Eschatology

- Some people have asked about the final project
  - It will be released in two weeks
  - Students can work alone or in a pair
- You will implement a generational GC algorithm
  - Copying collection in the nursery
  - Mark/Sweep collection in the mature space
  - Those working alone will implement simpler algorithms
- I'd like to get an idea of who's working with who
  - Drop me an e-mail if you're planning on forming a team

Eschatology II

- Final two classes on December 1st and 8th
  - Shorter special topics
  - Current state of the art
  - Review of the semester

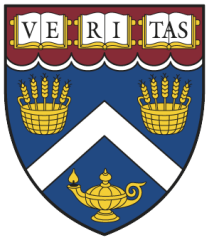
- Suggest topics by e-mail or discussion forum
  - Related areas of interest
  - Things that you'd like to hear more about
  - Anything that wasn't clear during the semester

Eschatology III

- A reminder about the final exam
  - Take-home format
  - Released on the Monday of finals week
  - Due on the Friday

- All material that we've covered is fair game
  - Lecture content
  - Experience when implementing in SimpleJava
  - Assigned readings

Debugging and Profiling



Debugging

- Any mechanism that lets the developer trace code
  - `System.out.println`
  - `new Throwable().printStackTrace()`

- Sometimes stack tracing isn't enough
  - Need deeper insight into code
  - Ability to step through line by line
  - Want to inspect variables or heap objects

Traditional Debugging

- Build application in debug mode
  - Include extra symbol information
  - Turn off optimization
- Use debugger tools to inspect the process
  - Can start with the debugger enabled
  - Or attach the debugger to a running process

Debugging Difficulties

- Debugger can change program characteristics
  - Introduce timing delays
  - Hide race conditions
  - Change cache behavior
- Some bugs don't manifest in debug builds
  - Aggressive optimization changes behavior
  - Operations may have undefined results
  - Developers get away with errors in debug mode

Class File Debug Support

- Java class files ship with debug information
  - File name
  - Line numbers
  - Local variable names
- Bytecode is not optimized
  - Very simple translation from Java to bytecode
  - Makes decompilation simple

Interpreter Debugging

- Well understood process
  - We've run through many examples in class
  - Simple mapping from bytecode to program state
- Breakpoints can be set between bytecodes
  - We've already seen how safepointing works
- Performance is less important
  - Interpreter performance isn't great anyway

Mixed Mode Execution

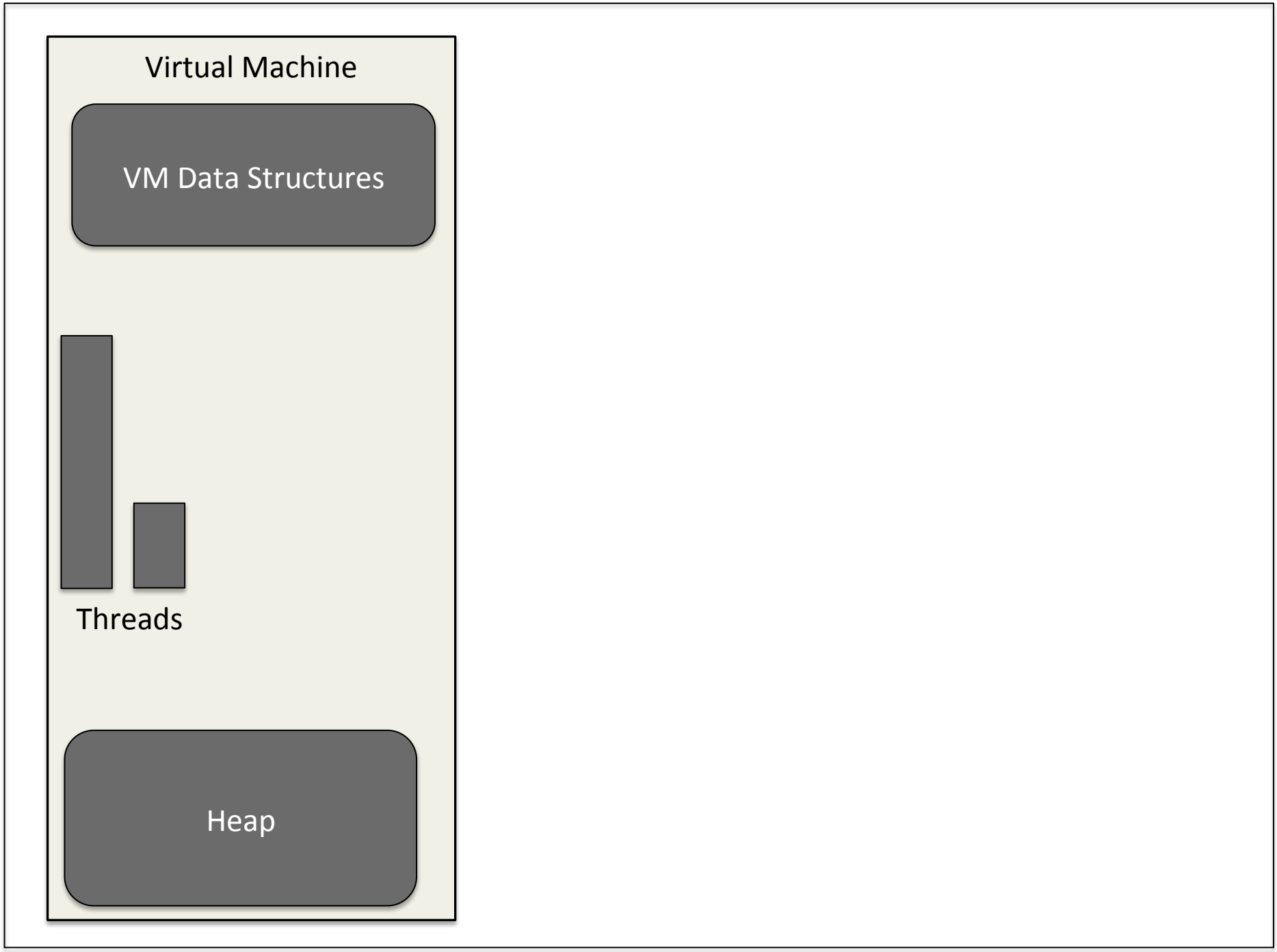
- Stacks can go between JIT and interpreter
  - Both modes need to support debugging
  - Need to transition smoothly
- May want to hot-swap executing code
  - On-stack replacement
- Threads can also enter native code
  - Native debugging is still a research question

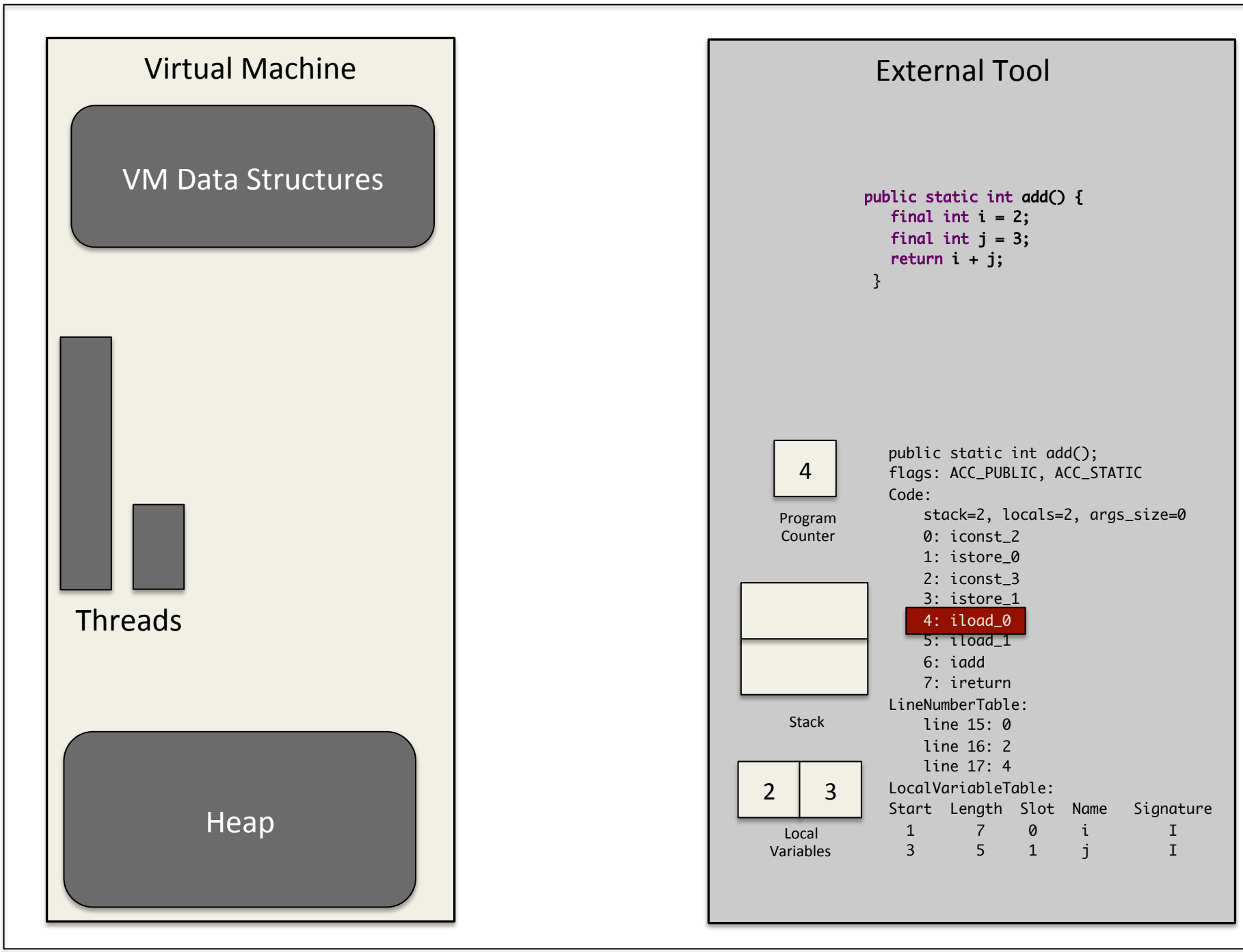
Mixed Mode Execution

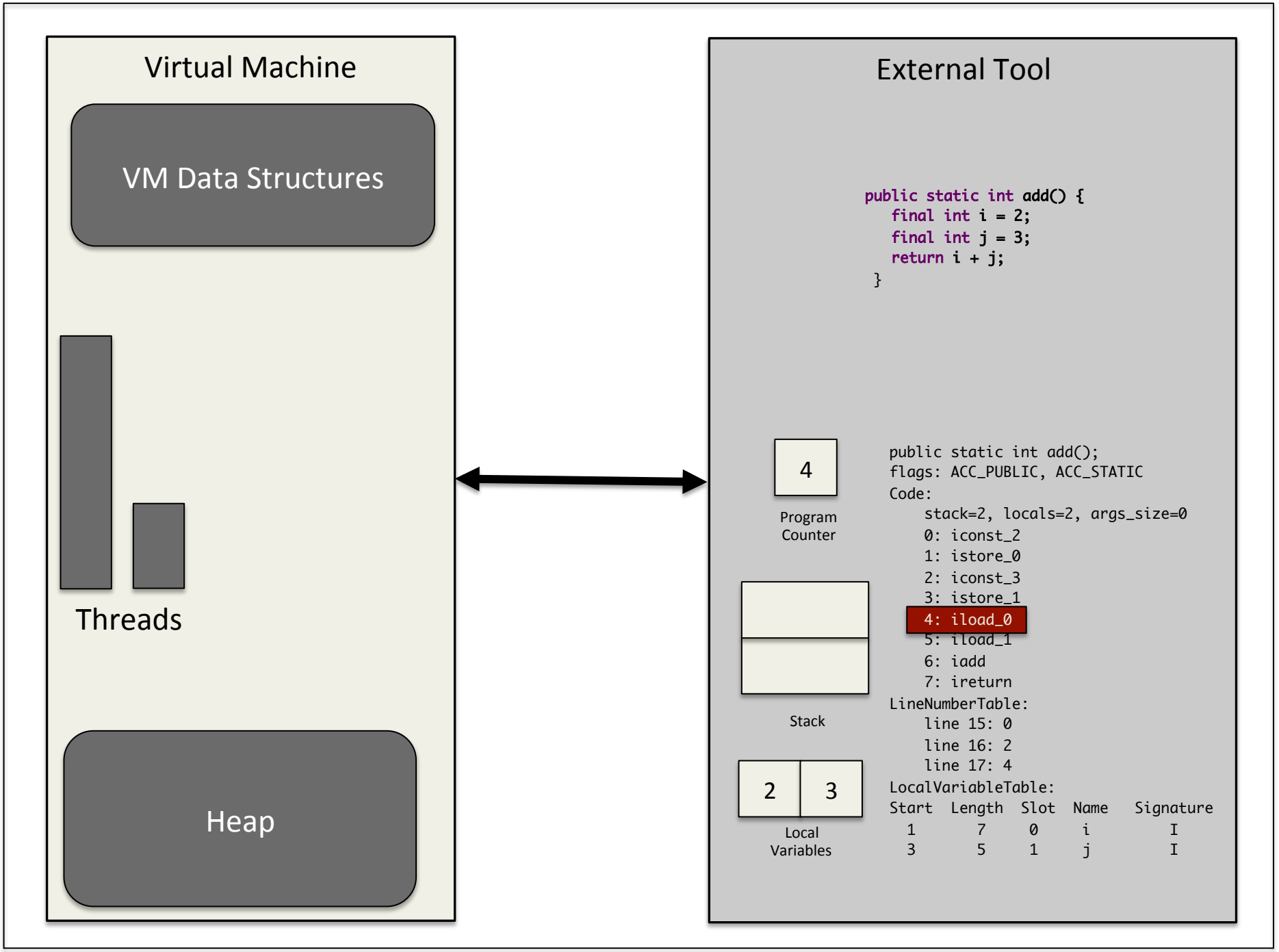
- Differences can crop up between modes
  - Less common than in native applications
  - Fewer undefined operations
  - Stricter checking of bytecode
- Some things can still be affected
  - Memory layout changes
  - GC behavior
  - Lots of timing changes

