```
oid GC_do_local_mark(mse *local_mark_stack, mse *local_top)

  unsigned n;
  define N_LOCAL_ITERS 1

  ifdef GC_ASSERTIONS
    /* Make sure we don't hold mark lock. */
  GC_acquire_mark_lock();
  GC_release_mark_lock();
  endif
  for (;;) {
      for (n = 0; n < N_LOCAL_ITERS; ++n) {
      local_top = GC_mark_from(local_top, local_mark_stack,
                    local_mark_stack + LOCAL_MARK_STACK_SIZE);
      if (local_top < local_mark_stack) return;
      if (local_top - local_mark_stack >= LOCAL_MARK_STACK_SIZE/2) {
      GC_return_mark_stack(local_mark_stack, local_top);
      return;
      }
  }
  if (GC_mark_stack_top < GC_first_nonempty &&
      GC_active_count < GC_helper_count
      && local_top > local_mark_stack + 1) {
      /* Try to share the load, since the main stack is empty,    */
      /* and helper threads are waiting for a refill.     */
      /* The entries near the bottom of the stack are likely  */
      /* to require more work.  Thus we return those, eventhough  */
      /* it's harder.                         */
      mse * p;
      mse * new_bottom = local_mark_stack
              + (local_top - local_mark_stack)/2;
      GC_ASSERT(new_bottom > local_mark_stack
            && new_bottom < local_top);
      GC_return_mark_stack(local_mark_stack, new_bottom - 1);
```

Conservative Garbage Collection (für C)

Christian Höglinger, 0256505

Sa. 14.1.2006

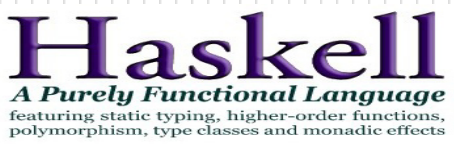
Überblick

- Motivation
- Conservative Garbage Collection
  - Problemstellung
  - Allgemeine Aufgaben
- Boehm-Demers–Weiser Collector
  - Lösung für C
  - Beispiele
- Probleme
- Performance

Motivation 1/2

- Bisher: Unterstützung durch
  - Compiler
  - Laufzeitumgebung
- GC ohne solche Unterstützung?
  - C, C++
  - In weiterer Folge
    - Modula-3
    - Haskell
    - ...

```
What is Modula-3? Modula-3 is a systems programming language that
descends from Mesa, Modula-2, Cedar, and Modula-2+. It also resembles
its cousins Object Pascal, Oberon, and Euclid.
```



Motivation 2/2

- Motivation speziell für C
  - Weit verbreitet
  - Altmodische Speicherverwaltung
- Speicherverwaltung in C
  - Manuell
  - Aufwändig
  - Sehr fehleranfällig für komplexere Programme

Conservative Garbage Collection 1/3

- Bisher: *Type accurate* Collectors
  - GC kennt Typen
    - Wo finden sich Referenzen
    - Zusammenarbeit mit dem Compiler
    - Information durch Laufzeitumgebung
  - Zumindest:
    Unterscheidung *pointer – kein pointer*

Conservative Garbage Collection 2/3

- *Conservative* Collectors
  - Keine Information von Compiler/Umgebung
  - Jedes „Wort" potentielle Referenz bis Gegenteil bewiesen
  - Andererseits, möglicherweise falsche Referenzen
    - Konservativ
    - Userdaten dürfen nicht geändert werden (tagging...)
  - Ambigous roots collection

Conservative Garbage Collection 3/3

- Besonders geeignet fuer Mark & Sweep
  - Copying erfordert update der Referenzen
  - Nicht garantiert, dass Referenz ein Pointer ist!
- Aufgaben
  - Referenzen erkennen
  - *root set* finden
    - Stack
    - Register
    - Globale Variablen
  - Mark
  - Sweep

Boehm-Demers-Weiser Collector 1/4

- Kriterien für C Garbage Collector
  - GC benötigt nur Laufzeit, wenn tatsächlich verwendet
    - Reference Counting unmöglich
  - Koexistenz mit bestehenden ...
    - Betriebssystemen
    - Bibliotheken
  - Kooperation mit bestehenden Compilern

