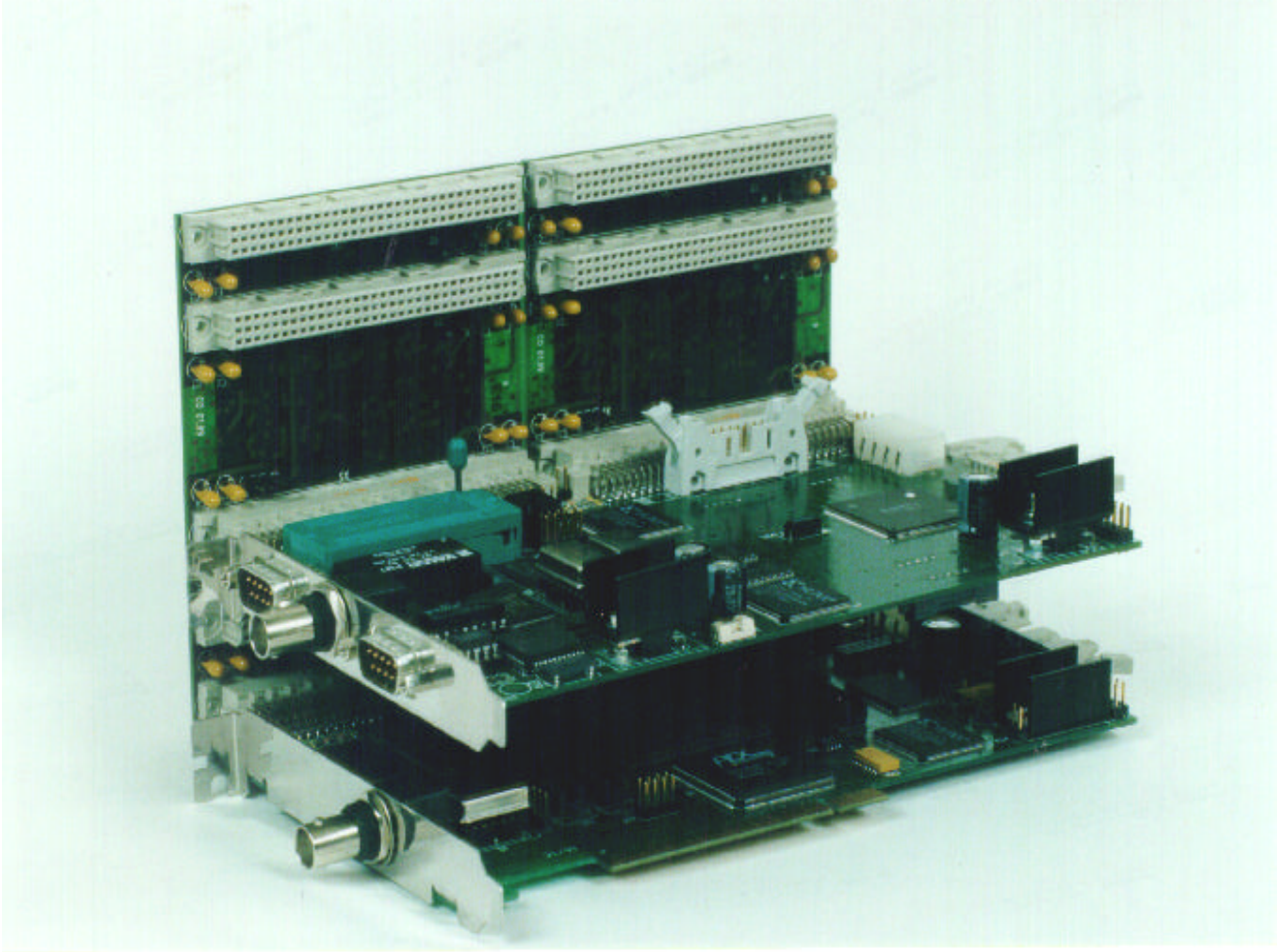
SPYDER-Virtex-X2: Architektur



SPYDER-CORE-P2/SH3: Architektur



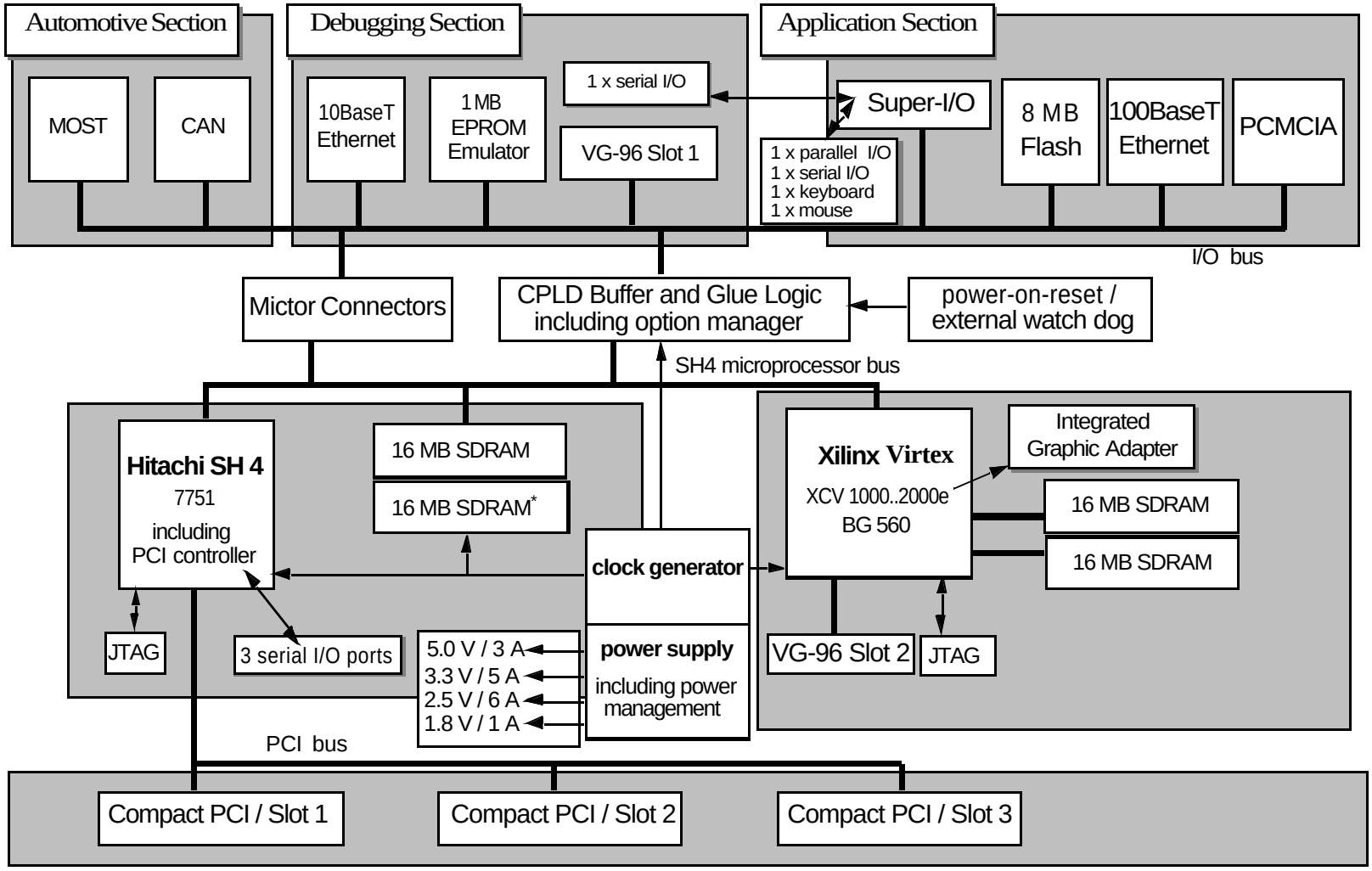
SPYDER-System: HW/SW Integration



Aktuelle und zukünftige Arbeiten

- **Automotive high-end Anwendung auf Basis SH3-7709A und RTOS VxWorks**
- **VxWorks-BSP auf Basis von SPYDER-CORE-P2/SH3**
- **zukünftige Entwicklung des SPYDER-Systems**
  - **Hitachi SH4-7751-PCI embedded contr.**
  - **Xilinx Virtex-E mit bis zu 2.Mio gates/chip**
  - **LAN-based**
  - **RTOS: full VxWorks-BSP support**
  - **Windows-CE**

SPYDER-CORE-SH4



*) optional

Eingebettete Systeme im Internet







Anzeigen von Web-Inhalten
(Geräte für den Verbraucher)
Embedded Web Browser

Geräteschnittstelle und -verwaltung
(Industrielle Anwendungen)
Embedded Web Server

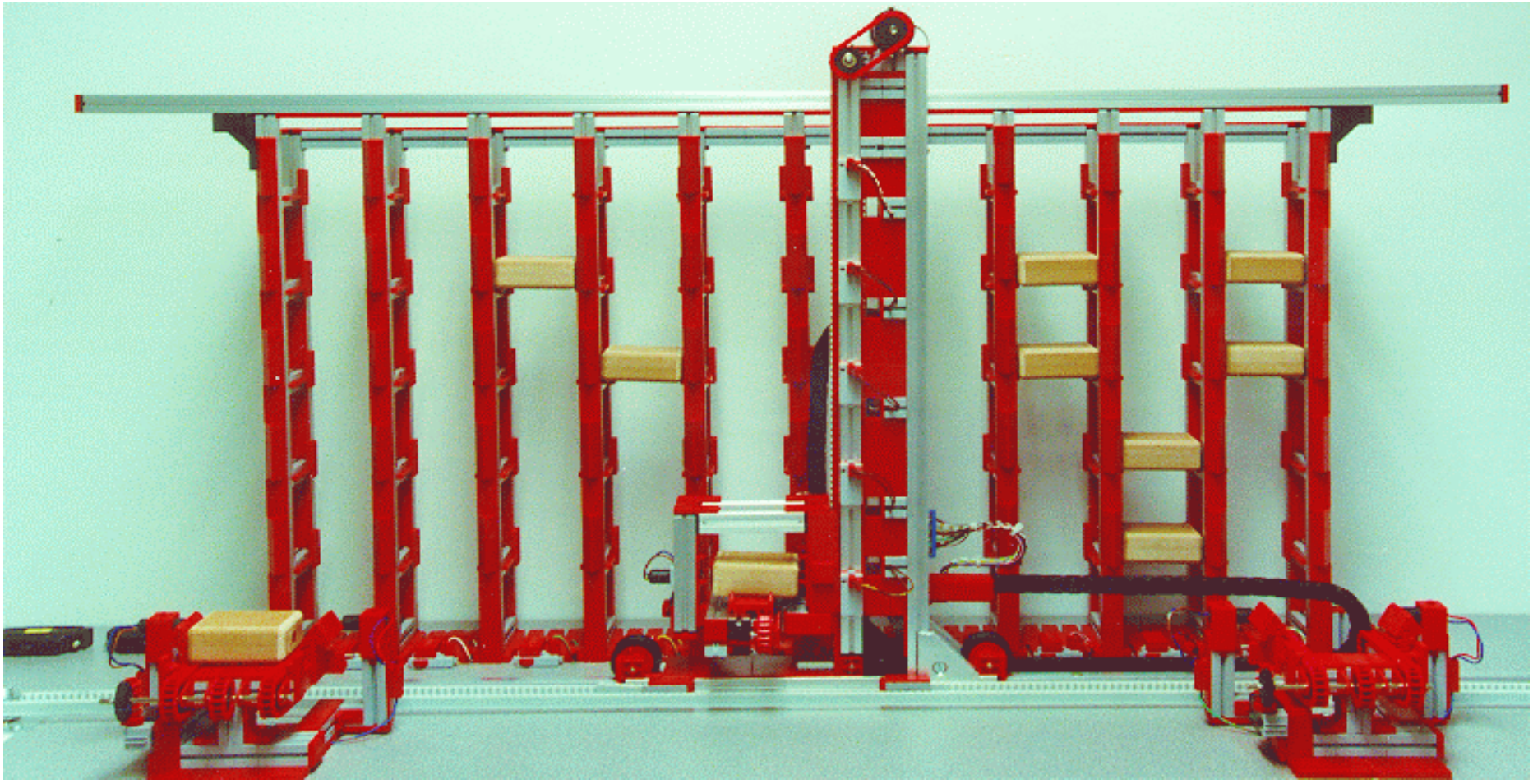
Vernetzungen: Embedded Systems und Internet

Geräte	*Beispiel-Anwendungen*	*Kommuni-kation*	*Firmen*
Kommuni-kation	PDAs, Funktelefone, Pager	Thin client	Nokia
Automobil-Elektronik	Armaturenbrett (GPS und Verkehrsmeldungen) Rücksitz-Clients (Unterhaltung) On-board Web-Server (Instandhaltung und Diagnose)	Fat client Fat client Fat server	Daimler Benz, BMW, etc. GM/Sun/ Netscape/ Microsoft (Auto PC)
Haus-Anwendungen	Fernseher und Set Top Box Bildschirm-Telefone Verbraucher-NCs „weiße“ Haus-Anwendungen (Diagnose, Management und Zusammenarbeit)	Fat Client Thin client Thin client Thin server (eingeschränkte Displays und Speichergrößen)	Microsoft/ WebTV InfoGear

...Vernetzungen: Embedded Systems und Internet

Geräte	*Beispiel-Anwendungen*	*Kommuni-kation*	*Firmen*
Bürosysteme	Kopierer Drucker Faxgeräte (Gerätekonfiguration und Systemmanagement)	Thin server Thin server Thin server (eingeschrkt. Displays und Speichergr.)	Hewlett-Packard,. Tektronix
Telekommu-nikation	Switch, Netzwerk-Kommunikations-Server etc.	Thin server	Cisco
Messen und Testen	Test- und Meß-Instrumentierung Netzwerk-Kameras etc.	Thin servers/clients IP Multicast	Hewlett-Packard Axis Communi-cations, HP

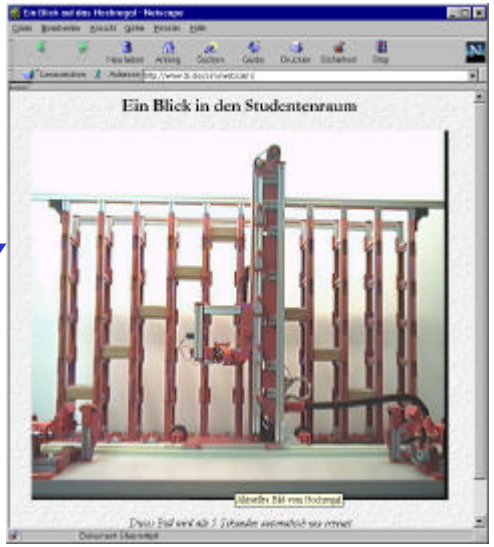
Eingebettete Systeme im Internet (Beispiel)



Ansteuerung eines Hochregallagers

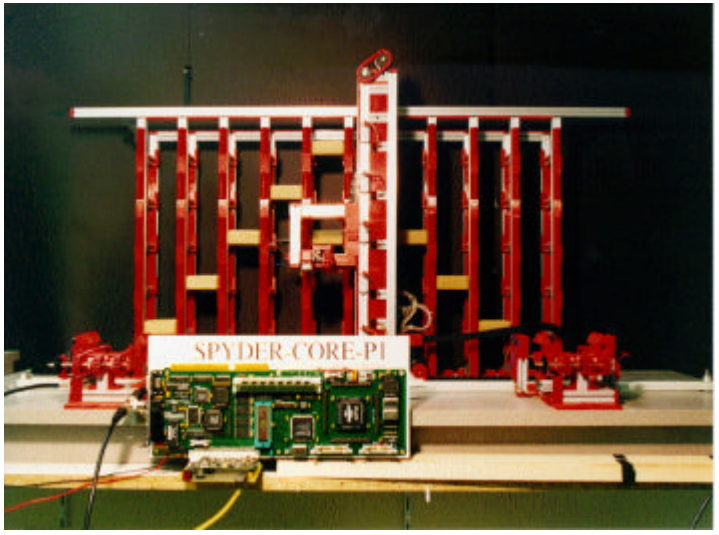


Browser mit Client-Applet



Browser zur Überwachung

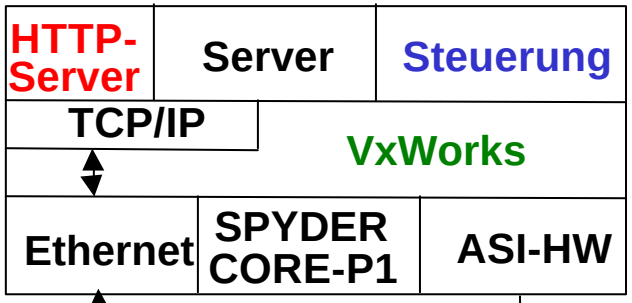
Hochregal-lager



Implementierung der Ansteuerung

Anwendungsspezifisches Client-Applet



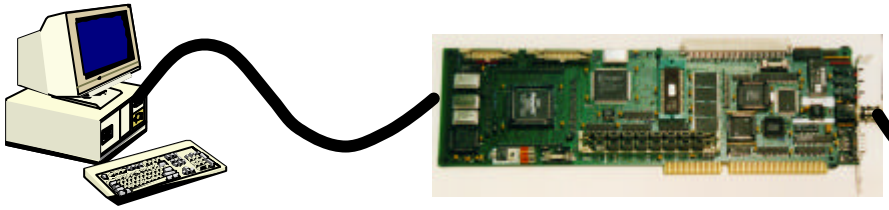


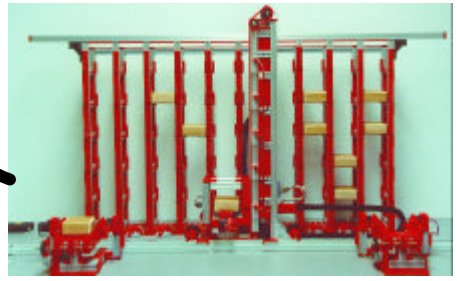
TCP/IP

ASI - Kabel

ASI-Slaves

Sensor
Aktuator





Vorteile der Internet-Technologie für ES

- **Geringe Kosten (kein proprietäres GUI notwendig)**
- **Bekannte Umgebung für den Anwender**
- **Verbesserter Zugriff auf das System**
- **Einfache Produktanpassung**

Vorteile von Java

- **Portabel**
- **Sauberes OO-Konzept**
- **Sicher**
- **Internet-fähig**
- **Dynamisch**

Probleme der Java-Technologie

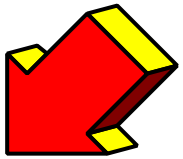
- **Performance**
- **Speicherplatzverbrauch**
- **Vorhersagbare Ausführungszeiten**
- **Fehlender Zugriff auf Adressen**

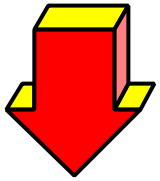
Problempunkt: Java-Performance

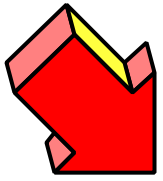
- **Optimierungsmöglichkeiten bei der Anwendungsentwicklung:**
  - **Java Compiler**
  - **Romizer**
- **Optimierungsmöglichkeiten zur Laufzeit:**
  - **Just in Time Compiler**
  - **Unterstützung durch Hardware**

Problempunkt: Speicherplatzverbrauch

Anpassen der Java-Plattform an die Anwendung







PersonalJava

- weiche Echtzeitanforderungen
- Internetverbindung
- Grafikunterstützung

EmbeddedJava

- harte Echtzeitanforderungen
- keine Internetverbindung
- keine Grafikunterstützung

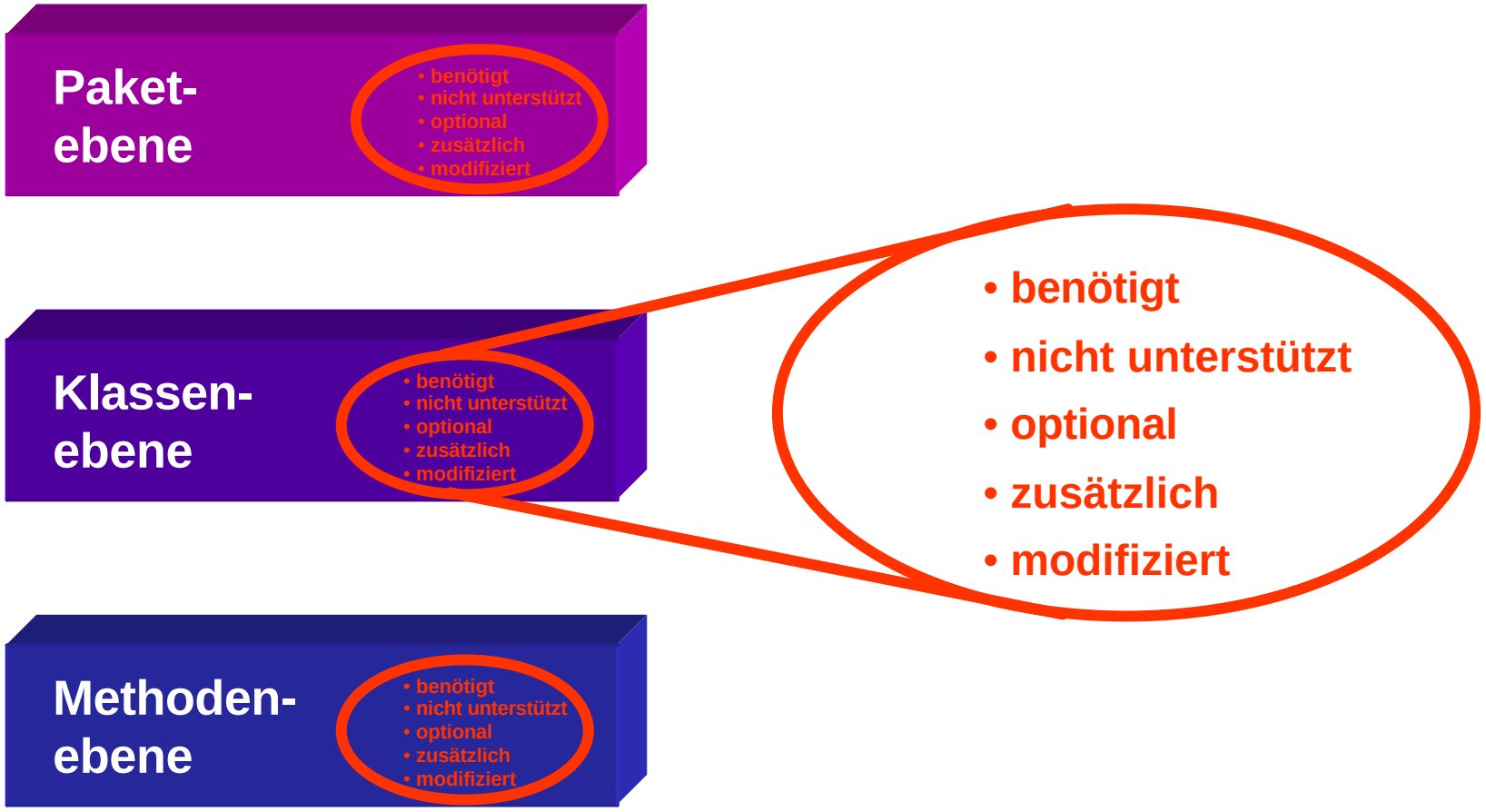
Java Card

Javafähige Chipkarten

PersonalJava-Konzepte

- **Anwendungsentwicklung und -test auf einem Hostsystem (PC oder Workstation)**
- **Anpassung für die spezifische Infrastruktur der Zielplattform**
- **Speicherplatzbedarf der Classfiles wurde reduziert**

