Garbage Collection: Introduction

Mainack Mondal
Sandip Chakraborty

CS 60203
Autumn 2024

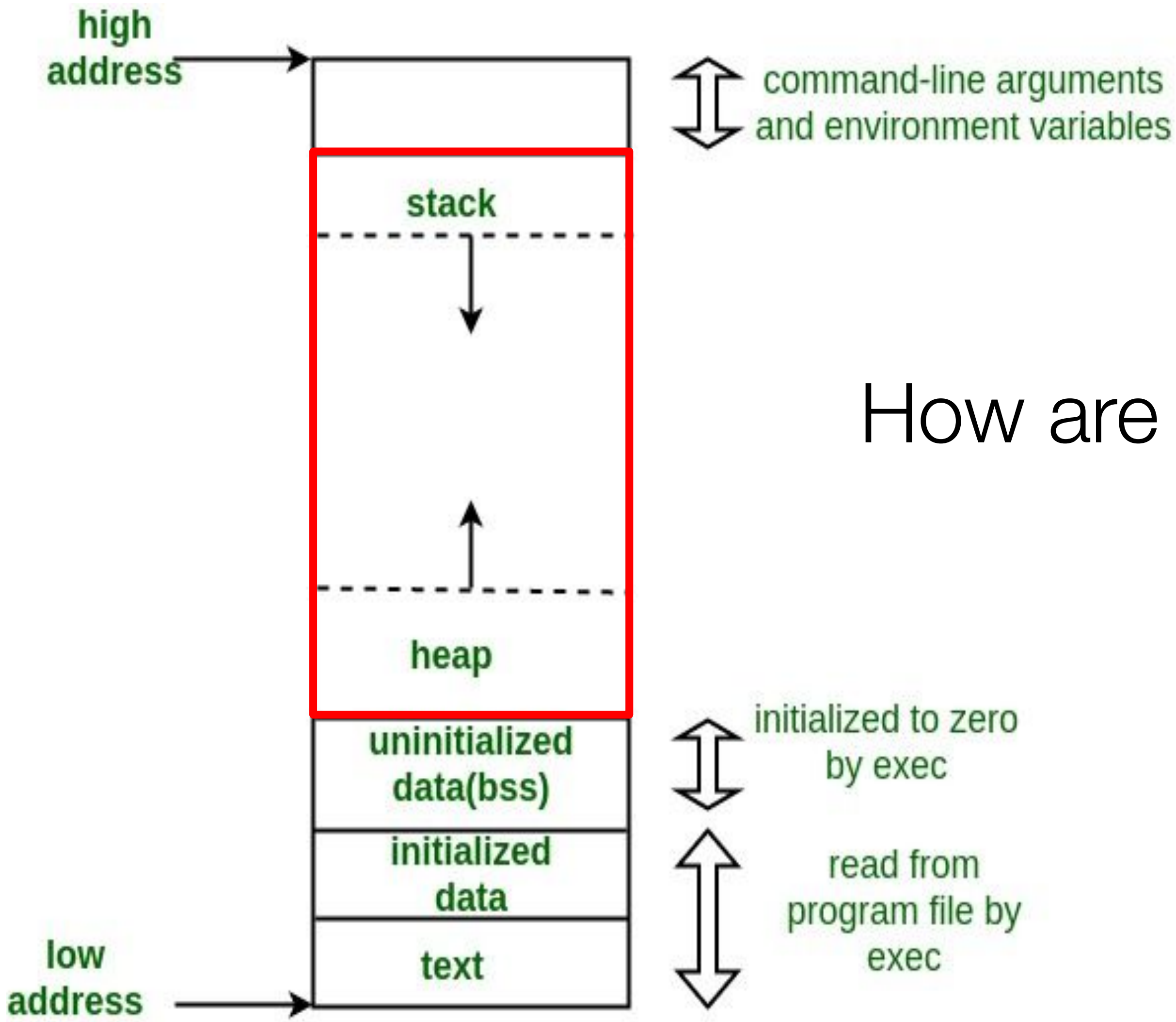
Outline

- Recap of Memory Structure and Memory Leaks
  - Overview of Memory Structure
  - What is Memory Leak?
  - Some Case Studies on Memory Leaks

- Introduction to Garbage Collection
  - What is Garbage? Examples
  - Why Garbage Collection ?
  - The Perfect Garbage Collector

Memory Structure and Memory Leaks

Memory Structure



How are Stack and Heap different?

