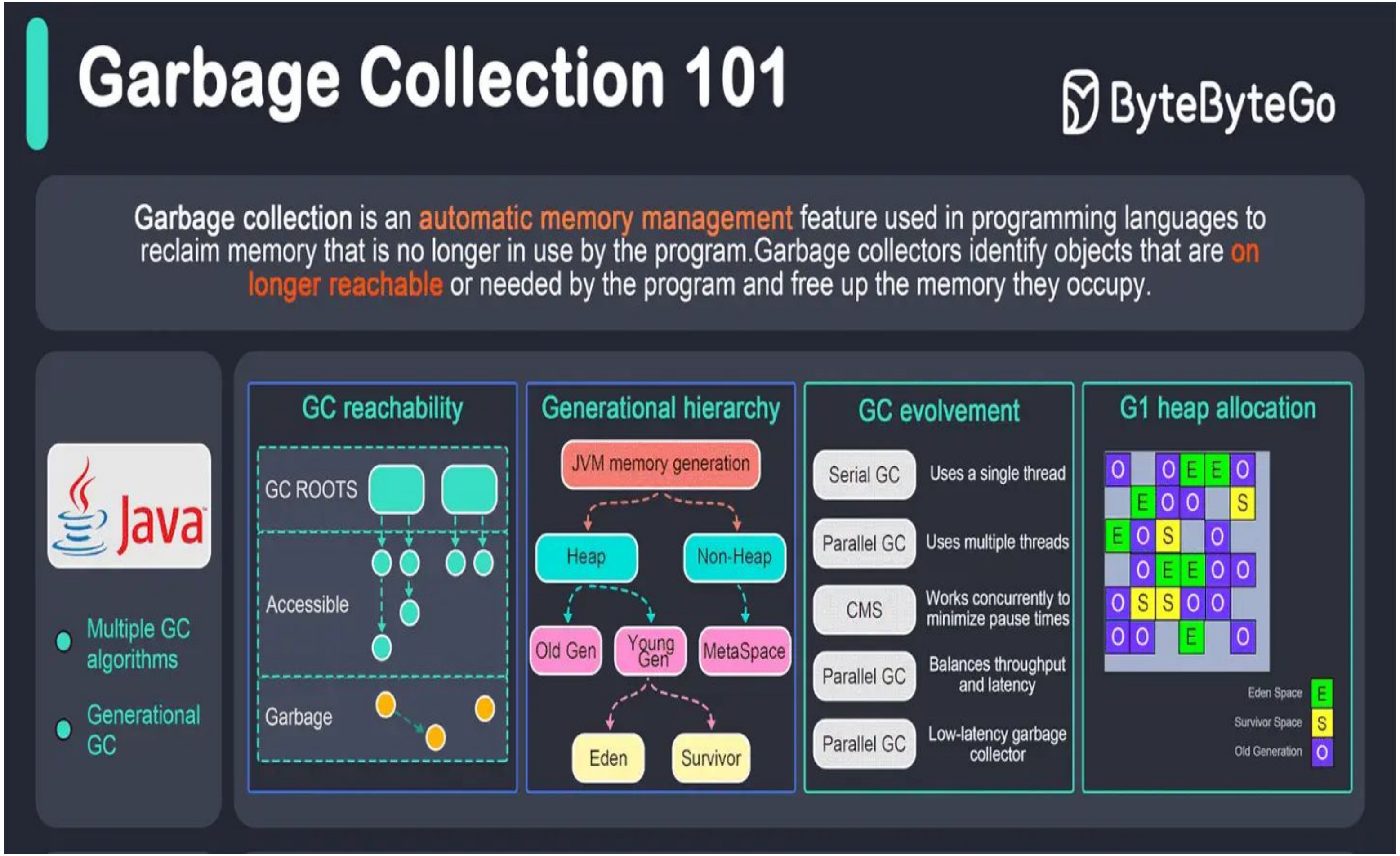
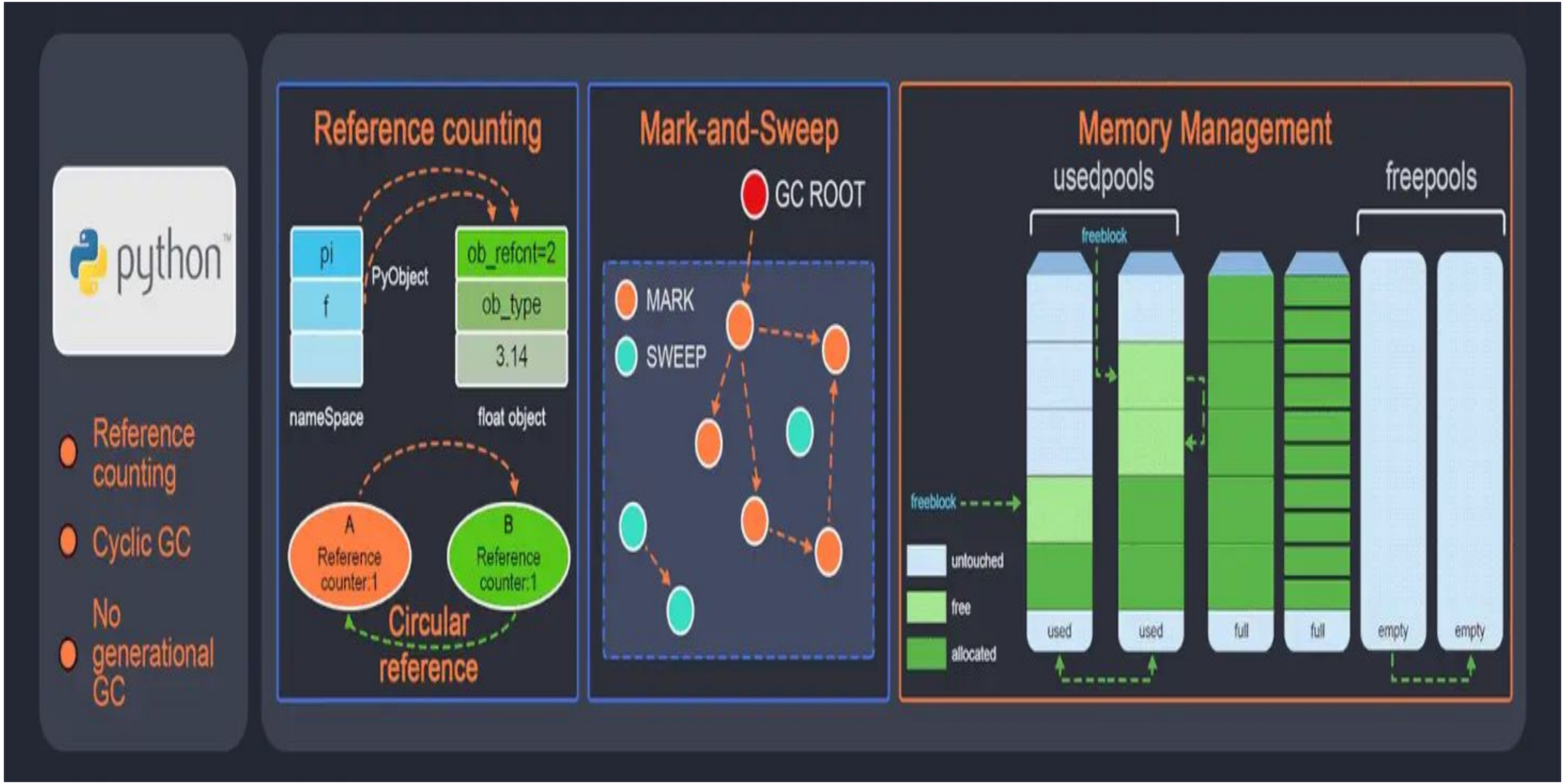
Java Virtual Machine 2

