Ein konfigurierbarer, visueller Cache-Simulator unter spezieller Berücksichtigung komponenten-basierter Modellierung mit Java Beans

Holger Morgenstern

6. März 2001

Universität Tübingen
Wilhelm-Schickard-Institut für Informatik

Inhalt

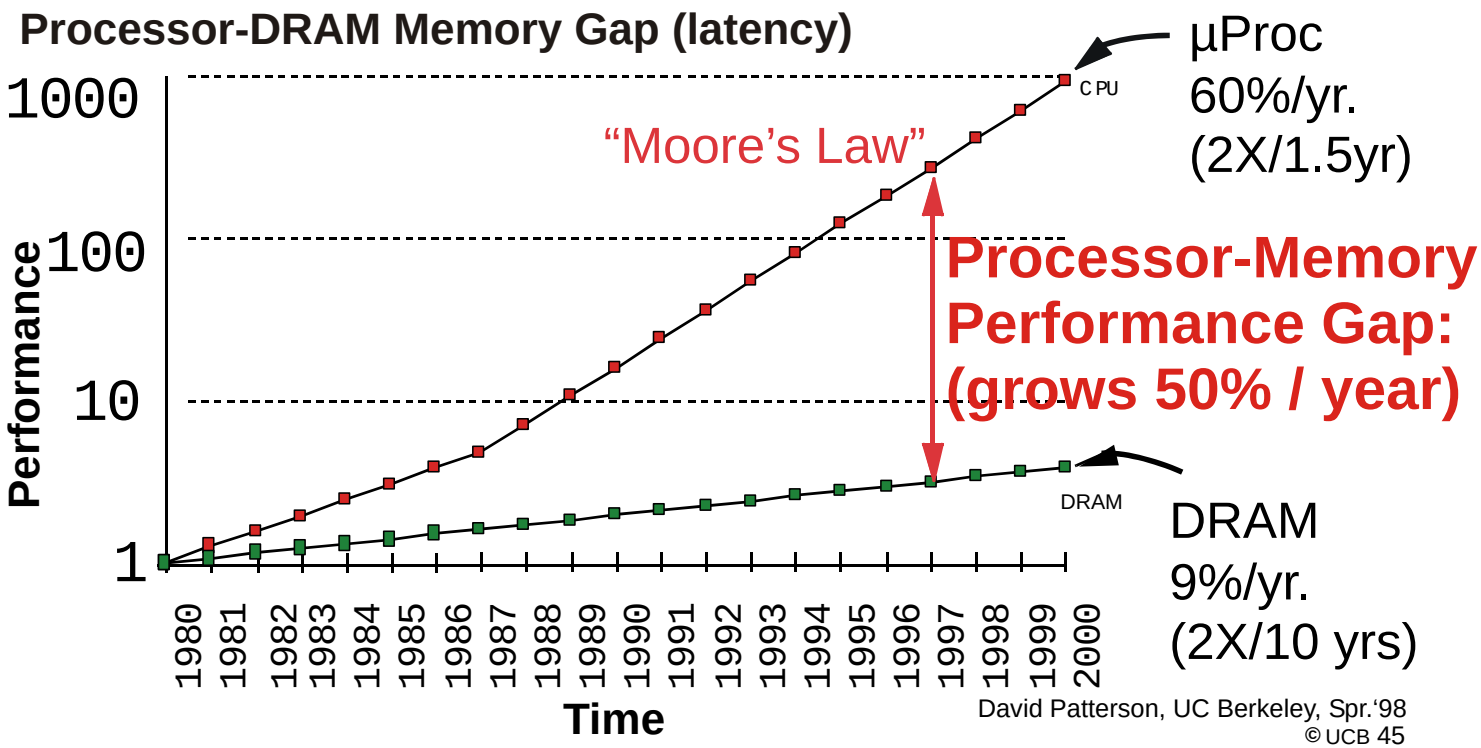
- **Ziele dieser Arbeit**
- **Cache-Speicher**
  - Grundidee
  - Designparameter
- **Realisierung**
  - JavaBeans™
  - Abstrakter Aufbau eines Simulators
  - Eventmodell
  - Simulator Beans
- **Anwendungen**
  - Klassenbibliothek
  - Grafische Entwicklungsumgebungen
- **Zusammenfassung**

Ziele dieser Arbeit

- **Entwicklung eines Softwarepakets zum Aufbau von Cache-Simulatoren**
  - komponentenbasiert
  - erweiterbar
  - einfache Anwendbarkeit
  - visuelle Simulation
  - trace-gesteuerte Simulation
  - einsetzbar in größeren Simulationsmodellen

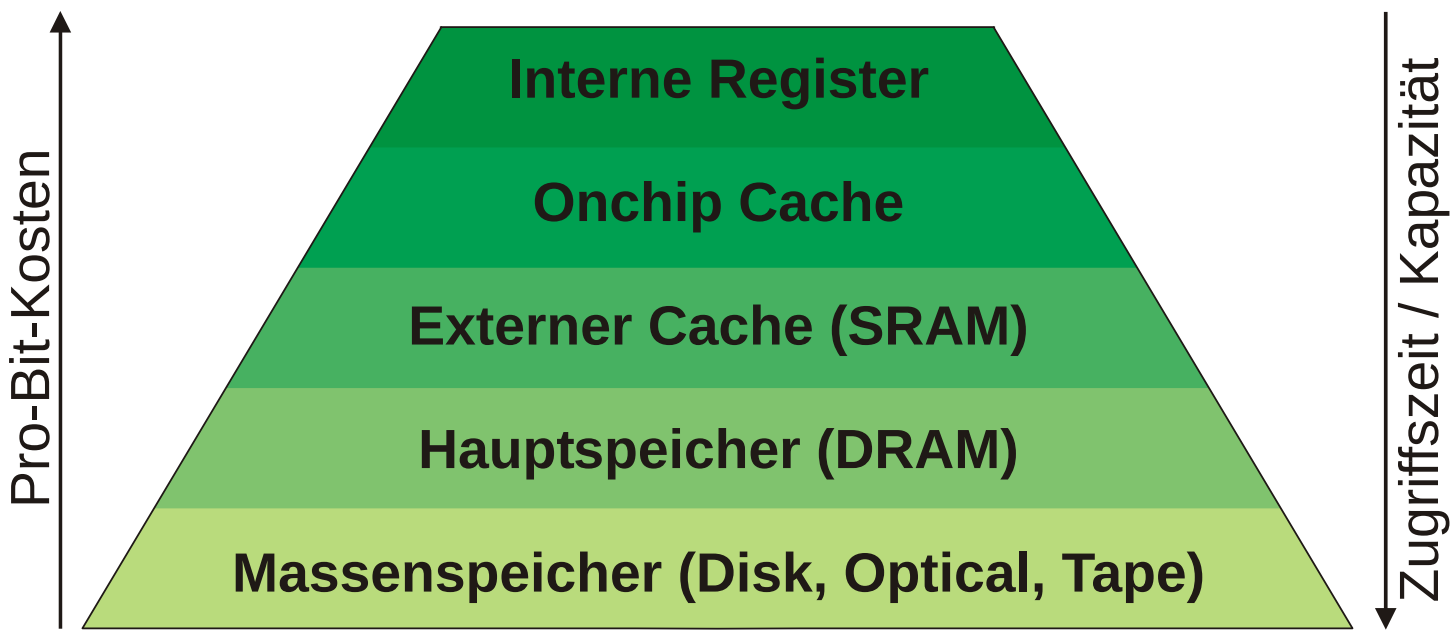
Cache-Speicher

- **Gewünscht: möglichst großer Speicher**
  - Preis der günstigsten Technologie (DRAM)
  - Zugriffszeit der schnellsten Technologie (Register,SRAM)
  - Überbrückung von Geschwindigkeitslücken z.B.:



Cache-Speicher

- **Lösung: Cache als transparentes Bindeglied zwischen unterschiedlich schnellen Medien einer Speicherhierachie**

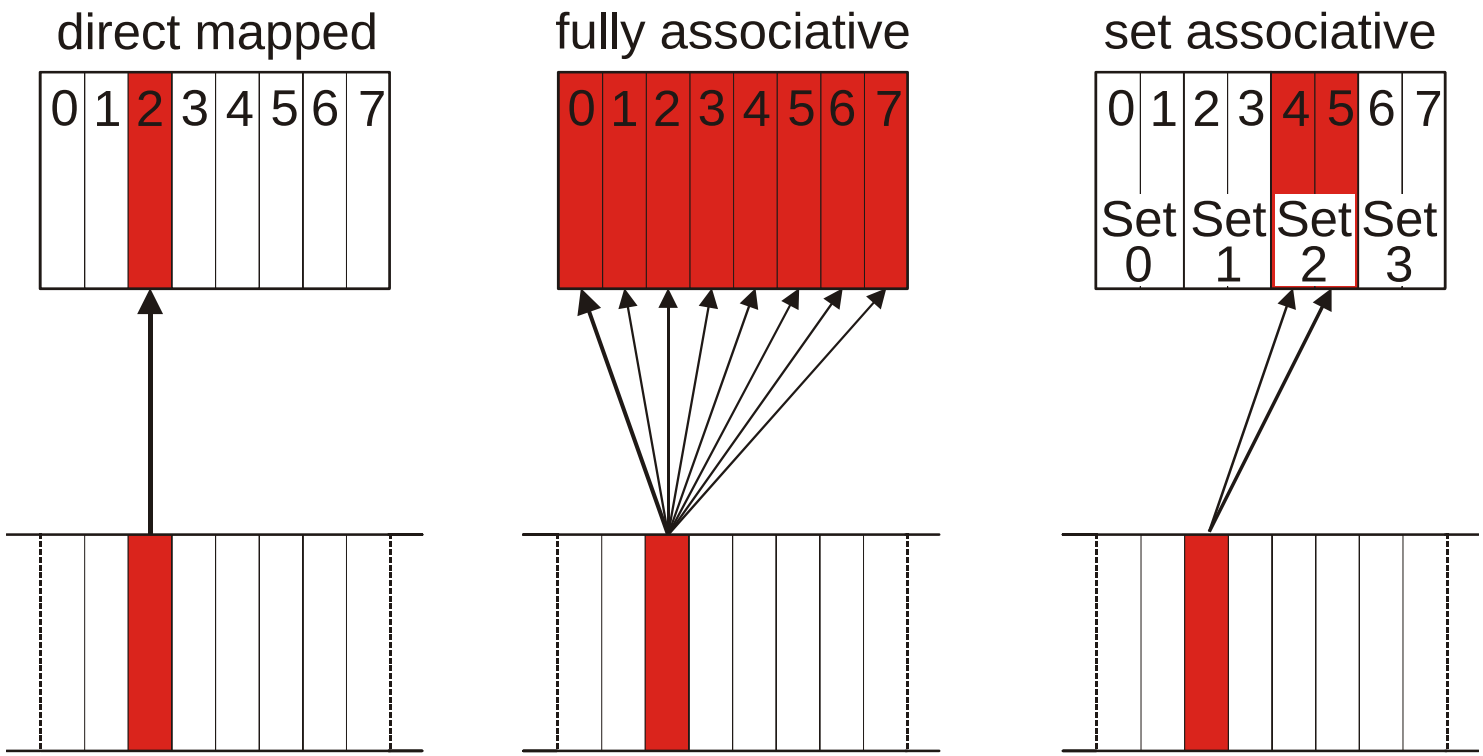


Cache-Speicher

- **Cache = schneller Zwischenspeicher für am wahrscheinlichsten referenzierte Daten**
- **Lokalitätseigenschaft von Programmen**
  - 80% Ausführungszeit i.a. nur 10% Instruktionen
  - räumlich: sequentieller Code, Arrays
  - zeitlich: Programmschleifen, Stacks
- **Terminologie**
  - *Zeile* (*block, line*): kleinste Verwaltungseinheit im Cache
  - *hit*: Datum im Cache gefunden, *miss*: Datum nicht im Cache
  - *miss rate*: Anteil Zugriffe, die zu einen *miss* führen
  - *miss penalty*: zusätzliche Zeit zur Bearbeitung eines *miss*
  - *tag*: Teil einer Speicheradresse zur Identifikation des aktuellen Inhalts einer Cache Zeile

