Garbage collection

The Basics

What is memory management?

- Programs contain
  - Objects
  - Data
  - Occupy memory
- Runtime system must allocate and reclaim memory for program in an efficient manner
  - Why is this important?
  - Why is this hard?
  - Why is this interesting?

Allocation and Reclamation

- Allocation
  - Objects dynamically allocated on HEAP
  - malloc(), new()
- Reclamation
  - Manual/Explicit
    - free()
    - delete()
  - Automated
    - Garbage collection (GC)

Explicit memory management challenges

- Consumes software development time
  - new → allocate storage for new object
  - delete → reclaim storage
- Dangling pointers (reclaim too soon)

```
Foo* p = new Foo();
Foo* q = p;
delete p;
p->DoSomething();
p = NULL;
q->ProcessFoo();
```

  - **Statically undecidable**
  - **Problem for developers**

Explicit memory management challenges

- Memory leak (never reclaim)

```
#include <stdlib.h>
void f(void){
     void* s;
     s = malloc(50);
     return;
}

int main(void){
     while (1) f();
     return 0;
}
```

Explicit memory management pluses

- Efficiency can be very high
- Puts the programmer in control

Automated memory management

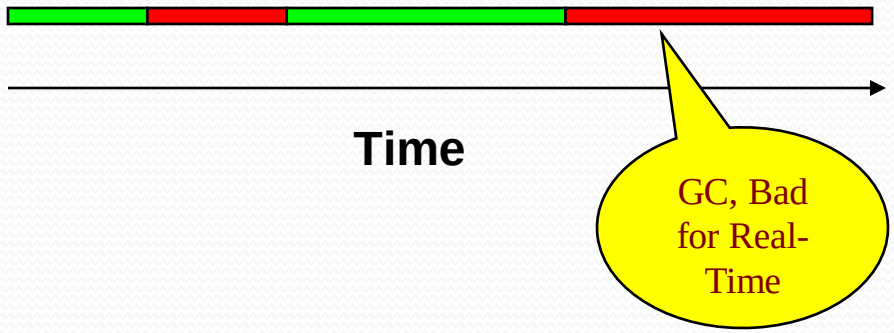
- Runtime system automatically
  - Detects dead objects (garbage detection)
  - Reclaims dead objects (garbage reclamation)
  - Garbage collection
- Preserves software development time
  - Relieves programmer burden
  - Less prone to errors
- Utilized by most modern OOP and scripting languages
  - Python, Java, C#, php

Garbage collection challenges

- Occurs an unpredictable times
- Duration is unbounded
- Performance efficiency issues

```
public void f(){
     startLaser();
     Obj o = new Obj();
     stopLaser();
}

public static void main(…){
     while (true) f();
}
```



Runtime system performs GC

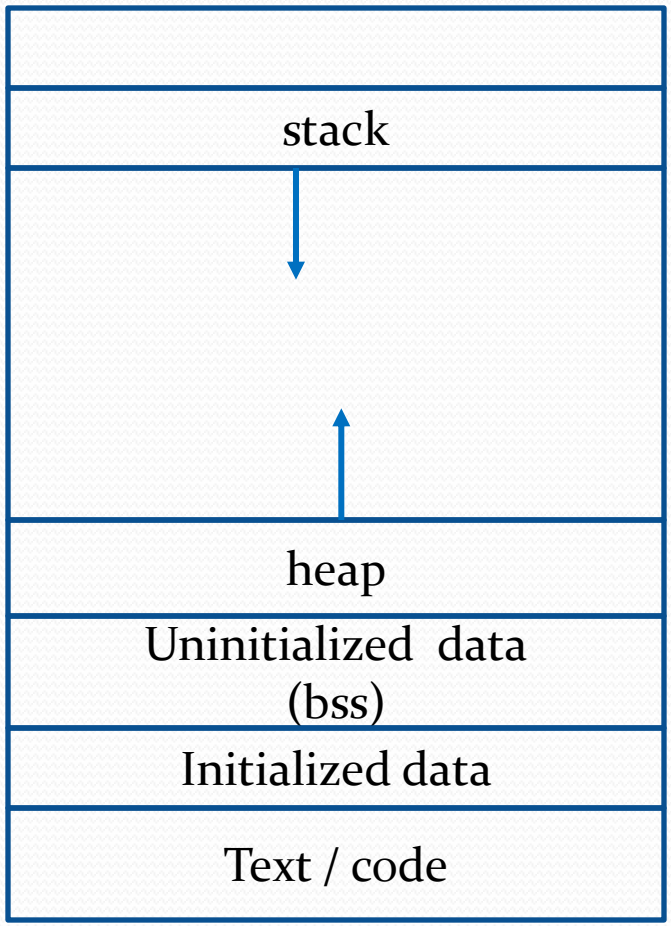
- E.g. Java virtual machine (JVM)
  - Software execution engine that executes your Java programs
  - Java interpreter that converts byte code into OS specific commands
  - Handles related tasks
    - Memory management (GC implemented in JVM)
    - Security
    - Multithreading