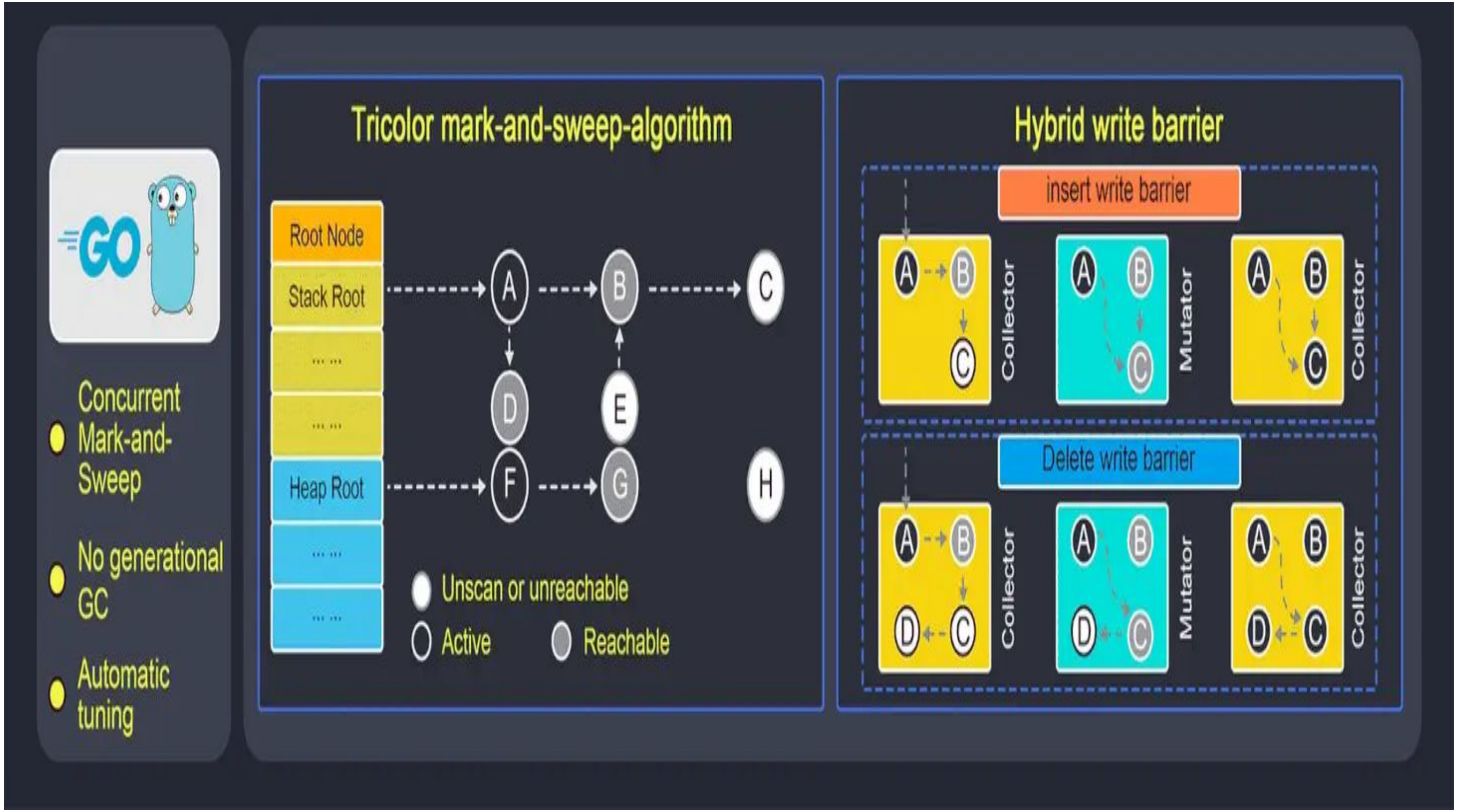
https://blog.bytebytego.com/p/ep125-how-does-garbage-collection