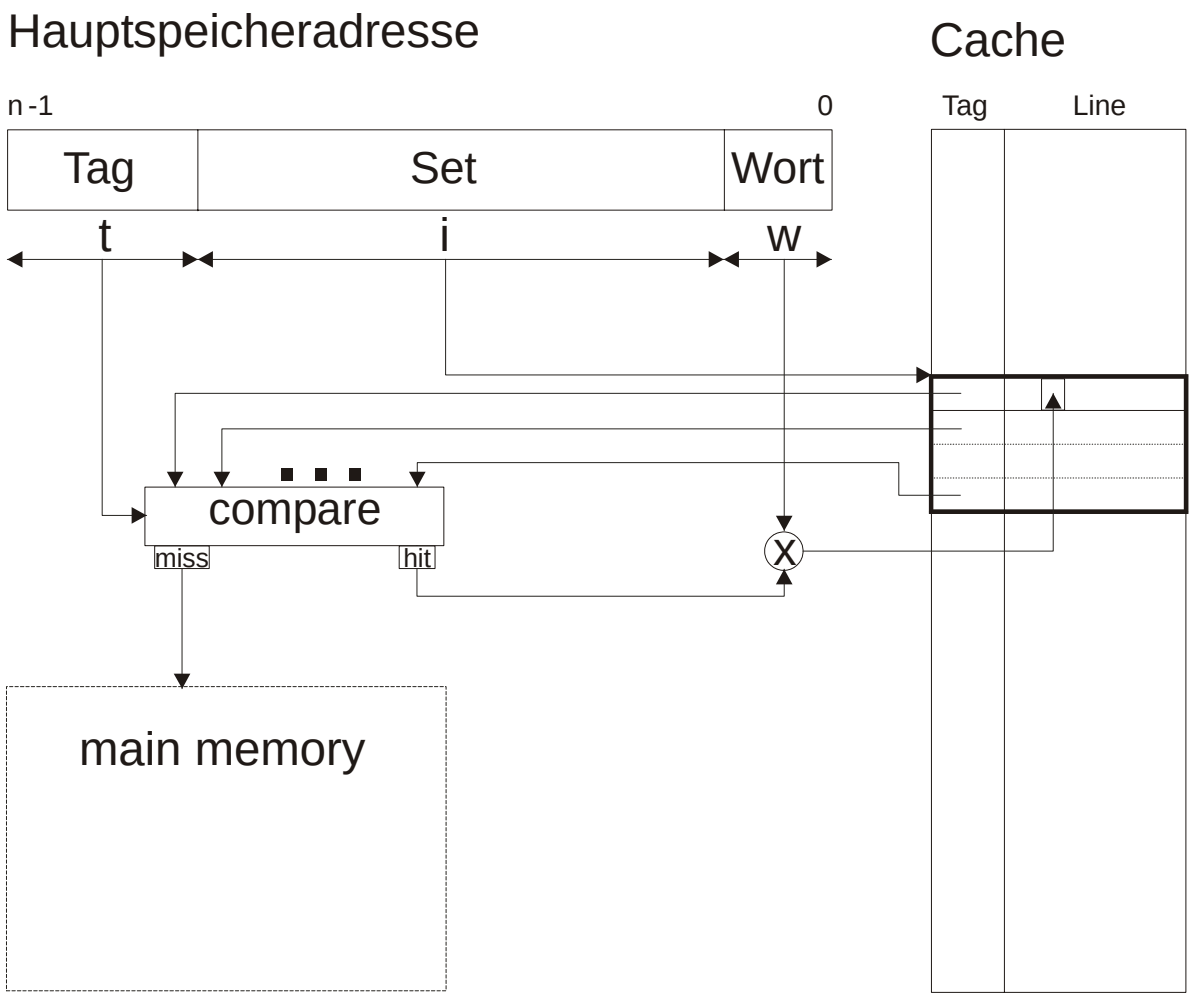
Cache-Speicher

- **Wo wird ein ein Block gespeichert? (*mapping*)**
  - *direct mapped*: in genau einer Zeile
  - *fully associative*: in jeder Zeile
  - *set associative*: in jeder Zeile des Sets



Cache-Speicher

- **set associative mapping, Cachezugriff**



Cache-Speicher

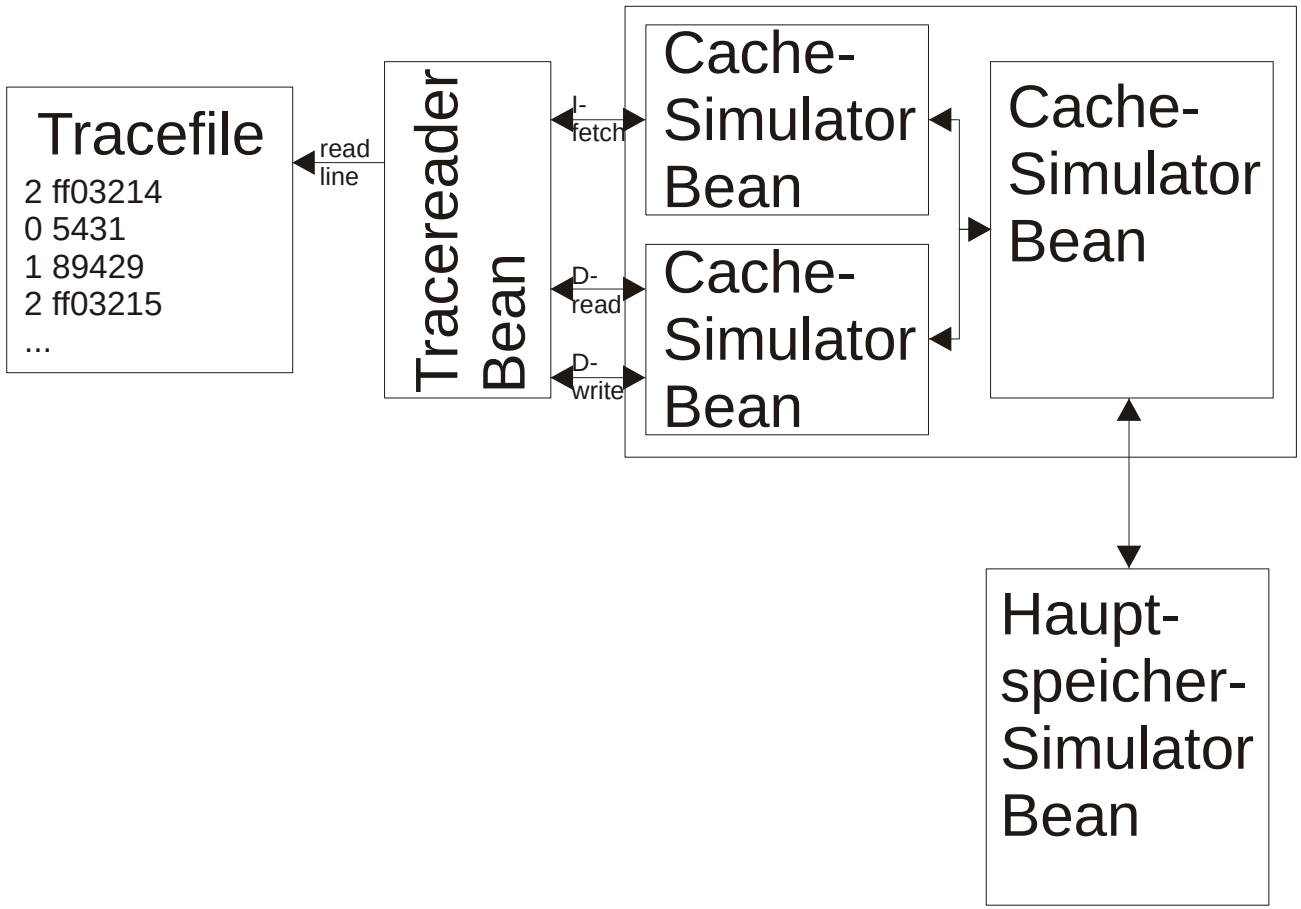
- **Welche Zeile soll bei einem Miss ersetzt werden?**
  - *direct mapped*: keine Auswahl
  - *set oder fully associative*:
    - random: zufällig Auswahl
    - LRU: am längsten unreferenzierte Zeile
- **Was geschieht bei Schreibzugriffen?**
  - *allocate on write*: Block bei Schreibzugriff in Cache laden?
  - *write through*: Schreiboperationen sofort weitergeben
  - *write back*: Schreiboperationen erst beim Ersetzten der Zeile weitergeben