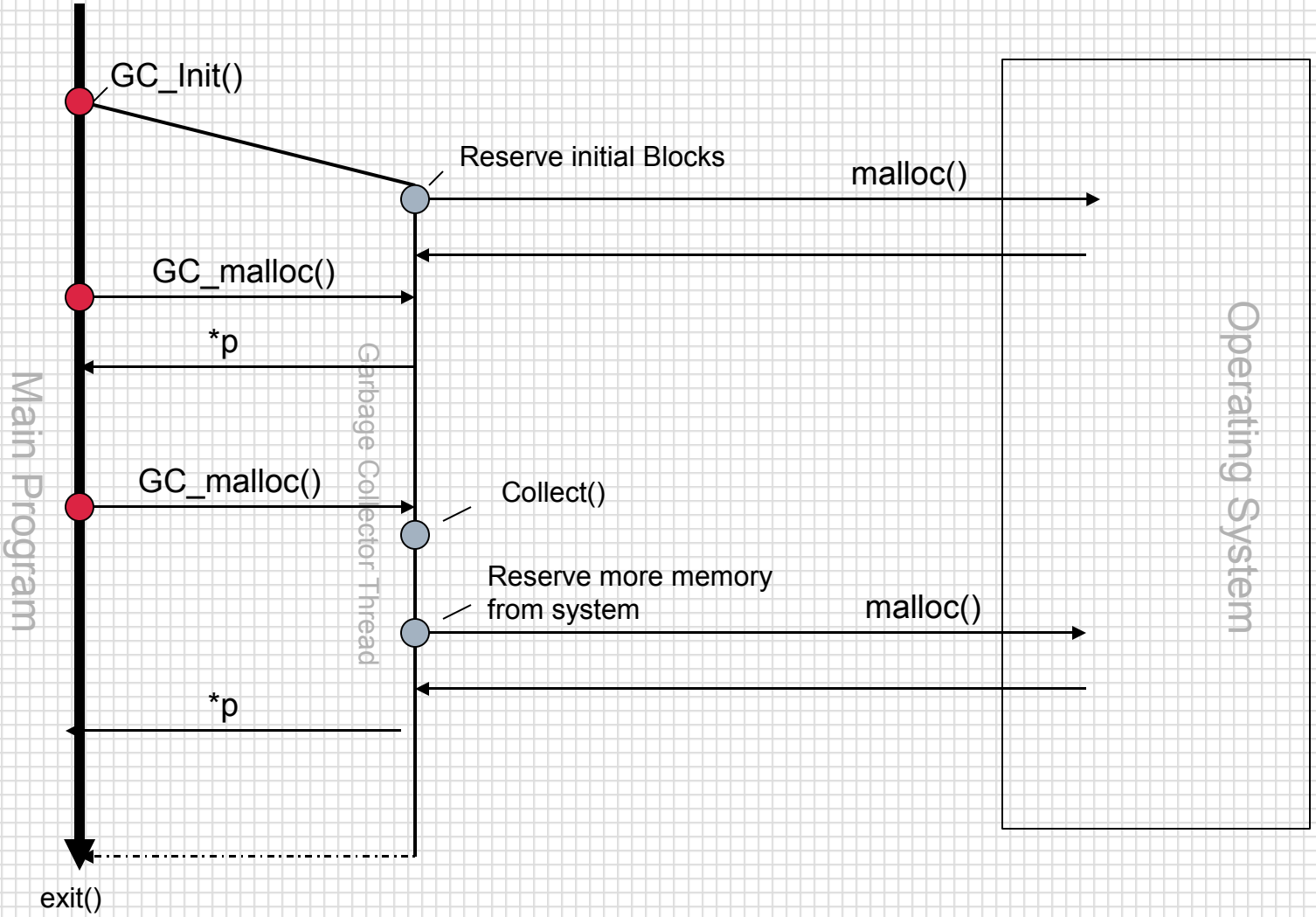
Boehm-Demers-Weiser Collector 2/4

- Conservative GC für C/C++
- Ende der 80er
- Mittlerweile für viele Systeme
- Anwender:
  - Mono
  - Mozilla
  - GCJ
- Inkrementell möglich
  - Hier nicht berücksichtigt