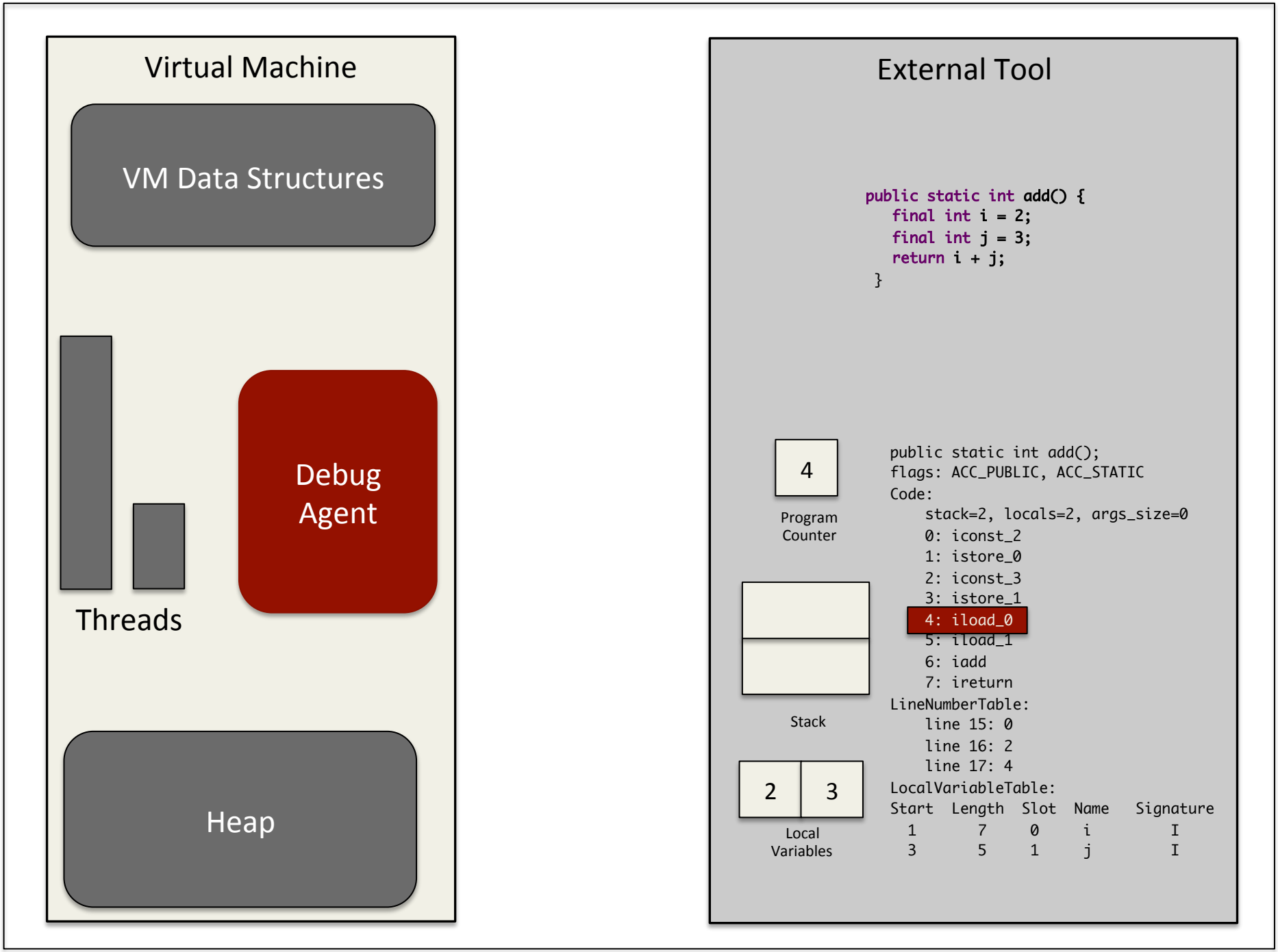
Out of Process Debugging

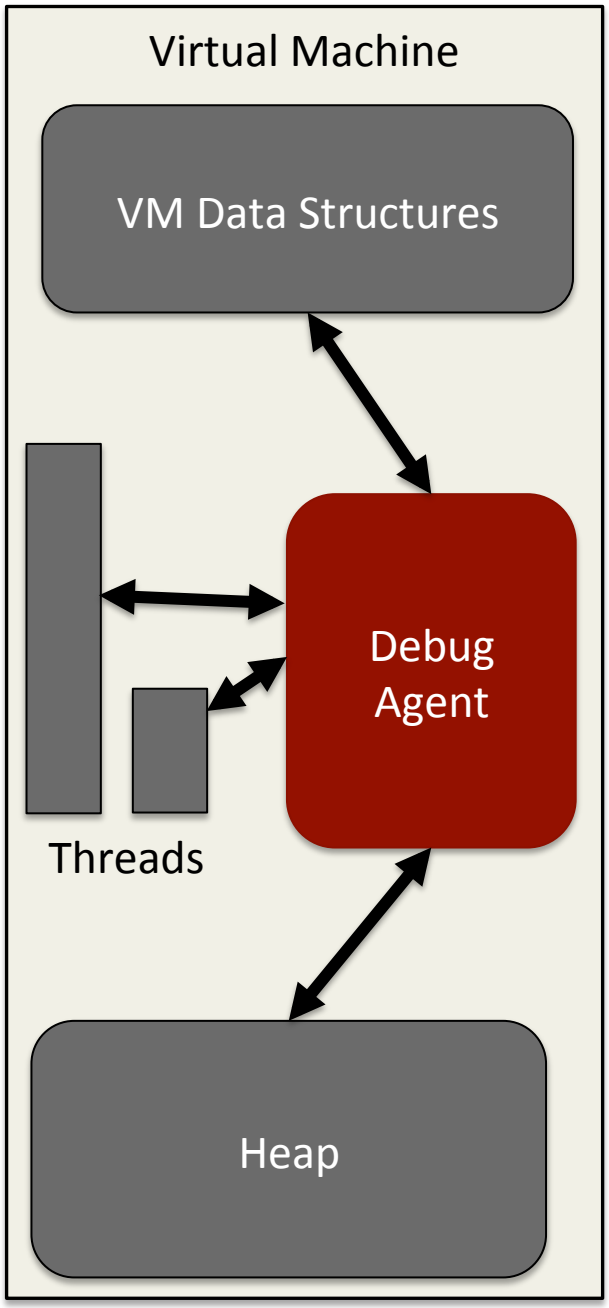
- Running in debug mode can be slow
  - Artificially low optimization level
  - Overhead of debugger operations
- Want to minimize the overhead
  - Particularly when the debugger's not running
- Launch debugger as separate process
  - Debugger's work doesn't interfere with the program


Virtual Machine
VM Data Structures
Threads
Heap


Virtual Machine
VM Data Structures
Threads
Heap
External Tool
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

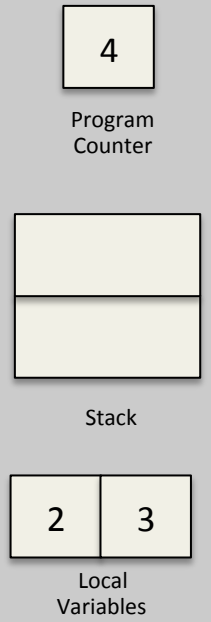

Virtual Machine
VM Data Structures
Threads
Heap
External Tool
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
External Tool
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

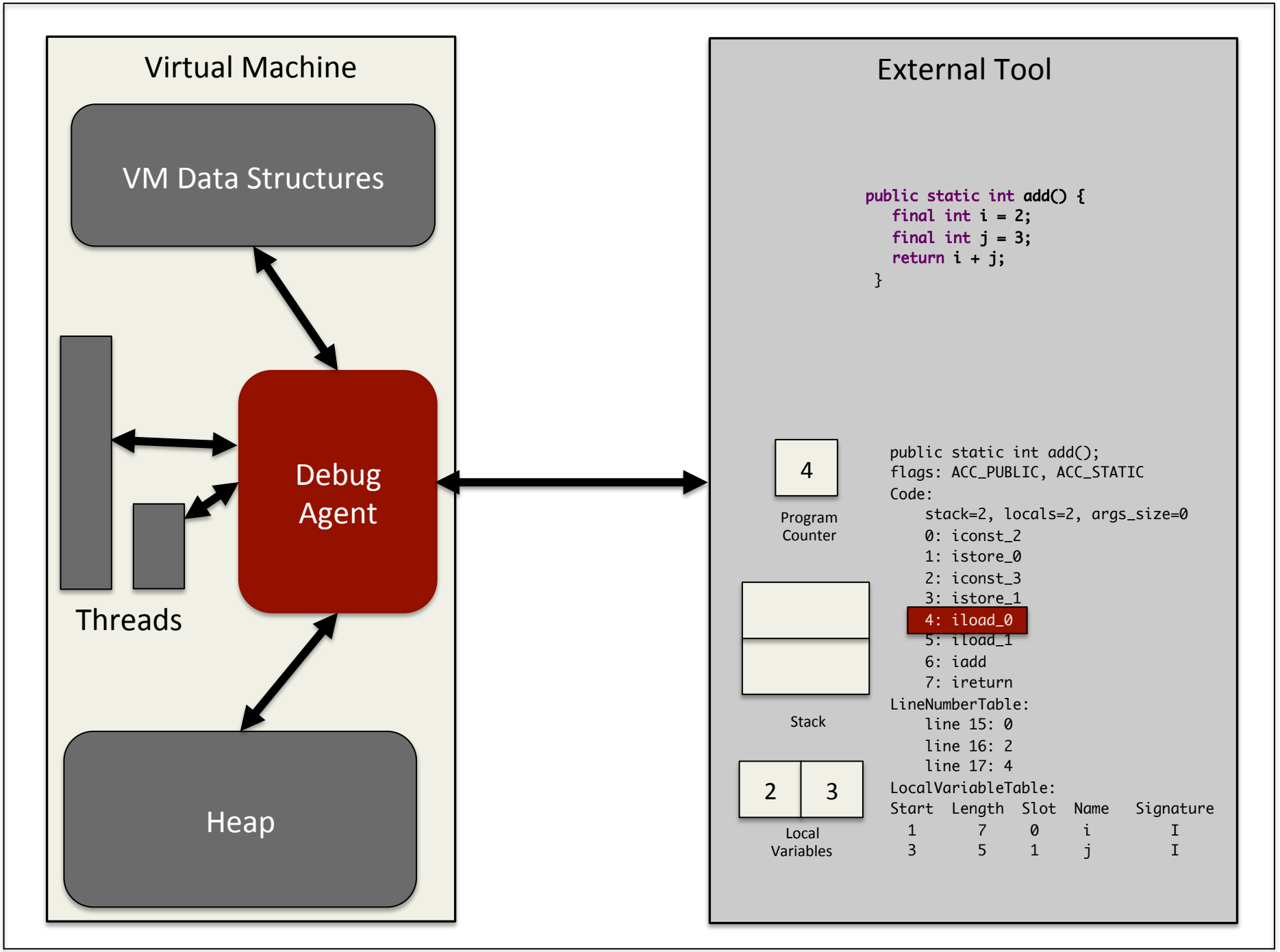


External Tool

```
public static int add() {
   final int i = 2;
   final int j = 3;
   return i + j;
 }
```




```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i        I
  3       5     1     j        I
```


Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
External Tool
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

JVMTI

- The JVM Tools Interface
  - Optional part of the VM specification
  - Introduced with version 1.5 of the JDK
  - Supports external tools that interact with the VM
- Disabled by default
  - Load agent on the command line
- Tool determines which features to turn on

JVMTI Agents

- Written in native language
  - Interface definitions in the JDK include directory
  - Generally a small piece of code
- Agent runs in the VM's address space
- Two-way communication with the VM
  - VM notifies agent of interesting events
  - Agent can control the VM
- Agent relays information to external tool

Loading an Agent

- VM loads the agent and calls startup function
  - `Agent_OnLoad`
- VM only partially created when agent launches
  - System property values have been set
  - No classes loaded
  - No objects created
  - No bytecodes executed
- Failure in agent creation leads to VM shutdown

Attaching an Agent

- Agents can be added once the VM is running
  - Uses `Agent_OnAttach` initialization method
- May have more limited functionality
  - May not be able to instrument running classes
  - Overhead for some features may be higher
- Failure in agent attach ignored by the VM

Timing

- Actions in the `Agent_OnLoad` call restricted
  - Parts of the VM not yet created
- Early loading allows agent to be more powerful
  - Capture VM creation events
  - Instrument system bytecode
  - Influence the newly-created VM

Unloading an Agent

- Agent can create an `Agent_OnUnload` function
  - Free up any resources held by the agent
- Triggers for unloading implementation specific
  - May be manual or automatic
- Called on clean VM shutdown
  - Not called if the VM crashes

Accessing VM Functionality

- The JVMTI agent uses a set of VM functions
  - Environment pointer to thread data and functions
  - Double indirection to allow function replacement
  - Allowable methods limited in some cases
- Control is inverted from JNI
  - Most agent threads not created by Java method call
  - Can't assume that they will eventually return
  - Complicates object reference management