Realisierung

- **JavaBeans™**
  - Komponententechnologie für Java™
  - Plattformunabhängigkeit
  - Wiederverwendbarkeit
  - Visuelle Erstellung von Applets und Anwendungen
  - Persistente Konfiguration
  - Herstellerunabhängige Verwendbarkeit von Entwicklungsumgebungen
- **Spezifikation**
  - öffentlicher, parameterloser Konstruktor
  - *getter*- und *setter*-Methoden für *properties*
  - Serialisierung und Deserialisierung
  - Kommunikation über Java Eventmodell
  - Signaturschemas für geforderte Methoden

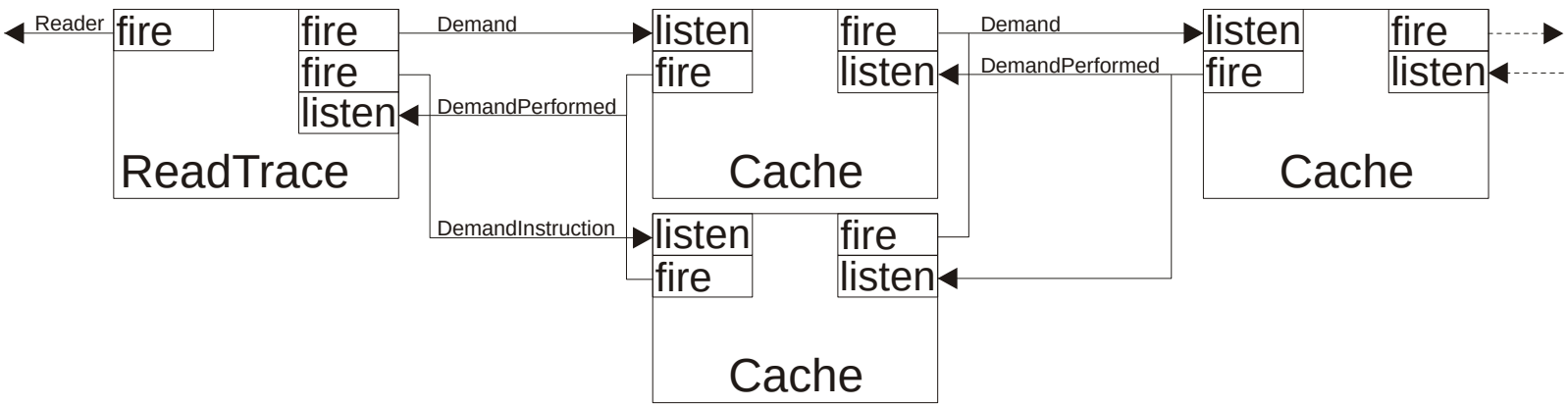
Realisierung

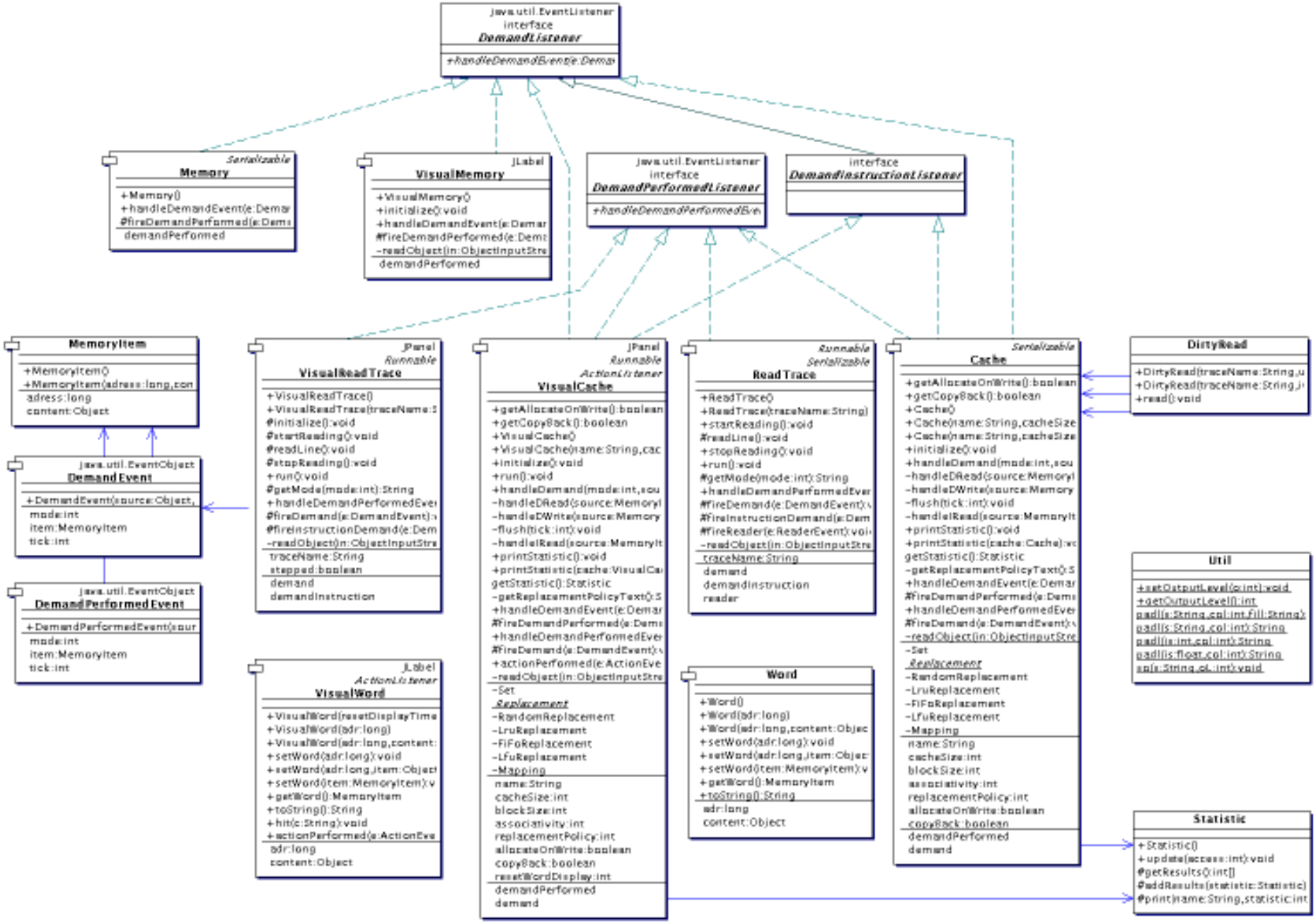
- **Abstrakter Aufbau des Cache-Simulators**



