CSE 31
Computer Organization

Lecture 10 – C Memory Management (wrap up)
and Integer Representation

Announcements

- Labs
  - Lab 3 grace period ends this week
  - Lab 4 due this week (with 7 days grace period after due date)
    - » Demo is REQUIRED to receive full credit
  - Lab 5 out ***next*** week
    - » Due at 11:59pm on the same day of your lab after next (with 7 days grace period after due date)
    - » You must demo your submission to your TA within 14 days from posting of lab
    - » Demo is REQUIRED to receive full credit
- Reading assignments
  - Reading 02 (zyBooks 2.1 – 2.9) due tonight, 22-FEB and Reading 03 (zyBooks 3.1 – 3.7, 3.9) due 06-MAR
    - » Complete **Participation Activities** in each section to receive grade
    - » IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses
- Homework assignment
  - Homework 01 (zyBooks 1.1 – 1.5) due tonight, 22-FEB and Homework 02 (zyBooks 2.1 – 2.9) due 27-FEB
    - » Complete **Challenge Activities** in each section to receive grade
    - » IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcements

- Project 01
  - Due 17-MAR
  - Can work in teams of 2 students
    - » Each team member must identify teammate in “Comments…” text-box at the submission page
    - » If working in teams, each student must submit code (can be the same as teammate) and demo individually
    - » Grade can vary among teammates depending on demo
  - Demo required for project grade
    - » No partial credit for submission without demo
  - No grace period
    - » Must complete submission and demo by due date.

Automatic Memory Management

- Dynamically allocated memory is difficult to track
  - Why not track it automatically?

- If we can keep track of what memory is in use, we can reclaim everything else.
  - Unreachable memory is called garbage, the process of reclaiming it is called garbage collection.

- So how do we track what is in use?
  - Start with all pointers in global variables and local variables (root set).
  - Recursively examine dynamically allocated objects we see a pointer to.
  - We can do this in constant space by reversing the pointers on the way down

- We will cover 3 schemes to collect garbage

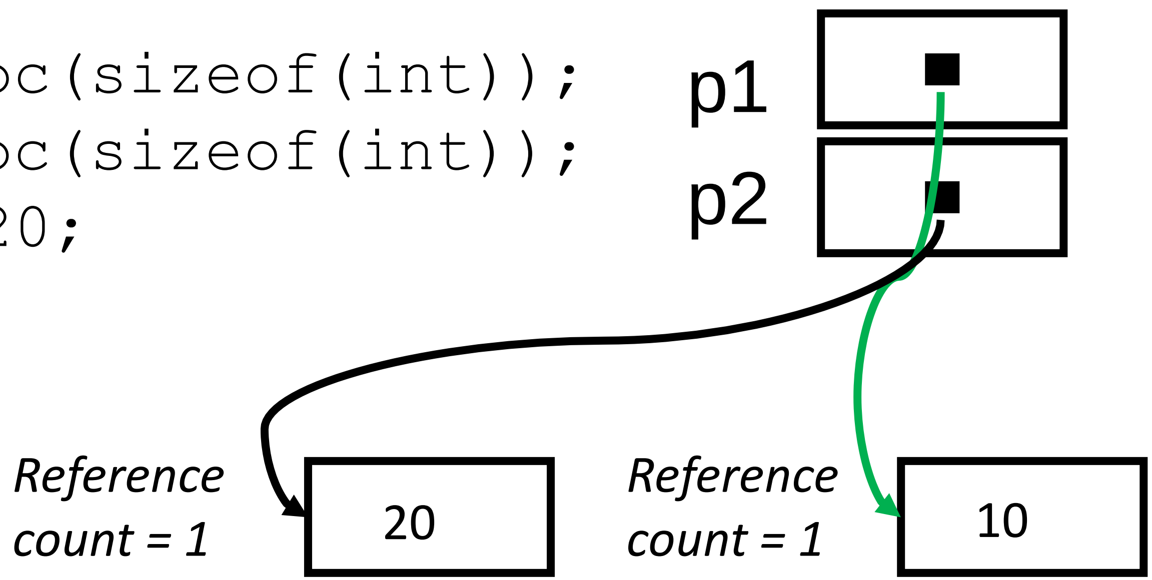
Scheme 1: Reference Counting

- For every chunk of dynamically allocated memory, keep a count of number of pointers that point to it.
- When the count reaches 0, reclaim the memory.
- Simple assignment statements can result in a lot of work, since may update reference counts of many items

Reference Counting Example

- For every chunk of dynamically allocated memory, keep a count of number of pointers that point to it.
  - When the count reaches 0, reclaim.