Capabilities

- Set of events and permissions for the agent
  - Specified in `Agent_OnLoad`
- Capabilities are implementation dependent
  - Not all VMs will support all capabilities
  - Capabilities determine agent portability
  - Not all capabilities available after startup
- VM need only provide requested capabilities
  - Can tailor configuration for performance

Event Handling Capabilities

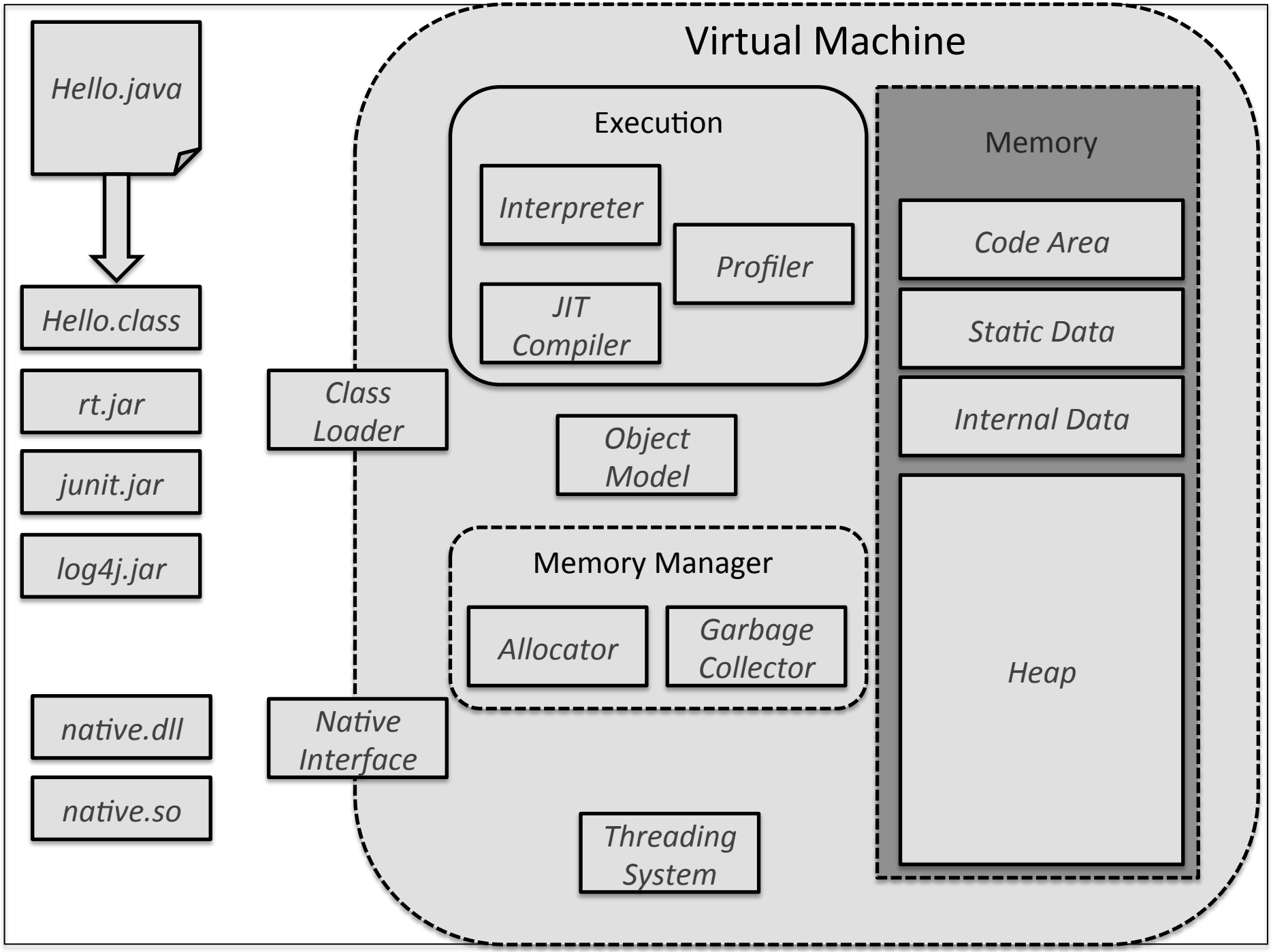
- Some capabilities cause callbacks to event handlers
  - Field modification
  - Field access
  - Method entry/exit
  - Class loading
  - JIT Compilation
  - Native method binding
  - Some allocation
  - Garbage collection
  - Resource exhaustion

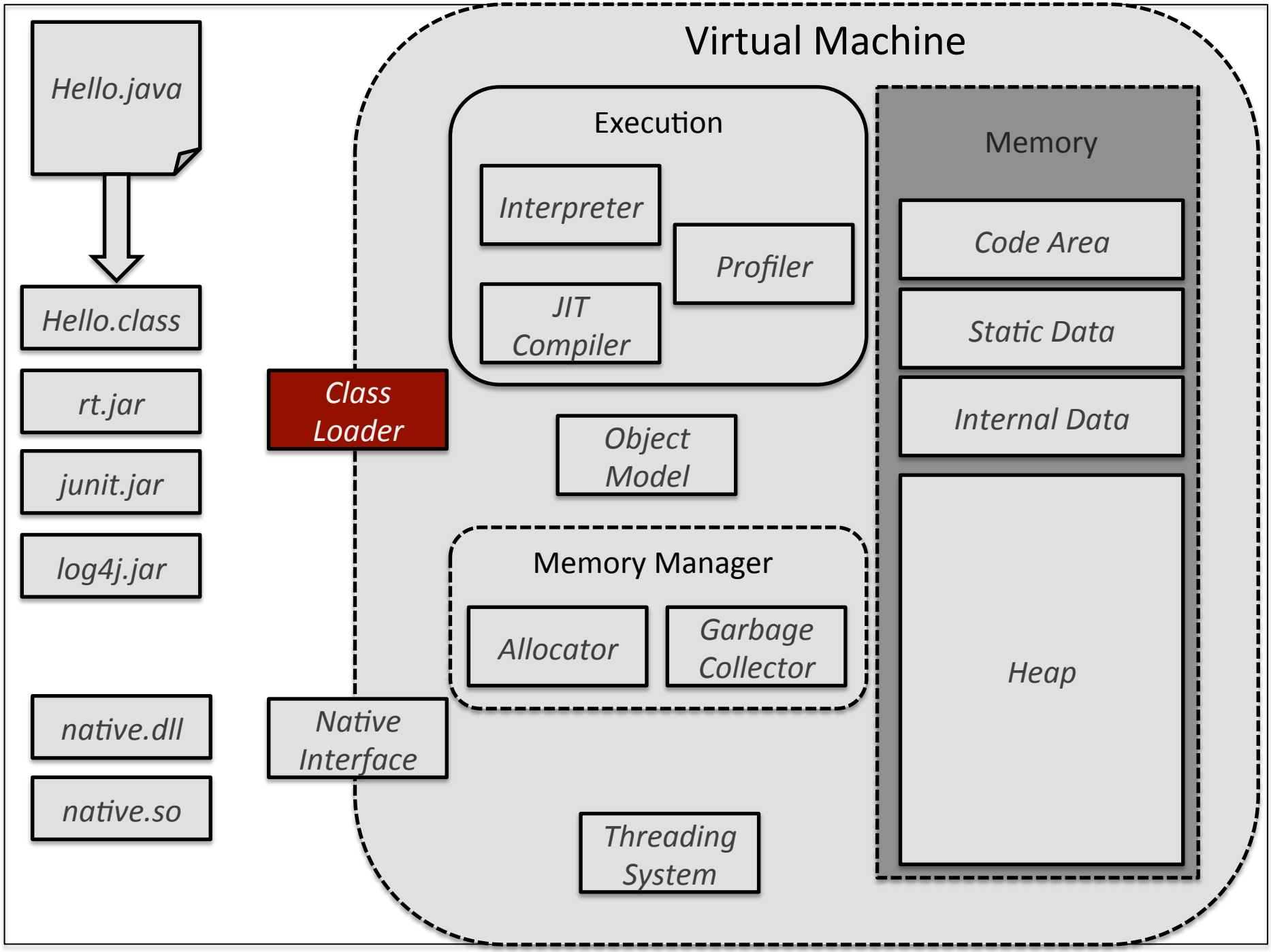
System Monitoring Capabilities

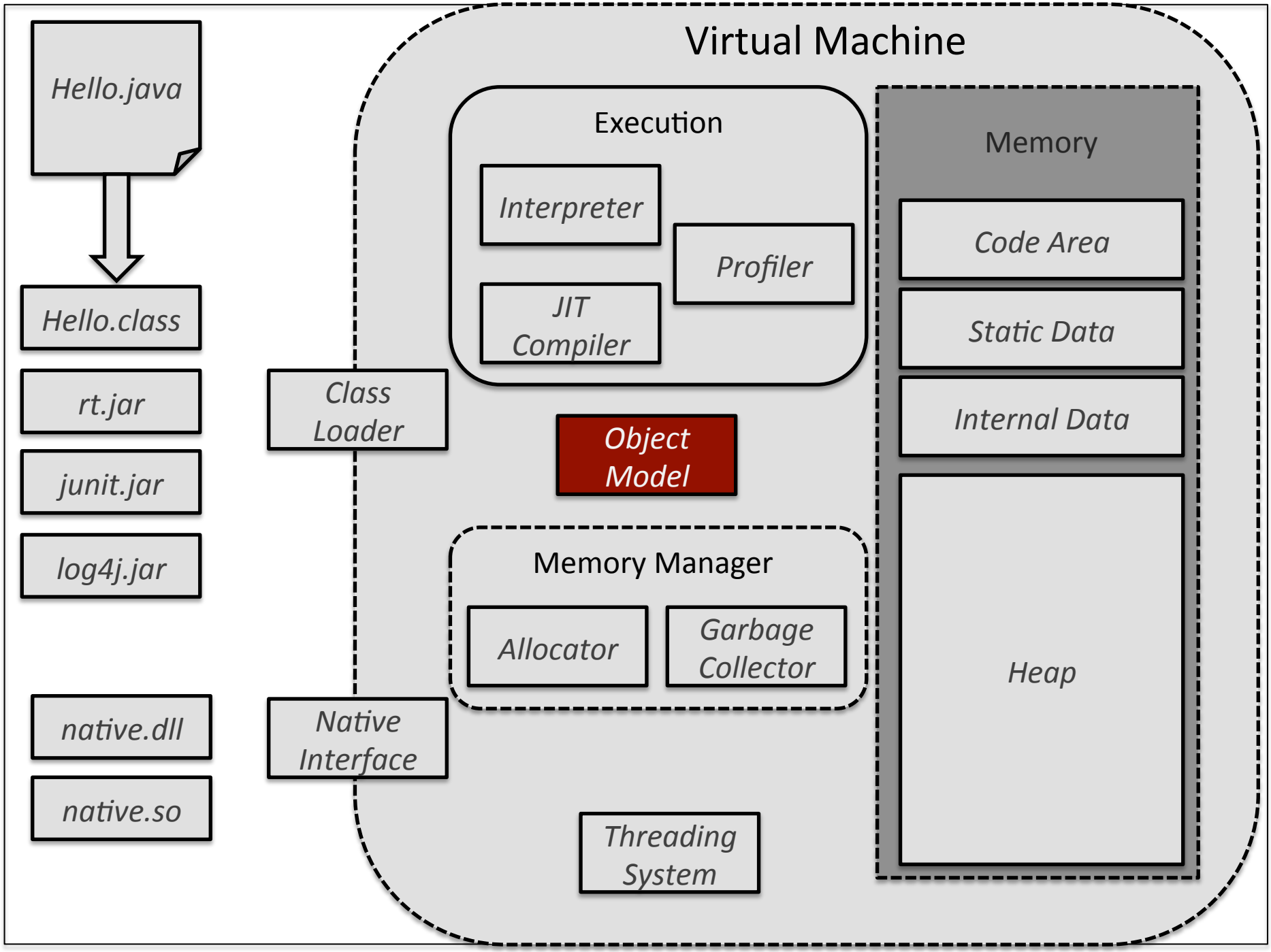
- Others allow tools to access system information
  - Get monitor states
  - Get bytecodes for a method
  - Get source file and line numbers
  - Get class constant pool
  - Get CPU time for a given thread