Realisierung

- **Delegation Event Modell**
  - Event Object: Signalisierung, Kapselung relevanter Daten
  - Event Source: Eventerzeugung, Listener-Registrierung
  - Event Listener: Interessierte Objekte
- **Eventmodell der Simulator Beans**
  - *DemandEvent*: Anforderung Speicherzugriff, Datentransport
  - *DemandPerformedEvent*: Zugriff ausgeführt, Datentransport
  - Schnittstellen: *Demand*, *DemandInstruction*, *DemandPerformed*




java.util.EventListener
interface
DemandListener
+handleDemandEvent(e:Demand
Serializable
Memory
+Memory()
+handleDemandEvent(e:Demar
#fireDemandPerformed(e:Dem
demandPerformed
JLabel
VisualMemory
+VisualMemory()
+initialize():void
+handleDemandEvent(e:Demar
#fireDemandPerformed(e:Dem
-readObject(in:ObjectInputStre
demandPerformed
java.util.EventListener
interface
DemandPerformedListener
+handleDemandPerformedEve
interface
DemandInstructionListener
MemoryItem
+MemoryItem()
+MemoryItem(adress:long,con
adress:long
content:Object
java.util.EventObject
DemandEvent
+DemandEvent(source:Object,
mode:int
item:MemoryItem
tick:int
java.util.EventObject
DemandPerformedEvent
+DemandPerformedEvent(sour
mode:int
item:MemoryItem
tick:int
JPanel
Runnable
VisualReadTrace
+VisualReadTrace()
+VisualReadTrace(traceName:S
#initialize():void
#startReading():void
#readLine():void
#stopReading():void
+run():void
#getMode(mode:int):String
+handleDemandPerformedEve
#fireDemand(e:DemandEvent):
#fireInstructionDemand(e:Dem
-readObject(in:ObjectInputStre
traceName:String
stepped:boolean
demand
demandInstruction
JPanel
Runnable
ActionListener
VisualCache
+getAllocateOnWrite():boolean
+getCopyBack():boolean
+VisualCache()
+VisualCache(name:String,cac
+initialize():void
+run():void
+handleDemand(mode:int,sou
-handleDRead(source:Memory
-handleDWrite(source:Memory
-flush(tick:int):void
-handleRead(source:MemoryIt
+printStatistic():void
+printStatistic(cache:VisualCa
getStatistic():Statistic
-getReplacementPolicyText():S
+handleDemandEvent(e:Demar
#FireDemandPerformed(e:Dem
+handleDemandPerformedEve
#FireDemand(e:DemandEvent):
+actionPerformed(e:ActionEve
-readObject(in:ObjectInputStre
-Set
Replacement
-RandomReplacement
-LruReplacement
-FiFoReplacement
-LFuReplacement
-Mapping
name:String
cacheSize:int
blockSize:int
associativity:int
replacementPolicy:int
allocateOnWrite:boolean
copyBack:boolean
resetWordDisplay:int
demandPerformed
demand
Runnable
Serializable
ReadTrace
+ReadTrace()
+ReadTrace(traceName:String)
+startReading():void
#readLine():void
+stopReading():void
+run():void
#getMode(mode:int):String
+handleDemandPerformedEver
#fireDemand(e:DemandEvent):
#FireInstructionDemand(e:Dem
#FireReader(e:ReaderEvent):voi
-readObject(in:ObjectInputStre
traceName:String
demand
demandInstruction
reader
Serializable
Cache
+getAllocateOnWrite():boolean
+getCopyBack():boolean
+Cache()
+Cache(name:String,cacheSize
+Cache(name:String,cacheSize
+initialize():void
+handleDemand(mode:int,sou
-handleDRead(source:Memory
-handleDWrite(source:Memory
-flush(tick:int):void
-handleRead(source:MemoryIt
+printStatistic():void
+printStatistic(cache:Cache):vc
getStatistic():Statistic
-getReplacementPolicyText():S
+handleDemandEvent(e:Demar
#FireDemandPerformed(e:Dem
+handleDemandPerformedEve
#FireDemand(e:DemandEvent):
-readObject(in:ObjectInputStre
-Set
Replacement
-RandomReplacement
-LruReplacement
-FiFoReplacement
-LFuReplacement
-Mapping
name:String
cacheSize:int
blockSize:int
associativity:int
replacementPolicy:int
allocateOnWrite:boolean
copyBack:boolean
demandPerformed
demand
DirtyRead
+DirtyRead(traceName:String,u
+DirtyRead(traceName:String,i
+read():void
Util
+setOutputLevel(o:int):void
+getOutputLevel():int
padR(s:String,col:int,fill:String):
padR(s:String,col:int):String
padR(s:int,col:int):String
padR(s:float,col:int):String
spl(s:String,pL:int):void
JLabel
ActionListener
VisualWord
+VisualWord(resetDisplayTime
+VisualWord(adr:long)
+VisualWord(adr:long,content:
+setWord(adr:long):void
+setWord(adr:long,item:Object
+setWord(item:MemoryItem):v
+getWord():MemoryItem
+toString():String
+hit(c:String):void
+actionPerformed(e:ActionEve
adr:long
content:Object
Word
+Word()
+Word(adr:long)
+Word(adr:long,content:Objec
+setWord(adr:long):void
+setWord(adr:long,item:Objec
+setWord(item:MemoryItem):v
+getWord():MemoryItem
+toString():String
adr:long
content:Object
Statistic
+Statistic()
+update(access:int):void
#getResults():int[]
#addResults(statistic:Statistic)
#print(name:String,statistic:int

