Precise Garbage Collection for C

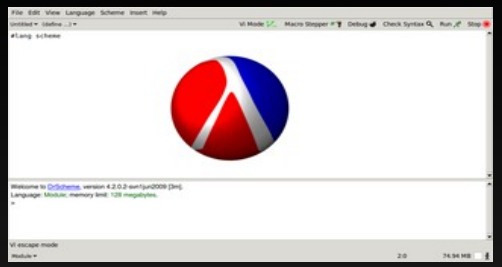
Jon Rafkind*
Adam Wick+
John Regehr*
Matthew Flatt*

* University of Utah
+ Galois, Inc.

Motivation

C

- Used to implement important programs
  - Web servers
  - Operating systems
  - Virtual machines



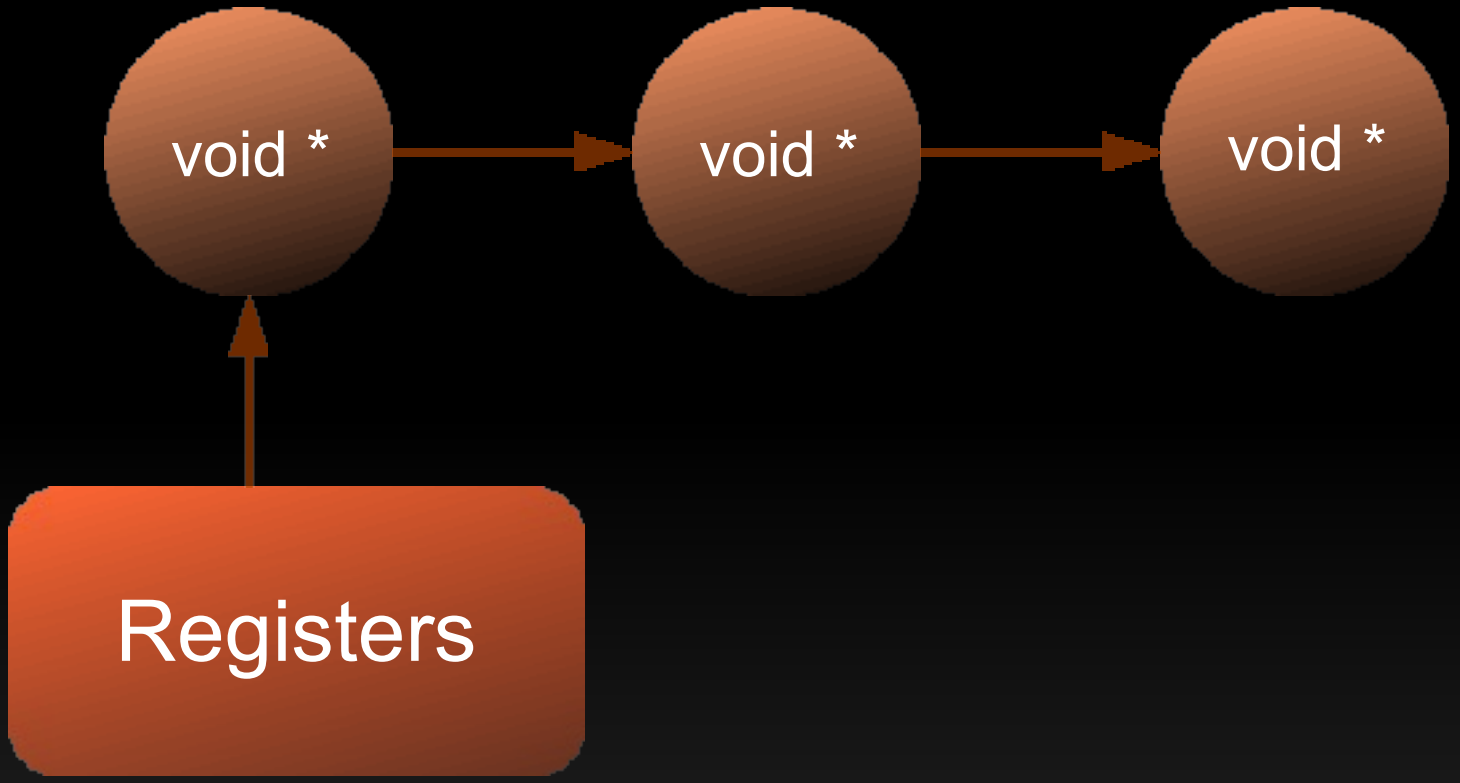
- Memory management is difficult to get right

Conservative GC

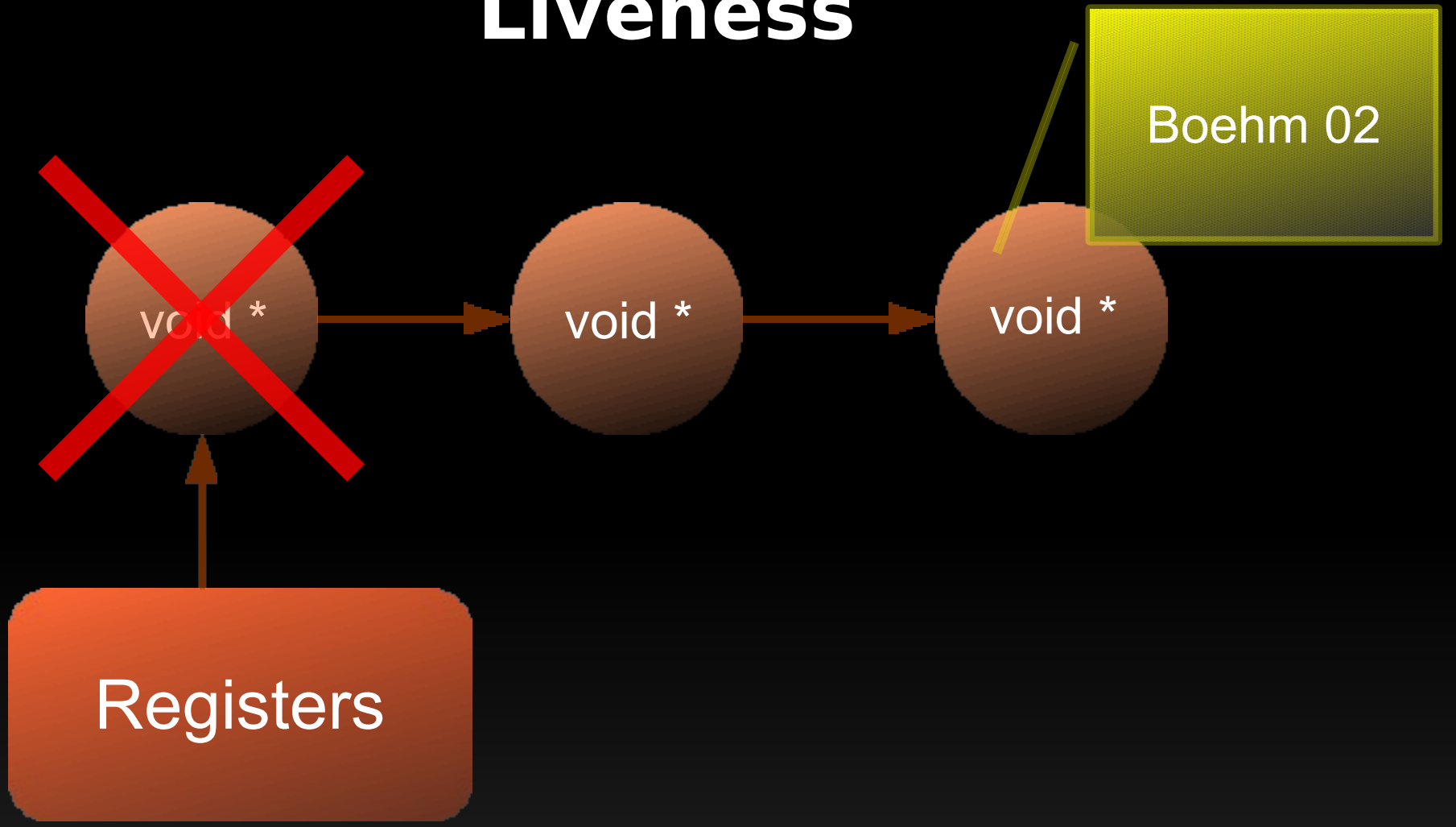
Scans for pointer values

- Works well for weakly typed languages
  - Objects are stored in memory as sequences of bytes
- Unobtrusive to the original program
- Works for many C programs

Liveness



Liveness



Long running programs have possibility of liveness errors

Liveness inaccuracy

Machine state outside the realm of C

Registers

Non-pointer values as pointers

void * x	0xba324100
long y	0xbd100000

Mitigating liveness

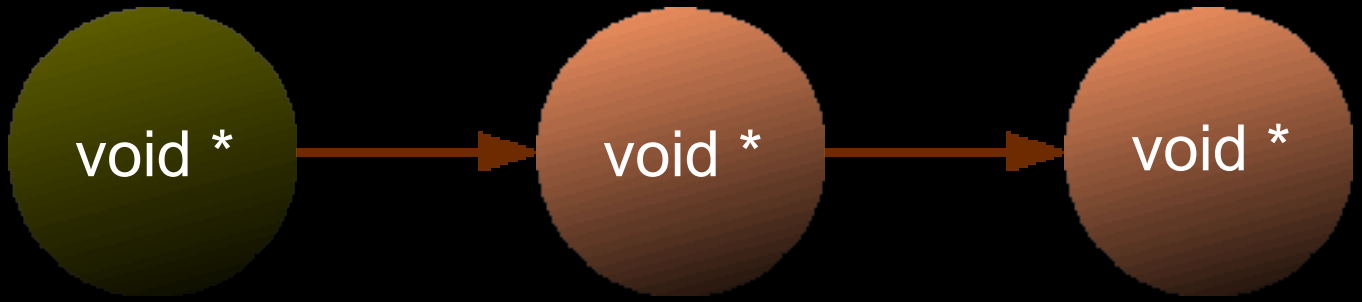
Atomic blocks (memory regions with no pointers)

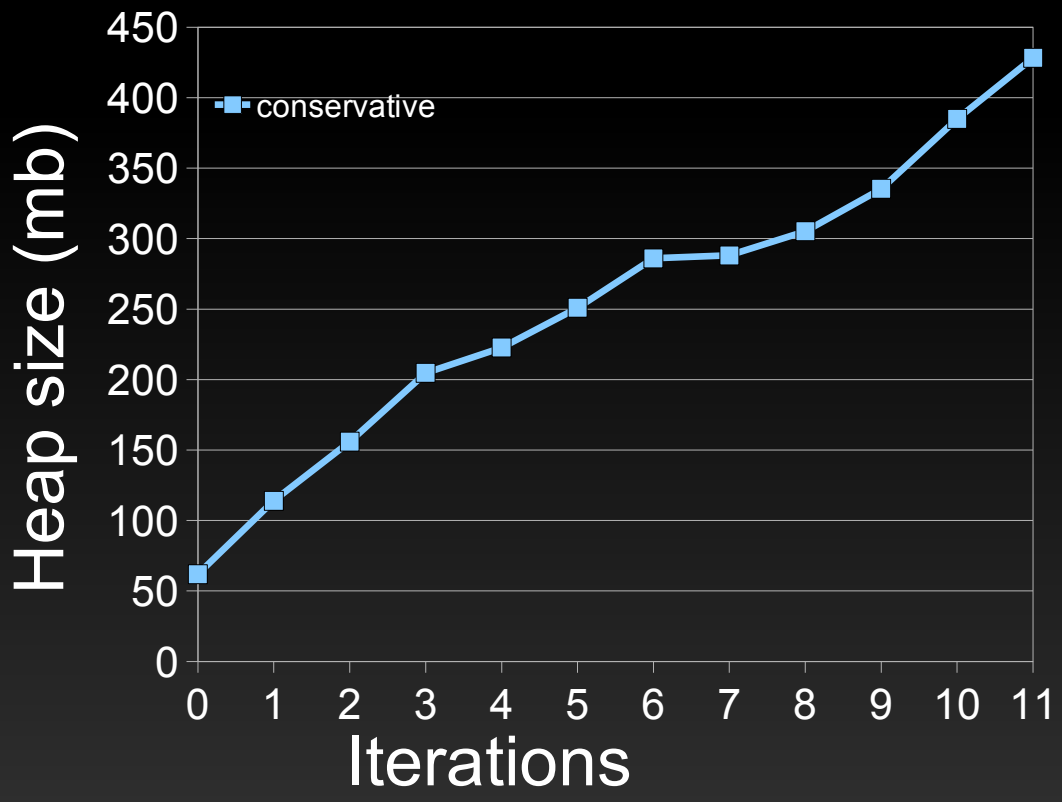
Custom tracers

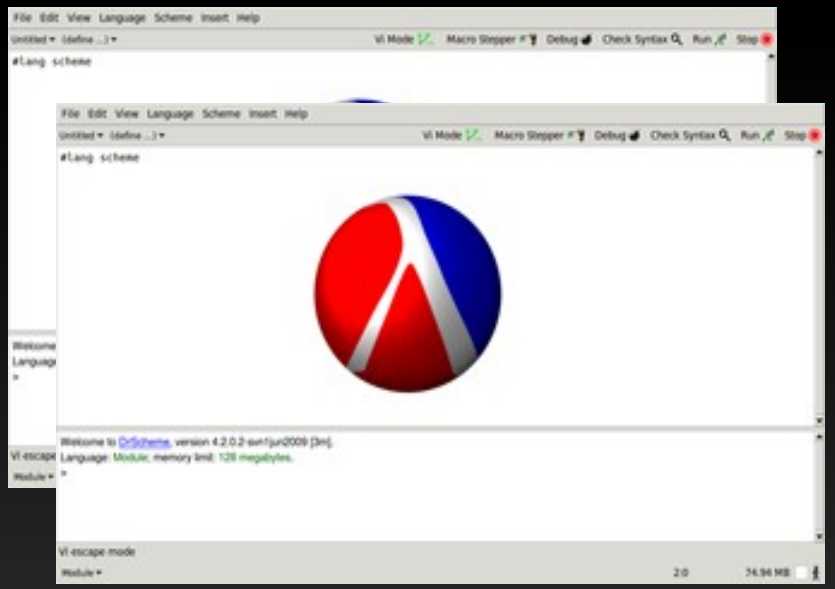
Liveness errors have non-zero chance of occurring

