Memory Management with Explicit Regions [1]

David Gay and Alex Aiken
EECS Department
University of California, Berkeley

presented by:
Alexander Baumgartner

Department of Computer Science
University of Salzburg

January 20, 2011

Table of contents

State of the art 1998

- Douglas T. Ross 1967; The AED free storage package [2]
  $\Rightarrow$ Available space is partitioned into storage zones
- Kiem-Phong Vo 1996; Vmalloc: A General and Efficient Memory Allocator [3]
  $\Rightarrow$ Allows different allocation strategies (region- and/or objectbased)
- ...and a lot of more

What was new?

- Safe, region-based memory management
- Comparing performance with standard malloc/free implementations

Explicit regions

Region r = newregion();

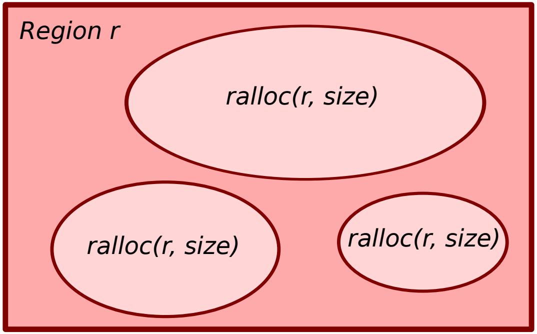


Figure: Allocation inside a region

deleteregion(r);

Safe regions - implementation overview

- Extended C ⇒ C@
- Normal *pointers vs. region @pointers
- Reference count per region
- Deleteregion checks reference count before freeing

Region pointers

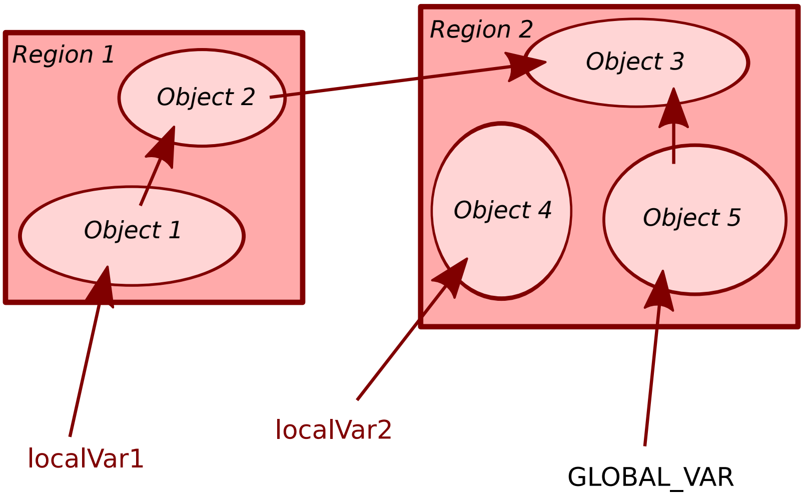


Figure: Different types of region pointers

Change region pointers

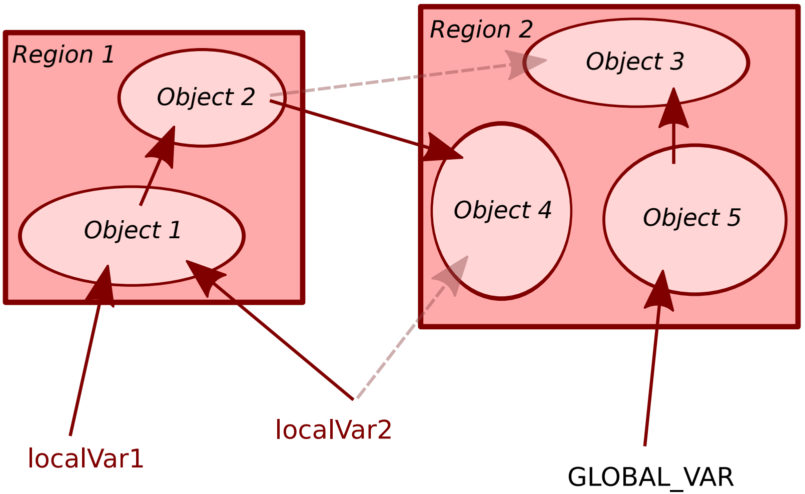


Figure: Changing region pointers

Change region pointers - Implication

- Decrement old regions rc
- Increment new regions rc
- We need to know the region of a region pointer
- $\Rightarrow$ How to provide a regionof function?
- $\Rightarrow$ DETAIL 1: Managing regions