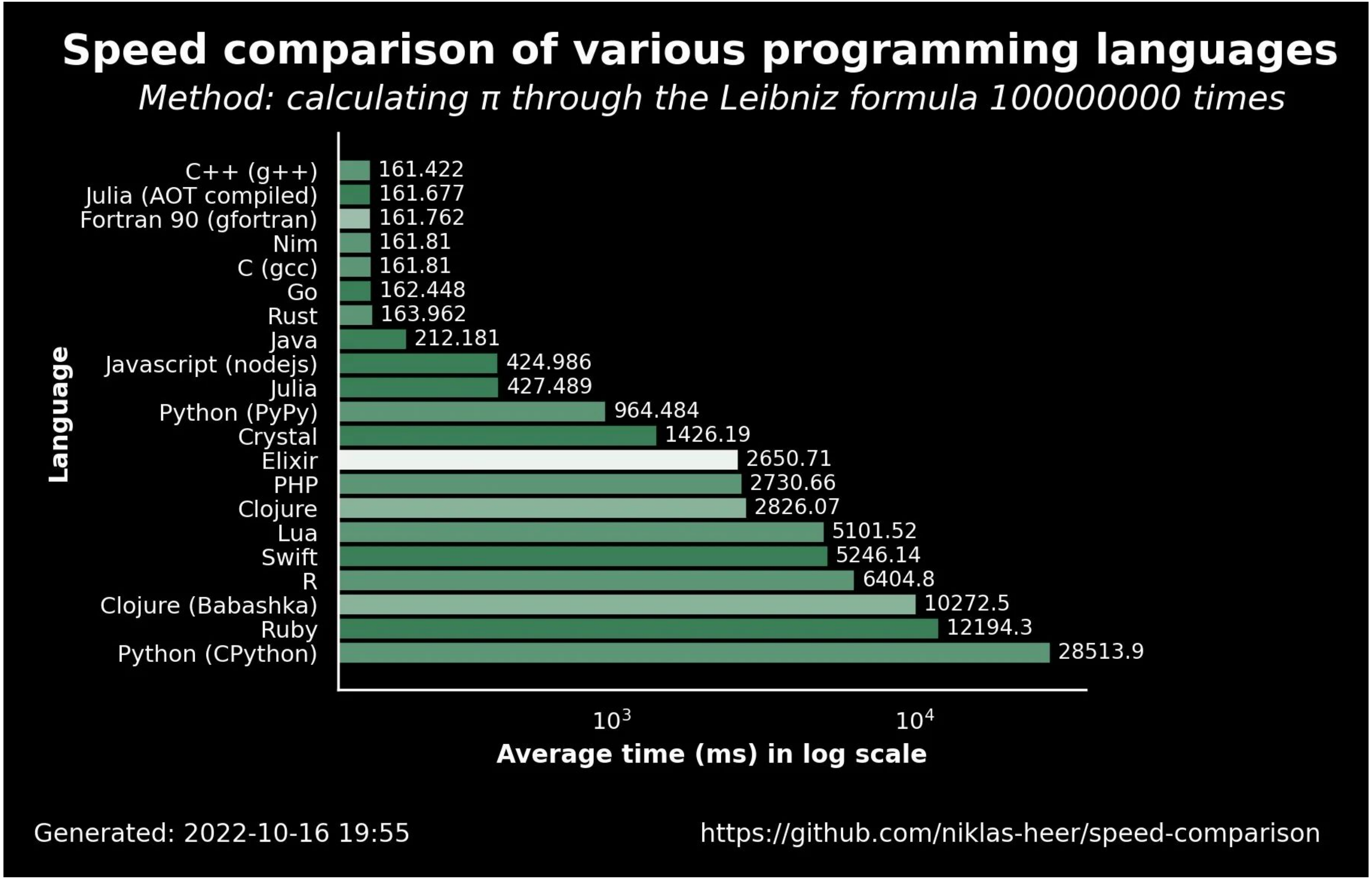


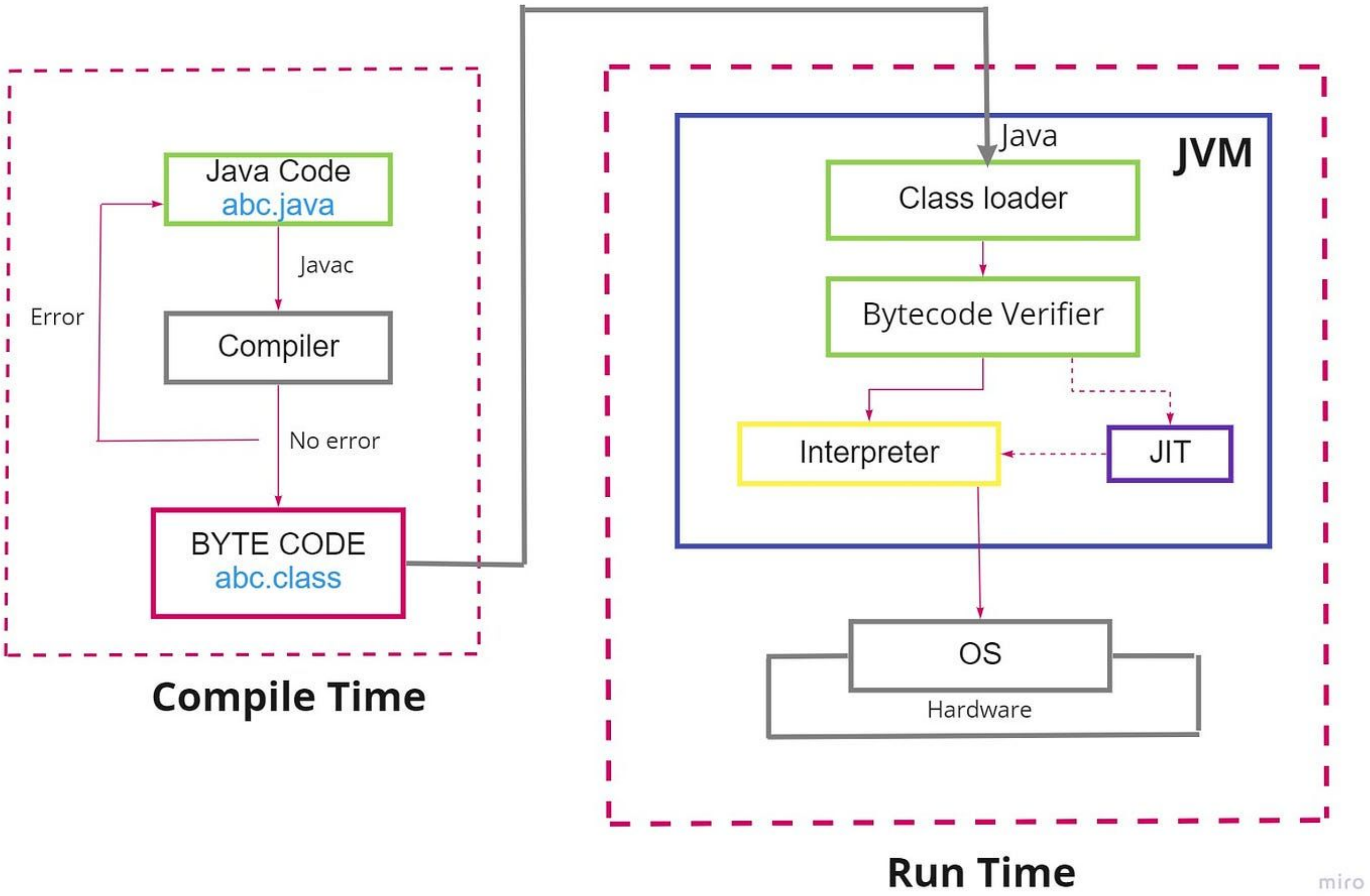
https://blog.bytebytego.com/p/ep125-how-does-garbage-collection



https://blog.bytebytego.com/p/ep125-how-does-garbage-collection




Speed comparison of various programming languages
Method: calculating π through the Leibniz formula 100000000 times
Language
C++ (g++)
Julia (AOT compiled)
Fortran 90 (gfortran)
Nim
C (gcc)
Go
Rust
Java
Javascript (nodejs)
Julia
Python (PyPy)
Crystal
Elixir
PHP
Clojure
Lua
Swift
R
Clojure (Babashka)
Ruby
Python (CPython)
161.422
161.677
161.762
161.81
161.81
162.448
163.962
212.181
424.986
427.489
964.484
1426.19
2650.71
2730.66
2826.07
5101.52
5246.14
6404.8
10272.5
12194.3
28513.9
10^3
10^4
Average time (ms) in log scale
Generated: 2022-10-16 19:55
https://github.com/niklas-heer/speed-comparison


Java Code
abc.java
Javac
Compiler
Error
No error
BYTE CODE
abc.class
Compile Time
Java
JVM
Class loader
Bytecode Verifier
Interpreter
JIT
OS
Hardware
Run Time
miro

Search Labs | AI Overview