Memory leaks

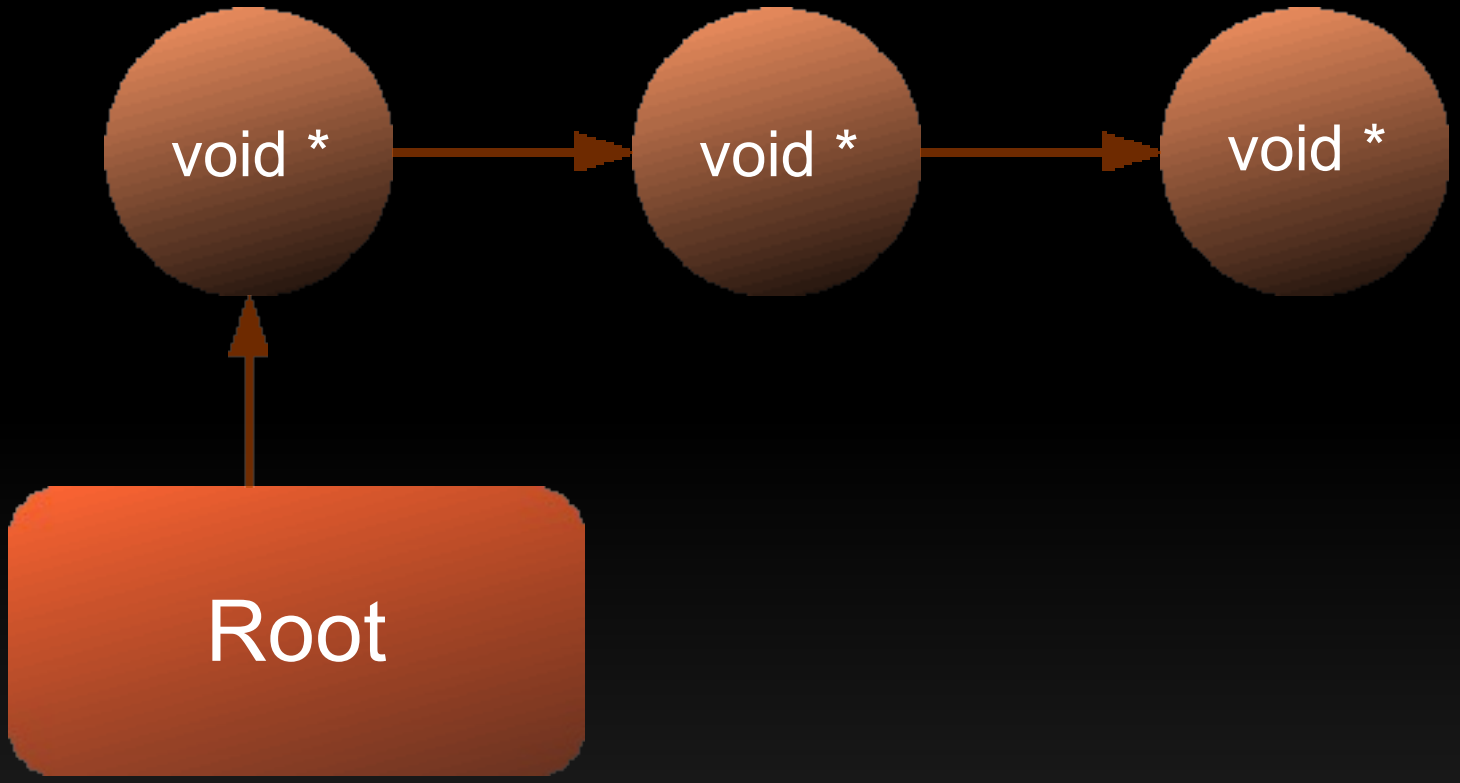
thread 1 thread 2 thread 3







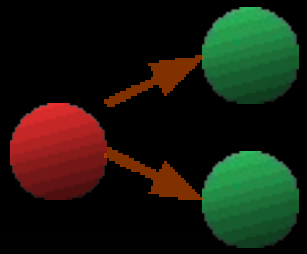
Precise GC



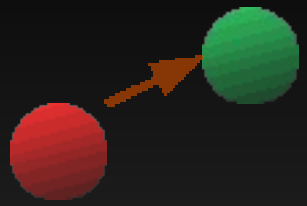
Precise GC

Liveness is apparent at the source level

```
struct painter * p = ...;
p->brush = ...;
p->color = ...;
```



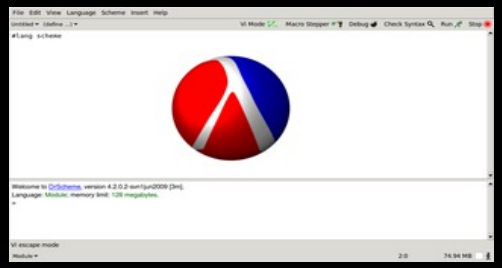
```
p->brush = 0;
```



Precise GC

Successfully added a precise garbage collector to *long running* C programs

PLT Scheme



ZSnes Emulator

Nano text editor

Linux

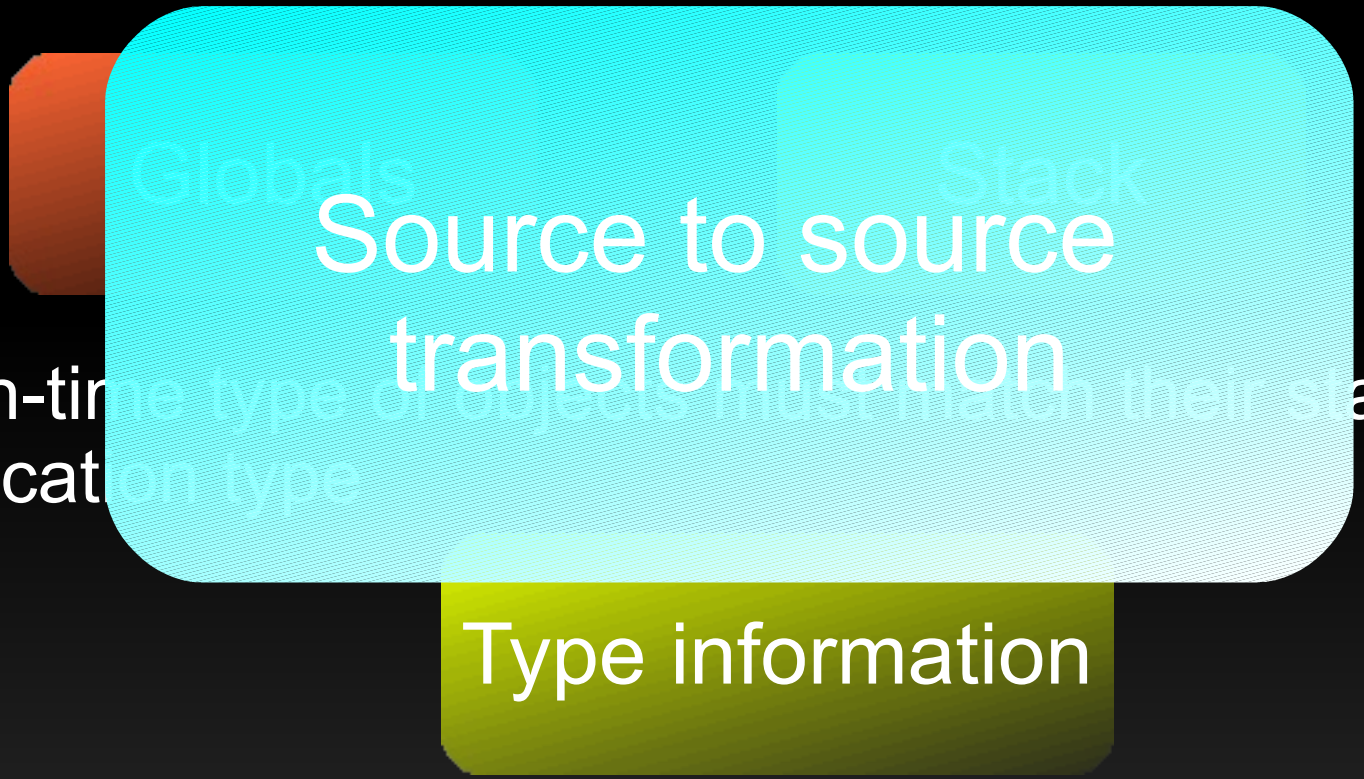
Precise

All live objects must be reachable through pointers starting from roots

Run-time type of objects must match their static allocation type

Precise for C

All live objects must be reachable through pointers starting from roots



Run-time type of objects must match their static allocation type

Insight

C programs provide enough type information to precisely distinguish pointers from non-pointers

Transformation

- Register global variables as roots
- Dynamically register stack variables
- Rewrite allocations with run-time information
- Add traversal routines for structures

Precise for C

All live objects must be reachable through pointers starting from roots

Globals

Stack

Run-time type of objects must match their static allocation type

Type information

Roots

```
int * counter;
int * maker(int * value){
  int * x = malloc(10);
  *x = *value;
  *counter += 1;
  return x;
}
```

Shadow stack

Henderson 02

```
int * counter;
int * maker(int * value){

  int * x = malloc(10);
  *x = *value;
  *counter += 1;

  return x;
}
```

```
void * frame[3];
frame[0] = POINTER;
frame[1] = &x;
frame[2] = &value;
GC_set_stack(frame);
```

Shadow stack

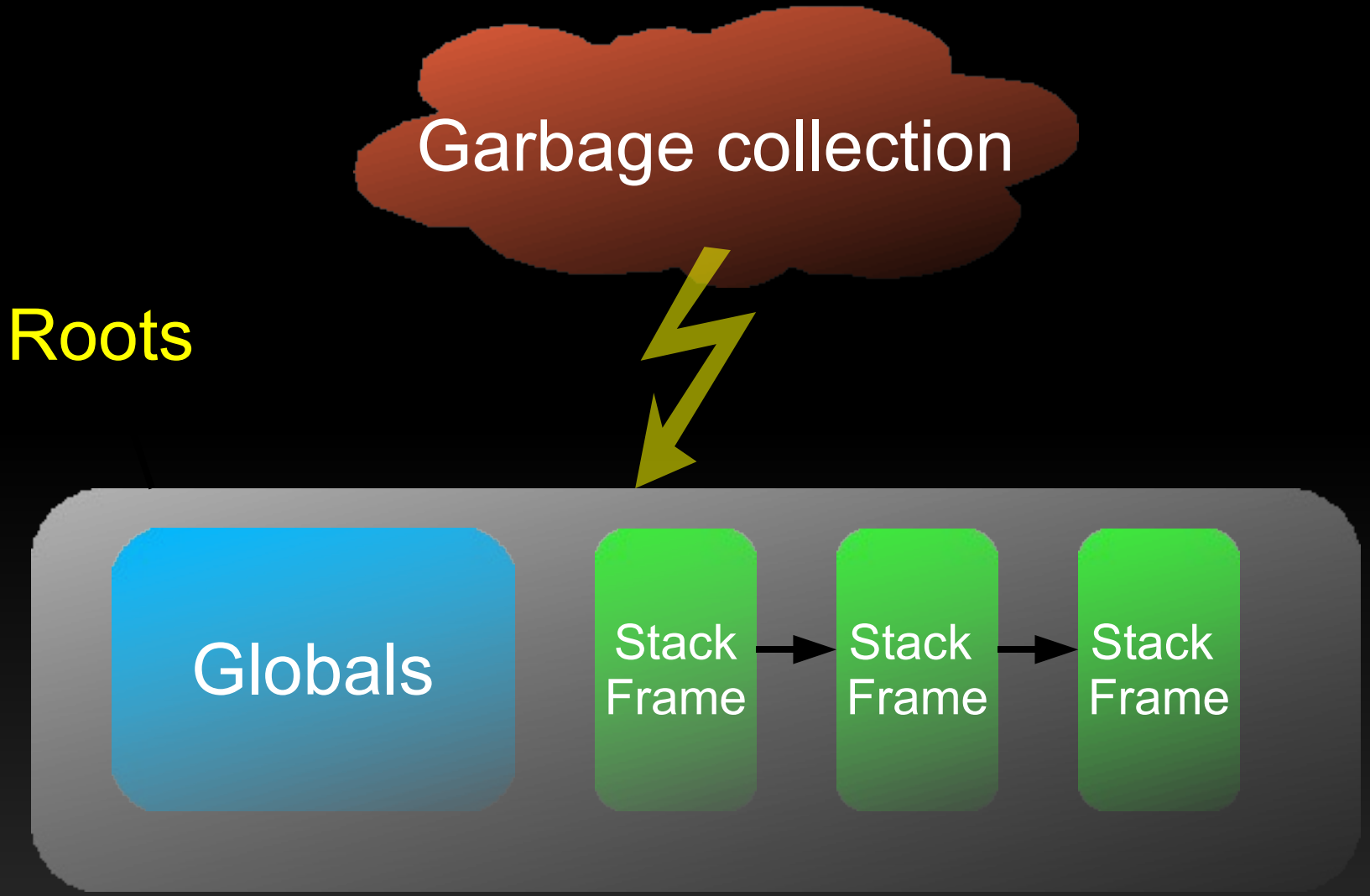
Henderson 02

```
int * counter;
int * maker(int * value){

  int * x = GC_malloc(10);
  *x = *value;
  *counter += 1;

  return x;
}
```

```
void * frame[3];
frame[0] = POINTER;
frame[1] = &x;
frame[2] = &value;
GC_set_stack(frame);
```

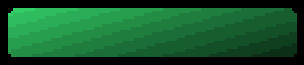
```
GC_set_stack(last_stack);
```

Shadow stack

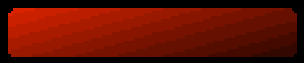


Stack optimizations

Call graph analysis detects functions that do not need shadow stacks



Potentially allocates



Does not allocate

Gzip: 80% non-allocating
H264: 40% non-allocating

Precise for C

All live objects must be reachable through pointers starting from roots

Globals

Stack

Run-time type of objects must match their static allocation type

Type information

Allocation rules

Heuristic used to statically detect run-time type

var = malloc(<expr>);

sizeof(t) – allocate single t structure
sizeof(t) * *e* – allocate array of t structures
sizeof(t)* * *e* – allocate pointer array
e – allocate atomic block