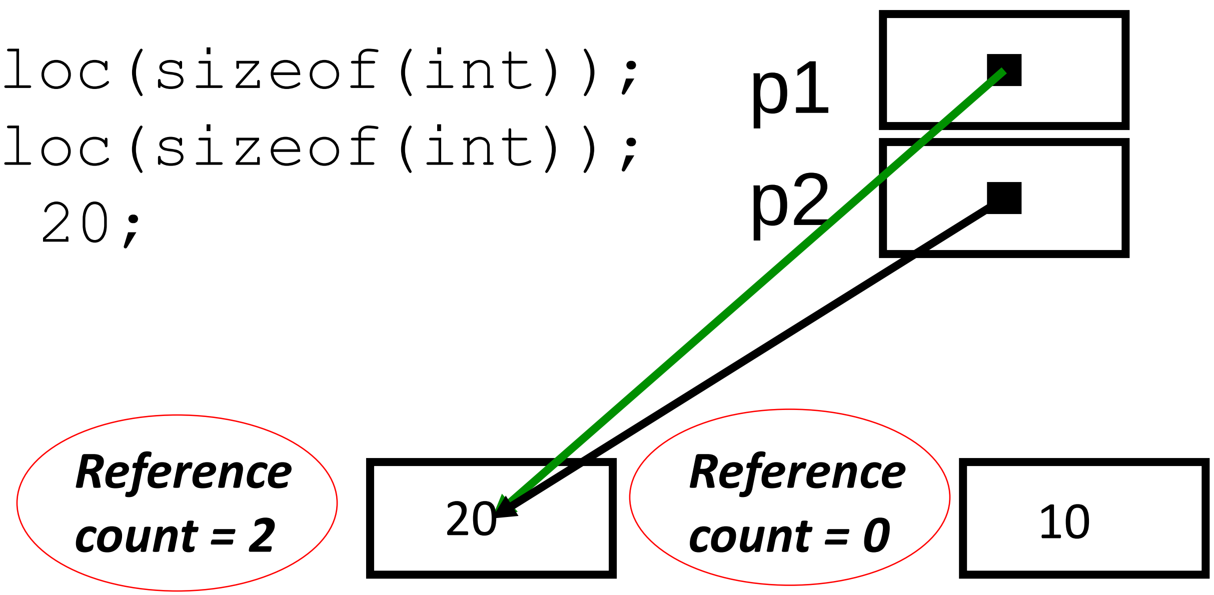
```
int *p1, *p2;
p1 = (int *)malloc(sizeof(int));
p2 = (int *)malloc(sizeof(int));
*p1 = 10; *p2 = 20;
```



Reference Counting Example

- For every chunk of dynamically allocated memory, keep a count of number of pointers that point to it.
  - When the count reaches 0, reclaim.

```
int *p1, *p2;
p1 = (int *)malloc(sizeof(int));
p2 = (int *)malloc(sizeof(int));
*p1 = 10; *p2 = 20;
p1 = p2;
```



Reference Counting (`p1`, `p2` are pointers)

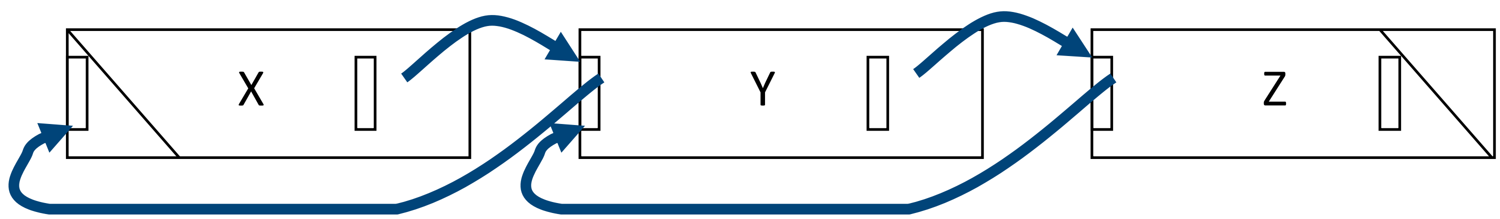
```
p1 = p2;
```

- Increment reference count for `p2`
- If `p1` held a valid value, decrement its reference count
- If the reference count for `p1` is now 0, reclaim the storage it points to.
  - If the storage pointed to by `p1` was pointing to other pointers, decrement all their reference counts, and so on…
- Must also decrement reference count when local variables cease to exist.

