luminis
Conversing worlds

What you didn't know you wanted to know about…

the Java Virtual Machine

Angelo van der Sijpt
@_angelos

https://github.com/angelos/wydkywtkatjvm

Angelo van der Sijpt

Fellow

Luminis Eindhoven

Krav maga

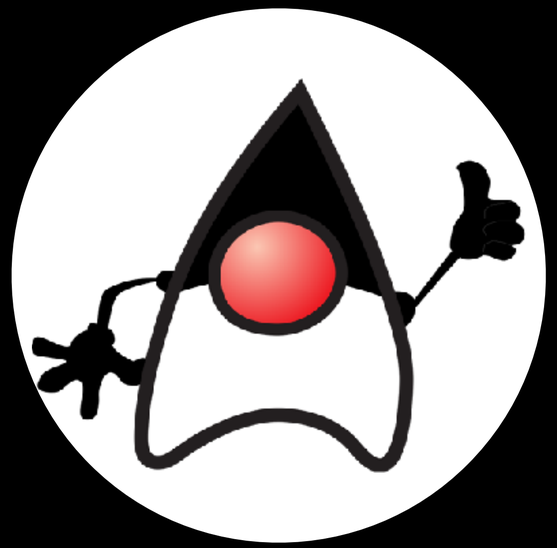
Running

OCR

@_angelos

luminis

Conversing worlds



JDK

JRE

Oracle JVM

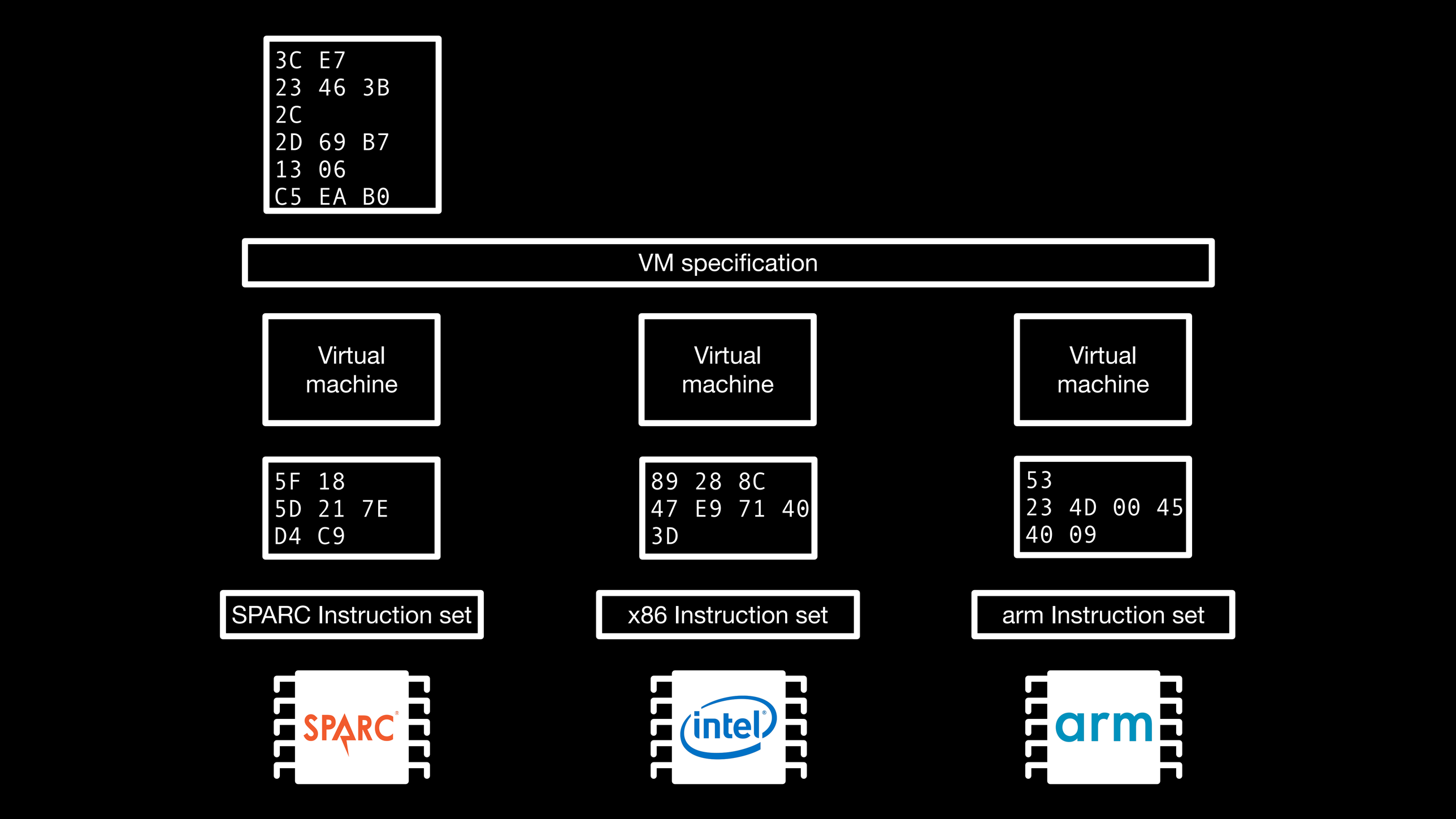
Class library
`rt.jar`

Compiler &
dev tools

Java®
Virtual
Machine
Specification

Java®
SE APIs

Java®
Language
Specification


3C E7
23 46 3B
2C
2D 69 B7
13 06
C5 EA B0
VM specification
Virtual machine
Virtual machine
Virtual machine
5F 18
5D 21 7E
D4 C9
89 28 8C
47 E9 71 40
3D
53
23 4D 00 45
40 09
SPARC Instruction set
x86 Instruction set
arm Instruction set
SPARC
intel
arm

bytecode & VM operation

garbage collection

optimization

Dalvik

!

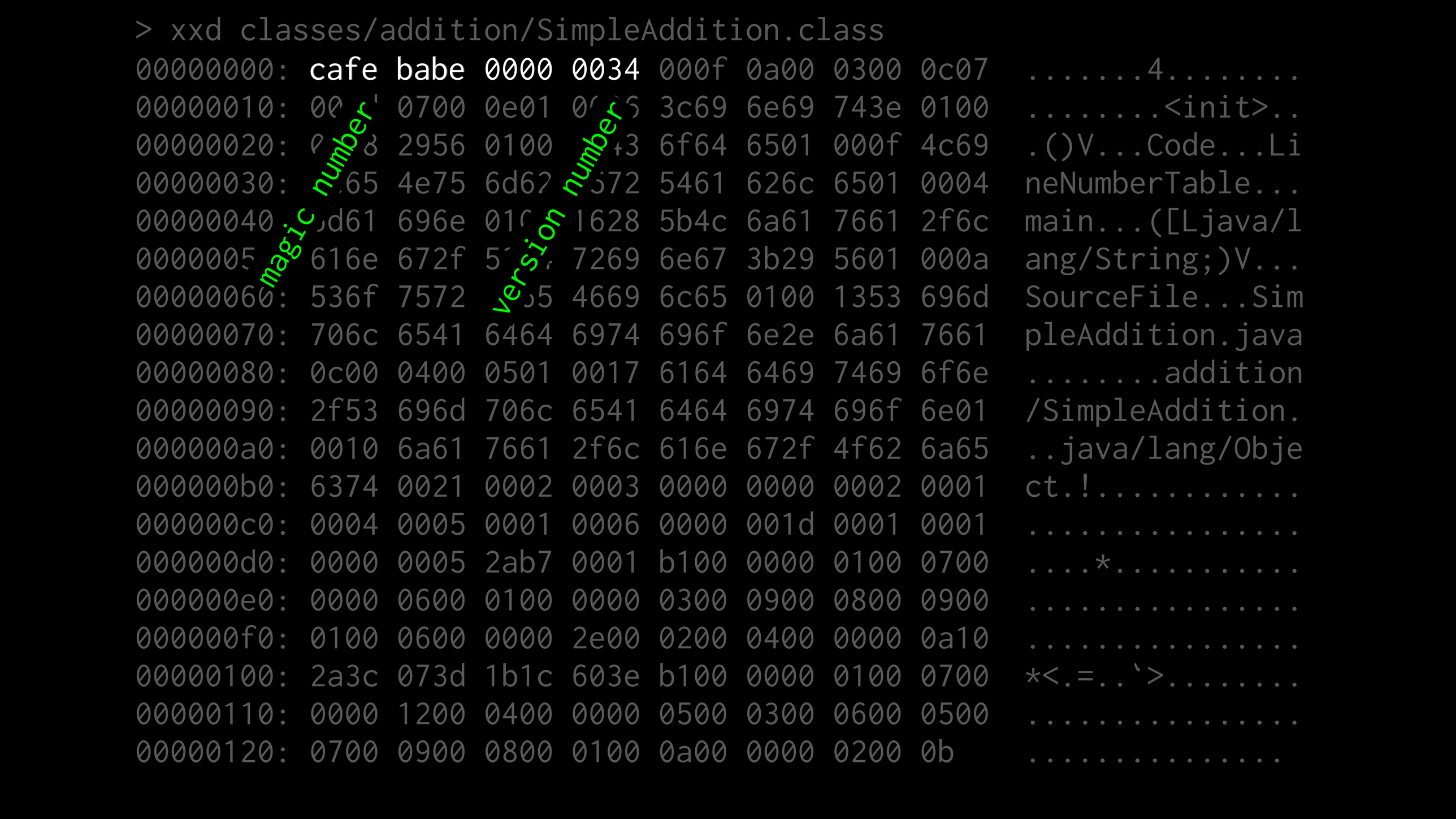
```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = a + b;
    }
}
```

```
SimpleAddition
  |-addition
  |  |-SimpleAddition.java
```

```
> javac -d classes addition/SimpleAddition.java
```

```
classes
 |-addition
 |  |-SimpleAddition.class
```

```
> xxd classes/addition/SimpleAddition.class
00000000: cafe babe 0000 0034 000f 0a00 0300 0c07  .......4........
00000010: 000d 0700 0e01 0006 3c69 6e69 743e 0100  ........<init>..
00000020: 0328 2956 0100 0443 6f64 6501 000f 4c69  .()V...Code...Li
00000030: 6e65 4e75 6d62 6572 5461 626c 6501 0004  neNumberTable...
00000040: 6d61 696e 0100 1628 5b4c 6a61 7661 2f6c  main...([Ljava/l
00000050: 616e 672f 5374 7269 6e67 3b29 5601 000a  ang/String;)V...
00000060: 536f 7572 6365 4669 6c65 0100 1353 696d  SourceFile...Sim
00000070: 706c 6541 6464 6974 696f 6e2e 6a61 7661  pleAddition.java
00000080: 0c00 0400 0501 0017 6164 6469 7469 6f6e  ........addition
00000090: 2f53 696d 706c 6541 6464 6974 696f 6e01  /SimpleAddition.
000000a0: 0010 6a61 7661 2f6c 616e 672f 4f62 6a65  ..java/lang/Obje
000000b0: 6374 0021 0002 0003 0000 0000 0002 0001  ct.!............
000000c0: 0004 0005 0001 0006 0000 001d 0001 0001  ................
000000d0: 0000 0005 2ab7 0001 b100 0000 0100 0700  ....*...........
000000e0: 0000 0600 0100 0000 0300 0900 0800 0900  ................
000000f0: 0100 0600 0000 2e00 0200 0400 0000 0a10  ................
00000100: 2a3c 073d 1b1c 603e b100 0000 0100 0700  *<.=..`>........
00000110: 0000 1200 0400 0000 0500 0300 0600 0500  ................
00000120: 0700 0900 0800 0100 0a00 0000 0200 0b    ..............
```