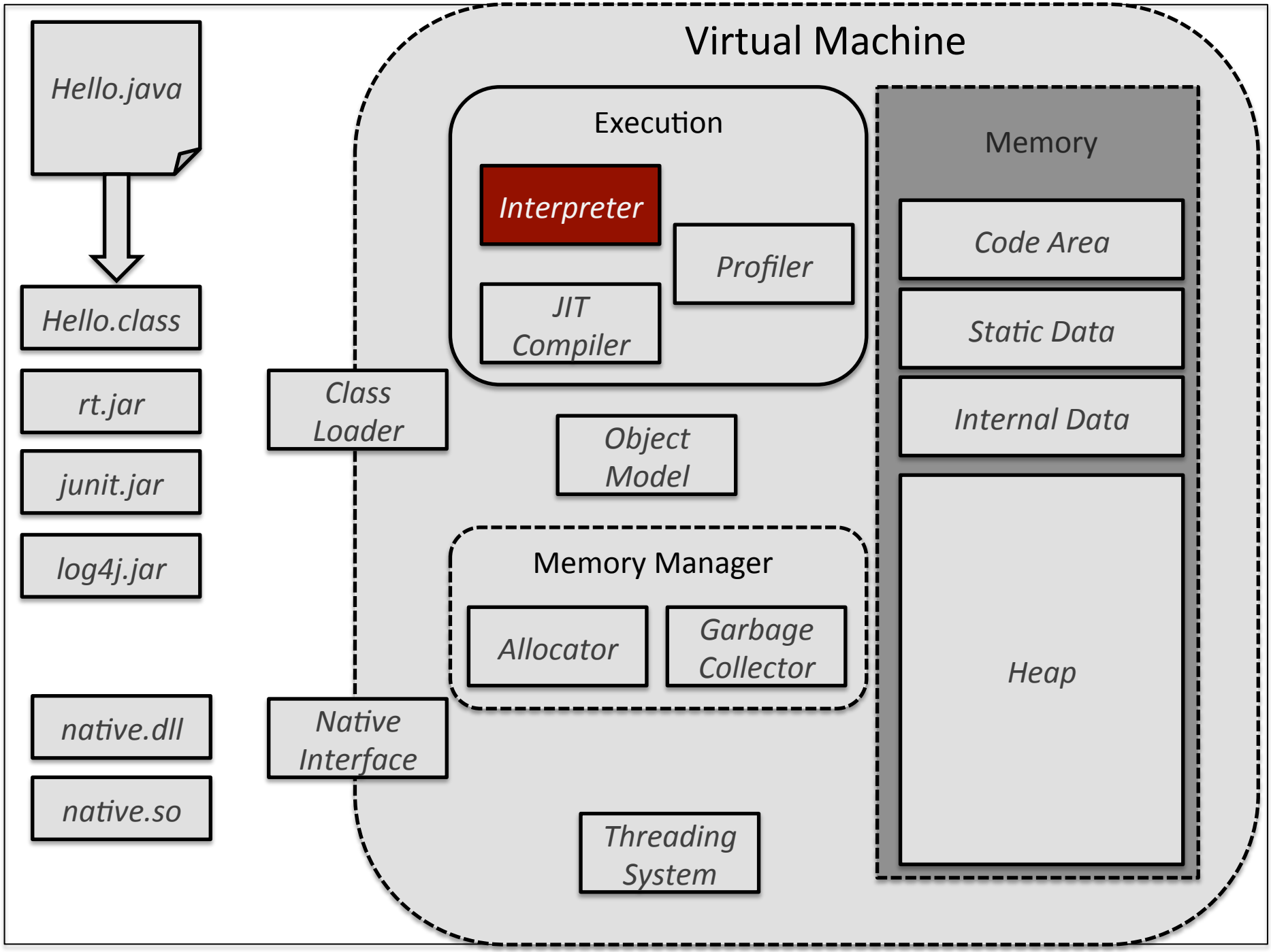
Agent Functionality Capabilities

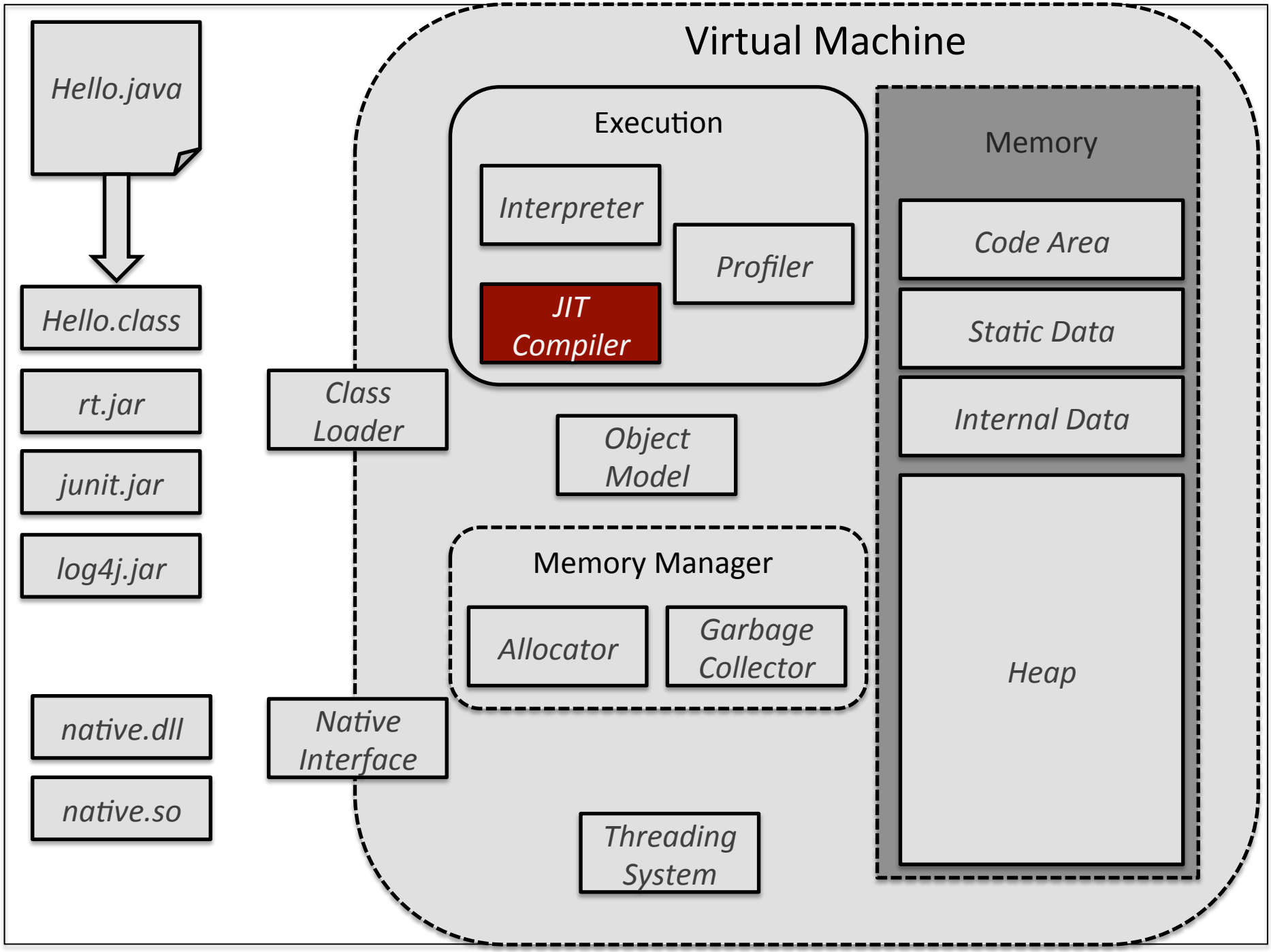
- Other capabilities let the agent perform actions
  - Tag objects on the heap
  - Pop frames from the stack or force method return
  - Redefine or re-transform classes
  - Stop, suspend or interrupt threads
  - Set breakpoints
  - Single-step through code
  - Generate events for other agents