Reference Counting Flaws

- Extra overhead added to assignments, as well as ending a block of code.

- Does not work for circular structures!
  - E.g., doubly linked list:



Scheme 2: Mark and Sweep Garbage Collection

- Keep allocating new memory until memory is exhausted, then try to find unreachable memory.

- Consider objects in a graph, chunks of memory (objects) are graph nodes, pointers to memory are graph edges.
  - Edge from A to B $\rightarrow$ A stores pointer to B

- Can start with the root set, perform a graph traversal, find all reachable memory!

- 2 Phases:
  1. Mark used nodes
  2. Sweep free ones, returning list of free nodes

Mark and Sweep

- Graph traversal is relatively easy to implement recursively

```
void traverse(struct graph_node *node) {
    /* visit this node */
    foreach child in node->children {
        traverse(child);
    }
}
```

- But with recursion, state is stored on the execution stack.
  - Garbage collection is invoked when not much memory left
  - …Oops!

- As before, we could traverse in constant space (by reversing pointers)

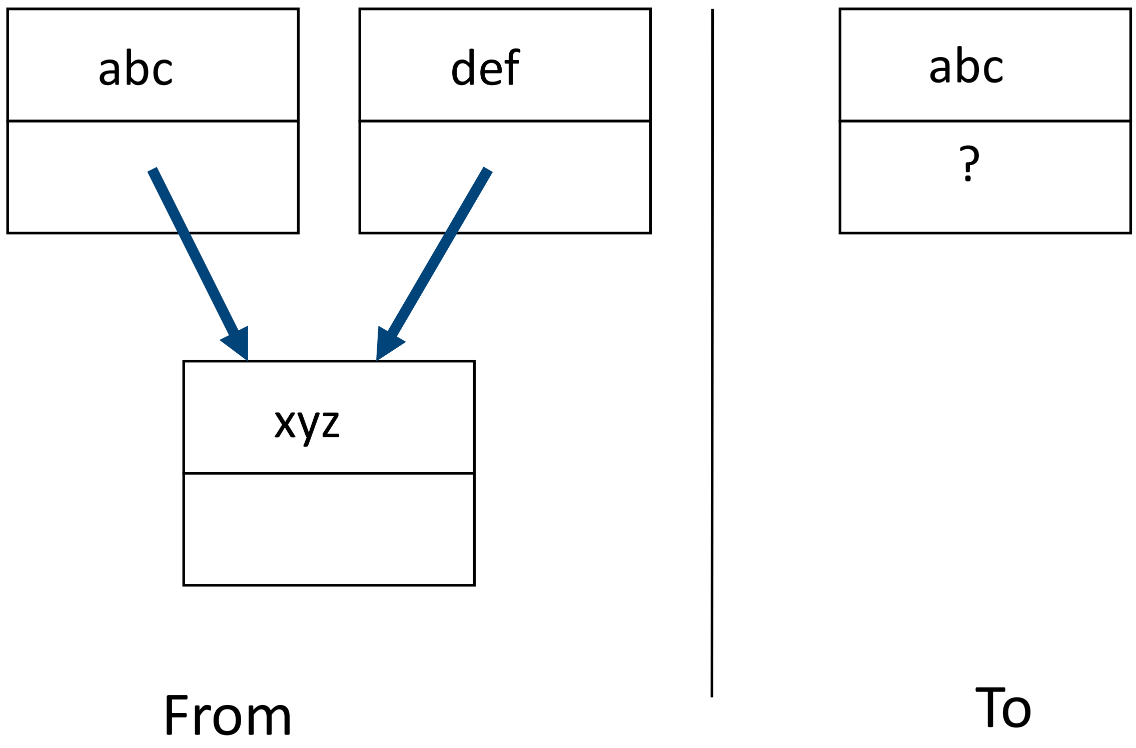
Scheme 3: Copying Garbage Collection

- Divide memory into two spaces, only one in use at any time.

