Image Persistence

Virtuelle Maschinen 2008

Murat Knecht, Christian Schwarz

Gliederung

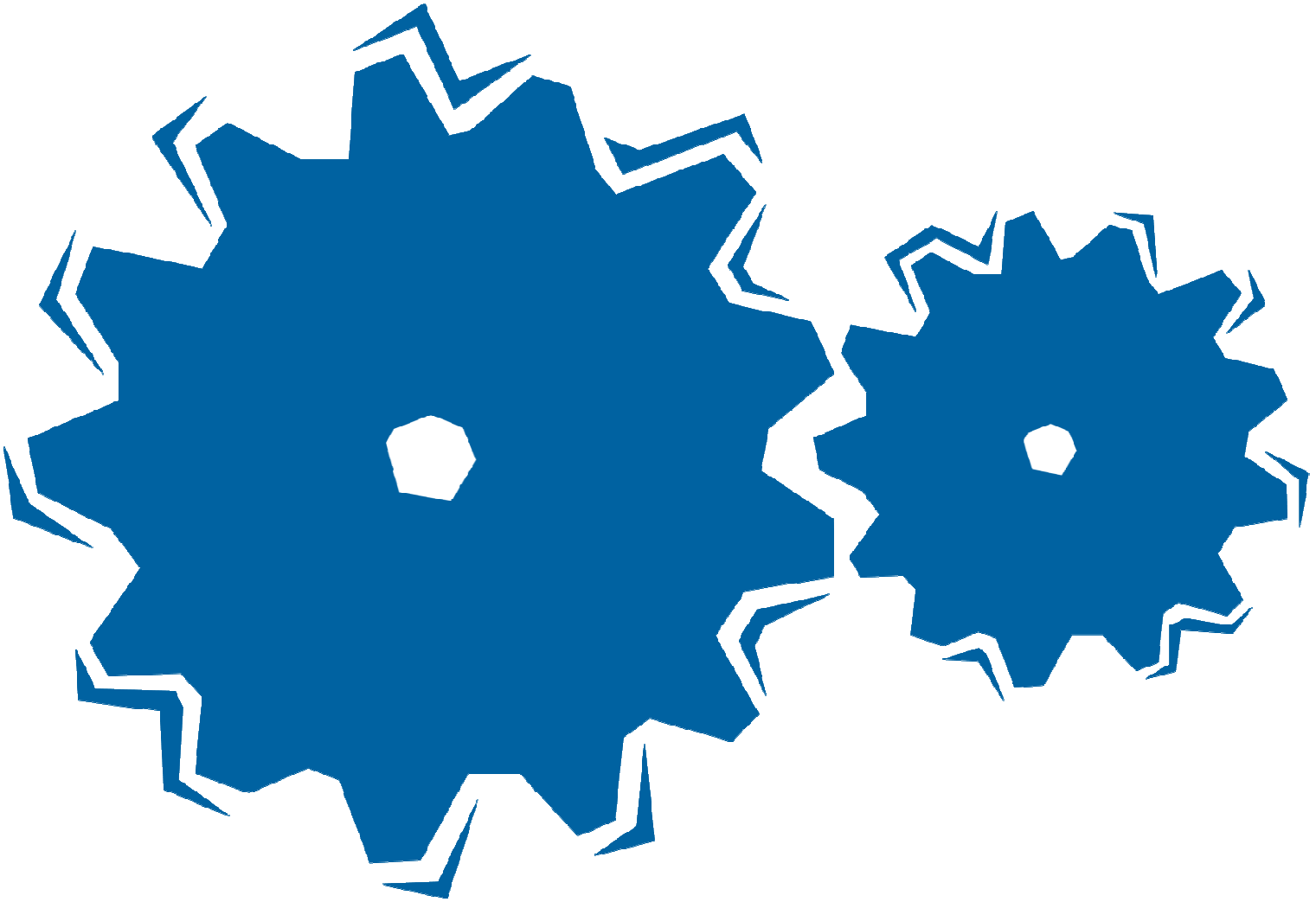
- Features
- Demo
- Speichern des Images
  - Callback, Standard- und Improved Image
- Laden des Images
  - Pointerrelokation, Primitivencode
- Demo
- Ausblick