Memory Structure

What do you think what happens when you write :

- `int x = 10;`
- `int arr[1000];`
- `int *arr = (int*)malloc(1000*sizeof(int));`

Major Areas of Memory Structure

- **Stack**
  - Fixed Stack
    - Has fixed size and content
    - Allocated at compile time
  - Variable Stack
    - Variable size and content (activation records)
    - Used for managing function calls and returns

- **Heap**
  - Fixed Size but variable content
  - Dynamic Allocation
    - Eg: `new` in C++ and Java, `malloc` in C

Memory Leaks

Memory Leaks happen when you allocate a memory in heap then you forget to free it

Why should we bother about it ?

- Reduces Performance
    - Because, it reduces the amount of available memory, and ultimately the application slows down or crashes

Have you ever seen this?

Memory Leaks (Contd.)

1) Will this code snippet cause memory leak?

```c
void do_something() {
    int arr[150];

    // do something

    return;
}
```

2) What about this?

```c
void process_data(int iterations) {
    for (int i = 0; i < iterations; i++) {
        int *data = (int *)malloc(sizeof(int) * 100);
        if (!data) {
            fprintf(stderr, "Memory allocation failed\n");
            exit(EXIT_FAILURE);
        }
        // Process data
    }
}
```

Memory Leaks (Contd.)

3) What about this ?

```c
void f()
{
    int* ptr = (int*)malloc(sizeof(int));

    /* Do some work */

    /* Memory allocated by malloc is released */
    free(ptr);
    return;
}
```

4) And… What about this ?

```c
void modify_string(char **str) {
    *str = (char *)malloc(50);
    if (!*str) return;

    strcpy(*str, "Modified String");
}

int main() {
    char *str = (char *)malloc(50);
    if (!str) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }
    modify_string(&str);

    free(str);
    return 0;
}
```

Memory Leaks and Software Engineering

- Memory leaks may sound funny, however they are not.
- Usually memory leaks are,
  - Low-impact until critical
    - Many web applications suffer from undetected memory leaks due to their subtle and cumulative effects
  - Hard to diagnose
    - What if your TODO app is taking more memory than DOTA?
  - Trivial to resolve, once diagnosed
    - You added `addEventListener` but forgot to call `removeEventListener`
    - You added a `DOM` node but forgot to remove it

A great reading: [Memory leaks: the forgotten side of web performance]()

Memory Leaks in the Wild

Memory Leak in **NASA's Mars Pathfinder**: [Source: Link]

- The Mars Pathfinder mission experienced frequent system resets due to a priority inversion problem, where a high-priority task was blocked by a low-priority task holding a resource that was also needed by a medium-priority task.
- The rover's operating system suffered from a memory leak due to tasks not properly releasing resources, causing exhaustion of available memory and system instability.

Memory Leaks **Android KitKat**: Android 4.4 (KitKat) had a memory leak issue related to its media player and surface flinger components. (Source: Link)

Memory Leaks in **Firefox**(Link), **Windows Server**(Link), **Windows Vista** (stackoverflow link, MSFN), etc.

How to detect Memory Leaks?

Using Memory Debuggers like WinDbg, Valgrind, sanitizers, etc.

- Valgrind (For more info, read: Lecture slides on Valgrind, Valgrind Home)
  - used for various purposes like memory leak detection, profiling etc.
  - does runtime interception
  - basically, runs your program in a sandbox
  - inserts its own instructions to do debugging stuff (Read about Dynamic Binary Instrumentation)

- Sanitizers (Address, Thread, Leak, Memory etc.) [Source: Google/Sanitizers]
  - compile time instrumentation (`-fsanitize=memory`)
  - provides both compile time and runtime analysis
  - how do they work? [Read: MemorySanitizer]

How to resolve Memory Leaks?

Manual Memory Management: Too tedious, more chances of mistakes

Then? Is there a way to automatically manage memory?