> xxd classes/addition/SimpleAddition.class
00000000: cafe babe 0000 0034 000f 0a00 0300 0c07 ....4........
00000010: 0[illegible] 0700 0e01 0[illegible] 3c69 6e69 743e 0100 ....<init>..
00000020: [illegible] 2956 0100 [illegible] 6f64 6501 000f 4c69 .()V...Code...Li
00000030: [illegible] 4e75 6d62 [illegible] 5461 626c 6501 0004 neNumberTable...
00000040: [illegible] 6e01 [illegible] 5b4c 6a61 7661 2f6c main...([Ljava/l
00000050: 616e 672f 5[illegible] 7269 6e67 3b29 5601 000a ang/String;)V...
00000060: 536f 7572 [illegible] 4669 6c65 0100 1353 696d SourceFile...Sim
00000070: 706c 6541 6464 6974 696f 6e2e 6a61 7661 pleAddition.java
00000080: 0c00 0400 0501 0017 6164 6469 7469 6f6e ....addition
00000090: 2f53 696d 706c 6541 6464 6974 696f 6e01 /SimpleAddition.
000000a0: 0010 6a61 7661 2f6c 616e 672f 4f62 6a65 ..java/lang/Obje
000000b0: 6374 0021 0002 0003 0000 0000 0002 0001 ct.!............
000000c0: 0004 0005 0001 0006 0000 001d 0001 0001 ....
000000d0: 0000 0005 2ab7 0001 b100 0000 0100 0700 ....*...........
000000e0: 0000 0600 0100 0000 0300 0900 0800 0900 ....
000000f0: 0100 0600 0000 2e00 0200 0400 0000 0a10 ....
00000100: 2a3c 073d 1b1c 603e b100 0000 0100 0700 *<.=..`>........
00000110: 0000 1200 0400 0000 0500 0300 0600 0500 ....
00000120: 0700 0900 0800 0100 0a00 0000 0200 0b   ....
magic number
version number

```
> xxd classes/addition/SimpleAddition.class
00000000: cafe babe 0000 0034 000f 0a00 0300 0c07  .......4........
                                6e69 743e 0100  ........<init>..
                                6501 000f 4c69  .()V...Code...Li
                                626c 6501 0004  neNumberTable...
                                6a61 7661 2f6c  main...([Ljava/l
                                3b29 5601 000a  ang/String;)V...
                                0100 1353 696d  SourceFile...Sim
                                6e2e 6a61 7661  pleAddition.java
00000080: 0c00 0400 0501 0017 6164 6469 7469 6f6e  ........addition
00000090: 2f53 696d 706c 6541 6464 6974 696f 6e01  /SimpleAddition.
000000a0: 0010 6a61 7661 2f6c 616e 672f 4f62 6a65  ..java/lang/Obje
000000b0: 6374 0021 0002 0003 0000 0000 0002 0001  ct.!............
000000c0: 0004 0005 0001 0006 0000 001d 0001 0001  ................
000000d0: 0000 0005 2ab7 0001 b100 0000 0100 0700  ....*...........
000000e0: 0000 0600 0100 0000 0300 0900 0800 0900  ................
000000f0: 0100 0600 0000 2e00 0200 0400 0000 0a10  ................
00000100: 2a3c 073d 1b1c 603e b100 0000 0100 0700  *<.=..`>........
00000110: 0000 1200 0400 0000 0500 0300 0600 0500  ................
00000120: 0700 0900 0800 0100 0a00 0000 0200 0b    ..............
```

```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = a + b;
    }
}
```

0005 2ab7

```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
        int a = 42; // 0x2A in hex
        int b = 4;
        int result = a + b;
    }
}
```

0a10 2a3c

0005 2ab7

0a10 2a3c

Constants

00 (0x00)	*nop*
01 (0x01)	*aconst_null*
02 (0x02)	*iconst_m1*
03 (0x03)	*iconst_0*
04 (0x04)	*iconst_1*
05 (0x05)	*iconst_2*
06 (0x06)	*iconst_3*
07 (0x07)	*iconst_4*
08 (0x08)	*iconst_5*
09 (0x09)	*lconst_0*
10 (0x0a)	*lconst_1*
11 (0x0b)	*fconst_0*
12 (0x0c)	*fconst_1*
13 (0x0d)	*fconst_2*
14 (0x0e)	*dconst_0*
15 (0x0f)	*dconst_1*
16 (0x10)	*bipush*
17 (0x11)	*sipush*
18 (0x12)	*ldc*
19 (0x13)	*ldc_w*
20 (0x14)	*ldc2_w*

Loads

21 (0x15)	*iload*
22 (0x16)	*lload*
23 (0x17)	*fload*
24 (0x18)	*dload*
25 (0x19)	*aload*
26 (0x1a)	*iload_0*
27 (0x1b)	*iload_1*
28 (0x1c)	*iload_2*
29 (0x1d)	*iload_3*
30 (0x1e)	*lload_0*
31 (0x1f)	*lload_1*
32 (0x20)	*lload_2*
33 (0x21)	*lload_3*
34 (0x22)	*fload_0*
35 (0x23)	*fload_1*
36 (0x24)	*fload_2*
37 (0x25)	*fload_3*
38 (0x26)	*dload_0*
39 (0x27)	*dload_1*
40 (0x28)	*dload_2*
41 (0x29)	*dload_3*
42 (0x2a)	*aload_0*
43 (0x2b)	*aload_1*
44 (0x2c)	*aload_2*
45 (0x2d)	*aload_3*
46 (0x2e)	*iaload*

Stores

54 (0x36)	*istore*
55 (0x37)	*lstore*
56 (0x38)	*fstore*
57 (0x39)	*dstore*
58 (0x3a)	*astore*
59 (0x3b)	*istore_0*
60 (0x3c)	*istore_1*
61 (0x3d)	*istore_2*
62 (0x3e)	*istore_3*
63 (0x3f)	*lstore_0*
64 (0x40)	*lstore_1*
65 (0x41)	*lstore_2*
66 (0x42)	*lstore_3*
67 (0x43)	*fstore_0*
68 (0x44)	*fstore_1*
69 (0x45)	*fstore_2*
70 (0x46)	*fstore_3*
71 (0x47)	*dstore_0*
72 (0x48)	*dstore_1*
73 (0x49)	*dstore_2*
74 (0x4a)	*dstore_3*
75 (0x4b)	*astore_0*
76 (0x4c)	*astore_1*
77 (0x4d)	*astore_2*
78 (0x4e)	*astore_3*
79 (0x4f)	*iastore*

0005 2ab7

0a10 2a3c

Constants		Loads		Stores	
00 (0x00)	*nop*	21 (0x15)	*iload*	54 (0x36)	*istore*
01 (0x01)	*aconst_null*	22 (0x16)	*lload*	55 (0x37)	*lstore*
02 (0x02)	*iconst_m1*	23 (0x17)	*fload*	56 (0x38)	*fstore*

bipush *bipush*

Operation Push `byte`

Format

bipush
byte

Forms *bipush* = 16 (0x10)

Operand Stack ... →

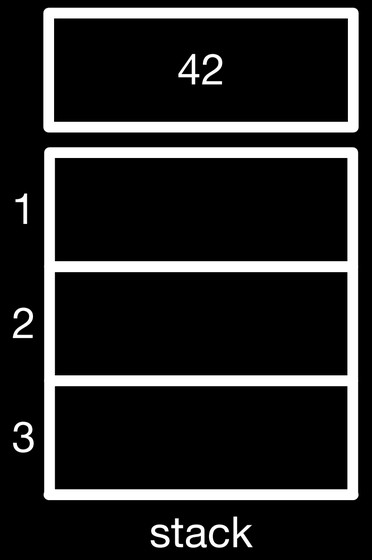
..., *value*

Description The immediate *byte* is sign-extended to an `int` *value*. That *value* is pushed onto the operand stack.

```
> javap -c -p -classpath classes addition/SimpleAddition
Compiled from "SimpleAddition.java"
public class addition.SimpleAddition {
  public addition.SimpleAddition();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: iadd
       8: istore_3
       9: return
}
```

```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```

```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```



bipush ***bipush***

Operation Push `byte`

Format

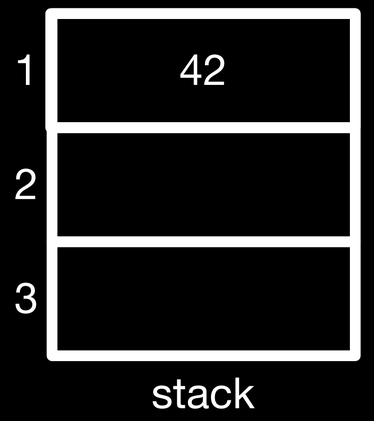
bipush
byte

Forms *bipush* = 16 (0x10)

Operand Stack ... →
..., *value*

Description The immediate *byte* is sign-extended to an `int` *value*. That *value* is pushed onto the operand stack.

```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
```



istore_<n> *istore_<n>*

Operation Store `int` into local variable

Format *istore_<n>*

Forms *istore_0* = 59 (0x3b)
istore_1 = 60 (0x3c)
istore_2 = 61 (0x3d)
istore_3 = 62 (0x3e)

Operand Stack ..., *value* →
...

Description The *<n>* must be an index into the local variable array of the current frame (§2.6). The *value* on the top of the operand stack must be of type `int`. It is popped from the operand stack, and the value of the local variable at *<n>* is set to *value*.

Notes Each of the *istore_<n>* instructions is the same as *istore* with an *index* of *<n>*, except that the operand *<n>* is implicit.

```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
```


iconst_<i> *iconst_<i>*

Operation Push `int` constant

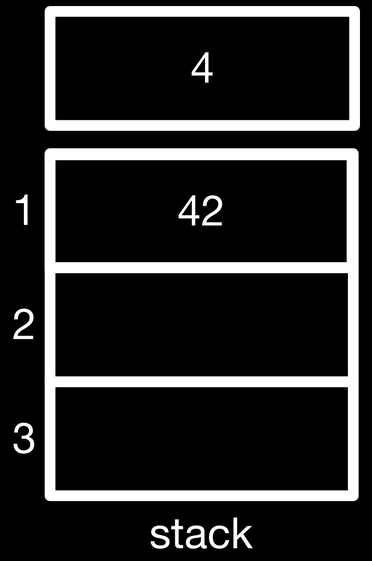
Format *iconst_<i>*

Forms
iconst_m1 = 2 (0x2)
iconst_0 = 3 (0x3)
iconst_1 = 4 (0x4)
iconst_2 = 5 (0x5)
iconst_3 = 6 (0x6)
iconst_4 = 7 (0x7)
iconst_5 = 8 (0x8)

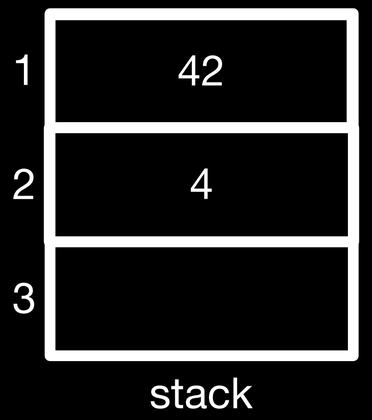
Operand Stack ... →
..., <i>

Description Push the `int` constant <i> (-1, 0, 1, 2, 3, 4 or 5) onto the operand stack.

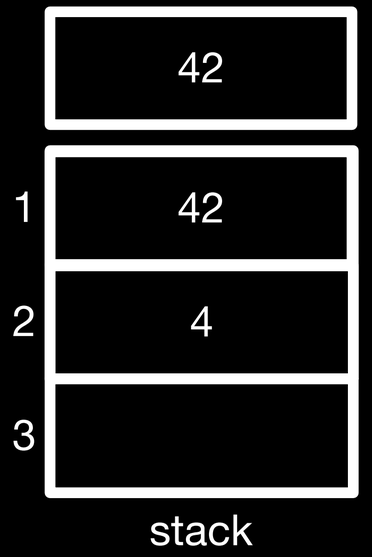
Notes Each of this family of instructions is equivalent to *bipush* <i> for



```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```



```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
```



iload_<n> *iload_<n>*

Operation Load `int` from local variable

Format *iload_<n>*

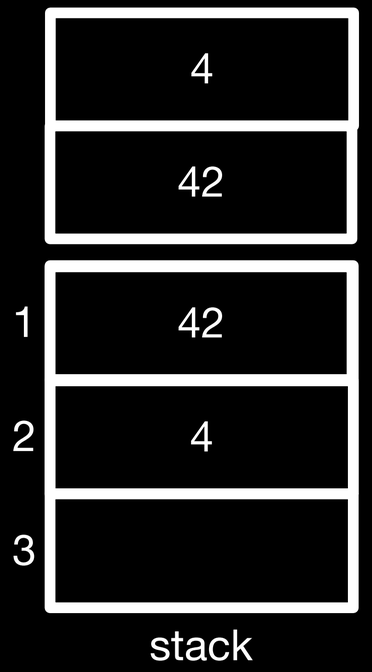
Forms *iload_0* = 26 (0x1a)
iload_1 = 27 (0x1b)
iload_2 = 28 (0x1c)
iload_3 = 29 (0x1d)

Operand Stack ... →
..., *value*

Description The *<n>* must be an index into the local variable array of the current frame (§2.6). The local variable at *<n>* must contain an `int`. The *value* of the local variable at *<n>* is pushed onto the operand stack.

Notes Each of the *iload_<n>* instructions is the same as *iload* with an *index* of *<n>*, except that the operand *<n>* is implicit.