Boehm-Demers-Weiser Collector 3/4

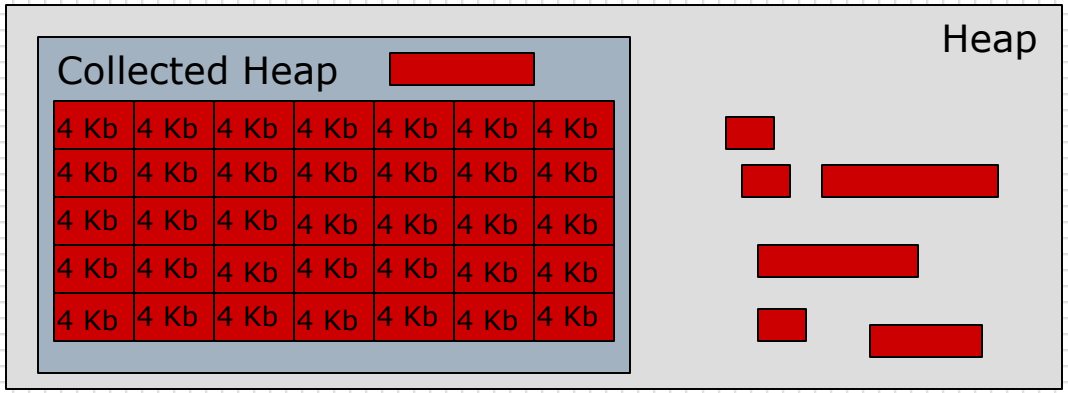
- Implementiert als Bibliothek
  - Bei Bedarf muss gegen diese Bibliothek gelinkt werden
  - Kriterien erfüllt
- Schnittstelle zum Collector
  - GC_INIT()
  - malloc() → GC_MALLOC()
  - realloc() → GC_REALLOC()
  - GC_MALLOC_ATOMIC()
  - free() → GC_FREE()

