Sicherheitskritische Echtzeitsysteme mit Java

Andy Walter, COO
22.06.2010

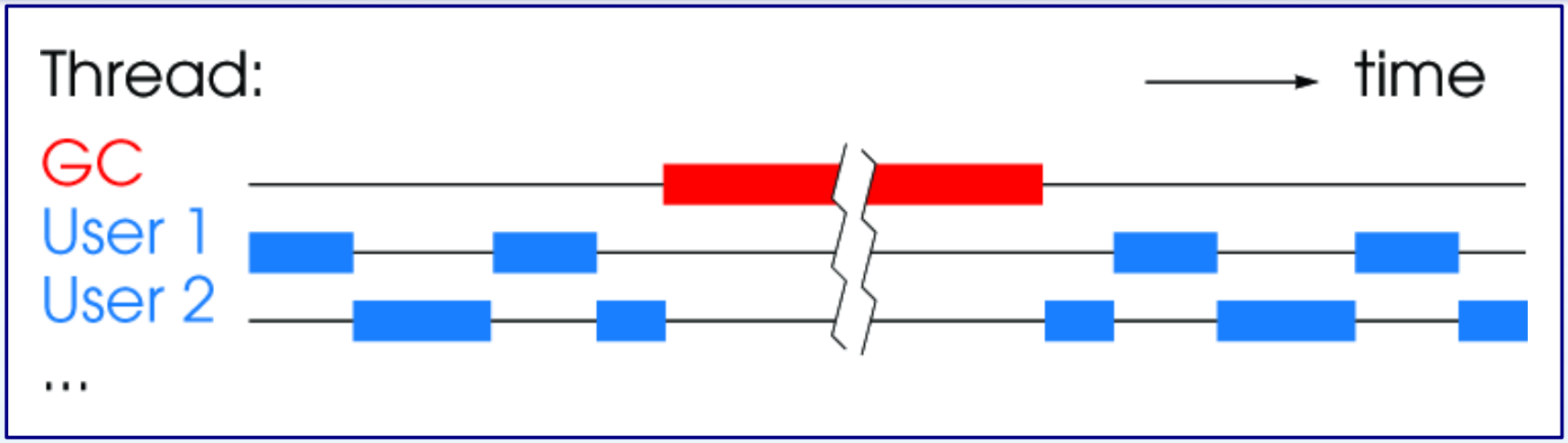
Trends bei kritischer Software

- Komplexität der Anwendungen nimmt zu
- C/C++ fehleranfällig und gefährlich
- Java:
  - Moderne Entwicklungswerkzeuge
  - Sichere Programmiersprache
  - Umfangreiche Standardbibliotheken
- Kann Java für sicherheitskritische Anwendungen zertifiziert werden?

Agenda

- Echtzeit Java Ansätze
  - RTSJ
  - Echtzeit Speicher Verwaltung (Realtime Garbage Collector)
- Java Zertifizierungs Ansätze
  - Safety Critical Java
  - DO-178C

Klassischer Garbage Collector



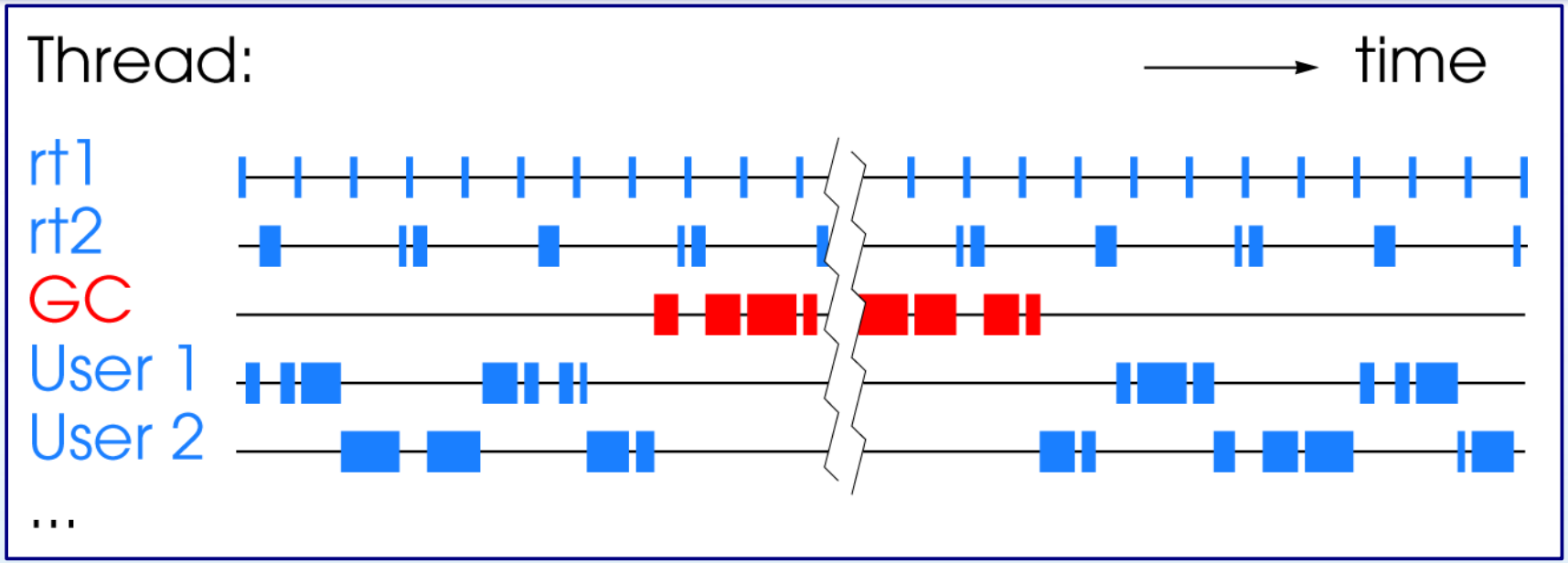
GC kann Ausführung beliebig lange unterbrechen