Features

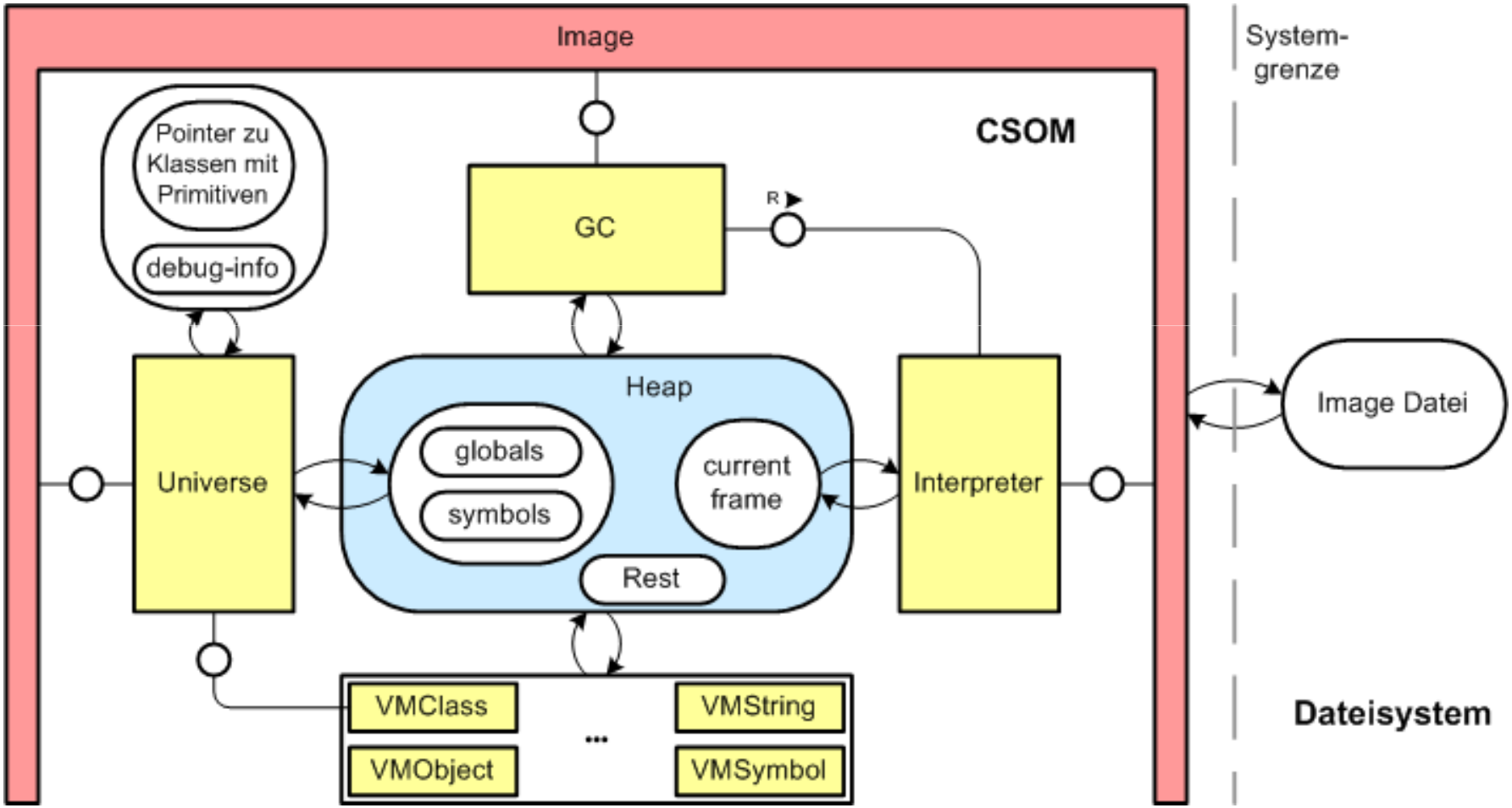
- Laufzeit kann jederzeit in Image abgelegt und aus Image geladen werden
- Image kann verschlüsselt werden

- Image kann momentan nicht komprimiert werden

Demo



Architektur (VM)



Änderungen (Smalltalk)

- Image
- *system* object
  - Nachrichten zum Speichern / Laden

```
System = (
hibernate: imagename = ( self hibernate: imagename
                         withEncryption: CRYPT_NONE usingKey: 0. )
hibernate: string withEncryption: integer
                         usingKey: integer = primitive
inspectImage: string = primitive
loadImage: imagename = ( self loadImage: imagename usingKey: 0. )
loadImage: string usingKey: integer = primitive
)
```

- Snake

Speichern

- Alle Informationen der Laufzeit werden gespeichert
- Zusätzliche Schritte werden angewendet
  - „Filter“ Mechanismus
  - Kompression, Verschlüsselung, …

