CPSC-224
Software Development
Memory Allocation

Yu Wang

wangy2@gonzaga.edu

Jan 31, 2025

Announcement

- Quiz Day (Questions on Canvas, 15mins, 100points)
- Homework1 (Check the updates on Canvas)
  - CPSC 224 - GitHub Classroom Assignment Instructions
  - Deadline for submission: Feb. 12
- Homework2 coming soon

Daily Attendance (01)

- Scan the QR Code for yourself

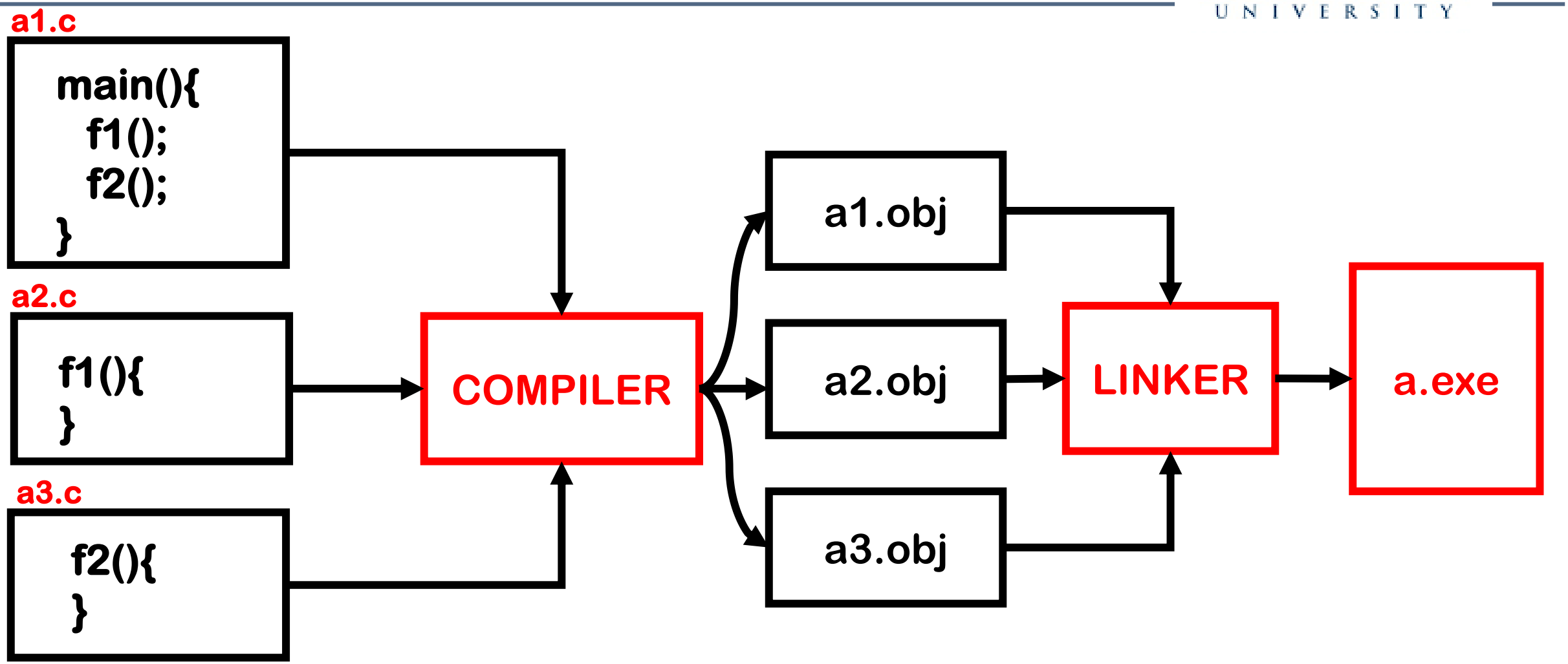
Daily Attendance (02)