Deleting a region

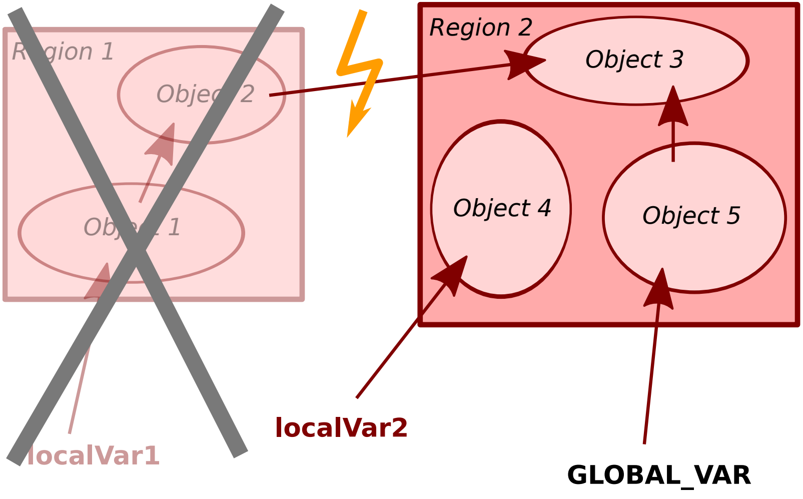


Figure: Deleting a region – deleteregion(Region1);

Deleting a region - Implication

- Region pointers may point to objects inside an other region
- Decrement other regions rc
- We need all region pointers inside the regions space
- ⇒ How to find all region pointers?
- ⇒ DETAIL 2: Region Scan

Performance of local region pointers

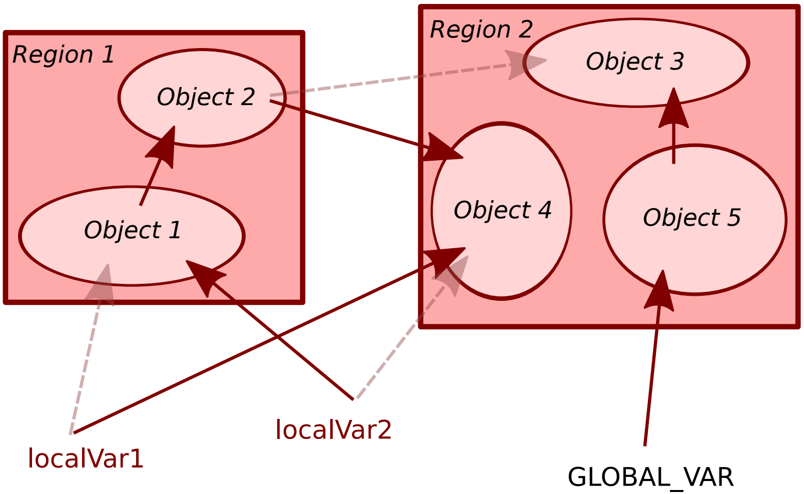


Figure: Changing region pointers

Performance of local region pointers - Implication

- Exchanging region pointers as shown in previous figure:

```
void @tmp = localVar1; // Region1.rc++
localVar1 = localVar2; // Region2.rc++ AND Region1.rc--
localVar2 = tmp;       // Region1.rc++ AND Region2.rc--
tmp = null;            // Region1.rc--
```

- We need regionof operations for identifying the pointer's region and increment/decrement the reference counts.
- A lot of work is done for nothing
- ⇒ How to get good performance?
- ⇒ DETAIL 3: Local variables