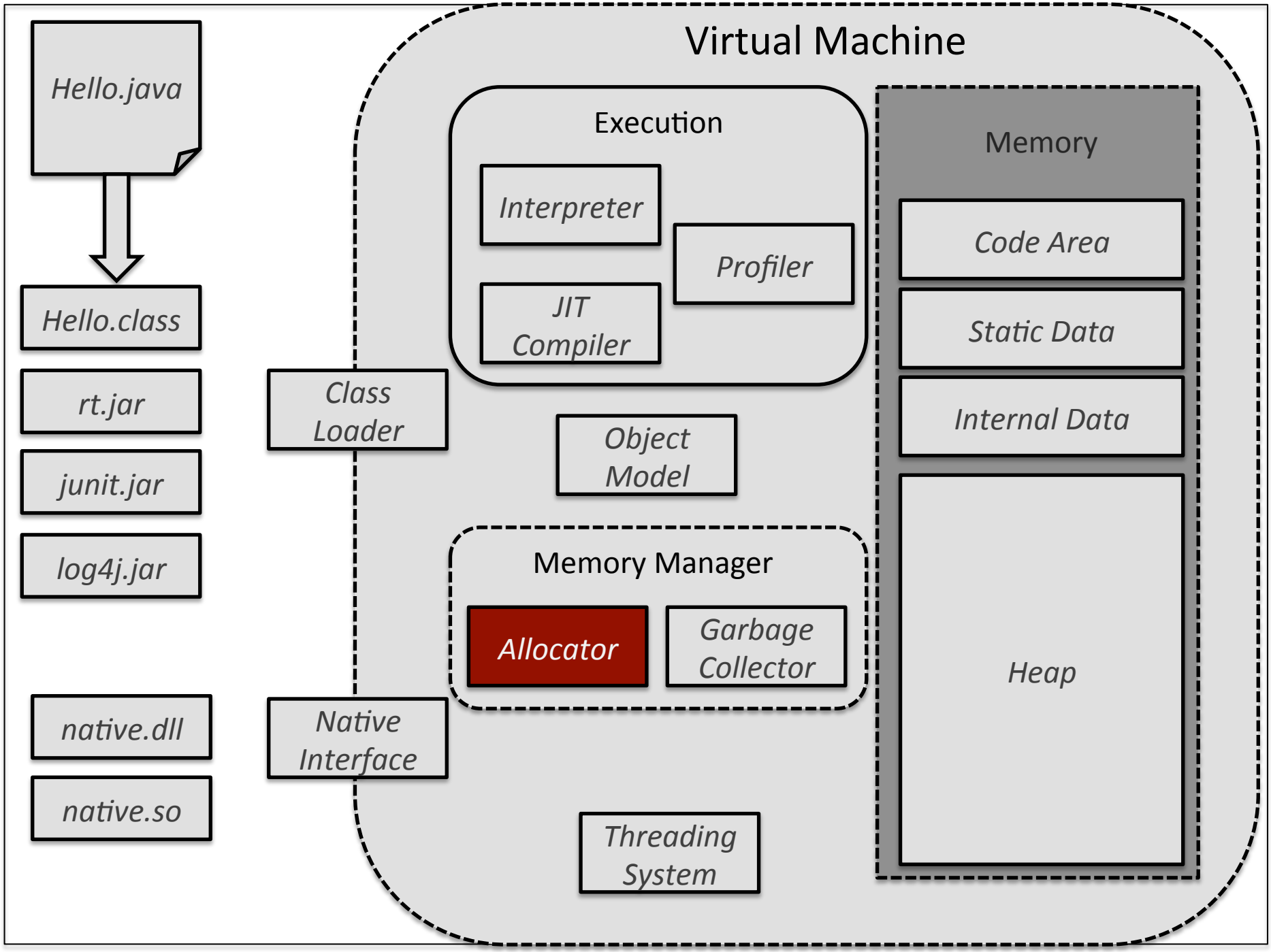

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

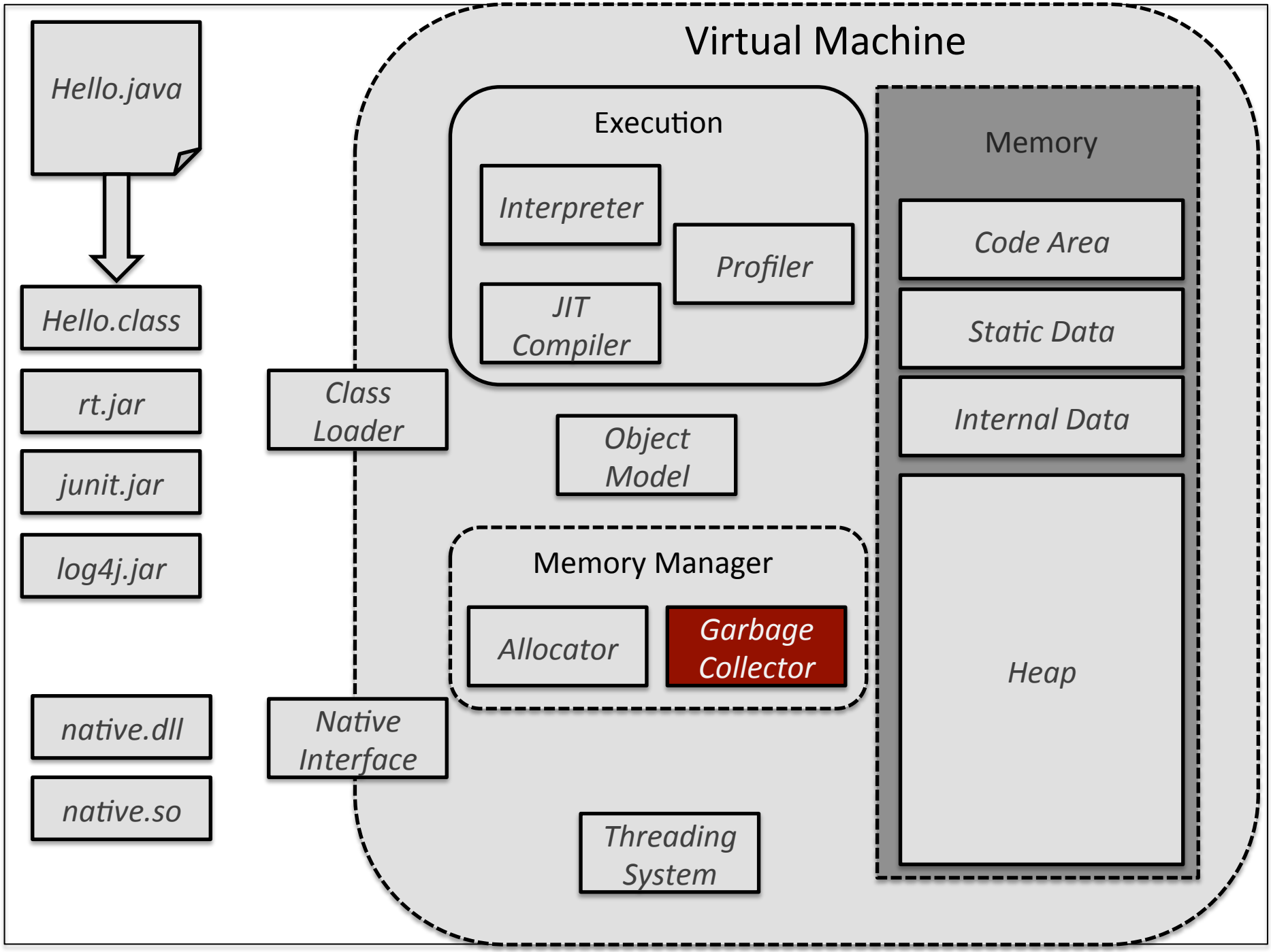

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

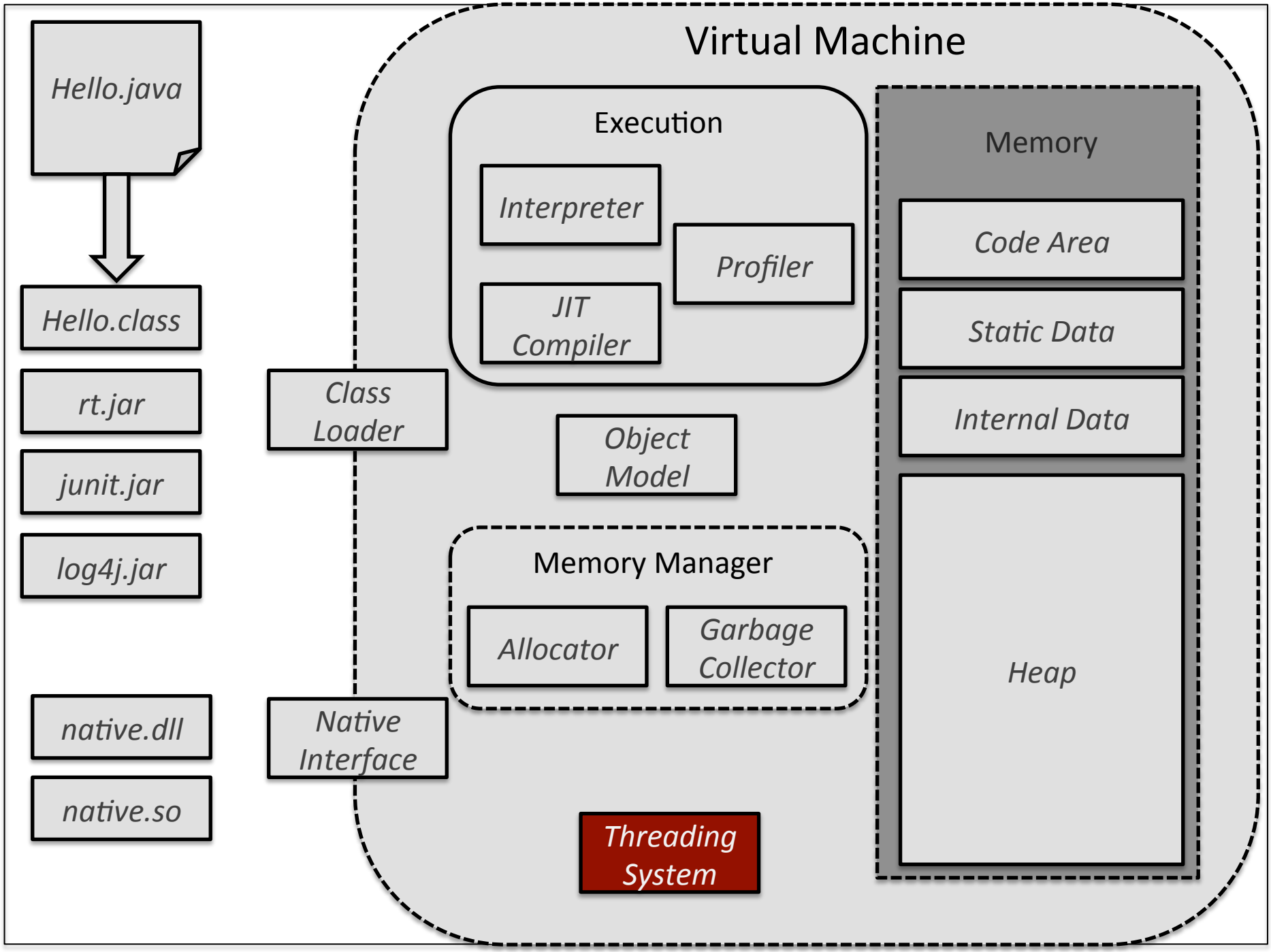

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

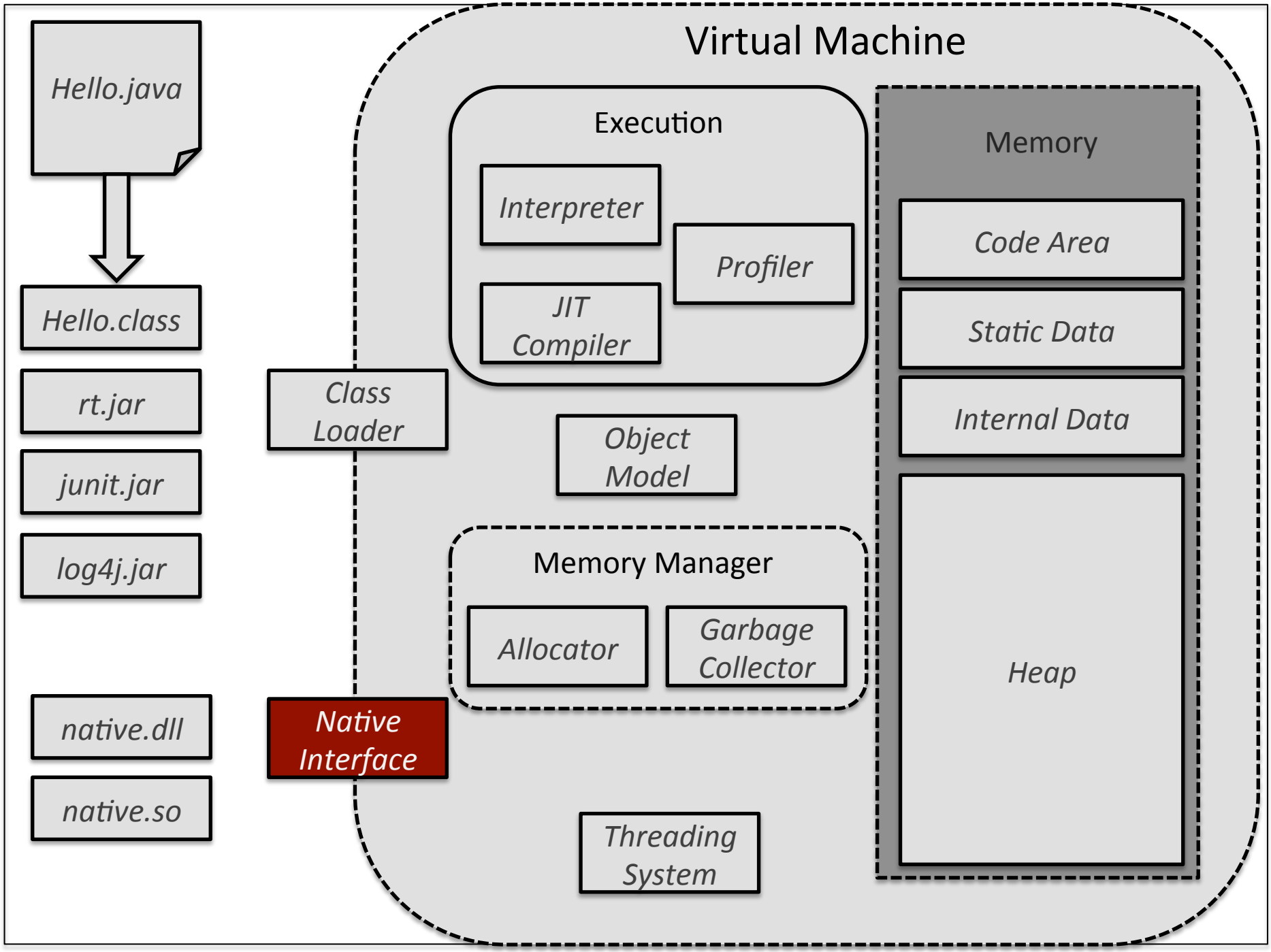

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap


Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap


Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