```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```



```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
```


iadd *iadd*

Operation Add `int`

Format *iadd*

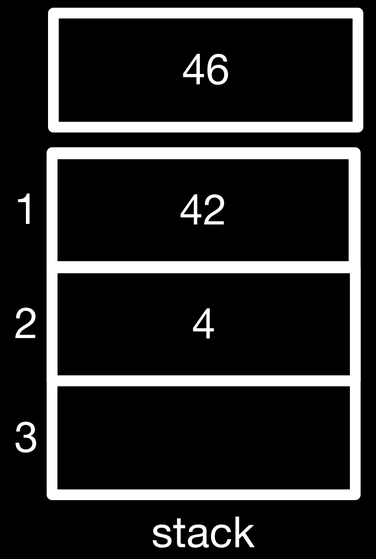
Forms *iadd* = 96 (0x60)

Operand Stack ..., *value1*, *value2* →
..., *result*

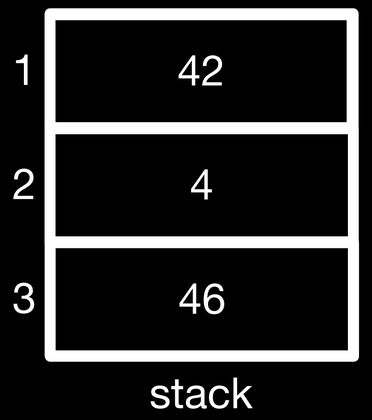
Description Both *value1* and *value2* must be of type `int`. The values are popped from the operand stack. The `int` *result* is *value1* + *value2*. The *result* is pushed onto the operand stack.

The result is the 32 low-order bits of the true mathematical result in a sufficiently wide two's-complement format, represented as a value of type `int`. If overflow occurs, then the sign of the result may not be the same as the sign of the mathematical sum of the two values.

Despite the fact that overflow may occur, execution of an *iadd* instruction never throws a run-time exception.



```
10 2a 3c 07 3d 1b 1c 60 3e b1

0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```



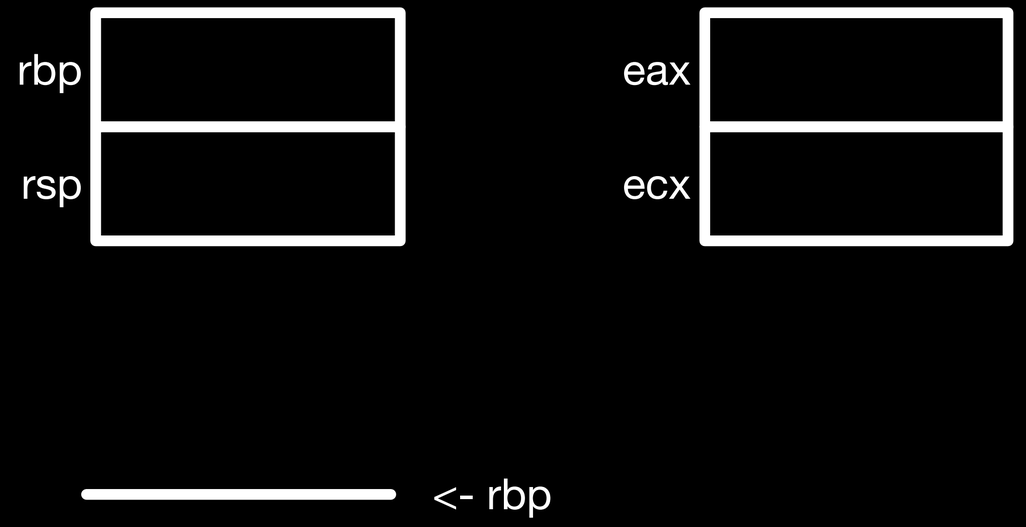
```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = a + b;
    }
}
```

```c
#include <stdio.h>

int main() {
  int a = 42;
  int b = 17;
  int result = a + b;
}
```

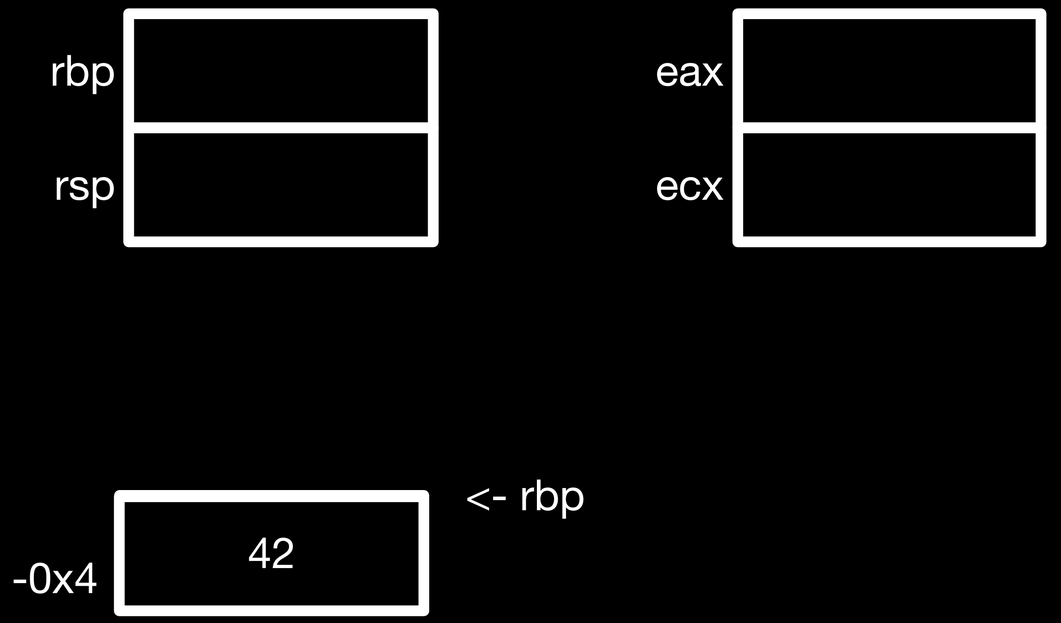
```
0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```

```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

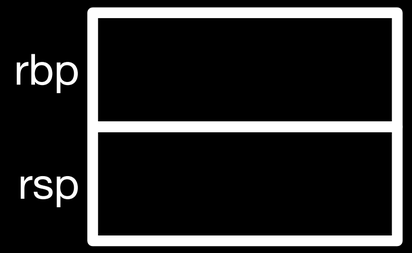
```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

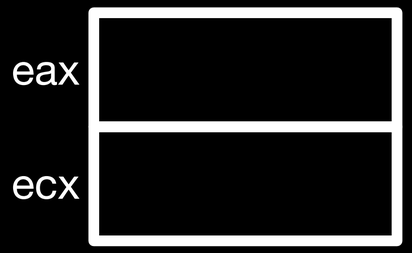


```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

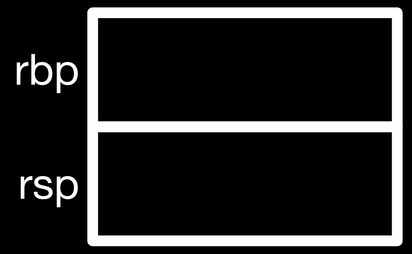


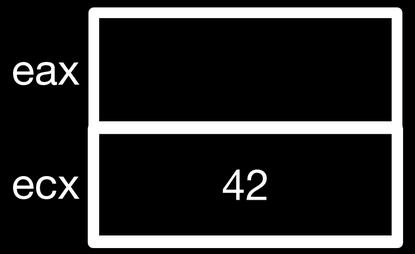
```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

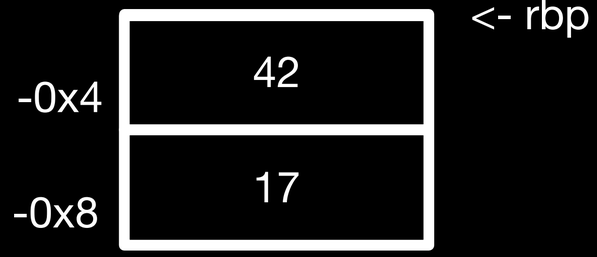




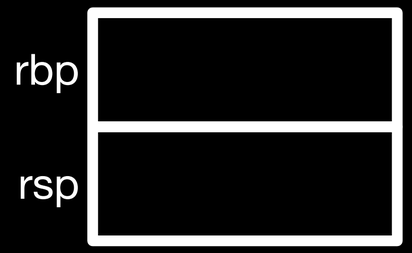
```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

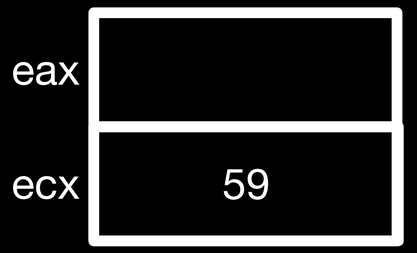


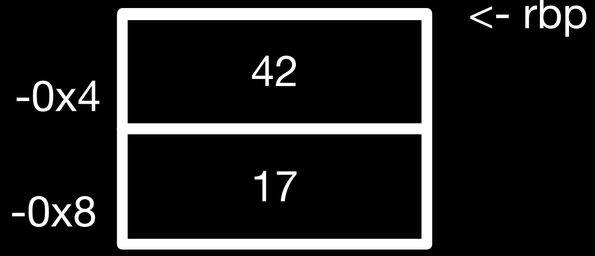




```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```







```
0: bipush        42
2: istore_1
3: iconst_4
4: istore_2
5: iload_1
6: iload_2
7: iadd
8: istore_3
9: return
```

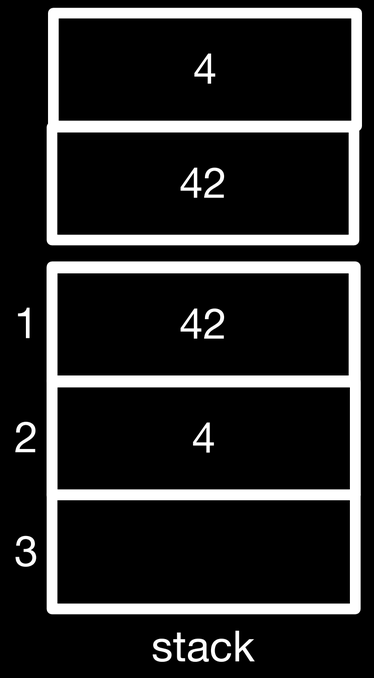
stack based

```
0000000100000f90  pushq %rbp
0000000100000f91  movq  %rsp, %rbp
0000000100000f94  xorl  %eax, %eax
0000000100000f96  movl  $0x2a, -0x4(%rbp)
0000000100000f9d  movl  $0x11, -0x8(%rbp)
0000000100000fa4  movl  -0x4(%rbp), %ecx
0000000100000fa7  addl  -0x8(%rbp), %ecx
0000000100000faa  movl  %ecx, -0xc(%rbp)
0000000100000fad  popq  %rbp
0000000100000fae  retq
```

register based

```java
public class CallAMethod {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = add(a, b);
    }

    private static int add(int a, int b) {
      return a + b;
    }
}
```

```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```



```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
```


invokestatic ***invokestatic***

Operation Invoke a class (`static`) method

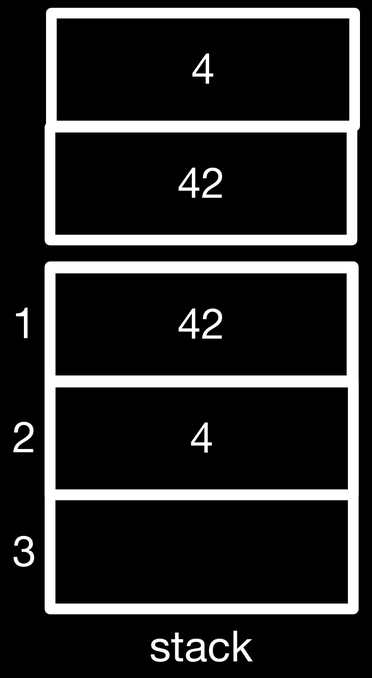
Format

invokestatic
indexbyte1
indexbyte2

Forms *invokestatic* = 184 (0xb8)

Operand Stack ..., [*arg1*, [*arg2* ...]] → ...

Description The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at that index must be a symbolic reference to a method or an interface method (§5.1), which gives the name and descriptor (§4.3.3) of the method or interface method as well as a symbolic reference to the class or interface in which



```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```

```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```