Anpassung von PersonalJava



PersonalJava Speicherplatzverbrauch

	PersonalJava 1.1 Anwendungsumgebung	*Java 1.1.6 Anwendungsumgebung*
Anzahl der Klassen	**913**	**911**
Gesamtgröße der Classfiles	**1.997MB**	**2.022MB**
Gesamtgröße des Speicherplatzverbrauchs	**1.593MB**	**2.222MB**

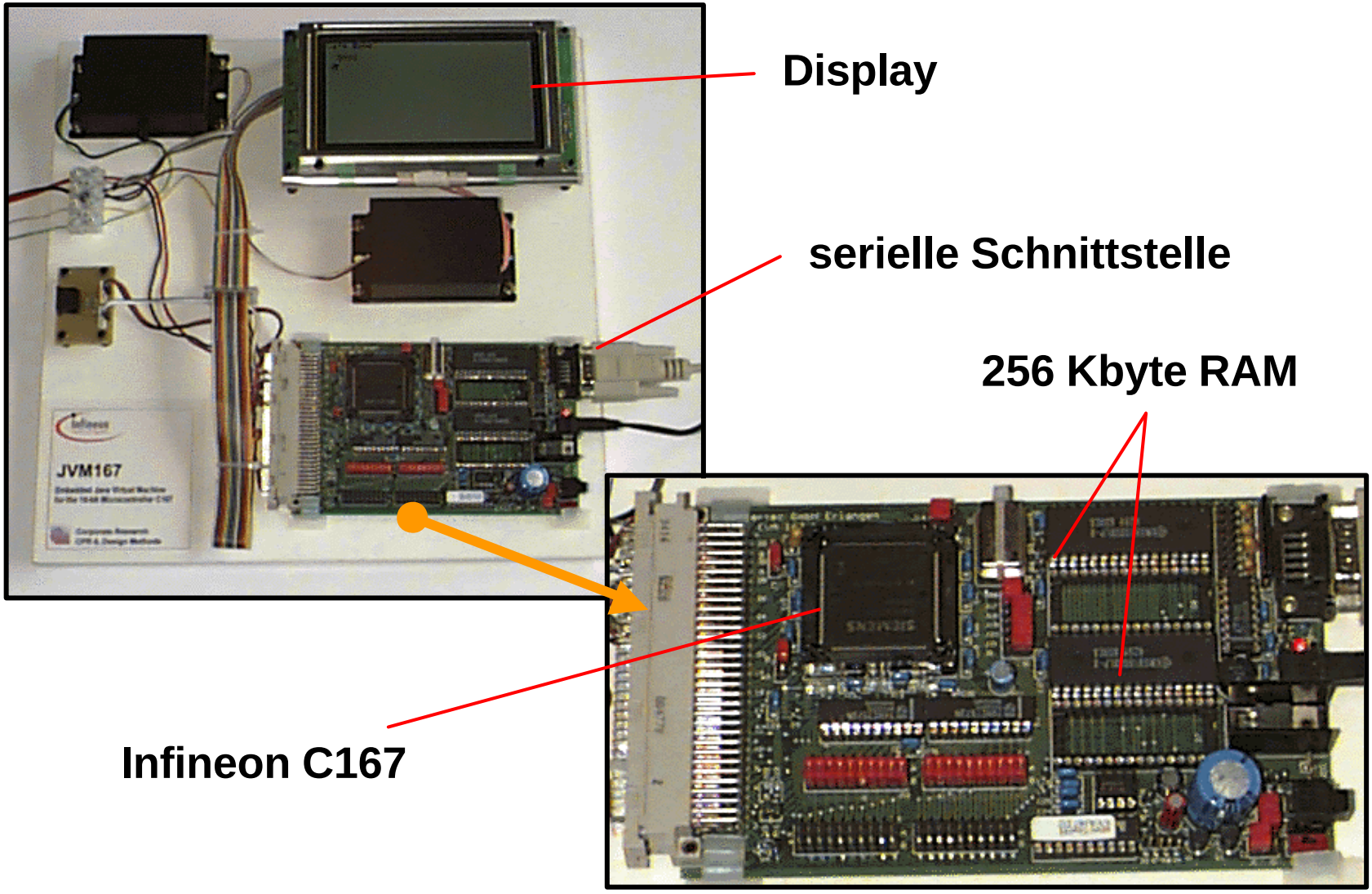
Quelle: Sun Microsystems, Inc.

EmbeddedJava-Konzepte

Für stark eingeschränkte eingebettete Systeme:

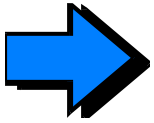
- **Vorhersagbare Ausführungszeiten**
- **Stark eingeschränkte Systemressourcen (RAM/ROM)**
- **Härtere Performance-Anforderungen**
- **Keine oder eingeschränkte Grafikausgabe**

Anwendungsbeispiel



EmbeddedJava - Optimierungsstrategien

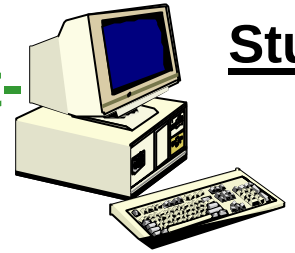
Problem	*Lösung*
Performance	Ausführung des Bytecodes aus dem ROM heraus
Speicherplatz-verbrauch	``Omission is the better part of compression´´



Ersetzen des „*load-link-verify*”-Zyklusses

Performance-Steigerung durch Romizer

Hostplatt-form



Stufe 1: Verwendung der EmbeddedJava-Entwicklungstools von Sun:
JavaFilter, JavaCodeCompact und JavaDataCompact

➡ plattformunabhängig

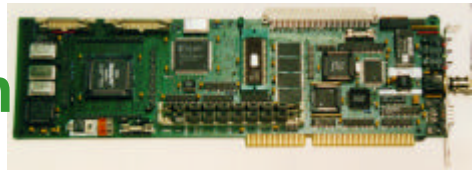
Hostplattform



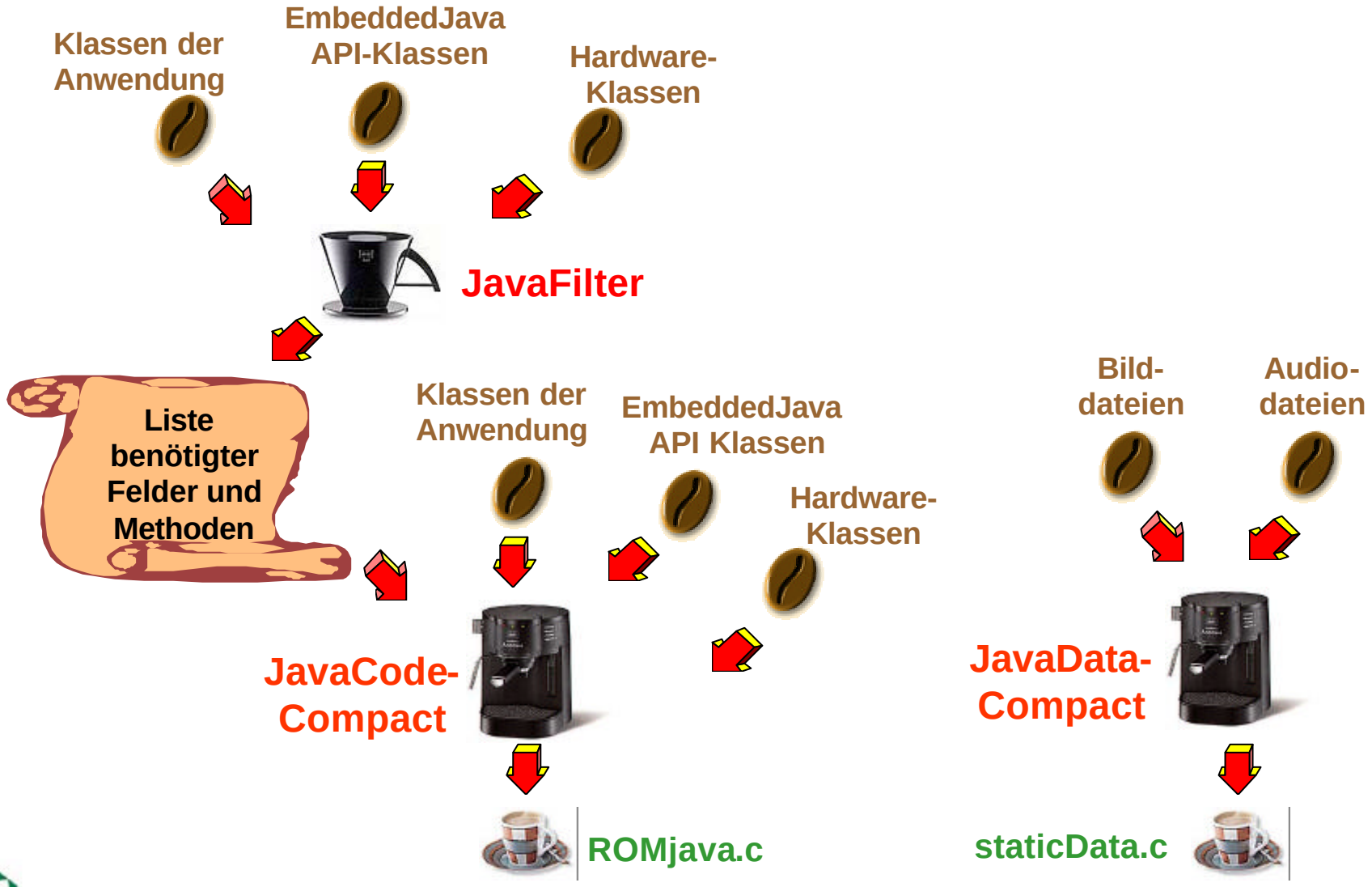
Stufe 2: Entwicklungstools für die Zielplattform:
Compiler und Linker

➡ plattformabhängig

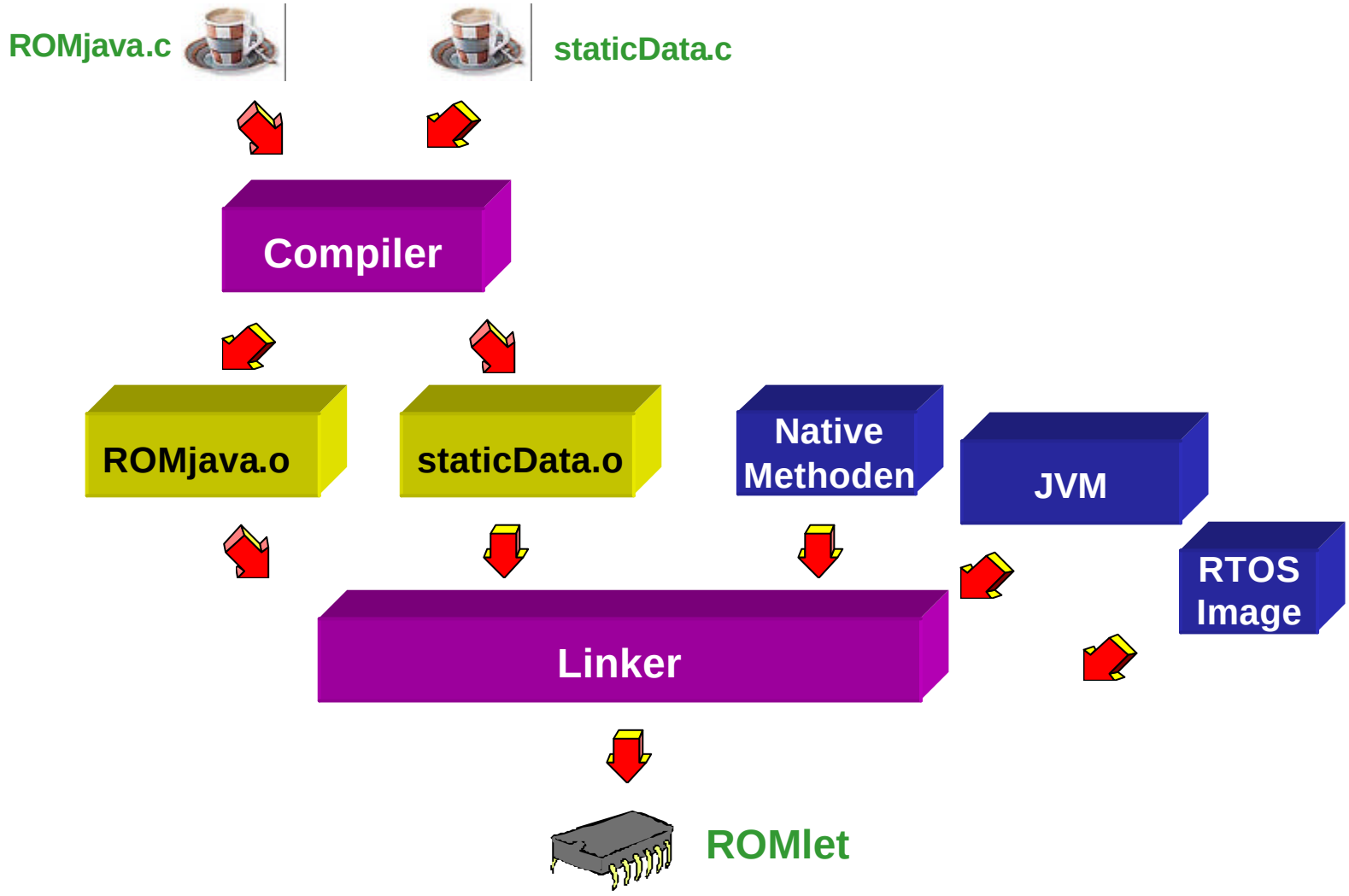
Zielplattform



EmbeddedJava Entwicklungstools (Stufe 1)



Erzeugung des ROMlet‘s (Stufe 2)



EmbeddedJava - Software-Updates

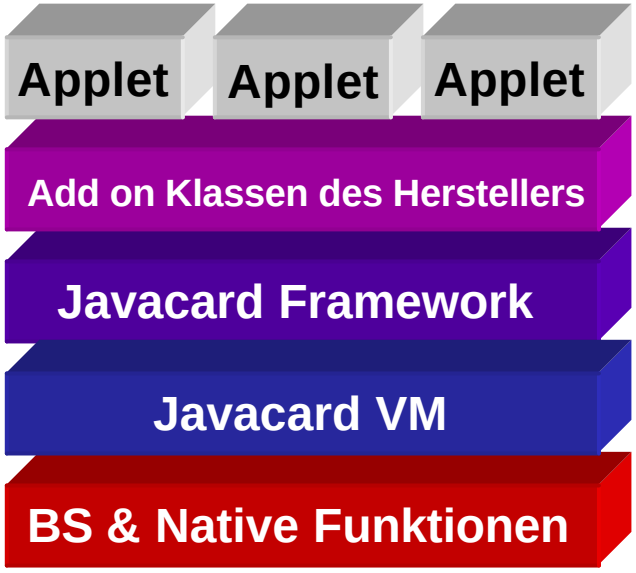
- Updates im Flash ROM
- Erstes Patchlet zeigt ins ROM
- Update-Patchlet zeigt auf die aktuellen Klassen
- Updates werden mit Java Mix erzeugt



JavaCard Konzepte

Philosophie: JavaCard = Java-fähige Smart Card

- einheitliche Sicht des Systems
- mehrere Anwendungen möglich (Applets)
- dynamisches Laden von Anwendungen
- objektorientierte Anwendungsentwicklung



Systemanforderungen

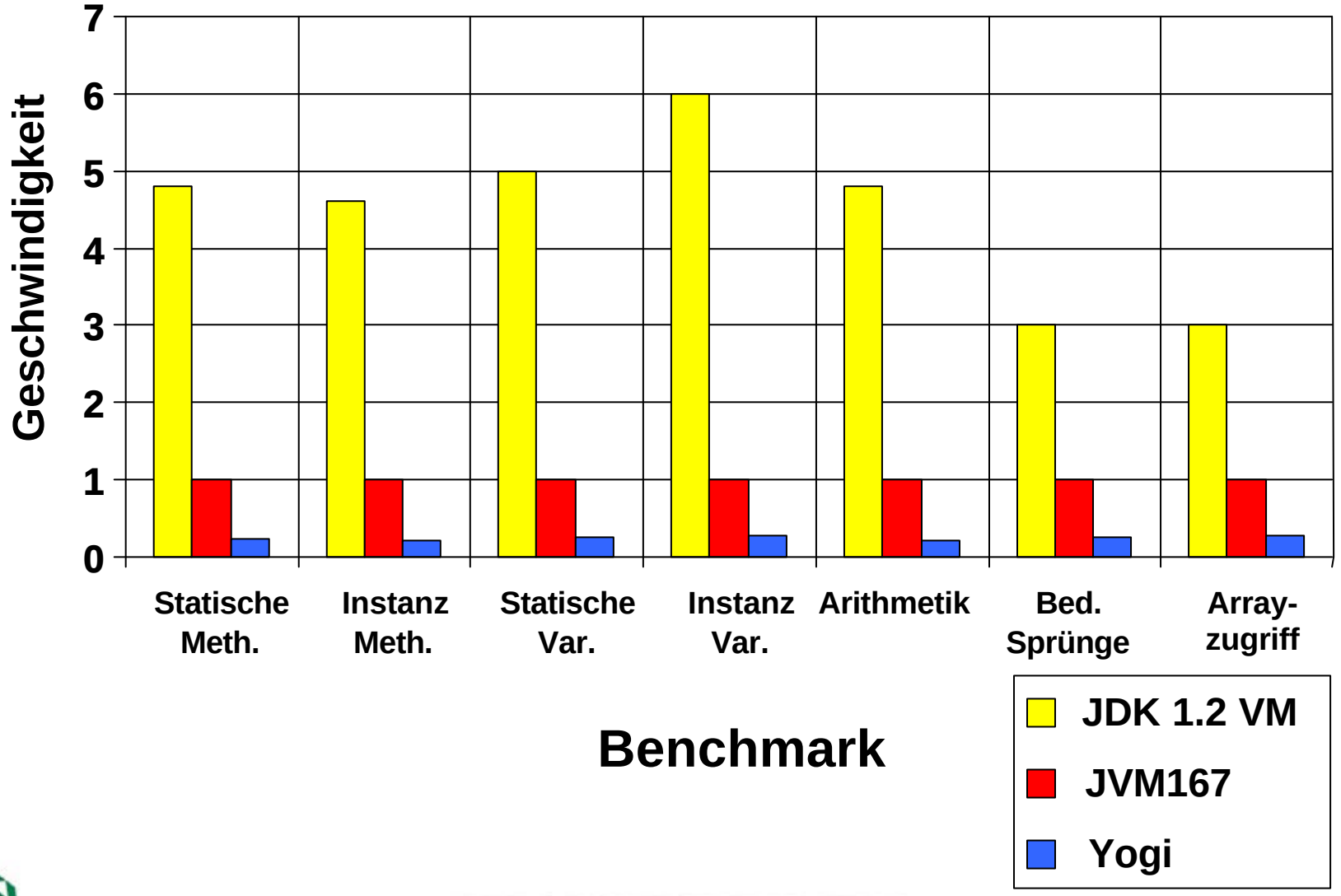
	Java JDK	PersonalJava	EmbeddedJava	JavaCard
Größe des RAM	> 4 MByte	< 1 MByte	< 512 KByte	512 Bytes
Größe des ROM	4-8 MByte	< 2 MByte	< 512 KByte	16 Kbyte
CPU Typ	100 MHz+	50 MHz+	25 MHz+	300 KIB

Quelle: JavaSoft

Vergleich von virtuellen Java Maschinen

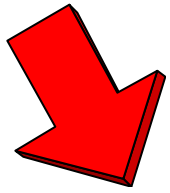
	JDK 1.2 VM	JVM167	Yogi
Architektur	Pentium II	C167	ST7
Takt	400,0 MHz	20,0 MHz	4,3 MHz
CPU Wortbreite	32 Bit	16 Bit	8 Bit
VM Wortbreite	32 Bit	16 Bit	16 Bit
Größe der Zeiger	32 Bit	32 Bit	16 Bit
Programmier-sprache	C	C	Assembler

Benchmark-Ergebnisse



Probleme von PJ/EJ-Plattformen

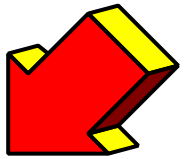
- **Firmen definieren eigene Umgebungen**
- **Code von Drittanbietern unter Umständen nicht einsetzbar**
- **Zielplattformen sind zu unterschiedlich**

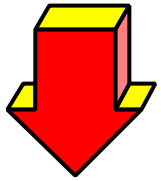


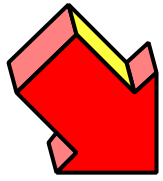
”One size doesn‘t fit all !”