Realisierung

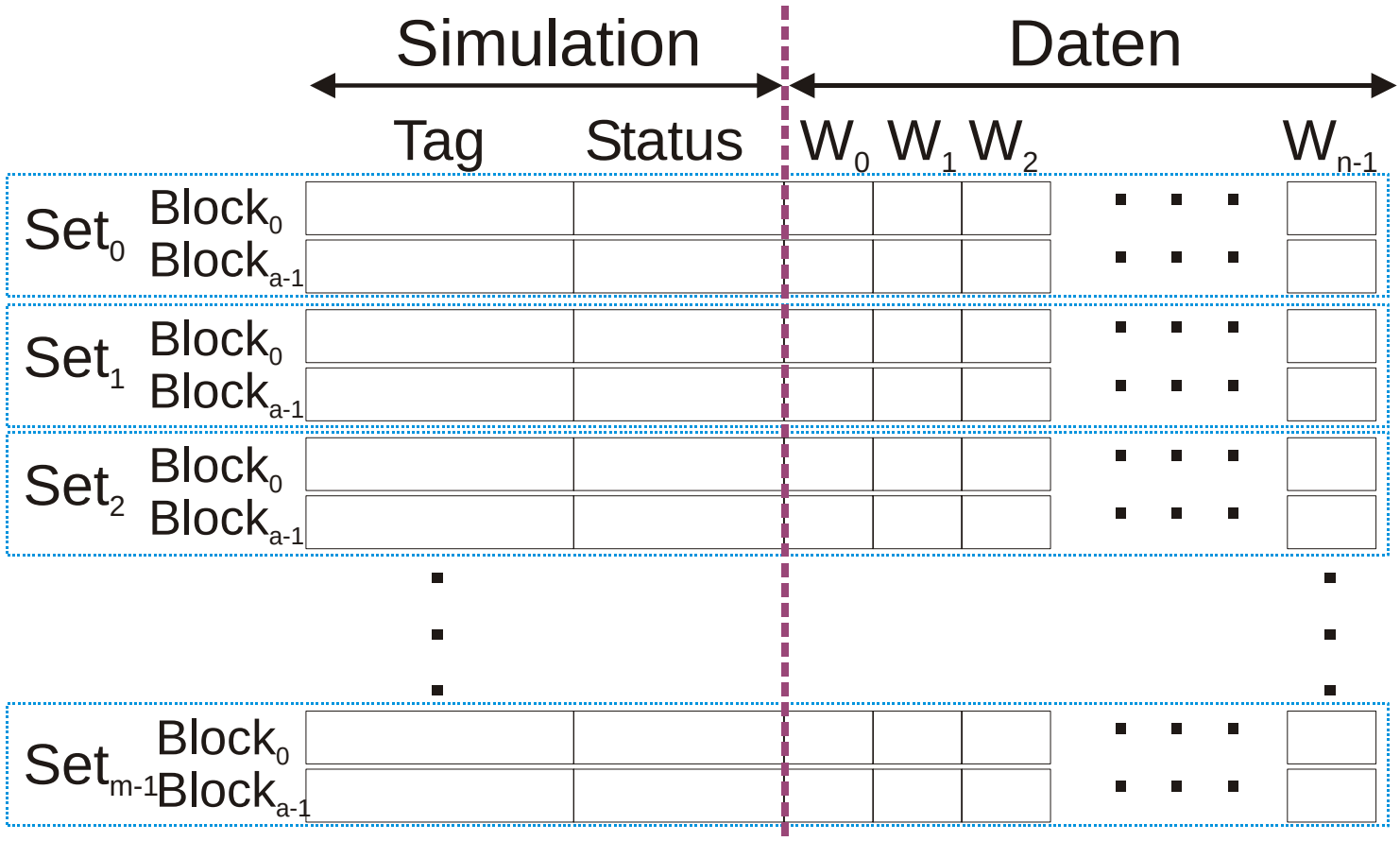
- **Simulator Beans**
  - jeweils visuell und nichtvisuell
  - *ReadTrace*:
    - Verwaltung Tracefile
    - Steuerung Simulatormodell
  - Cache:
    - Modellierung Cachefunktionalität
    - Visualisierung
    - Statistik
  - Memory:
    - Modellierung einer Speichereinheit
    - Abschluß Simulationsmodell

Realisierung

- **Cache Beanproperties - modellierte Designparameter**
  - *cacheSize*: Cachegröße in Byte
  - *blockSize*: Größe der modellierten Cachezeilen
  - *associativity*: direct mapped = 1, fully = # Cachezeilen
  - *replacementPolicy*: LRU, Random, FiFo, LFU
  - *allocateOnWrite*: Ladeverhalten bei Schreibzugriffen
  - *copyBack*: Schreibstrategie
- **Registrierbare Event Listener**
  - *DemandEventListener*
  - *DemandInstructionListener*
  - *DemandPerformedListener*
- **Implemtierte Listener-Interfaces**
  - *DemandEventListener*
  - *DemandInstructionListener*
  - *DemandPerformedListener*

Realisierung

- **Cache Bean - Datenmodell**



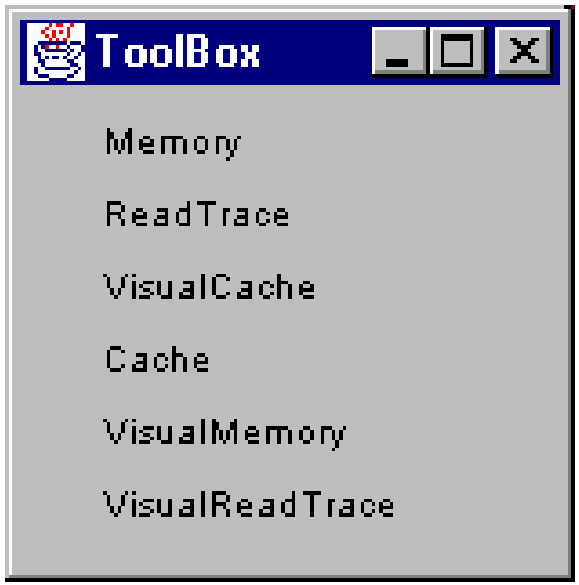
Anwendung

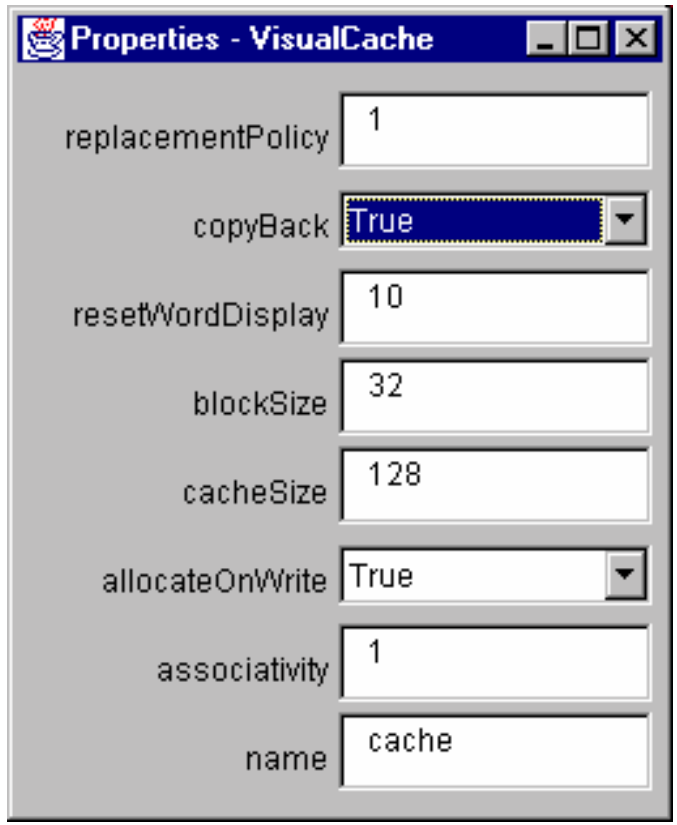
- **Klassenbibliothek**
  - Cache Beans können auch „klassisch“ angewendet werden
  - Objekte erzeugen + initialisieren + platzieren:
    - VisualCache iCache = new VisualCache();
    - iCache.setCacheSize(8192); ...
    - tp.addTab(iCache.getName(),iCache);...
  - Eventkette aufbauen:
    - reader.addDemandInstructionListener(iCache);...
    - iCache.addDemandPerformedListener(reader);...
    - memory.addDemandPerformedListener(iCache);...
  - Statistik ausgeben (falls gewünscht):
    - if (myReaderEvent.isFinished()) {
      - iCache.printStatistic(dCache);...}
  - Applikation übersetzten und starten