The Mandelbrot set is computationally intensive because it requires iterating a complex mathematical function for each pixel in an image, and due to its fractal nature, the number of iterations needed to determine if a point belongs to the set can vary drastically depending on its location, often requiring a large number of calculations to accurately render intricate details, especially near the boundary of the set.

Key points about the computational intensity of the Mandelbrot set:

Fractal complexity:

The Mandelbrot set exhibits self-similarity, meaning that zooming into any part reveals similar patterns repeating at smaller scales, leading to an infinite level of detail that needs to be calculated for accurate rendering.

Iterative process:

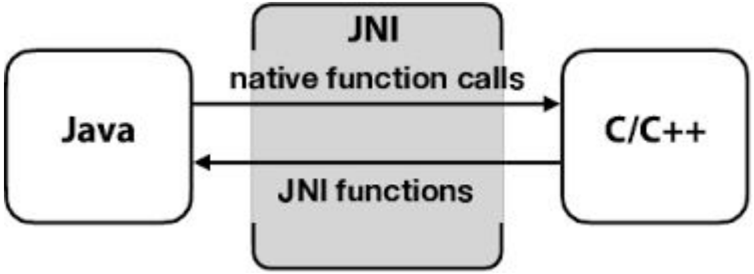
To determine if a point belongs to the Mandelbrot set, a complex number is repeatedly squared and added to itself (iteration) until either it diverges to infinity (not in the set) or remains bounded (in the set).