Transforming allocations

var = malloc(sizeof(struct cat));

sizeof(t) – allocate single t structure
*sizeof(t) * e* – allocate array of t structures
sizeof(t) * e* – allocate pointer array
e – allocate atomic block

var = GC_malloc(gc_tag_struct_cat, sizeof(struct cat));

Traversal functions

```
struct cat {
  int * paws;
  char * nose;
};

void gc_struct_cat_mark(struct cat * c){
  GC_mark(c->paws);
  GC_mark(c->nose);
}

void gc_struct_cat_repair(struct cat * c){
  GC_repair(&c->paws);
  GC_repair(&c->nose);
}
```

Used during mark phase

Generate traversal functions

Updates references to moved objects

Result of transformation

The transformed source program will operate semantically equivalent to what it did before and is capable of supporting a moving collector

C Support

Internal pointers

```
Int * x = obj->x;
```

Pointer arithmetic

```
char * p = ...;
while (*p){
   p++;
}
```

Unions

Active variant is tracked

```
union object {
  int value;
  void * info;
};

union object o;
o.value = 1;
GC_autotag(&o, 0);
o.info = get();
GC_autotag(&o, 1);
```

External libraries

Libraries that store pointers

- Annotate objects that should not be moved by the collector

Memory allocated by libraries

- Safe to use because the GC will ignore it

Unsupported C

Pointer obfuscation

```
int * p = malloc(20);
```

GC cannot traverse the pointer

Constructing pointers

```
int * x = (int*) 0x323429;
```

Liveness issues

Ruby

Pointers as integers

```
long make(){
```

GC cannot find pointer

Perl

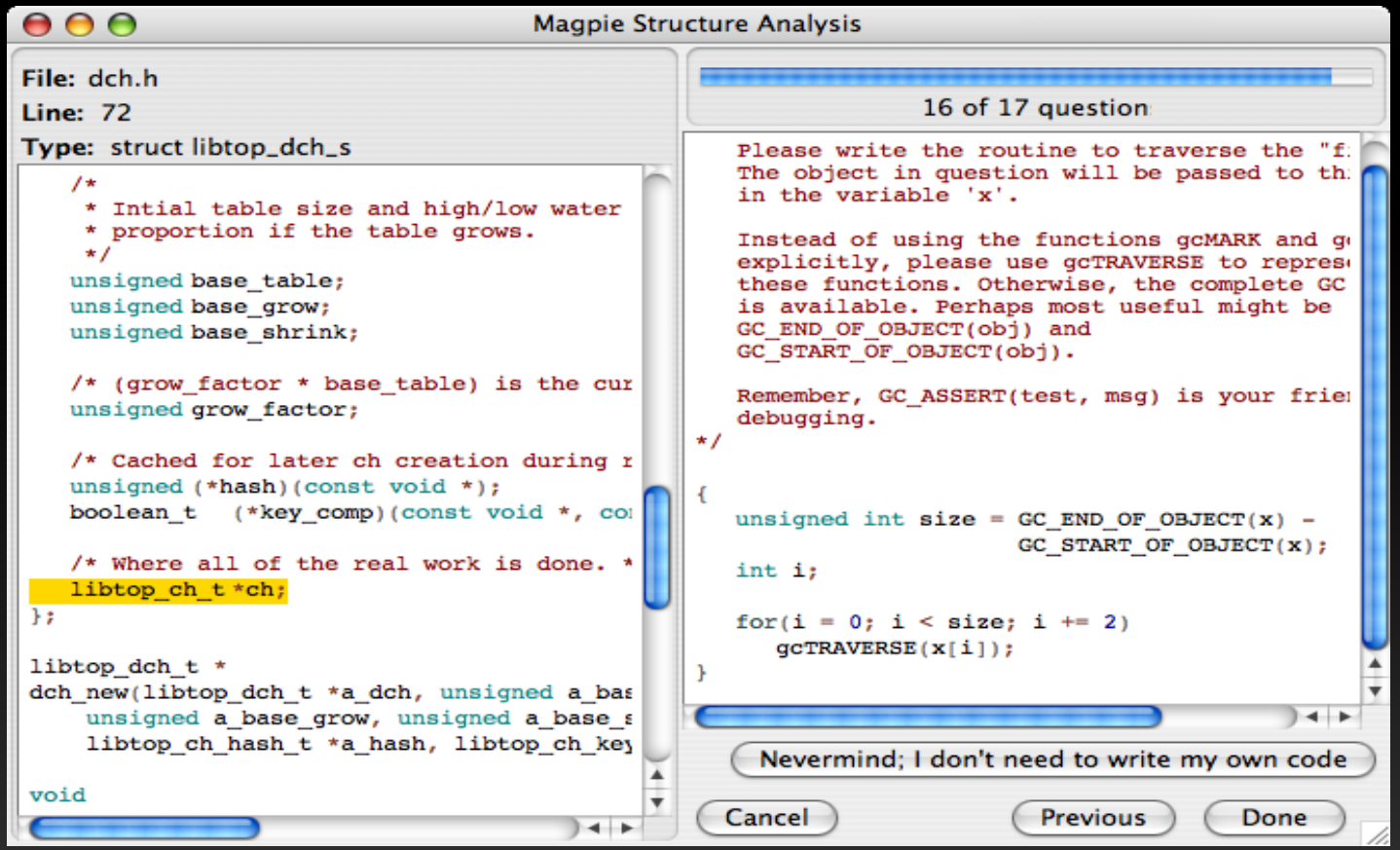
Arrays of open sized arrays

```
};
```

GC doesn't know how large individual elements are

Tool support

GUI that verifies allocation heuristic and traversal functions



Tool support

Two tools that can be used on C programs

- ~2500 line Ocaml program based on CIL (Necula 2002)
- Capable of parsing the Linux kernel source
- Fully automatic