Problempunkt: Speicherplatzverbrauch (alter Ansatz)

Anpassen der Java Plattform an die Anwendung







PersonalJava

- weiche Echtzeit-anforderungen
- Internetverbindung
- Grafikunterstützung

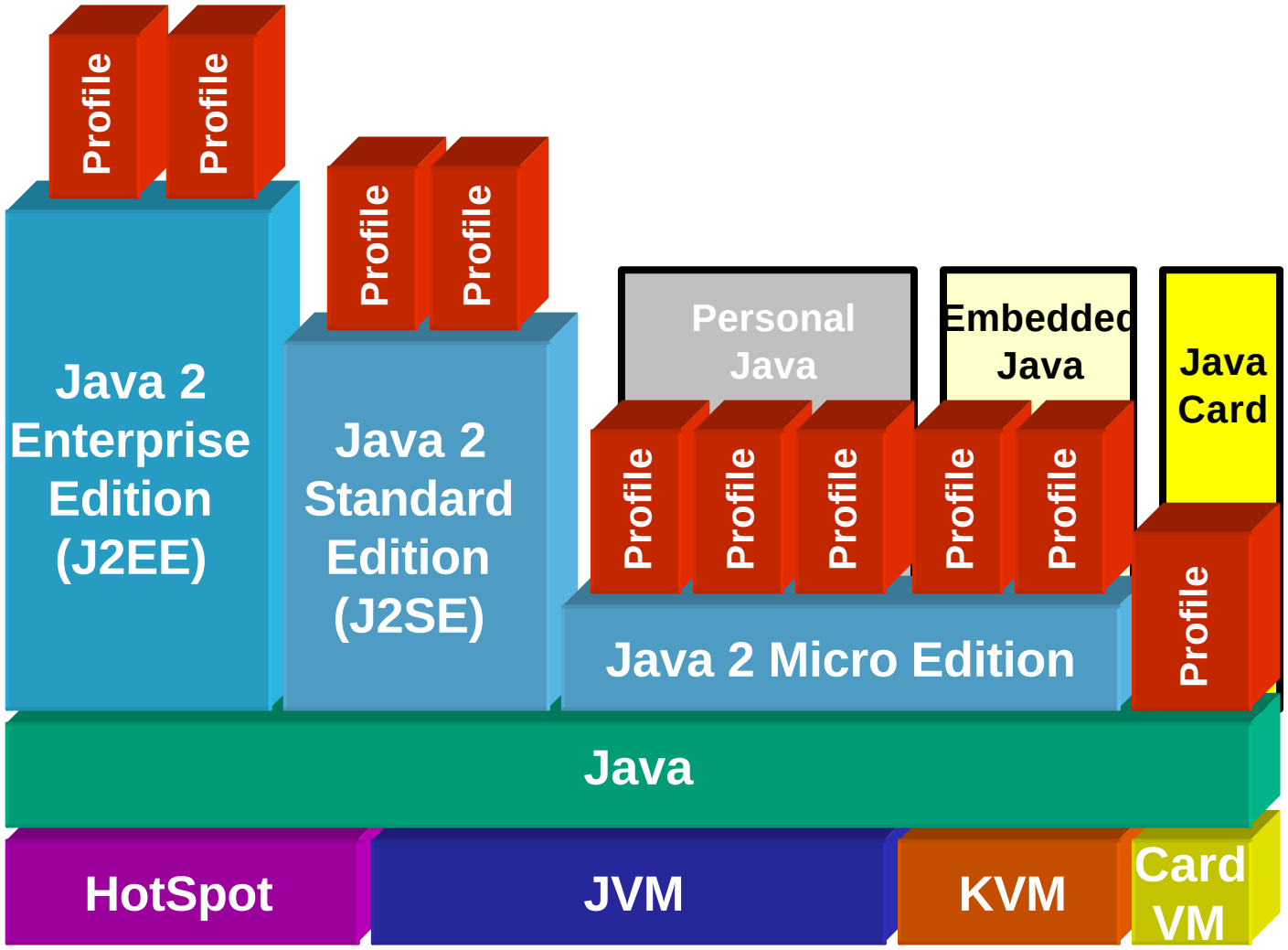
EmbeddedJava

- harte Echtzeit-anforderungen
- keine Internetverbindung
- keine Grafikunterstützung

Java Card

Javafähige Chipkarten

Java 2 Plattformen



Java 2 Micro Edition

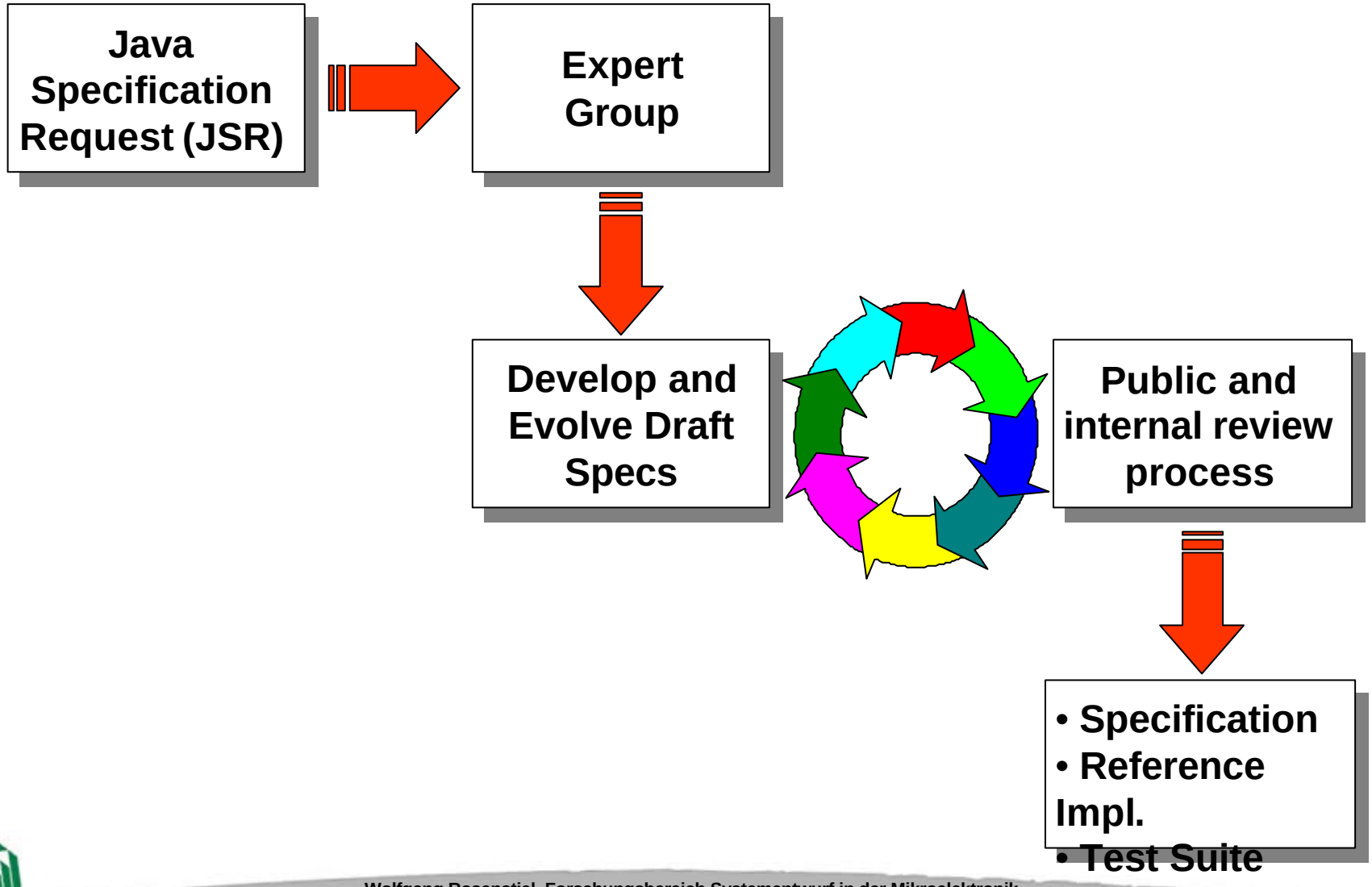
- Adressiert elektronische Massenprodukte
- Eingeschränkter Speicherplatz
- Verteilen von Funktionalität
- Hochoptimierte Laufzeitumgebung

  ⇨ KVM
- Jini Unterstützung
- Referenzplattform Palm V

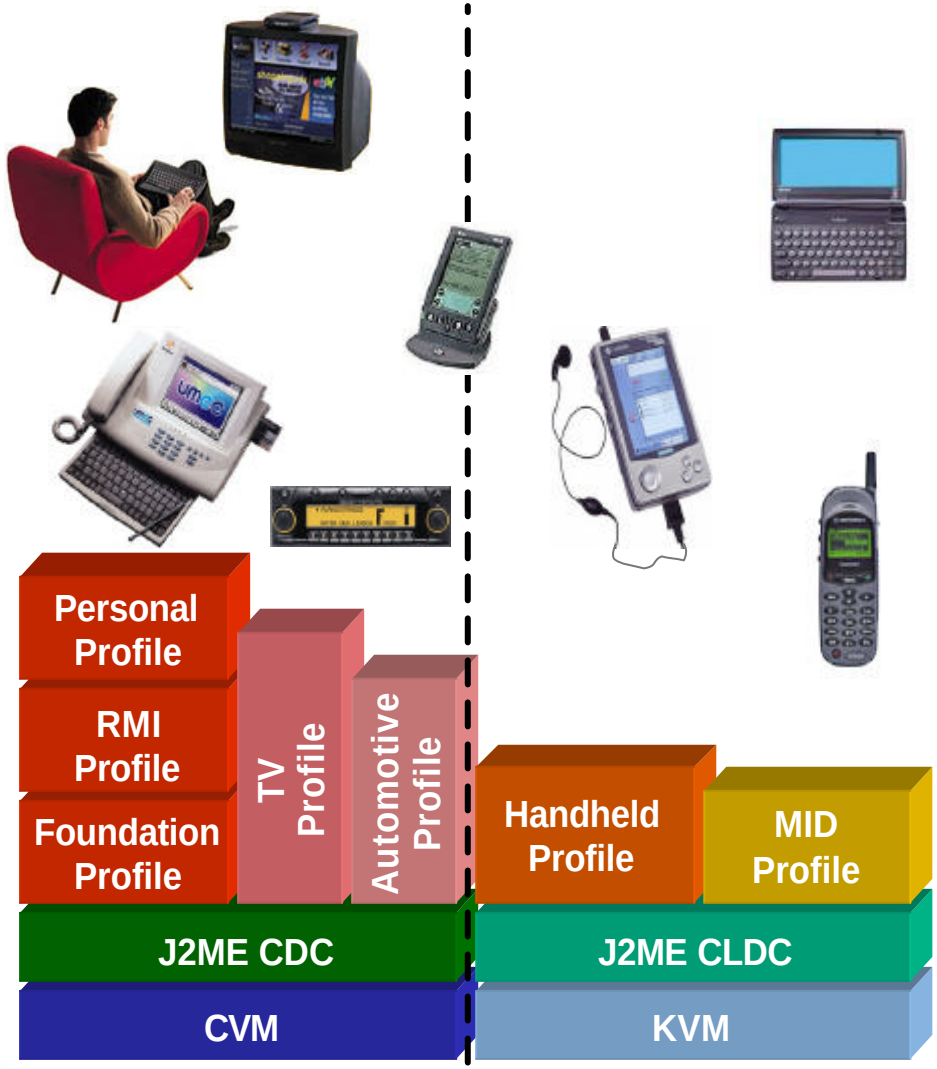
Profile

- Menge von API‘s für jede Edition
- Adressieren den vertikalen Markt
- Java Community Process für neue Profile
- Bsp:
  - Profil für drahtlose Telefone
  - PDA Profil
  - Profil für die Automobilindustrie

Java Community Process



Java 2 Micro Edition (J2ME)



J2ME CLDC

- Unterstützung der Java Sprachspezifikation ist eingeschränkt
- Funktionalität der JVM ist eingeschränkt
- Sehr eingeschränktes API (`lang`, `io`, `util`)

J2ME CDC

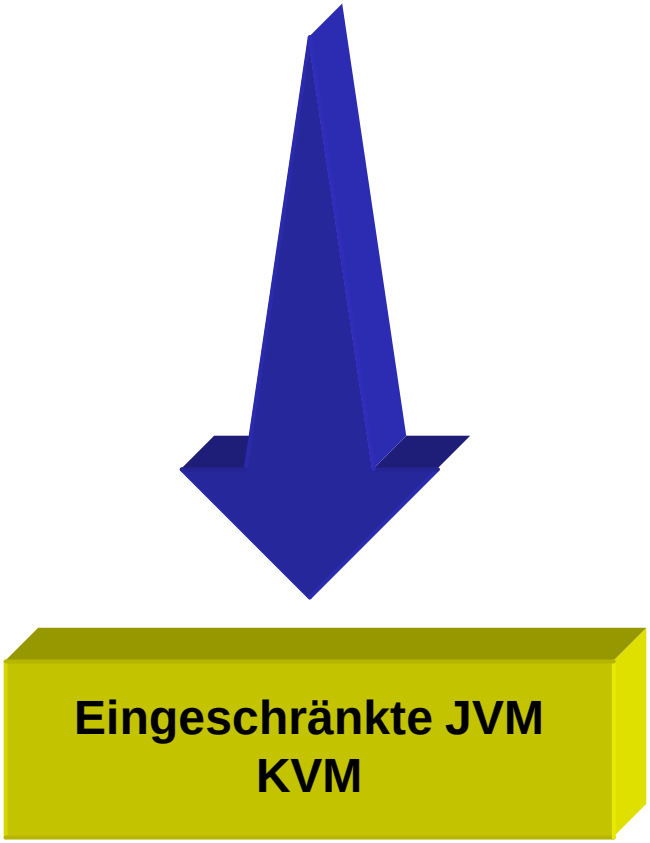
- Volle Unterstützung der Java Sprachspezifikation
- JVM mit voller Funktionalität
- JNI 1.1
- Profile für die Erweiterung in Richtung J2SE

Einschränkungen der J2ME - CLDC

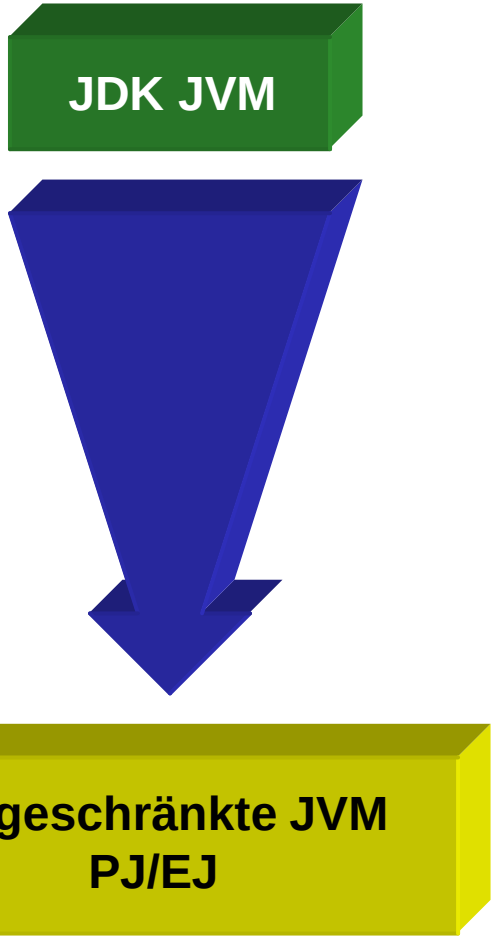
- Keine eigenen Classloader definierbar
- Keine `Float`/`Double`-Datentypen
- Eingeschränktes Thread-API
  - Klasse `ThreadGroup` fehlt
  - Methoden `interrupt()`, `stop()`, `destroy()` fehlen
- Kein herkömmlicher JNI-Support
  - JNI in KVM integriert

Entwicklung der KVM

„Blank Slate” - Ansatz



„Full Tablet” - Ansatz



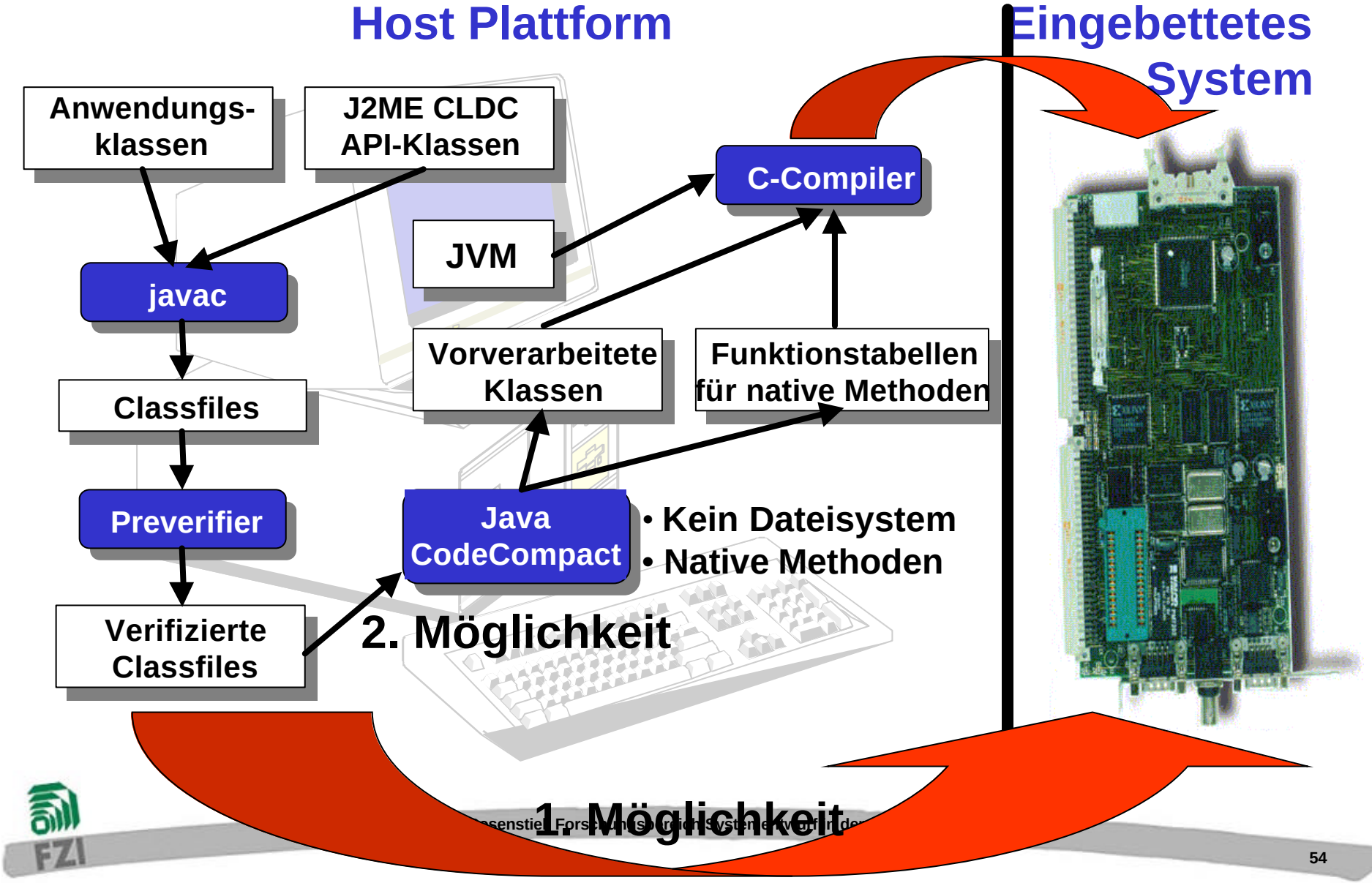
Leistungsmerkmale der KVM

- 40 KByte Laufzeitumgebung
- 128 KByte Klassenbibliotheken
- 16/32 Bit RISC/CISC-Prozessoren 16MHz+
- Entwicklung zusammen mit Motorola, 3Com, Bull, Fujitsu

Architektur der KVM

- Interpreter-basierte JVM
- Konfigurierbare Laufzeitumgebung
  - mit/ohne Threadunterstützung
  - mit/ohne `long`/`float`/`double` Datentypen
- Plattformunabhängiges Threading-System
- Kapselung plattformabhängiger Teile in einer Datei
- Einfache, nicht-kompaktierende Garbage Collection

Anwendungsentwicklung für die KVM



Beispiel einer Entwicklungsumgebung

VisualAge for ES der Firma IBM:

- Tools für die Anwendungsentwicklung
  - jxeLink (VisualAge ROMizer)
  - Remote Debugger
  - HTTP Client/Server Framework
  - Degas (Framework für GUI-Anwendungen in ES)
  - Profiler (geplant)
- Optimierte Klassenbibliothek
- Konfigurierbare JVM J9
- J9-Simulationsumgebung für Windows

J9 Virtual Machine für ES

- **Kompatibel mit der JNI-Spezifikation**
- **Reduzierter Speicherplatzverbrauch**
  - ➡ **Minimalkonfiguration + erforderliche Funktionalität**
- **Optimierte Speicherverwaltung**
  - ➡ **Speicherkompaktierung**
  - ➡ **Kontrolle der Speicherverwaltung durch Entwickler**
- **Determinismus**
  - ➡ **Verwendet Thread-System des Echtzeit-BS**
  - ➡ **Hohe Anzahl unterschiedlicher Thread-Priortäten und zuverlässiges Thread-Scheduling**
- **Abarbeitung vorverarbeiteter Klassen aus dem ROM möglich**
  - ➡ **jxeLink-Tool**

Vorteile des jxeLink Romizers

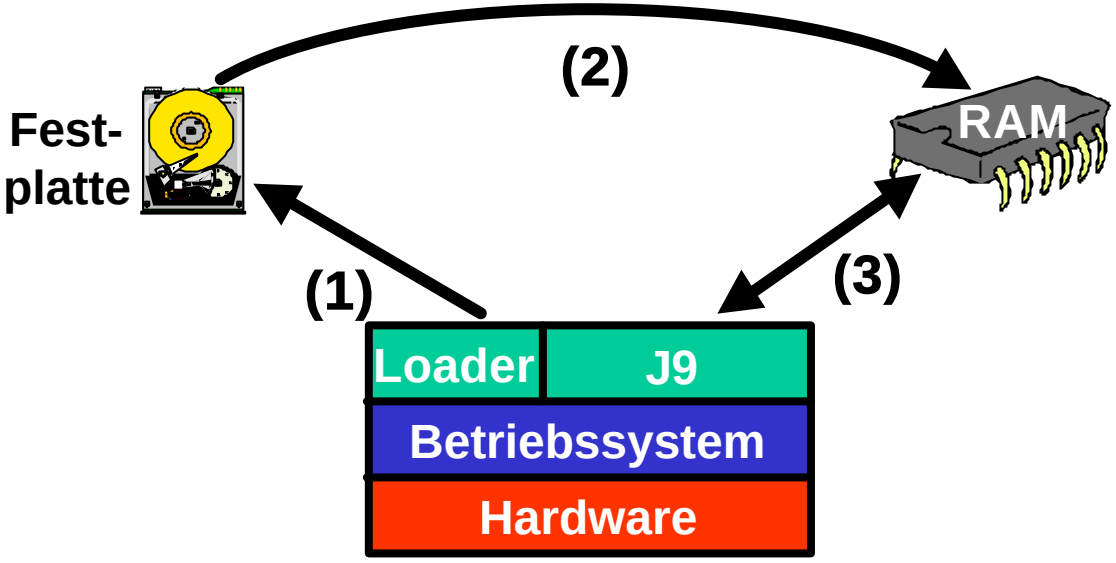
- **Reduzierte Startup-Zeit**
- **Festplatte oder Netzwerkverbindung nicht erforderlich**
- **Entfernung redundanter Datenstrukturen**
- **Entfernung von nicht verwendeten Feldern und Methoden**

Bsp.:	**Vorher: 276 Klassen mit 425k ROM und 112k RAM**
	Nachher: 230 Klassen mit 271k ROM und 78k RAM

- **Kompaktierung der JVM J9 durch Entfernung des ursprünglichen Classloaders**

Emulation der ROM-Anwendung

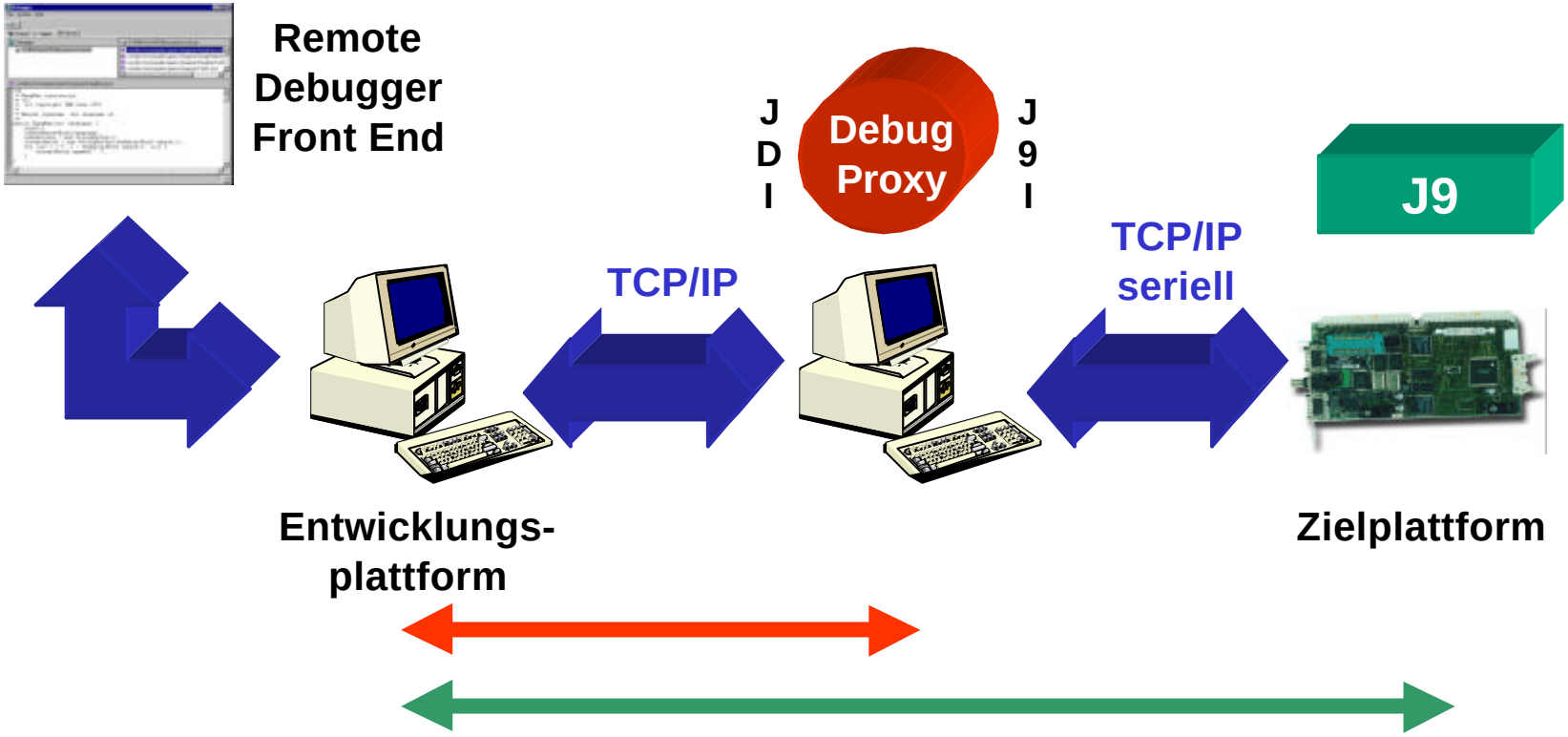
Problem: Zum Zeitpunkt der Anwendungsentwicklung muß die Abarbeitung des Codes aus dem ROM simuliert werden.