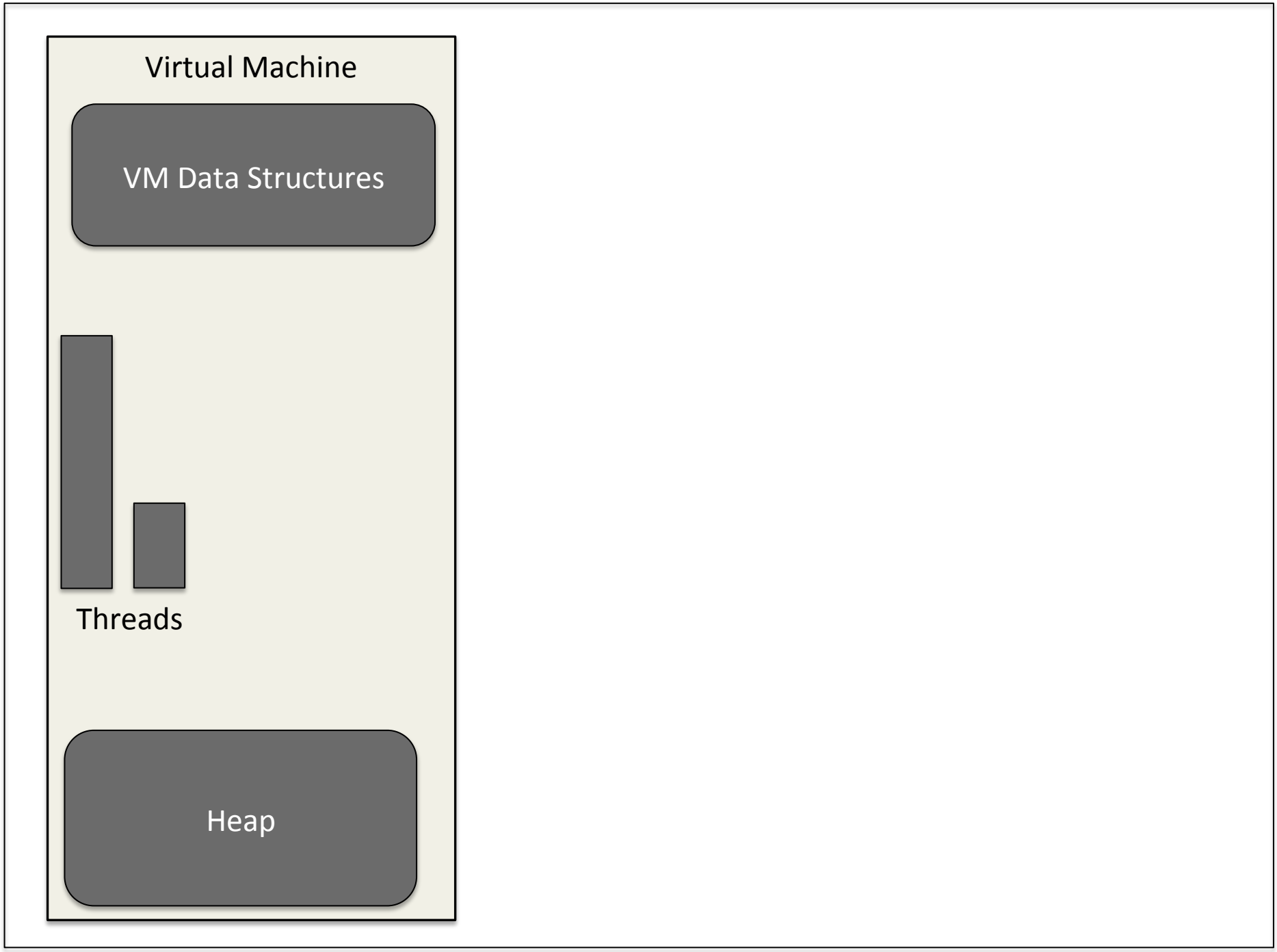

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

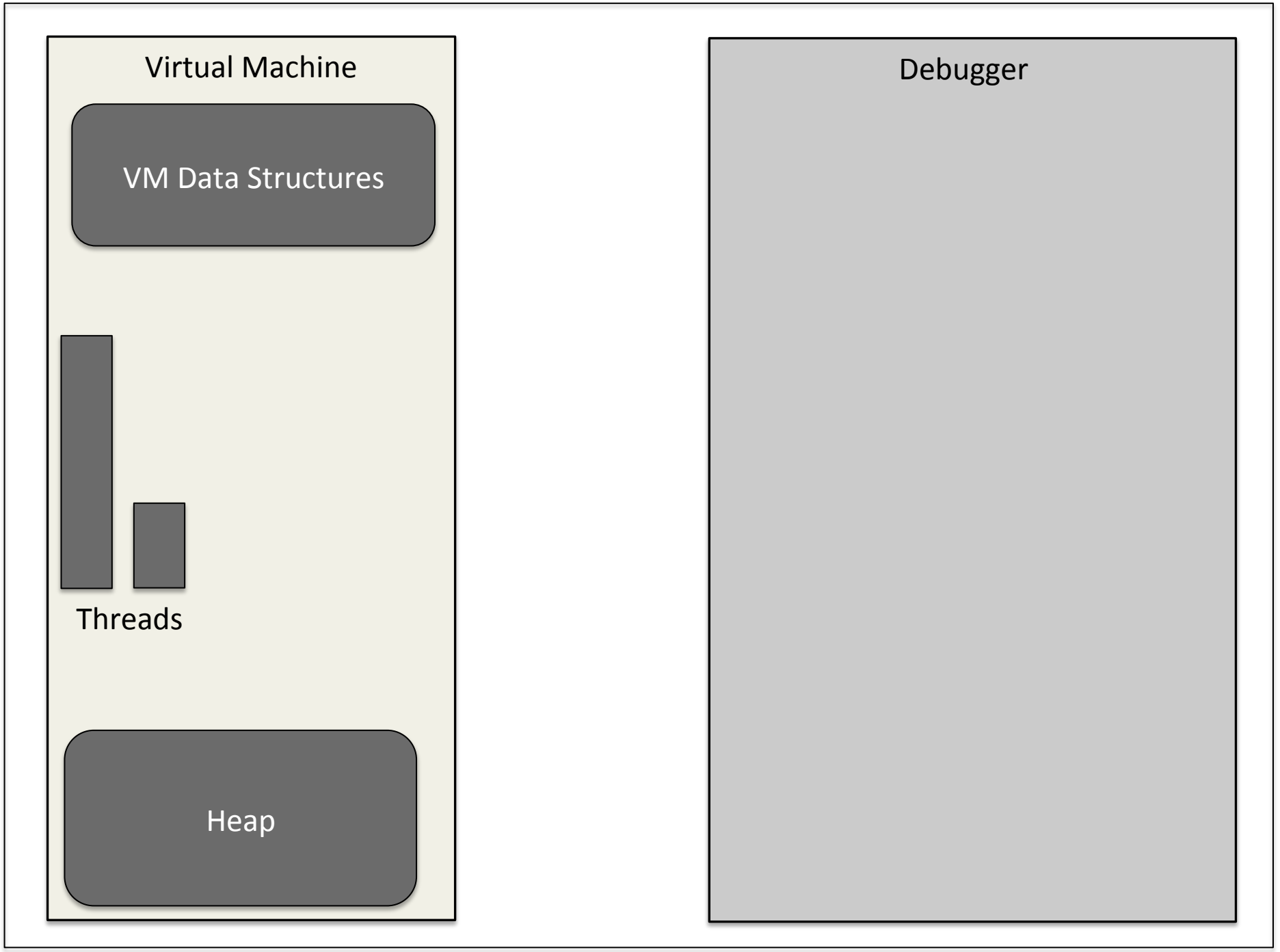

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

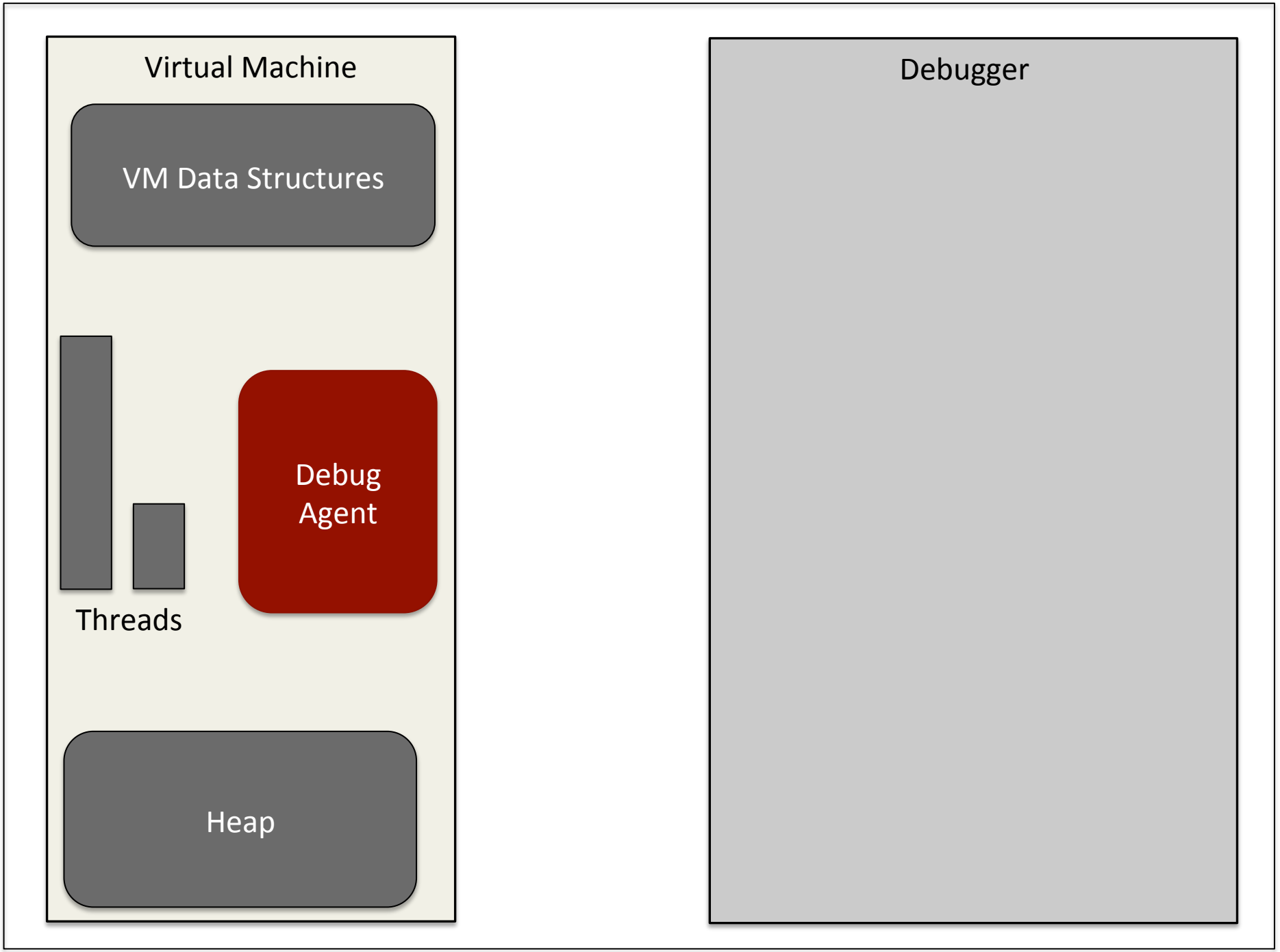

Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

Case Study: Debugging

- Build the external debug tool
- Build the JVMTI agent
  - May need a separate implementation per platform
- Register the agent for relevant capabilities
- Relay events to the external tool