```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```

```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
```


ireturn ***ireturn***

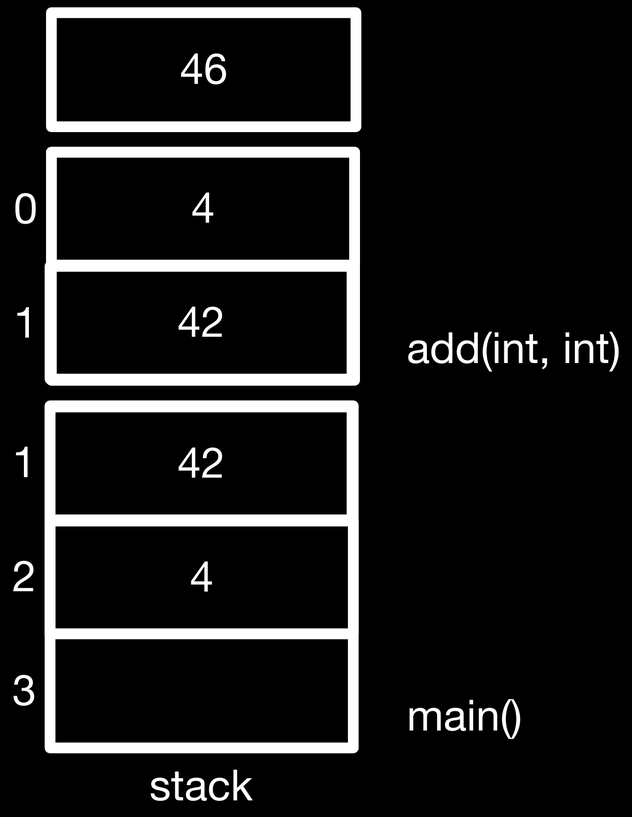
Operation Return `int` from method

Format | *ireturn* |

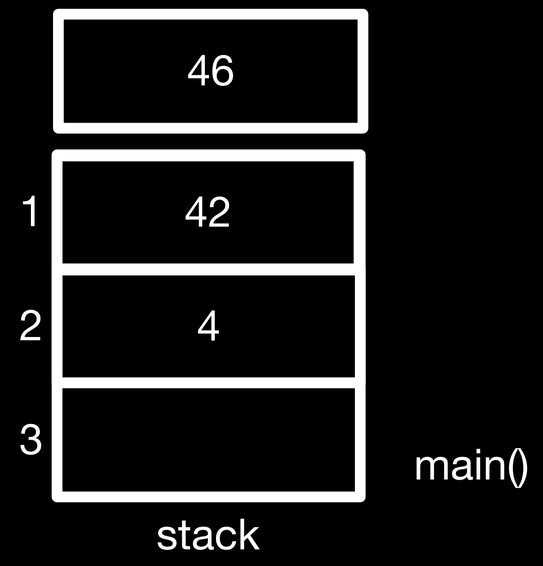
Forms *ireturn* = 172 (0xac)

Operand Stack ..., *value* →
[empty]

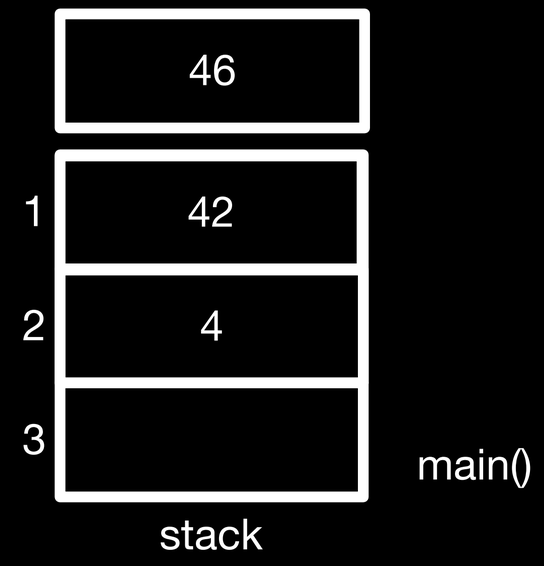
Description The current method must have return type `boolean`, `byte`, `char`, `short`, or `int`. The *value* must be of type `int`. If the current method is a `synchronized` method, the monitor entered or reentered on invocation of the method is updated and possibly exited as if by execution of a *monitorexit* instruction (§*monitorexit*) in the current thread. If no exception is thrown, *value* is popped from the operand stack of the current frame (§2.6) and pushed onto the operand stack of the frame of the invoker. Any other values on the operand stack of the current method are discarded.



```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```



```
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        42
       2: istore_1
       3: iconst_4
       4: istore_2
       5: iload_1
       6: iload_2
       7: invokestatic  #2
      10: istore_3
      11: return

  private static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn
}
```



```java
public class Objects {
    public static void main(String[] args) {
      Object o = new Object();
      o.toString();
    }
}
```

```
> javap -c -p -classpath classes Objects
public class Objects {
  public Objects();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: new           #2                  // class java/lang/Object
       3: dup
       4: invokespecial #1                  // Method java/lang/Object."<init>":()V
       7: astore_1
       8: aload_1
       9: invokevirtual #3                  // Method java/lang/Object.toString:
                                            // ()Ljava/lang/String;
      12: pop
      13: return
}
```

```
0: new           #2                  // class java/lang/Object
                                     java/lang/Object."<init>":()V
                                     java/lang/Object.toString:
                                     lang/String;
```


new *new*

Operation Create new object

Format

new
indexbyte1
indexbyte2

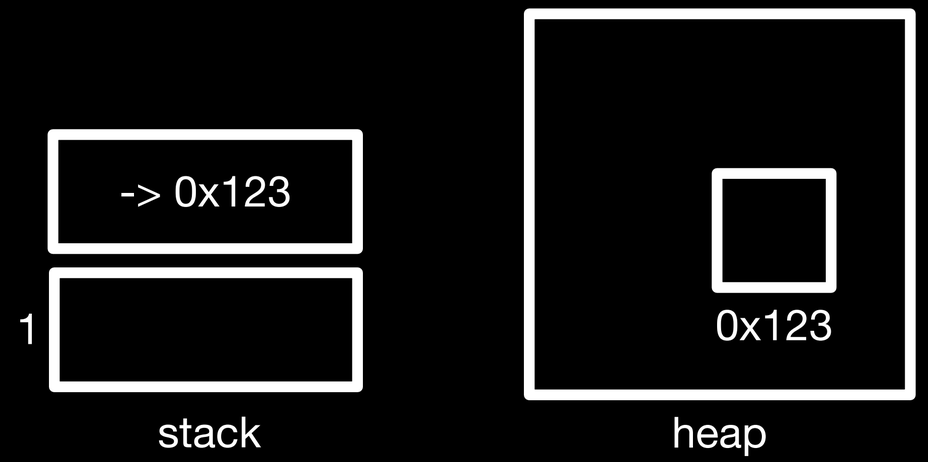
Forms *new* = 187 (0xbb)

Operand Stack ... → ..., *objectref*

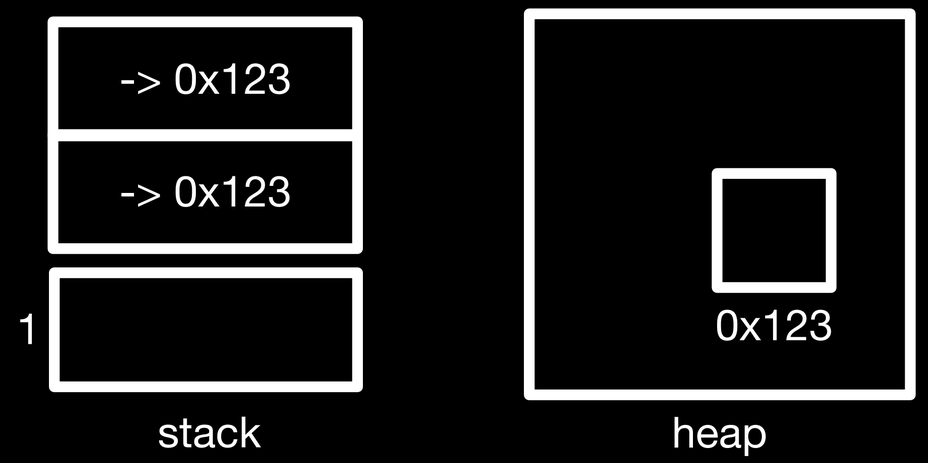
Description The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at the index must be a symbolic reference to a class or interface type. The named class or interface type is resolved (§5.4.3.1) and should result in a class type. Memory for a new instance of that class is allocated from the garbage-collected heap, and the instance variables of the new object are initialized to their default initial values (§2.3, §2.4). The *objectref*, a `reference` to the instance, is pushed onto the operand stack.

On successful resolution of the class, it is initialized if it has not already been initialized (§5.5).

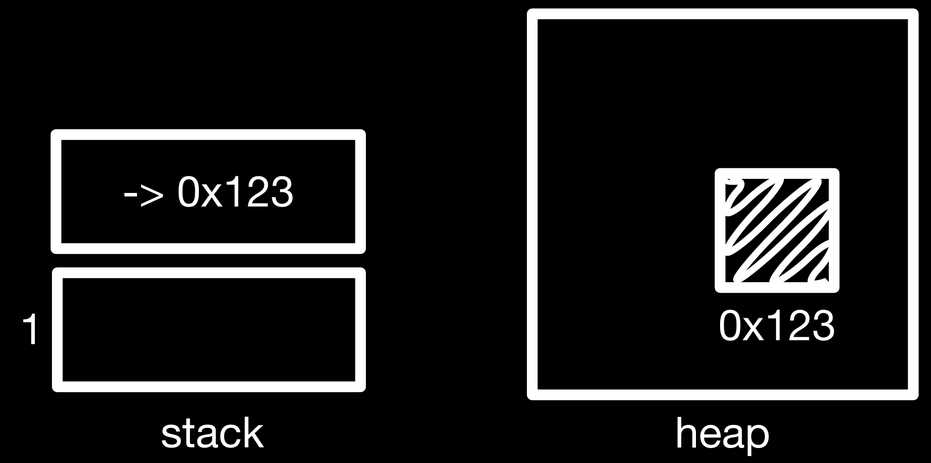
Linking During resolution of the symbolic reference to the class or



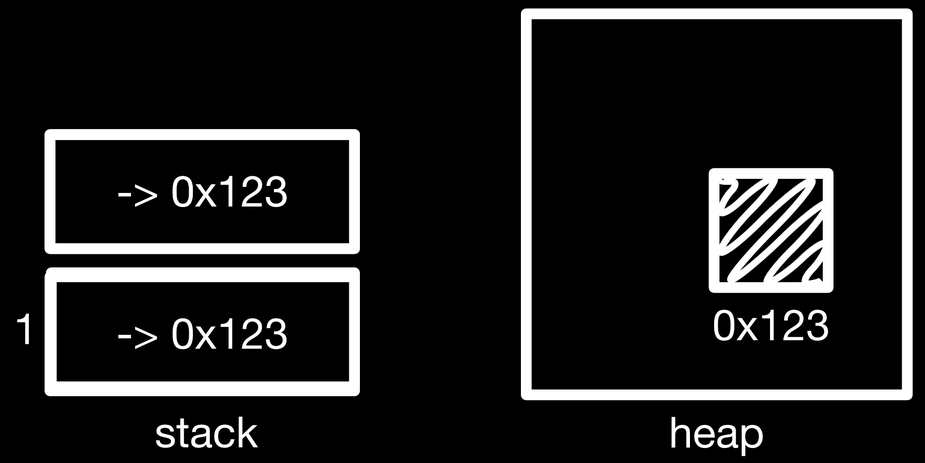
```
 0: new           #2                  // class java/lang/Object
 3: dup
 4: invokespecial #1                  // Method java/lang/Object."<init>":()V
 7: astore_1
 8: aload_1
 9: invokevirtual #3                  // Method java/lang/Object.toString:
                                      // ()Ljava/lang/String;
12: pop
13: return
```



```
 0: new           #2                  // class java/lang/Object
 3: dup
 4: invokespecial #1                  // Method java/lang/Object."<init>":()V
 7: astore_1
 8: aload_1
 9: invokevirtual #3                  // Method java/lang/Object.toString:
                                      // ()Ljava/lang/String;
12: pop
13: return
```

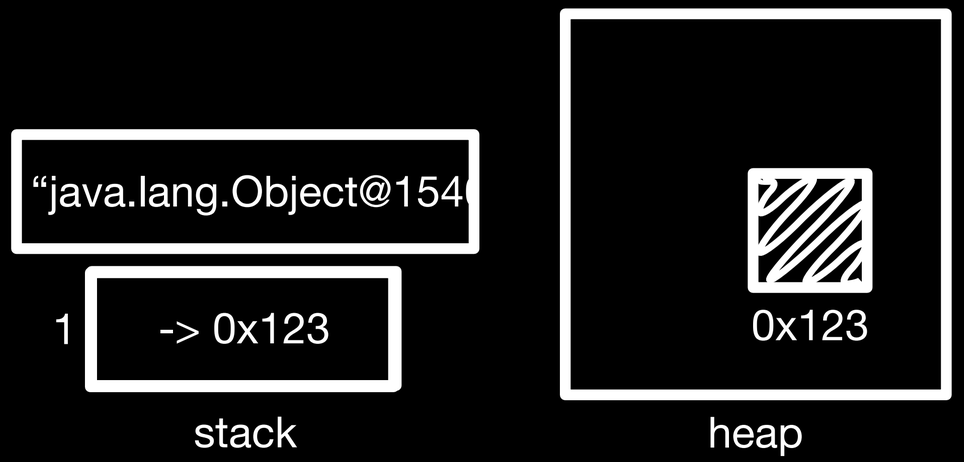


```
 0: new           #2                  // class java/lang/Object
 3: dup
 4: invokespecial #1                  // Method java/lang/Object."<init>":()V
 7: astore_1
 8: aload_1
 9: invokevirtual #3                  // Method java/lang/Object.toString:
                                      // ()Ljava/lang/String;
12: pop
13: return
```



```
 0: new           #2                  // class java/lang/Object
 3: dup
 4: invokespecial #1                  // Method java/lang/Object."<init>":()V
 7: astore_1
 8: aload_1
 9: invokevirtual #3                  // Method java/lang/Object.toString:
                                      // ()Ljava/lang/String;
12: pop
13: return
```

!