1. Spezieller Classloader für ***.jxe-Dateien
2. Kopieren der ***.jxe-Datei ins RAM
3. J9 arbeitet mit den Klassen aus dem RAM

Debugging

- Remote-Debugging auf der Zielplattform
- Debugging einer Anwendung in der J9-Simulationsumgebung



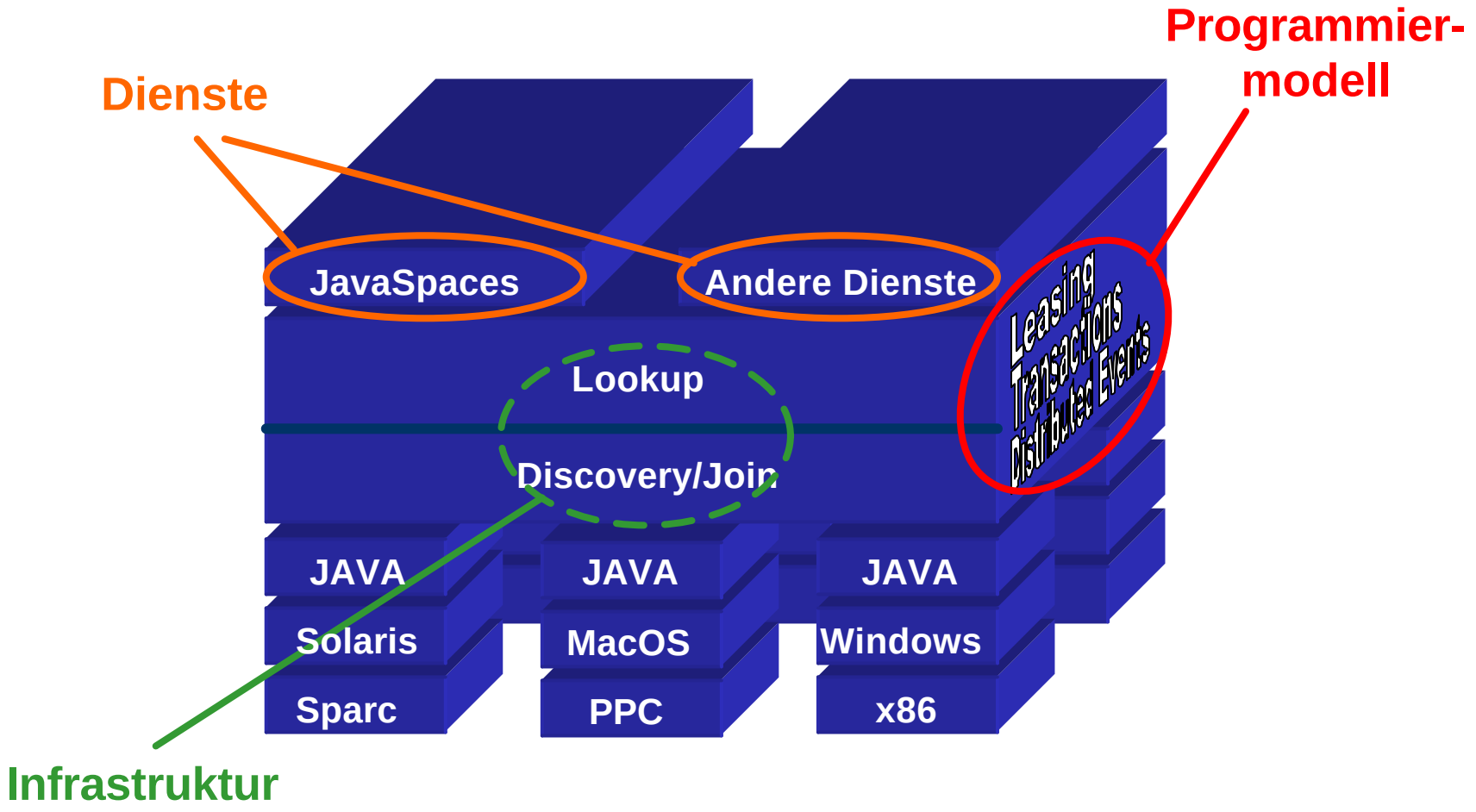
Jini - Java Intelligent Network Infrastructure



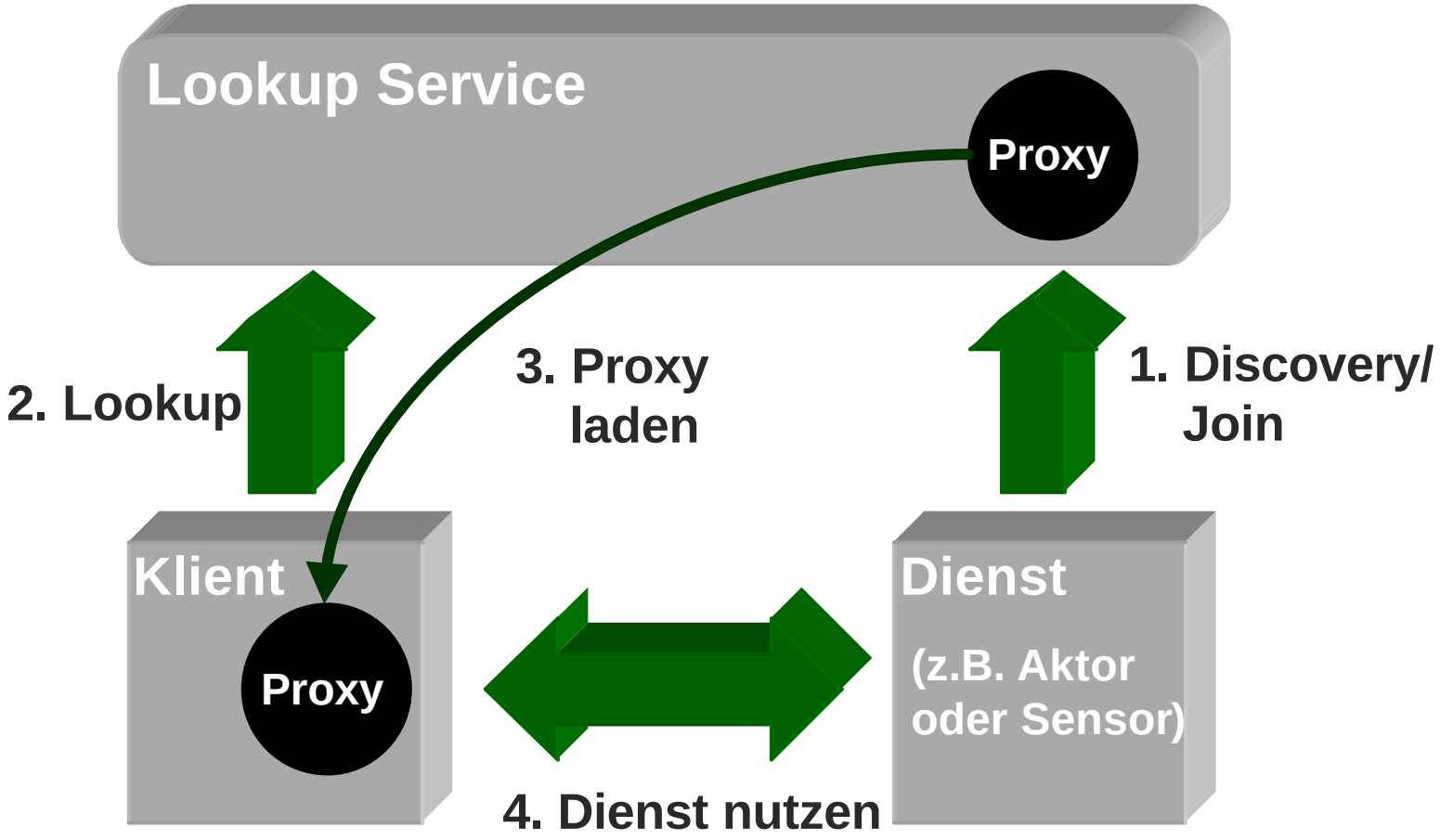
Jini-Konzepte

- **Verteilte Programmierung**
- **Leasing**
- **Verteilte Ereignisse**
- **Verteilte Transaktionen**
- **Erweitertes Sicherheitskonzept**

Jini-Architektur



Jini - Grundlagen



Vorteile einer Jini-basierten Systemarchitektur

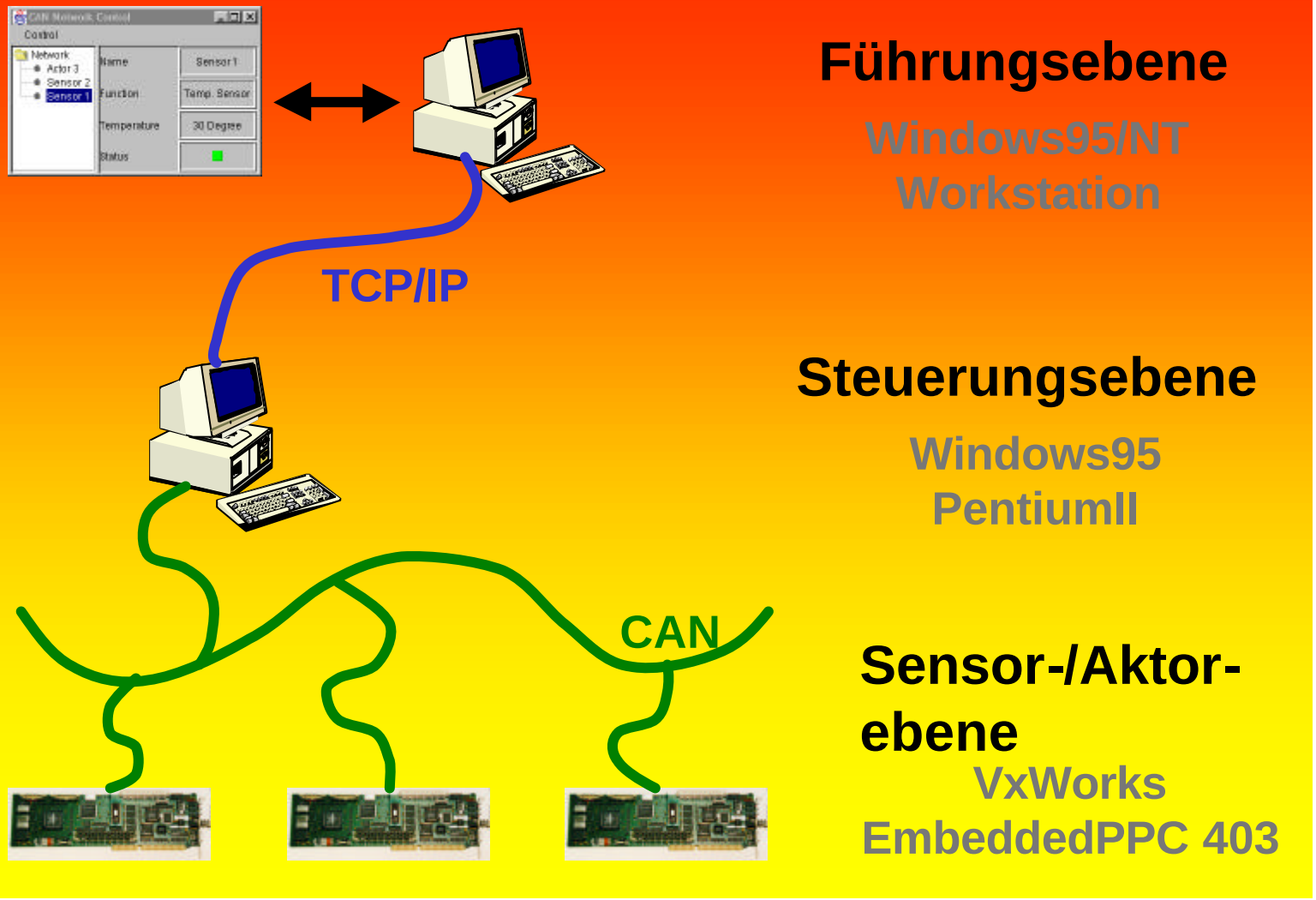
- **Plug-and-Play Technologie ermöglicht Baukastenlösungen**
- **Skalierbarkeit und Flexibilität**
- **Konzentration auf die Anwendung**
- **Komponentenbasierte Anwendungs-entwicklung**

Vergleich zwischen Jini und Java

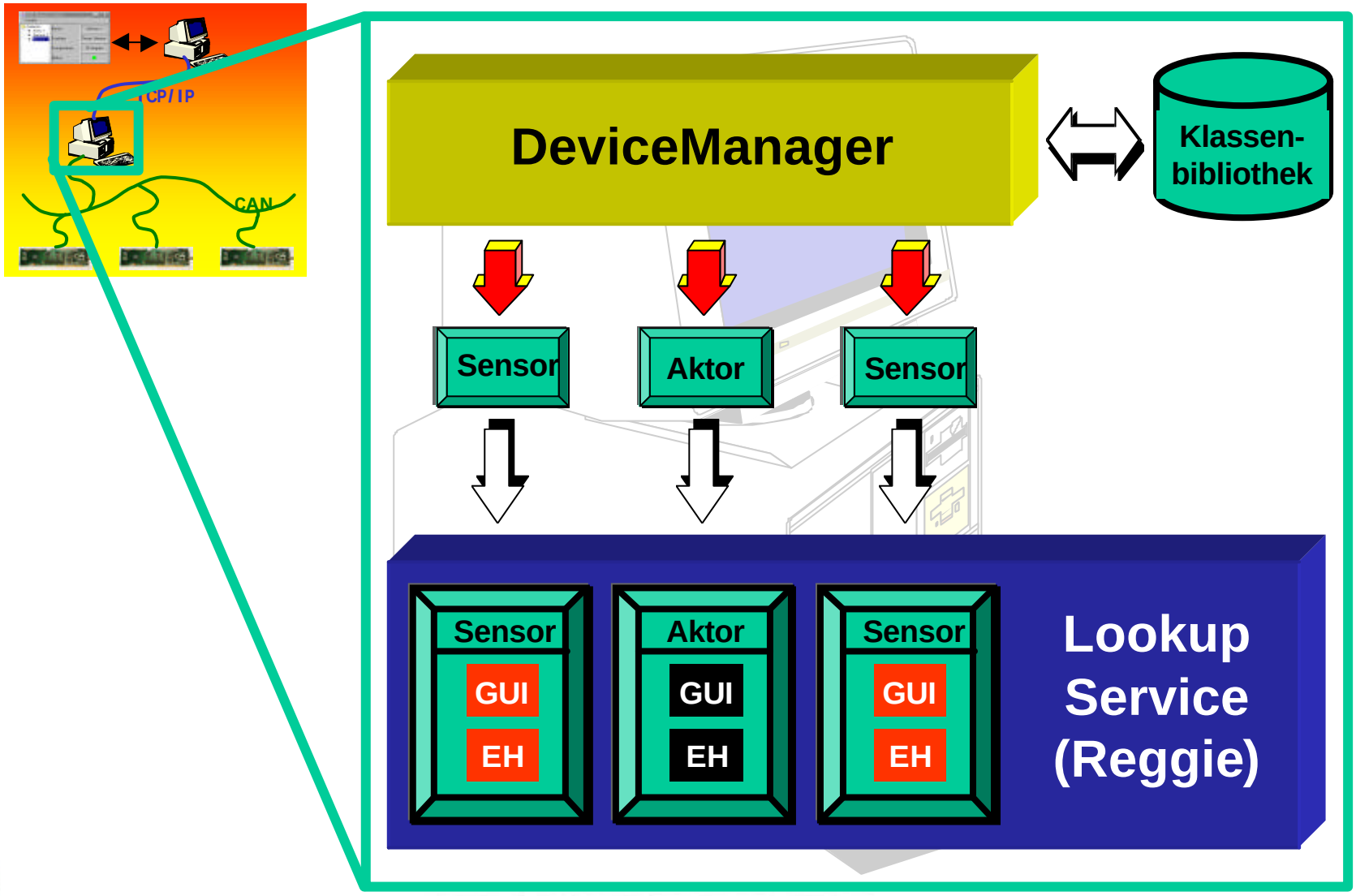
Jini = netzwerkfähige Java-Technologie

	Infrastruktur	Programmiermodell	Dienste
Basis Java	Java VM RMI Java Sicherheits-modell	Java APIs JavaBeans ...	JNDI Enterprise Beans JTS ...
Java + Jini	Erweitertes RMI Discovery Verteilte Sicherheit Lookup	Leasing Two Phase Commit Verteilte Ereignisse	JavaSpaces Two Phase Commit Manager

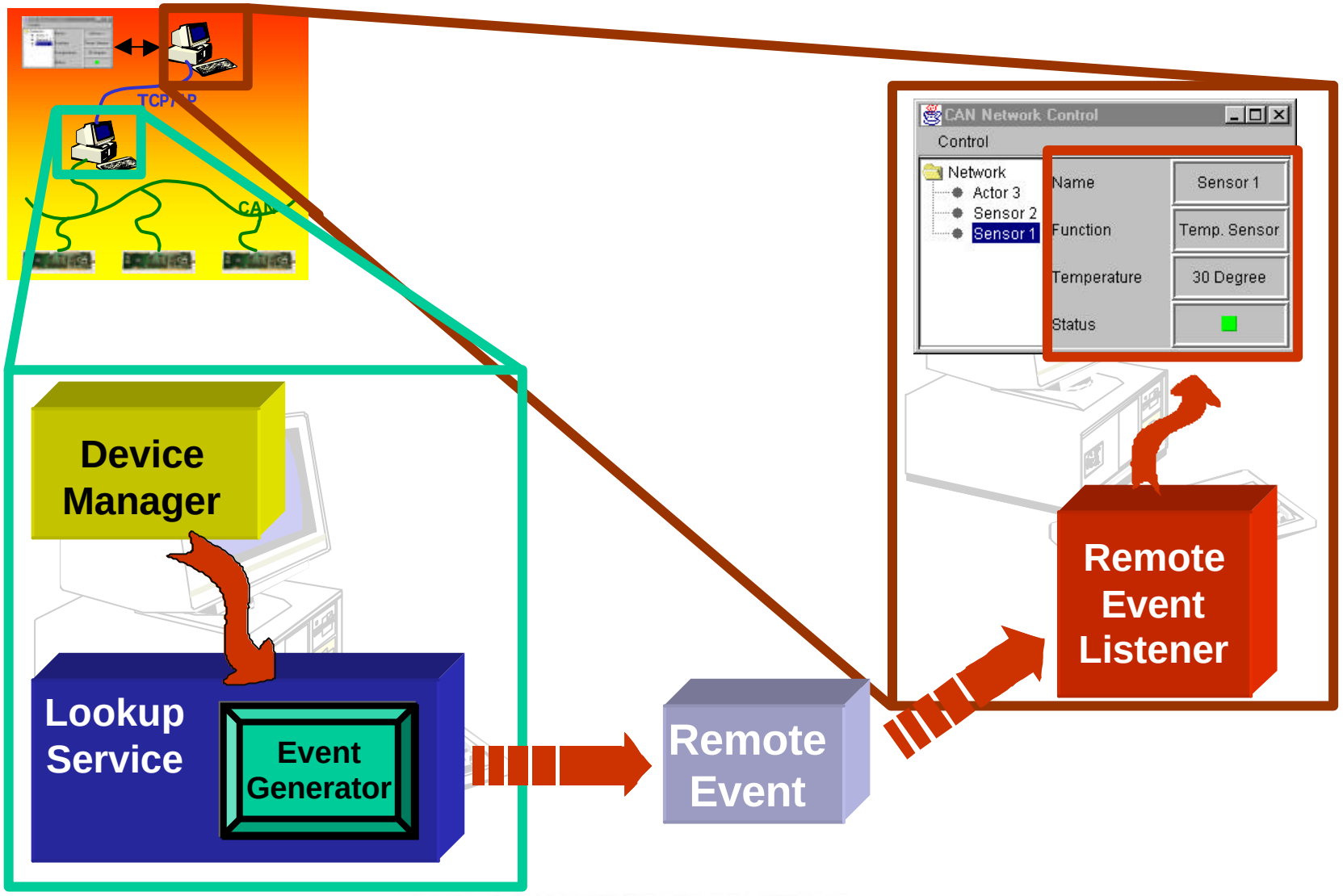
Steuerung/Überwachung eines CAN-Netzes



Systemarchitektur der Anwendung



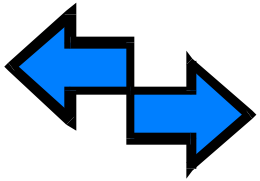
Verteilte Ereignisbehandlung



PJ, EJ und die J2ME

Kaffe Custom Edition

- PersonalJava 1.1.1 - Implementierung
- Quell-Code für geringe Gebühr
- Große Entwickler-gemeinde



KVM der J2ME-CLDC

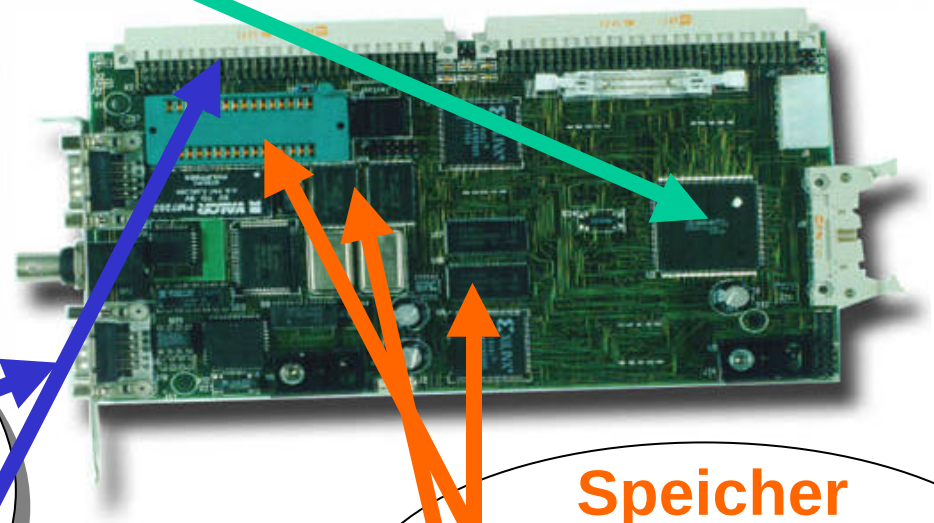
- Referenz-Impl. der J2ME-CLDC Spez.
- Quell-Code unter der SCSL
- Neueste Entwicklung im Rahmen der Java 2 Plattform