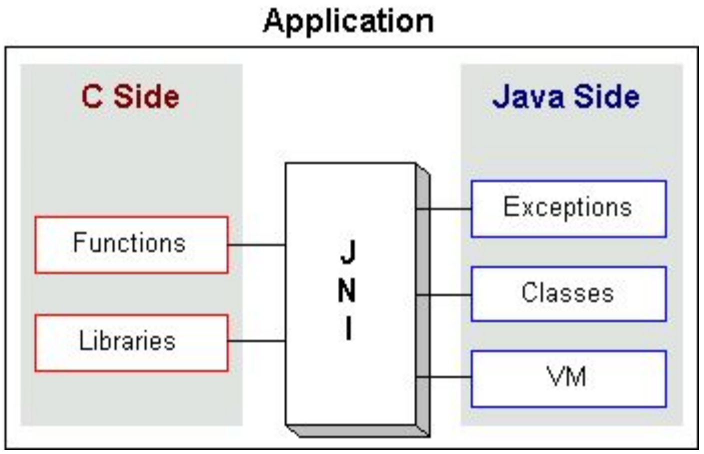
Large number of iterations:

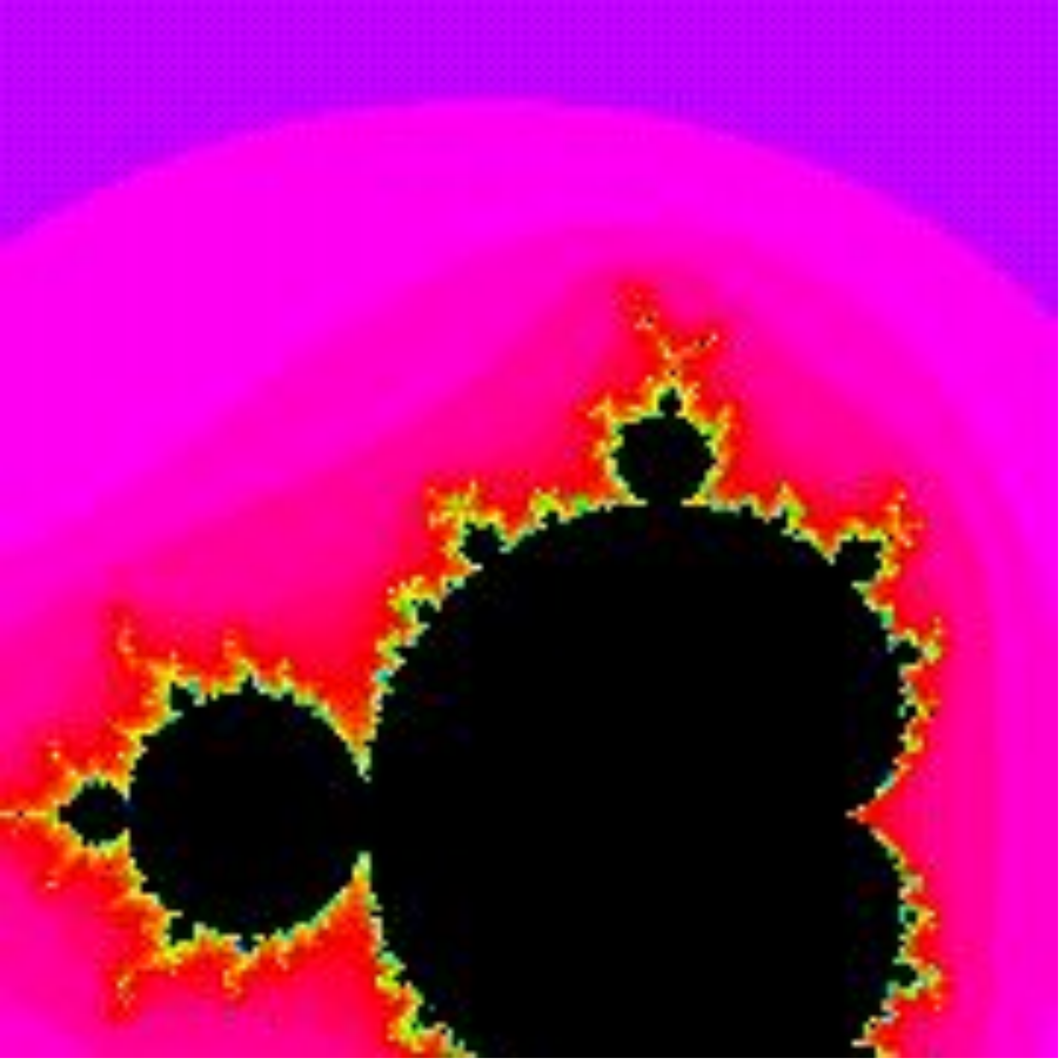
Depending on the location of a point, it can take a large number of iterations to determine if it belongs to the set, especially near the boundary where complex patterns emerge.