Yes!!

RAII: Resource Allocation is Initialisation

- It states that when allocating some memory, you should define its lifetime.
- Stack allocated objects are provided RAII by default
- What about Heaps?
    - For this we have smart pointers in C++ (std::unique_ptr, std::shared_ptr)
- However, not fully automatic
    - Needs to follow the RAII idiom (you cannot use pointers like raw pointers)

Garbage Collection

(Slides Courtesy: Vitaly Shmatikov)

Cells and Liveness

Cells: data item in the heap

- Cells are “pointed to” by pointers held in registers, stack, global/static memory, or in other heap cells

Roots: registers, stack locations, global/static variables

A cell is live if its address is held in a root or held by another live cell in the heap

What is Garbage?

Garbage is a block of heap memory that cannot be accessed by the program.

- An allocated block of heap memory does not have a reference to it (cell is no longer “live”)
- Another kind of memory error: a reference exists to a block of memory that is no longer allocated

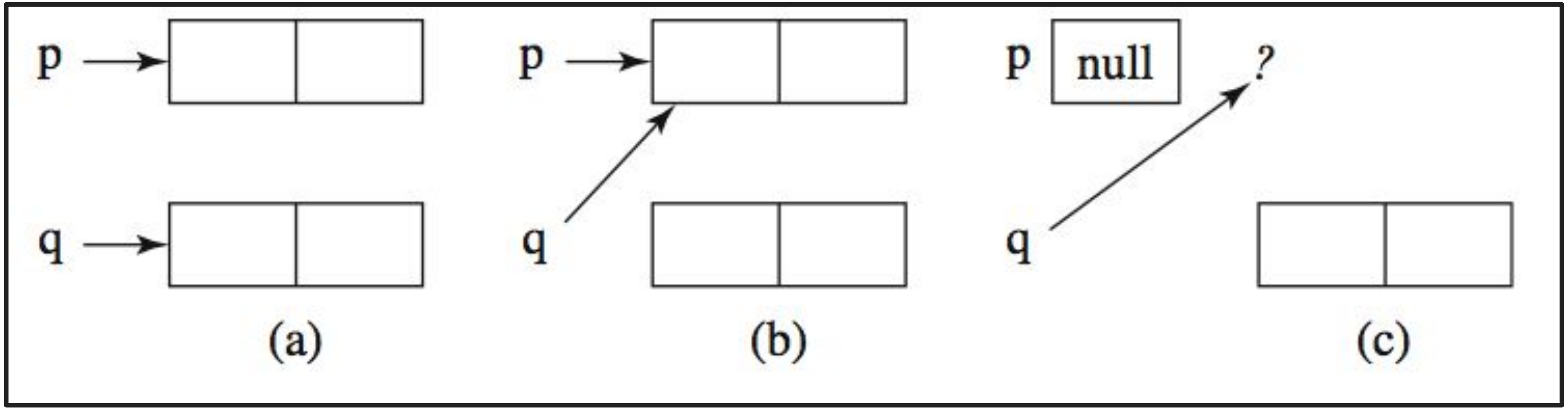
Garbage collection (GC) - automatic management of dynamically allocated storage.

- Reclaim unused heap blocks for later use by program

Garbage - An Example

```
class node {
     int value;
     node next;
}
node p, q;
```

```
p = new node();
q = new node();
q = p;
delete p;
```



GC and Programming Languages

- GC is not a language feature
- GC is a pragmatic concern for automatic and efficient heap management
  - Cooperative langs: Lisp, Scheme, Prolog, Smalltalk …
  - Uncooperative languages: C and C++
    - Although GC libraries have been built (Read: Boehm GC)
- GC in some well known languages:
  - Object Oriented Languages: Java, Go etc.
  - Functional Languages: Haskell, ML

The Perfect Garbage Collector

- No visible impact on program execution
- Works with any program and its data structures
  - For example, handles cyclic data structures
- Collects garbage (and only garbage) cells quickly
  - Incremental; can meet real-time constraints
- Has excellent spatial locality of reference
  - No excessive paging, no negative cache effects
- Manages the heap efficiently
  - Always satisfies an allocation request and does not fragment