Callback

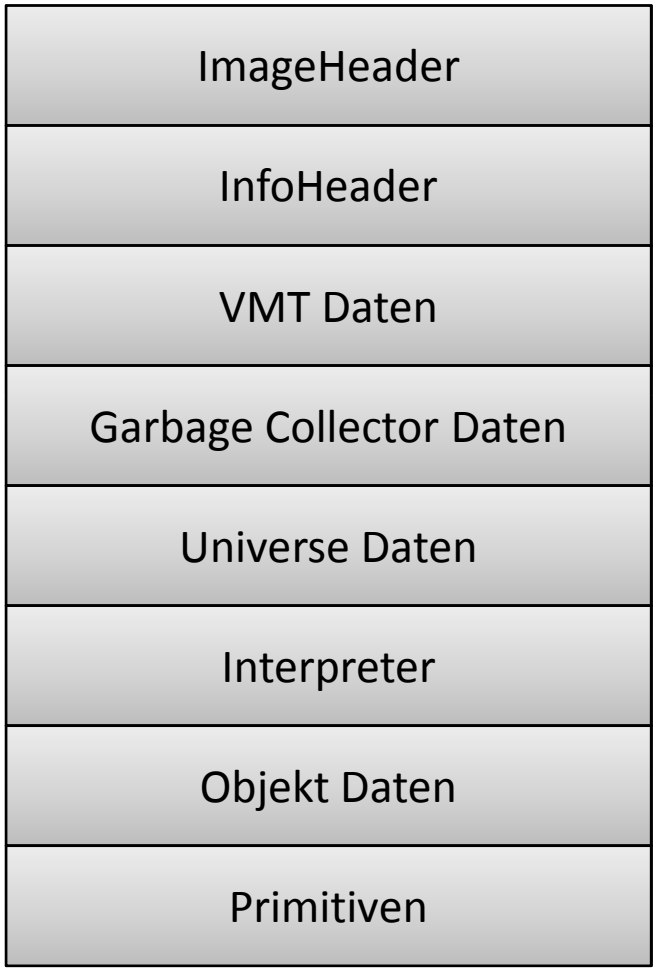
- Komponenten laden und speichern per Callback
- Reihenfolge, in der Elemente ihre Daten speichern/laden muss gleich sein
- Signaturen:

```
bool image_store_byte_sequence(const INTEGER size,
                               const void* const byte_sequence);
```

```
void* image_load_byte_sequence(INTEGER* size, bool retain);
```

Dateiformat

- Image besteht aus zwei Hauptteilen:
  - Image Header
  - Imagedaten
- Image Header ist fast konstant
- Imagedaten sind abhängig vom Imageformat



Dateiformat

- Image besteht aus zwei Hauptteilen:
  - Image Header
  - Imagedaten
- Image Header ist fast konstant
- Imagedaten sind abhängig vom Imageformat

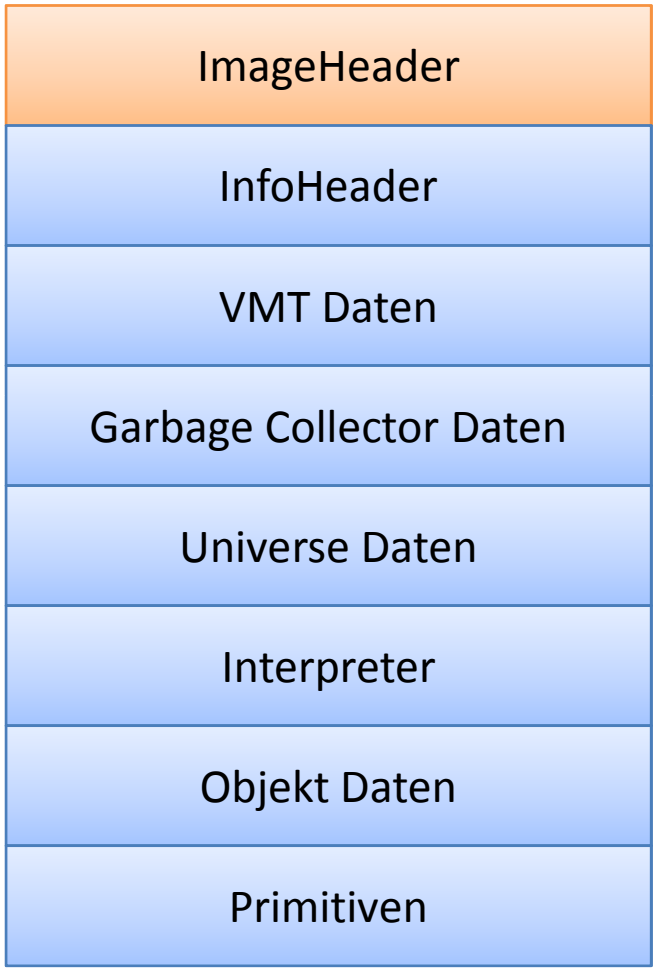
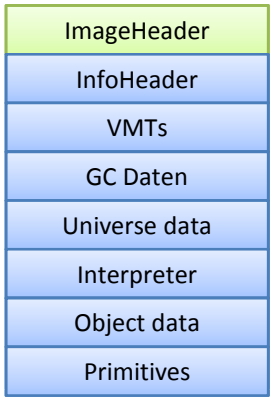


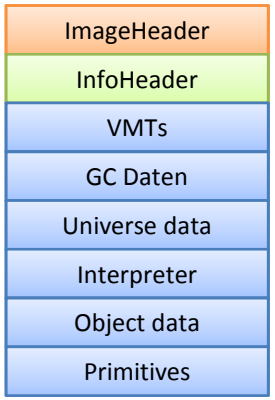
Image Header

- Enthält generelle Informationen über das Image
  - Magic Constant – 0xCAFE2342
  - Word Length in Bytes (4 oder 8)
  - Version des Images
  - Zusätzliche Imageformat Informationen