Major concerns

- Explicit memory management
  - Reclaiming objects at the right time
- Garbage collection
  - Discriminating live objects from garbage
- Both
  - Fast allocation
  - Fast reclamation
  - Low fragmentation

Layout of a program in memory

High address



Low address

Command line args and environment variables

Initialized to 0 by exec

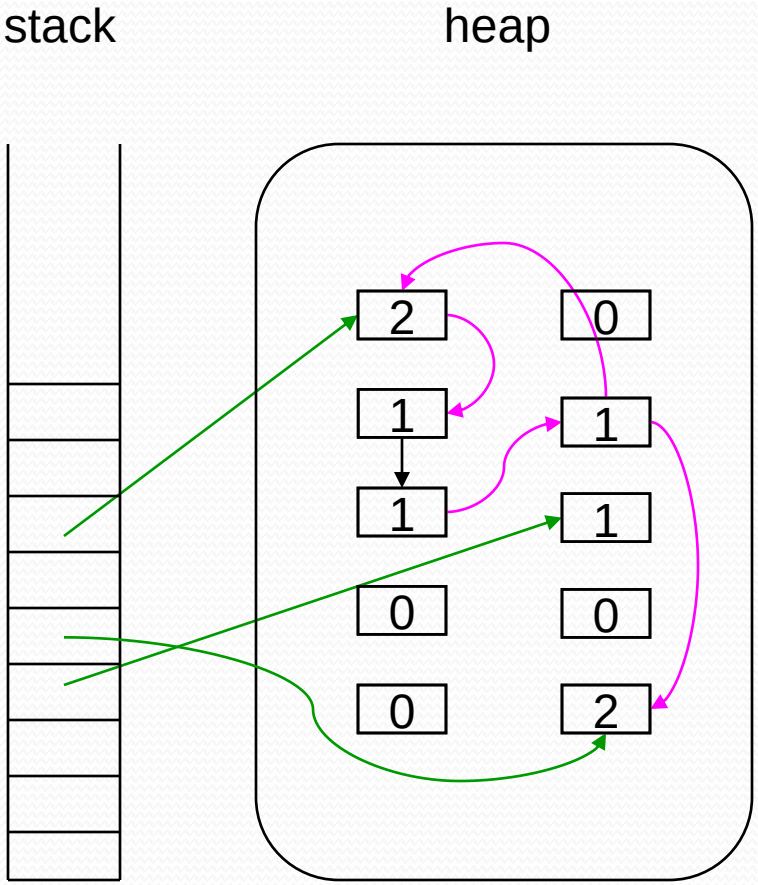
Read from program file by exec

Determining object liveness

- Live objects are needed in the computation
  - Now or in the future
- Prove that an object is not live (dead) and reclaim its storage
- Reclaim dead objects soon, after it is last used
- How do we estimate *liveness* in practice?
  - Approximate *liveness* by reachability from outside the heap
    - Unreachable objects are garbage (reclaim storage)
    - Reachable objects are live and must not be reclaimed