```
1 interface Iface {
2     void m();
3 }
```

```
1 class Impl implements Iface {
2   public void m() {
3     //
4   }
5 }
```

```
1 public class Interfaces {
2     public static void main(String[] args) {
3       Iface i = new Impl();
4       i.m();
5   }
6 }
```

```
interface Iface {
  public abstract void m();
}
```

```
class Impl implements Iface {
//
  public void m();
    Code:
       0: return
}
```

```
class Interfaces {
//
  public static void main(java.lang.String[]);
    Code:
       0: new           #2                  // class Impl
       3: dup
       4: invokespecial #3                  // Method Impl."<init>":()V
       7: astore_1
       8: aload_1
       9: invokeinterface #4,  1            // InterfaceMethod Iface.m:()V
      14: return
}
```

```
interface Iface {
  public abstract void m();
}
```

```
class Impl implements Iface {
//
  public void m();
    Code:
       0: return
}
```

```
class Interfaces {
//
  public static void main(java.lang.String[]);
    Code:
       0: new           #2                  // class Impl
       3: dup
       4: invokespecial #3                  // Method Impl."<init>":()V
       7: astore_1
       8: aload_1
       9: invokeinterface #4,  1            // InterfaceMethod Iface.m:()V
      14: return
}
```

!

JVM

java to bytecode

stack based instruction set

allocation & invocation

…and more!
- concurrency primitives
- instanceof
- String, arrays

Joe Mnemonic
JVM Jockey

rarely worries about memory allocation

all JVM-based languages use the same (high-level) plumbing