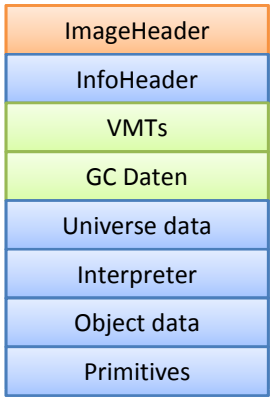
Info Header

- Enthält Image spezifische Informationen:
  - Zeitpunkt des Speicherns
  - Offsets zu den einzelnen Dateibestandteilen



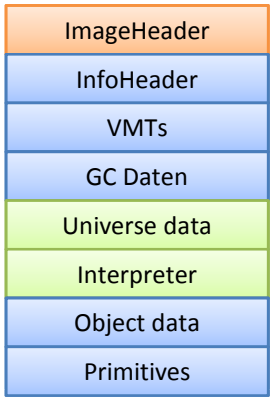
VMT und Garbage Collector Daten

- VMT Pointer der VM Klassen werden gespeichert
  - Benötigt, um beim Laden alte auf neue VMTs mappen zu können
  - VMT* werden in einem *struct* abgelegt
- GC speichert Heap + Meta-Infos
  - Heap enthält (fast) alle Daten der Objekte
  - Meta: Startadresse, Größe, Freier Speicher, Beginn der free-entry Liste



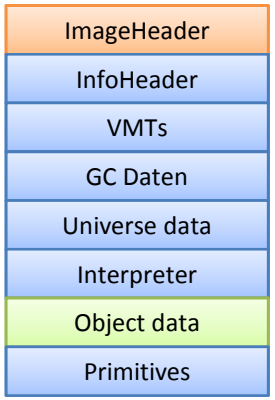
Universe + Interpreter Daten

- Universe:
  - Globals und Symboltable
  - Systemkonfiguration
- Interpreter:
  - *CurrentFrame



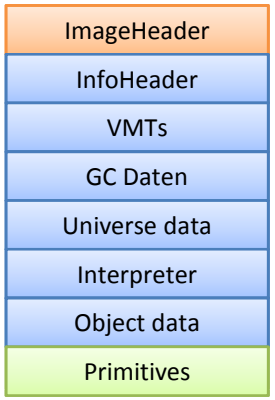
Objekt Daten

- Nicht alle Objektdaten liegen im Heap
  - embedded_string
  - (plain_string)
- Objektbaum wird traversiert
  - Virtuelle Methode von *VMObject*
  - Rekursiv über *VMObject.fields*
    - … wenn das Feld in den Heap zeigt
  - GC-Feld dient als „Schon bearbeitet“ Marker