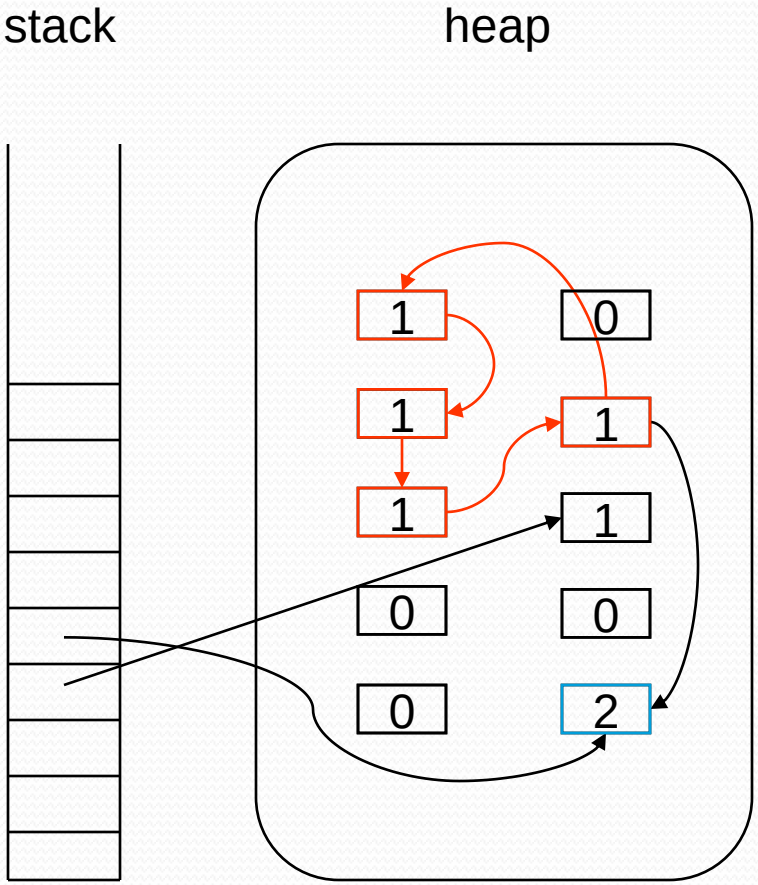
Identifying garbage

- **reference counting (reachability)**
- An integer is associated with every object, summing
  - Stack references
  - Heap references
- Objects with reference count of zero are dead



Problems with reference counting

- Standard problem is that objects in cycles (and those touched by such objects) cannot be collected (reclaimed)
- Overhead of counting can be high



Identifying garbage

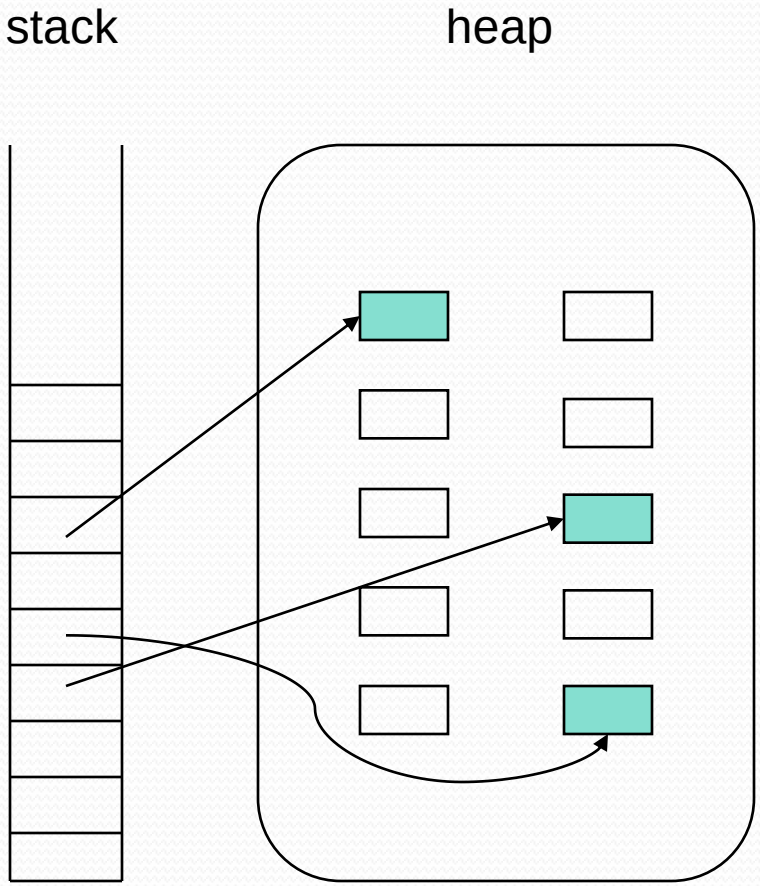
- **Tracing (reachability)**
- Trace reachability from root set
  - Processor registers
  - Program stack
  - Global variables
- Objects traced are reachable
- All other objects are unreachable (garbage)

The marking phase

- To find the dead objects, use the process of *calculatus eliminatus*
  - Find all live objects
  - All others are dead