Boehm-Demers-Weiser Collector 4/4



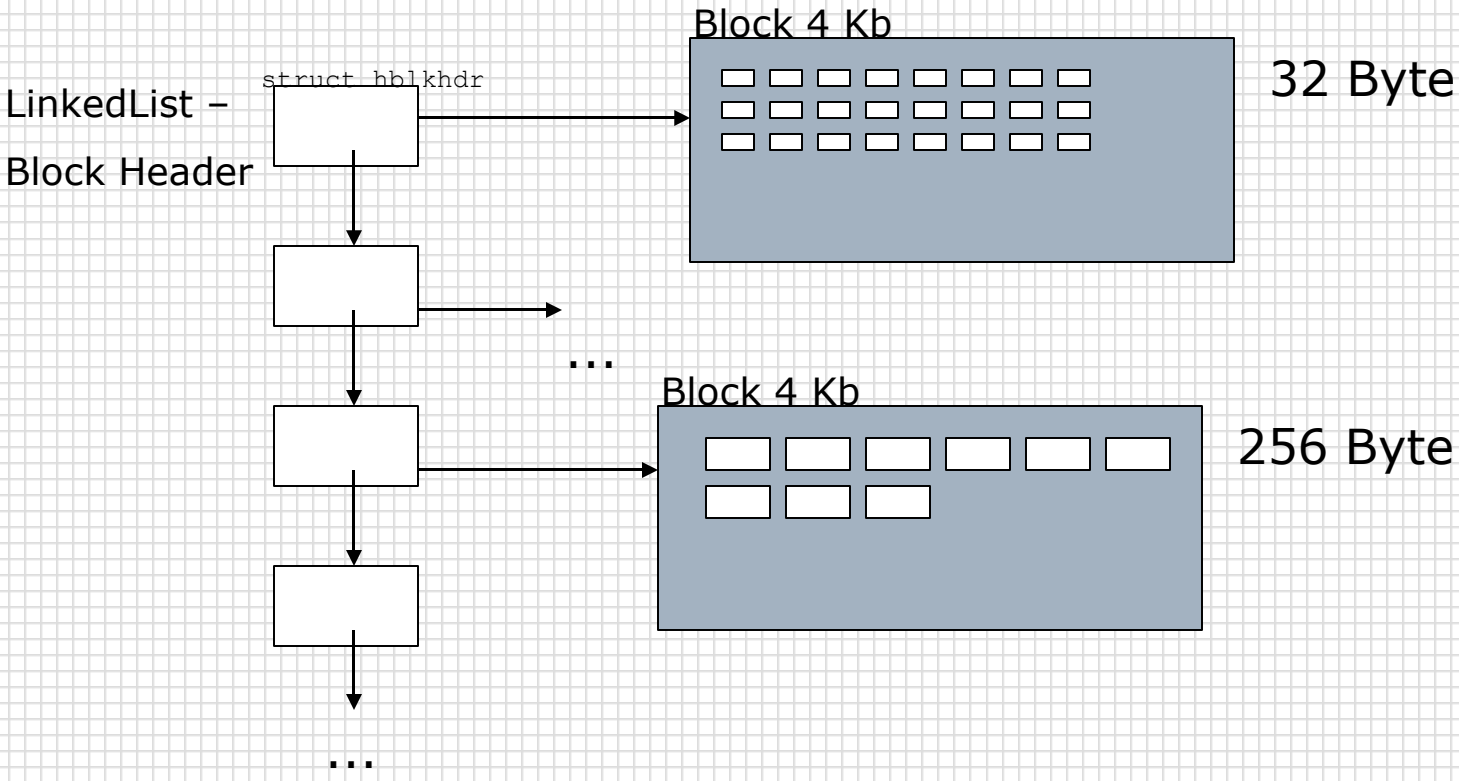
Allocation

- 2 Heaps
  - können parallel verwendet werden
  - Benutzerprogramm
  - Garbage Collector selbst
  - andere Bibliotheken verwenden Heap wie üblich
- Collected Heap
  - 4Kb Blocks



Allocation - Blocks

- Eine Objektgrösse pro Block
- Separate free-lists für verschiedene Objektgrössen



Allocation – alloc()

- Hole aus free-list (first fit)
- Wenn nichts frei...
  - sweep() auf Blocks richtiger Größe
    - eventuell ausreichend
  - gc()
    - Voraussetzung: genug allocation
  - neue Blocks vom System anfordern