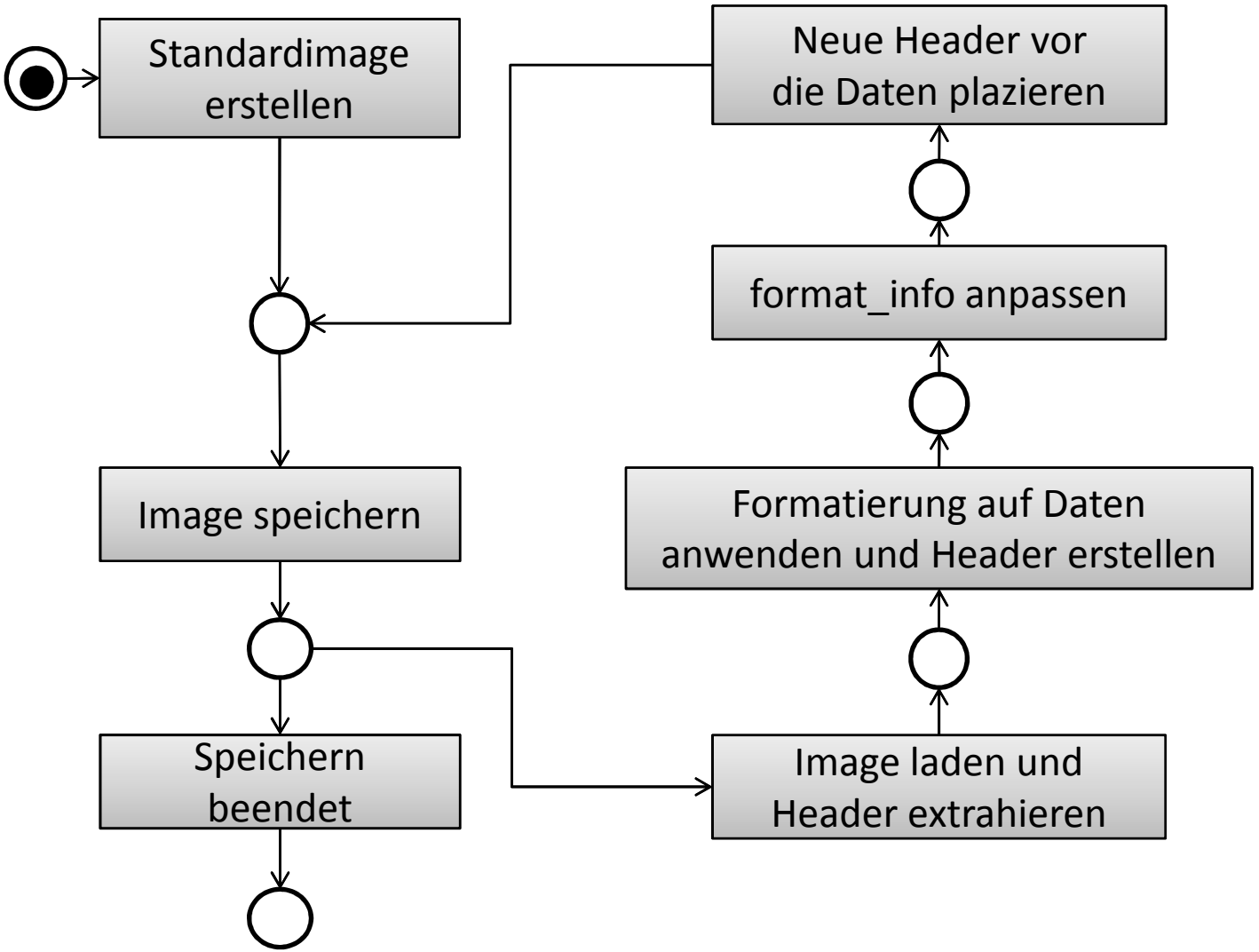


Primitiven

- Klassen mit Primitiven registrieren sich bei *Universe*
  - Wenn Code nachgeladen wird
- Registrierung wird gespeichert