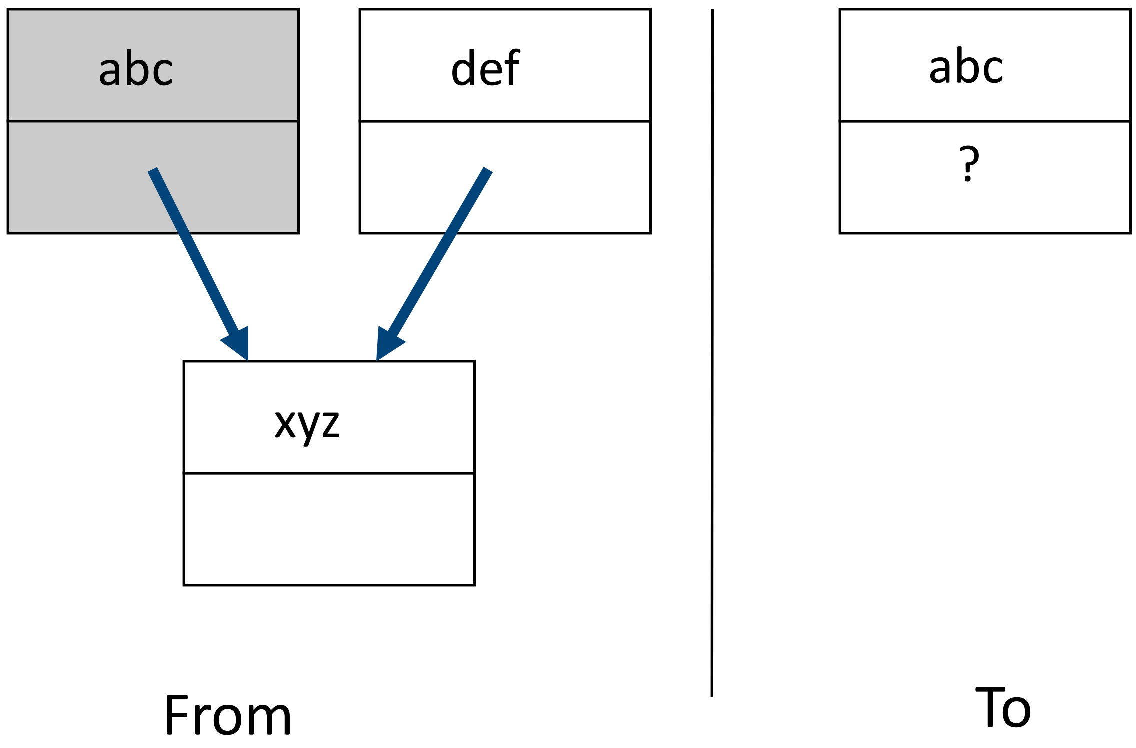
- When active space is exhausted, traverse the active space, copying all objects to the other space, then make the new space active and continue.
  - Only reachable objects are copied!

- Use “forwarding pointers” to keep consistency
  - Simple solution to avoiding having to have a table of old and new addresses, and to mark objects already copied

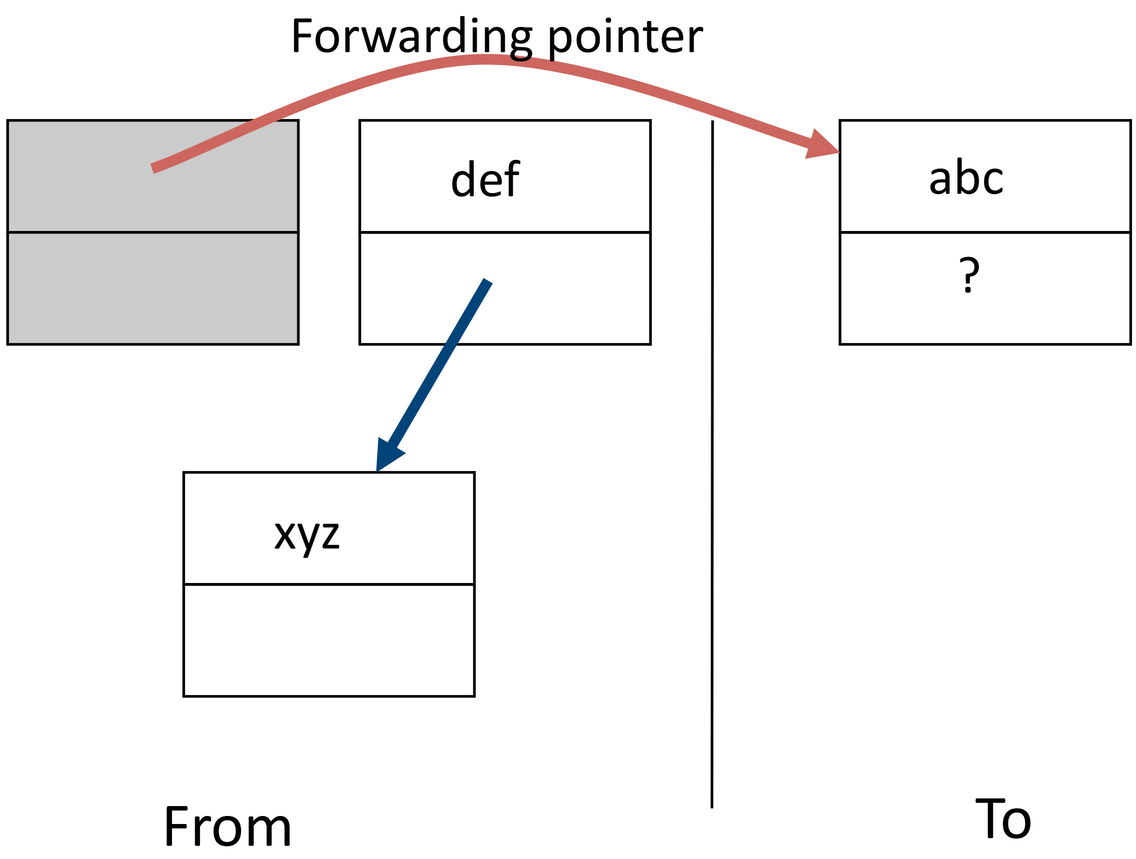
Forwarding Pointers: 1st copy "abc"