- Scan the QR Code for yourself

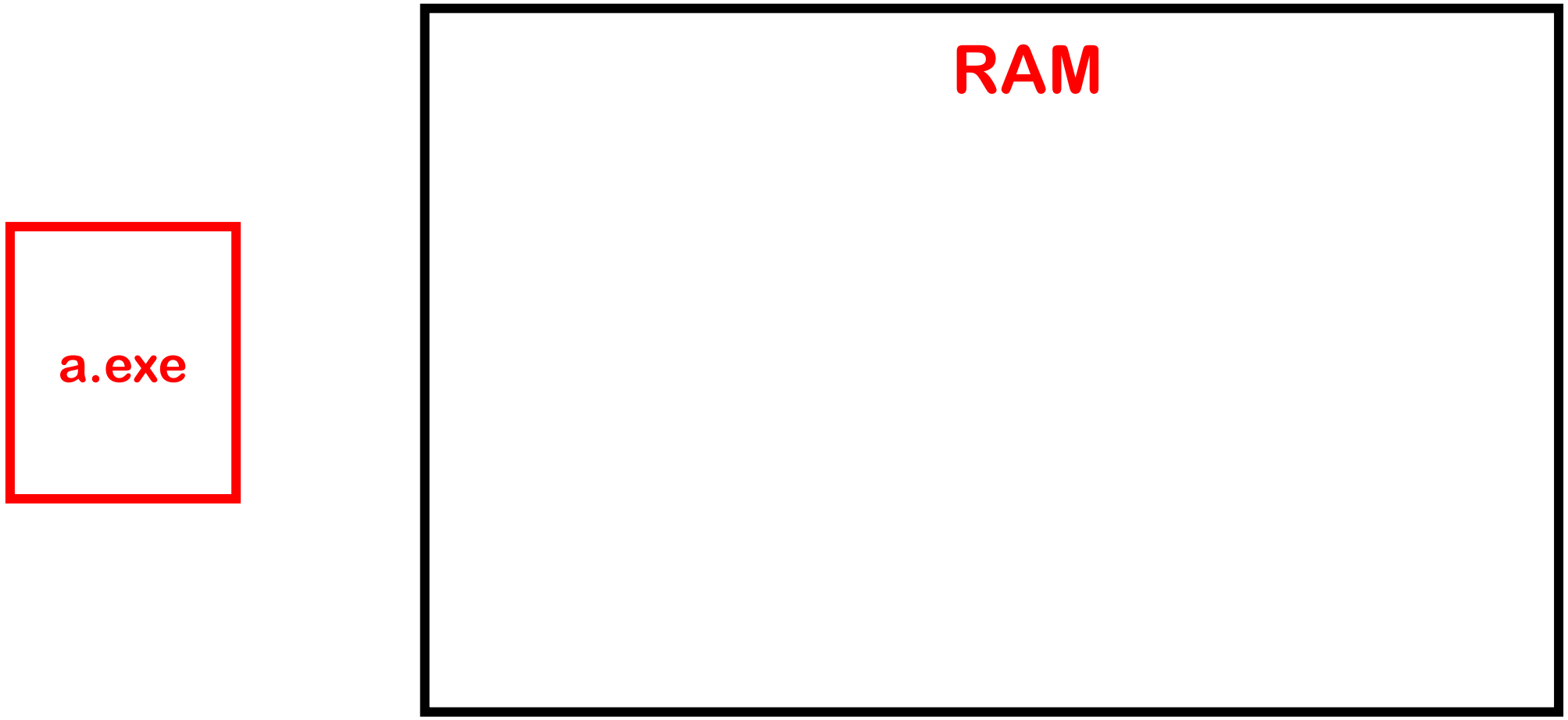
Review - Last Class

- ✓ We learned – What is JVM, JRE and JDK
- ✓ We learned – The compelling process in C and Java. What is the difference between them.
- ✓ We learned – The important facts about memory allocation in java.
- ✓ We learned – Heap and Stack
- ✓ We learned – The relationship between pointers and stack.