SPYDER-CORE-P2

Mikrocontroller

- Hitachi-SH7729
- DPS-Einheit on-chip
- 16 KB Befehls-/Datencache
- 133 MHz interner/ 66 MHz externer Bustakt

Echtzeitbetriebs-system VxWorks



Schnittstellen

- Ethernet (10 Base-2)
- 3 Serielle Schnittstellen
- CAN-Bus
- 96-polige Erweiterungs-stecker

Speicher

- 1 MB EPROM
- 16 MB SDRAM
- 4 MB FlashRAM

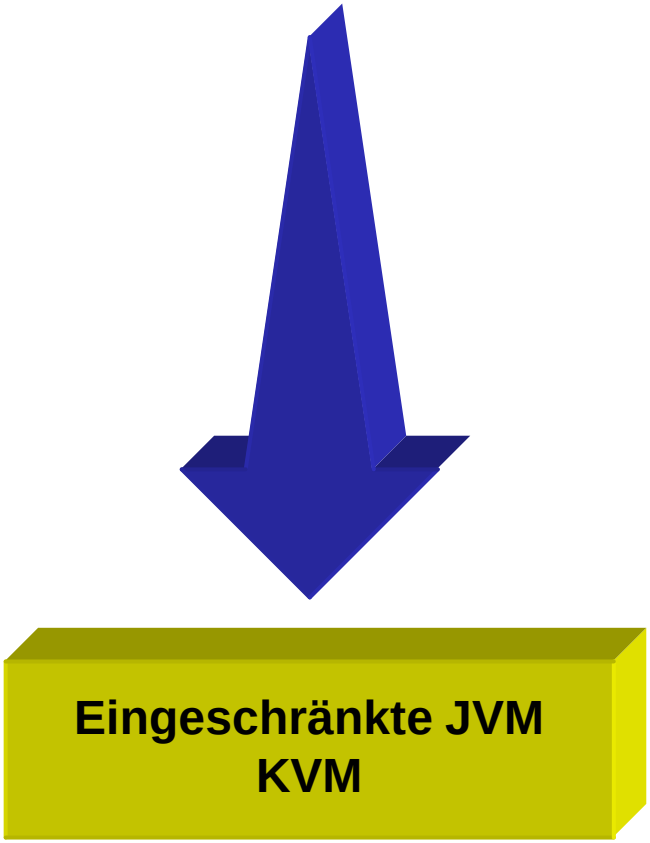
Bewertungskriterien



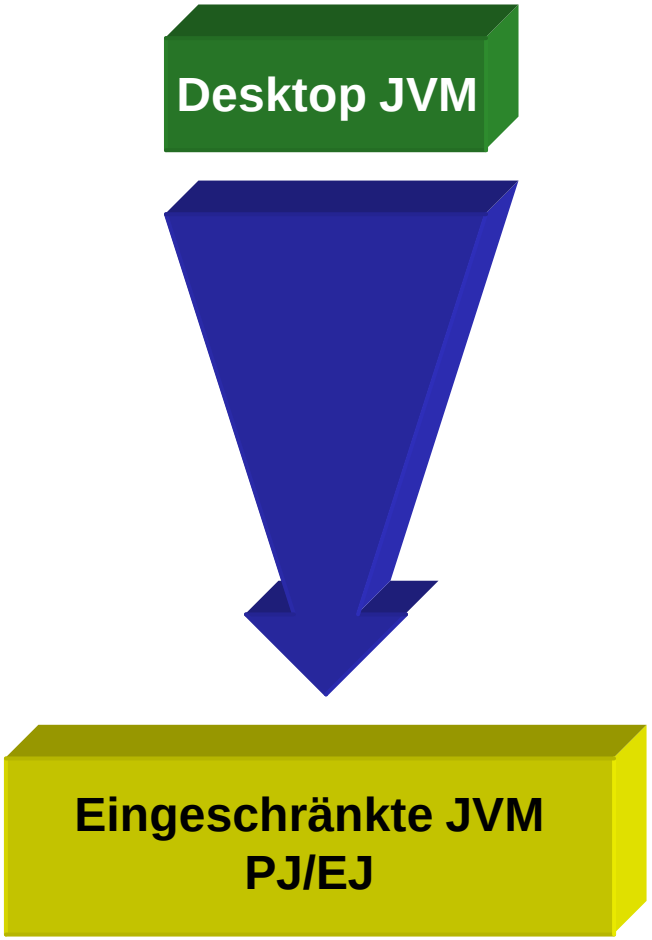
- **Funktionalität**
  - **JVM Design**
  - **Klassenbibliothek**
  - **Threading**
- **Speicherplatzverbrauch**
  - **Laufzeitumgebung**
  - **Klassenbibliothek**
- **Laufzeitverhalten**
  - **Optimierungsmöglichkeiten**
- **Portierungsaufwand**

Implementierung der JVM

„Blank Slate” - Ansatz



„Full Tablet” - Ansatz



Funktionalität

	Java Version	JIT	Interpreter	JNI	Java RMI	`float double`
Kaffe Custom	1.1	✓	✓	✓	✗	✓
KVM	1.2	✗	✓	✗	✗	✗

Kaffe

- Volle Java-Sprach-spezifikation
- PersonalJava 1.1
- JIT / Interpreter

KVM

- Eingeschränkter Java-Sprachumfang
- J2ME-CLDC
- Interpreter

Bewertungskriterien

- **Funktionalität**
  - **JVM Design**
  - **Klassenbibliothek**
  - **Threading**



- **Speicherplatzverbrauch**
  - **Laufzeitumgebung**
  - **Klassenbibliothek**
- **Laufzeitverhalten**
  - **Optimierungsmöglichkeiten**
- **Portierungsaufwand**

Optimierte Java-Laufzeitumgebungen

	JDK 1.2	JDK 1.1.5	Kaffe Custom	KVM CLDC mit ROMIZER	KVM CLDC ohne ROMIZER
JVM ohne Debug Code	2400 KB	843 KB	342 KB	437 KB	161 KB
JVM mit Debug Code	-	-	1012 KB	1004 KB	567 KB
Klassen-bibliothek	10600 KB	8900 KB	1124 KB	-	160 KB

- keine Objekt-serialisierung

- Größe der JVM mit Debug-Code ist für die Portierung wichtig

- keine Doubles/ Floats
- spezieller Preverifier

Preverifier

	JVM-Code	*Speicherplatzverbrauch zur Laufzeit*
J2SE	50 KB	30 – 100 KB RAM
J2ME	8 KB	<100 Bytes RAM

- **In C implementiert**
  - ➡ Programm ist in Teilen in der JVM 1.1.8 enthalten
- **Inlining von Subroutinen**
  - ➡ Linearer Scan- statt iterativer Datenfluss-Algorithmus
- **STACK-Map Attribut**
  - ➡ Zusätzliches Attribut wird in Classfile eingefügt
  - ➡ Verursacht 5% größere Classfiles
  - ➡ Beschreibt für jede Methode die Typen der lokalen Variablen und Operanden auf dem Stack

Eingeschränkte Klassenbibliotheken

	JDK 1.2	JDK 1.1.5	Kaffe Custom	KVM CLDC mit ROMIZER	KVM CLDC ohne ROMIZER
JVM ohne Debug Code	2400 KB	843 KB	342 KB	437 KB	161 KB
JVM mit Debug Code	-	-	1012 KB	1004 KB	567 KB
Klassen-bibliothek	10600 KB	8900 KB	1124 KB	-	160 KB

PersonalJava

- keine RMI
- keine Security
- keine Reflection

J2ME CLDC

- `java.lang`
- `java.io`
- `java.util`

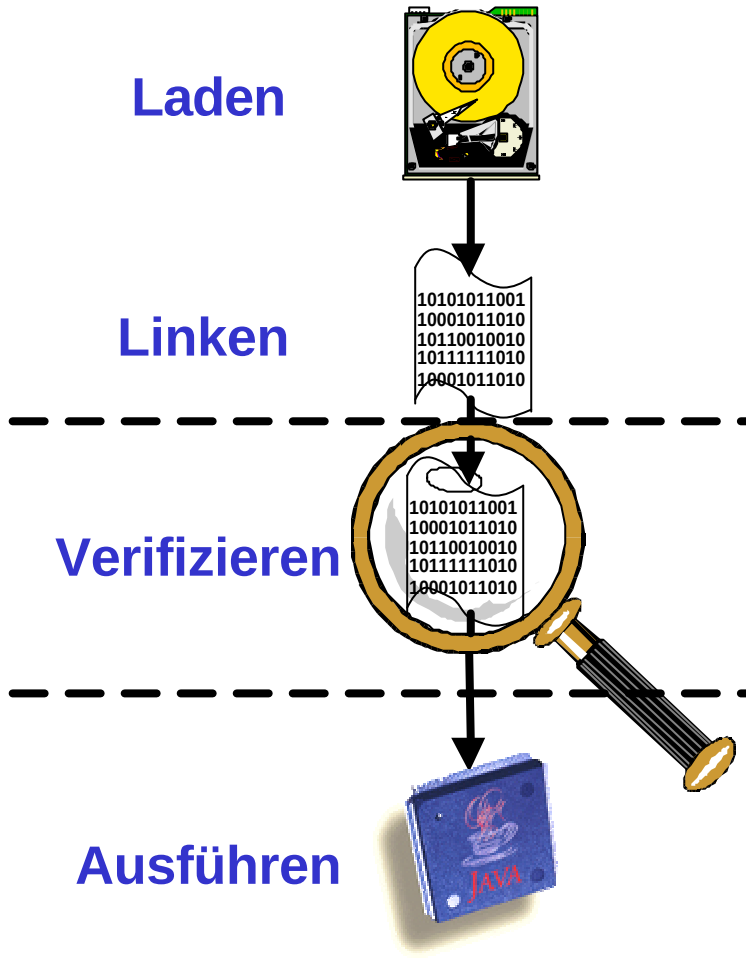
Bewertungskriterien

- **Funktionalität**
  - **JVM Design**
  - **Klassenbibliothek**
  - **Threading**
- **Speicherplatzverbrauch**
  - **Laufzeitumgebung**
  - **Klassenbibliothek**



- **Laufzeitverhalten**
  - **Optimierungsmöglichkeiten**
- **Portierungsaufwand**

Optimierung des Laufzeitverhaltens

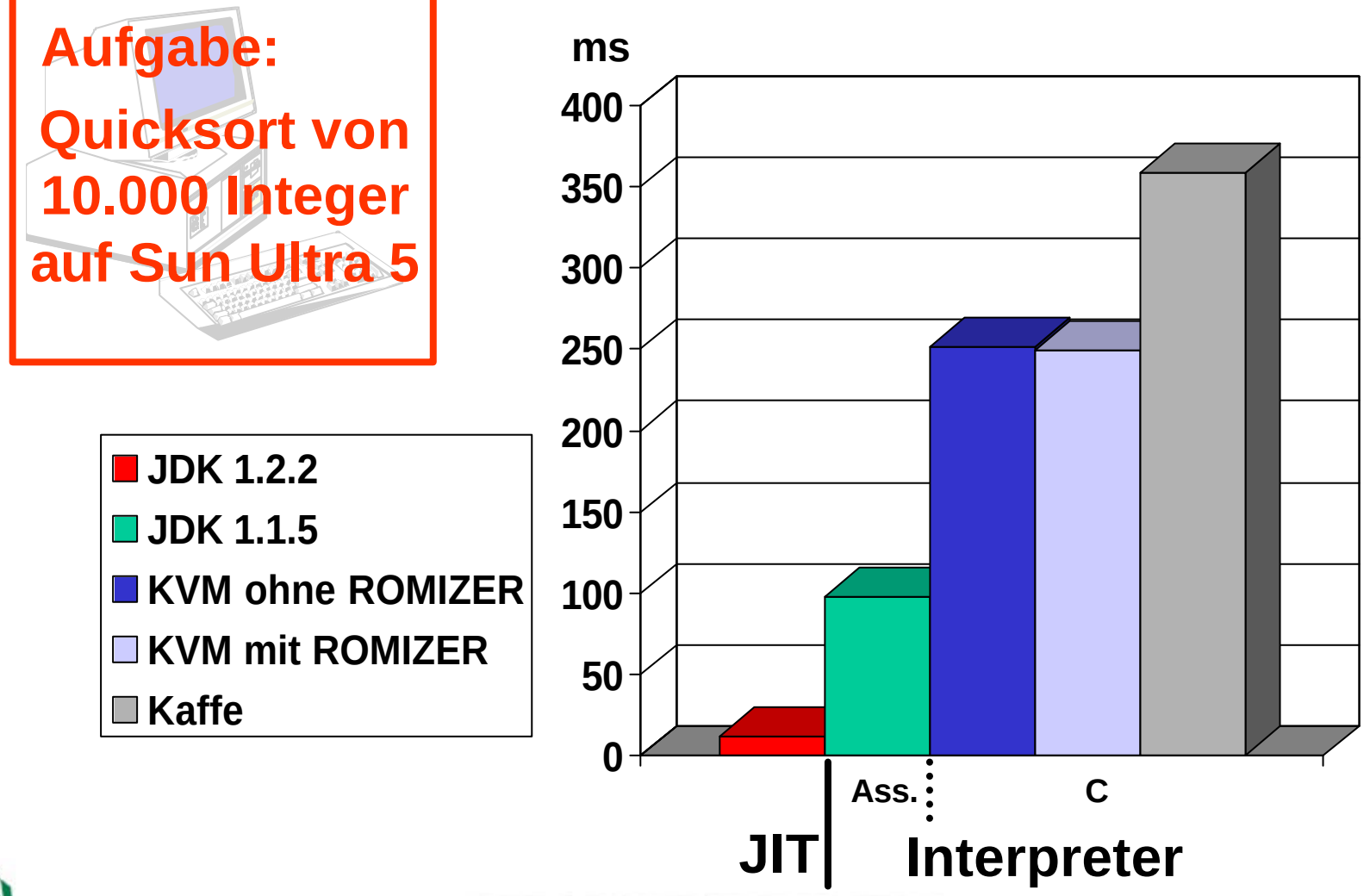


- JavaCodeCompact
- Preverifier
- JIT-Compiler
- Native Java Compiler
- Java-Prozessoren

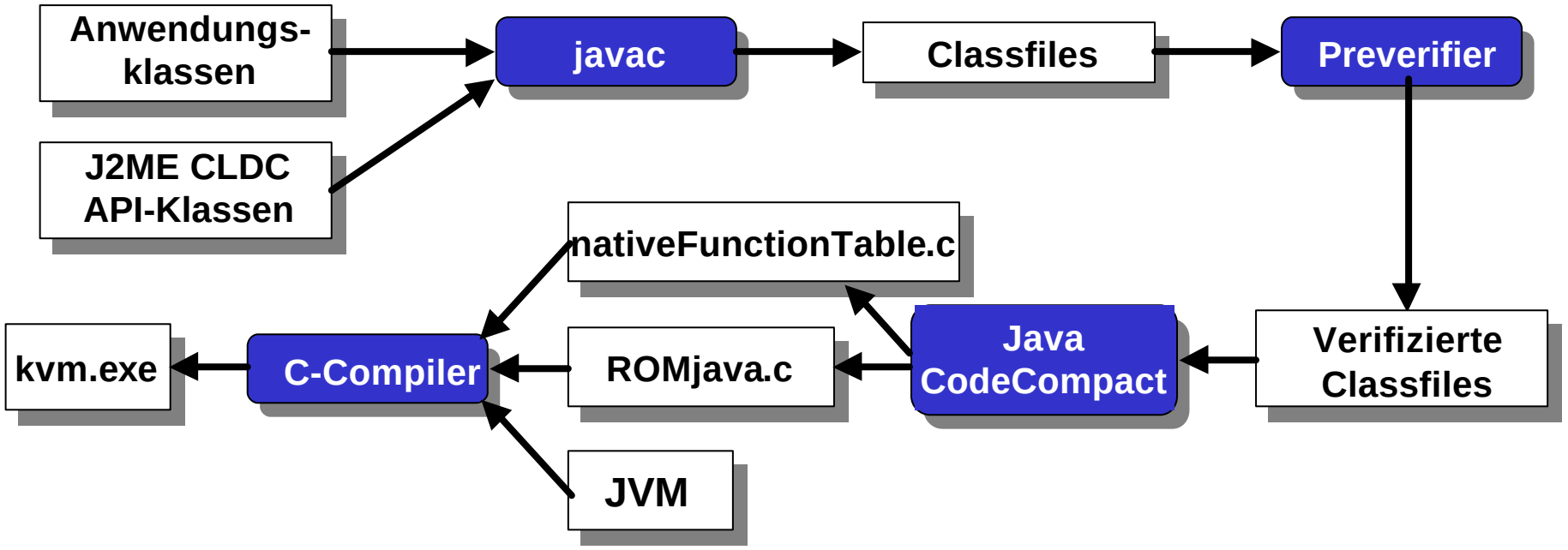
Laufzeitoptimierungen des Interpreters

Aufgabe:

Quicksort von 10.000 Integer auf Sun Ultra 5



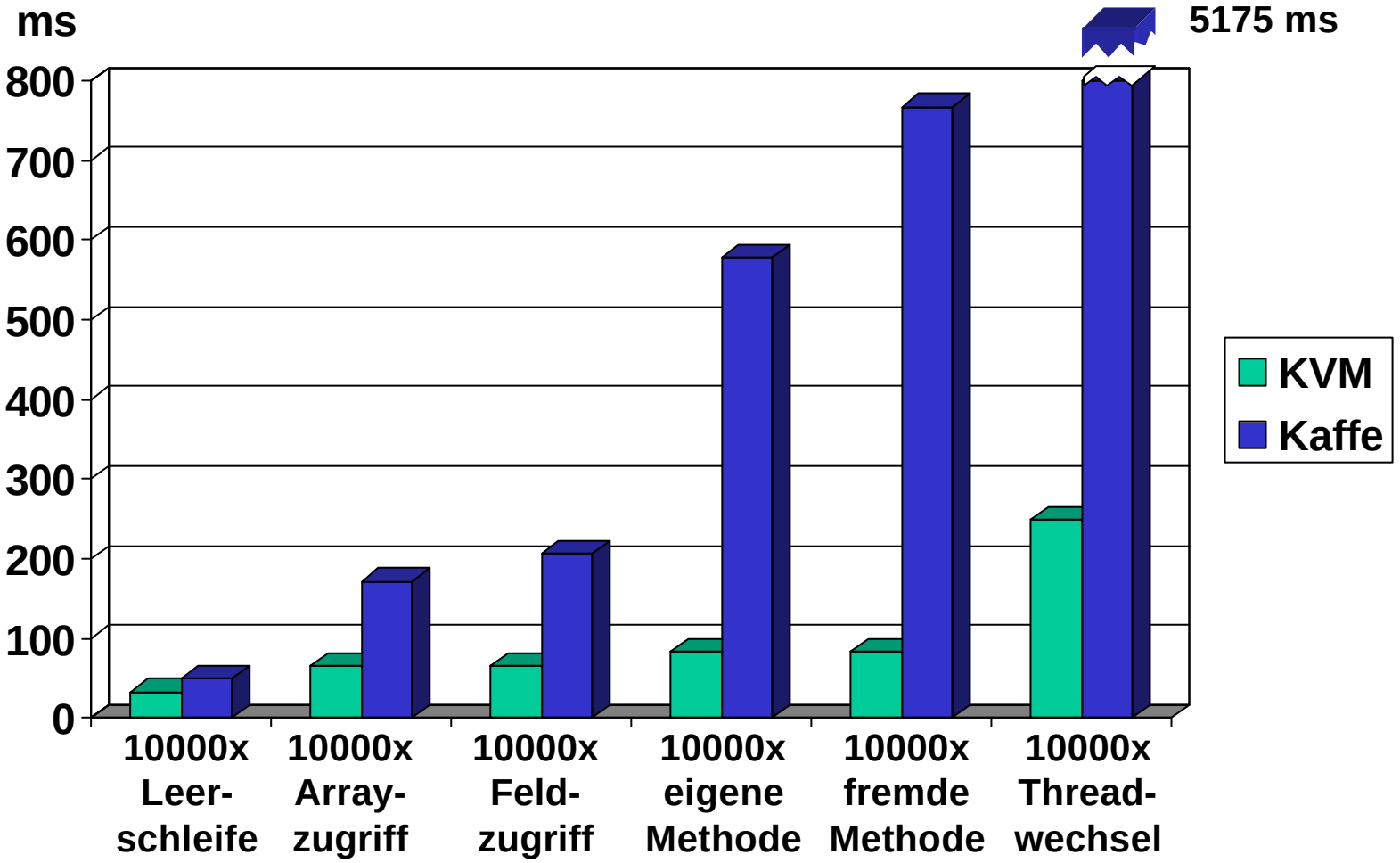
Laufzeitoptimierung durch ROMIZER



Bsp.:

	Kaffe (ohne ROMIZER)	*KVM CLDC ohne ROMIZER*	*KVM CLDC mit ROMIZER*
JVM	342 KB	161 KB	437 KB
Klassenbibliothek	1124 KB	160 KB	-
Startup der Systemklassen (Object, Class, String, System)	81.000 us	6556 us	8 us

Allgemeine Leistungsmessungen



Bewertungskriterien

- **Funktionalität**
  - **JVM Design**
  - **Klassenbibliothek**
  - **Threading**
- **Speicherplatzverbrauch**
  - **Laufzeitumgebung**
  - **Klassenbibliothek**
- **Laufzeitverhalten**
  - **Optimierungsmöglichkeiten**
- **Portierungsaufwand**

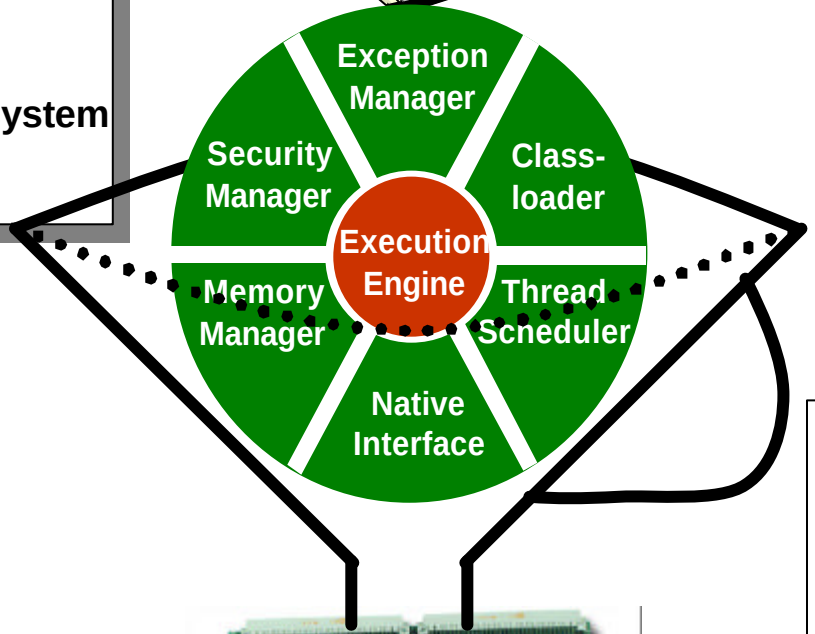


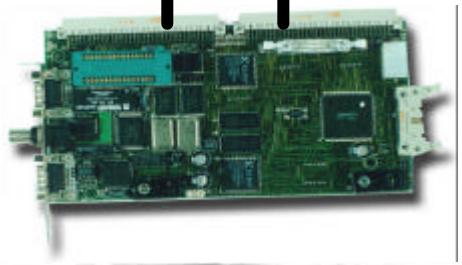
Portierung einer Java Virtual Machine

Desktop

- 333 MHz UltraSPARC-II i
- 384 MB Hauptspeicher
- 64 KB L1-Cache
- 2 MB L2-Cache
- Desktop-Betriebssystem
- Dateisystem