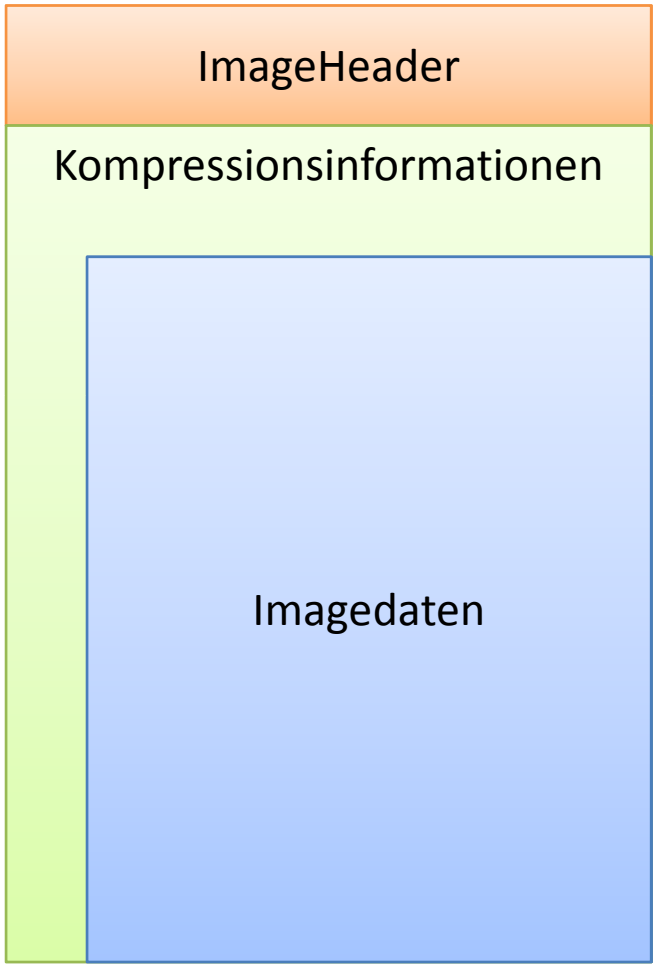


Zusätzliche Features



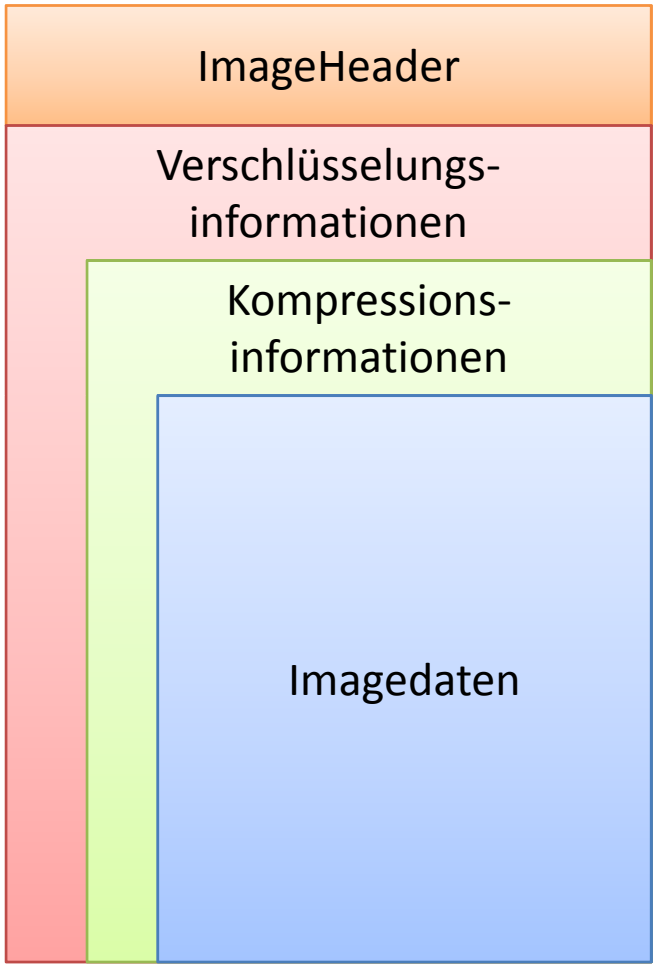
Kompression

- Kompression ist optionaler Schritt
- Imageheader wird angepasst
- Kompressionsheader wird erstellt
- Imagedaten werden komprimiert gespeichert

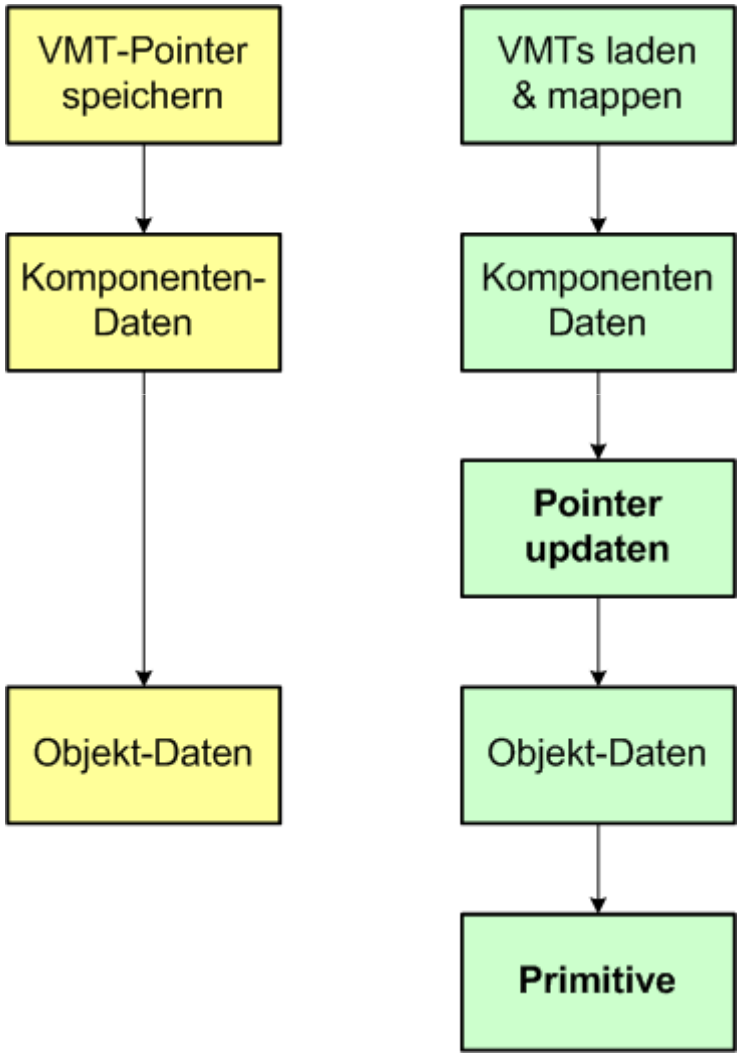


Verschlüsselung

- Optionaler Schritt
- Verschlüsselungsheader wird erstellt
- Untergeordnete Daten werden verschlüsselt
- ImageHeader und EncryptionHeader speichern