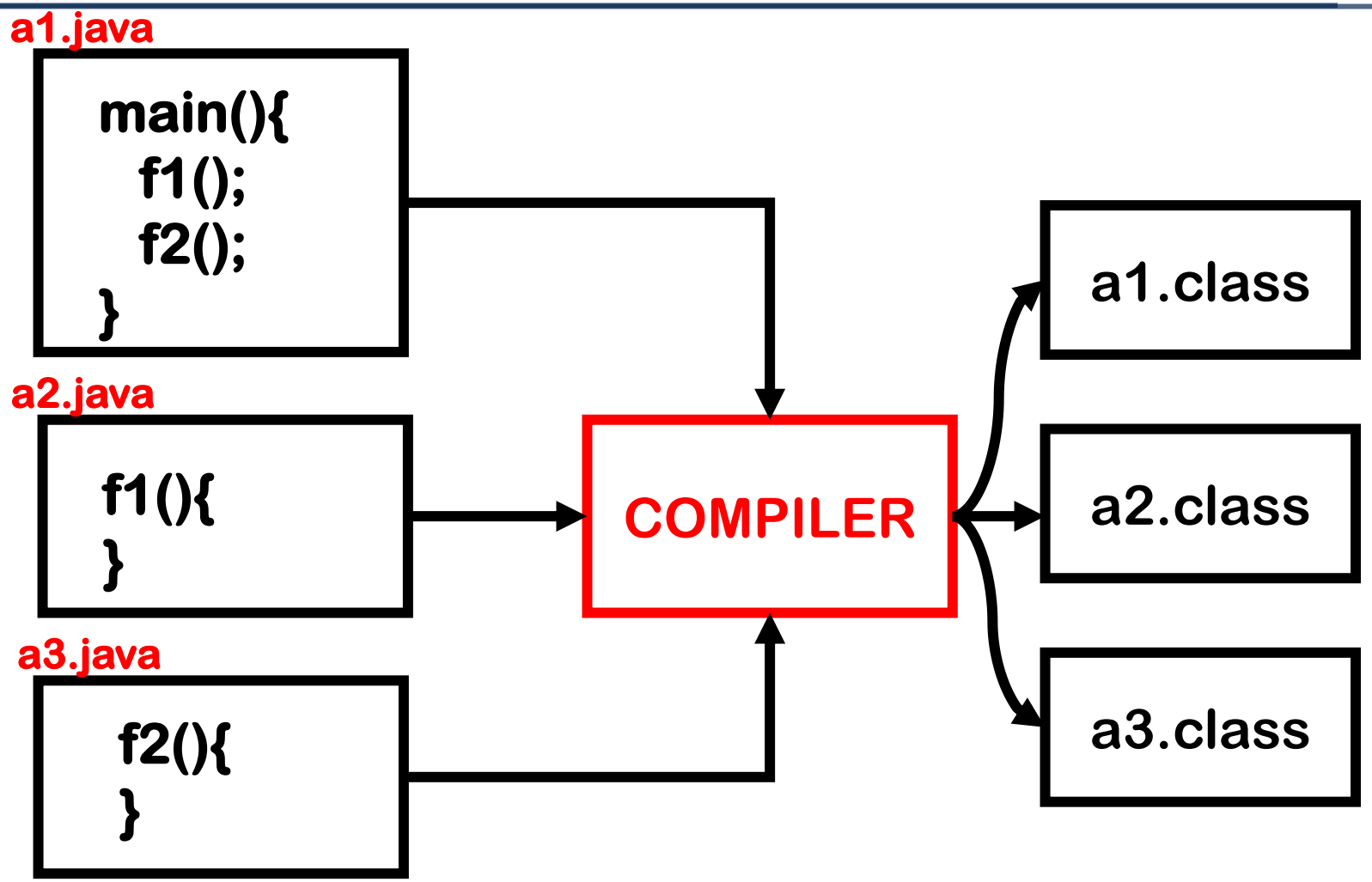
Java Virtual Machine (JVM) – C Example



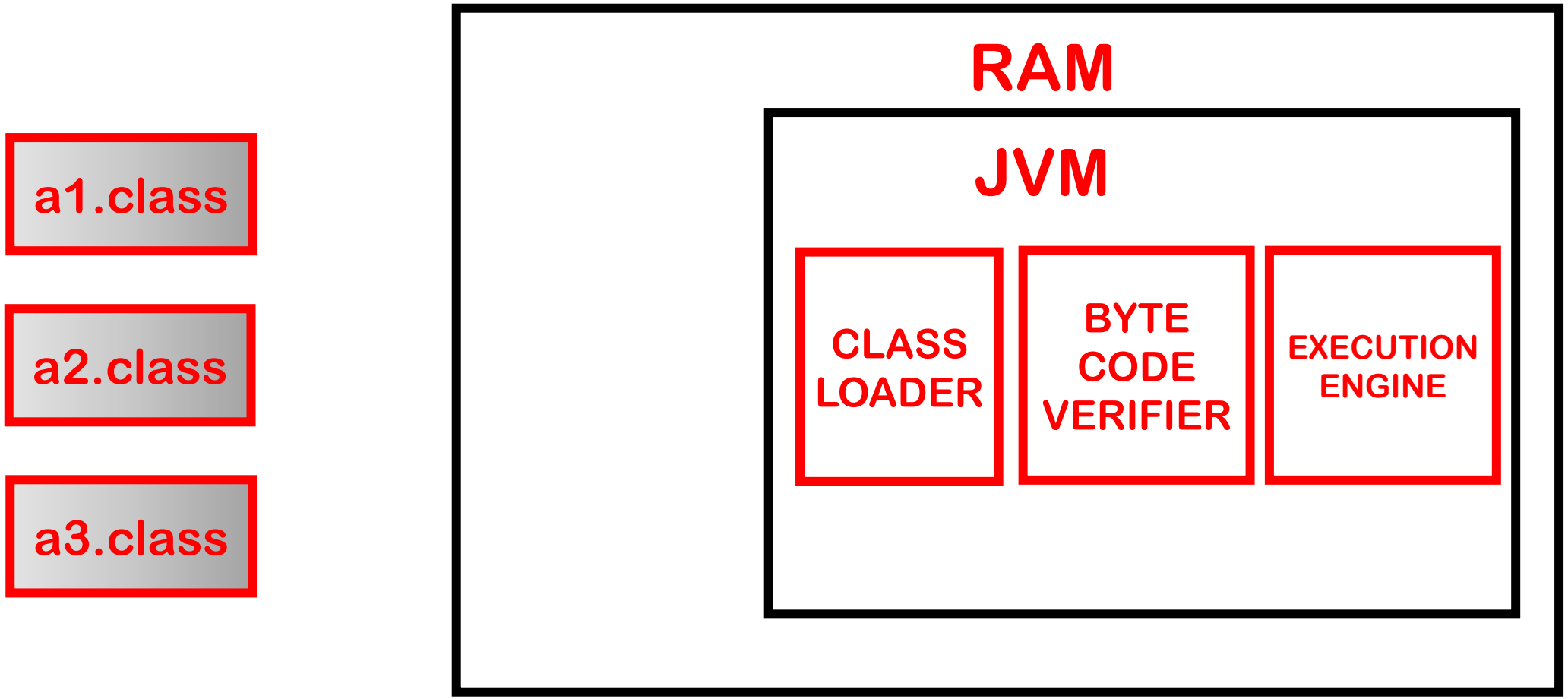
Java Virtual Machine (JVM) – C Process



Java Virtual Machine (JVM) – Java Process

