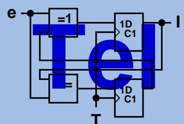
Realisierung einer integrierten Speicherverwaltung mit der Unterstützung schwacher Referenzen für den Prozessor SHAP

– Vortrag zur Diplomarbeit –

Peter Reichel
peter.reichel@mailbox.tu-dresden.de

Technische Universität Dresden
Institut für Technische Informatik



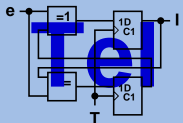
Inhalt

1 Einleitung

2 Stand der Technik

3 Ansatz zur erweiterten Speicherverwaltung

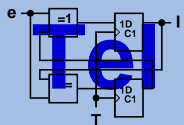
4 Zusammenfassung



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Einleitung

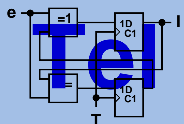
1 Einleitung



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Aufgabenstellung

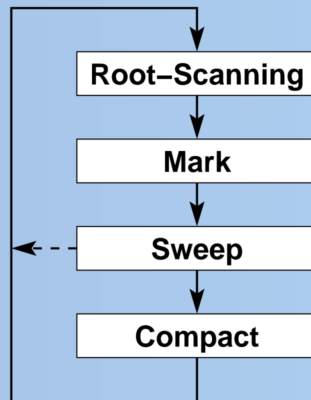
1. **Literaturstudium** zu eingebetteten System mit HW GC.
2. Sondierung der **Ansätze zur Impl. schwacher Referenzen** in Standardsystemen mit GC.
3. **Entwurf von Konzepten** zur Unterstützung erweiterter Phasen des Objektlebenszyklusses.
4. Evaluierung der Realisierungsmöglichkeiten unter Abwägung von Leistungsfähigkeit und Wartbarkeit.
5. Implementierung der erweiterten Speicherverwaltung für die **SHAP-Plattform**.
6. Test der Speicherverwaltung an repräsentativen Beispielen.
7. Zusammenfassung und Dokumentation des Entwurfs und der erzielten Ergebnisse.

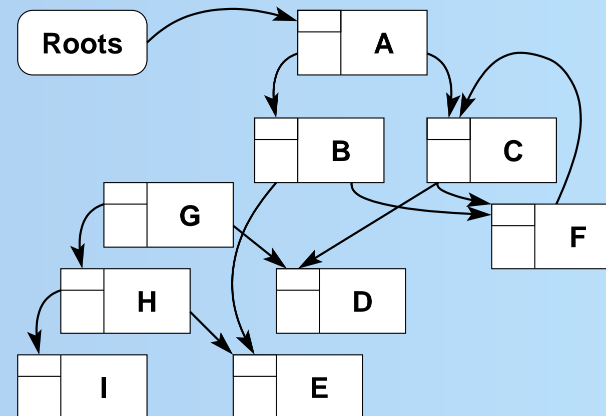


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Garbage-Collection

- **Ziel:** Automatische Freigabe *nicht länger benötigter* Ressourcen
- **Ideal:** Unmittelbare Freigabe nach letztem Zugriff
  $\Rightarrow$ Blick in die Zukunft
- **Real:** Freigabe *nicht referenzierter* Objekte
  $\Rightarrow$ Traversieren des Objektgraphen (*tracing*)