can read a little bytecode
—"just like Tank in The Matrix!"

new *new*

Operation Create new object

Format

new
indexbyte1
indexbyte2

Forms *new* = 187 (0xbb)

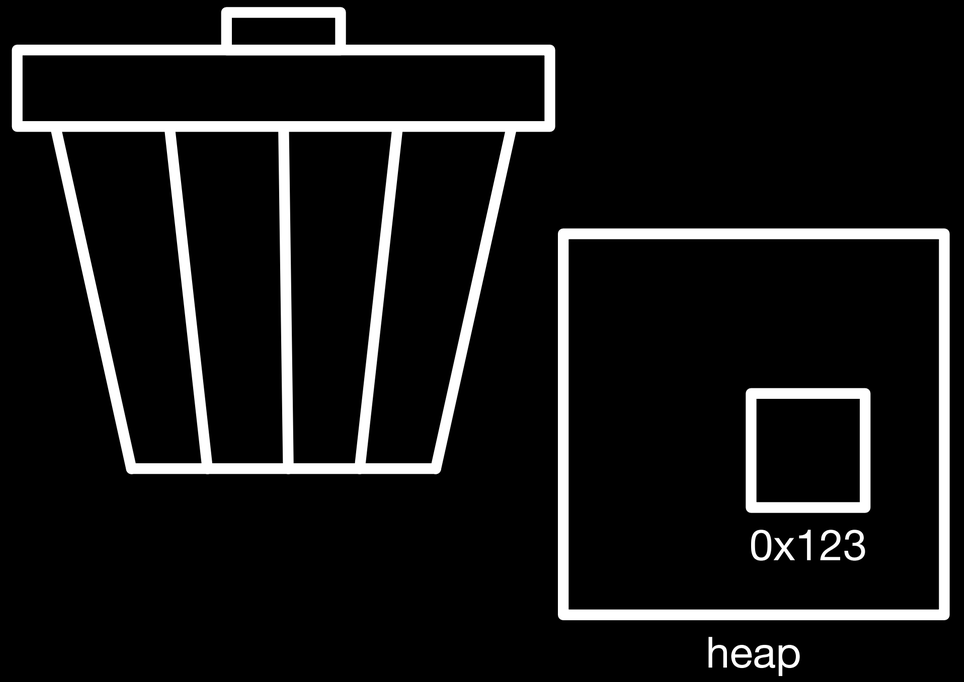
Operand Stack ... →

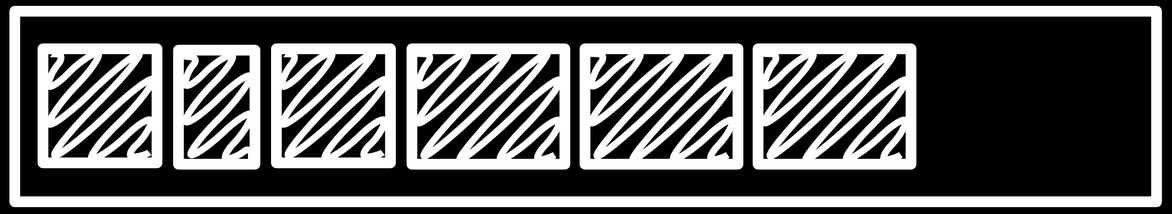
..., *objectref*

Description The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at the index must be a symbolic reference to a class or interface type. The named class or interface type is resolved (§5.4.3.1) and should result in a class type. Memory for a new instance of that class is allocated from the garbage-collected heap, and the instance variables of the new object are initialized to their default initial values (§2.3, §2.4). The *objectref*, a `reference` to the instance, is pushed onto the operand stack.

On successful resolution of the class, it is initialized if it has not already been initialized (§5.5).

Linking During resolution of the symbolic reference to the class or

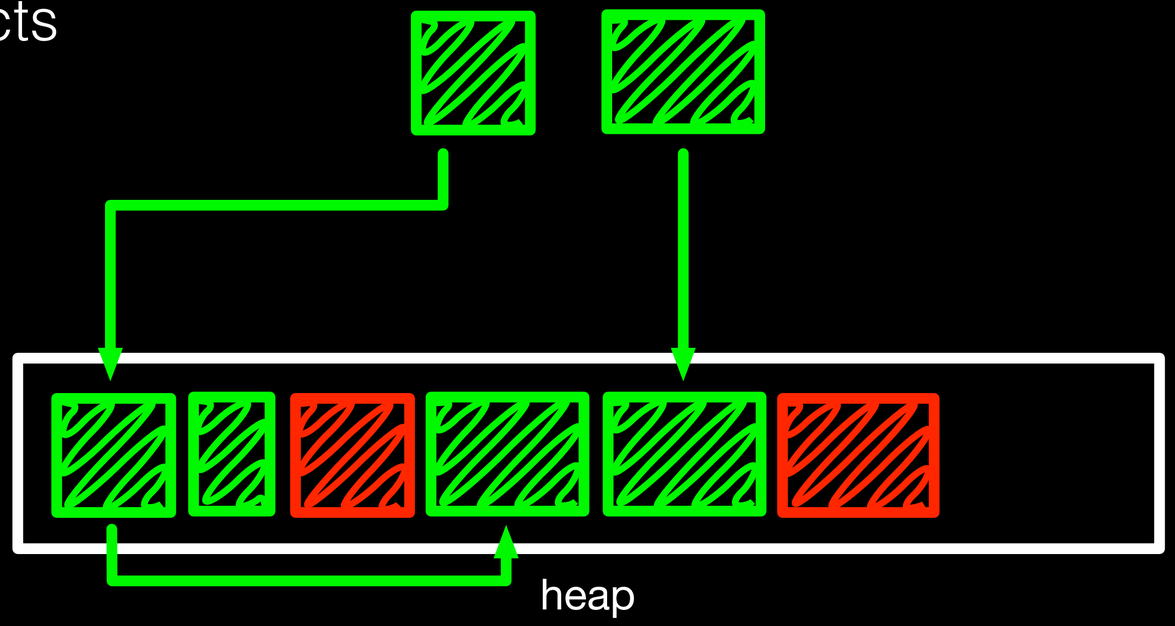


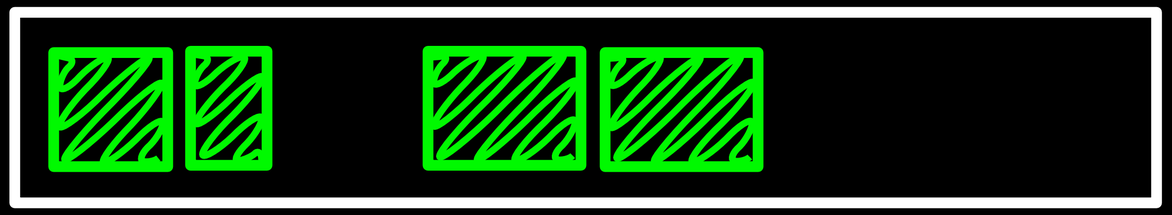


heap

garbage collection roots

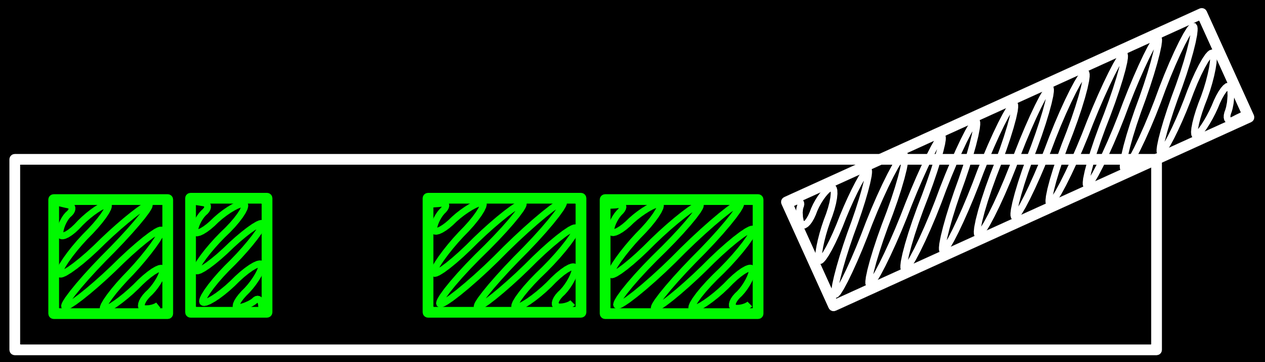
- local variables
- active Thread objects
- static variables
- JNI objects



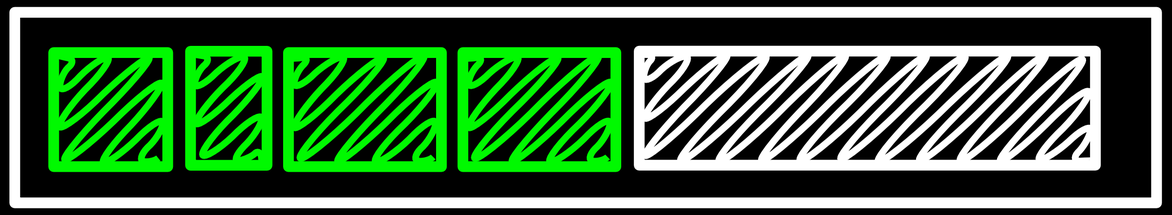


heap

“mark-and-sweep”

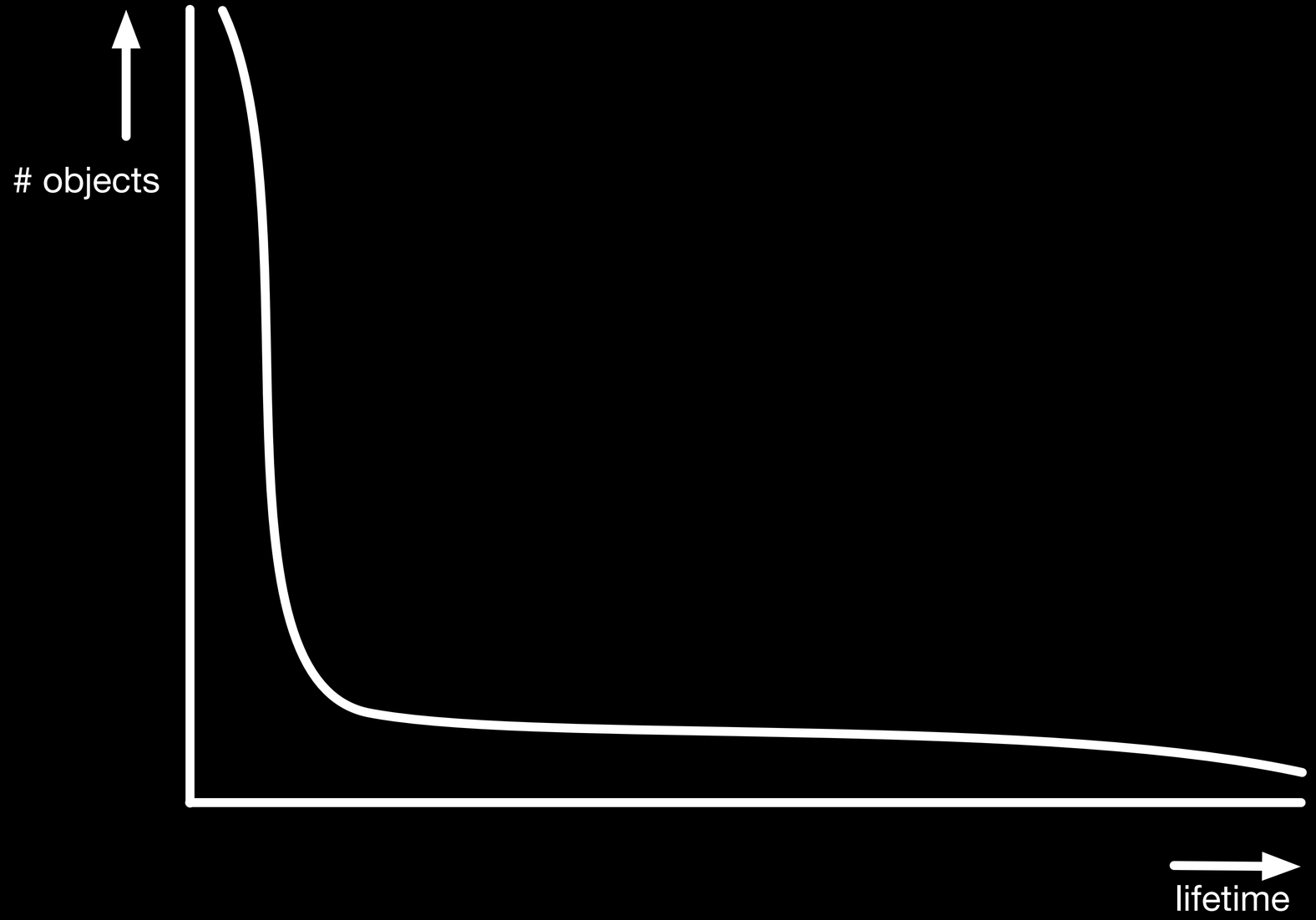


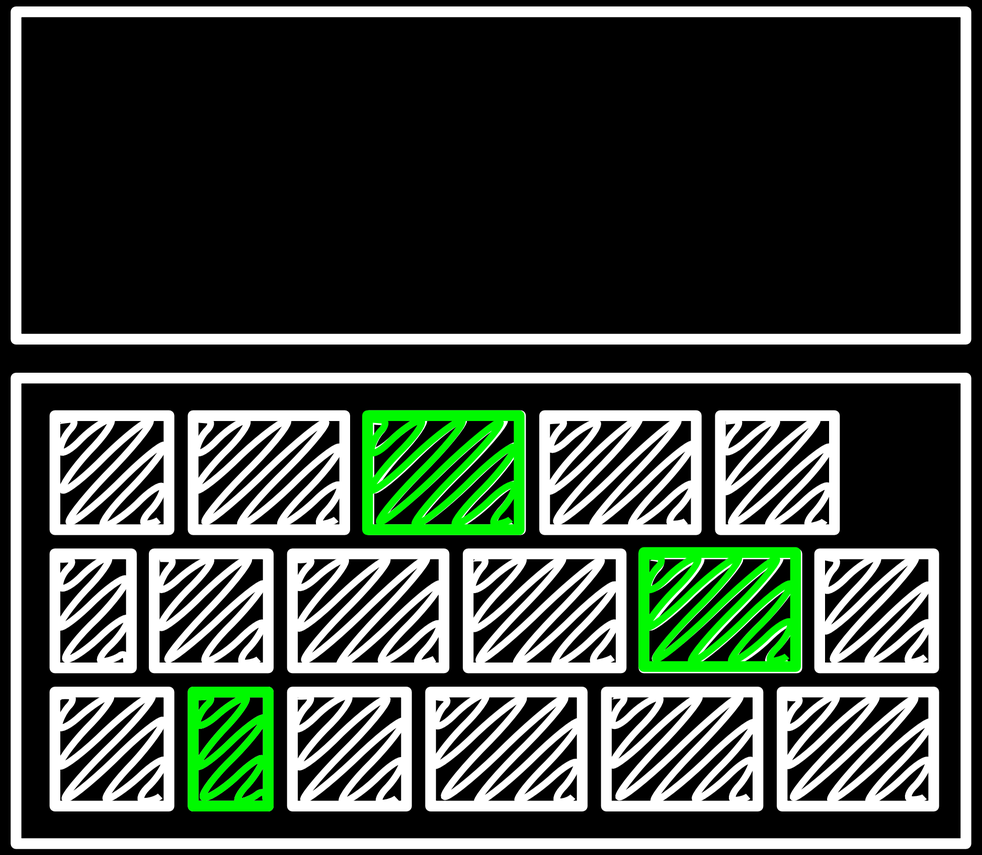
heap

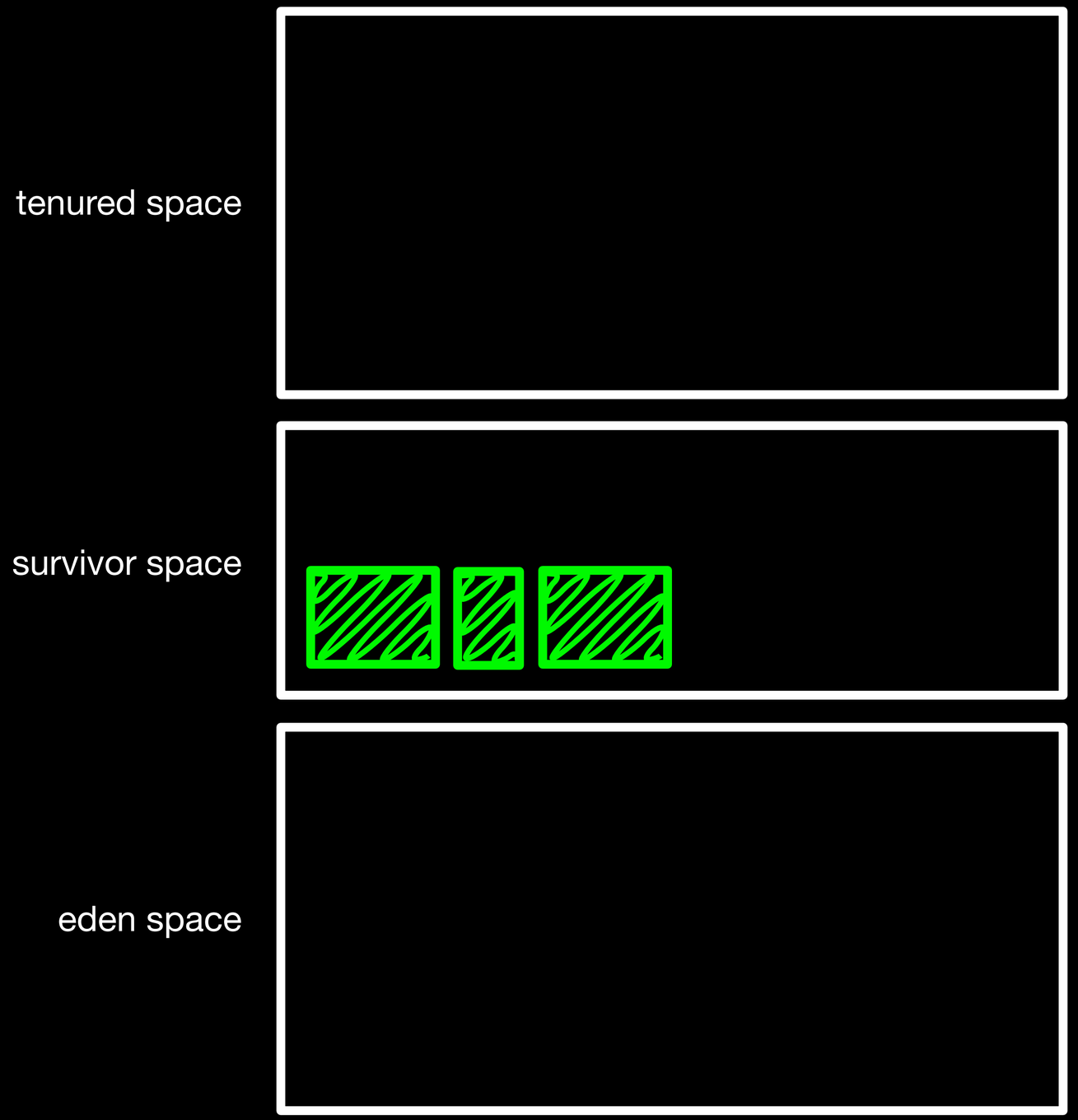


heap

“compaction”

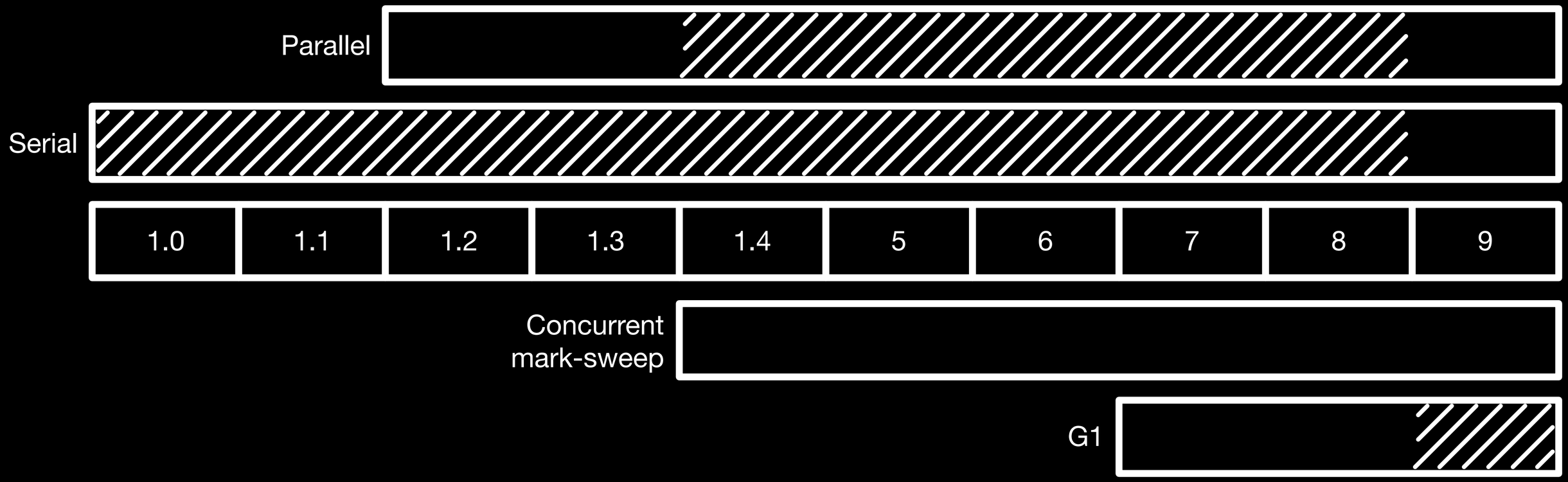

objects
lifetime





“generational”

stop-the-world



mostly concurrent

ZGC
Shenandoah