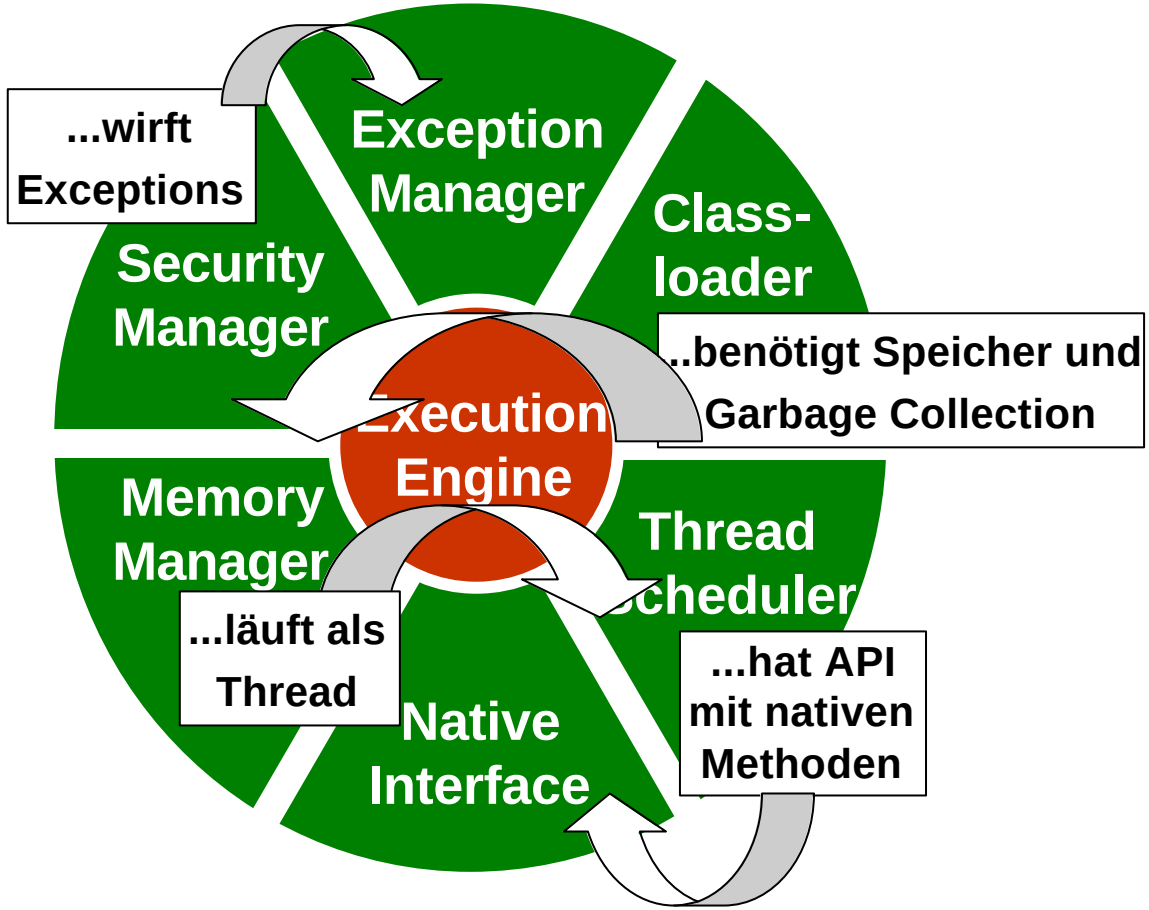




Eingebettetes System

- 166 MHz Hitachi SH7729
- 16 MB Hauptspeicher
- 16 KB Cache
- Echtzeit-Betriebssystem
- RAM-Dateisystem

Aufbau der Java Virtual Machine



Probleme bei der Portierung

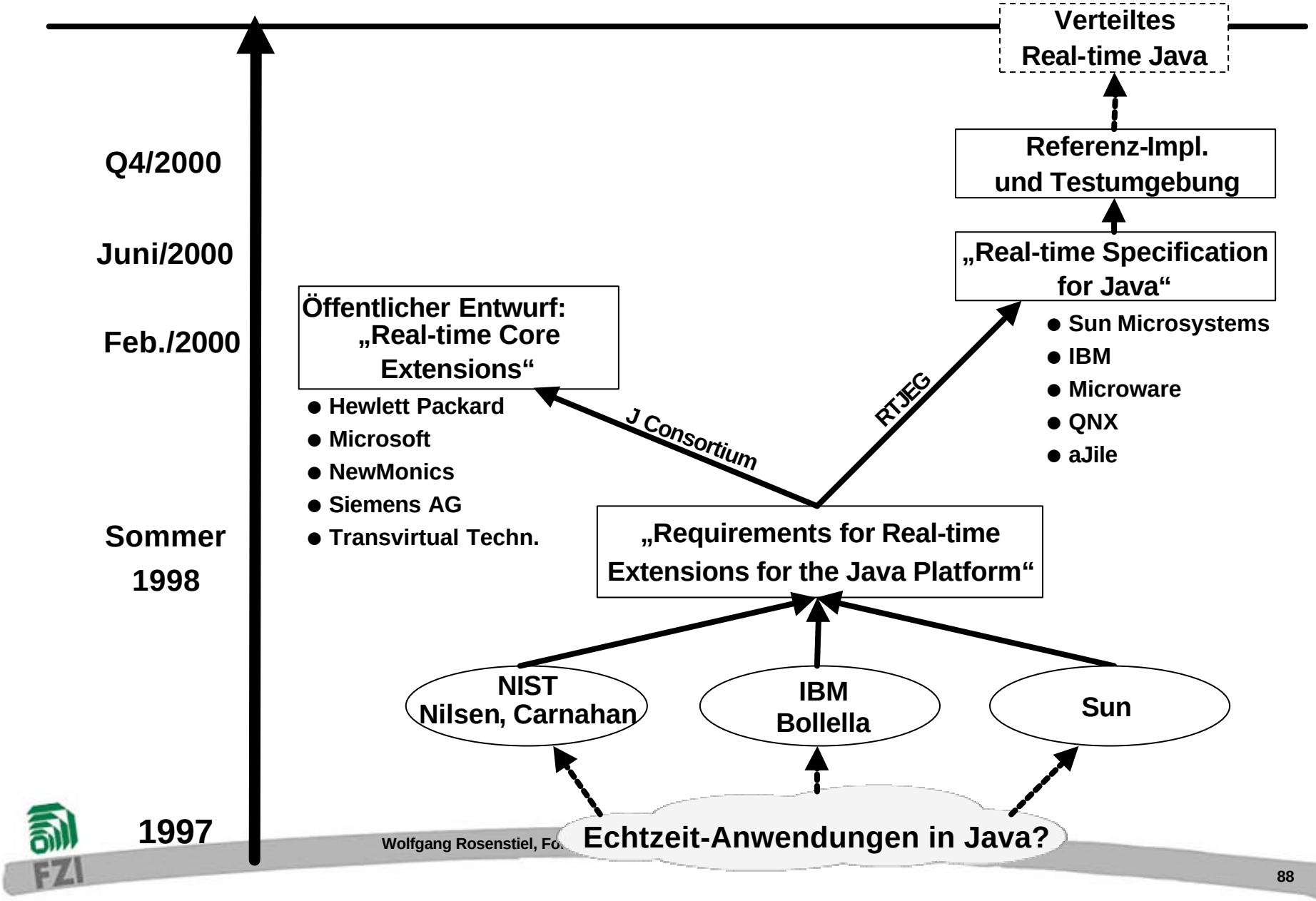
- **Threading:** Native oder Greenthreads?
- **Classloader:** Dateisystem oder EmbeddedJava Impl.
- **Debugging:** Native Code und Bytecode gemischt
- **Betriebssystemfunktionalität:** u.U. auf dem Echtzeitsystem stark eingeschränkt
- **Quellcode:** für Desktop Systeme sehr umfangreich

	JDK 1.1.6 Greenthreads (loc)	*JDK 1.1.6 Native Threads (loc)*	*KVM (loc)*	*Kaffe (loc)*
Plattform-abhängiger Code	7294 + 4941	3643 + 4941	1828	36328
Plattform-unabhängiger Code	82203	82203	27646	

Bewertung im Überblick

Bewertungskriterium	*Kaffe Custom Edition*	*J2ME-CLDC*
Funktionalität	+	-
Java-Sprachumfang	Voller Sprachumfang	Eingeschränkt
Verifier	Nicht vorhanden	Optimiert
Klassenbibliothek	PersonalJava 1.1	java.lang.*, java.io.*, java.util.*
Speicherplatz-verbrauch	-	+
Laufzeitumgebung	-	+
Klassenbibliothek	-	+
Laufzeitverhalten	-	+
Performance	-	+
Optimierungs-möglichkeiten	-	+
Portierungsaufwand	-	+

Real-Time Java Geschichte



Vergleich der beiden Spezifikationen

Real-Time Core Extensions

- Legt Java Syntax und Tools um Echtzeit-BS Dienste
- Erweitert gebräuchliche virtuelle Maschinen
- Grundlage für die Entwicklung höherer Architekturen

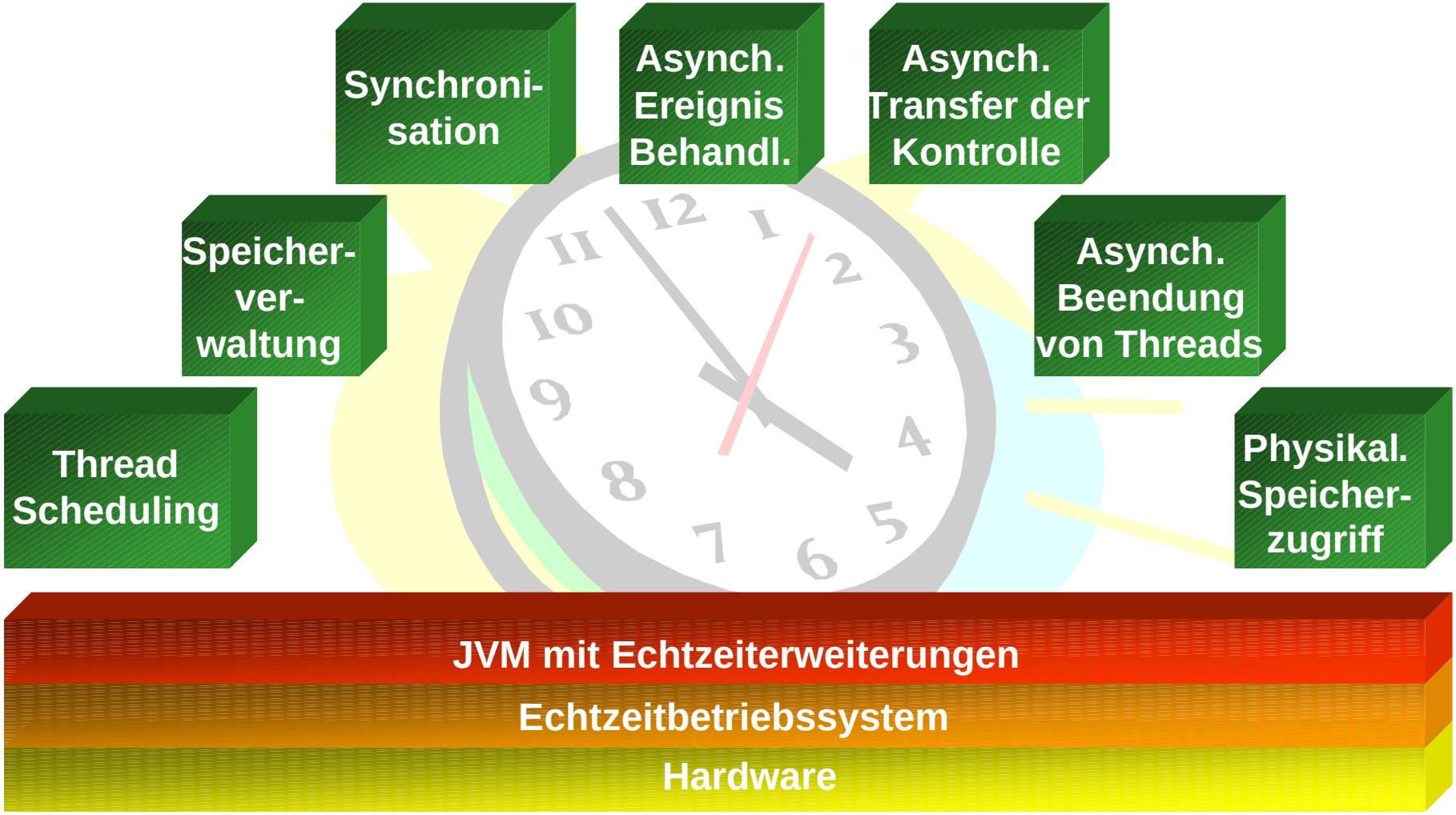
Real-Time Specification for Java

- Erweitert die JVM Spezifikation um Echtzeiteigenschaften
- Verändert das interne Design der virtuellen Maschine
- Adressiert durch dünne Spezifikation ein breites Anwendungsgebiet

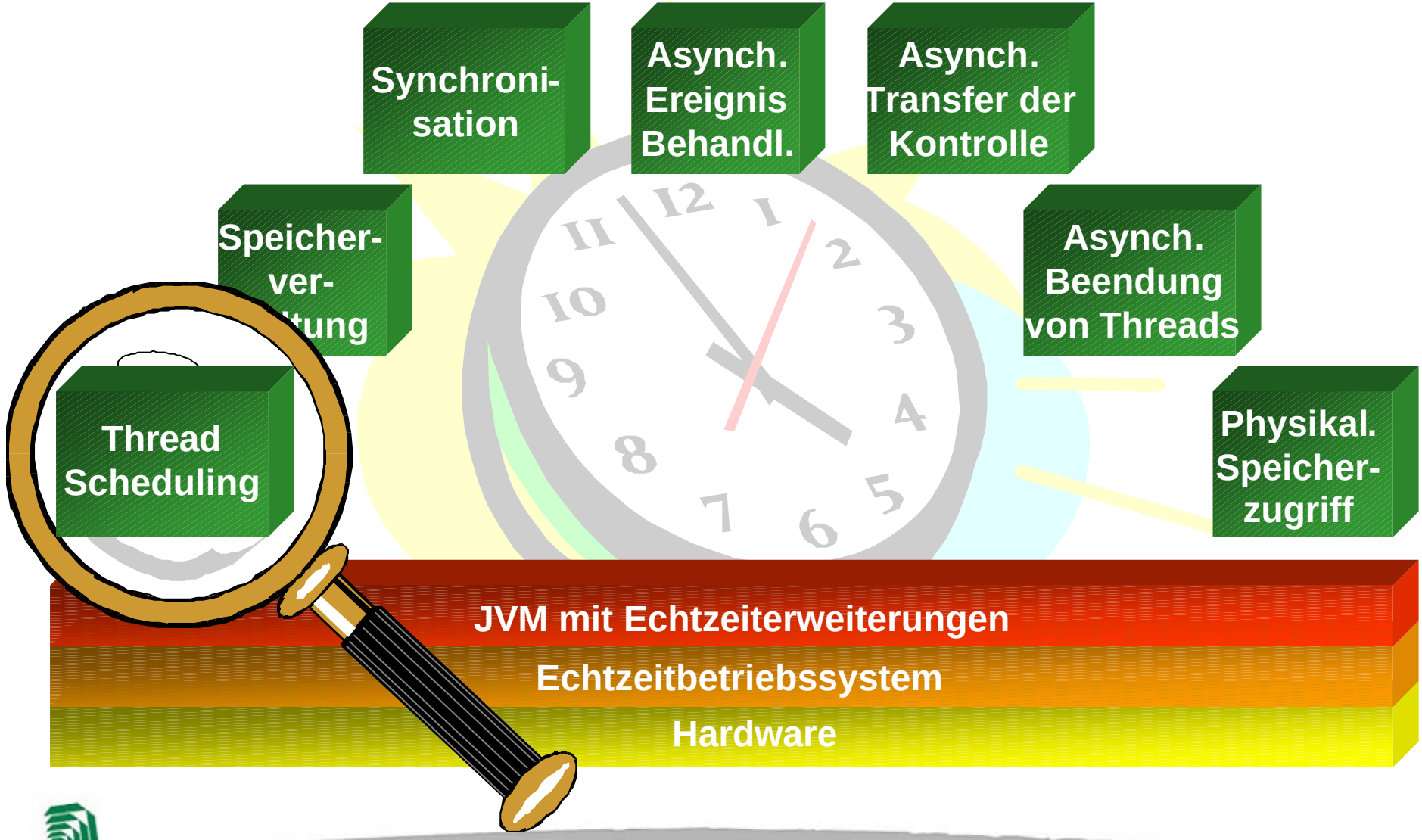
Leitlinien für die RTSJ Entwicklung

- **Rückwärts Kompatibilität**
  - ➡ Normale Java Anwendungen sollen auch auf RTSJ Implementierungen ausführbar sein
- **Für jede Java Plattform geeignet**
  - ➡ Spezifikationen dürfen keine speziellen Java-Umgebungen voraussetzen
- **Vorhersagbare Ausführung**
  - ➡ Vorhersagbare Ausführung hat höchste Priorität
- **Keine syntaktischen Erweiterungen**

Erweiterungen durch die RTSJ



Erweiterungen durch die RTSJ

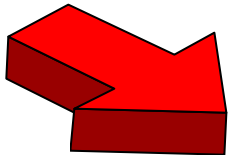


Lösung des Scheduling Problems



Problem:

Kein eindeutiges Thread Scheduling auf unterschiedlichen Java-Plattformen



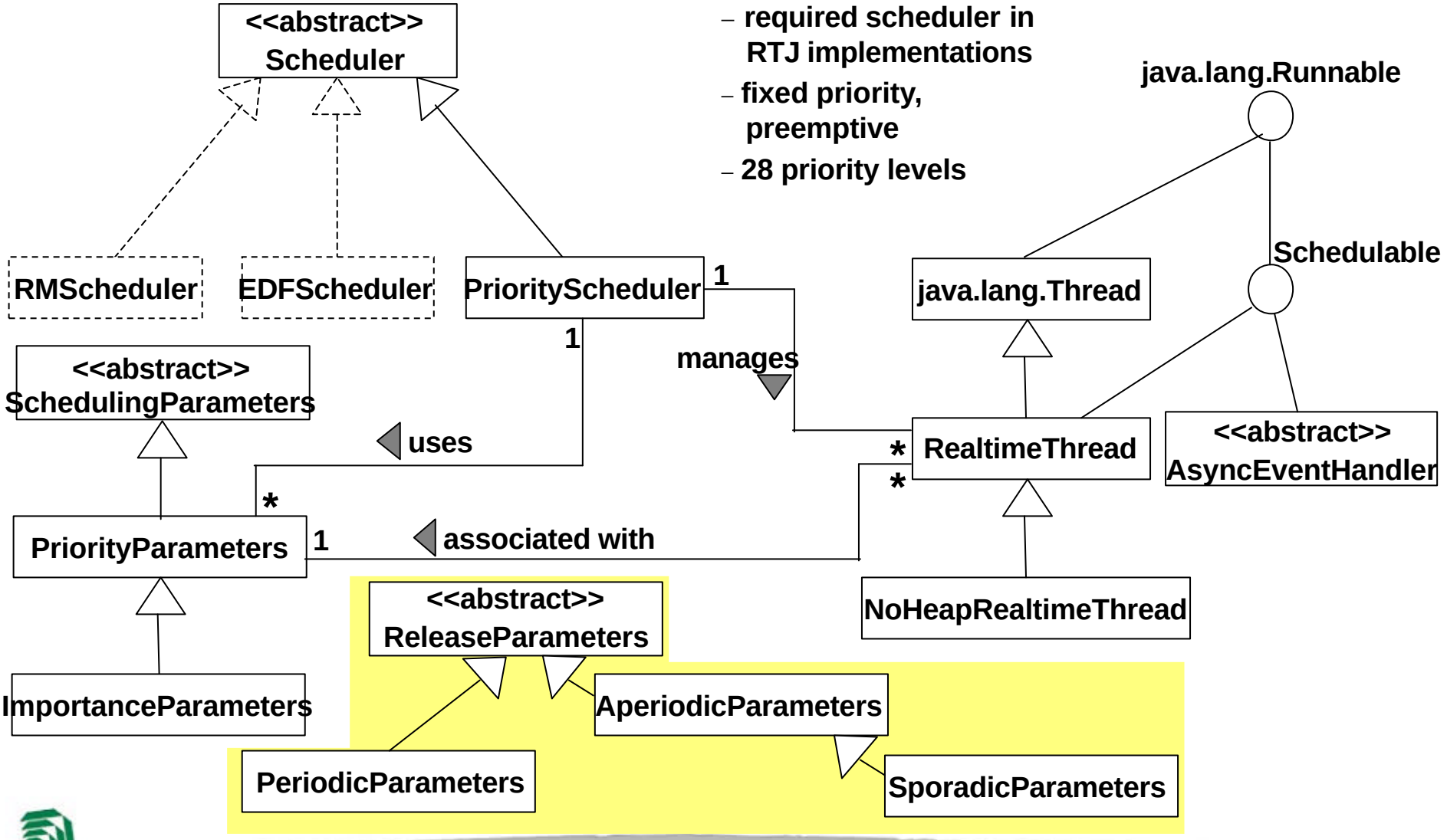


Flexibles Scheduling Framework welches aktuelle *und* zukünftige Anforderungen von Echtzeitsystemen genügt.