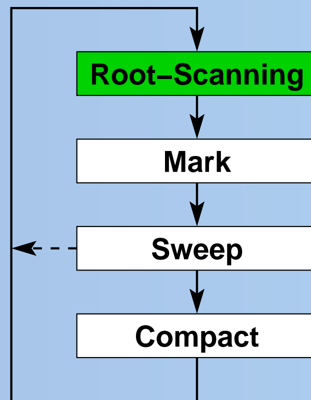


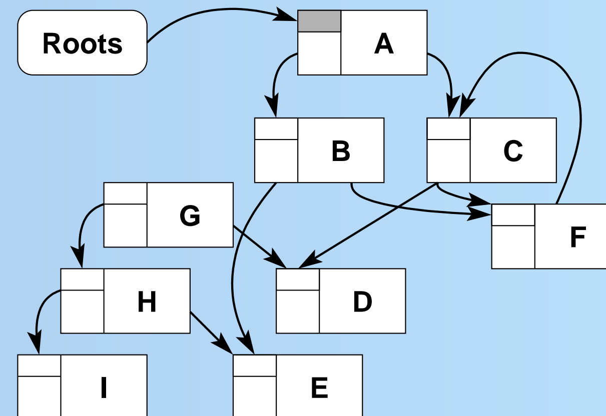


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Garbage-Collection

- **Ziel:** Automatische Freigabe *nicht länger benötigter* Ressourcen
- **Ideal:** Unmittelbare Freigabe nach letztem Zugriff
  $\Rightarrow$ Blick in die Zukunft
- **Real:** Freigabe *nicht referenzierter* Objekte
  $\Rightarrow$ Traversieren des Objektgraphen (*tracing*)

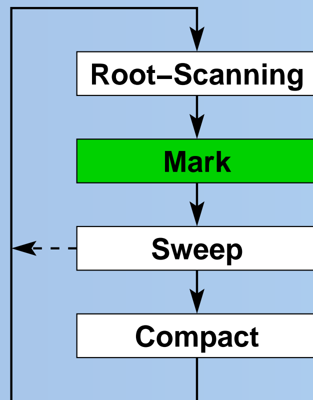


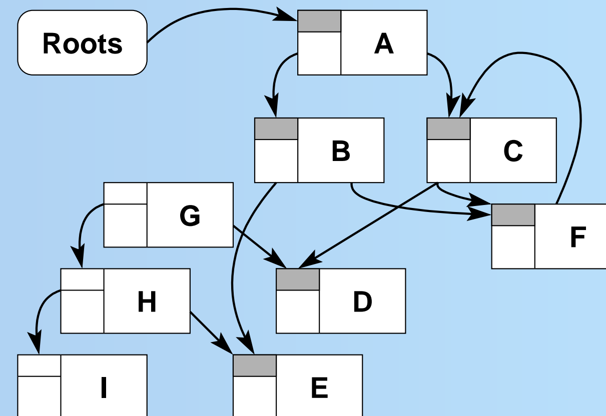


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Garbage-Collection

- **Ziel:** Automatische Freigabe *nicht länger benötigter* Ressourcen
- **Ideal:** Unmittelbare Freigabe nach letztem Zugriff
  $\Rightarrow$ Blick in die Zukunft
- **Real:** Freigabe *nicht referenzierter* Objekte
  $\Rightarrow$ Traversieren des Objektgraphen (*tracing*)

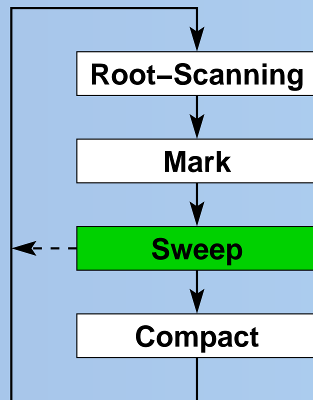


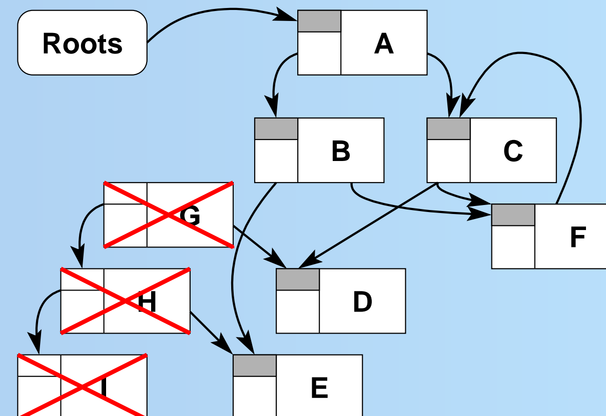


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Garbage-Collection

- **Ziel:** Automatische Freigabe *nicht länger benötigter* Ressourcen
- **Ideal:** Unmittelbare Freigabe nach letztem Zugriff
  $\Rightarrow$ Blick in die Zukunft
- **Real:** Freigabe *nicht referenzierter* Objekte
  $\Rightarrow$ Traversieren des Objektgraphen (*tracing*)