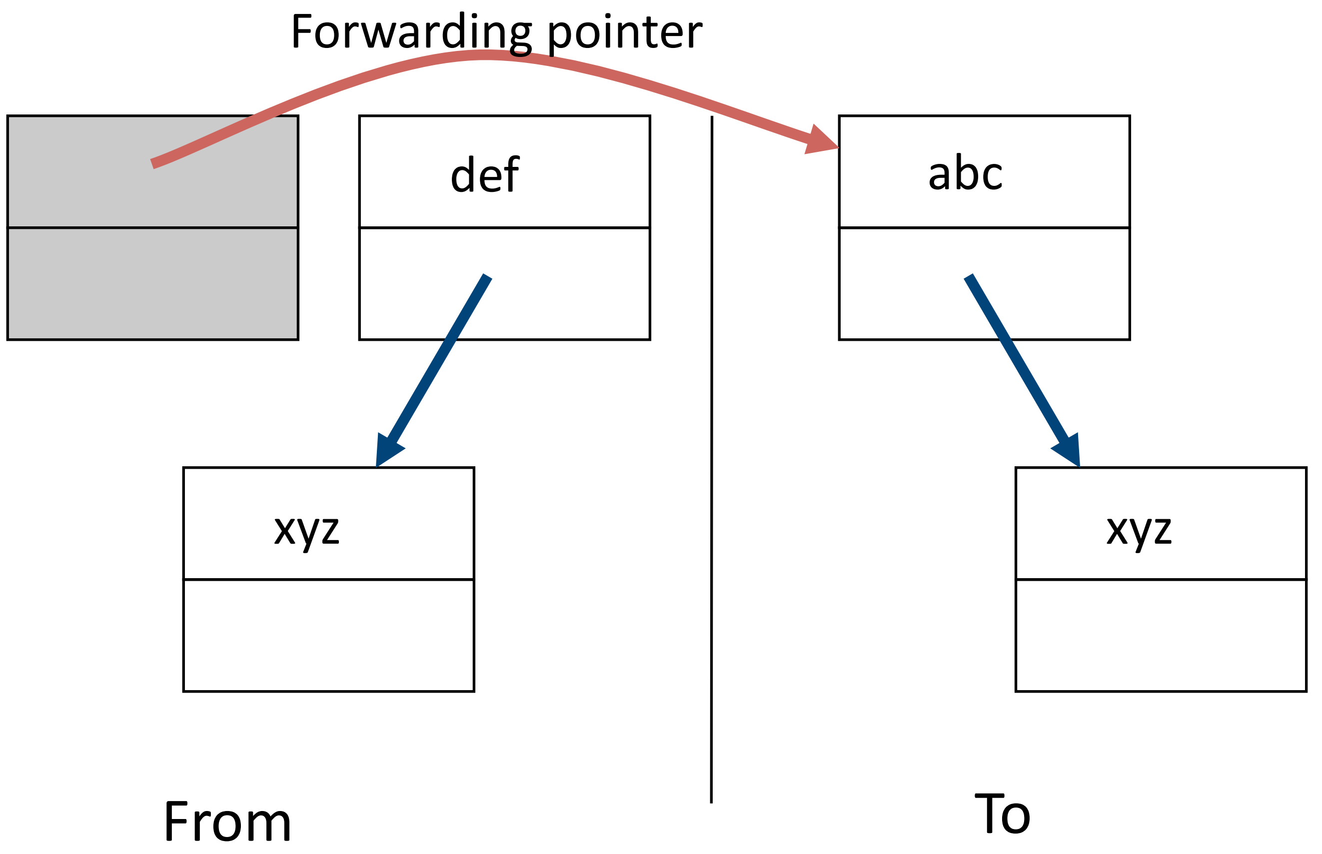
Forwarding Pointers: leave ptr to new abc