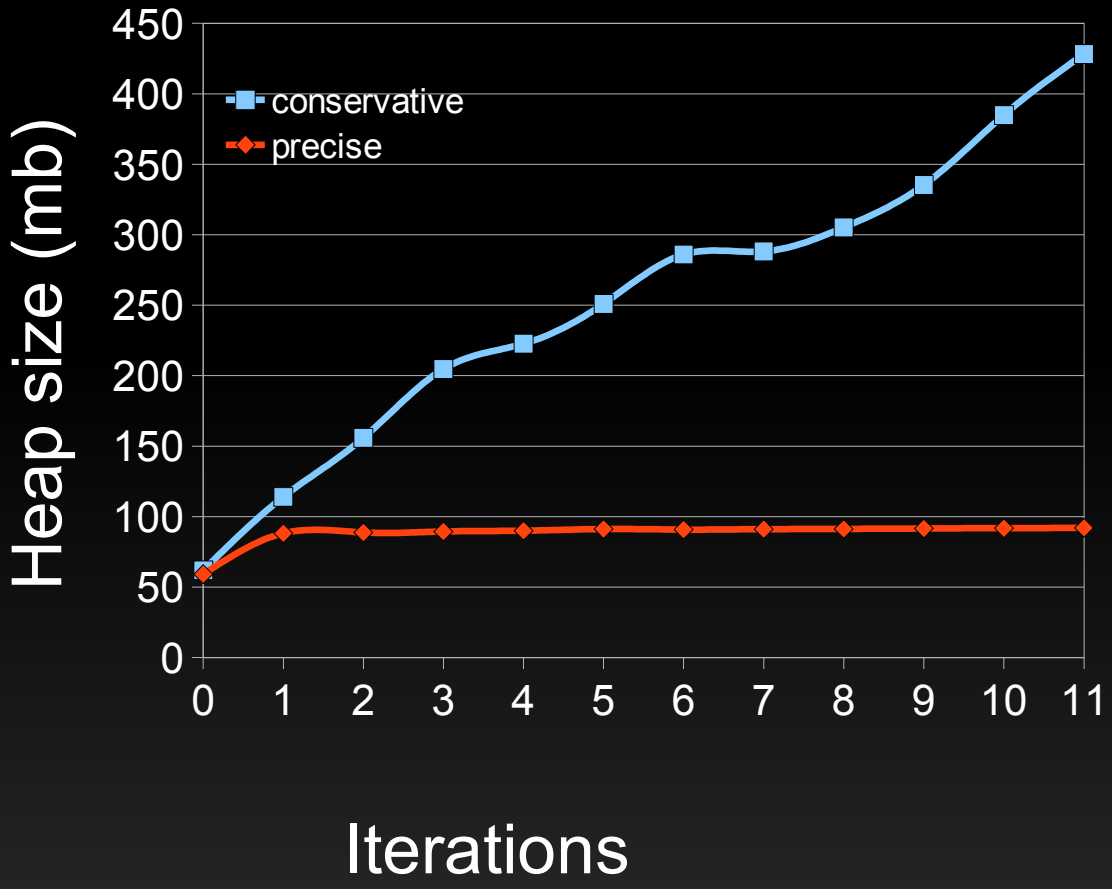
Benchmarks

Precise vs Conservative

- memory
- runtime

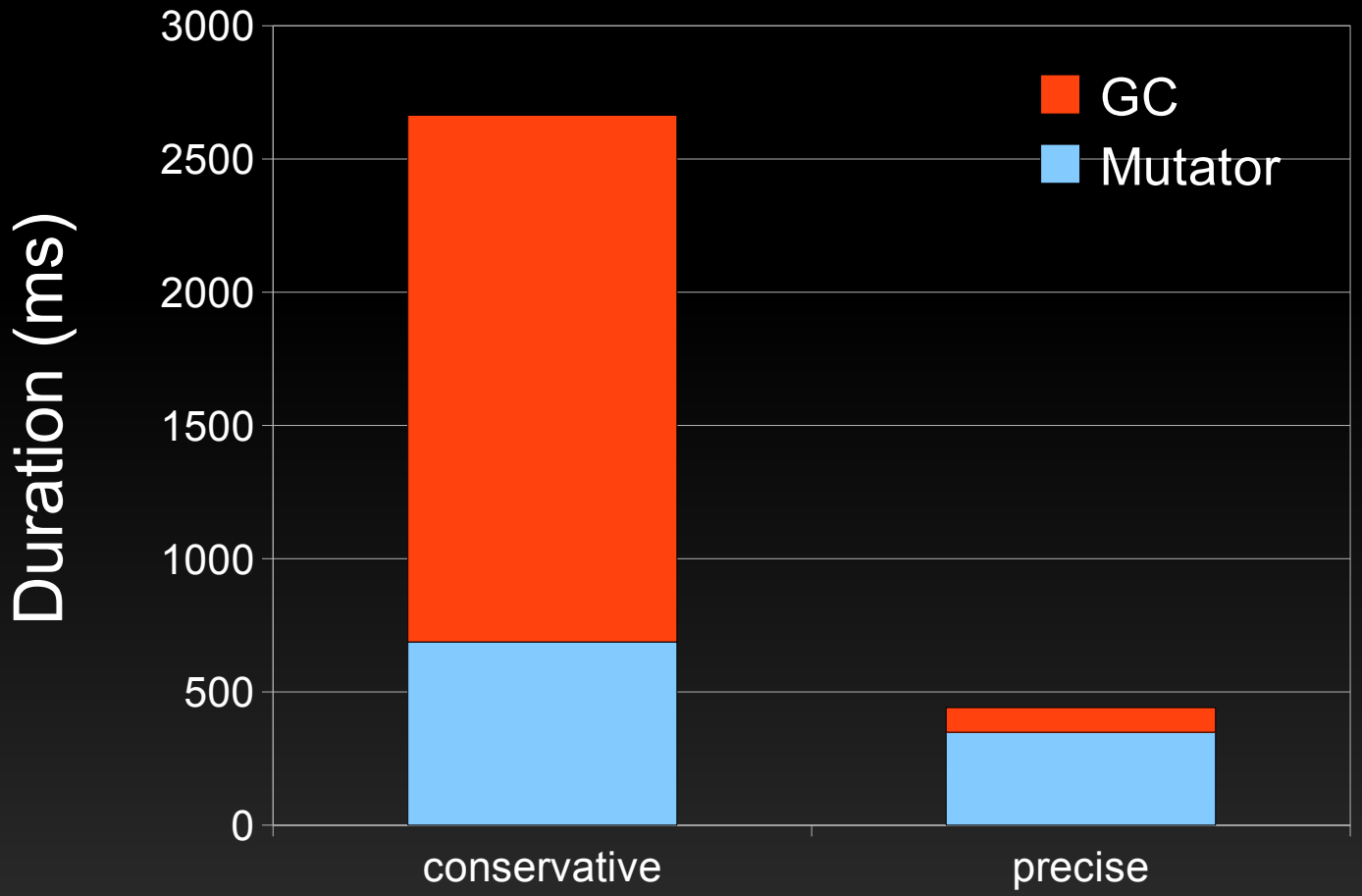
Source transformation overhead

Precise is bounded



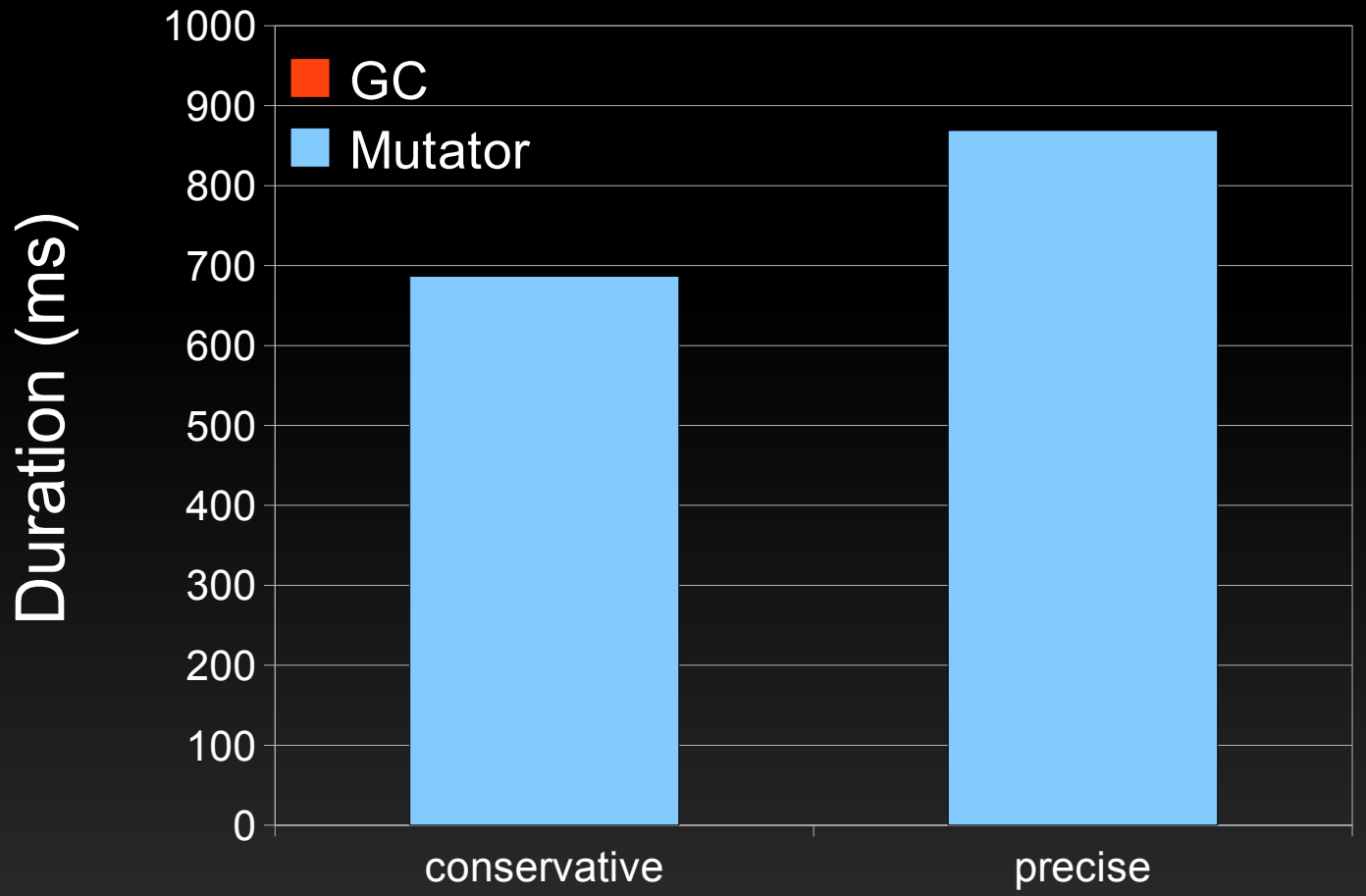
PLT Scheme Performance

Benchmark: CPStack



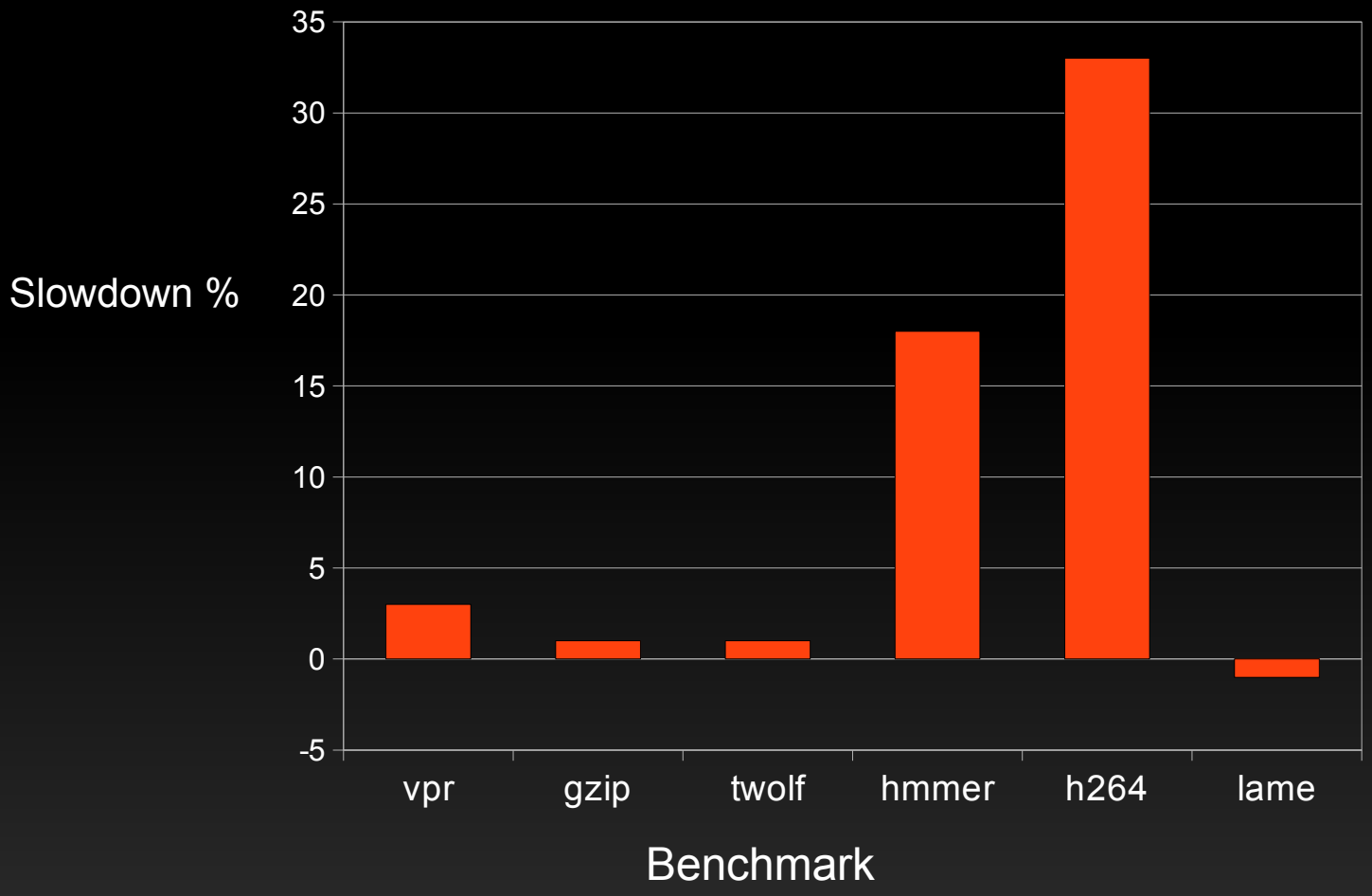
PLT Scheme Performance

Benchmark: Takl



GC overhead

Spec 2k benchmarks



Linux kernel

- Test the limits of precise for C
- Longest running program in a system
- Harsh C environment

Linux kernel: Implementation

Transformed 74,975 lines of C code

- Ext3 and ipv4

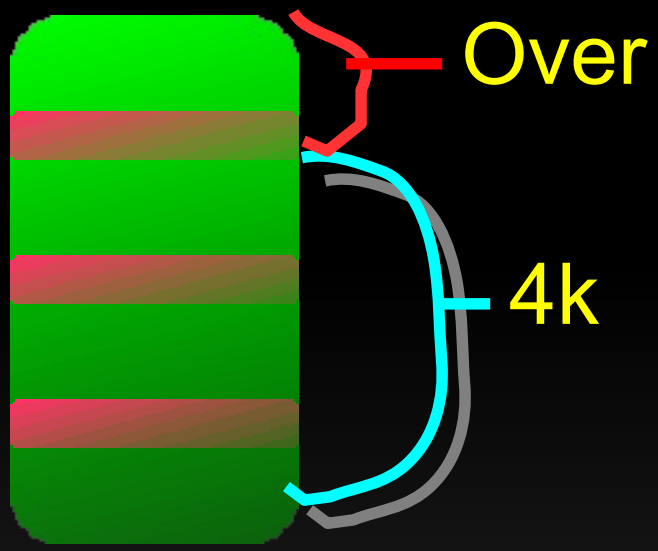
85 manual changes

GC is non-moving / non-generational

- Interface to non-transformed modules

Linux kernel: Stack

Stack limit reached with additional shadow stack information



Linux kernel: Memory regions

Separate memory regions

kmalloc

Physically contiguous

vmalloc

Virtually contiguous

GC only returns physically contiguous memory

Linux kernel: heuristics

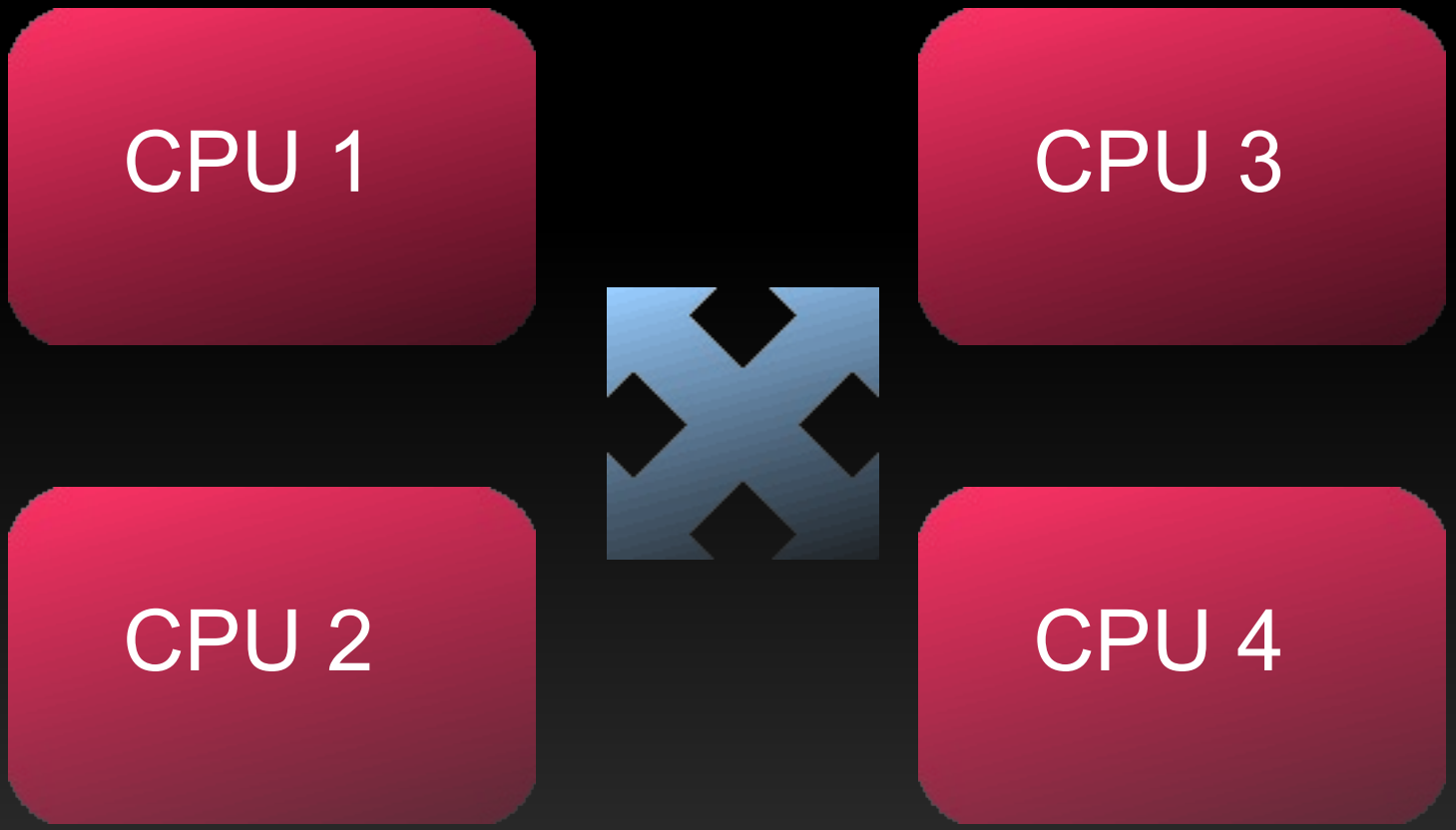
Allocation heuristic broken

X = kmalloc(sizeof(...))
Y = kmem_cache_alloc(cache)