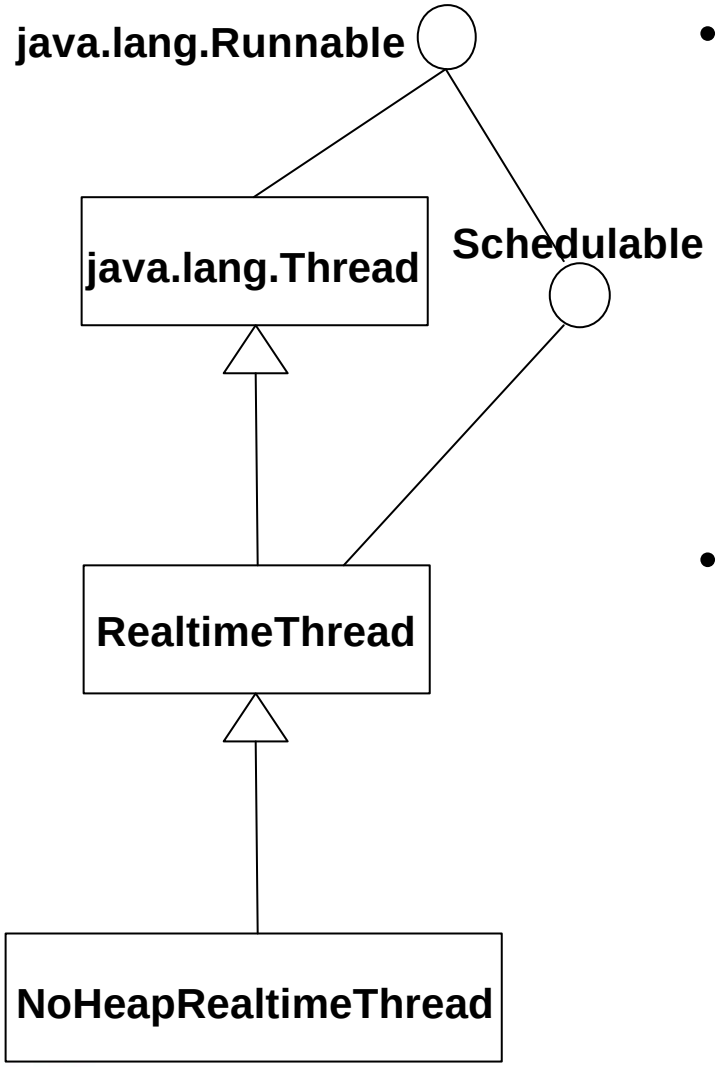
Design Entscheidungen

- **Scheduling-Module implementieren die Scheduling-Strategie**
- **Scheduling-Module verwalten planbare Objekte (schedulable objects)**
- **`RealtimeThreads` implementieren die Echtzeiteigenschaften der Anwendung**
- **Parameter-Klassen definieren die Eigenschaften planbarer Objekte**

Design des Scheduling Frameworks

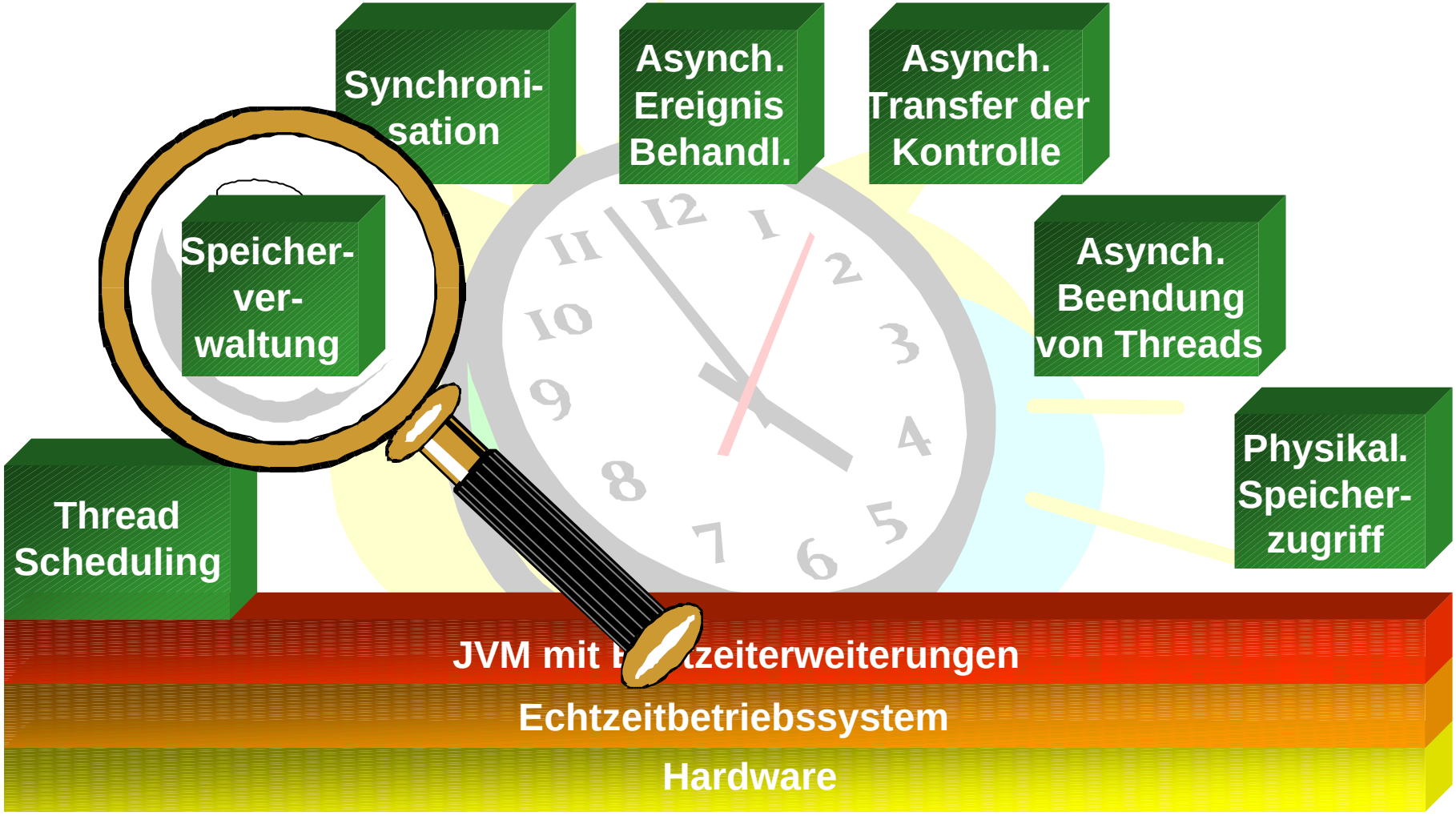


Real-Time Threads



- **RealtimeThread**
  - Verwaltet durch Scheduler
  - Kann Speicher außerhalb des Heaps adressieren
  - Asynchroner Transfer des Kontrollflusses und Terminierung möglich
  - Kann auf physikalischen Speicher zugreifen
- **NoHeapRealtimeThread**
  - Kann keine Objekte auf dem Heap anlegen oder adressieren
  - Darf keine Referenzen auf Objekte auf dem Heap beeinflussen
  - Muß mit einem Speicherrahmen erzeugt werden
  - Kann den Garbage Collector **sofort** unterbrechen

Erweiterungen durch die RTSJ

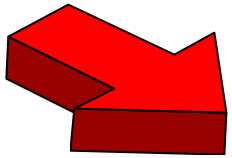


Speicherverwaltung der RTSJ



Problem:

Java Spezifikation macht keine detaillierten Aussagen über Speicher und GC

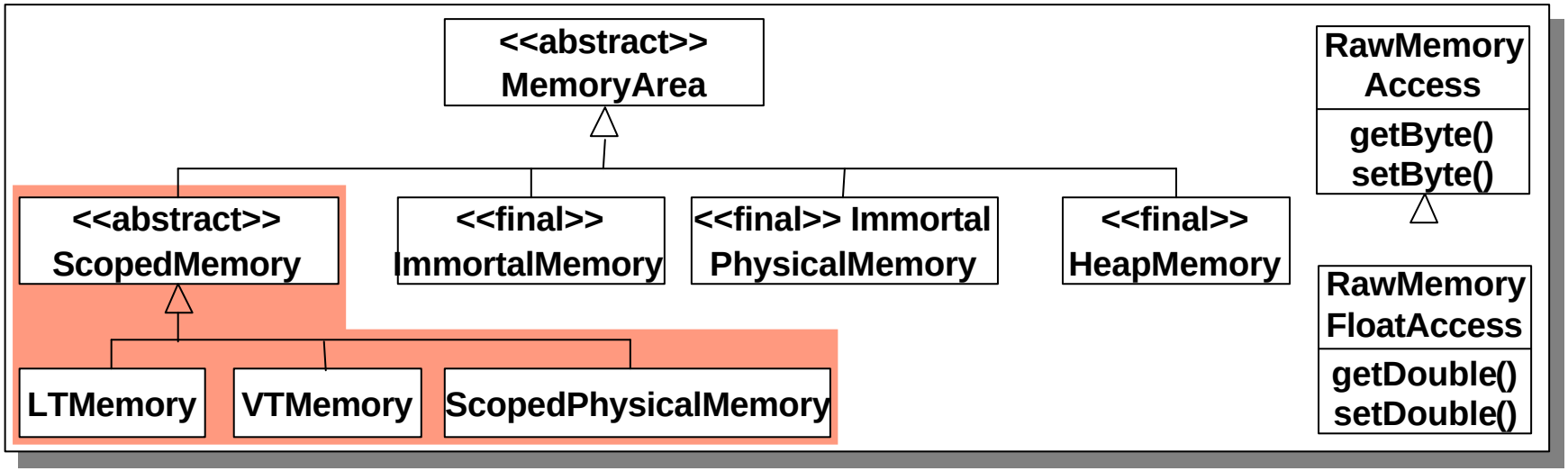


Einführung von Speicherbereichen:



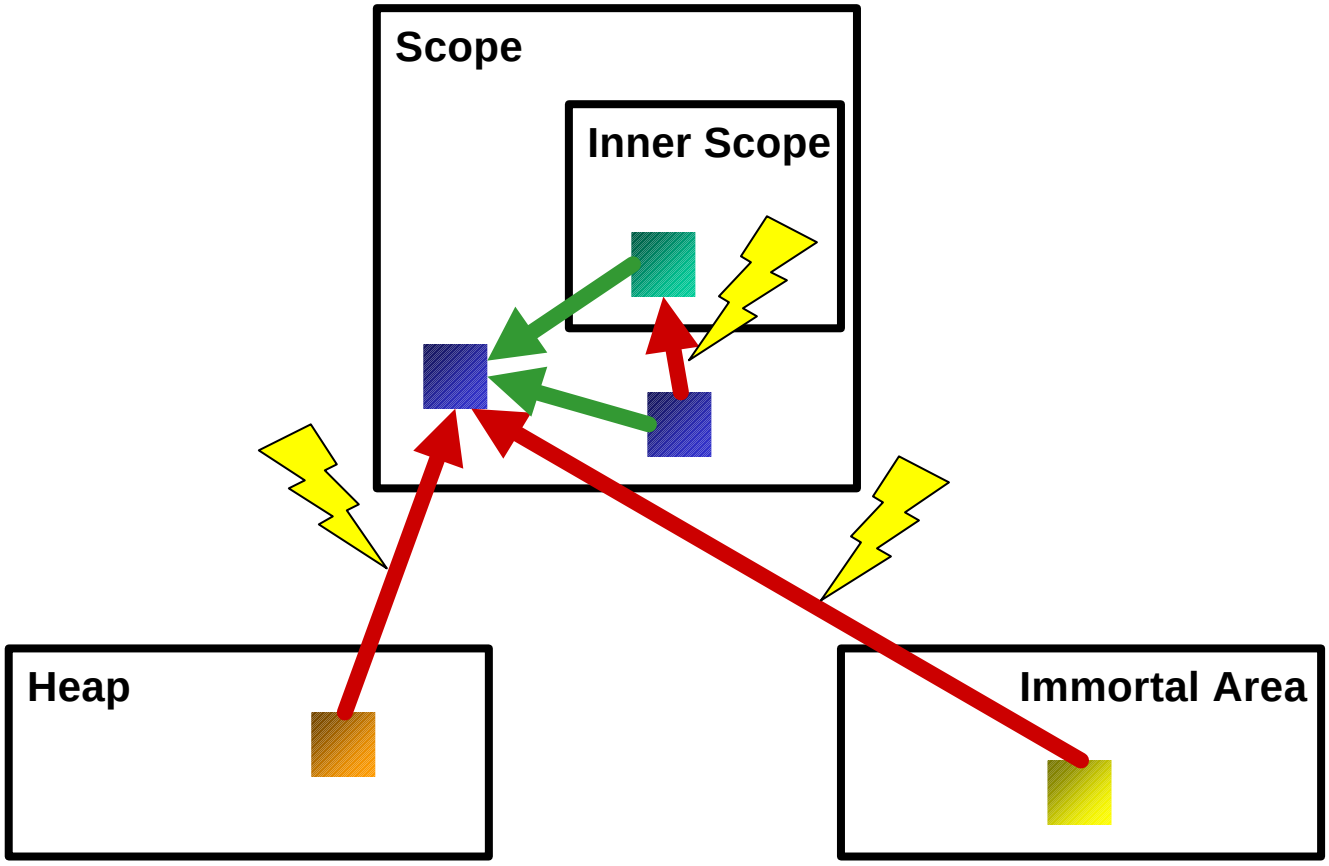
- **Speicherrahmen (Scoped memory)**
- **Unsterblicher Speicher (Immortal memory)**
- **Physikalischer Speicher (Physical memory)**
- **Heap Speicher**

Speicherrahmen (Scoped Memory)

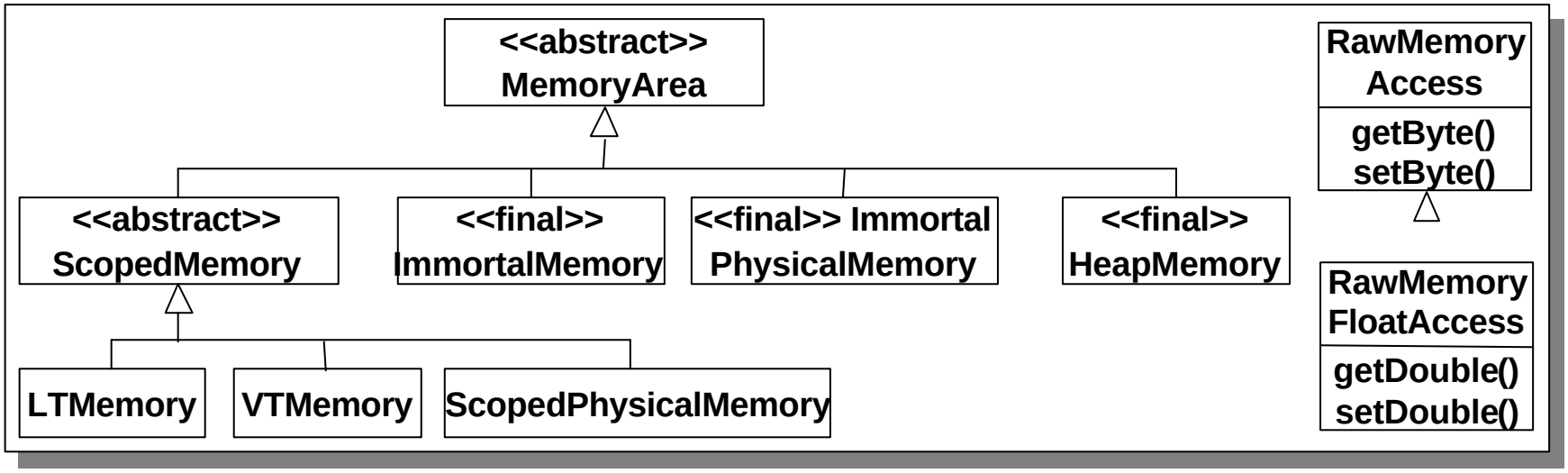


- **Speicherbereich mit begrenzter Lebenszeit**
  - ➡ **Kontrolliert mit Referenzzähler**
- **LTMemory**
  - ➡ **Ausführungszeit von new linear zur Größe des Objektes**
- **VTMemory**
  - ➡ **Ausführungszeit von new ist variabel**
- **Zuweisung von Objektreferenzen wird von der JVM kontrolliert**

Einschränkung von Objektreferenzen in Speicherrahmen

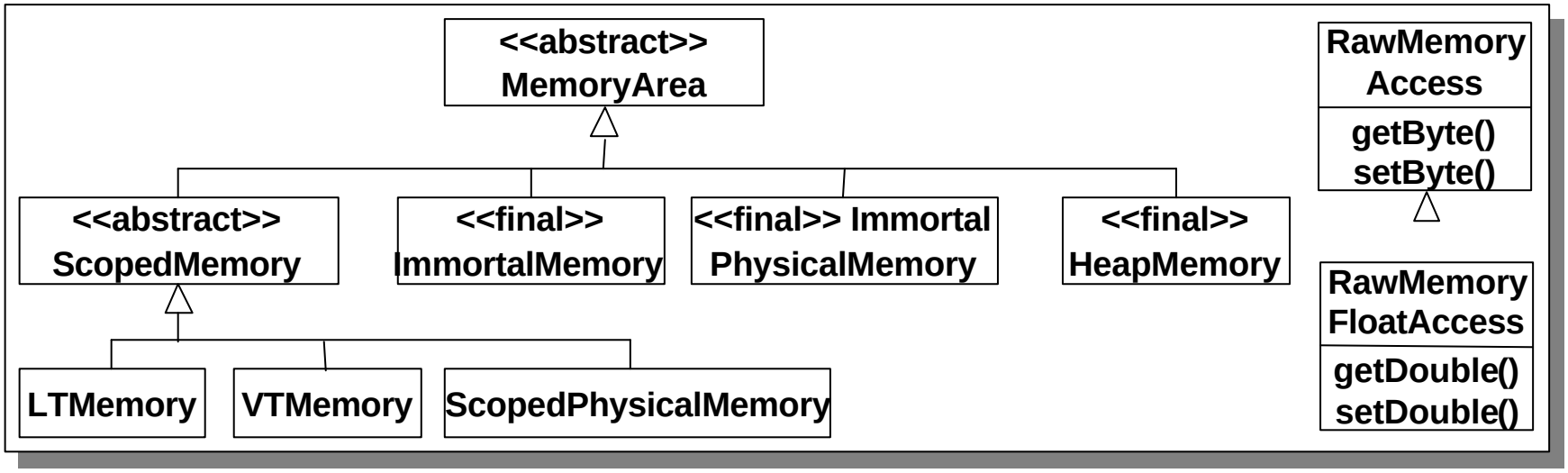


Unsterblicher Speicher (Immortal Memory)



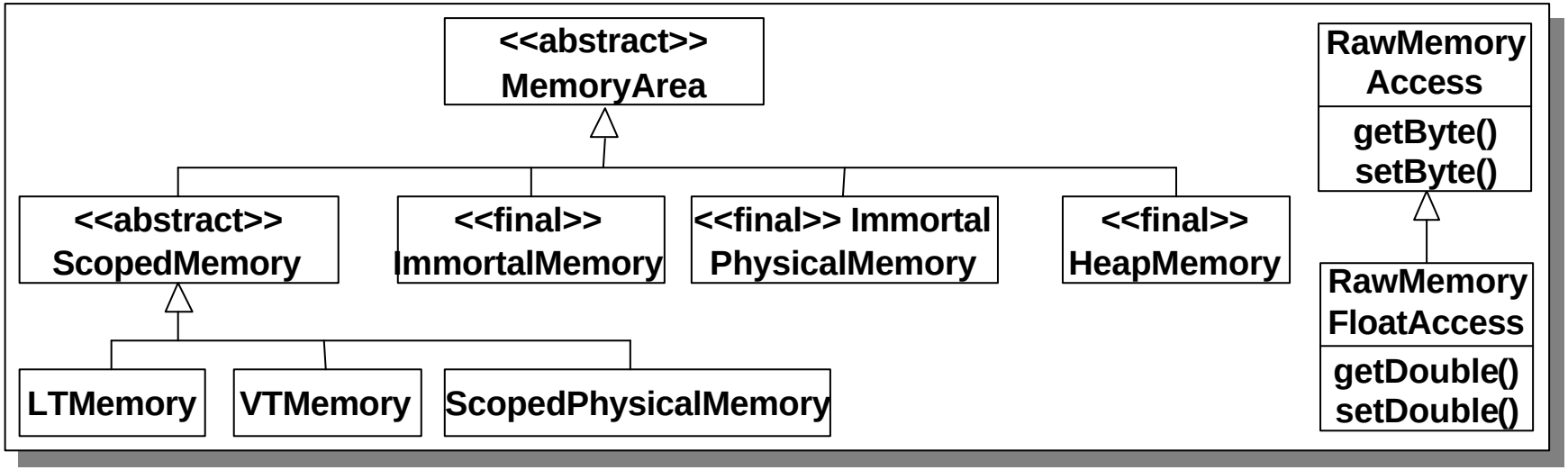
- Wird von allen Threads benutzt
- Einer pro JVM
- Nur Referenzen zu Objekten im selben Speicherbereich oder auf dem Heap erlaubt
- Objekte im `ImmortalMemory`-Bereich leben bis zum Ende der Anwendung
- Objekte im `PhysicalImmortalMemory`-Bereich leben länger als die Anwendung

Heap Speicher



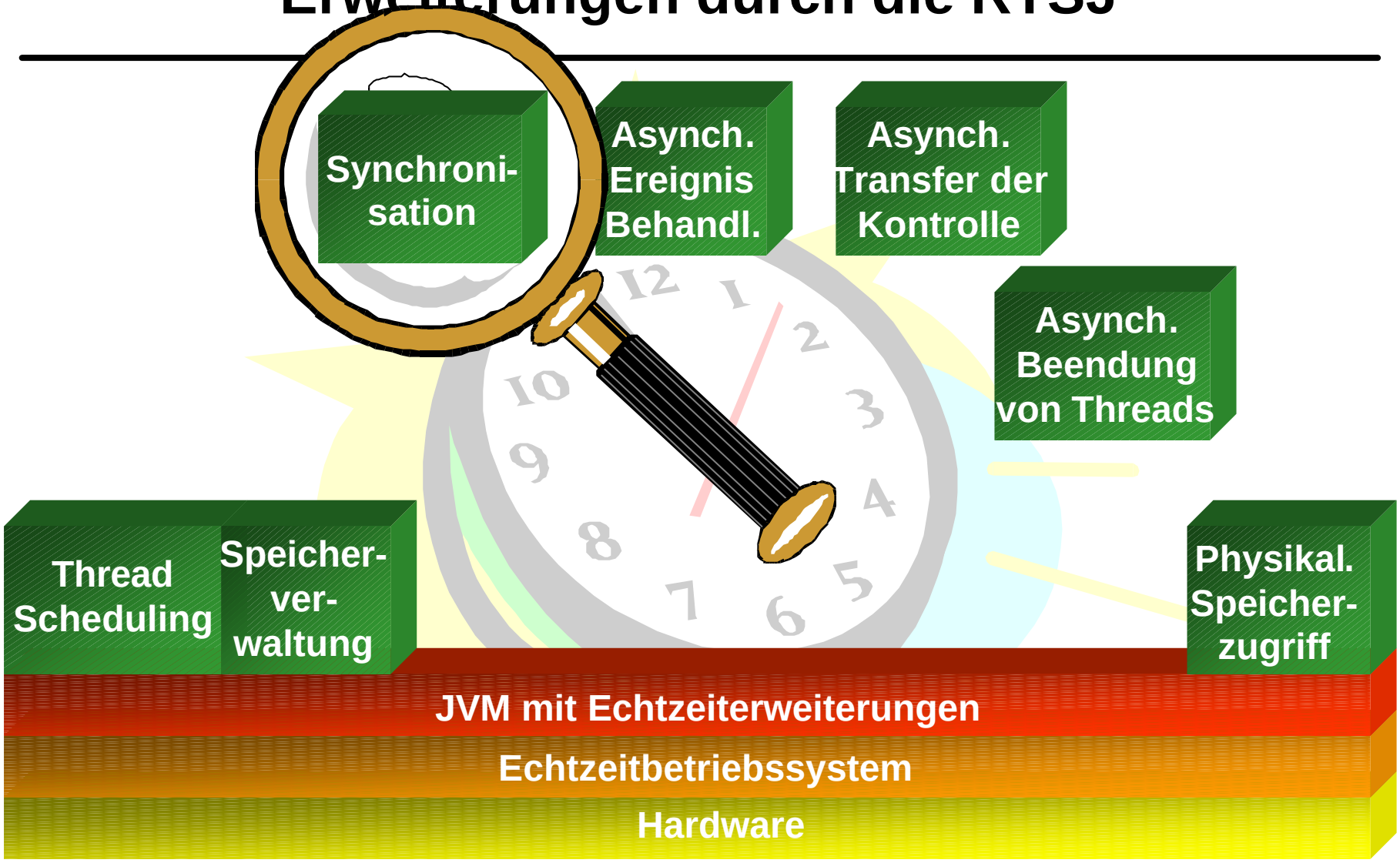
- **Singleton Objekt**
- **Kontrolliert Speicherzugriffe auf den Java-Heap von Objekten die in Speicherrahmen residieren**

Physikalischer Speicher



- Modelliert einen physikalischen Speicherbereich als Bytesequenz
- Implementierung von
  - Gerätetreibern
  - Memory-mapped I/O
  - Flash Speicher
  - Batteriebetriebenes RAM
- Referenzen zu Java-Objekten nicht erlaubt