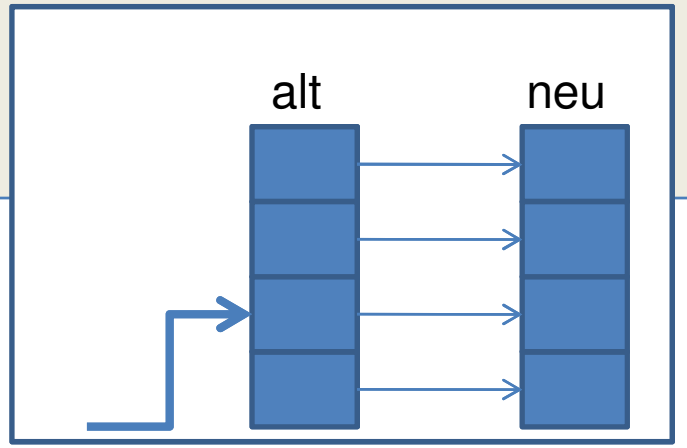
Speichern & Laden



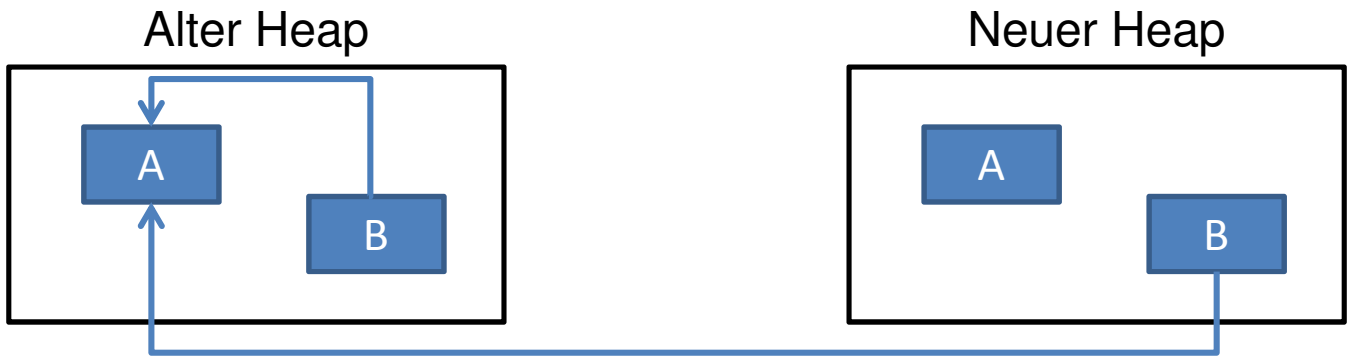
VMT Rekonstruktion

- Es wird über das VMT struct iteriert
- Neuer Pointer wird zurück gegeben

```
void* image_translate_vmt_pointer(const void* const old_pointer) {
        [...]
        int items = sizeof(VMT_Addresses) / sizeof(void*);
        for (int curr = 0; curr < items; curr++)
                if (old_pointer == ((void**)&old_vmt_pointers)[curr])
                        return ((void**)&current_vmt_pointers)[curr];
        [...]
        return NULL;
}
```