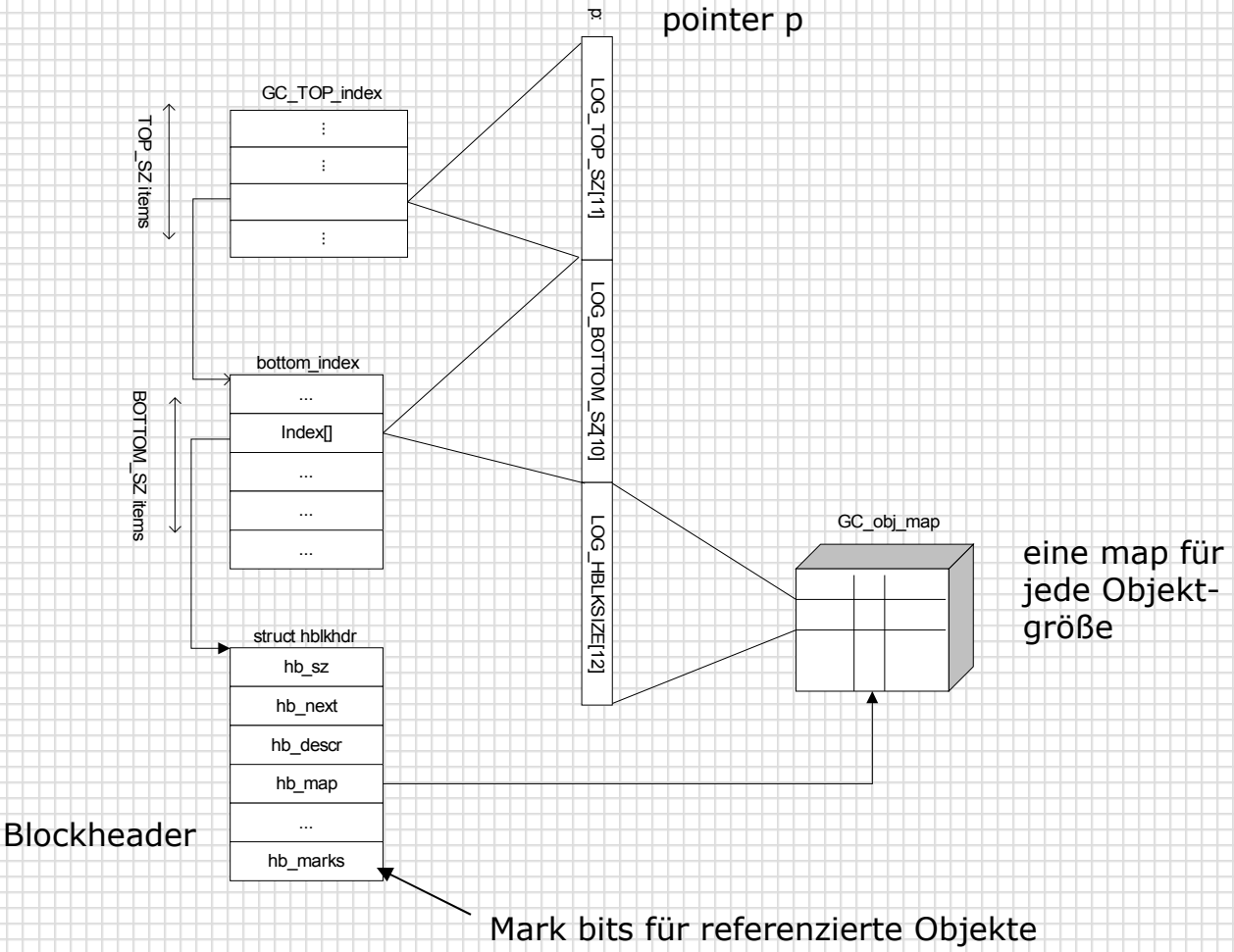
Finden von Referenzen

- Root-set finden
  - Systemabhänig
    - Anzahl Register, welche?
    - Stack Layout
- Pointer identifizieren
  - Jedes Wort – Referenz
  - „The key to success is the ability to determine the validity of a potential pointer accurately and cheaply“ [Jones96]
    - Zu konservativ
      - Space leaks
    - Pointer wird nicht als solcher erkannt
      - Crash
      - Falsches Ergebnis

Kriterien für Referenzen

- pointer p:
  - p > GC_MIN_HEAP_ADDR
  - P < GC_MAX_HEAP_ADDR
  - Block, der Objekt von p beinhaltet allocated?
  - Object in entpsprechender Object-Map enthalten?
- Alle Kriterien erfüllt?
  - Mark bit in Blockheader setzen

2-level Suchstrategie



gc()

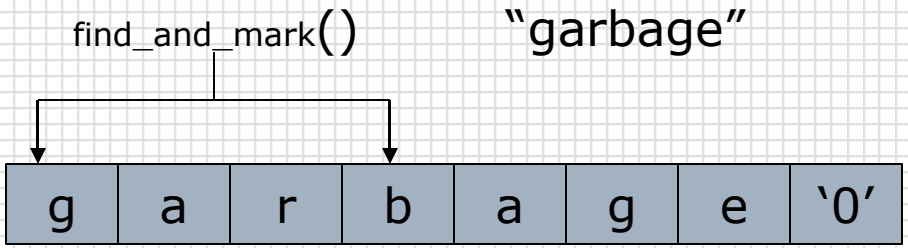
```
gc() {
    set* roots;

    find_potential_roots(roots);

    foreach root in roots {
        mark(root);
    }

    sweep(); // …
}
```

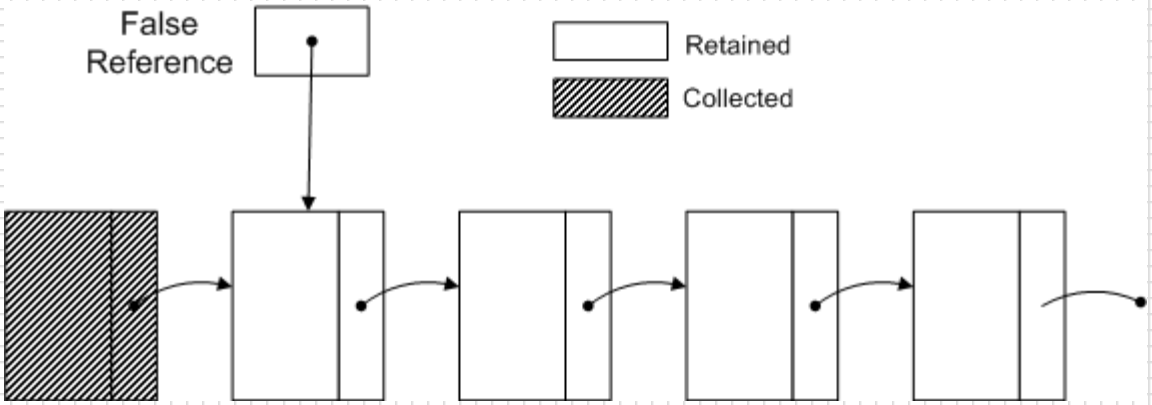
```
mark(*p) {
        obj* o = find_and_mark(p);

        if (o != NULL)  {
            foreach word w in o {
                mark(w);
            }
        }
}
```

```
find_and_mark(*p) {
    // vorige Folie
}
```

Space leaks

- Falsche Referenzen:
  - Integers
  - Strings
  - Bitmaps...