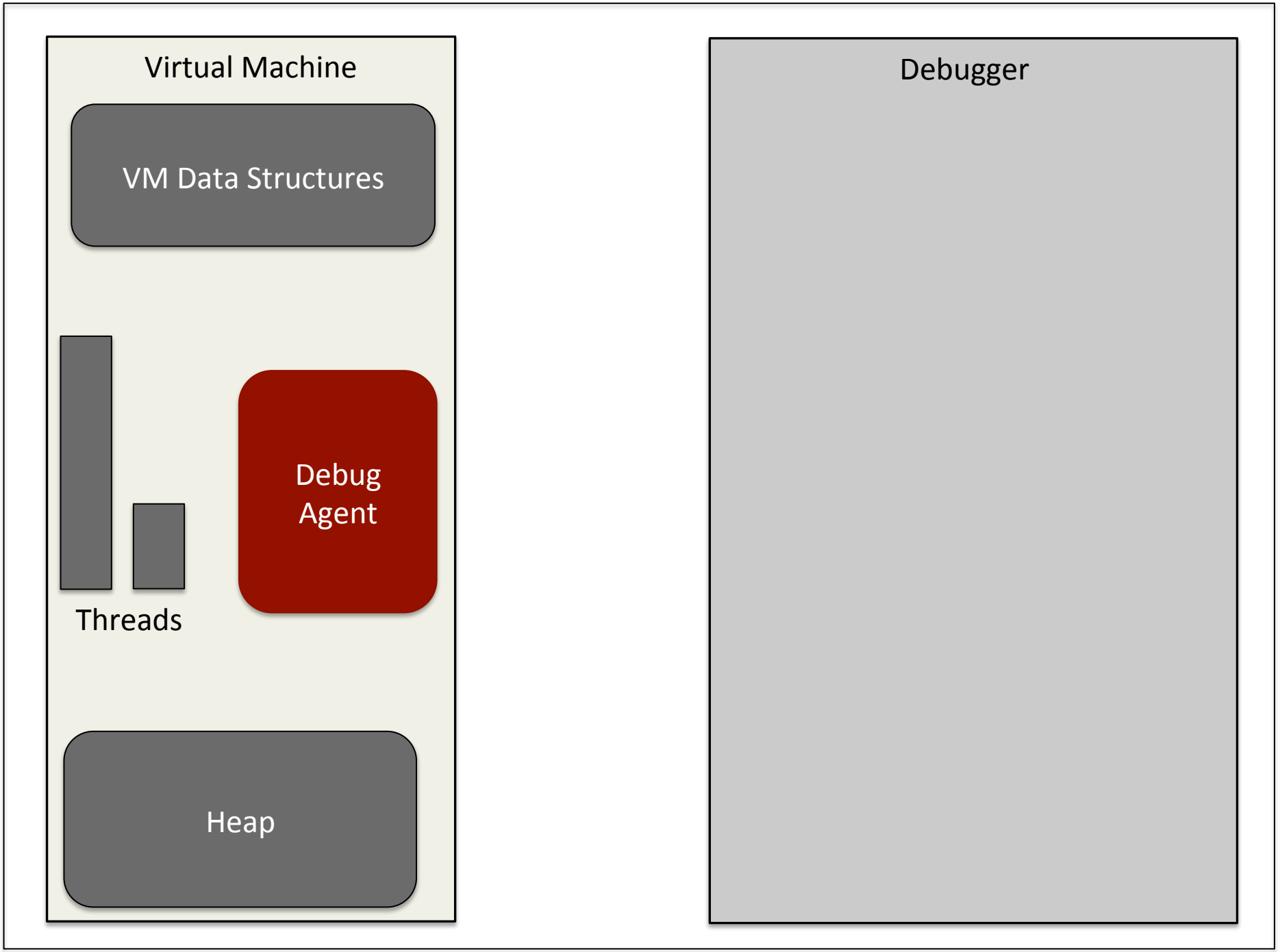

Virtual Machine
VM Data Structures
Threads
Heap

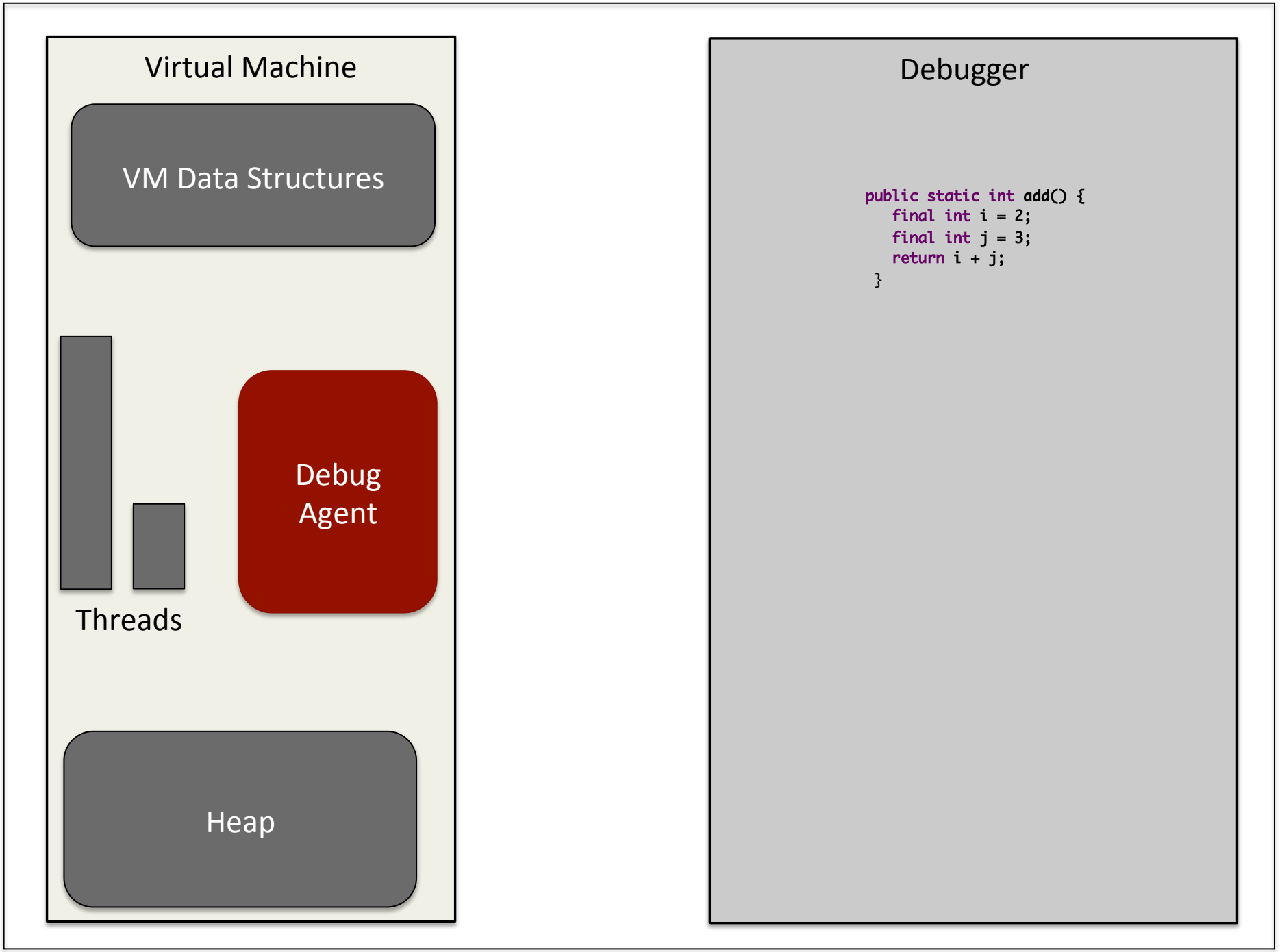

Virtual Machine
VM Data Structures
Threads
Heap
Debugger

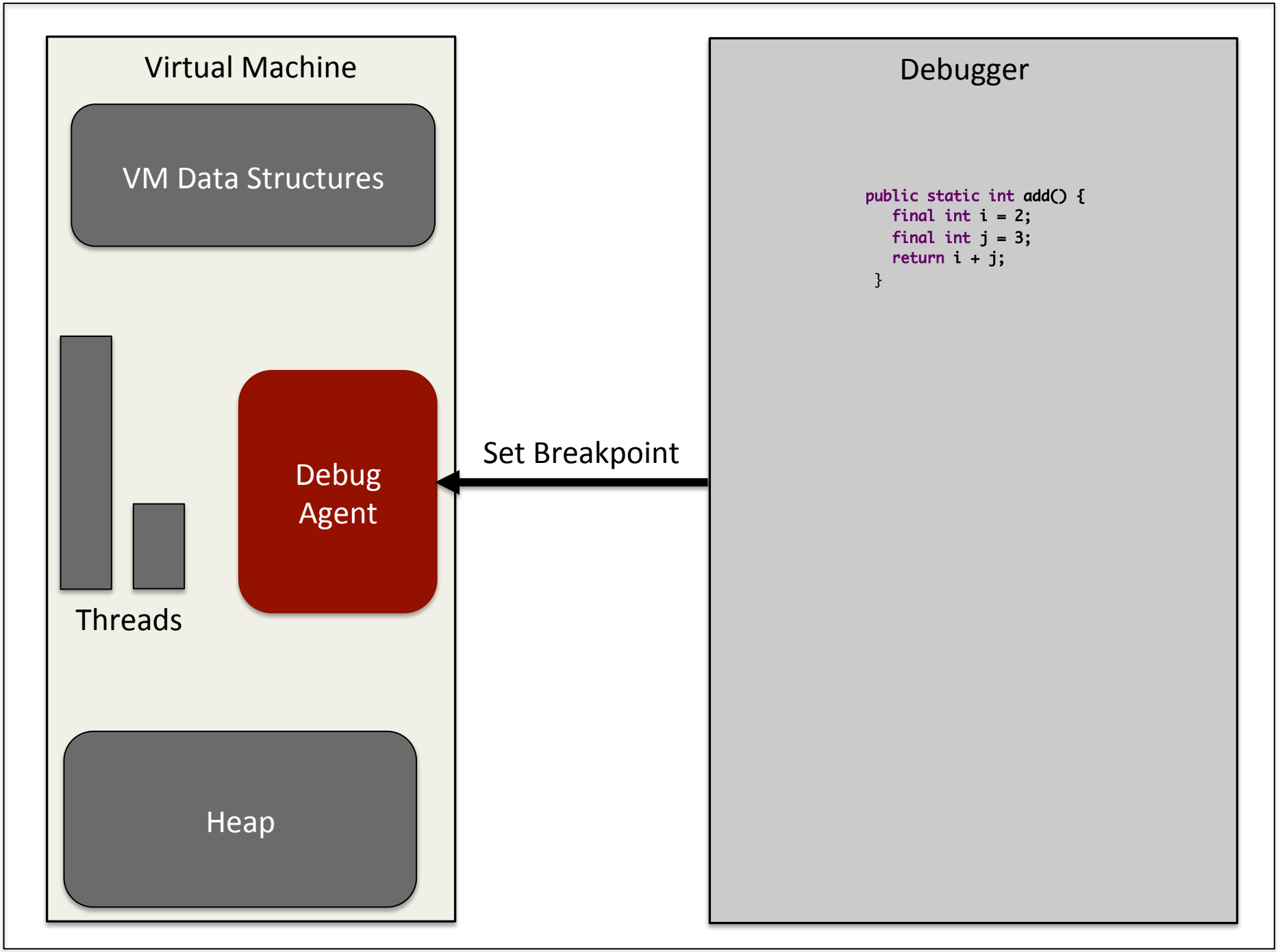

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger

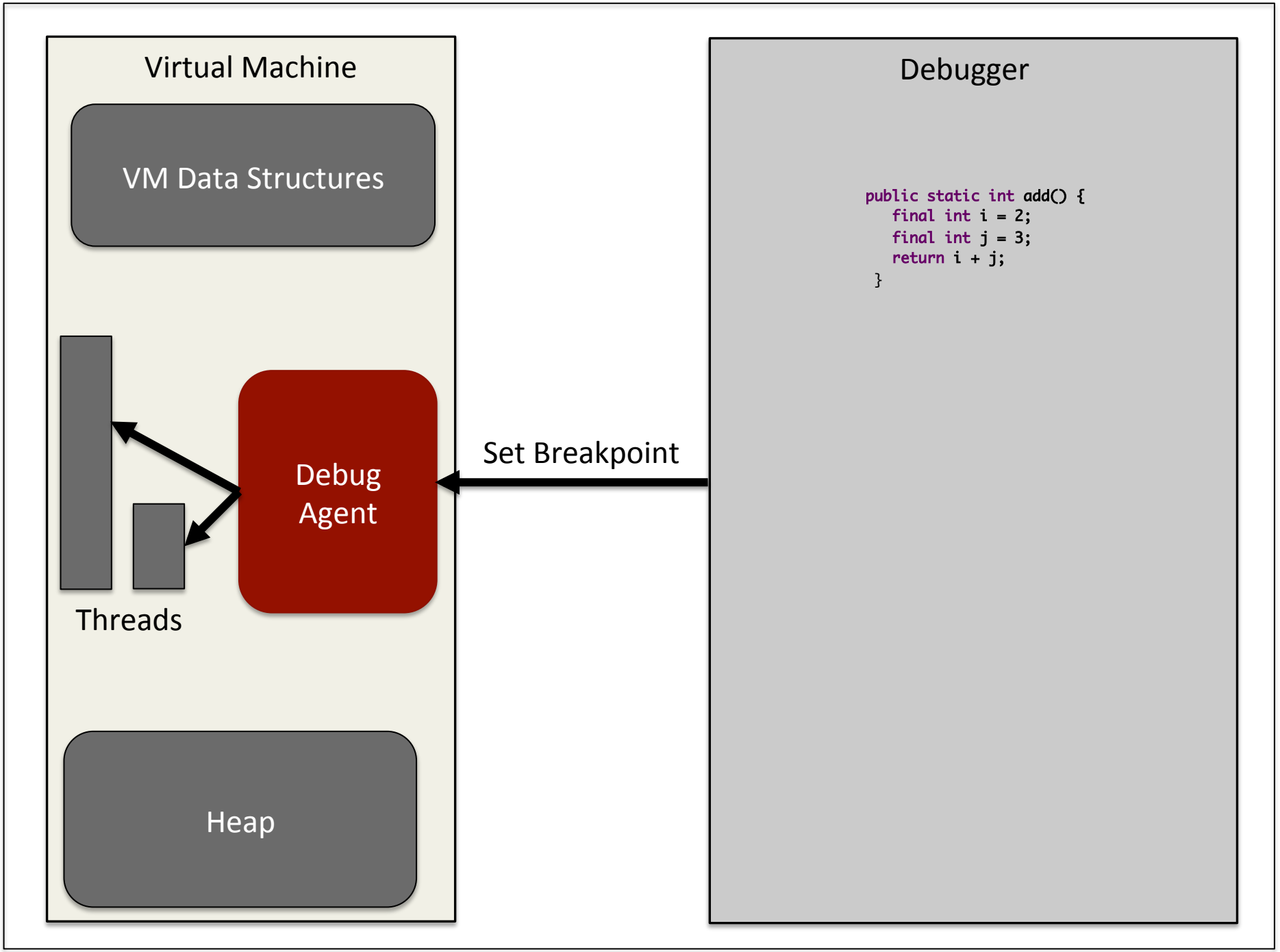
Capabilities

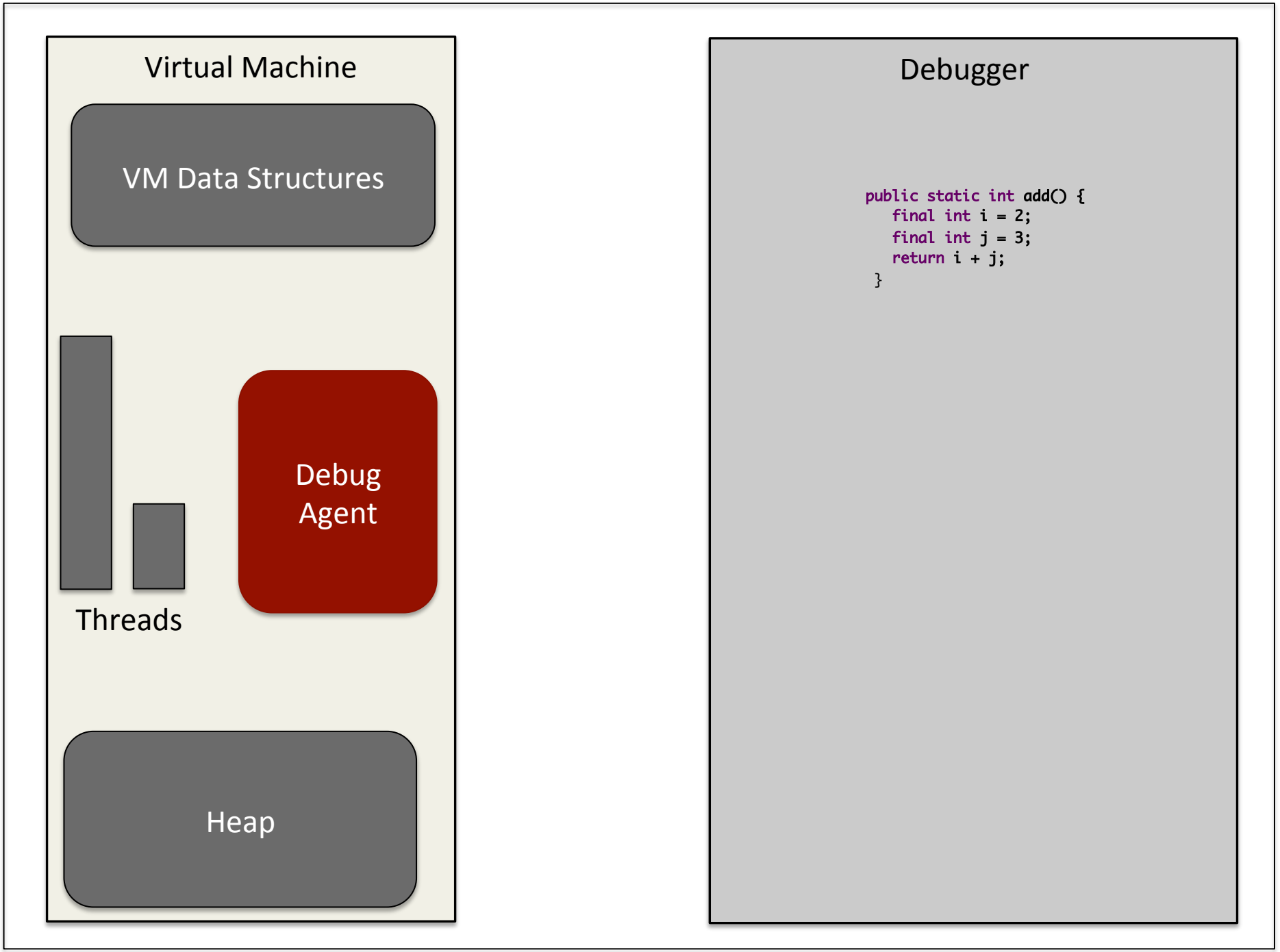
- Select the capabilities for the debugger
  - Receive exception events
  - Ability to tag objects
  - Ability to set breakpoints
  - Ability to signal threads
  - Ability to single-step through code
- Register the capabilities and callbacks at startup