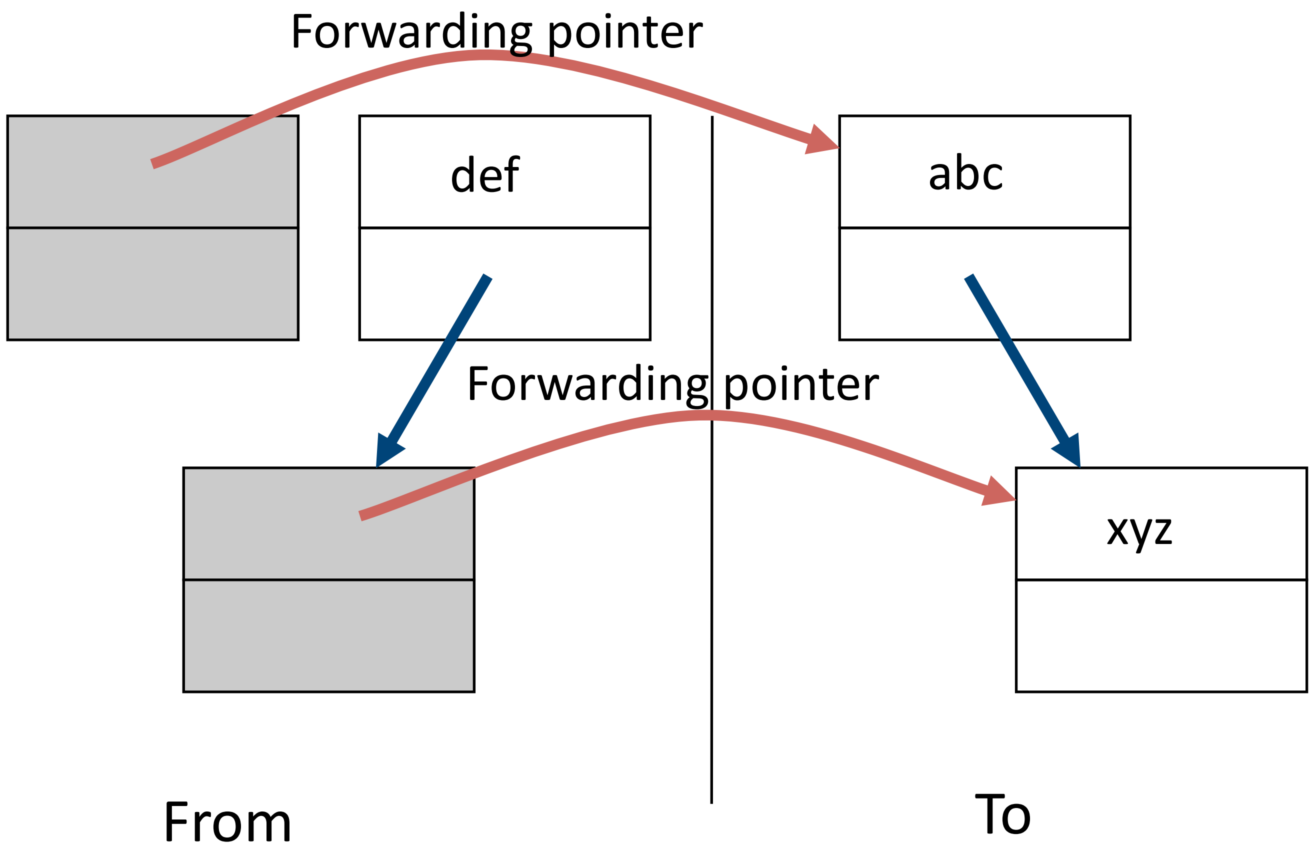
Forwarding Pointers : now copy “xyz”