- GC_MALLOC_ATOMIC()
  - Muss nicht initialisiert werden

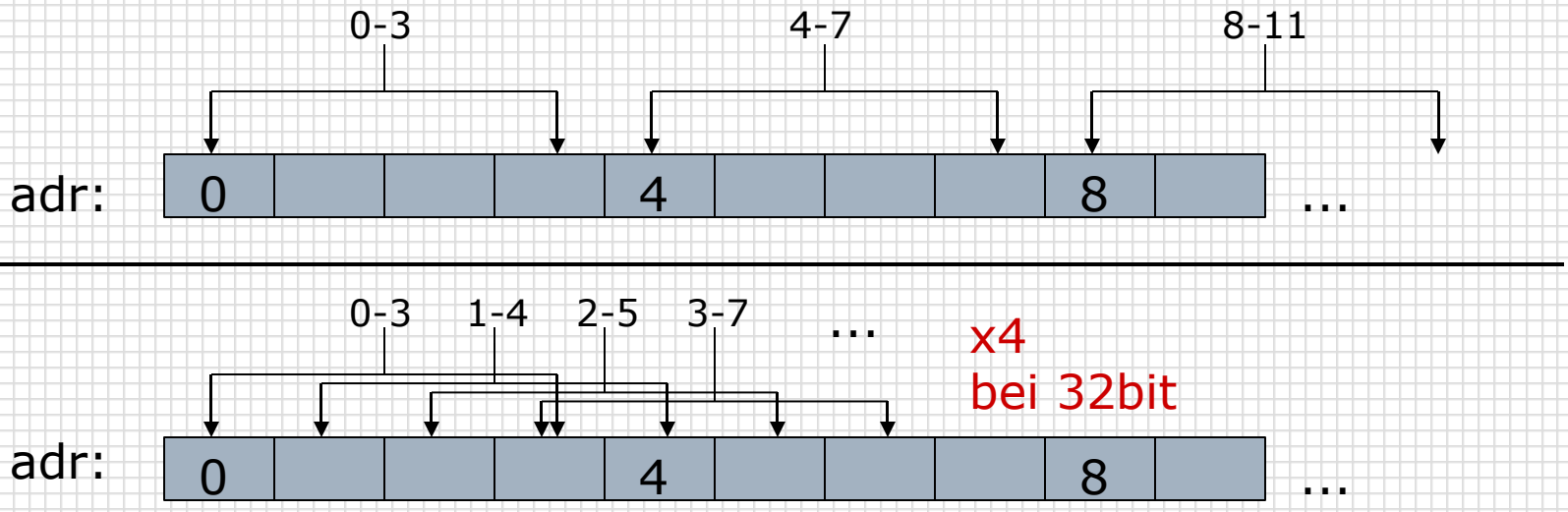
Blacklists

- Referenzen die auf den Heap verweisen, aber Tests nicht bestehen
- Blacklisted Heap-Blocks meiden
- Objekte so anlegen, dass blacklisted Referenz nicht in das Objekt zeigt (interior pointers...)

- Erste Collection bevor Speicher zugewiesen wurde
  - Falsche Referenzen aus statischen Programmteilen eliminieren (auf blacklist)
- Benötigt mehr Speicher
  - Problem, ausreichend grossen Speicher für grosse Objekte zu finden

Pointer Alignment

- Zusätzliches Problem, wenn Zeiger nicht ausgerichtet sein müssen.



Problem: Compileroptimierungen

Interior pointers

```
/* Original */

for (i = 0; i < SIZE; i++) {
    ...x[i]...
}
...x...;


/* Optimized (Strength reduction)*/

xend = x + SIZE;
for (; x < xend; x++) {
    ...*x...;
}
x -= SIZE;
...x...;
```

Hoffnungslos

```
/* Original */

sum = 0;
for (i = 0; i < SIZE; i++) {
    sum += x[i] + y[i];
}


/* Optimized */

sum = 0;
diff = x - y;
xend = x + SIZE;

for (; x < xend; x++) {
    sum += (*x) + (*(x + diff));
}
x -= SIZE;
y = x + diff;
```

Leak Detection

- Spezieller Modus für Boehm-Demers-Weiser Collector
- Bei Aufruf von GC_MALLOC()...
  - Aufrufende Methode merken
  - Debugging Information notwendig
- GC_FREE() anstatt free()
- Nur Mark()
- Quellen von memory leaks können erkannt werden.
- In Verwendung z.B. beim Mozilla Projekt

Performance: CGC vs. Explicit MM [Zorn 93]