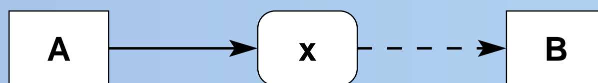




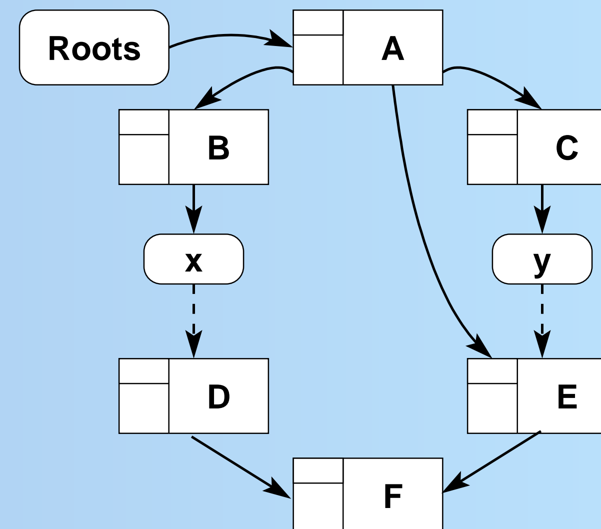
Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen (1)

- **Idee:** werden während Tracing *nicht* verfolgt
  $\Rightarrow$ verhindern nicht die Freigabe durch den GC!
- **Wiederbelebung** (*Resurrection*):
  erzeugen einer *starken* Referenz möglich, wenn Objekt noch nicht freigegeben wurde
- realisiert über Referenz- oder Proxy-Objekte



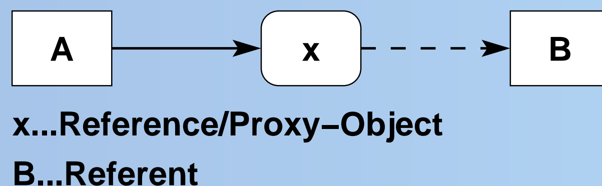
x...Reference/Proxy–Object
B...Referent

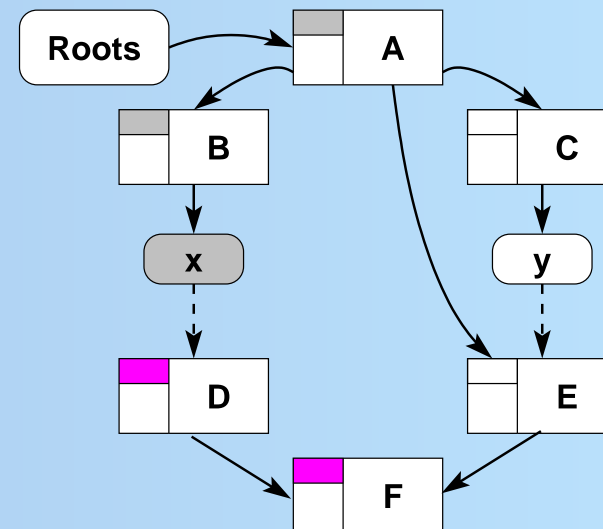


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen (1)

- **Idee:** werden während Tracing *nicht* verfolgt
  $\Rightarrow$ verhindern nicht die Freigabe durch den GC!
- **Wiederbelebung** (*Resurrection*):
  erzeugen einer *starken* Referenz möglich, wenn Objekt noch nicht freigegeben wurde
- realisiert über Referenz- oder Proxy-Objekte