Type of Y is cannot be discerned

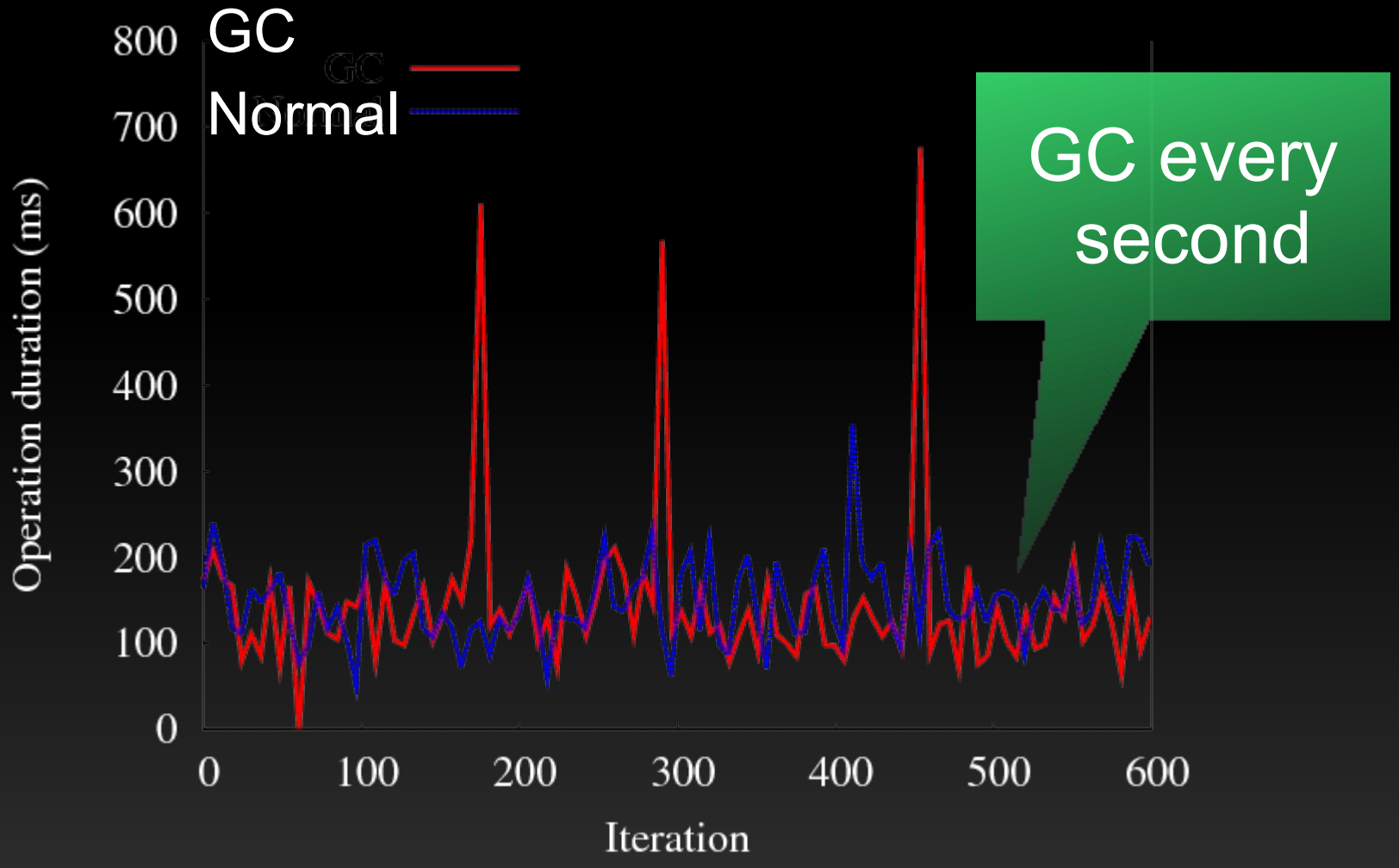
Linux kernel: finalizers

RCU (Read-Copy-Update) is a finalizer mechanism in Linux



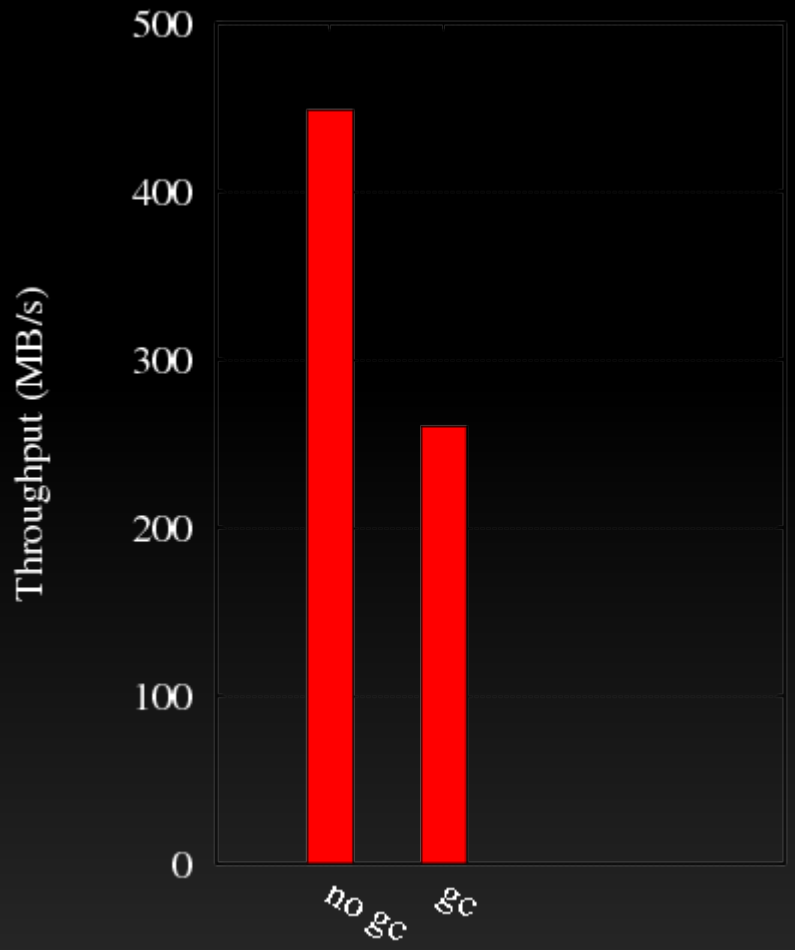
Performance

dbench filesystem benchmark

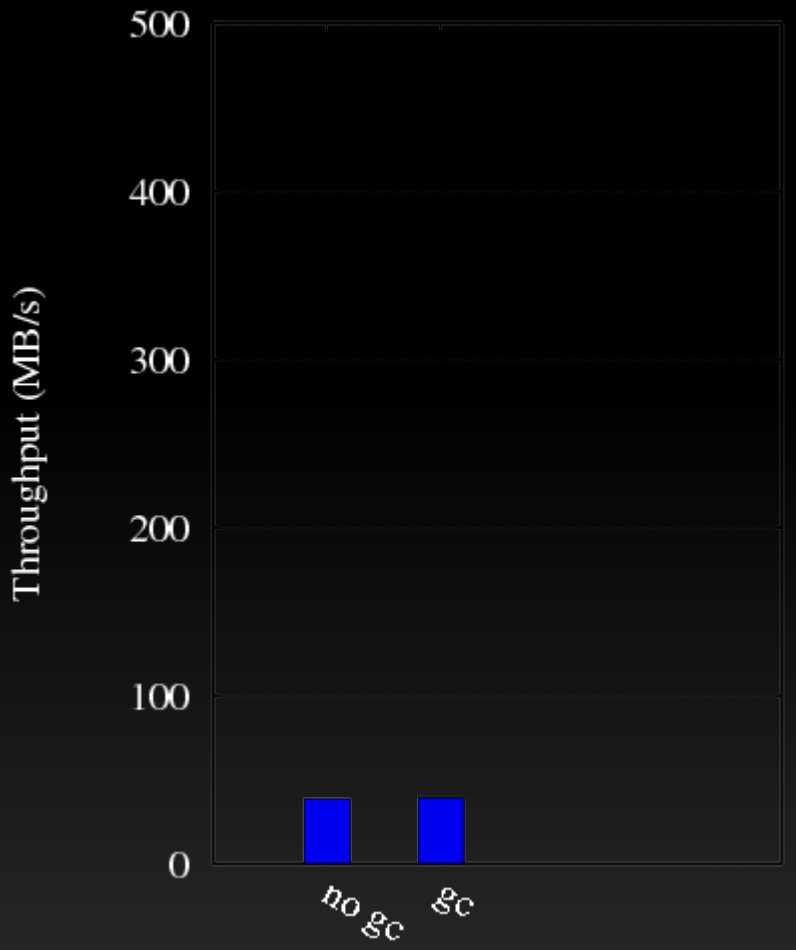


Performance

dd on memory filesystem



dd on disk filesystem



Conclusions

- Precise garbage collection is a practical for C because most C programs embed enough information required to extract the required properties
- Long running programs benefit from precise garbage collection

Thank you

rafkind@cs.utah.edu

Precise Garbage Collection for C

Jon Rafkind

Adam Wick

John Regehr

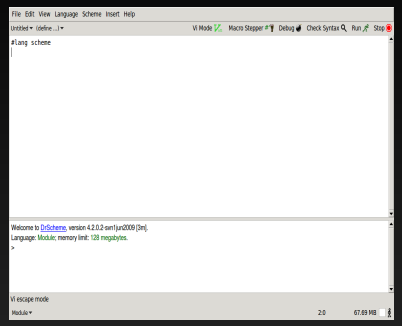
Matthew Flatt

C is a weakly typed language

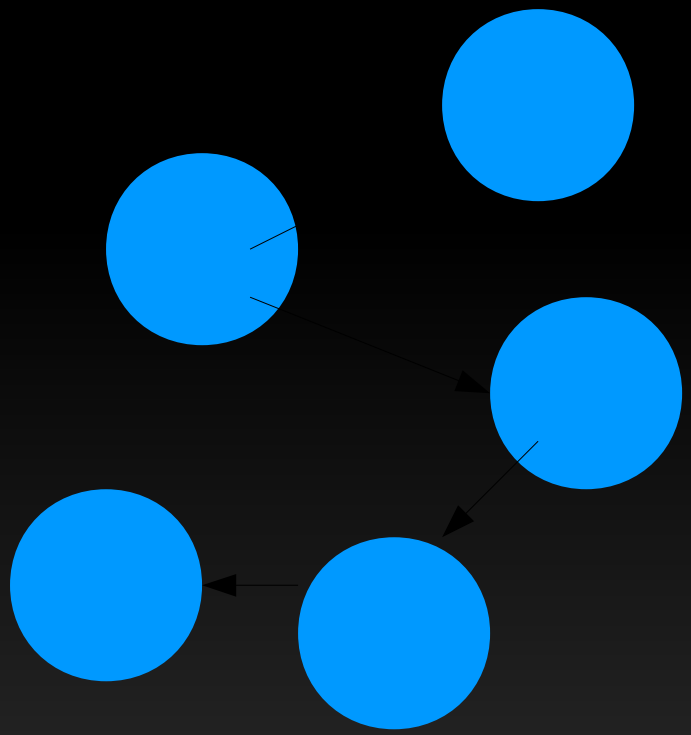
Magpie

Written a tool that makes C programs use precise garbage collection

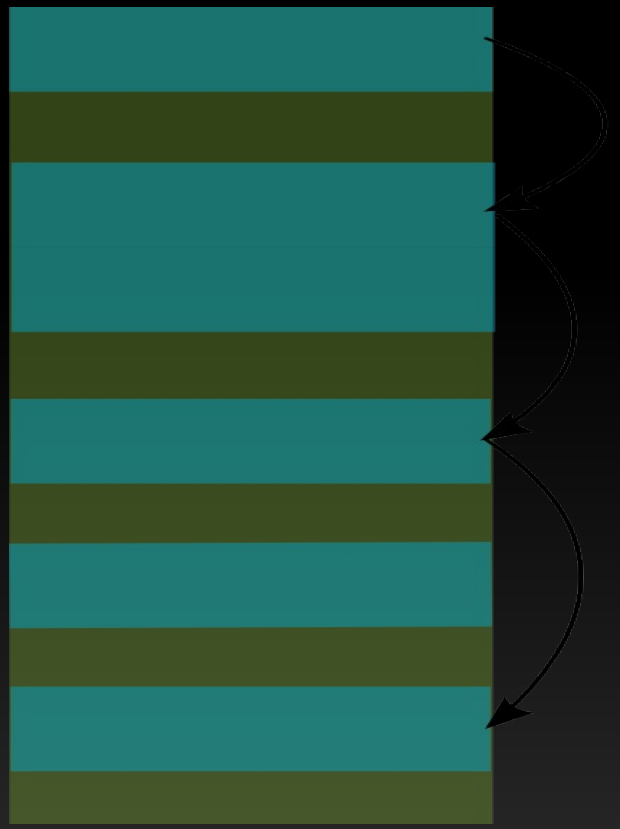
Applied to PLT Scheme, Linux kernel, benchmarks