- 6 verschiedene Programme
- 5 Management-Strategien
  - 4 Explizite ~ 4 Implementierungen von malloc(), free()
  - 1 Automatische: Boehm-Demers-Weiser Collector

Program	Lines of code	Objects allocated ($\times 10^6$)	Bytes allocated ($\times 10^6$)	Execution time (seconds)	Percentage of time in malloc	Percentage of time in free
cfrac	6000	3·81	65·0	216	10·9	14·4
espresso	15,500	1·66	105	237	8·0	16·4
gawk	8500	4·47	170	144	12·0	26·7
ghostscript	29,500	0·92	89	147	9·7	20·6
perl	34,500	0·36	8·3	174	8·7	14·7
yacr	9000	0·88	28·8	92	28·0	0·1

Performance - CPU

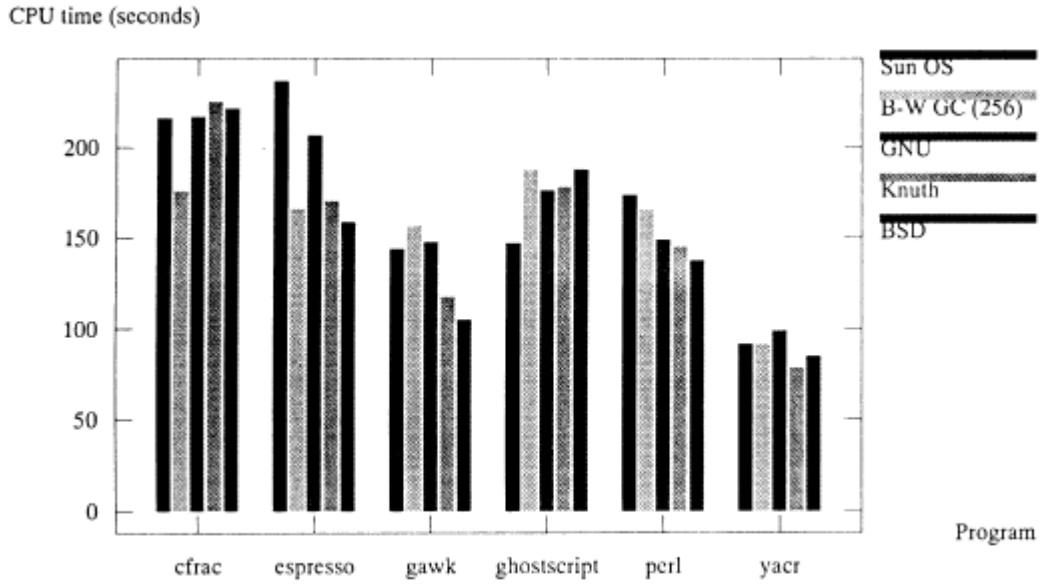
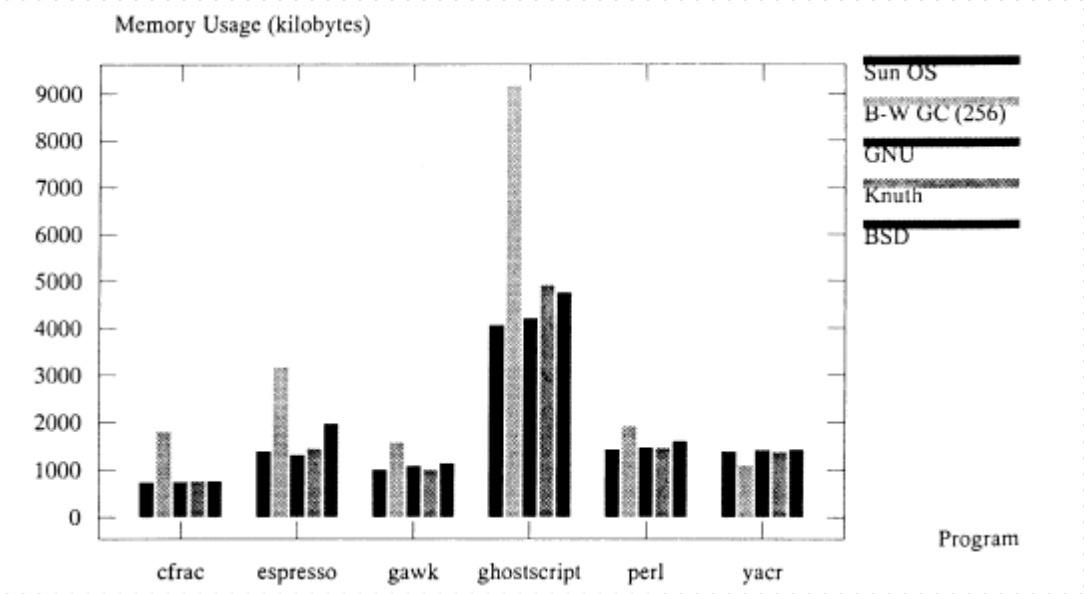


