# Dynamic Memory Allocation II
# April 1, 2004

Topics

- Explicit doubly-linked free lists
- Segregated free lists
- Garbage collection
- Memory-related perils and pitfalls

# Freeing With Explicit Free Lists

*Insertion policy*: Where in the free list do you put a newly freed block?

- LIFO (last-in-first-out) policy
  - Insert freed block at the beginning of the free list
  - Pro: simple and constant time
  - Con: studies suggest fragmentation is worse than address ordered.
- Address-ordered policy
  - Insert freed blocks so that free list blocks are always in address order
    - I.e. addr(pred) < addr(curr) < addr(succ)
  - Con: requires search
  - Pro: studies suggest fragmentation is better than LIFO



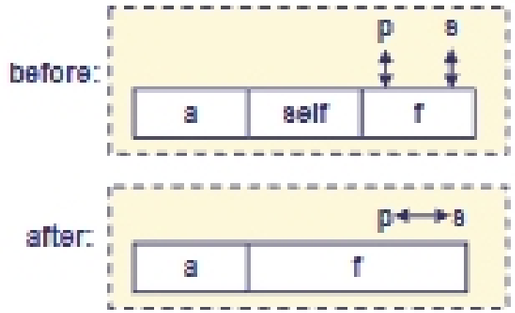
# Freeing With a LIFO Policy

Case 1: a-a-a

- Insert self at beginning of free list


Case 2: a-a-f

- Splice out next, coalesce self and next, and add to beginning of free list

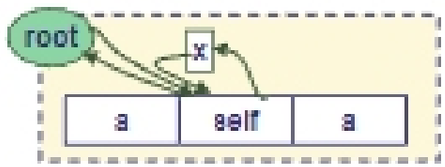





# Freeing With a LIFO Policy

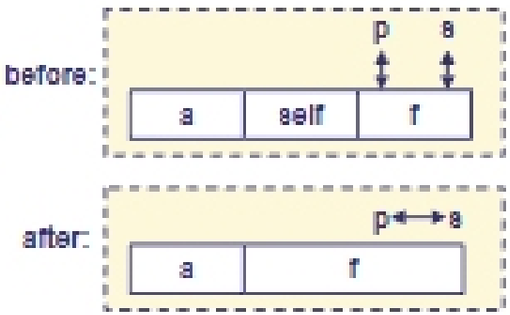
Case 1: a-a-a

- Insert self at beginning of free list




Case 2: a-a-f

- Splice out next, coalesce self and next, and add to beginning of free list

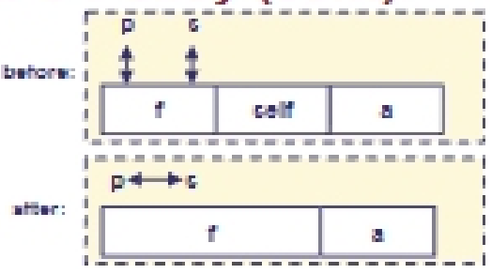





# Freeing With a LIFO Policy (cont)

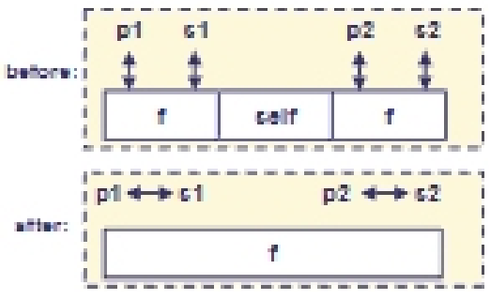
Case 3: f-a-a

- Splice out prev, coalesce with self, and add to beginning of free list




Case 4: f-a-f

- Splice out prev and next, coalesce with self, and add to beginning of list



# Explicit List Summary

Comparison to implicit list:

- Allocate is linear time in number of free blocks instead of total blocks -- much faster allocates when most of the memory is full
- Slightly more complicated allocate and free since needs to splice blocks in and out of the list
- Some extra space for the links (2 extra words needed for each block) Does this increase internal frag?

Main use of linked lists is in conjunction with segregated free lists

- Keep multiple linked lists of different size classes, or possibly for different types of objects



# Keeping Track of Free Blocks

*Method 1*: *Implicit list* using lengths -- links all blocks

*Method 2*: *Explicit list* among the free blocks using pointers within the free blocks

*Method 3*: *Segregated free list*

- Different free lists for different size classes

*Method 4*: Blocks sorted by size

- Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key