```java
public class GC {
  public static void main(String[] args) {
    for (int i = 0; i < 2000000; i++) {
      if (i % 100000 == 0) {
        System.out.println("> " + i);
      }
      Thing t = new Thing(i);
    }
  }

  private static class Thing {
    int i, j, k;
    public Thing(int i) {
      this.i = this.j = this.k = i;
    }
  }
}
```

```
> java -XX:+PrintGCDetails -Xmx16m GC
> 0
> 100000
[GC (Allocation Failure) [PSYoungGen: 4096K-


[
  GC (Allocation Failure)
  [PSYoungGen: 4096K->448K(4608K)]
  4096K->456K(15872K), 0.0007283 secs
]
[Times: user=0.00 sys=0.00, real=0.00 secs]
```

```java
public class GC {
  public static void main(String[] args) {
    for (int i = 0; i < 2000000; i++) {
      if (i % 100000 == 0) {
        System.out.println("> " + i);
      }
      Thing t = new Thing(i);
    }
  }

  private static class Thing {
    int i, j, k;
    public Thing(int i) {
      this.i = this.j = this.k = i;
    }
  }
}
```

```
> java -XX:+PrintGCDetails -Xmx16m GC
> 0
> 100000
[GC (Allocation Failure) [PSYoungGen: 4096K-
> 200000
> 300000
[GC (Allocation Failure) [PSYoungGen: 4544K-
> 400000
> 500000
> 600000
> 700000
> 800000
> 900000
> 1000000
> 1100000
> 1200000
> 1300000
> 1400000
> 1500000
> 1600000
> 1700000
```

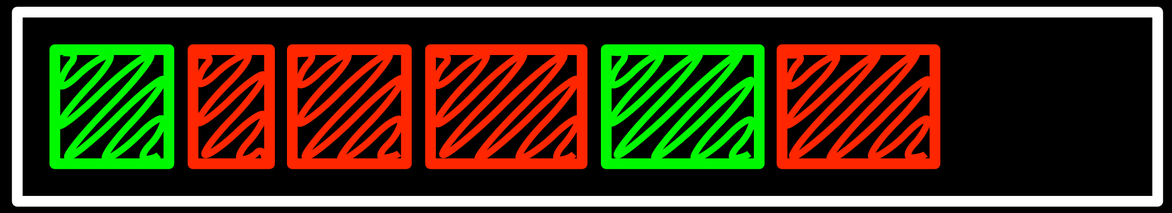
Garbage collection

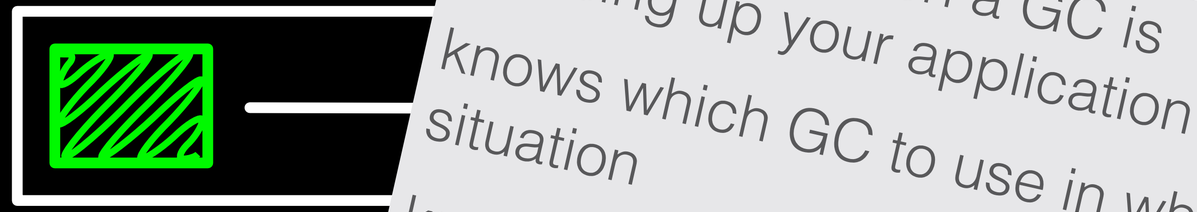
mark and sweep

compaction

generational

stop-the-world vs concurrent





Joe Mnemonic
JVM Jockey

understands when a GC is messing up your application

knows which GC to use in what situation

knows memory management is never an issue—until it is

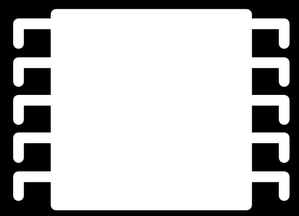
10 2a 3c 07 3d 1b 1c 60 3e b1

VM specification

Virtual machine

55 0f 4f 8f 0a b4 01 00 b4 2a c1 05 7f 3b 11

Instruction set



10 2a 3c 07 3d 1b 1c 60 3e b1

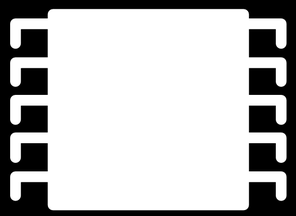
VM specification

Virtual machine

2b e2 2e 66 e2 ed 20 7f b6 73

level 0: interpreted

Instruction set



10 2a 3c 07 3d 1b 1c 60 3e b1

hotspot

VM specification

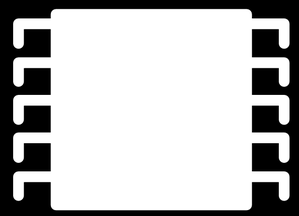
Virtual machine

level 1: compiled

“tiered compilation”

```
01 55 48 89 e5 31 c0 c7 45 fc 2a 00 00 00 c7 45 f8
04 00 00 00 8b 4d fc 03 4d f8 89 4d f4 5d c3 e5 c3
```

Instruction set



```c
#include <stdio.h>

int main() {
    for (int i = 0; i < 3; i++) {
      printf("%d", i);
    }
}
```

```
> gcc unroll.c; otool -vt a.out
100000f40  pushq   %rbp
100000f41  movq    %rsp, %rbp
100000f44  subq    $0x10, %rsp
100000f48  movl    $0x0, -0x4(%rbp)
100000f4f  movl    $0x0, -0x8(%rbp)
100000f56  cmpl    $0x3, -0x8(%rbp)
100000f5a  jge     0x100000f82
100000f60  leaq    0x47(%rip), %rdi
100000f67  movl    -0x8(%rbp), %esi
100000f6a  movb    $0x0, %al
100000f6c  callq   0x100000f8c
100000f71  movl    %eax, -0xc(%rbp)
100000f74  movl    -0x8(%rbp), %eax
100000f77  addl    $0x1, %eax
100000f7a  movl    %eax, -0x8(%rbp)
100000f7d  jmp     0x100000f56
100000f82  movl    -0x4(%rbp), %eax
100000f85  addq    $0x10, %rsp
100000f89  popq    %rbp
100000f8a  retq
```

“loop unrolling”

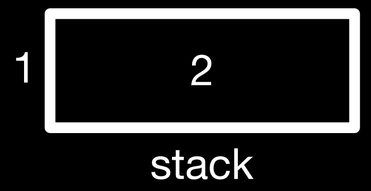
level 2-4: optimized

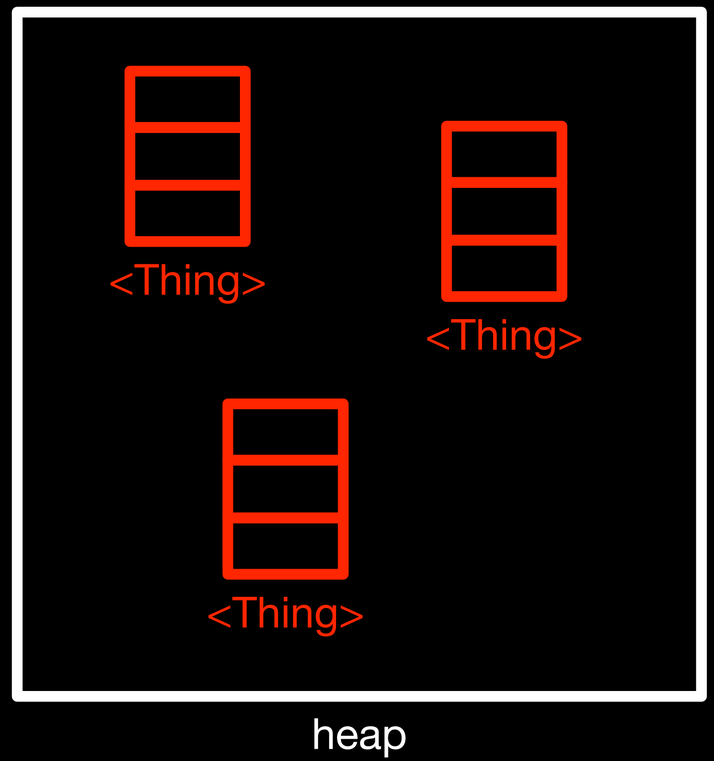
```
> gcc unroll.c -O3; otool -vt a.out
100000f50  pushq   %rbp
100000f51  movq    %rsp, %rbp
100000f54  pushq   %rbx
100000f55  pushq   %rax
100000f56  leaq    0x55(%rip), %rbx
100000f5d  xorl    %esi, %esi
100000f5f  xorl    %eax, %eax
100000f61  movq    %rbx, %rdi
100000f64  callq   0x100000f90
100000f69  movl    $0x1, %esi
100000f6e  xorl    %eax, %eax
100000f70  movq    %rbx, %rdi
100000f73  callq   0x100000f90
100000f78  movl    $0x2, %esi
100000f7d  xorl    %eax, %eax
100000f7f  movq    %rbx, %rdi
100000f82  callq   0x100000f90
100000f87  xorl    %eax, %eax
100000f89  addq    $0x8, %rsp
100000f8d  popq    %rbx
100000f8e  popq    %rbp
100000f8f  retq
```

```
 1 public class GC {
 2   public static void main(String[] args) {
 3     long time = System.nanoTime();
 4     for (int i = 0; i < 2000000; i++) {
 5       if (i % 100000 == 0) {
 6         time = reportTiming(i, time);
 7       }
 8       Thing t = new Thing(i);
 9     }
10   }
11
12   private static class Thing {
13     int i, j, k;
14     public Thing(int i) {
15       this.i = this.j = this.k = i;
16     }
17   }
18 //
19 }
```

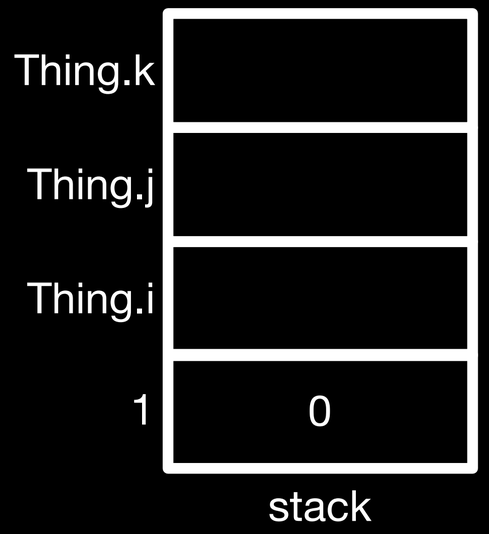
```
> java -XX:+PrintGCDetails -Xmx16m GC
>          0,      1725 ns
>     100000,  19149662 ns
[GC (Allocation Failure) [PSYoungGen: 4096K-
>     200000,   2233039 ns
>     300000,    215536 ns
>     400000,    182316 ns
>     500000,    171801 ns
>     600000,    184038 ns
>     700000,    189395 ns
>     800000,    210997 ns
>     900000,    183533 ns
>    1000000,    179427 ns
>    1100000,    179514 ns
>    1200000,    166294 ns
>    1300000,    184697 ns
>    1400000,    174437 ns
>    1500000,    181699 ns
>    1600000,    178296 ns
>    1700000,    213224 ns
>    1800000,    173726 ns
```

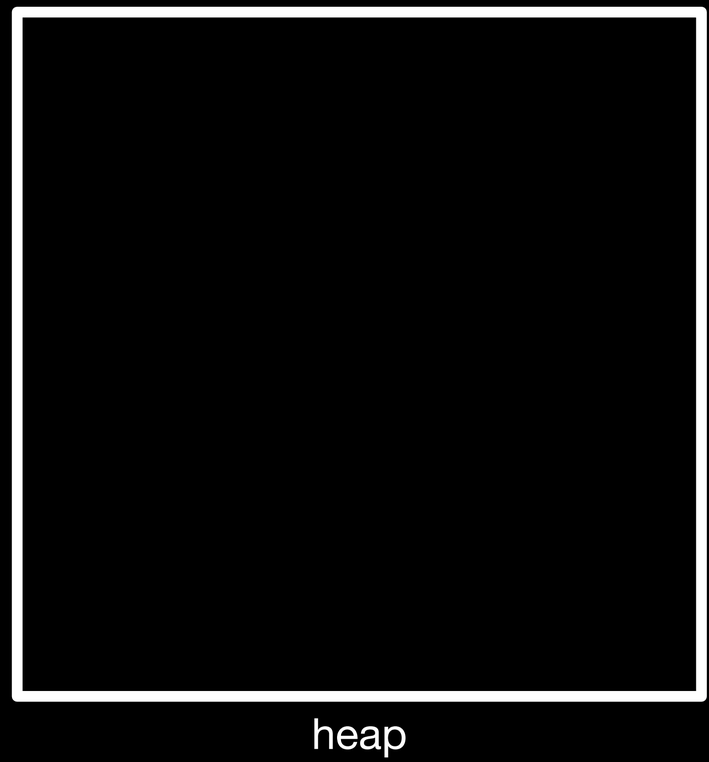
```java
public class GC {
  public static void main(String[] args) {
    for (int i = 0; i < 2000000; i++) {
      if (i % 100000 == 0) {
        System.out.println("> " + i);
      }
      Thing t = new Thing(i);
    }
  }

  private static class Thing {
    int i, j, k;
    public Thing(int i) {
      this.i = this.j = this.k = i;
    }
  }
}
```