Anwendung

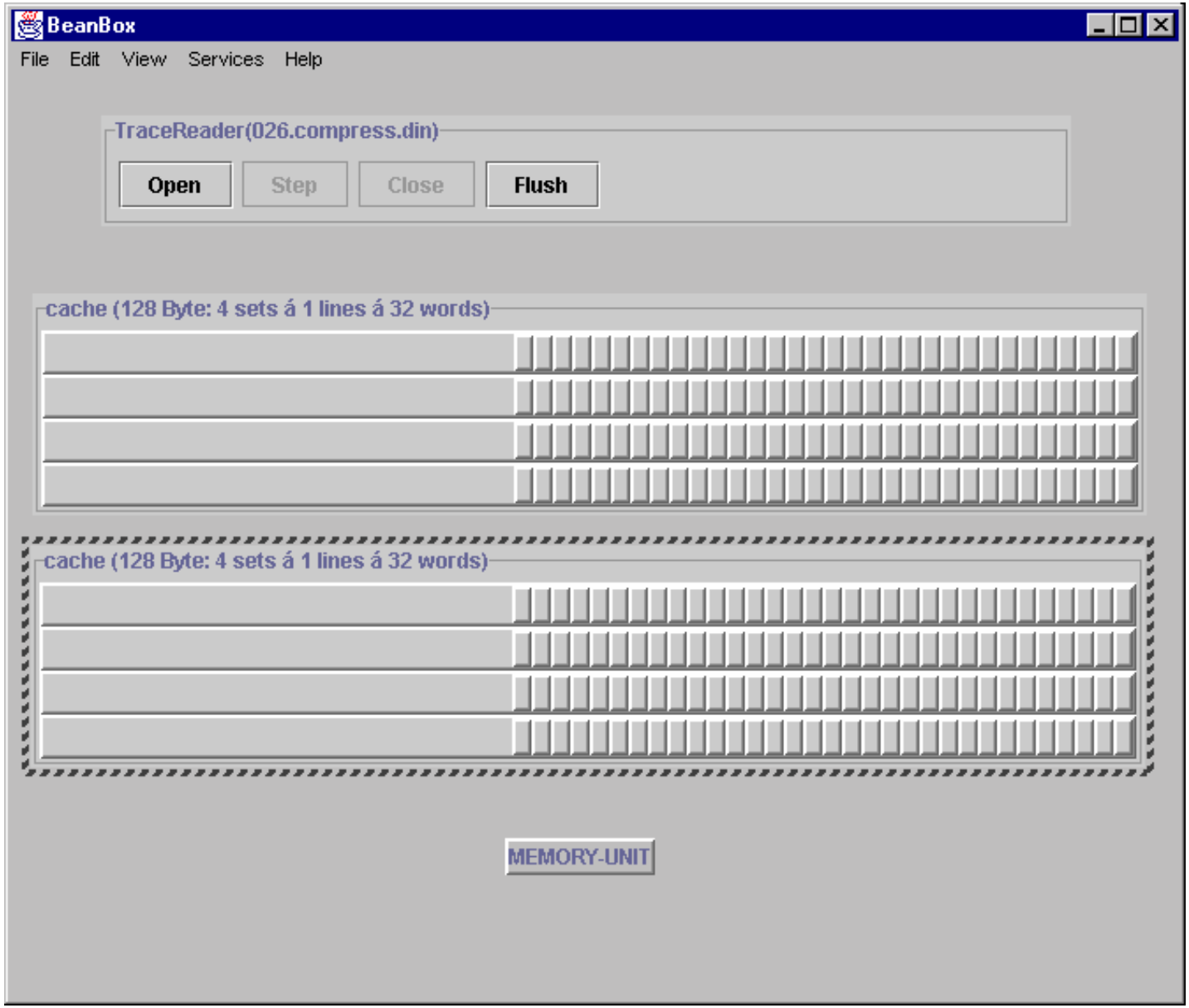
- **Grafische Entwicklungsumgebungen (z.B.: BeanBox)**
  - Beans dem System hinzufügen
  - Aus „Toolbox“ auswählen
  - Visuell platzieren
  - Mittels „PropertyEditor“ konfigurieren
  - Eventverbindungen visuell aufbauen
  - Design speichern, generieren, testen
  - Applet oder Anwendung kann automatisch erstellen werden