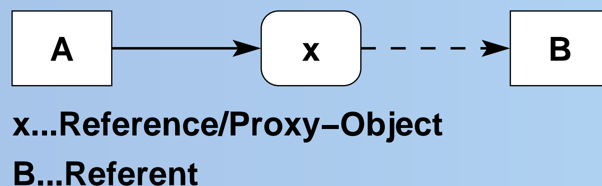


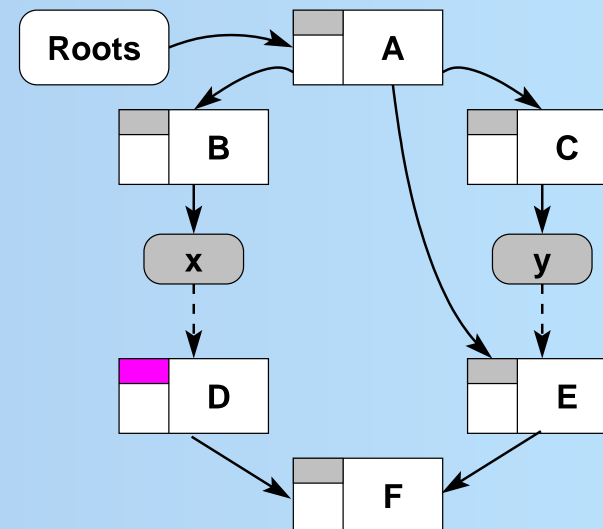


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen (1)

- **Idee:** werden während Tracing *nicht* verfolgt
  $\Rightarrow$ verhindern nicht die Freigabe durch den GC!
- **Wiederbelebung** (*Resurrection*):
  erzeugen einer *starken* Referenz möglich, wenn Objekt noch nicht freigegeben wurde
- realisiert über Referenz- oder Proxy-Objekte

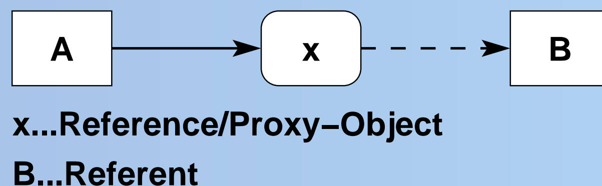


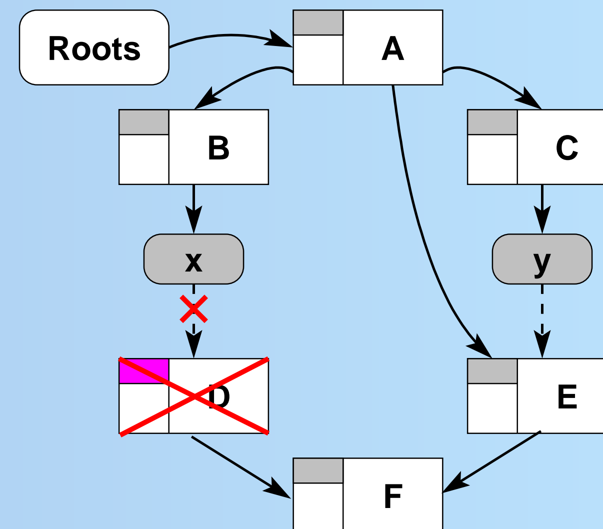


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen (1)

- **Idee:** werden während Tracing *nicht* verfolgt
  $\Rightarrow$ verhindern nicht die Freigabe durch den GC!
- **Wiederbelebung** (*Resurrection*):
  erzeugen einer *starken* Referenz möglich, wenn Objekt noch nicht freigegeben wurde
- realisiert über Referenz- oder Proxy-Objekte

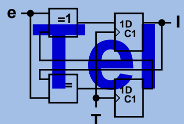




Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen (2)

- Benachrichtigung über Freigabe *beliebiger* Objekte
- können *Finalizer* in vielen Fällen ersetzen
- effektive Verwaltung von Ressourcen
  - schließen geöffneter Dateien
  - beenden von Netzwerkverbindungen
- **aber:** erheblicher Mehraufwand für GC!



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Stand der Technik

2 Stand der Technik

Schwache Referenzen in Java

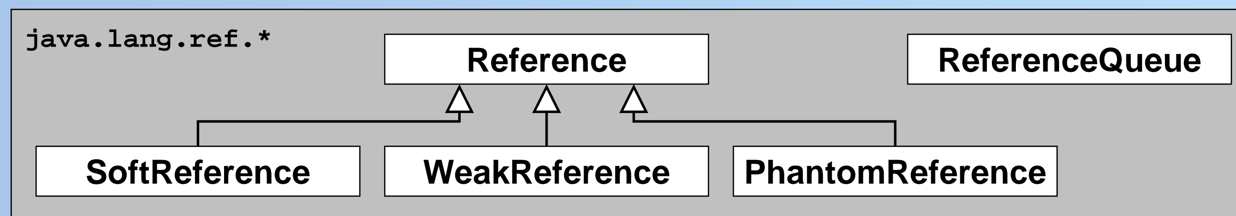
- Basisklasse `java.lang.ref.Reference`
- Anlegen einer schwachen Referenz:

```
Object referent = new Object();
ReferenceQueue q = new ReferenceQueue();
Reference ref = new WeakReference(referent, q);
referent = null;
```