Implementation details

- DETAIL 1: Managing regions

  motivated by: We need to know the region of a region pointer

- DETAIL 2: Region scan

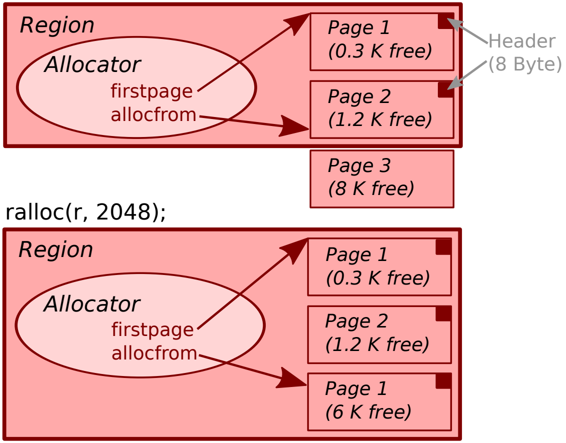
  motivated by: We need all region pointers inside the regions space

- DETAIL 3: Local variables

  motivated by: A lot of work is done for nothing

Managing regions by blocks

- Allocating blocks (=page)
- Page belongs to one region and contains header infos



Implementation details

- DETAIL 1: Managing regions

  motivated by: We need to know the region of a region pointer

- DETAIL 2: Region scan

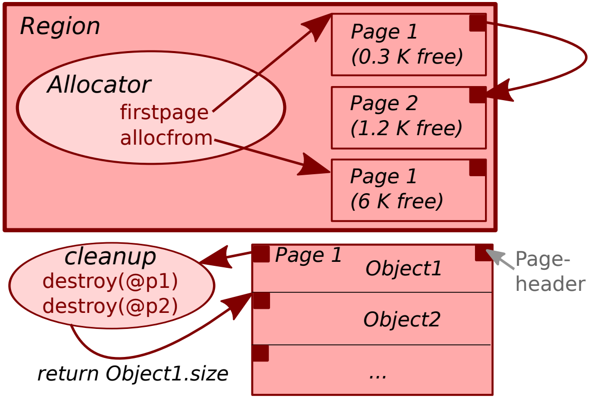
  motivated by: We need all region pointers inside the regions space

- DETAIL 3: Local variables

  motivated by: A lot of work is done for nothing

Region scan

- Objects containing region pointers have to offer a cleanup function.
- ralloc(region, size, cleanup) // cleanup is stored in front of object
- destroy(@pointer) // decrements region.rc if necessary
- cleanup has to return the objects size



Implementation details

- DETAIL 1: Managing regions

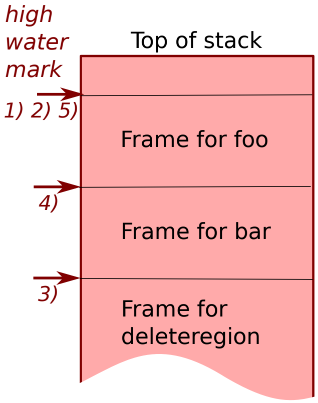
  motivated by: We need to know the region of a region pointer
- DETAIL 2: Region scan

  motivated by: We need all region pointers inside the regions space
- DETAIL 3: Local variables

  motivated by: A lot of work is done for nothing

Local variables - high water mark

- Only `deleteregion` needs exact rc
- `High water mark` is always above call frame
- `Deleteregion` performs stack-scan and sets `high water mark`
- ⇒ Writes to local variables NEVER updates rc



```
 void foo(struct baz @x) {
   ... doSomeStuff ...
1)
   bar(x);
5)
   return;
 }
 void bar(struct baz @x) {
   Region tmp = newregion();
   ... doSomeOtherStuff ...
2)
3) deleteregion(tmp);
4)
   return;
 }
```

Results

- Compared themselfes with 3 different malloc/free implementations
- 6 allocation-intensive programs (cfrac,gröbner,mudlle,lcc,tile,moss)
- Unsave regions are never slower and up to 16% faster
- Save regions are from 5% slower to 9% faster
- ?Save? regions use from 9% less to 19% more memory than Doug Lea

The End

Thank You!

[1] David Gay and Alex Aiken. Memory management with explicit regions. *PLDI '98 Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation*, pages 313–323, .

[2] Douglas T. Ross. The aed free storage package. *Communications of the ACM, 10(8) 1967*, pages 481–492.

[3] Kiem-Phong Vo. Vmalloc: A general and efficient memory allocator. *Softwarepractice and Experience, 26(3) 1996, 10(8) 1967*, pages 357–374.

[4] David Gay and Alex Aiken. Language support for regions. *PLDI '01 Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 70–80, .