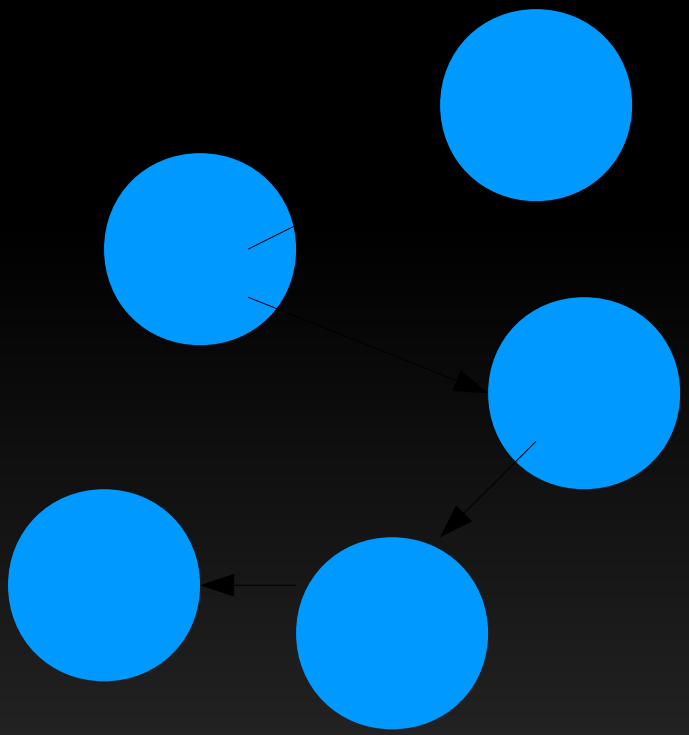
Precise collection



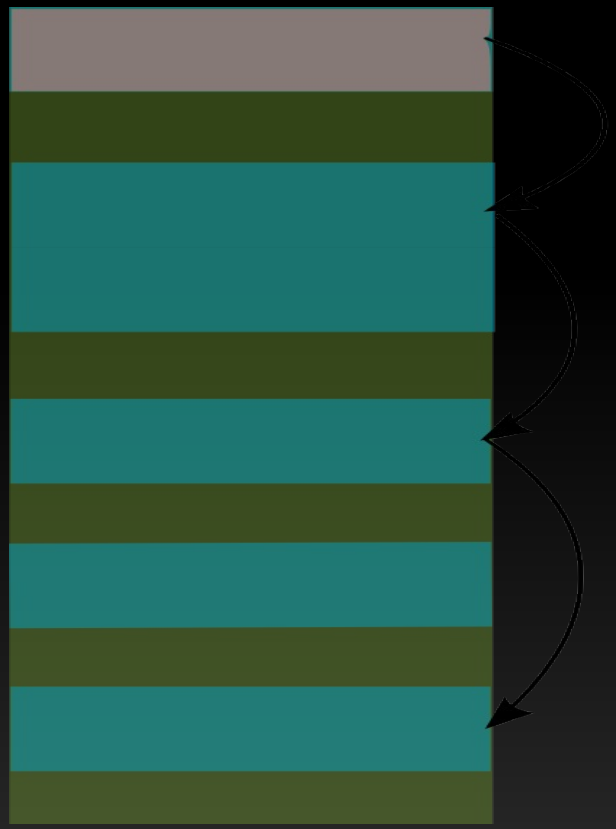
Conservative collection



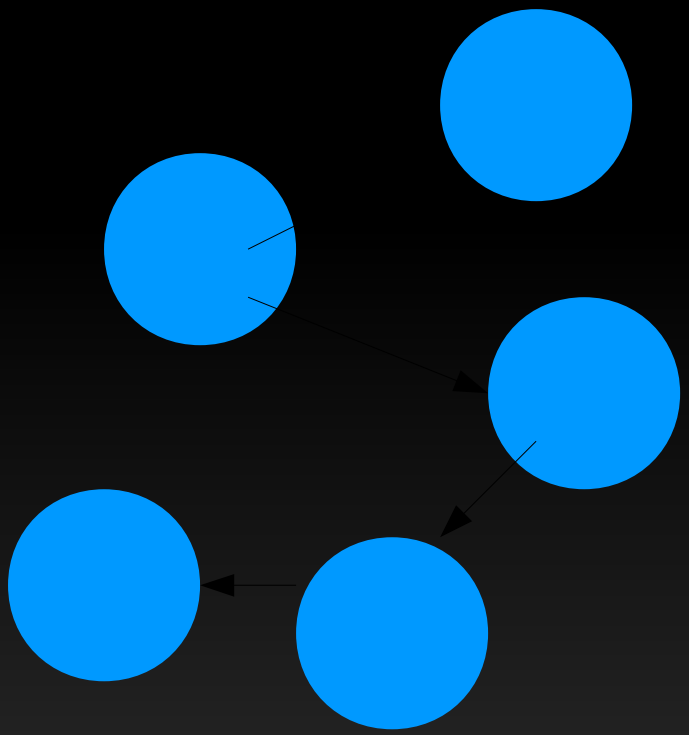
Precise collection



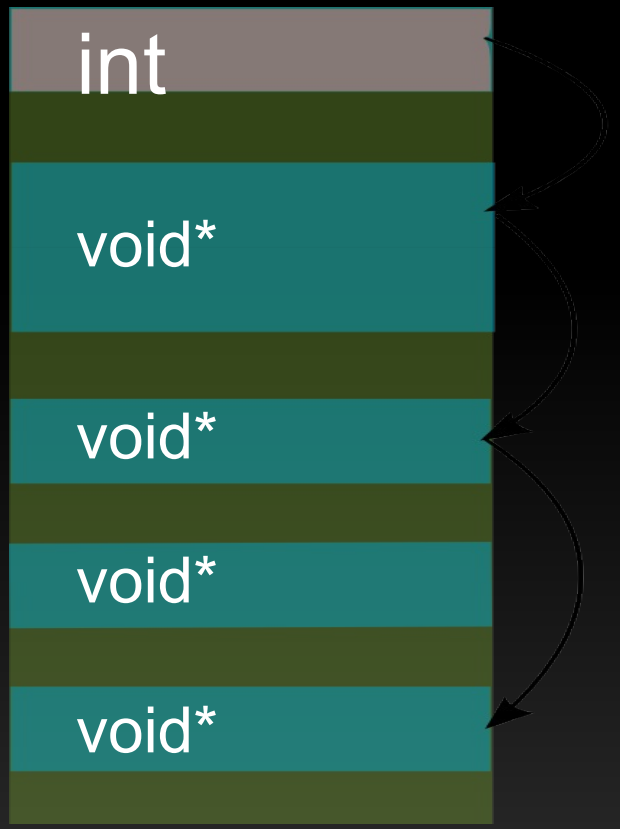
Conservative collection



Precise collection

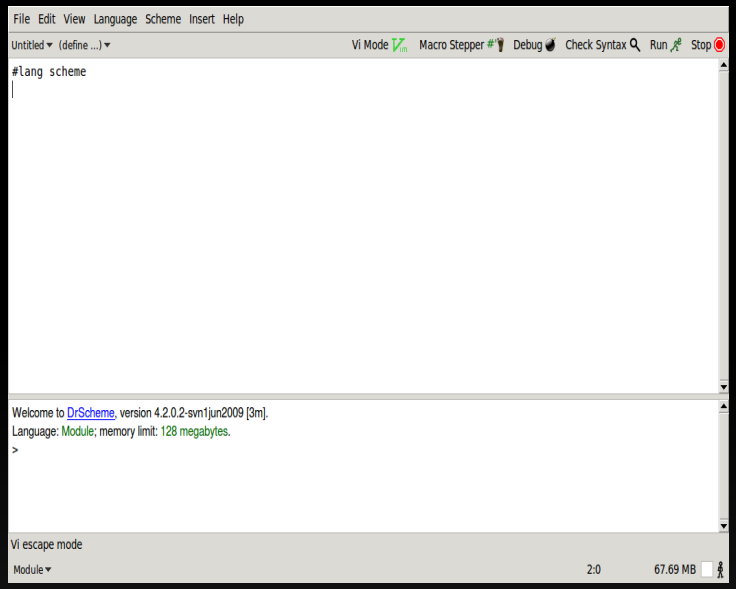


Conservative collection



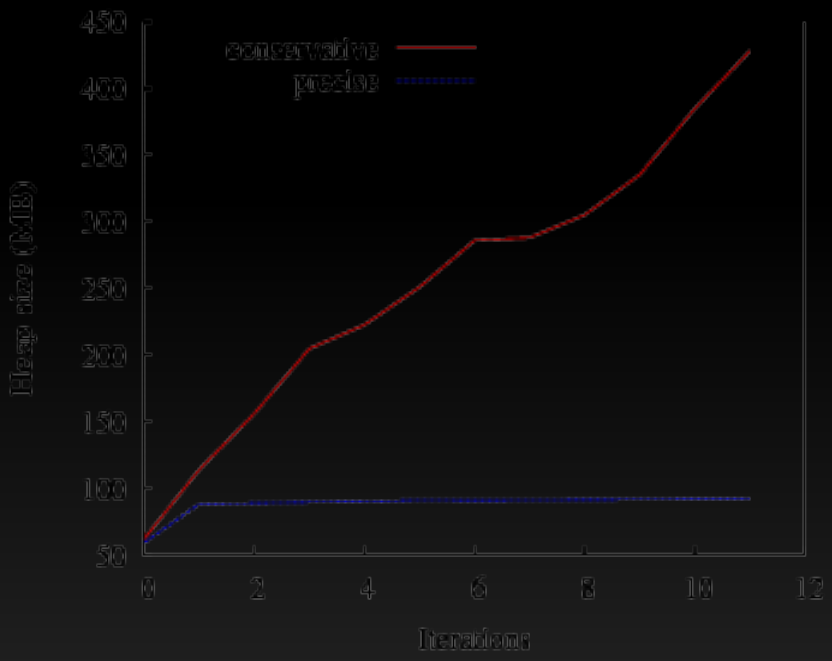
Lists of threads and stacks create unreclaimable chains

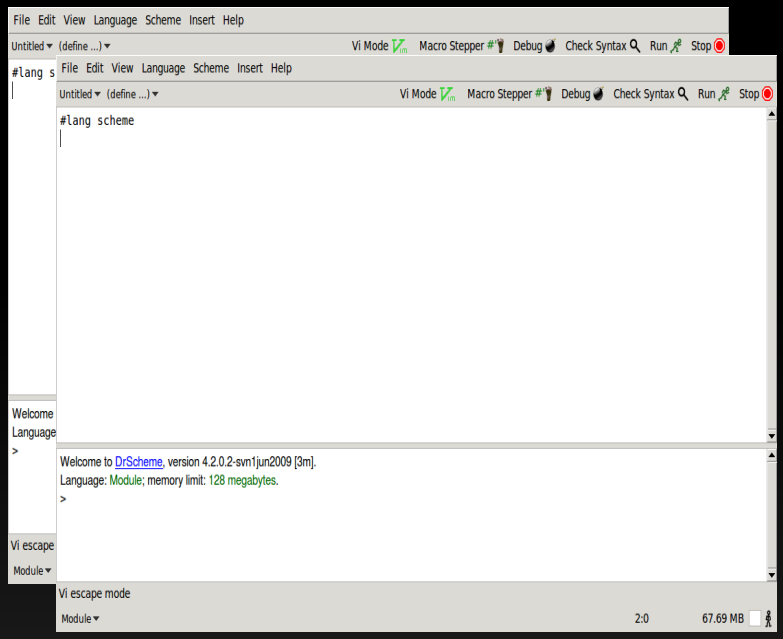
Web servers, OS's, etc can experience unstable memory usage with conservative collectors



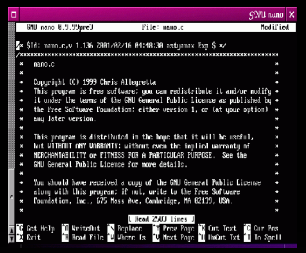
- Core of Drscheme written in C
- 10+ years of managing Boehm collector

Certain classes of programs are susceptible to memory leaks





ZSNes emulator – Super Nintendo emulator



Nano – text editor

Unbounded memory with conservative, stable with precise

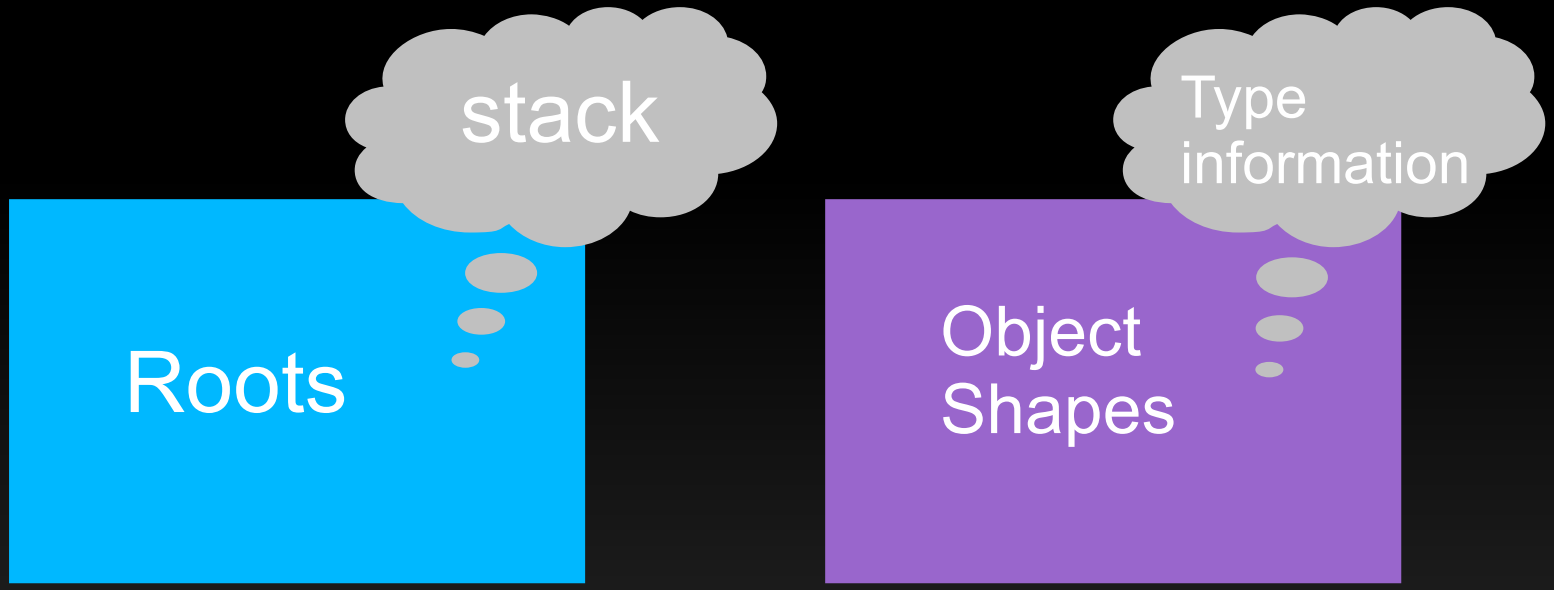
Source to source transformation of C code can embed information to let precise garbage collection work:

Invariants:

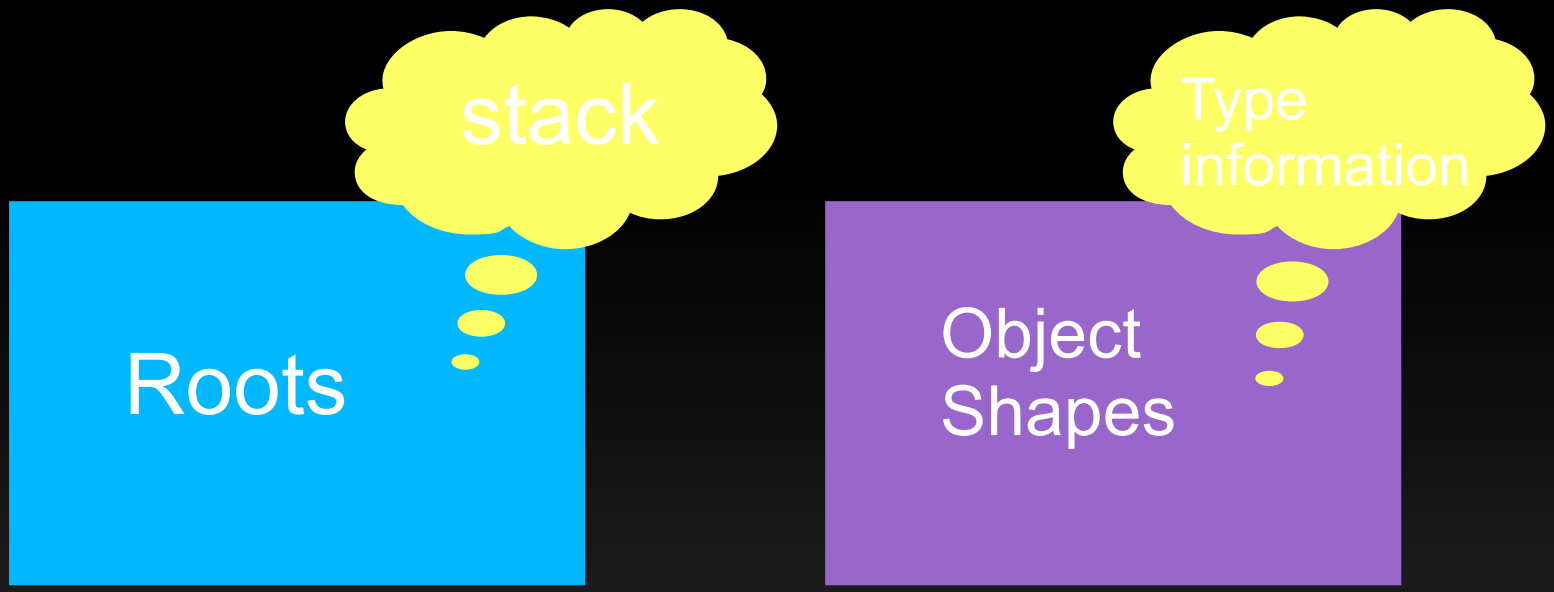
All pointers are known

Run-time type of objects is the same as their static allocation type

- Registers are not scanned for pointers
- All live pointers in the system are known