- Erzeugen einer starken Referenz mit `Reference.get()`
- Benachrichtigung über Freigabe:

  $\Rightarrow$ `Reference.get()` liefert `null`

  $\Rightarrow$ Einordnung des `Reference` Objekts in `ReferenceQueue`



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen in Java

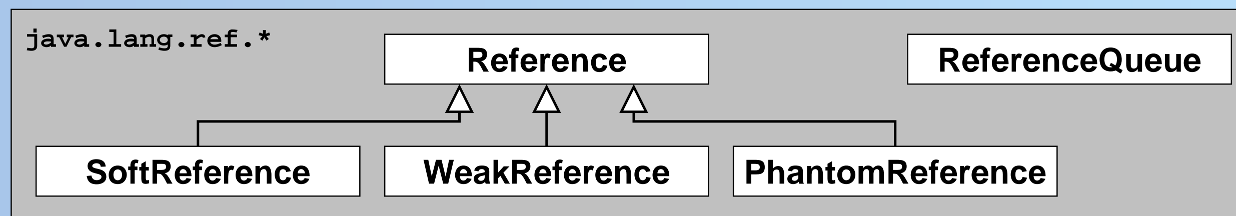
- Basisklasse `java.lang.ref.Reference`
- Anlegen einer schwachen Referenz:

```
Object referent = new Object();
ReferenceQueue q = new ReferenceQueue();
Reference ref = new WeakReference(referent, q);
referent = null;
```

- Erzeugen einer starken Referenz mit `Reference.get()`
- Benachrichtigung über Freigabe:

  ⇒ `Reference.get()` liefert `null`

  ⇒ Einordnung des `Reference` Objekts in `ReferenceQueue`



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Referenzen in Java

- Basisklasse `java.lang.ref.Reference`

- Anlegen einer schwachen Referenz:

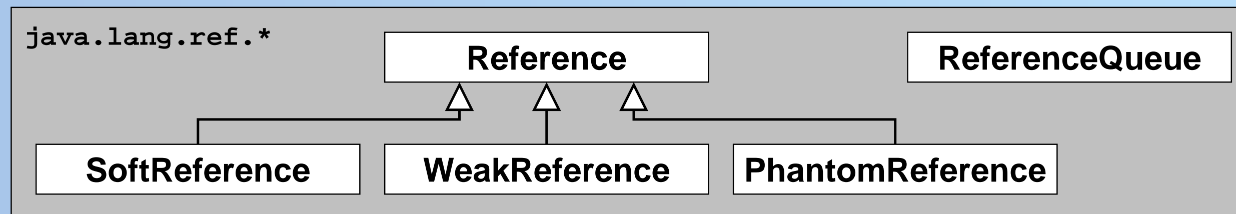
```
Object referent = new Object();
ReferenceQueue q = new ReferenceQueue();
Reference ref = new WeakReference(referent, q);
referent = null;
```

- Erzeugen einer starken Referenz mit `Reference.get()`

- Benachrichtigung über Freigabe:

  ⇒ `Reference.get()` liefert `null`

  ⇒ Einordnung des `Reference` Objekts in `ReferenceQueue`

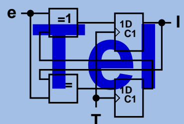


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Schwache Ref. im eingebetteten Bereich

Kaum Unterstützung für schwache Ref.