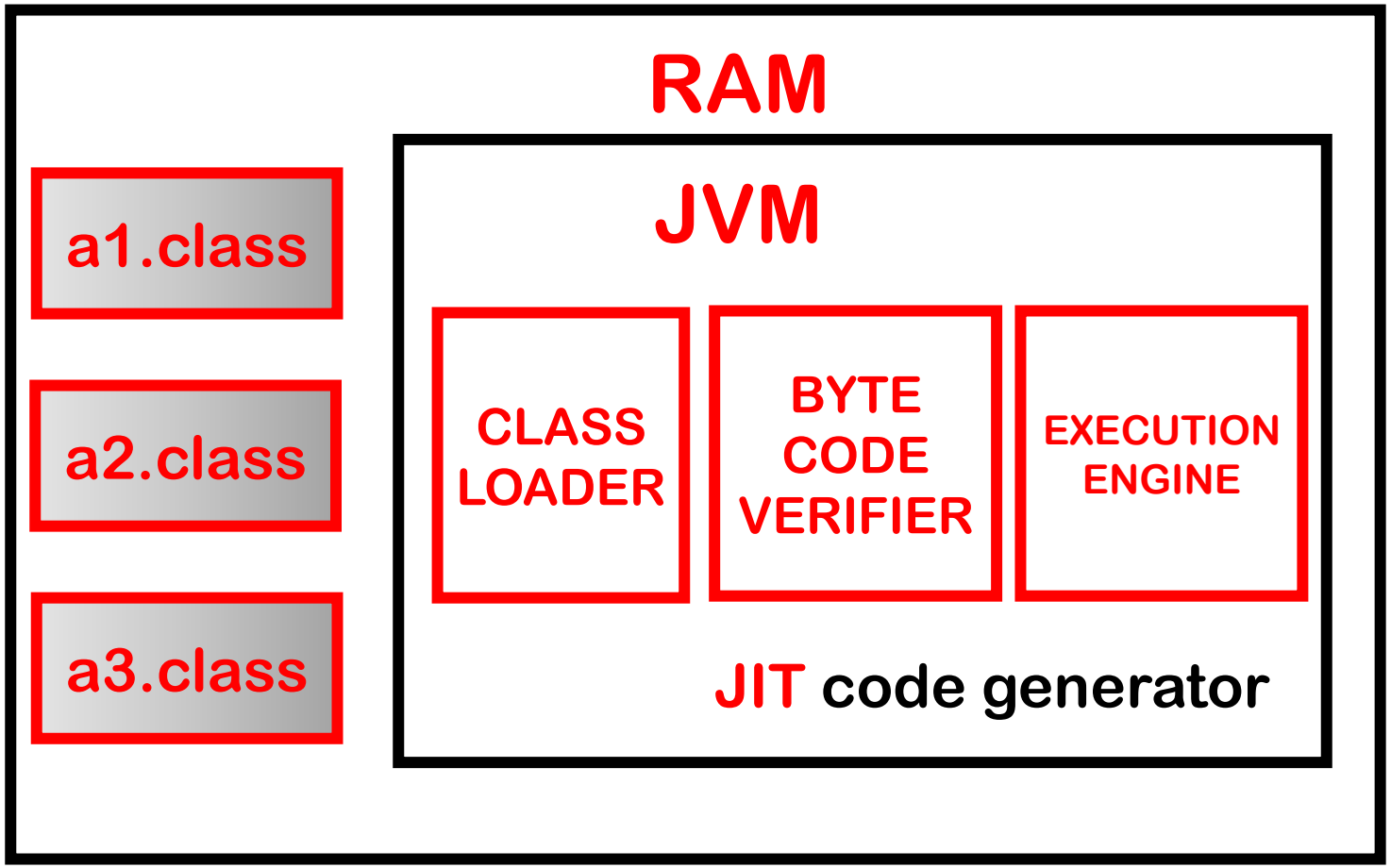


Java Virtual Machine (JVM) – Java Process

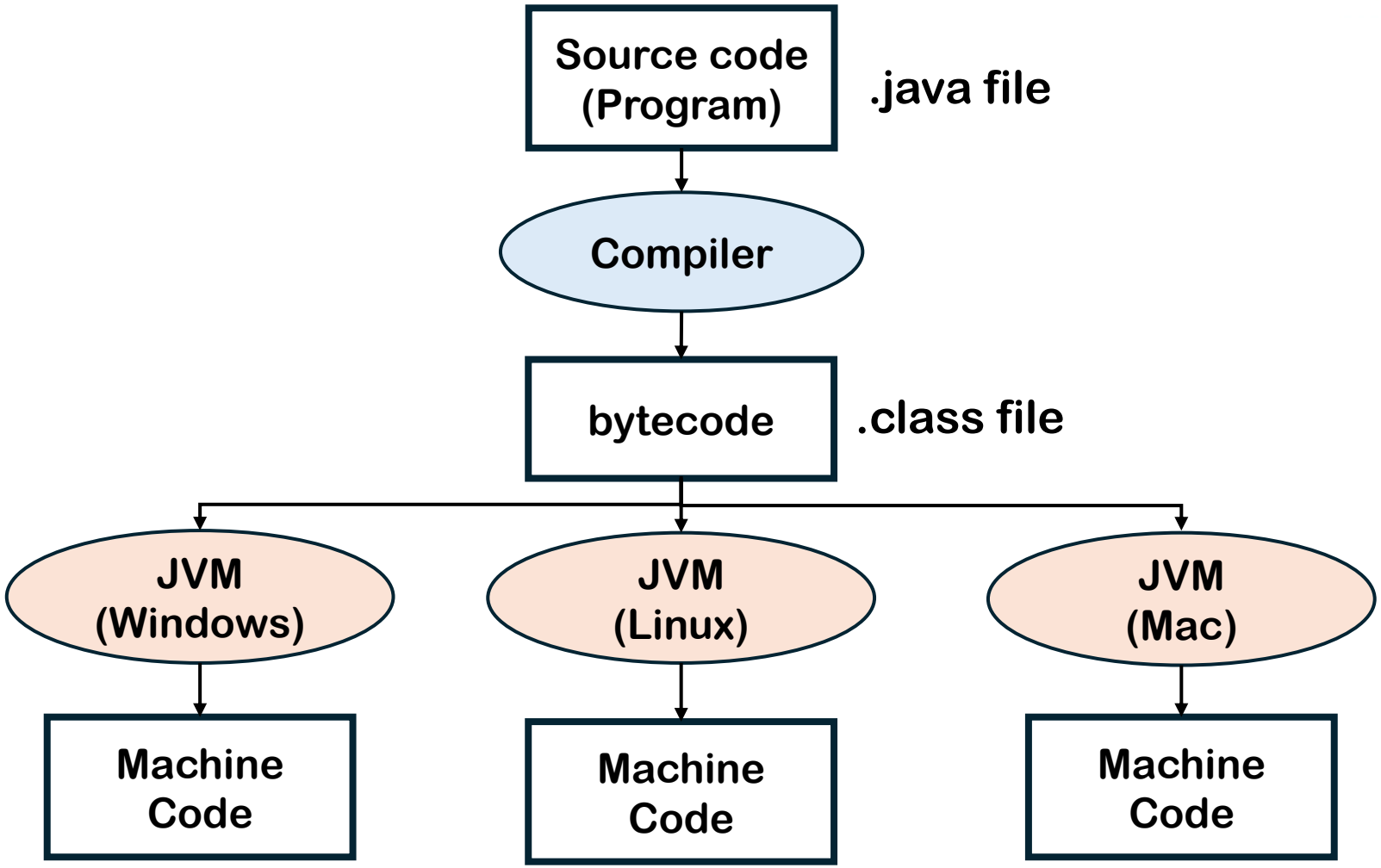


Java Virtual Machine (JVM) – Java Process

- JIT -The JIT (Just-In-Time) code generator is a critical