Forwarding Pointers: leave ptr to new xyz

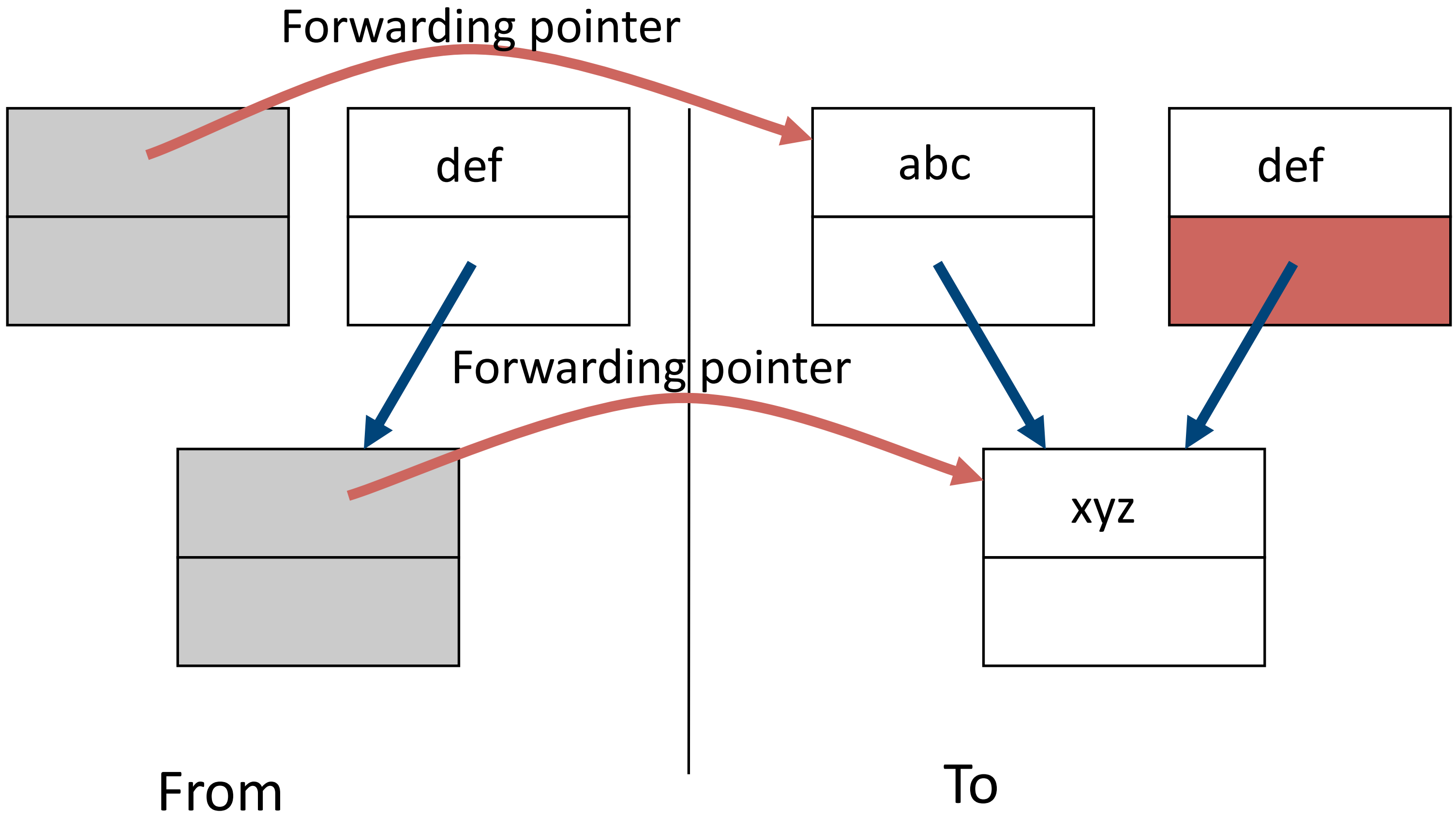


Forwarding Pointers: now copy "def"



Since xyz was already copied,
def uses xyz's forwarding pointer
to find its new location

Forwarding Pointers



Since xyz was already copied,
def uses xyz's forwarding pointer
to find its new location

Summary

- Several techniques for managing heap via malloc and free: best-, first-, next-fit
  - 2 types of memory fragmentation: internal & external; all suffer from some kind of frag.
  - Each technique has strengths and weaknesses, none is definitively best

- Automatic memory management relieves programmer from managing memory.
  - All require help from language and compiler
  - Reference Count: not for circular structures
  - Mark and Sweep: complicated and slow, works
  - Copying: Divides memory to copy good stuff