Pointer updaten

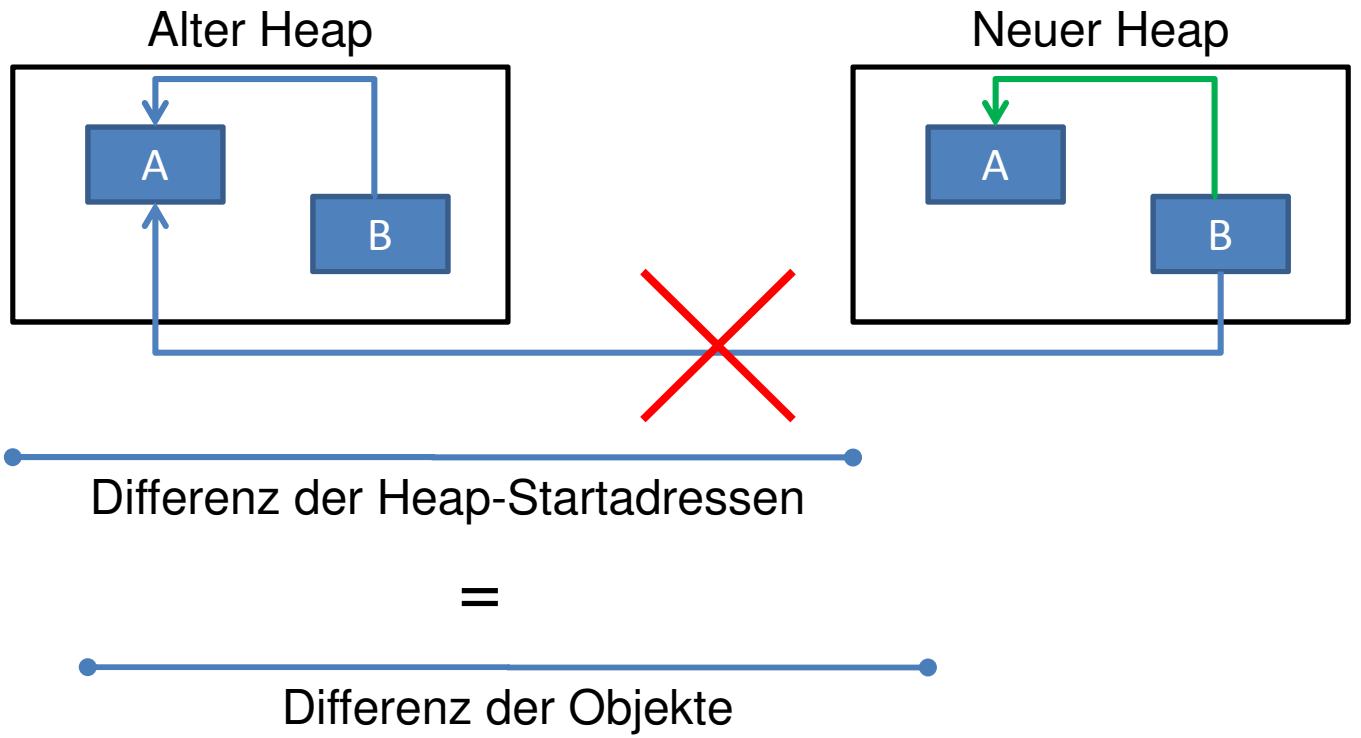


Differenz der Heap-Startadressen

=

Differenz der Objekte

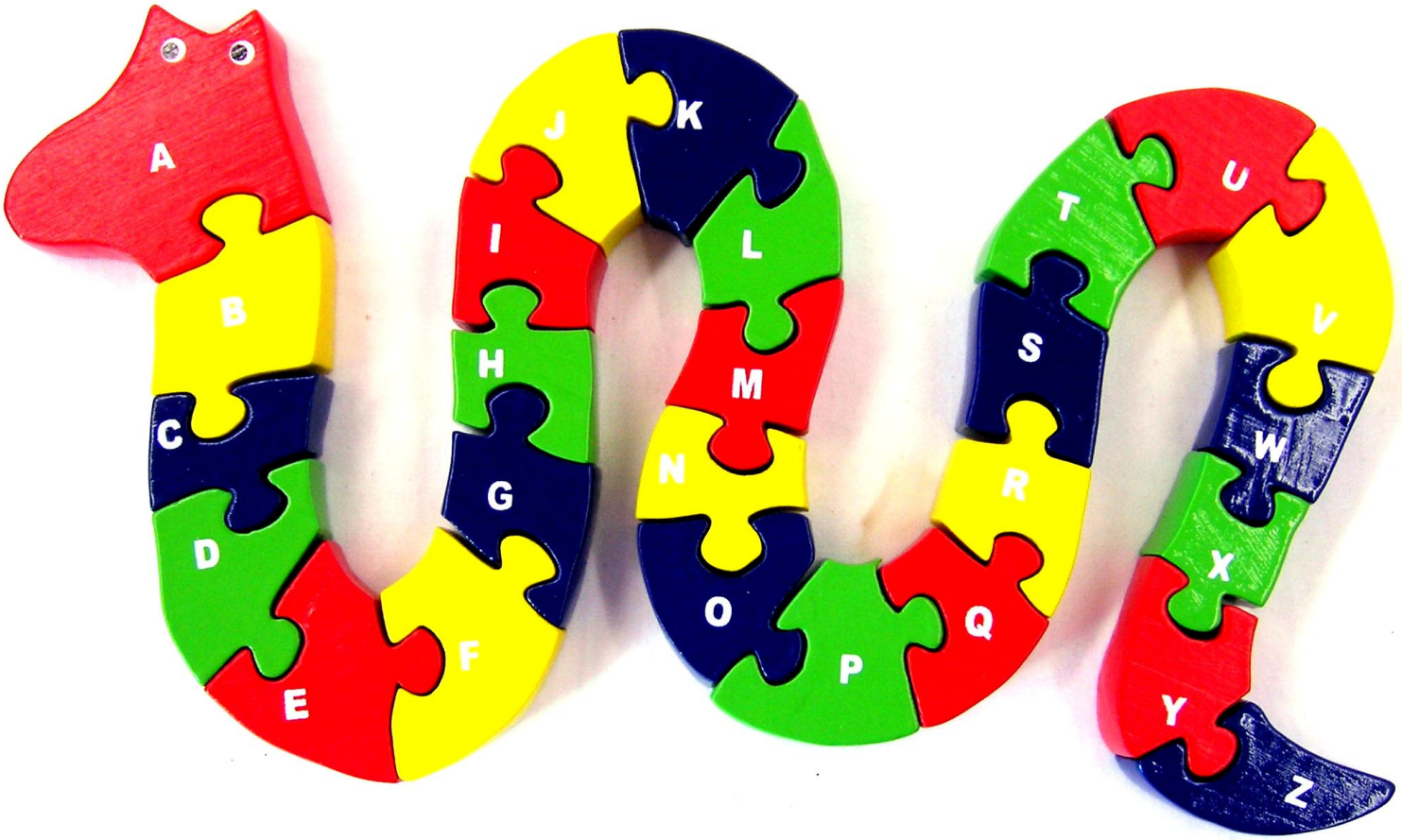
Pointer updaten



Objektdaten und Primitiven

- Objekte müssen ihre nicht im Heap liegenden Daten wieder laden
  - Objektbaum wird auf gleiche Art durchlaufen
  - Objekte deserialisieren ihre Daten daher in gleicher Reihenfolge
  - Speichern &
- Klassen, welche Primitiven nutzen, laden deren Code neu

Proof of Concept



Future Work

- Portabilität
  - Endianess ermitteln und anpassen
  - Image unabhängig von der Hostarchitektur speichern, wenn möglich
- Funktionalität
  - Nutzer mehr Interaktionsmöglichkeiten geben
  - Die letzten 30 Bit von ImageHeader.format_info sinnvoll nutzen

Future Work 2

- Snake
  - Neuzeichnen korrigieren
  - Highscore
  - Gegnerische Schlange
  - Weitere Items einführen
  - Labyrinthe bauen
  - Splitscreenmode
  - Onlinerangliste