- lange, unvorhersehbare Pausen

Realtime Specification for Java

- Java Community Standard (JSR 1, JSR 282)
- Größte Verbreitung für Echtzeit Java Anwendungen
- Neues Thread model: NoHeapRealtimeThread
  - Keine Unterbrechung durch Garbage Collector
  - Threads können keine Heap Objekte verwenden
- Keine Berücksichtigung von Zertifizierungsaspekten

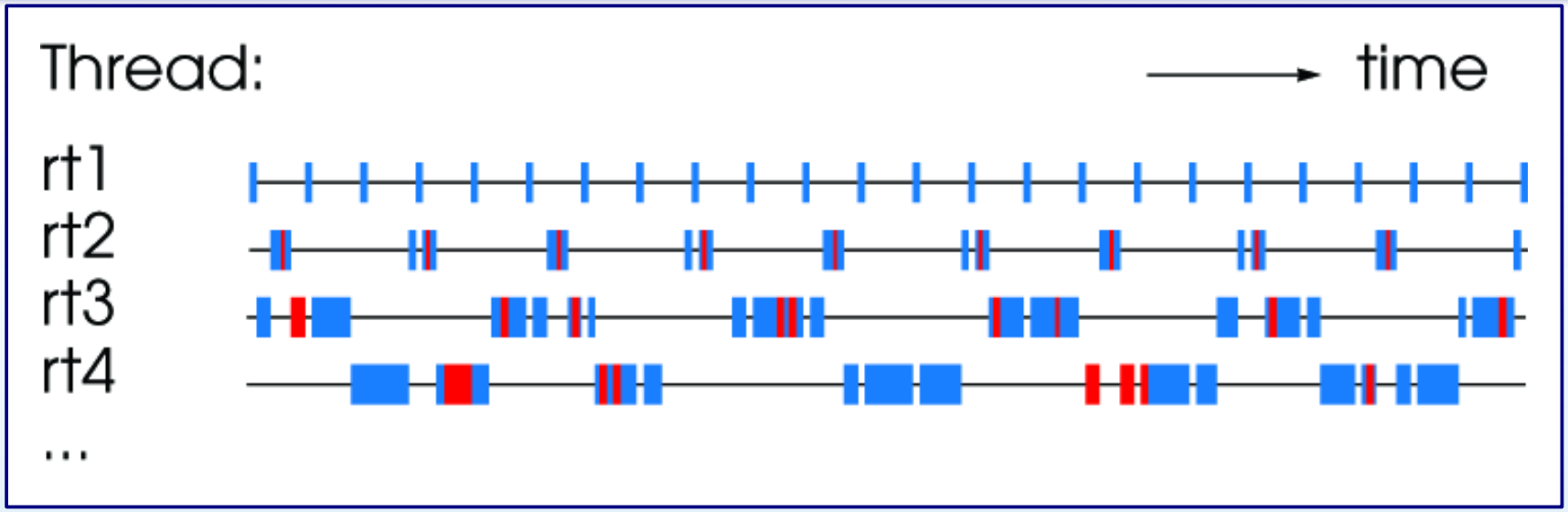
Garbage Collection in RTSJ



Echtzeit Anwendung mit RTSJ:

- Trennung von Echtzeit und Nicht-Echtzeit Teil
- Keine direkte Synchronisation möglich
- Keine Speicherverwaltung für Echtzeit Threads

Echtzeit Garbage Collector



Alle Java Threads sind Echtzeit Threads:

- GC arbeitet nur bei Objekt Allokation
- Obere Zeitschranke existiert für Allokation
- GC ist exakt und deterministisch
- Kommt z.B. bei JamaicaVM zum Einsatz

Agenda

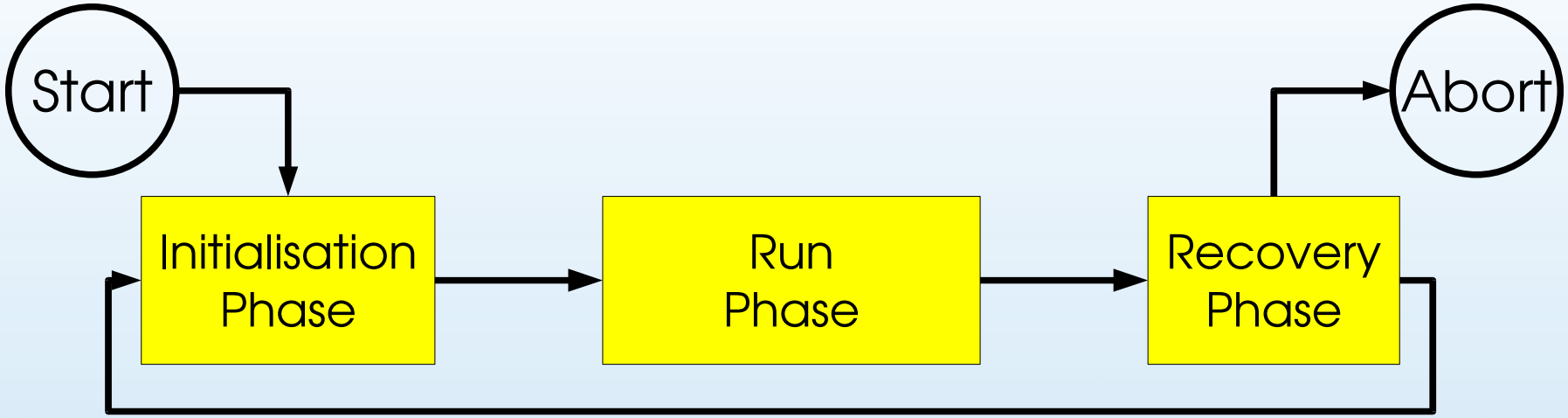
- Echtzeit Java Ansätze
  - RTSJ
  - Echtzeit Speicher Verwaltung (Realtime Garbage Collector)
- Java Zertifizierungs Ansätze
  - Safety Critical Java
  - DO-178C

Safety Critical Java Entwurf

- Entwurf für neuen Java Community Standard (JCP 302)
- Adressiert DO-178B, Level A
- Basiert auf RTSJ Untermenge
- ScopedMemory anstatt Garbage Collection
- Erweitertes Typsystem erleichtert statische Analysen

SC-Java adressiert nicht direkt Automotive. Ansätze sind jedoch übertragbar.

Safety Critical Java Entwurf

- Nur RealtimeThreads sind möglich
- Keine Heap Objekte / keine GC
- Objekt Allokation nur in Initialisierungs Phase
- 

- Feste Thread Prioritäten
- Immer Priority Ceiling Emulation
- OutOfMemoryError kann nicht auftreten

Klassen Bibliotheken für SCJava

- Im Wesentlichen
  - java.lang.*
  - java.lang.reflect.*
  - java.io.*
  - java.net.*
  - java.util.*
  - javax.realtime.*
- Obermengen für DO-178B, Level B und C werden später ergänzt

Neuer Standard DO-178C

- Keine Objekt Orientierung (OOT) in DO-178B
- OOT bringt
  - Wiederverwendbarkeit
  - Erweiterbarkeit
  - Code Effizienz
  - Moderne Programmier Paradigmen
  - Neue Gefahren
- DO-178C enthält Richtlinien zur Zertifizierung von OOT Anwendungen.

SC-205 / WG-71 Ausschuss

- Geleitet von RTCA and EUROCAE
- Neuer Software Standard für Avionik
  - DO-178B/ED-12B: Flight Software Regulations
  - DO-248B/ED-94B: Flight Software Addendum
  - DO-278/ED-109: Ground Support Software

Standard adressiert zivile Luftfahrt. Ansätze sind jedoch auf Automotive übertragbar.