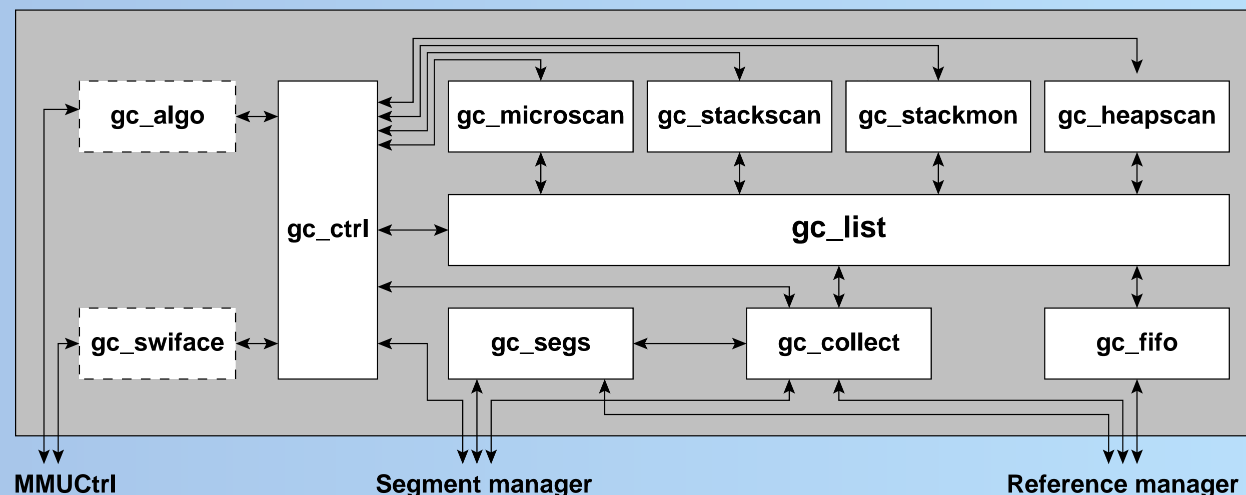
- JVMs für *general-purpose* Prozessoren meist nicht RT fähig
  - SUNs *KVM* (verwendet in zahlreichen Mobiltelefonen)
  - Squawk JVM (Forschungsprojekt von SUN Microsystems)
- von aktuellen Java-Prozessoren kaum unterstützt
  - JOP (Java Optimized Processor, TU Wien)
  - Komodo (Universität Augsburg)
  - aJile aJ-100 (aJile Systems Inc.)
- keine Unterstützung in Hardware-GCs
  - Meyer (2005): An On-Chip GC-Coprocessor for Embedded RT-Systems
  - Witawas Srisa-an (2003): Active Memory Processor
  - Gruian (2005): Designing a Concurrent Hardware Garbage Collector for Small Embedded Systems



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Momentaner Garbage Collector (1)

- Bestandteil der MMU
- Traversieren des Objektgraphen (*Mark-and-Sweep*)
- Freigabe durch Verschieben überlebender Objekte
- alle Komponenten in Hardware implementiert
  $\Rightarrow$ arbeitet vollständig parallel zum System

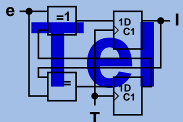


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Momentaner Garbage Collector (2)

Probleme:

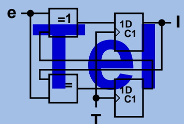
- relativ starre Struktur, schwer erweiterbar
- konservativer Scan des Heaps
- keine Unterstützung schwacher Referenzen
- wenig Interaktion mit dem Laufzeitsystem
- Erweiterung auf mehr als einen SHAP-Core nicht möglich



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Ansatz zur erweiterten Speicherverwaltung

3 Ansatz zur erweiterten Speicherverwaltung

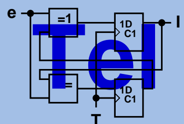


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Ziele

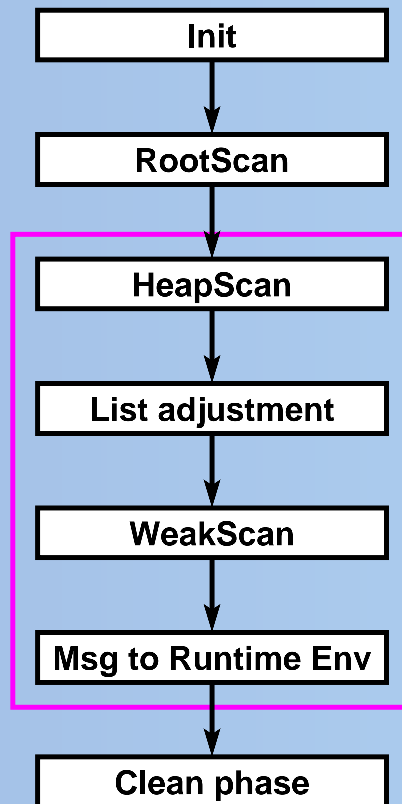
Ziele:

1. Steuerung in Software mittels separatem 32 Bit-Prozessor
2. Unterstützung der verschiedenen Typen schwacher Referenzen
3. bessere Interaktion mit dem Laufzeitsystem
4. exakter Scan des Heaps
5. Unterstützung mehrerer SHAP-Cores mit gemeinsamem Speicher



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Ablauf der Collection

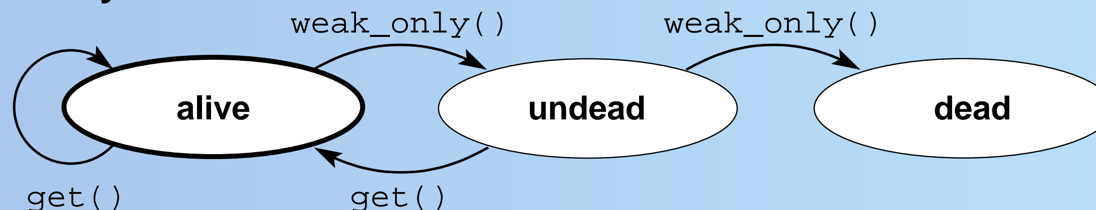


- typischer *Mark-and-Sweep* Ablauf
- Besonderheiten für schwache Referenzen:
  - während HeapScan:
    - ▷ nur stark erreichbare Objekte markieren
    - ▷ Liste schwacher Referenzen aufbauen
  - Abgleich der Liste mit Markierungen
    $\Rightarrow$ nur schwach erreichbar?
  - *WeakScan*:
    Scan der nur schwach erreichbaren Objekte
  - Mitteilung über alle nur schwach erreichbaren Objekte an Laufzeitsystem

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Behandlung schwacher Referenzen

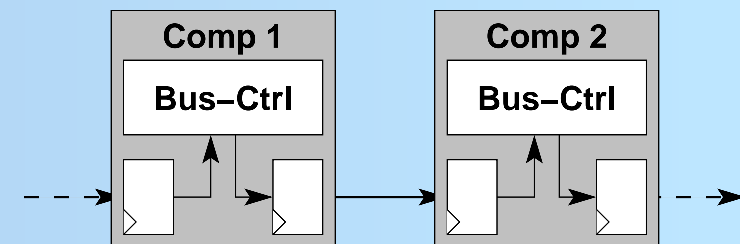
- separater Thread: *Reference-Handler*
- `Reference` hat internen *Erreichbarkeits-Zustand*
  - Anfangszustand: Alive
  - bei Übergang zu Dead: löschen der Referenz
- *Reference-Handler:* Aufruf von `Reference.only_weak()`
  - `synchronized` mit `Reference.get()`
  - löst Zustandsübergang aus
  - Wiederbelebung durch Aufruf von `get()`



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

GC-Bus

- **Ziel:** leicht auf mehrere SHAP-Cores erweiterbar
- **Idee:** *Daisy-chain*-artige Verkettung der Komponenten
- Übertragung von Befehlen und Daten zwischen Komponenten
  - Befehle: z.B. Initialisierung, Objekt scannen / verschieben
  - Daten: z.B. Referenzen, Markierungsbits
- Komponenten:
  - Prozessor zur Steuerung
  - Bildung des Root-Sets für jeden Core
  - Zugriff auf Heap (*HeapAccess*)
  - . . .



