

# Garbage Collection

---

Carsten Gips (HSBI)

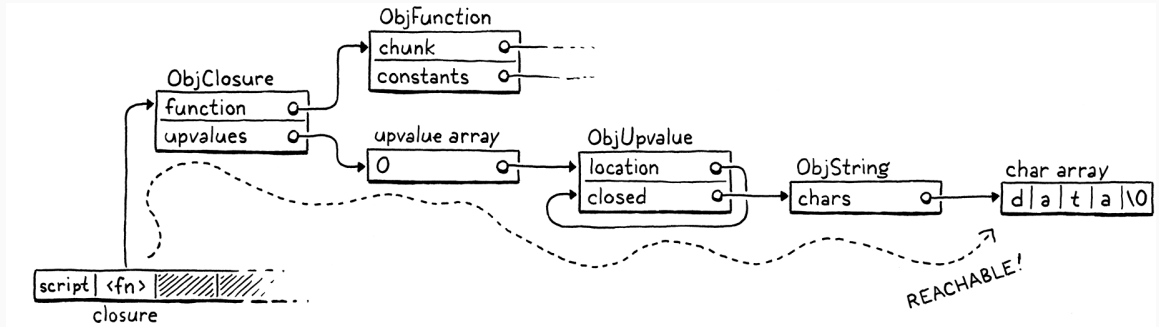
Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

## Ist das Code oder kann das weg?

```
x = 42  
y = 'wuppie'  
y = 'fluppie'  
print(y)
```

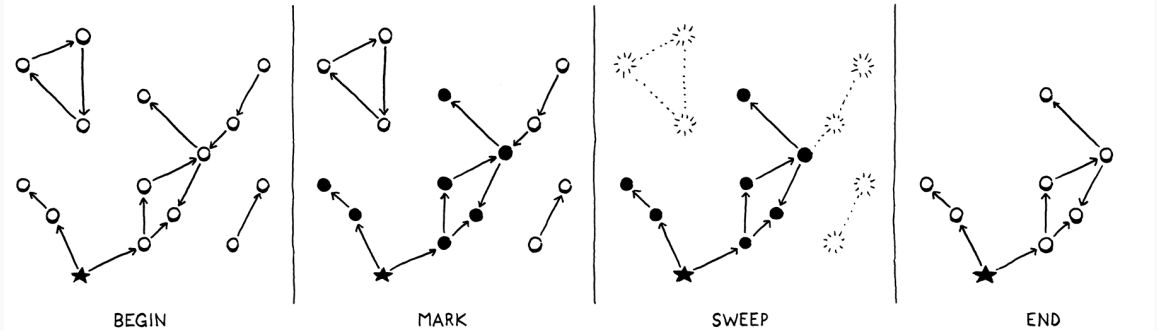
```
def foo():  
    x = 'wuppie'  
    def bar():  
        print(x)  
    return f  
  
fn = foo()  
fn()
```

# Erreichbarkeit



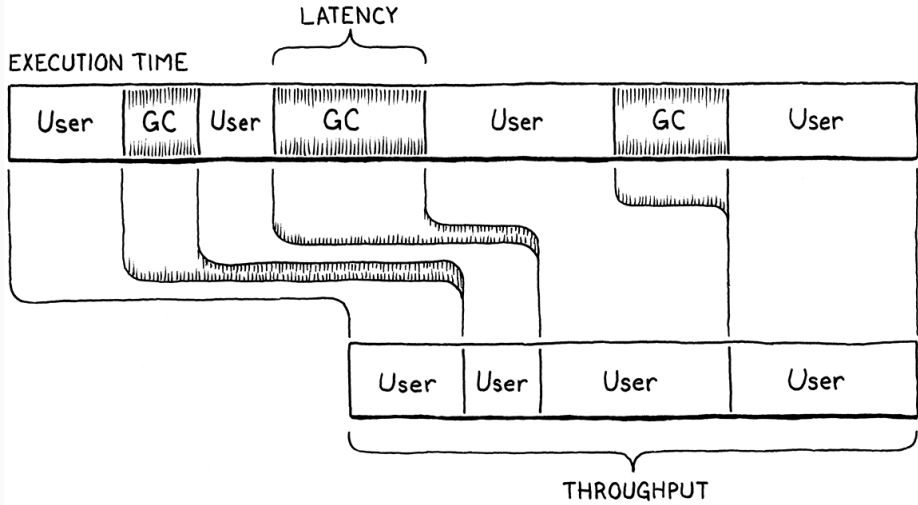
Quelle: reachable.png by Bob Nystrom on Github.com (MIT)

# “Präzises GC”: Mark-Sweep Garbage Collection



Quelle: mark-sweep.png by Bob Nystrom on Github.com (MIT)

# Metriken: Latenz und Durchsatz



Quelle: latency-throughput.png by Bob Nystrom on Github.com (MIT)

# Heuristik: Self-adjusting Heap

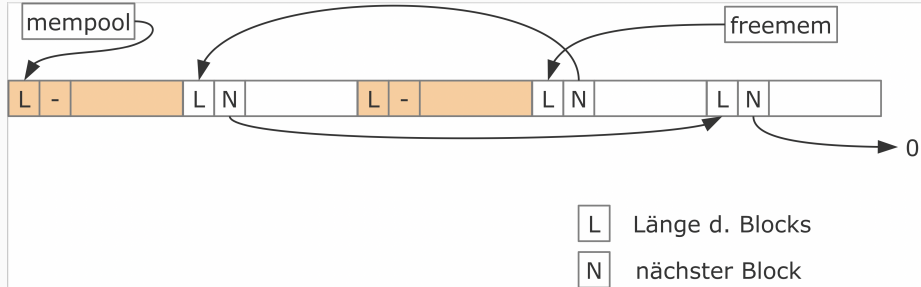
- GC selten: Hohe Latenz (lange Pausen)
- GC oft: Geringer Durchsatz