OOT Schlüssel Funktionen

- Nachweis nötig, dass OOT keine Schwachpunkte in der zertifizierten Anwendung einführt
  - Vererbung und Redefinition
  - Polymorphismus
  - Typ Konversion
  - Überladung
  - Exception Behandlung
  - Dynamische Speicher Verwaltung
  - Virtualisierung

Beispiel: Typ Konvertierung

- Schwachpunkte
  - Daten Verlust
  - Daten Korruption oder Exception
- Ziele
  - Typ Konvertierung muss sicher sein
- Datenflussanalyse kann Korrektheit beweisen
- In Java tritt spätestens zur Laufzeit eine Exception auf

Beispiel: Exceptions

- Trennung zwischen Ausnahmebehandlung und normalem Verhalten
- Schwachpunkte
  - Unbehandelte oder falsch behandelte Exceptions
- Ziele
  - Alle Exceptions müssen korrekt behandelt werden
  - Code Abdeckung berücksichtigt Exceptions
- Datenflussanalyse kann nachweisen, dass alle Exceptions behandelt wurden

Dynamische Speicherverwaltung

Schwachpunkte:

1. Mehrdeutige Referenzen
2. Speicherfragmentierung
3. Speicherlöcher
4. zu wenig Heap Speicher
5. Deallokation lebendiger Objekte
6. Verschobene Objekte
7. Indeterministisch Allokation oder Deallokation

Zielsetzung Speicherverwaltung

1. Zentrale Allokationsstelle
2. Verhinderung von Fragmentierung
3. Rechtzeitige Deallokation
4. Ausreichend Speicher
5. Zeiger Sicherheit
6. Atomische Verschiebung von Objekten
7. Deterministisch

Speicher Verwaltung

Technik	Mehrdeutige Referenzen	Speicherfragmentierung	Speicherlöcher	Zu wenig Speicher	Deallokation lebendiger Objekte	verschobene Objekte	Undeterministische Allokation / Deallokation
Manuelle Allokation	✗	?	✗	✗	✗	N/A	✓
Objekt Pool	✗	✗	✗	✗	✗	N/A	✓
Stack Allokation	✗	✓	✓	✗	✗	N/A	✓
Scope Allokation	✓	✓	✓	✗	✗	N/A	✓
Automatische Speicher Bereinigung	✓	✓	✓	✗	✓	✓	✓

✓ = automatisch verwaltet, ✗ = durch die Anwendung
N/A = nicht anwendbar, ? = schwer sicher zu stellen

Vorteile von Java zu C/C++

- Saubere Syntax und Semantik ohne Präprozessor
  - ➔ Bessere Code Analyse Möglichkeiten
- Mehrfachvererbung auf Schnittstellenebene
- Keine Zeigerarithmetik
- Sichere Deallokation
- Eindeutiger Dispatch Stil
- Starkes Typsystem
- Mit RTSJ, wohldefiniertes Task Modell

Sichere Entwicklung mit Java

	C	C++	Java	Java + DFA
Fehlerhafte Typ Konversion	✘	✘	o	✔
Deadlocks	✘	✘	✘	✔
Dangling Pointers	✘	✘	✔	✔
Fragmentierter Speicher	✘	✘	✔	✔
Array Überlauf	✘	✘	o	✔
Race Conditions	✘	✘	✘	✔
Uninitialisierte Variablen	✘	✘	✔	✔
Null Pointer Deref.	✘	✘	o	✔
Division by Zero	✘	✘	o	✔
Exceptions	✘	o	o	✔

✘ Gefahr, **o** Laufzeit Fehler, ✔ Erkannt während Entwicklung

Garbage Collector Zertifizierung

- Nicht mit allen GC möglich
  - Muss determinitisch sein; obere Zeitschranke
  - Muss maximalen Speicherbedarf berücksichtigen
  - Muss Allokationsrate berücksichtigen
- Beispiel: JamaicaVM GC
  - Keine Root Scanning und Kompaktierung
  - Inkementelle Mark und Sweep Schritte auf Blöcken fester Größe
  - Berücksichtigt Allokationsrate
  - GC arbeitet nur zur Allokationszeit
    - andere Threads unbeeinflusst

Zusammenfassung

- Zertifizierung von OOT wirft neue Fragen auf
- Automatische Speicherverwaltung sicherer als händisch (bei komplexen Aufgaben)
- Java macht komplexe Anwendungen zuverlässiger
- Klare Leitlinien für OOT im DO-178C Standard erleichtern Java Zertifizierung
- DO-178C wird GC zulassen
- Übertragung dieser Standards auf Automotive (ISO 26262) erforderlich