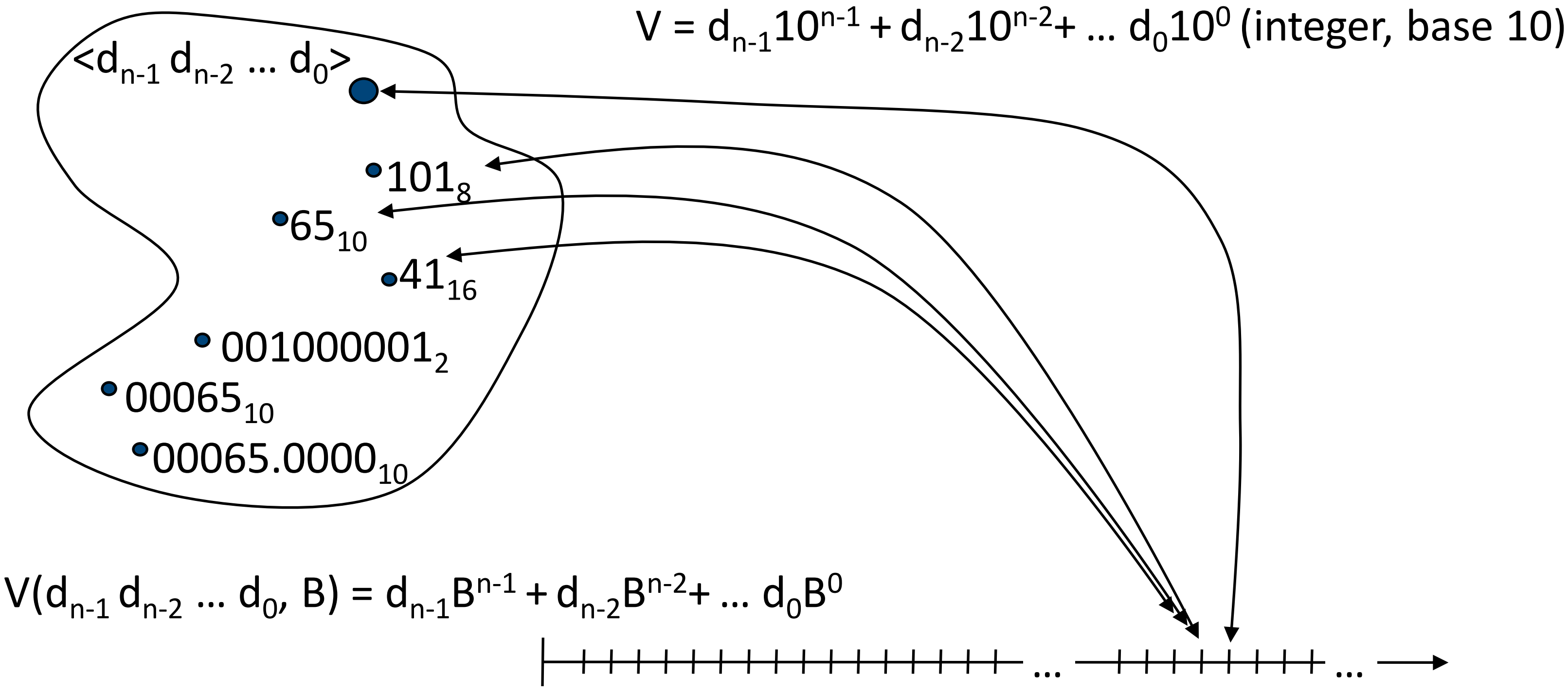
Number Representations

- What does this number mean?
  - 101

- Depends on what representation!

Representation and Meaning

- Objects are represented as collections of symbols (bits, digits)
- Their meaning is derived from what you do with them.



Representation (how many bits?)

- Characters?
  - 26 letters → 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation → 7 bits (in 8 bits) (“ASCII”)
  - standard code to cover all the world’s languages → 8,16,32 bits (“Unicode”) www.unicode.com

UNICODE

- Logical values?
  - 0 → False, 1 → True

- Color? Ex: *Red (00)* *Green (01)* *Blue (11)*

- Remember: N bits → at most 2^N things

How many bits to represent π ?

a) 1

b) 9 (π = 3.14, so that's 011 . 001 100)

c) 64 (Since modern computers are 64-bit machines)

d) Every bit the machine has!

e) ∞

We are going to learn how to represent floating point numbers later!

What to do with representations of numbers?

- Just what we do with numbers!
  - Add them
  - Subtract them
  - Multiply them
  - Divide them
  - Compare them

```
        1   1
        1   0   1   0
+       0   1   1   1
-----------------------
    1   0   0   0   1
```

- Example: 10 + 7 = 17
  - …so simple to add in binary that we can build circuits to do it!
  - subtraction just as you would in decimal
  - Comparison: How do you tell if X > Y ?