## Heuristik

- Beobachte den allozierten Speicher der VM
- Wenn Grenze überschritten: GC
- Größe des verbliebenen Speichers mal Faktor  $\Rightarrow$  neue Grenze

- Teile Heap in zwei Bereiche: "*Kinderstube*" und "*Erwachsenenbereich*"
- Neue Objekte werden in der Kinderstube angelegt
- Häufiges GC in Kinderstube
- Überlebende Objekte werden nach  $N$  Generation in den Erwachsenenbereich verschoben
- Deutlich selteneres GC im Erwachsenenbereich

## “Konservatives GC”: Boehm GC



- Idee: Nutze die interne Verwaltung des Heaps zum Finden von Objekten



# Reference Counting

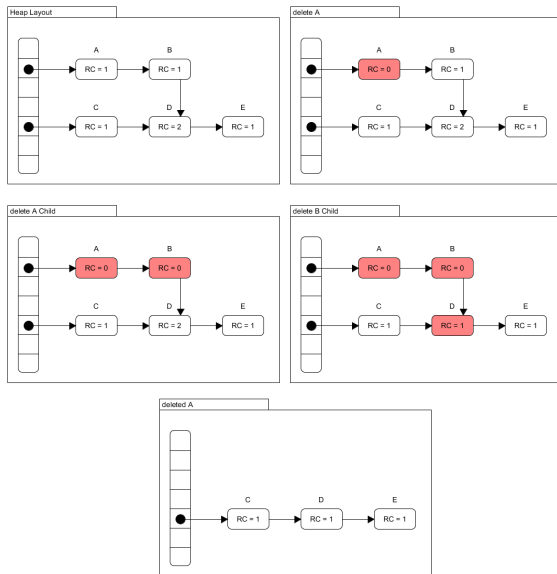
```
def new():  
    obj = alloc_memory()  
    obj.set_ref_counter(1)  
    return obj
```

```
def delete(obj):  
    obj.dec_ref_counter()  
    if obj.get_ref_counter() == 0:  
        for child in children(obj):  
            delete(child)  
        free_object(obj)
```

# Reference Counting

```
def new():  
    obj = alloc_memory()  
    obj.set_ref_counter(1)  
    return obj
```

```
def delete(obj):  
    obj.dec_ref_counter()  
    if obj.get_ref_counter() == 0:  
        for child in children(obj):  
            delete(child)  
        free_object(obj)
```



# Stop-and-Copy Garbage Collection

- Teile Heap in zwei Bereiche (A und B)
- Alloziere nur Speicher aus A (bis der Bereich voll ist)
- Stoppe Programmausführung und kopiere alle erreichbaren Objekte von A nach B
- Gebe gesamten Speicher in A frei
- Setze Programmausführung mit vertauschten Rollen von A und B fort

## Ziel

- Vergleich von verschiedenen GC-Algorithmen
- Wahl des optimalen Algorithmus für konkreten Anwendungsfall
- Overhead durch GC möglichst gering halten

## Setup

- “Warm up”: einige Iterationen des Benchmarks ohne Messung vorlaufen lassen, um bspw. Just-in-Time Kompilierung und Initialisierung aller Laufzeitkomponenten abzuschließen
- Anpassung der Heap-Größe an den Benchmark (falls Heap zu bestimmter Rate gefüllt werden soll)
- Überprüfung und Eliminierung von externen Faktoren, die das Ergebnis beeinflussen können
  - andere Prozesse
  - dynamische Frequenzskalierung der CPU
  - Vermeidung von Lese- und Schreibzugriffen, sofern diese nicht durch Test-Infrastruktur vorgegeben sind

## Relevante Metriken für Tests

- Durchsatz (welchen Anteil hat GC an der Laufzeit?)
- Latenz (welche Verzögerung erzeugt GC?)

## Szenarien

(stark vom getesteten GC-Algorithmus abhängig)

- konstant gefüllter Heap/leerer Heap
- Erzeugen und Löschen von stark referenzierten Objekten/temporären Objekten
- Für Generational GC: Testen von Overhead von Schreib- und Lesebarrieren durch Objektreferenzierung

- Pflege verkettete Liste aller Objekte in der VM
- Mark-Sweep-GC:
  1. Markiere alle Wurzeln (“grau”, aus Stack und Hashtabelle)
  2. Traversiere ausgehend von den Wurzeln alle Objekte und markiere sie
  3. Gehe die verkettete Liste aller Objekte durch und entferne alle nicht markierten
- Problem: Latenz und Durchsatz => Idee des “self-adjusting” Heaps
- Varianten/Alternativen: Generational GC, Boehm-GC, Reference Counting, Stop-and-Copy GC, ...



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

## Exceptions

- Image reachable.png by Bob Nystrom on Github.com (MIT)
- Image mark-sweep.png by Bob Nystrom on Github.com (MIT)
- Image latency-throughput.png by Bob Nystrom on Github.com (MIT)