Figure 2. CPU overhead of explicit and automatic forms of storage management

Program	SunOS (seconds)/ SunOS = 1	B–W GC (seconds)/ SunOS = 1	GNU (seconds)/ SunOS = 1	Knuth (seconds)/ SunOS = 1	BSD (seconds)/ SunOS = 1
cfrac	216·2/1·00	175·9/0·81	217·2/1·00	225·3/1·04	221·6/1·03
espresso	236·9/1·00	165·9/0·70	206·9/0·87	170·4/0·72	158·8/0·67
gawk	144·0/1·00	156·6/1·09	147·8/1·03	117·8/0·82	105·2/0·73
ghostscript	147·1/1·00	187·3/1·27	176·1/1·20	178·3/1·21	187·8/1·28
perl	173·7/1·00	165·6/0·95	149·2/0·86	145·1/0·84	137·7/0·79
yacr	91·6/1·00	91·2/1·00	99·2/1·08	78·5/0·86	85·0/0·93

Performance – Speicher



Program	SunOS (kilobytes)/ SunOS = 1	B–W GC (kilobytes)/ SunOS = 1	GNU (kilobytes)/ SunOS = 1	Knuth (kilobytes)/ SunOS = 1	BSD (kilobytes)/ SunOS = 1
cfrac	736/1·00	1807/2·45	748/1·02	760/1·03	761/1·03
espresso	1387/1·00	3166/2·28	1315/0·95	1448/1·04	1974/1·42
gawk	1006/1·00	1581/1·57	1086/1·08	1018/1·01	1134/1·13
ghostscript	4053/1·00	9167/2·26	4206/1·04	4908/1·21	4756/1·17
perl	1422/1·00	1901/1·34	1469/1·03	1470/1·03	1597/1·12
yacr	13,841/1·00	10,944/0·79	14,148/1·02	13,697/0·99	14,245/1·03

Zusammenfassung CGC

- Keine Hilfe durch Compiler/Umgebung
- Kern: Finden/Identifizieren von Referenzen
- Strategie: Referenzen sollten/dürfen nicht geändert werden.
- Probleme:
  - Unsichere Operationen
  - Optimierende Compiler
- Geschwindigkeit
  - Teils besser als bei explizitem MM
  - Abhängig vom konkreten Programm

```
#     ifdef GC_DLL
#       define GC_INIT() { GC_add_roots(GC_DATASTART, GC_DATAEND); }
#     else
#       define GC_INIT()
#     endif
#   endif
#   if defined(_AIX)
      extern int _data[], _end[];
#     define GC_DATASTART ((GC_PTR)((ulong)_data))
#     define GC_DATAEND ((GC_PTR)((ulong)_end))
#     define GC_INIT() { GC_add_roots(GC_DATASTART, GC_DATAEND); }
#   endif
# else
#  if defined(__APPLE__) && defined(__MACH__) || defined(GC_WIN32_THREADS)
#   define GC_INIT() { GC_init(); }
#  else
#   define GC_INIT()
#  endif /* !__MACH && !GC_WIN32_THREADS */
# endif /* !AIX && !cygwin */
#endif /* !sparc */

#if !defined(_WIN32_WCE) \
    && ((defined(_MSDOS) || defined(_MSC_VER)) && (_M_IX86 >= 300) \
        || defined(_WIN32) && !defined(__CYGWIN32__) && !defined(__CYGWIN__))
  /* win32S may not free all resources on process exit.  */
  /* This explicitly deallocates the heap.       */
    GC_API void GC_win32_free_heap ();
#endif

#if ( defined(_AMIGA) && !defined(GC_AMIGA_MAKINGLIB) )
  /* Allocation really goes through GC_amiga_allocwrapper_do */
# include "gc_amiga_redirects.h"
#endif

#if defined(GC_REDIRECT_TO_LOCAL) && !defined(GC_LOCAL_ALLOC_H)
#  include  "gc_local_alloc.h"
#endif

#ifdef __cplusplus
```

Ende...