Pixel-by-pixel calculation:

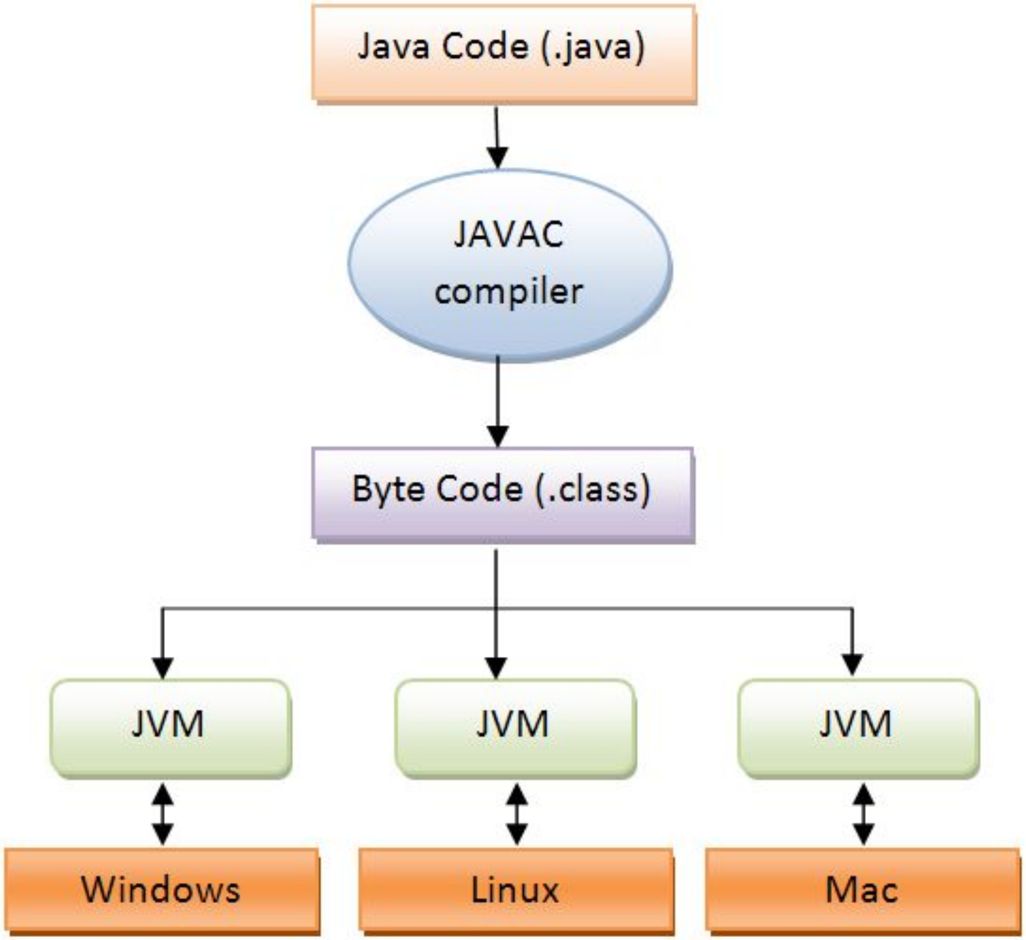
To generate a visual representation of the Mandelbrot set, each pixel in the image needs to be individually calculated based on its corresponding complex number.