component of the Java Virtual Machine (JVM) that improves the performance of Java applications by converting bytecode into native machine code at runtime .



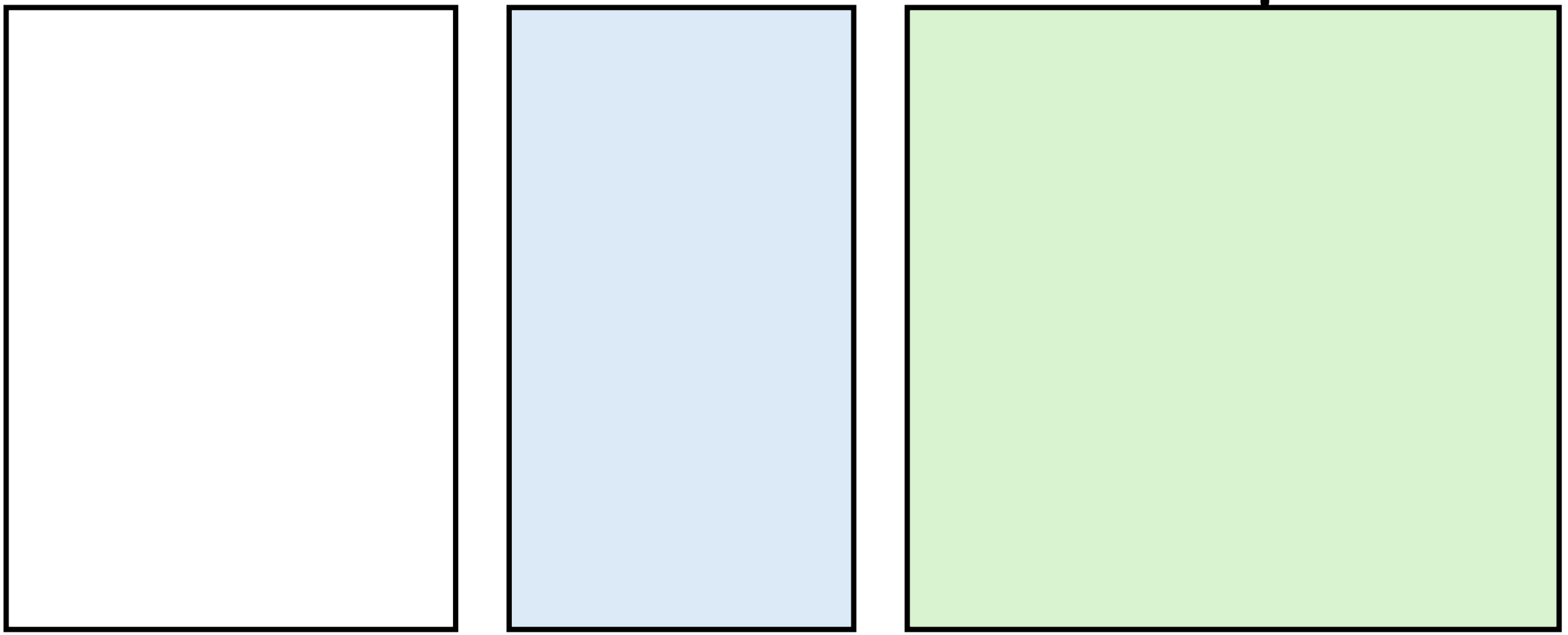
What is Java Bytecode?