Erweiterungen durch die RTSJ

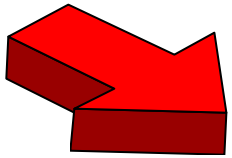


Synchronisation in der RTSJ



Problem:

Synchronisierung zwischen Java-Threads ist nicht genau spezifiziert

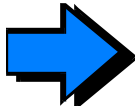




- **Präzise Semantik für das `synchronized`-Statement**
- **Prioritätsbasierte Warteschlangen mit FIFO**
- **Keine unbegrenzte Umkehr von Thread-Prioritäten**

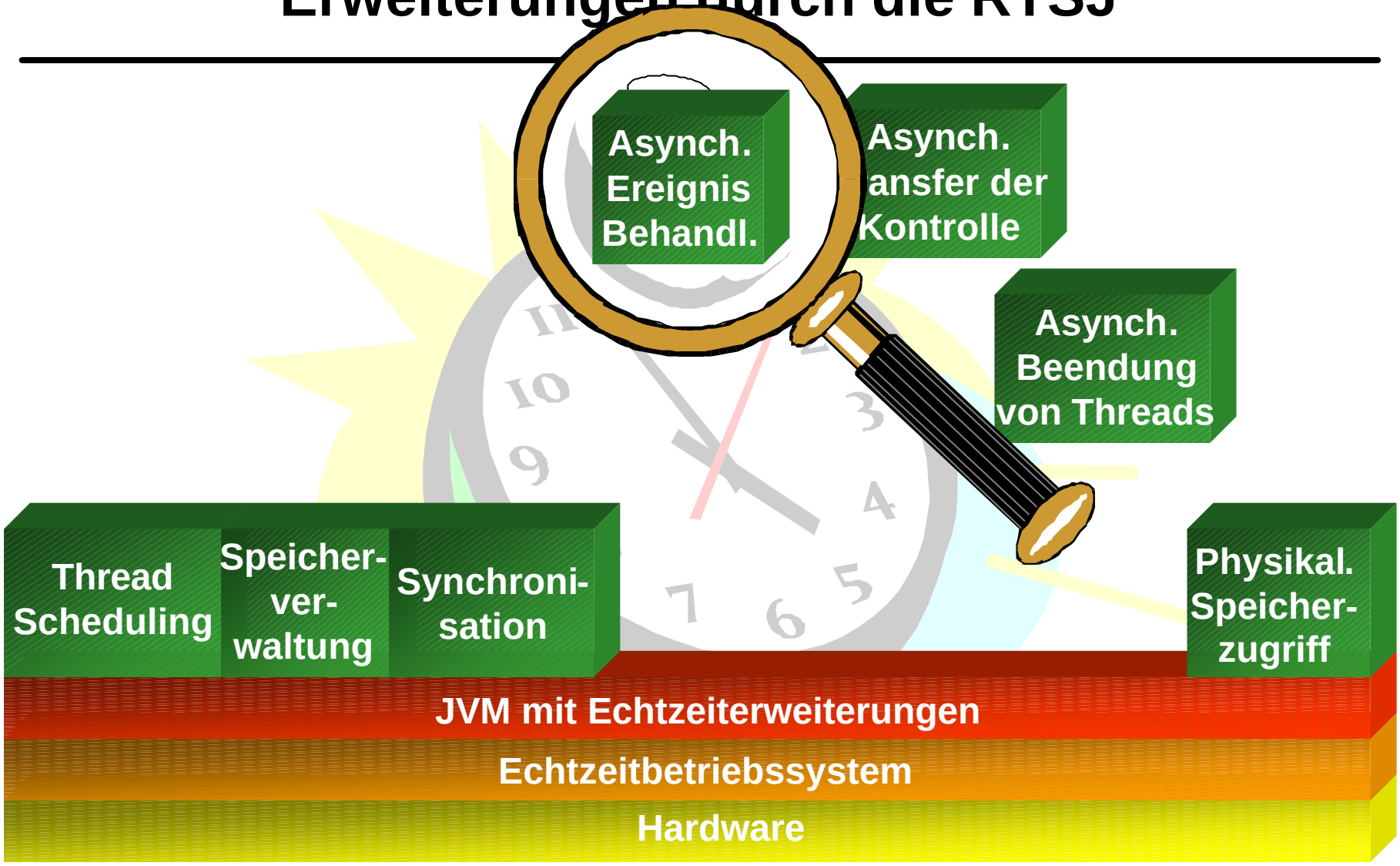
Mechanismen zur Thread-Synchronisierung

- **Implementierung des „Priority Inheritance“-Protokolls ist Standard**
- **„Priority ceiling“-Protokoll verfügbar wenn durch zugrundeliegende Plattform unterstützt**
- **Problem bei der Synchronisierung zwischen `RealtimeThread` und `java.lang.Thread`**



Drei Klassen implementieren wartefreie Schlangen

Erweiterungen durch die RTSJ

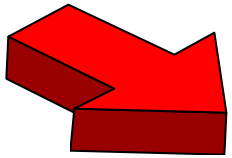


Asynchrone Ereignisbehandlung



Problem:

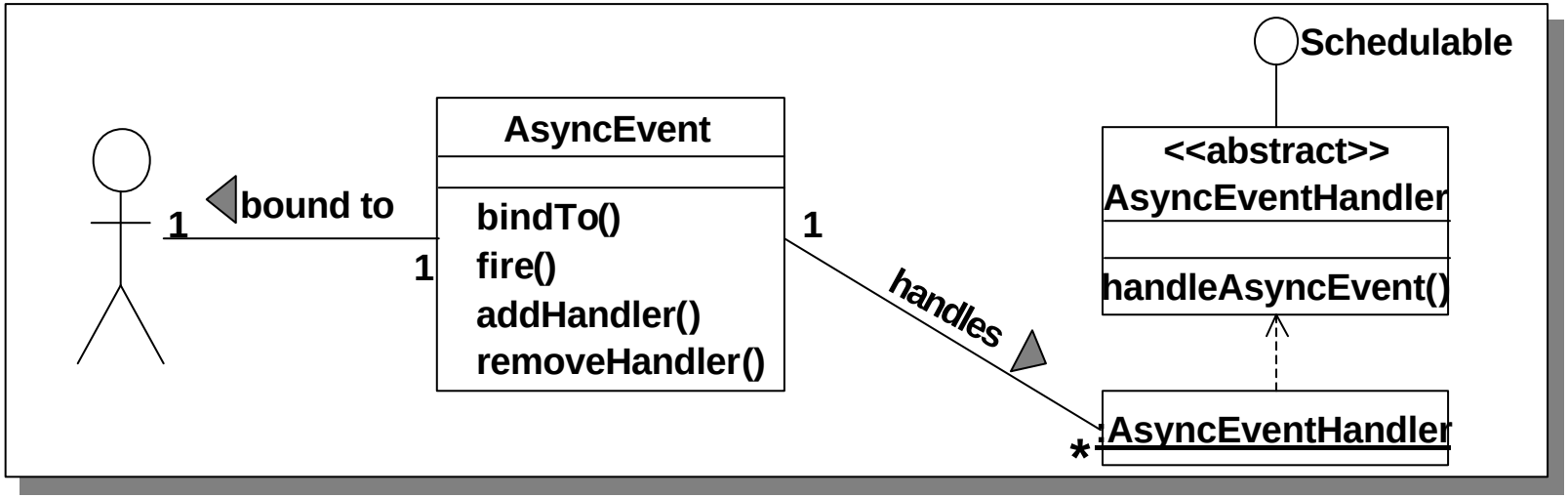
Die Reaktion auf externe Ereignisse wird in aktuellen JVM-Implementierungen nicht effizient unterstützt





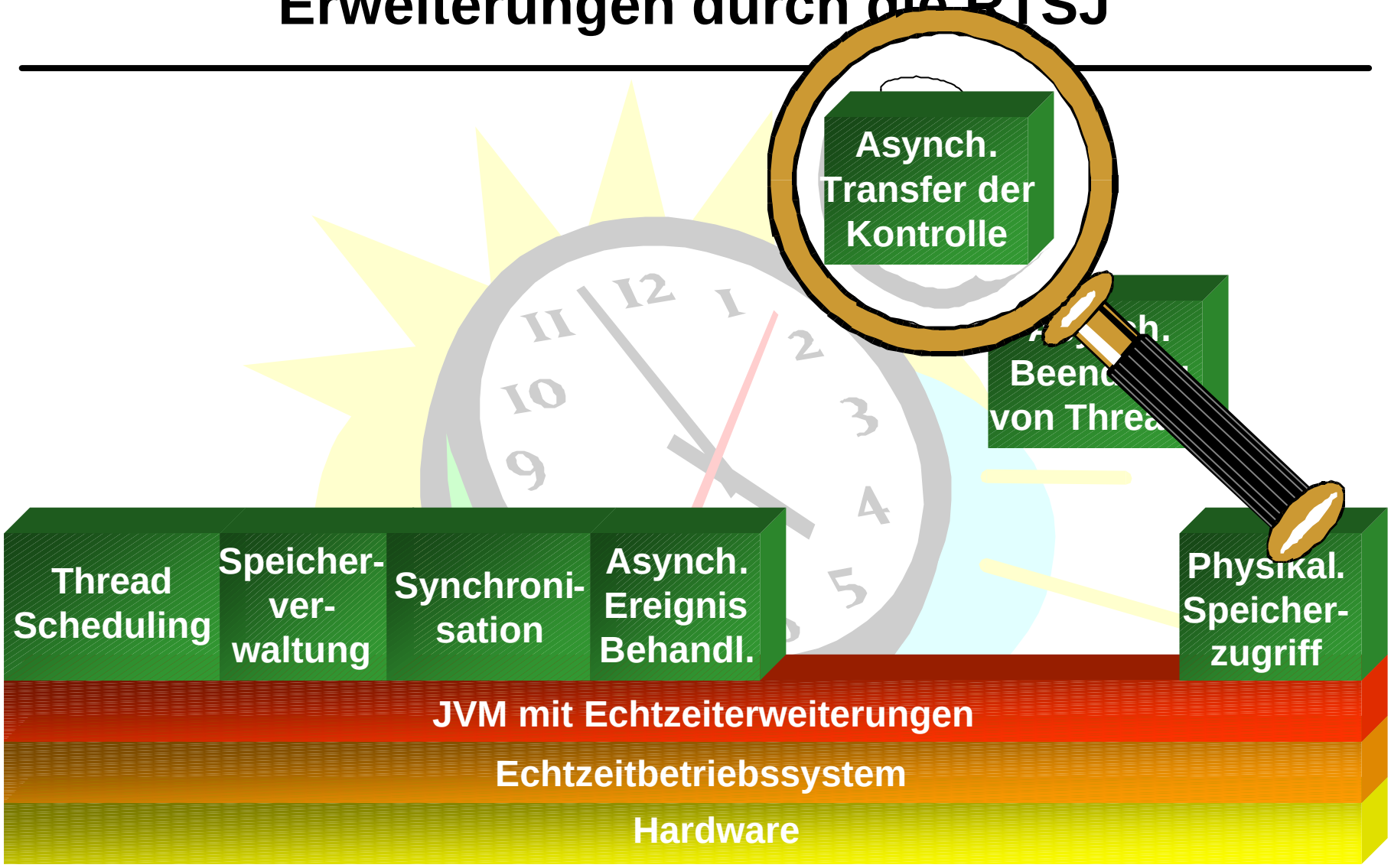
Java-Events mit planbaren Objekten kombinieren

Asynchrone Ereignisbehandlung



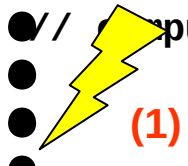
- `AsyncEvent`
  - Instanz an externes Ereignis gebunden
  - Reagiert auf Aufrufe von `AsyncEvent.fire()` oder
  - Wenn ein externes Ereignis auftritt
- `AsyncEventHandlers`
  - Instanz an einen `AsyncEvent` gebunden
  - Zu `RealtimeThread` semantisch äquivalent
  - `handleAsyncEvent()` ist die „run-method“

Erweiterungen durch die RTSJ



Asynchroner Transfer des Kontrollflusses

```
public int computePressure() {
  throws AsynchronouslyInterruptedException {
   int pressure;
       // compute new value of pressure;
       ●
       ●
       ●
       ●
       ●
  return pressure;
 }
```



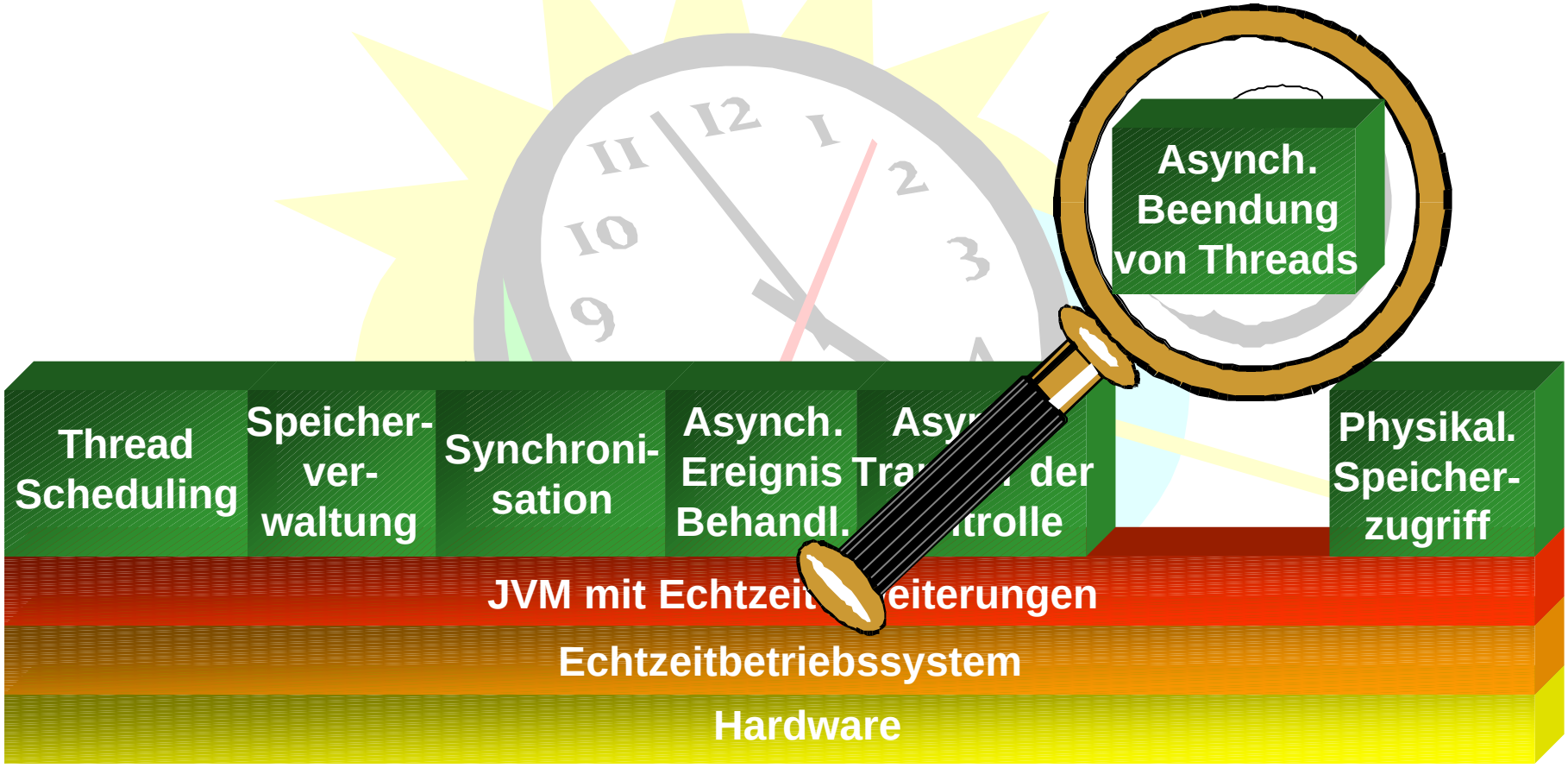
(1)

```
int oldPressure, newPressure;
 try {
  newPressure = computePressure();
 }
catch (AsynchronouslyInterruptedException e){
  // restore value of old Pressure
  (3) go to new thread of control
 }
```

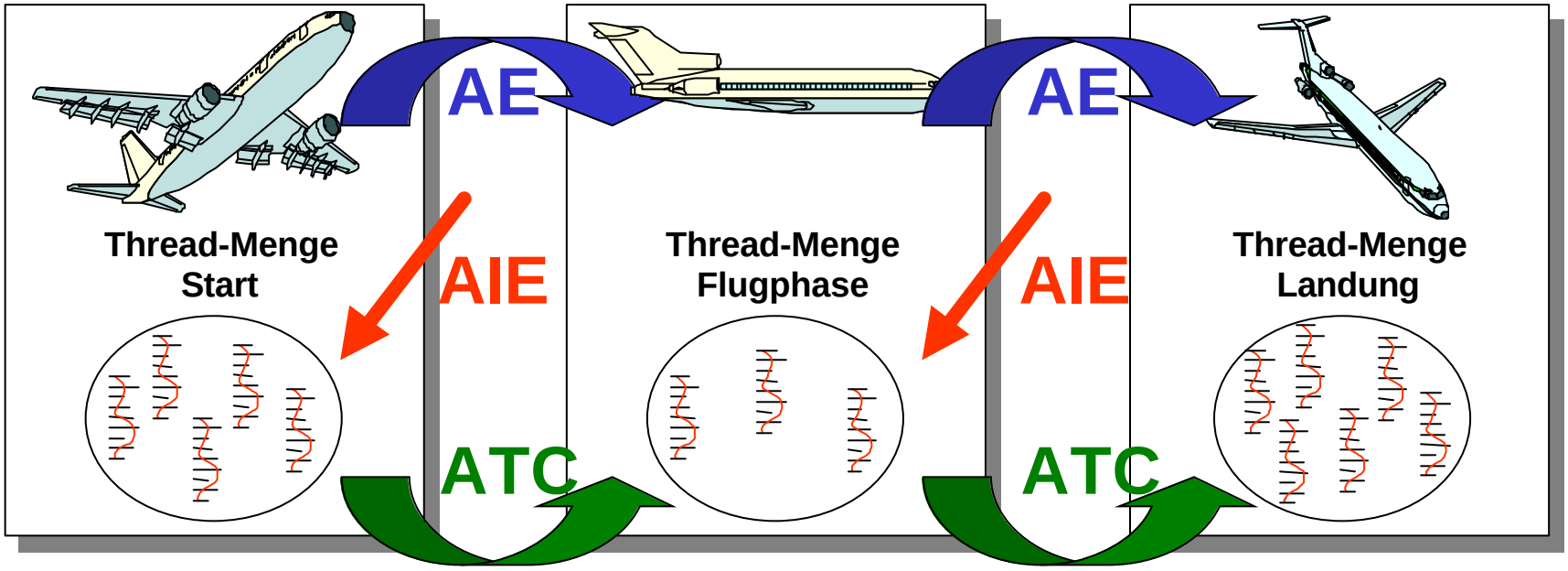
(2)

- **Unbedingter Wechsel des Kontrollflusses**
- **Modelliert durch Ausnahmebehandlung**
  - **Neue Klasse AIE**
- **Keine Rückkehr zum Ausgangspunkt**
- **`synchronized`-Methoden und -Befehle können so nicht unterbrochen werden**

Erweiterungen durch die RTSJ



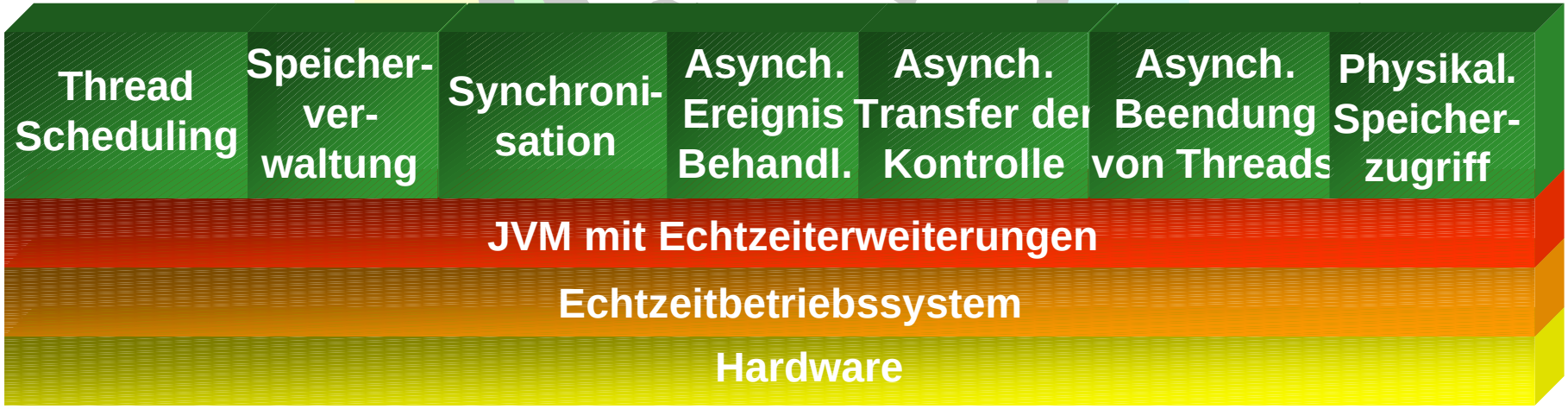
Asynchrone Terminierung von Threads



- `Thread.stop()` und `Thread.destroy()` sind unsicher
- Realisiert durch Kombination asynchroner Ereignisbehandlung und asynchronem Wechsel des Kontrollflusses
- Zustandsübergänge an `AsynEvent` gekoppelt
- `AsyncEventHandler` mit AIE gekoppelt

Zusammenfassung der RTSJ

- Zusätzliche Klassen bilden ein Framework für Echtzeitanwendungen
- Implementierung der JVM wird geändert
- Referenzimplementierung der RTSJ für 4Q 2000 geplant
- Spezifikation von Echtzeit-Java für verteilte Anwendungen wird z.Z. entwickelt



Grenzen reiner Softwarelösungen

- TCP/IP Infrastruktur für die entfernte Steuerung/Kontrolle?
- Sicherheit der Internet-Protokolle?
- Zertifizierung des dynamisch ladbaren Codes?
- Verwaltung verteilter Netzwerkumgebungen?



- Etherneteinheit integrieren
- Java-Bibliotheken zur Verschlüsselung
- Java/Jini auf einem Chip integrieren

Java rückt näher an eingebettete Systeme heran

Zusammenfassung

- **IT-Markt verlangt nach modernen Software-entwurfsmethoden**
- **Java bietet eine sichere und mächtige Software-entwurfsumgebung**
- **Probleme und Lösungsmöglichkeiten für Java**
- **Java 2 Micro Edition für den Verbrauchermarkt**
- **Jini: dynamische,flexible und verteilte Java-Umgebung**
- **Java zur Steuerung/Kontrolle eingebetteter Systeme**
  heute ⇨ Informationen anzeigen
  morgen ⇨ entfernte Zugriffe UND Steuerung

Literatur und Links

- Tim Linholm, Frank Yellin, ***The Java Virtual Machine Specification***, Addison Wesley, 1997.
- Greg Bollella, James Gosling, Benjamin M. Brosgol et.al., ***The Real-time Specification for Java***, Addison-Wesley Longman, Amsterdam, 2000.
- W. Keith Edwards, ***Core JINI***, The Sun Microsystems Press series, Prentice Hall, 1999.

- **Real-Time Java Experts Group:** http://www.rtj.org
- **J-Consortium:** http://www.j-consortium.org
- **Requirements for Real-time Extensions for the Java Platform:** http://www.nist.gov/rt-java