```java
public class GC {
  public static void main(String[] args) {
    for (int i = 0; i < 2000000; i++) {
      if (i % 100000 == 0) {
        System.out.println("> " + i);
      }
      Thing t = new Thing(i);
    }
  }

  private static class Thing {
    int i, j, k;
    public Thing(int i) {
      this.i = this.j = this.k = i;
    }
  }
}
```





optimization

interpretation -> tiered compilation

machine code optimization
- loop unrolling

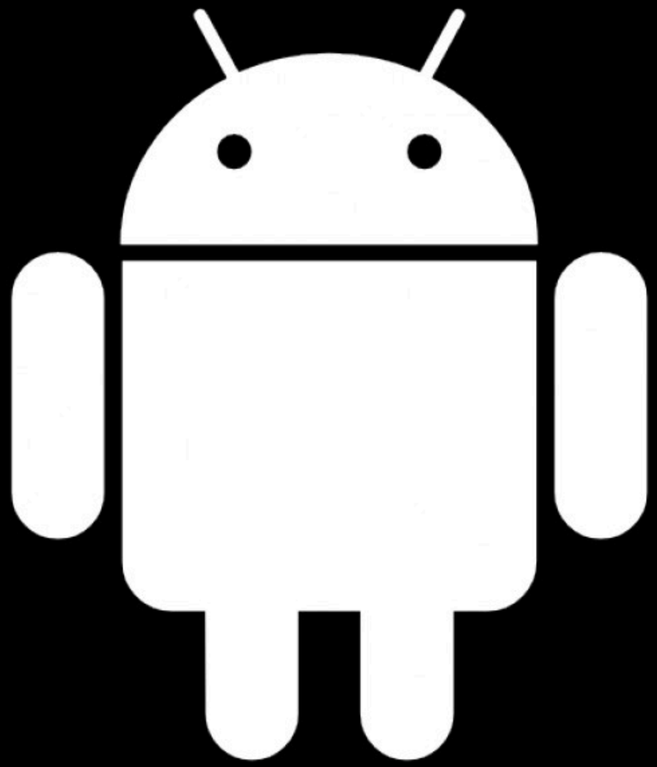
dynamic optimization
- escape analysis

…and more!
- out-of-order execution
- branch prediction
- dead code elimination

Joe Mnemonic
JVM Jockey

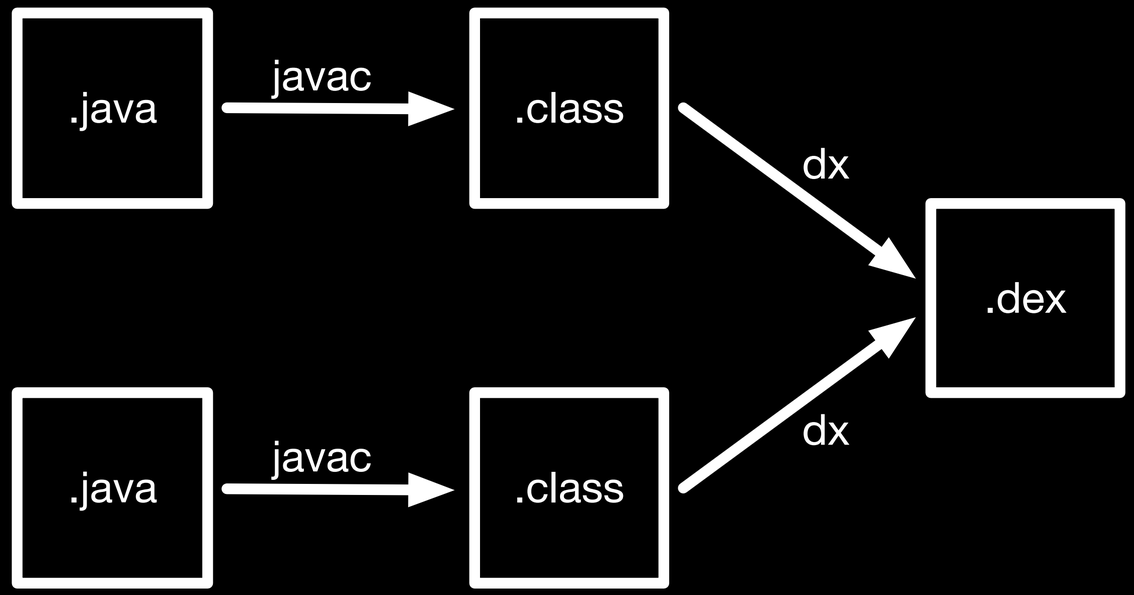
trusts the JVM to do better optimization than he can do in code

understands that JVM applications get better performance with longer run time



~~Dalvik VM~~

Android Runtime



```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = a + b;
    }
}
```

```
> xxd classes/SimpleAddition.dex
00000000: 6465 780a 3033 3500 5644 665b d728 539e  dex.035.VDf[.(S.
00000010: 8b16 d85b 7b07 f2f5 53c0 f902 419e 83ba  ...[{...S...A...
00000020: 4002 0000 7000 0000 7856 3412 0000 0000  @...p...xV4.....
00000030: 0000 0000 ac01 0000 0800 0000 7000 0000  ............p...
00000040: 0400 0000 9000 0000 0200 0000 a000 0000  ................
00000050: 0000 0000 0000 0000 0300 0000 b800 0000  ................
00000060: 0100 0000 d000 0000 5001 0000 f000 0000  ........P.......
00000070: 2201 0000 2a01 0000 4501 0000 5901 0000  "...*...E...Y...
00000080: 6e01 0000 7101 0000 7501 0000 8a01 0000  n...q...u.......
00000090: 0100 0000 0200 0000 0400 0000 0600 0000  ................
000000a0: 0400 0000 0200 0000 0000 0000 0500 0000  ................
000000b0: 0200 0000 1c01 0000 0000 0000 0000 0000  ................
000000c0: 0000 0100 0700 0000 0100 0000 0000 0000  ................
000000d0: 0000 0000 0100 0000 0100 0000 0000 0000  ................
000000e0: 0300 0000 0000 0000 9c01 0000 0000 0000  ................
000000f0: 0100 0100 0100 0000 9001 0000 0400 0000  ................
00000100: 7010 0200 0000 0e00 0100 0100 0000 0000  p...............
00000110: 9501 0000 0100 0000 0e00 0000 0100 0000  ................
00000120: 0300 063c 696e 6974 3e00 194c 6164 6469  ...<init>..Laddi
00000130: 7469 6f6e 2f53 696d 706c 6541 6464 6974  tion/SimpleAddit
00000140: 696f 6e3b 0012 4c6a 6176 612f 6c61 6e67  ion;..Ljava/lang
00000150: 2f4f 626a 6563 743b 0013 5369 6d70 6c65  /Object;..Simple
00000160: 4164 6469 7469 6f6e 2e6a 6176 6100 0156  Addition.java..V
00000170: 0002 564c 0013 5b4c 6a61 7661 2f6c 616e  ..VL..[Ljava/lan
00000180: 672f 5374 7269 6e67 3b00 046d 6169 6e00  g/String;..main.
00000190: 0300 070e 0005 0100 070e 1100 0000 0200  ................
000001a0: 0081 8004 f001 0109 8802 0000 0c00 0000  ................
000001b0: 0000 0000 0100 0000 0000 0000 0100 0000  ................
000001c0: 0800 0000 7000 0000 0200 0000 0400 0000  ....p...........
000001d0: 9000 0000 0300 0000 0200 0000 a000 0000  ................
000001e0: 0500 0000 0300 0000 b800 0000 0600 0000  ................
000001f0: 0100 0000 d000 0000 0120 0000 0200 0000  ......... ......
00000200: f000 0000 0110 0000 0100 0000 1c01 0000  ................
00000210: 0220 0000 0800 0000 2201 0000 0320 0000  . ......"....  ..
00000220: 0200 0000 9001 0000 0020 0000 0100 0000  ........ ......
00000230: 9c01 0000 0010 0000 0100 0000 ac01 0000  ................
```

```
> dexdump -d classes/SimpleAddition.dex
Processing 'classes/SimpleAddition.dex'...
Opened 'classes/SimpleAddition.dex', DEX version '035'
Class #0            -
  Class descriptor  : 'Laddition/SimpleAddition;'
  Access flags      : 0x0001 (PUBLIC)
  Superclass        : 'Ljava/lang/Object;'
  Interfaces        -
  Static fields     -
  Instance fields   -
  Direct methods    -
    #0              : (in Laddition/SimpleAddition;)
      name          : '<init>'
      type          : '()V'
      access        : 0x10001 (PUBLIC CONSTRUCTOR)
      code          -
      registers     : 1
      ins           : 1
      outs          : 1
      insns size    : 4 16-bit code units
0000f0:                                        |[0000f0] addition.SimpleAddition.<init>:()V
000100: 7010 0200 0000                         |0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V // m
000106: 0e00                                   |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=3
      locals        :
        0x0000 - 0x0004 reg=0 this Laddition/SimpleAddition;
```

```
0000f0:                                        |[0000f0] addition.SimpleAddition.<init>:()V
000100: 7010 0200 0000                         |0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V // m
000106: 0e00                                   |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=3
      locals        :
        0x0000 - 0x0004 reg=0 this Laddition/SimpleAddition;

    #1              : (in Laddition/SimpleAddition;)
      name          : 'main'
      type          : '([Ljava/lang/String;)V'
      access        : 0x0009 (PUBLIC STATIC)
      code          -
      registers     : 1
      ins           : 1
      outs          : 0
      insns size    : 1 16-bit code units
000108:                                        |[000108] addition.SimpleAddition.main:([Ljava/lang/String;)V
000118: 0e00                                   |0000: return-void
      catches       : (none)
      positions     :
        0x0000 line=5
        0x0000 line=8
      locals        :

  Virtual methods   -
  source_file_idx   : 3 (SimpleAddition.java)
```

```
0000f0:                                        |[0000f0] addition.SimpleAddition.<init>:()V
000100: 7010 0200 0000                         |0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V // m
000106: 0e00                                   |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=3
      locals        :
        0x0000 - 0x0004 reg=0 this Laddition/SimpleAddition;

    #1              : (in Laddition/SimpleAddition;)
      name          : 'main'
      type          : '([Ljava/lang/String;)V'
      access        : 0x0009 (PUBLIC STATIC)
      code          -
      registers     : 1
      ins           : 1
      outs          : 0
      insns size    : 1 16-bit code units
000108:                                        |[000108] addition.SimpleAddition.main:([Ljava/lang/String;)V
000118: 0e00                                   |0000: return-void
      catches       : (none)
      positions     :
        0x0000 line=5
        0x0000 line=8
      locals        :

  Virtual methods   -
  source_file_idx   : 3 (SimpleAddition.java)
```

```java
package addition;

public class SimpleAddition {
    public static void main(String[] args) {
        int a = 42; // 0x2A in hex
        int b = 4;
        int result = a + b;
    }
}
```

```java
public class SimpleAdditionWithPrint {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = 4;
      int result = a + b;
      System.out.println("> " + result);
    }
}
```

```
|0000: const/16 v0, #int 46 // #2e
|0002: sget-object v1, Ljava/lang/System;.out:Ljava/io/PrintStream; // field@0000
|0004: new-instance v2, Ljava/lang/StringBuilder; // type@0005
|0006: invoke-direct {v2}, Ljava/lang/StringBuilder;.<init>:()V // method@0004
|0009: const-string v3, "> " // string@0001
|000b: invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;.append:(Ljava/lang/Stri
|000e: move-result-object v2
|000f: invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;.append:(I)Ljava/lang/St
|0012: move-result-object v0
|0013: invoke-virtual {v0}, Ljava/lang/StringBuilder;.toString:()Ljava/lang/Strin
|0016: move-result-object v0
|0017: invoke-virtual {v1, v0}, Ljava/io/PrintStream;.println:(Ljava/lang/String;
|001a: return-void
```

register based

```java
public class SimpleAdditionWithRandom {
    public static void main(String[] args) {
      int a = 42; // 0x2A in hex
      int b = (int)(Math.random()*100);
      int result = a + b;
      System.out.println("> " + result);
    }
}
```

```
|[0001d8] addition.SimpleAdditionWithRandom.main:([Ljava/lang/String;)V
|0000: const/16 v0, #int 42 // #2a
|0002: invoke-static {}, Ljava/lang/Math;.random:()D // method@0003
|0005: move-result-wide v2
|0006: const-wide/high16 v4, #long 4636737291354636288
|0008: mul-double/2addr v2, v4
|0009: double-to-int v1, v2
|000a: add-int/2addr v0, v1
|000b: sget-object v1, Ljava/lang/System;.ou
|000d: new-instance v2, Ljava/lang/StringBui
|000f: invoke-direct {v2}, Ljava/lang/String
|0012: const-string v3, "> " // string@0001
|0014: invoke-virtual {v2, v3}, Ljava/lang/S
|0017: move-result-object v2
```

b0..cf 12x	*binop*/2addr vA, vB b0: add-int/2addr b1: sub-int/2addr b2: mul-int/2addr b3: div-int/2addr b4: rem-int/2addr b5: and-int/2addr b6: or-int/2addr b7: xor-int/2addr b8: shl-int/2addr	**A:** destination and first source register or pair (4 bits) **B:** second source register or pair (4 bits)

~~Dalvik~~ ART

single archive

compile-time optimization
- dead code elimination
- code-level optimization

register based

Joe Mnemonic
JVM Jockey

understands that Dalvik is an entirely different beast than the server Java VM

can explain the architectural tradeoffs for this different VM

JVM
java to bytecode
stack based instruction set
allocation & invocation
…and more!
    concurrency primitives
    instanceof
    arrays

Garbage collection
mark and sweep
compaction
generational
stop-the-world vs
    concurrent

Dalvik
single archive
compile-time optimization
register based

Optimization
interpretation -> tiered compilation
L2-4 machine code optimization
    loop unrolling
escape analysis
…and more!
    out-of-order execution
    dead code elimination

https://github.com/angelos/wydkywtkatjvm

Angelo van der Sijpt
@_angelos
angelo.vandersijpt@luminis.eu

luminis
Conversing worlds