Memory Allocation

- All data for primitive type variables is stored on the stack
- For reference types, the stack holds a pointer to the object on the heap
- When setting a reference type variable equal to another reference type variable, a copy of only the pointer is made
- Certain object types can not be operated on the heap

Memory Allocation



Memory Allocation

GONZAGA UNIVERSITY

Code

```
int a = 3;
int b = a;
b = 100;
Int [] c = {1,2,3,4};
Int [] d = c;
```

Stack

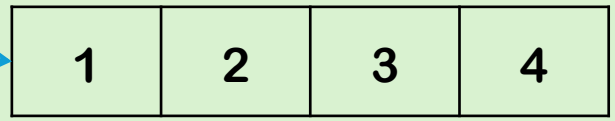
d =

c =

b = 100;

a = 3;

Heap



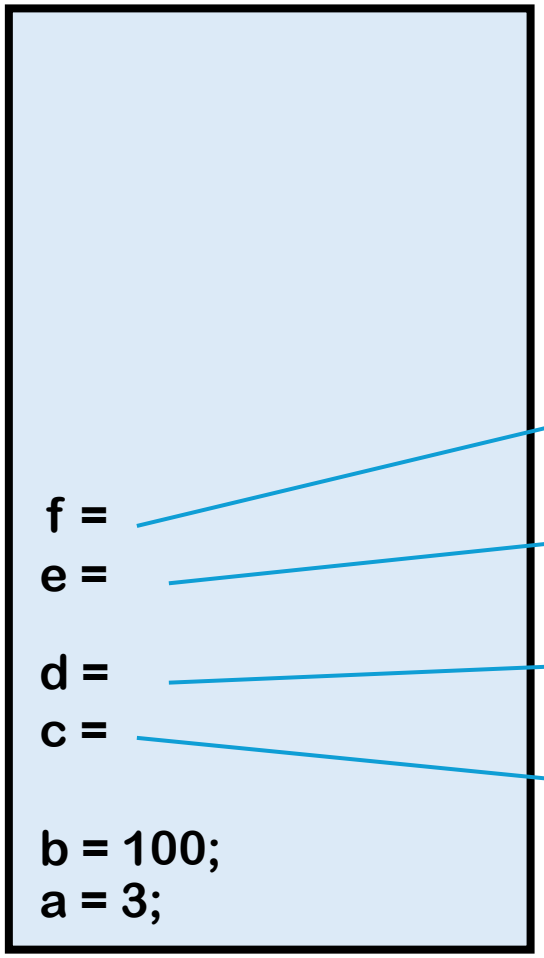
Memory Allocation

GONZAGA UNIVERSITY

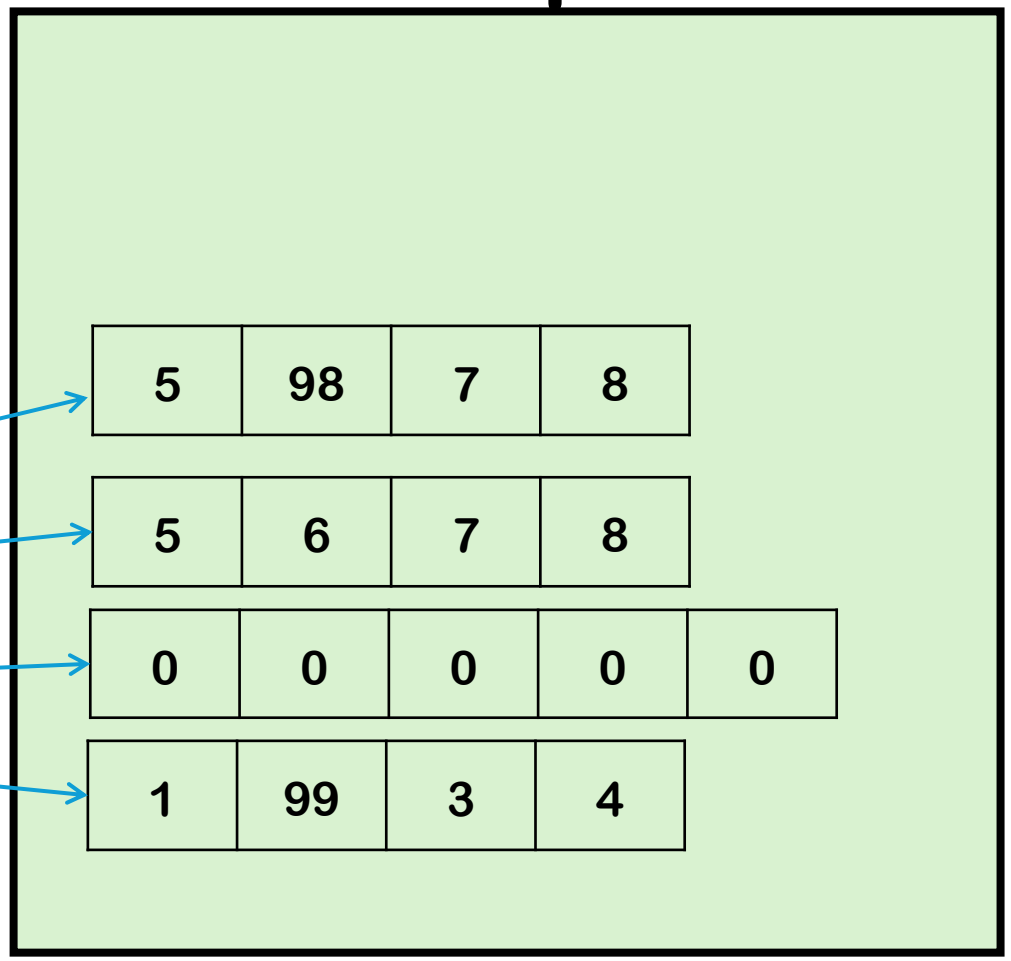
Code

```
int a = 3;
int b = a;
b = 100;
Int [] c = {1,2,3,4};
Int [] d = c;
d[1] = 99;
d = new int [5];
Int [] e = {5,6,7,8};
Int [] f = {5,6,7,8};
f[1] = 98;
```

