Garbage Collection

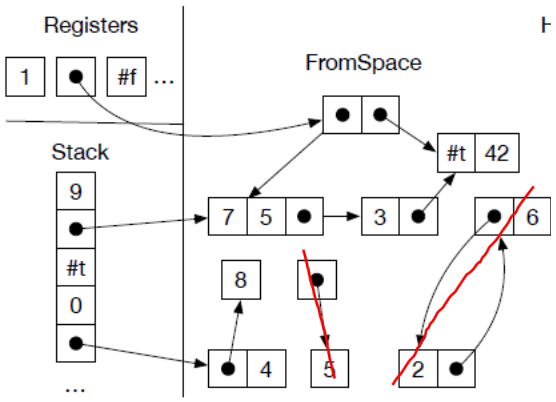
Garbage

No, not that stuff!

JOSEPH EID VIA GETTY IMAGES

Garbage

This is our kind of garbage:



Two-Space Copy Collector

Divide heap into two spaces: *FromSpace* and *ToSpace*

Initially all allocations go to the FromSpace

When there is not sufficient room for an allocation request: garbage collector goes to work.

What to keep?

Any data structure whose address is:

- in a register or
- on the procedure call stack

These addresses are called the *root set*.

Two-Space Copy Collector

Goals:

- Preserve all tuple that are reachable from the root set via a path of pointers, i.e. the *live* tuples, and
- Reclaim the memory of everything else, i.e. the *garbage*.

A copying collector accomplishes this by

1. Copying all of the live objects into the ToSpace and then
2. Treating the ToSpace as the new FromSpace and the old FromSpace as the new ToSpace.

Copy Collector Example

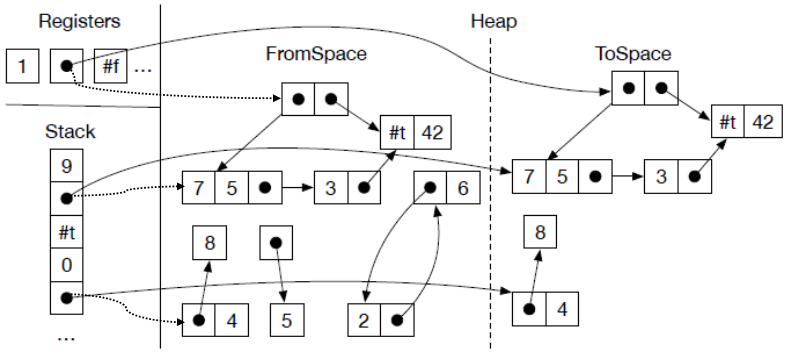


Figure 5.5: A copying collector in action.

Copy Collector

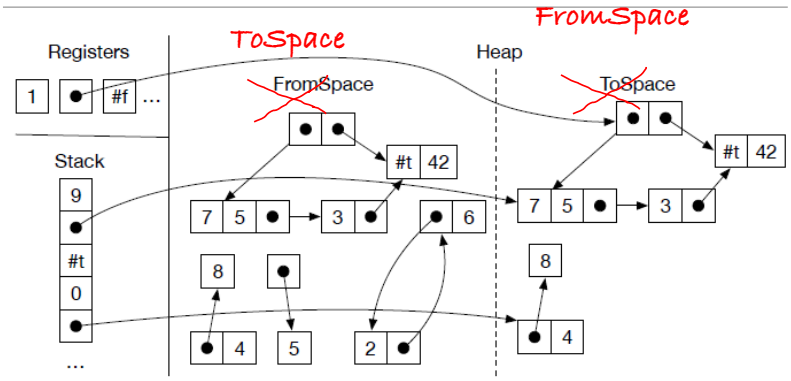
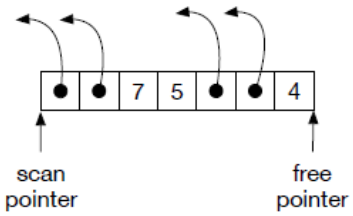
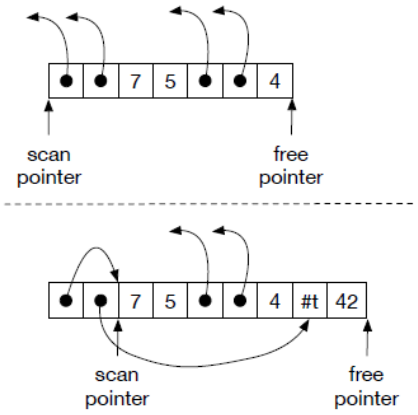


Figure 5.5: A copying collector in action.

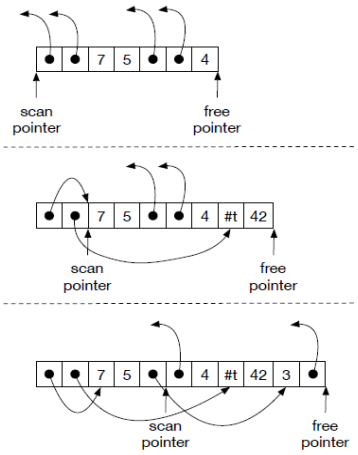
Copy Collector in Action
(via Cheney's Algorithm)



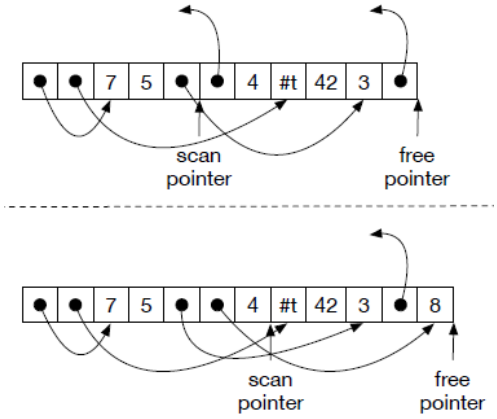
Copy Collector in Action
(via Cheney's Algorithm)



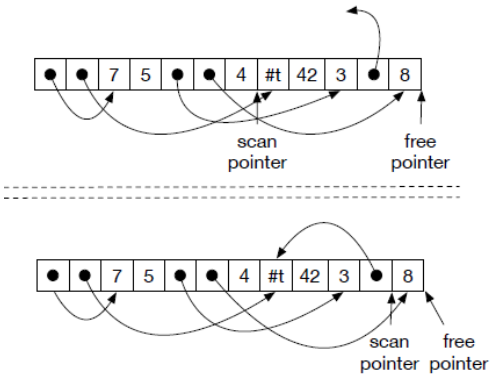
Copy Collector in Action
(via Cheney's Algorithm)



Copy Collector in Action
(via Cheney's Algorithm)



Copy Collector in Action
(via Cheney's Algorithm)



Data Representation

The garbage collector needs to distinguish between pointers and other kinds of data.

There are several ways to accomplish this.

1. Attached a tag to each object that identifies its type.
2. Store different types of objects in different regions.
3. Use type information from the program to either generate type-specific code for collecting or to generate tables that can guide the collector

Dynamically typed languages, such as Lisp, need to tag objects, so option 1 is a natural choice for those languages.

Option 3 is the best-performing choice for statically typed languages, but comes with a relatively high implementation complexity.