Anwendung

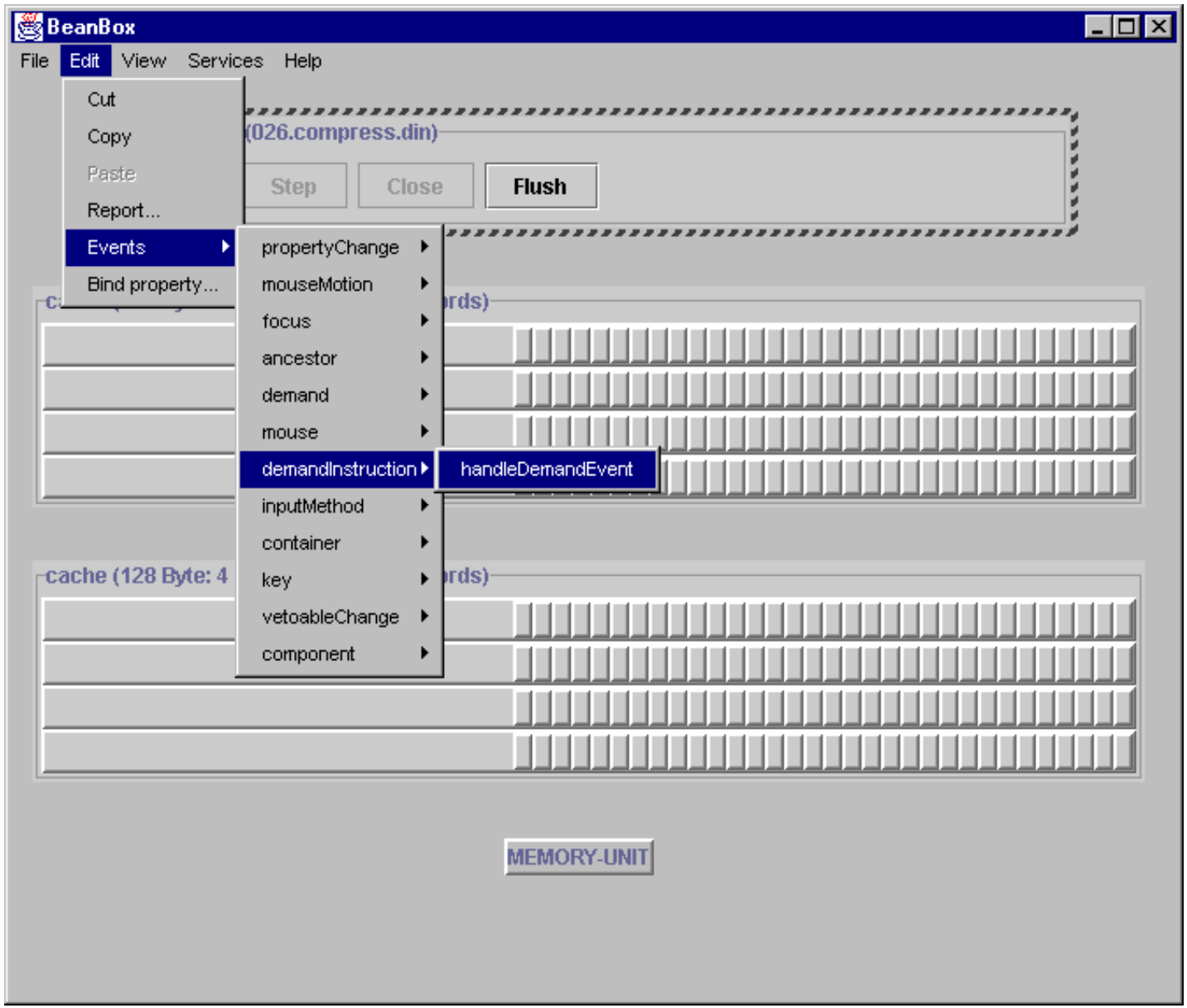




Anwendung



Anwendung



trace-driven cache-simulation

TraceReader(026.compress.din)

Open | Step | Close | Flush (375): IFetch > 40566c

Instructioncache | Datacache

Datacache (256 Byte: 4 sets á 4 lines á 16 words)

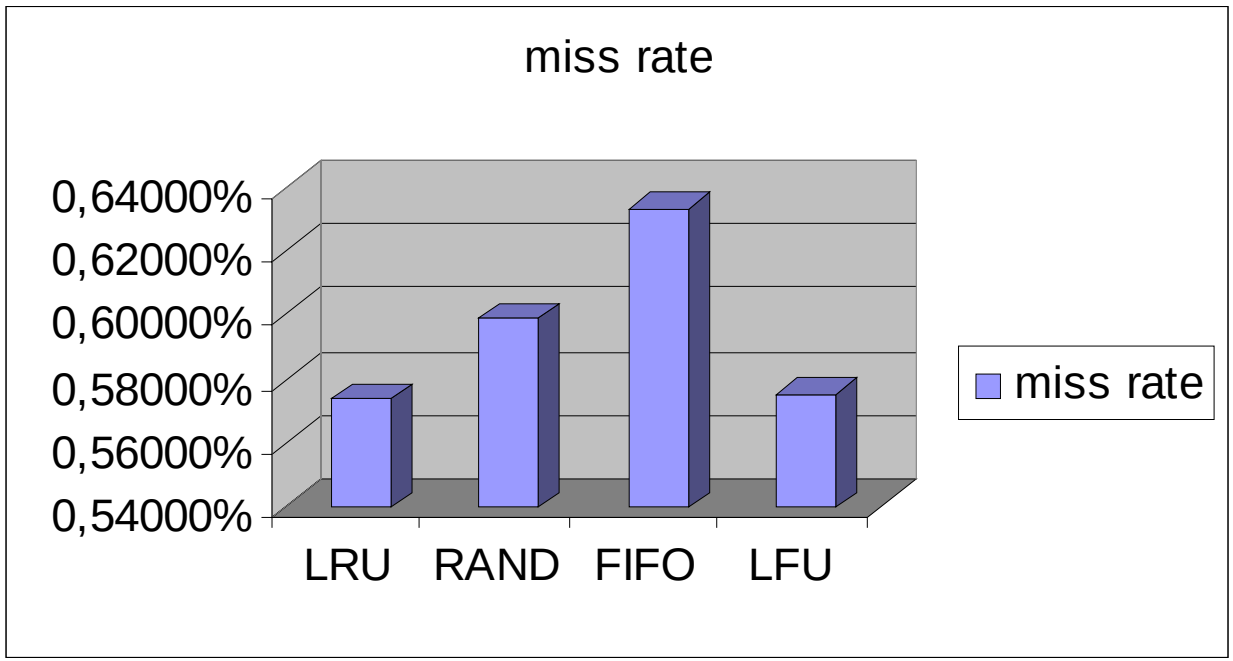
7ffebb80	340	340	1	d	W
7ffebb40	23	23	1	d	W
7ffebc40	45	45	1	d	W
7ffebb00	61	345	16	d	R R W W
10001710	368	368	1		R
100017d0	64	213	4	d	W
7ffebb10	69	364	25	d	W W R R
10001850	71	217	4	d	W
7ffebc60	2	17	2	d	W R
7ffebb60	29	37	4	d	W W W W
7ffebb20	80	358	11	d	R R
10001770	342	362	2	d	W
7ffebaf0	51	333	6	d	R
100017f0	260	309	2	d	W
10001870	267	313	2	d	W

Anwendung

C:\WINNT\System32\cmd.exe

```
E:\projects\uni\Cache>E:\Entwicklung\jdk1.2.2\bin\java.exe Beispiel1 eg.din

Initialisiere Instructioncache...
   - Cachegroesse    :8192
   - Assoziativitaet :1
   - Blockgroesse    :32
   - Replacement     :LRU
Instructioncache mit 256 sets á 1 lines á 32 words bereit.

Initialisiere Datacache...
   - Cachegroesse    :8192
   - Assoziativitaet :1
   - Blockgroesse    :32
   - Replacement     :LRU
Datacache mit 256 sets á 1 lines á 32 words bereit.

.
Instructioncache + Datacache:
            Metrics     Total     Instrn      Data      Read     Write
-------------------------------------------------------------------------
      Demand Fetches      1000        470       530       234       296
       Demand Misses        59         26        33        16        17
                         0,0590    0,0553    0,0623    0,0684    0,0574

E:\projects\uni\Cache>_
```

Anwendung



Zusammenfassung

- **Entwickelte Cache Beans zum Aufbau von Cachesimulatoren sind:**
  - komponentenbasiert ➔ JavaBeans
  - erweiterbar ➔ Objektorientierung
  - einfache Anwendbarkeit ➔ JavaBeans, visuelle Tools
  - visuelle Simulation ➔ flexible JavaSwing Komponenten
  - trace-gesteuerte Simulation ➔ ReadTrace Bean
  - einsetzbar in größeren Simulationsmodellen

    ➔ Datenmodellierung, nichtvisuelle Beans

Zusammenfassung

- **Cache-Speicher Trends**
  - Caches werden in dieser Form schon lange und immer noch prinzipiell gleich verwendet
  - Neuerungen durch Kombination mit anderen Konzepten
    - *Tracecache*: Cache + *branch prediction*