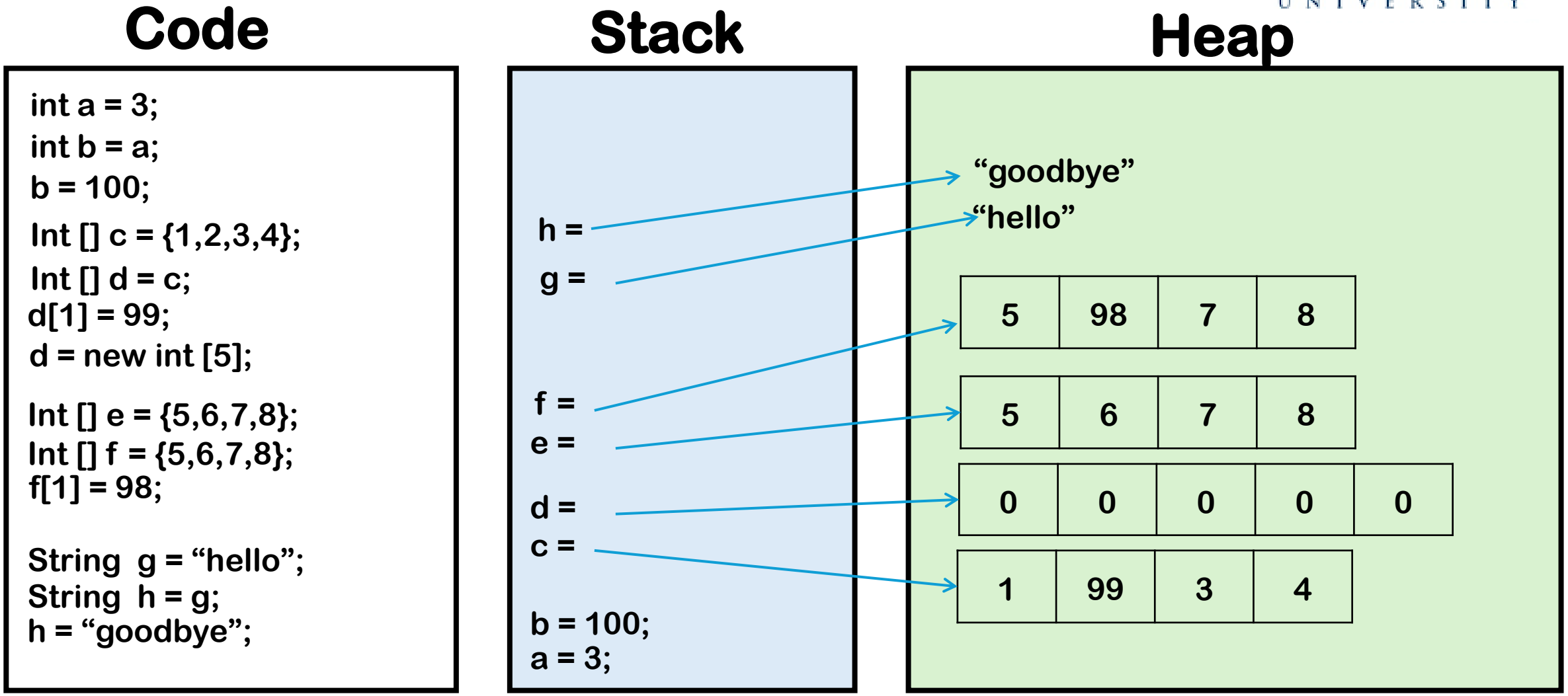
Stack



Heap



Memory Allocation

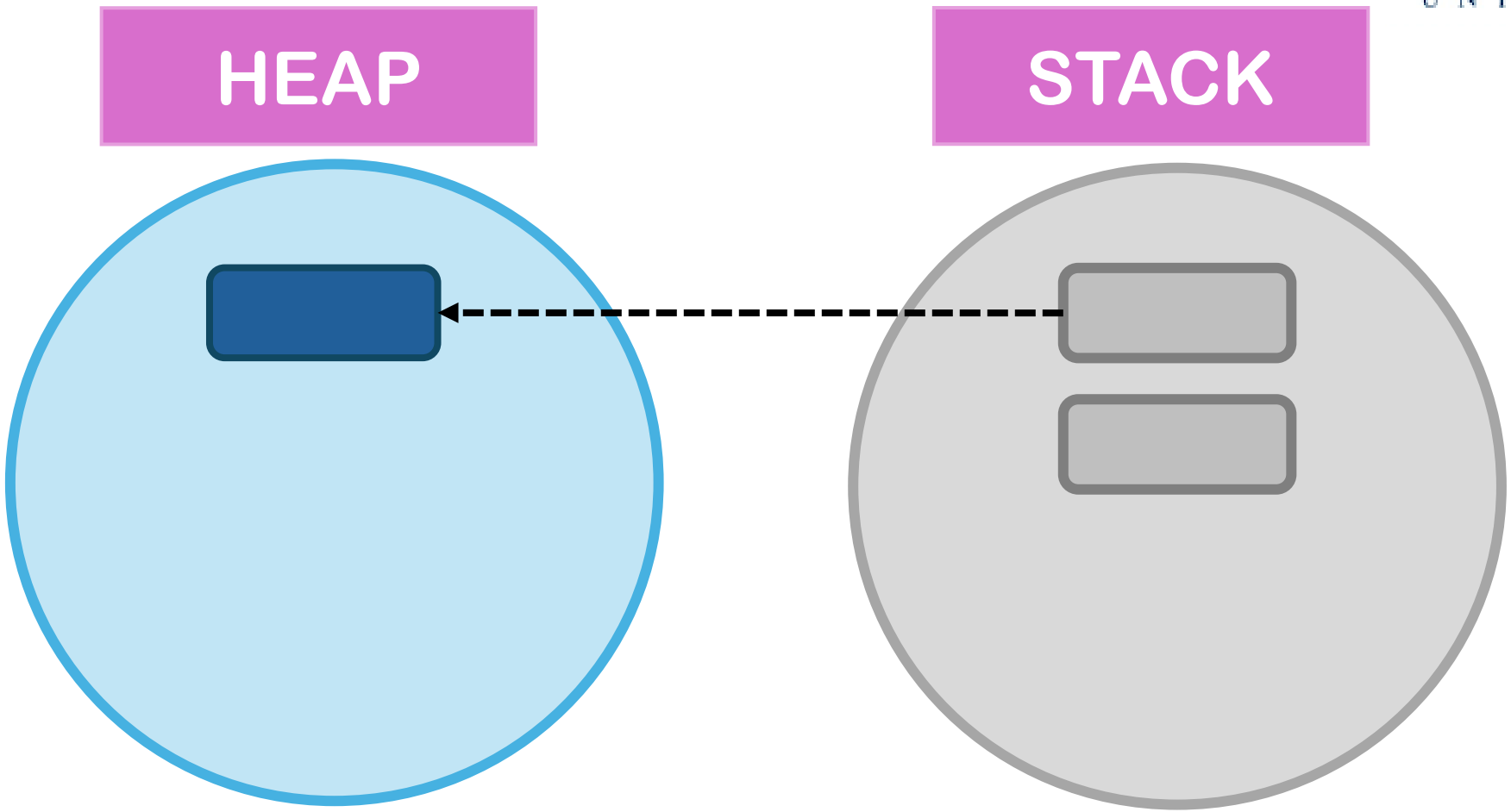


Relationship Between Pointers and Stack

GONZAGA UNIVERSITY

- Pointers can store stack addresses.
- Stack variables are automatically deallocated, so using a pointer to a stack variable after function return leads to dangling pointers.

Memory Allocation Summary



Garbage Collection

GONZAGA UNIVERSITY

What is Automatic Garbage Collection?

- Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

- In a programming language like C, allocating and deallocating memory is a manual process. In Java, process of deallocating memory is handled automatically by the garbage collector. The basic process can be described as follows.

Methods to Request Garbage Collection

Although we cannot force garbage collection, we can request it using:

- System.gc(); → Suggests that the JVM run the garbage collector.
- Runtime.getRuntime().gc(); → Another way to request GC.

Note: Calling System.gc() does not guarantee that garbage collection will run immediately.

Types of Garbage Collectors in Java

Java provides different types of garbage collectors:

1. **Serial Garbage Collector**: Works with a single thread, best for small applications.
2. **Parallel Garbage Collector**: Uses multiple threads for garbage collection, improving performance.
3. **CMS (Concurrent Mark-Sweep) Garbage Collector**: Reduces application pauses by collecting garbage concurrently.
4. **G1 (Garbage First) Garbage Collector**: Designed for large heap sizes and reduces pause times.

Supplement - arguments

In Java, **arguments** are the **actual values** that you pass to a method when you call it.

Parameters: When you define a method, you can specify variables that the method will accept as input. These are called parameters.

Arguments: When you call the method, you provide specific values for those parameters. These values are called arguments.

Supplement – arguments example

GONZAGA UNIVERSITY

```java
public class Example {

    public static void greet(String name) { // 'name' is a parameter
        System.out.println("Hello, " + name + "!");
    }

    Run | Debug
    public static void main(String[] args) {
        greet("Alice"); // "Alice" is an argument passed to the 'greet' method
    }
}
```

Any Questions?