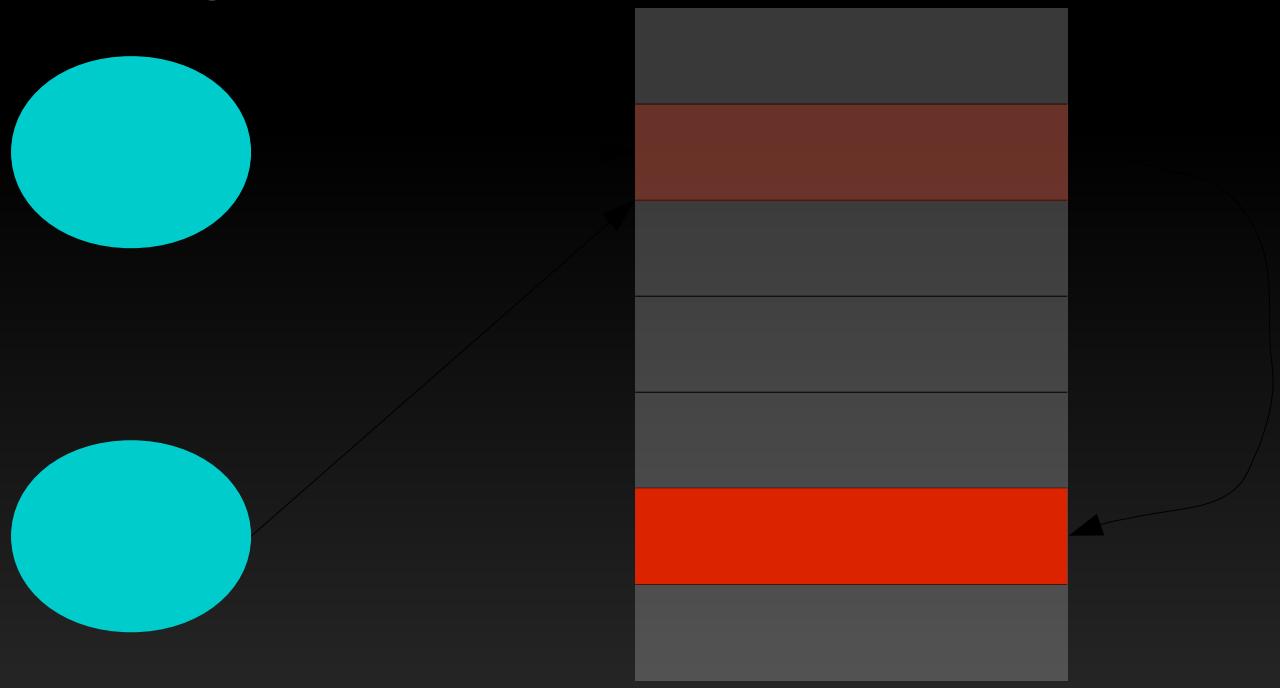
Source transformation to retain this information



var = malloc(<expr>);

sizeof(t) – allocate single t structure
sizeof(t) * *e* – allocate array of t structures
sizeof(t)* * *e* – allocate pointer array
e – allocate atomic block

Traverse objects:
•Marking
• Updating pointers



```
struct cat{
  int * paws;
  char * nose;
};


void gc_struct_cat_mark(struct cat * c){
  GC_mark(c->paws);
  GC_mark(c->nose);
}

void gc_struct_cat_repair(struct cat * c){
  GC_repair(&c->paws);
  GC_repair(&c->nose);
}
```

Generate traversal functions

Active variant is tracked

```
union object{
  int value;
  void * info;
};

union object o;
o.value = 1;
GC_autotag(&o, 0);
o.info = get();
GC_autotag(&o, 1);
```

Pointer obfuscation

```
int * p = malloc(20);
p - = 10;
p[10] = 2;
```

Pointers as integers

```
int make(){
  return malloc();
}
```

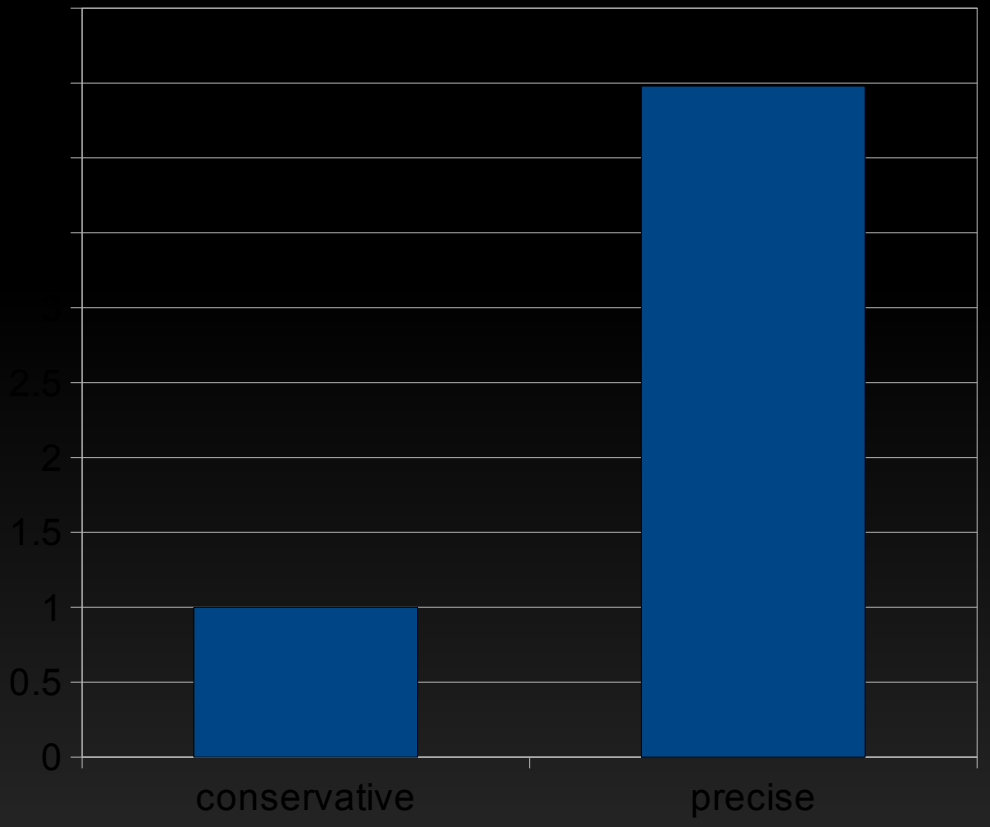
Constructing pointers

```
Int * x = (int*) 0x323429;
```

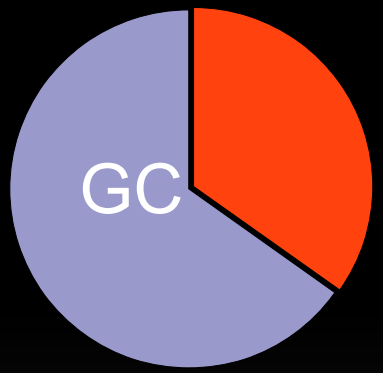
Arrays of open sized arrays

```
struct foo{
  …
  char rest[0];
};
```