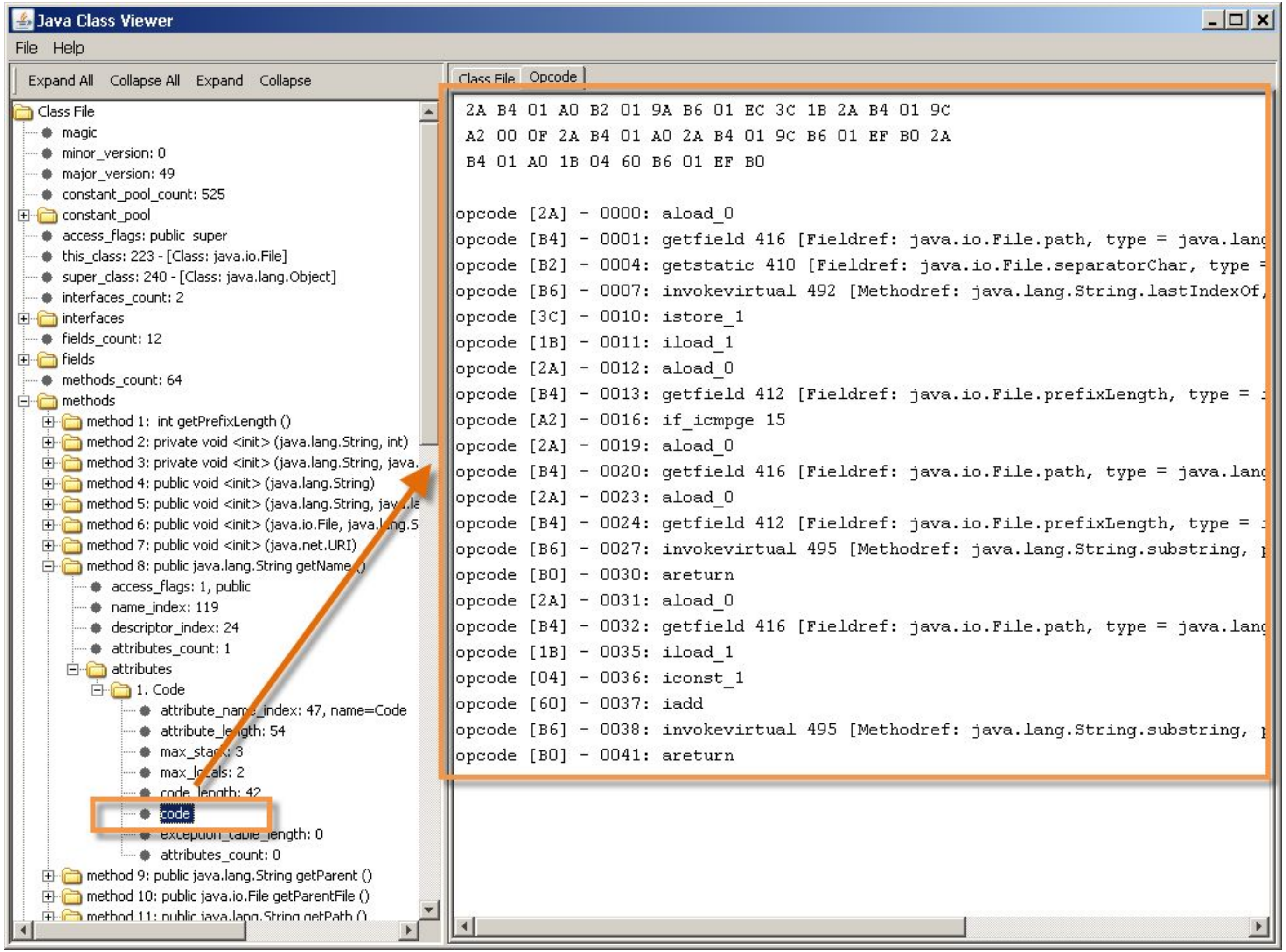






https://www.youtube.com/watch?v=b005iHf8Z3g


Java Code (.java)
JAVAC
compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac


Java Class Viewer
File Help
Expand All Collapse All Expand Collapse
Class File
magic
minor_version: 0
major_version: 49
constant_pool_count: 525
constant_pool
access_flags: public super
this_class: 223 - [Class: java.io.File]
super_class: 240 - [Class: java.lang.Object]
interfaces_count: 2
interfaces
fields_count: 12
fields
methods_count: 64
methods
method 1: int getPrefixLength ()
method 2: private void <init> (java.lang.String, int)
method 3: private void <init> (java.lang.String, java.
method 4: public void <init> (java.lang.String)
method 7: public void <init> (java.net.URI)
method 8: public java.lang.String getName ()
access_flags: 1, public
name_index: 119
descriptor_index: 24
attributes_count: 1
attributes
1. Code
attribute_length: 54
max_locals: 2
code
attributes_count: 0
method 9: public java.lang.String getParent ()
method 10: public java.io.File getParentFile ()
Class File Opcode
2A B4 01 A0 B2 01 9A B6 01 EC 3C 1B 2A B4 01 9C
A2 00 0F 2A B4 01 A0 2A B4 01 9C B6 01 EF B0 2A
B4 01 A0 1B 04 60 B6 01 EF B0
opcode [2A] - 0000: aload_0
opcode [B4] - 0001: getfield 416 [Fieldref: java.io.File.path, type = java.lang
opcode [B2] - 0004: getstatic 410 [Fieldref: java.io.File.separatorChar, type =
opcode [B6] - 0007: invokevirtual 492 [Methodref: java.lang.String.lastIndexOf,
opcode [3C] - 0010: istore_1
opcode [1B] - 0011: iload_1
opcode [2A] - 0012: aload_0
opcode [B4] - 0013: getfield 412 [Fieldref: java.io.File.prefixLength, type =
opcode [A2] - 0016: if_icmpge 15
opcode [2A] - 0019: aload_0
opcode [B4] - 0020: getfield 416 [Fieldref: java.io.File.path, type = java.lang
opcode [2A] - 0023: aload_0
opcode [B4] - 0024: getfield 412 [Fieldref: java.io.File.prefixLength, type =
opcode [B6] - 0027: invokevirtual 495 [Methodref: java.lang.String.substring,
opcode [B0] - 0030: areturn
opcode [2A] - 0031: aload_0
opcode [B4] - 0032: getfield 416 [Fieldref: java.io.File.path, type = java.lang
opcode [1B] - 0035: iload_1
opcode [04] - 0036: iconst_1
opcode [60] - 0037: iadd
opcode [B6] - 0038: invokevirtual 495 [Methodref: java.lang.String.substring,
opcode [B0] - 0041: areturn

Example: Evaluating a Simple Math Expression Using a Stack

Let's evaluate the expression:

$$(2+3) \times 4$$

using **push and pop operations** on a **stack-based virtual machine (like the JVM stack).**

Step 1: Convert to Postfix Notation (Reverse Polish Notation - RPN)

The JVM **stack-based execution** doesn't use infix notation like `2 + 3 * 4`. Instead, it converts the expression to **postfix notation:**

Copy Edit

```
2 3 + 4 *
```

Step 3: Equivalent JVM Bytecode

Here's how the **JVM bytecode** would execute this:

plaintext Copy Edit

```
ICONST_2      // Push 2 onto the stack
ICONST_3      // Push 3 onto the stack
IADD          // Pop 3, Pop 2 → Push (2 + 3) = 5
ICONST_4      // Push 4 onto the stack
IMUL          // Pop 4, Pop 5 → Push (5 * 4) = 20
```

```
Constant pool:
   #1 = Methodref          #10.#22        // java/lang/Object."<init>":()V
   #2 = Fieldref           #23.#24        // java/lang/System.out:Ljava/io/PrintStream;
   #3 = String             #25            // Usage: java HelloName <name> <number>
   #4 = Methodref          #26.#27        // java/io/PrintStream.println:(Ljava/lang/String;)V
   #5 = Methodref          #28.#29        // java/lang/Integer.parseInt:(Ljava/lang/String;)I
   #6 = Class              #30            // java/lang/NumberFormatException
   #7 = String             #31            // Error: The second argument must be a valid integer.
   #8 = InvokeDynamic      #0:#35         // #0:makeConcatWithConstants:(Ljava/lang/String;)Ljava/lang/String;
   #9 = Class              #36            // HelloName
  #10 = Class              #37            // java/lang/Object
  #11 = Utf8               <init>
  #12 = Utf8               ()V
  #13 = Utf8               Code
  #14 = Utf8               LineNumberTable
  #15 = Utf8               main
  #16 = Utf8               ([Ljava/lang/String;)V
  #17 = Utf8               StackMapTable
  #18 = Class              #38            // "[Ljava/lang/String;"
  #19 = Class              #39            // java/lang/String
  #20 = Utf8               SourceFile
  #21 = Utf8               HelloName.java
  #22 = NameAndType        #11:#12        // "<init>":()V
  #23 = Class              #40            // java/lang/System
  #24 = NameAndType        #41:#42        // out:Ljava/io/PrintStream;
  #25 = Utf8               Usage: java HelloName <name> <number>
  #26 = Class              #43            // java/io/PrintStream
  #27 = NameAndType        #44:#45        // println:(Ljava/lang/String;)V
  #28 = Class              #46            // java/lang/Integer
  #29 = NameAndType        #47:#48        // parseInt:(Ljava/lang/String;)I
  #30 = Utf8               java/lang/NumberFormatException
  #31 = Utf8               Error: The second argument must be a valid integer.
  #32 = Utf8               BootstrapMethods
  #33 = MethodHandle       6:#49          // REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithCo
/lang/invoke/MethodType;Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
  #34 = String             #50            // Hello \u0001
  #35 = NameAndType        #51:#52        // makeConcatWithConstants:(Ljava/lang/String;)Ljava/lang/String;
  #36 = Utf8               HelloName
  #37 = Utf8               java/lang/Object
  #38 = Utf8               [Ljava/lang/String;
  #39 = Utf8               java/lang/String
  #40 = Utf8               java/lang/System
  #41 = Utf8               out
  #42 = Utf8               Ljava/io/PrintStream;
  #43 = Utf8               java/io/PrintStream
  #44 = Utf8               println
  #45 = Utf8               (Ljava/lang/String;)V
  #46 = Utf8               java/lang/Integer
  #47 = Utf8               parseInt
  #48 = Utf8               (Ljava/lang/String;)I
  #49 = Methodref          #53.#54        // java/lang/invoke/StringConcatFactory.makeConcatWithConstants:(Ljava/la
odType;Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
  #50 = Utf8               Hello \u0001
  #51 = Utf8               makeConcatWithConstants
  #52 = Utf8               (Ljava/lang/String;)Ljava/lang/String;
```

```
   0: aload_0
   1: arraylength
   2: iconst_2
   3: if_icmpeq     15
   6: getstatic     #2                  // Field java/lang/Syst
   9: ldc           #3                  // String Usage: java H
  11: invokevirtual #4                  // Method java/io/Print
  14: return
  15: aload_0
  16: iconst_0
  17: aaload
  18: astore_1
  19: aload_0
  20: iconst_1
  21: aaload
  22: invokestatic  #5                  // Method java/lang/Int
  25: istore_2
  26: goto          39
  29: astore_3
  30: getstatic     #2                  // Field java/lang/Syst
  33: ldc           #7                  // String Error: The se
  35: invokevirtual #4                  // Method java/io/Print
  38: return
  39: iconst_0
  40: istore_3
  41: iload_3
  42: iload_2
  43: if_icmpge     64
  46: getstatic     #2                  // Field java/lang/Syst
  49: aload_1
  50: invokedynamic #8,  0              // InvokeDynamic #0:mak
  55: invokevirtual #4                  // Method java/io/Print
  58: iinc          3, 1
  61: goto          41
  64: return
Exception table:
   from    to  target type
    19    26    29   Class java/lang/NumberFormatException
LineNumberTable:
  line 3: 0
  line 4: 6
  line 5: 14
  line 8: 15
  line 12: 19
  line 16: 26
  line 13: 29
  line 14: 30
  line 15: 38
  line 18: 39
  line 19: 46
  line 18: 58
  line 21: 64
```

Corresponding Bytecode Analysis

Here's the relevant portion of the **disassembled bytecode** that implements the loop:

plaintext Copy Edit

```
39: iconst_0        // Push integer 0 (loop counter initialization: int i = 0)
40: istore_3        // Store it in local variable 3 (i)
41: iload_3         // Load loop counter (i)
42: iload_2         // Load max count (count)
43: if_icmpge 64    // If i >= count, jump to instruction 64 (exit loop)
46: getstatic #2    // Get System.out (for printing)
49: aload_1         // Load name
50: invokedynamic #8 // Concatenate "Hello " + name
55: invokevirtual #4 // Call println()
58: iinc 3, 1       // Increment loop counter i++
61: goto 41         // Jump back to start of loop condition check
64: return          // Exit method
```

Step-by-Step Execution

1. `iconst_0` → Pushes `0` onto the stack (initial loop counter `i = 0`).
2. `istore_3` → Stores it in **local variable 3** (`i`).
3. `iload_3` → Loads `i` from local variable 3.
4. `iload_2` → Loads `count` (number of repetitions).
5. `if_icmpge 64` → If `i >= count`, jump to **instruction 64** (exit loop).
6. `getstatic #2` → Gets `System.out` (for printing).
7. `aload_1` → Loads `name` from local variable 1.
8. `invokedynamic #8` → Uses `invokedynamic` to **concatenate** `"Hello " + name` at runtime.
9. `invokevirtual #4` → Calls `println()` to print the result.
10. `iinc 3, 1` → Increments `i` (`i++`).
11. `goto 41` → Jumps back to instruction `41` (loop condition check).
12. If `i < count`, repeat steps **3-11**.
13. If `i >= count`, jump to `64` and exit the loop.