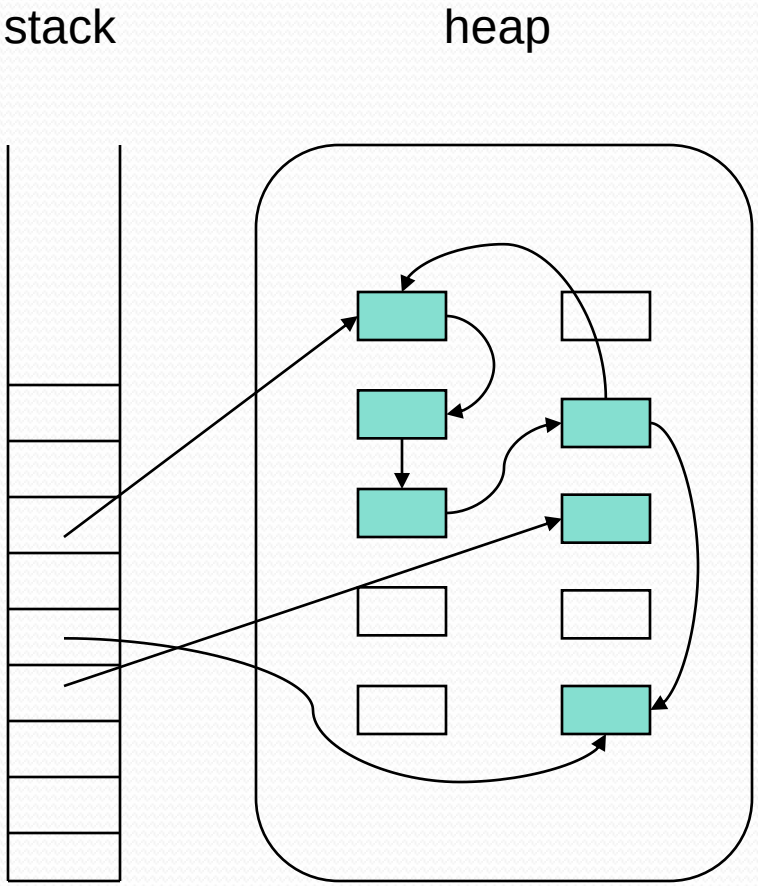
The marking phase

- To discover the dead objects, we
  - Find live objects
- Pointers from the stack to the heap make objects live



The marking phase

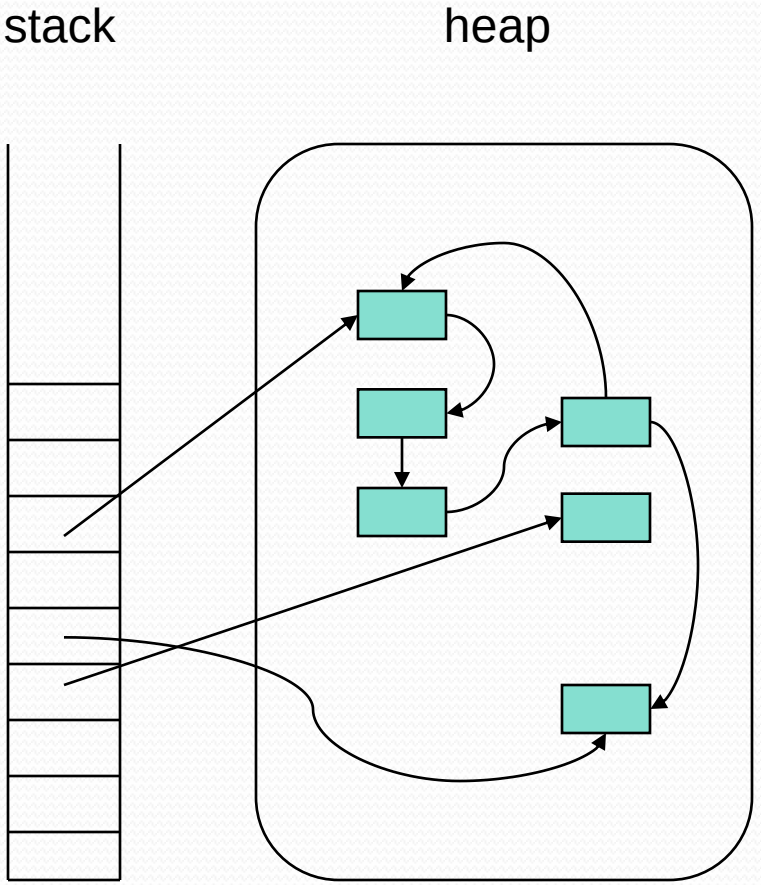
- To discover the dead objects, we
  - Find live objects



- Pointers from the stack to the heap make objects live
- These objects make other objects live

The sweep phase

- To discover the dead objects, we
  - Find live objects
  - Sweep all others away as dead



Mark and sweep: Tracing example

- To discover the dead objects, we
  - Find live objects
  - Sweep all others away as dead
  - Perhaps compact the heap
  - Problem:
    - Mark phase can take unbounded time