Benchmark: CPStack

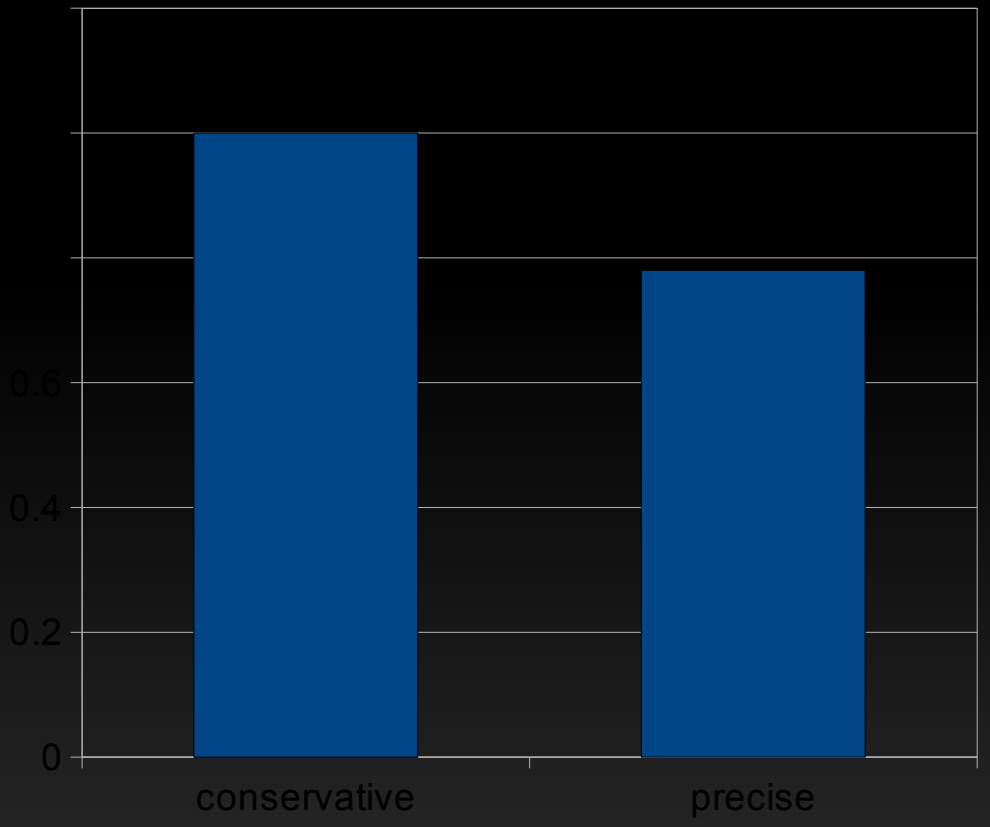


Conservative

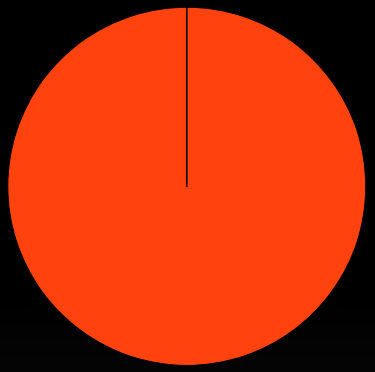


Precise

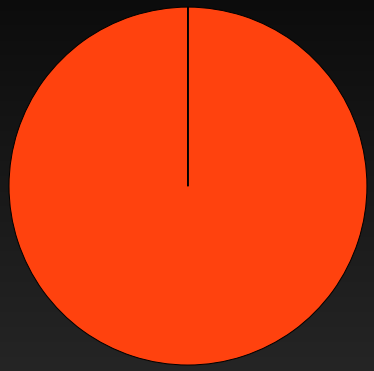
Benchmark: TakI

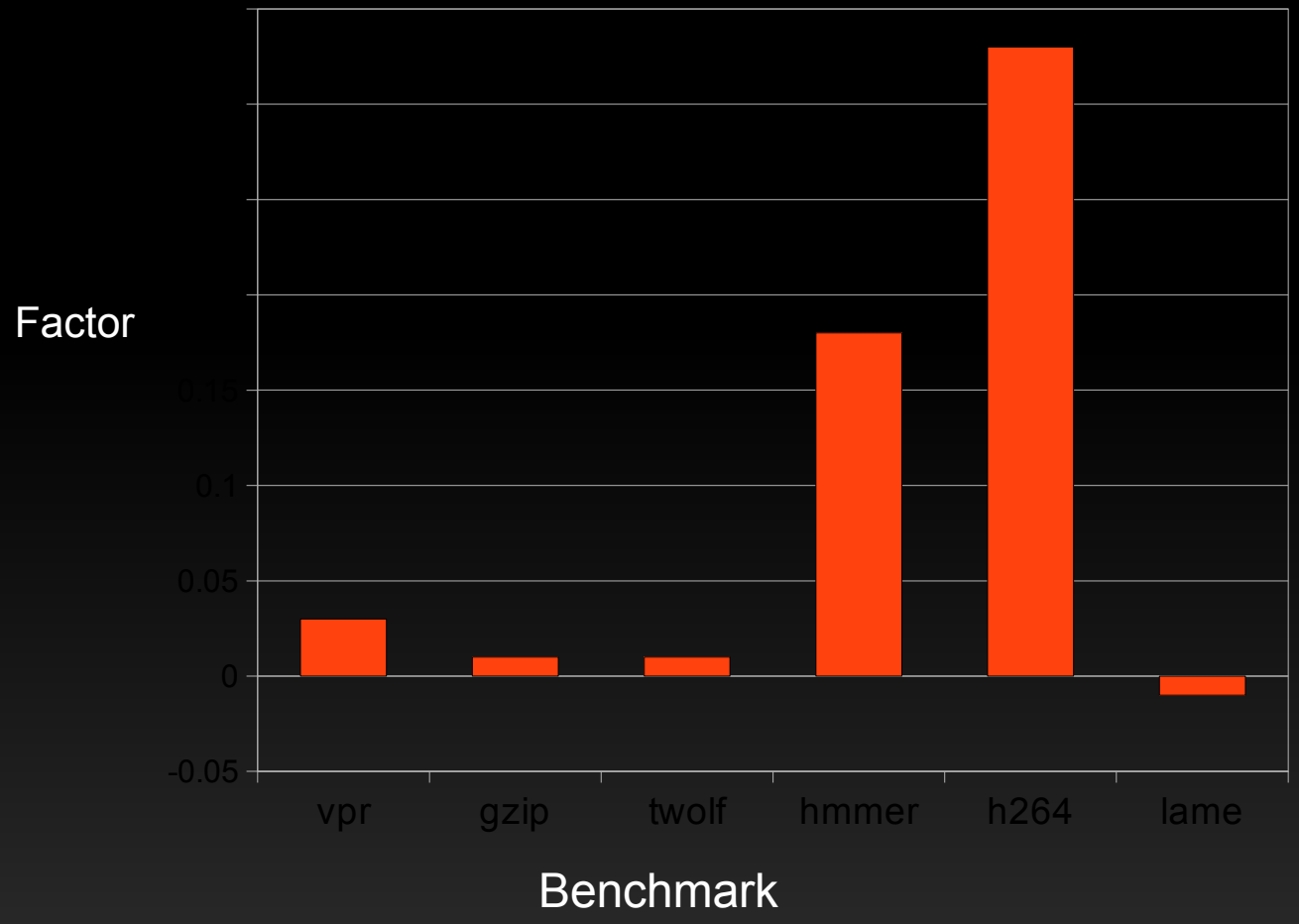


Conservative

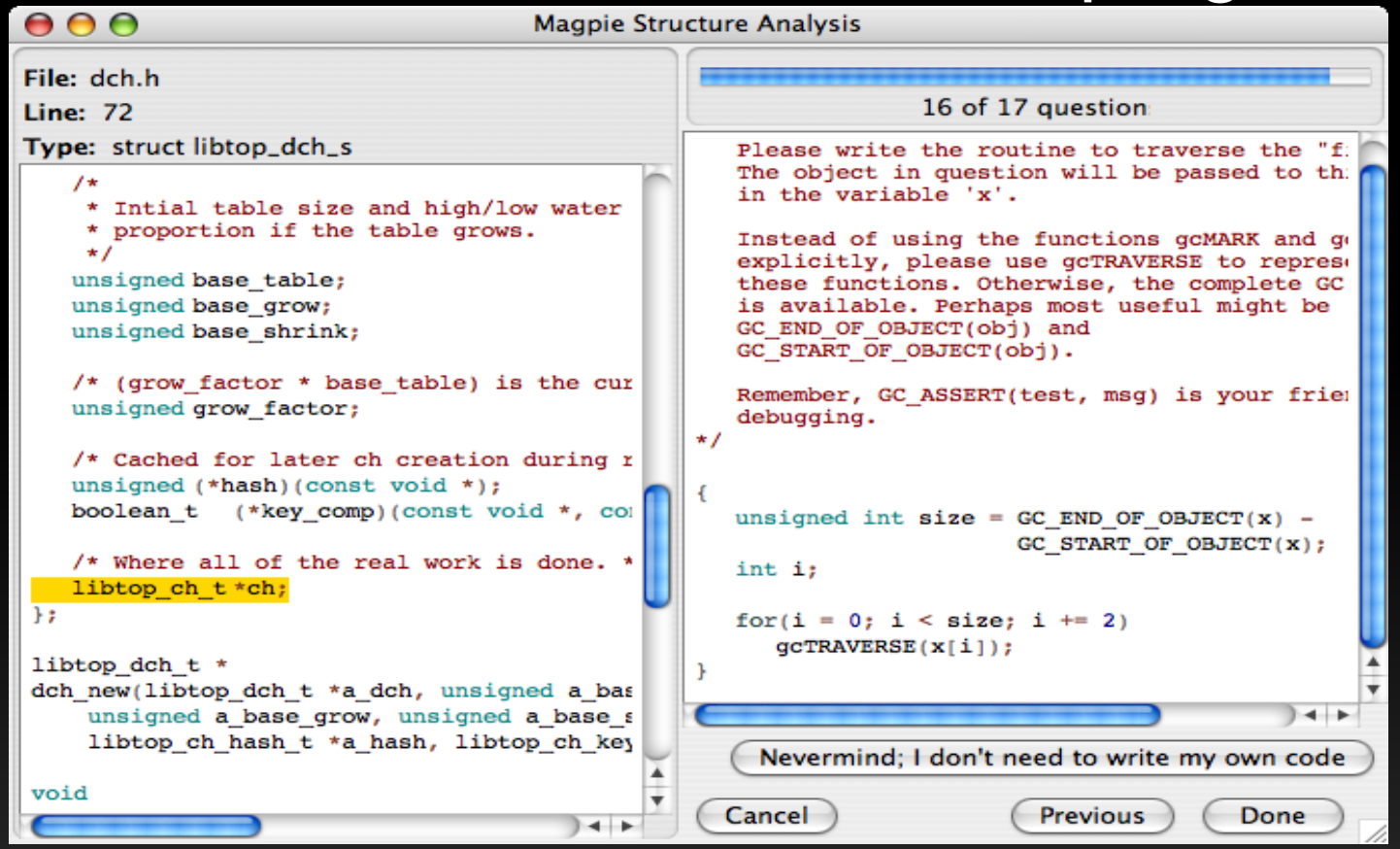


precise




Factor
0.15
0.1
0.05
0
-0.05
vpr
gzip
twolf
hmmer
h264
lame
Benchmark

Two tools that can be used on C programs

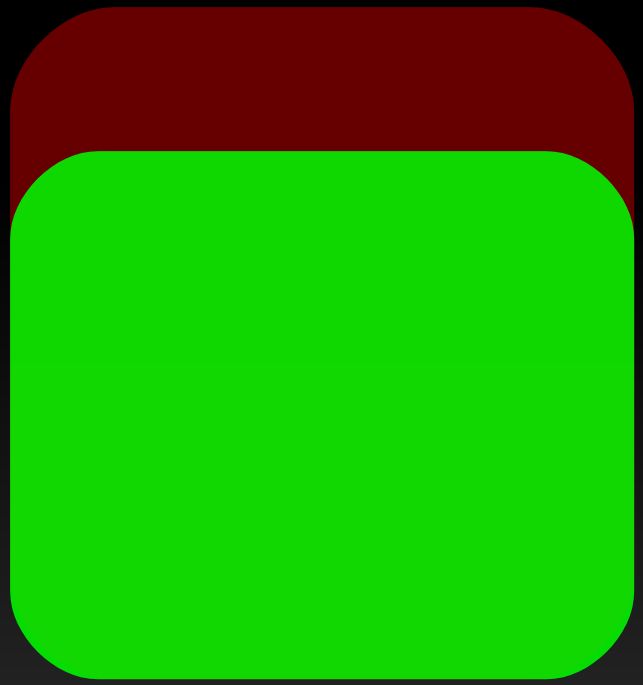


Two tools that can be used on C programs

- Based on Cil (Necula 2002)
- Parses C and GCC extensions
- Fully automatic

Too large to fully transform
- Subsystems:
  - Ext3
  - IPV4

User Mode Linux stack is limited to 4k – GC shadow stack went over this limit



Shadow stack

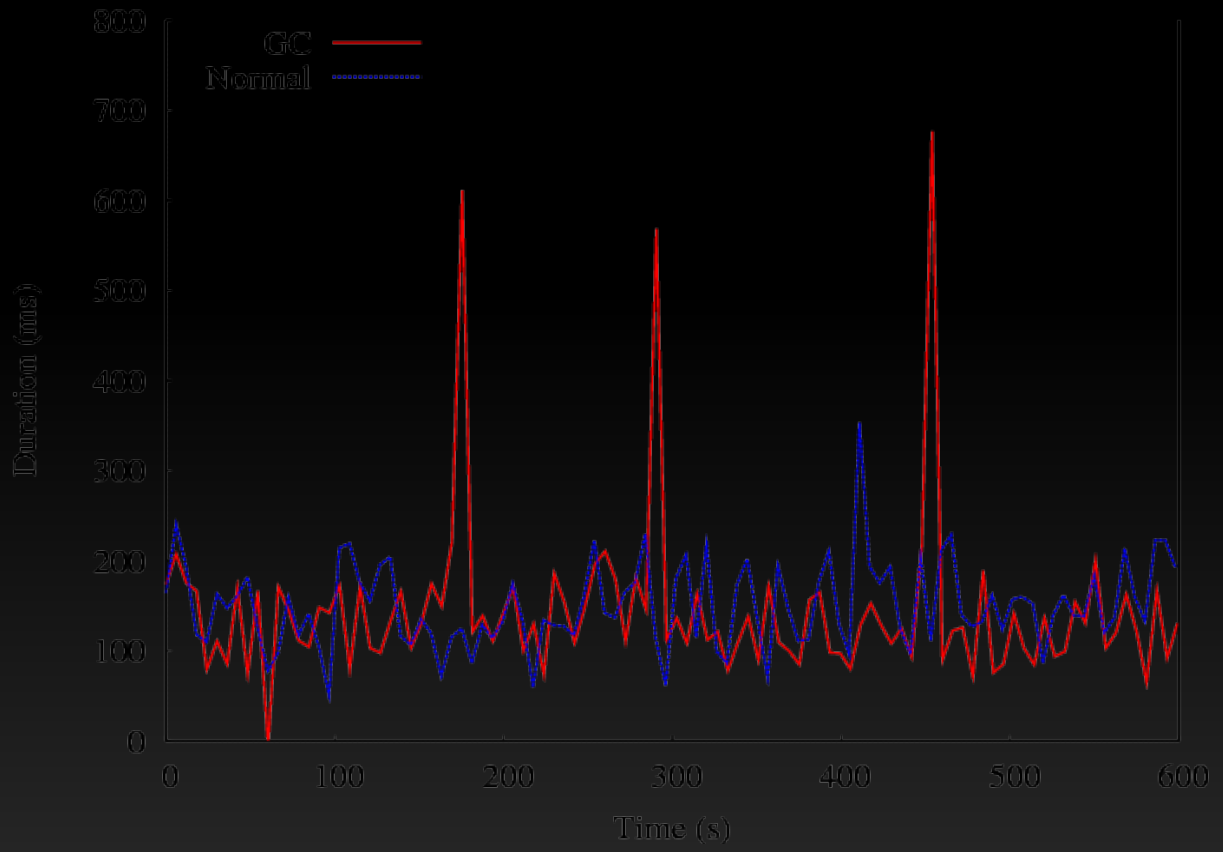
Manual changes

RCU (Read-Copy-Update) finalizers

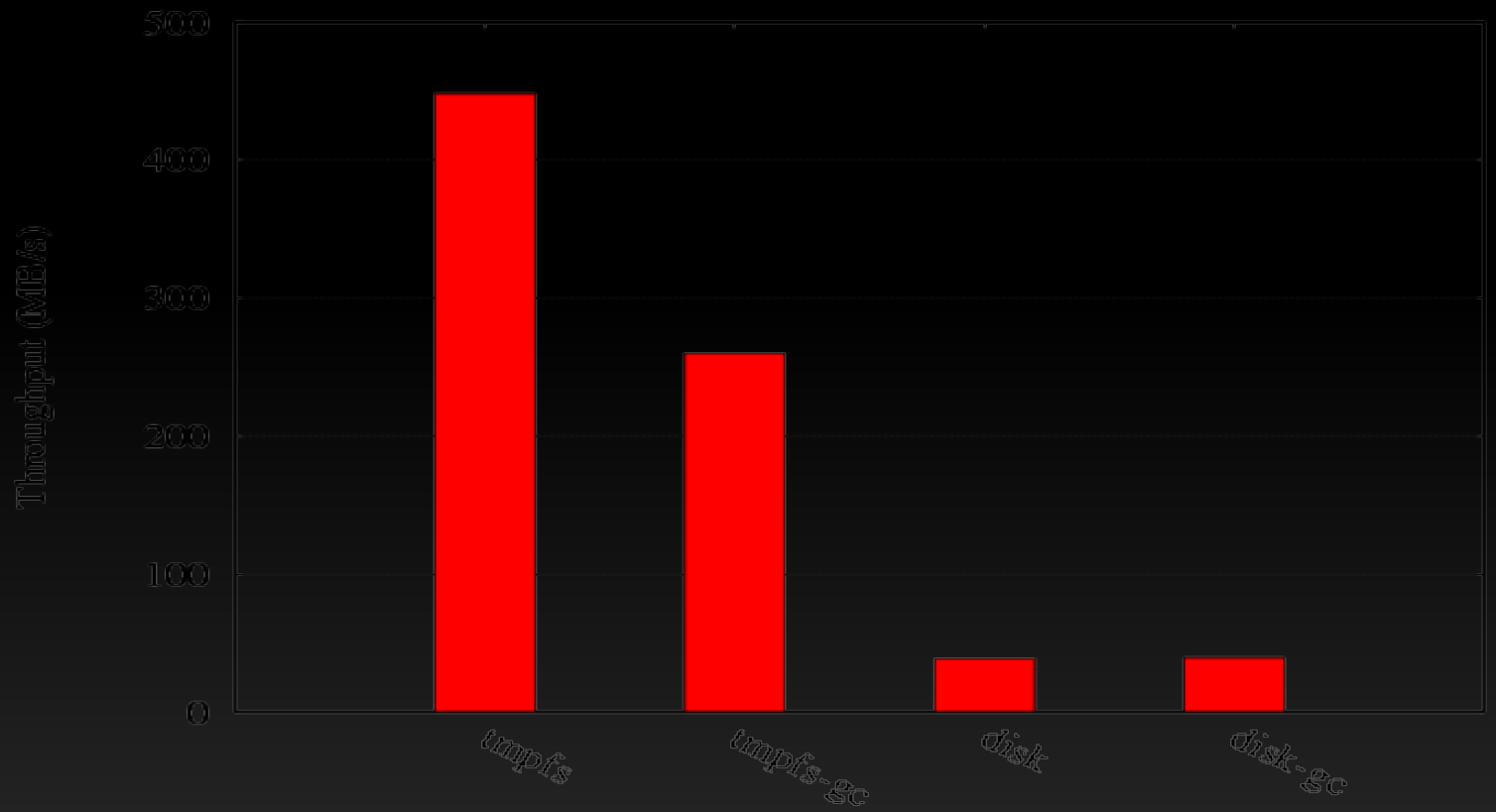
Important structures allocated with vmalloc

Custom allocators (kmem_cache_alloc)

GC can cause high latencies



GC overhead


Throughput (MB/s)
500
400
300
200
100
0
tmpfs
tmpfs-gc
disk
disk-gc

- Precise garbage collection is a practical alternative for C
- Long running programs benefit from precise garbage collection

CPStack

Takl