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

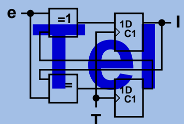
Steuerung in Software

- Verwendung eines separaten 32 Bit-Prozessors
  ⇒ möglichst „klein"
  ⇒ nicht spezifisch für FPGA eines Herstellers
  ⇒ C Compiler
- zwei mögliche Kanditaten ausgewählt:

ZPU[1]	**OpenFire**
Stackmaschine, eigener Befehlssatz	MicroBlaze-Clone (DLX)
eigene GCC-Toolchain	MicroBlaze GCC-Toolchain
992 LUTs (Spartan-3)	1414 LUTs (Spartan-3)[2]

[1] Architektur verändert, um mit weniger Programmspeicher auszukommen

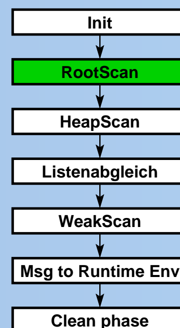
[2] davon 256 für Register-set

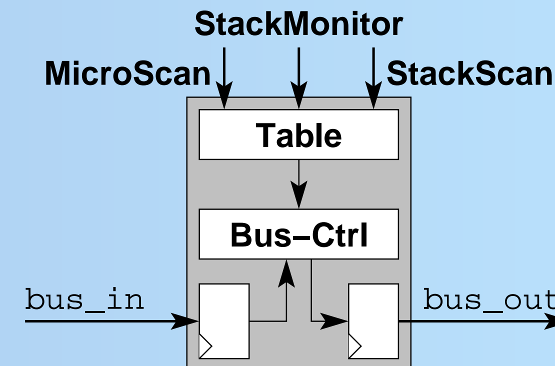


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Root-Scan

- *Rootset* für jeden SHAP-Core zusammengesetzt aus:
  1. Microcode-Variablen ⇒ *MicroScan*
  2. Stack ⇒ *StackScan*
- Tabelle für Markierungen
- Abfrage der Markierungen über GC-Bus

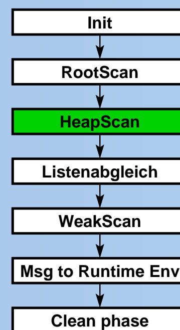


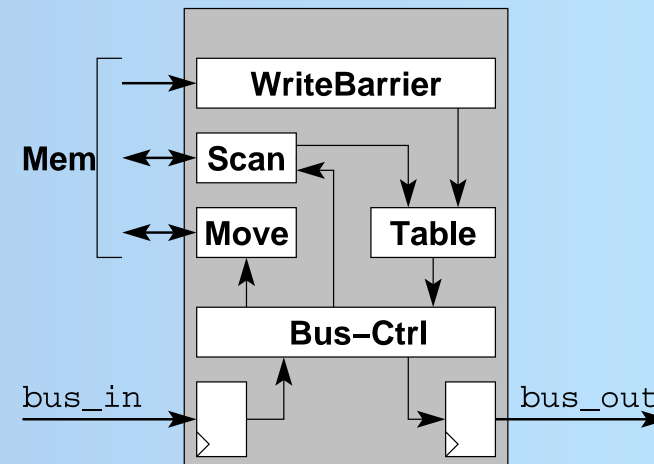


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Heap-Access

- Objekte scannen und verschieben
- Erfassen aller
  - *neu angelegten* Objekte sowie
  - aller überschriebenen Referenzen (*Write-Barrier*) im Heap
- Tabelle für Markierungen
- GC-Bus: Scannen / Verschieben, Abfrage von Markierungen

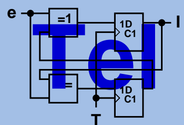




Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Zusammenfassung

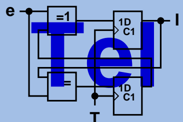
4 Zusammenfassung



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Zusammenfassung

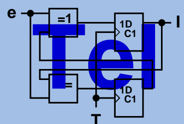
- **Ziel:** erweiterte Speicherverwaltung mit schwachen Referenzen
- leistungsfähiges Werkzeug zur effektiven Verwaltung von Ressourcen
- wenig Unterstützung für schwache Referenzen im eingebetteten Bereich
- erweiterte Speicherverwaltung:
  - vollständig parallel zum Systemkern
  - Steuerung durch separaten Prozessor
  - Behandlung schwacher Referenzen mit Hilfe des Laufzeitsystems
  - Möglichkeit der Erweiterung auf mehrere SHAP-Cores gegeben



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Danke

Vielen Dank für Ihre Aufmerksamkeit!



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de