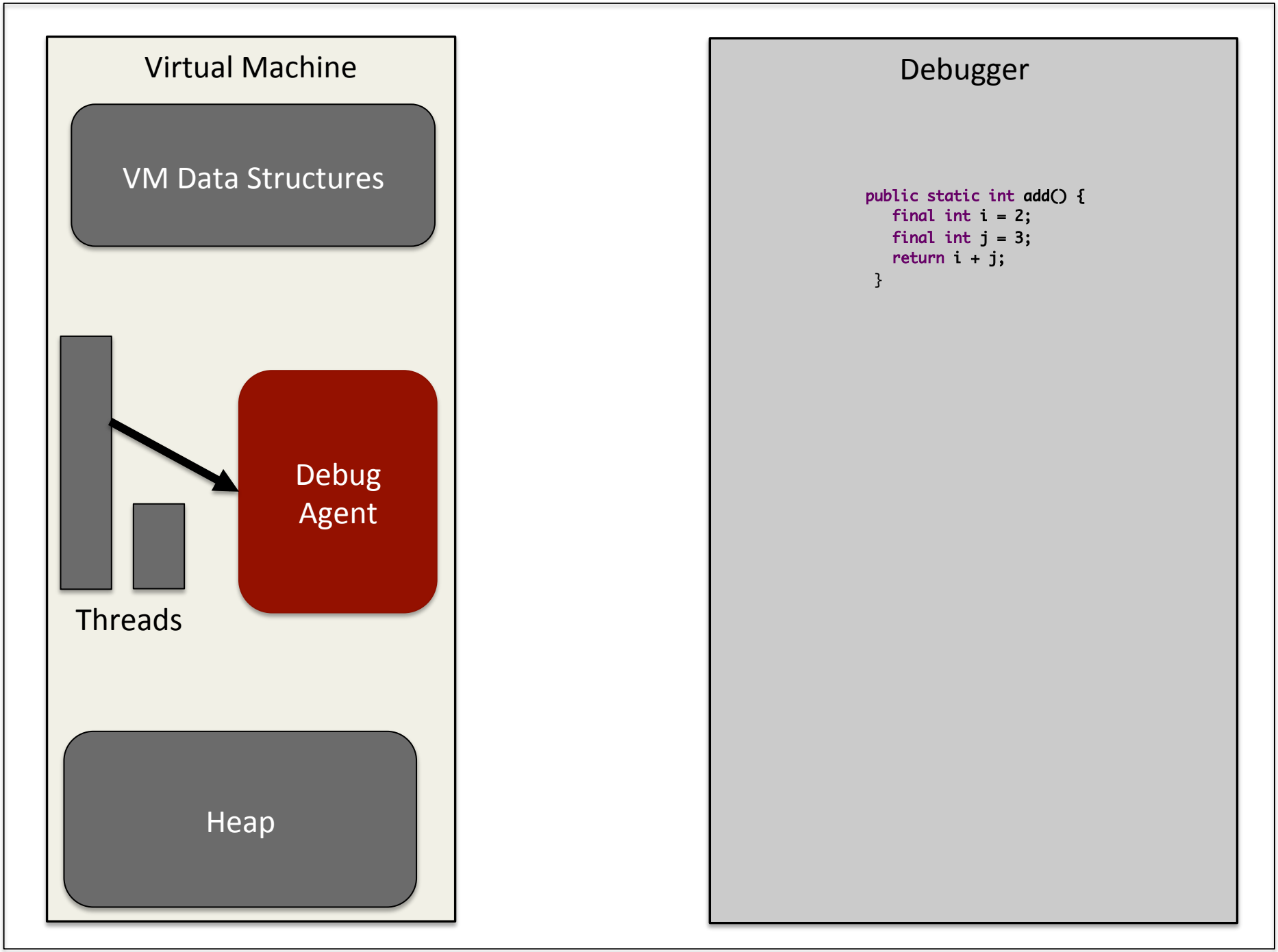

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger

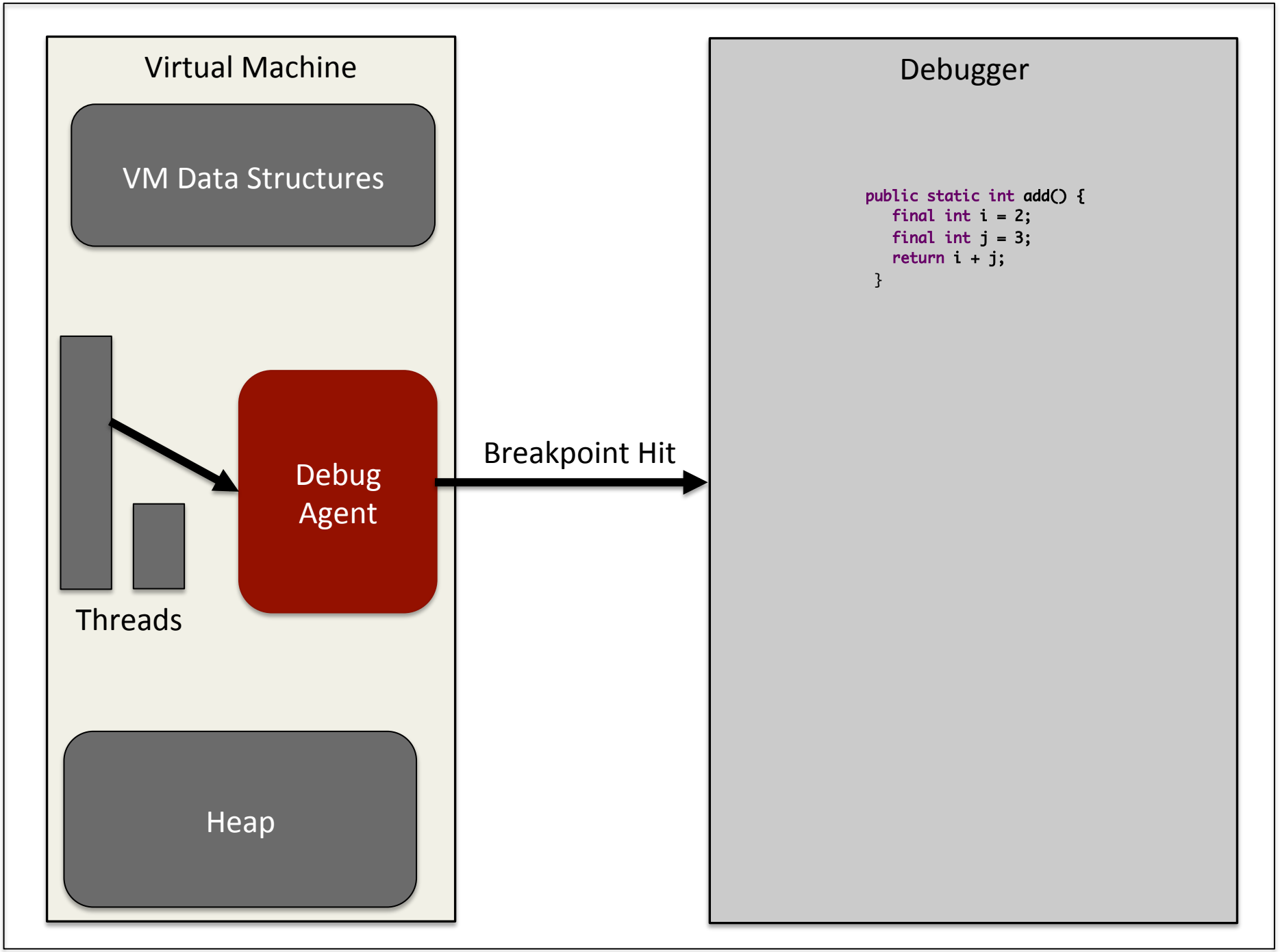

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

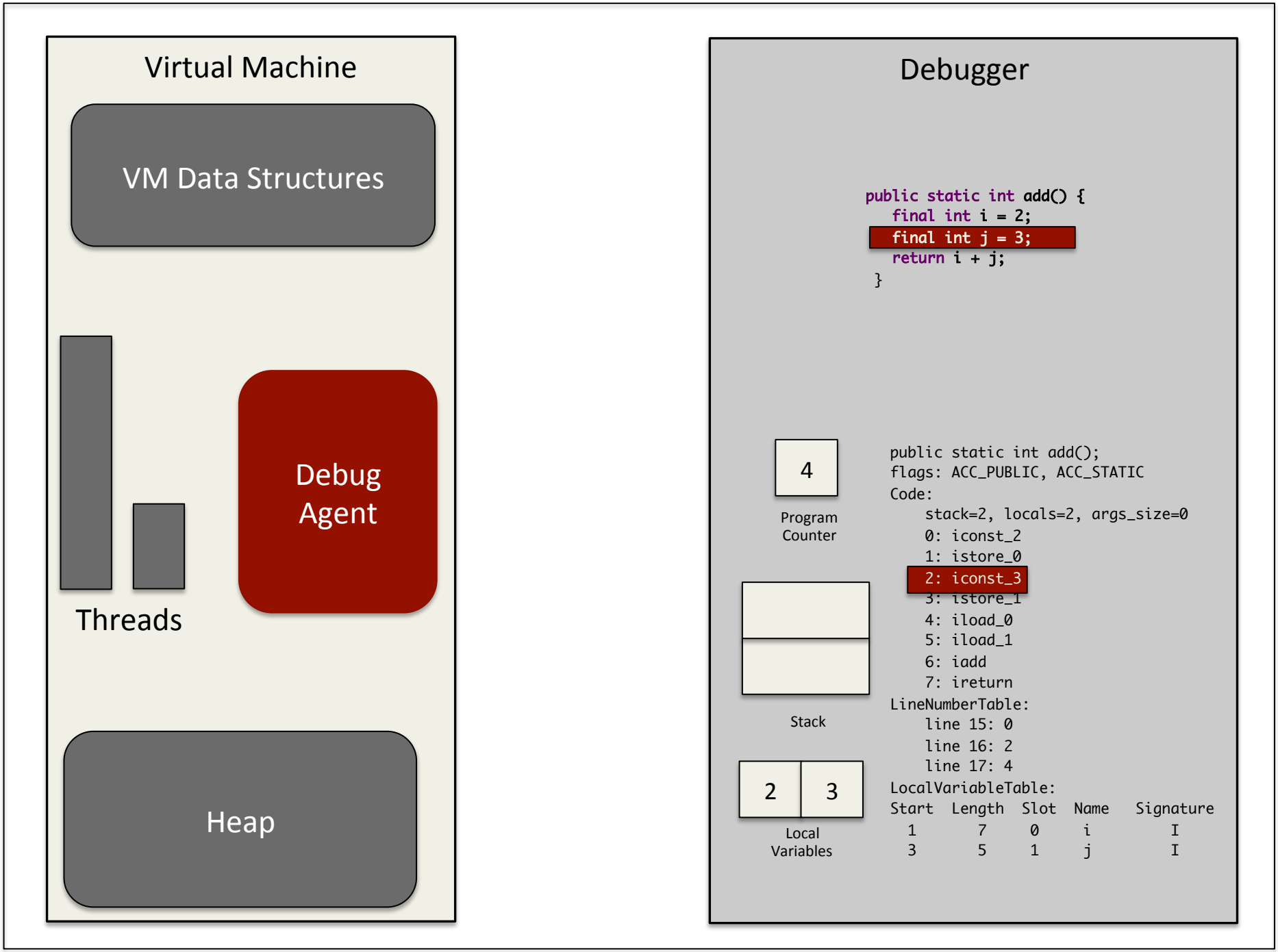

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Set Breakpoint
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

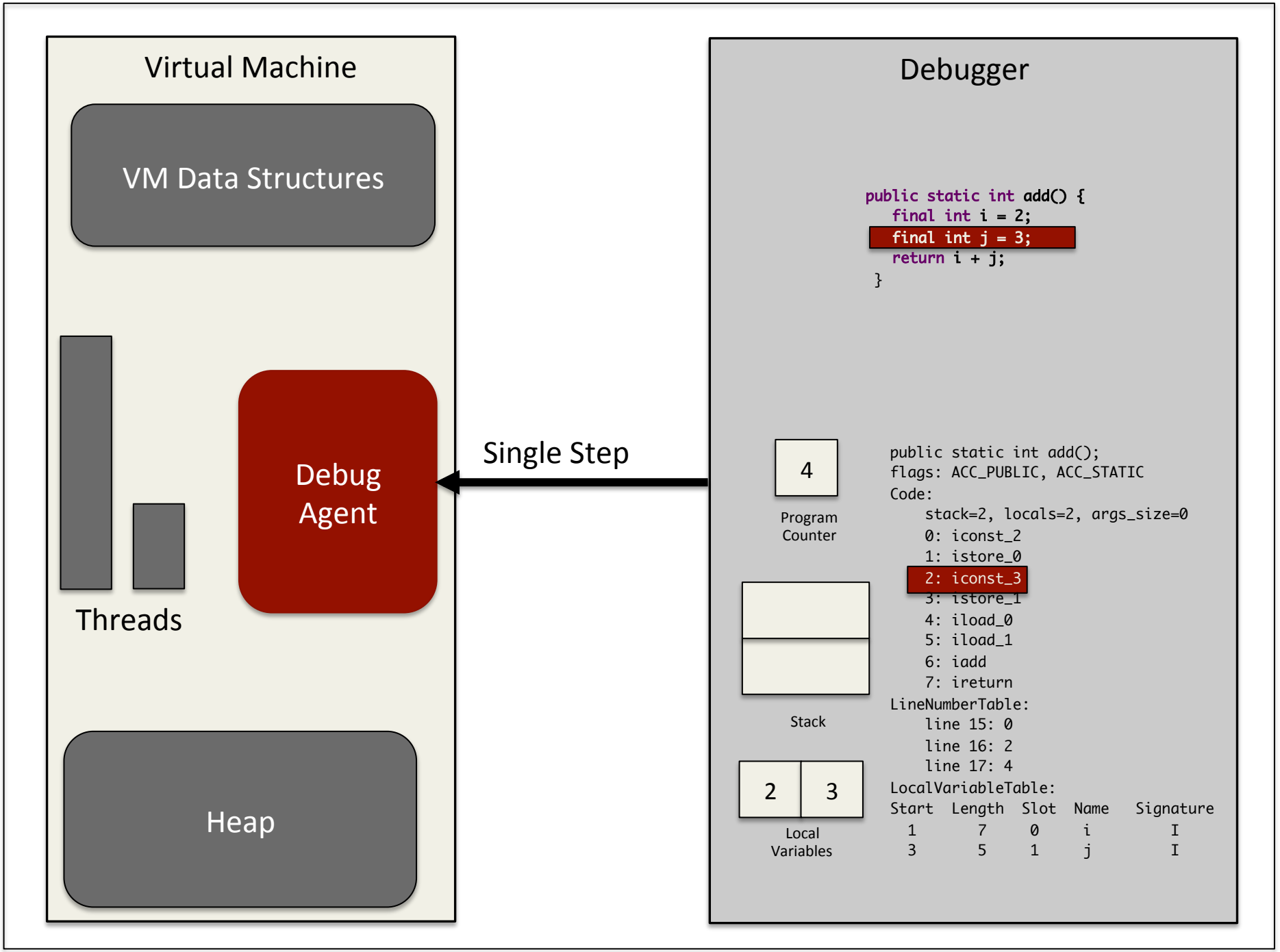

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Set Breakpoint
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

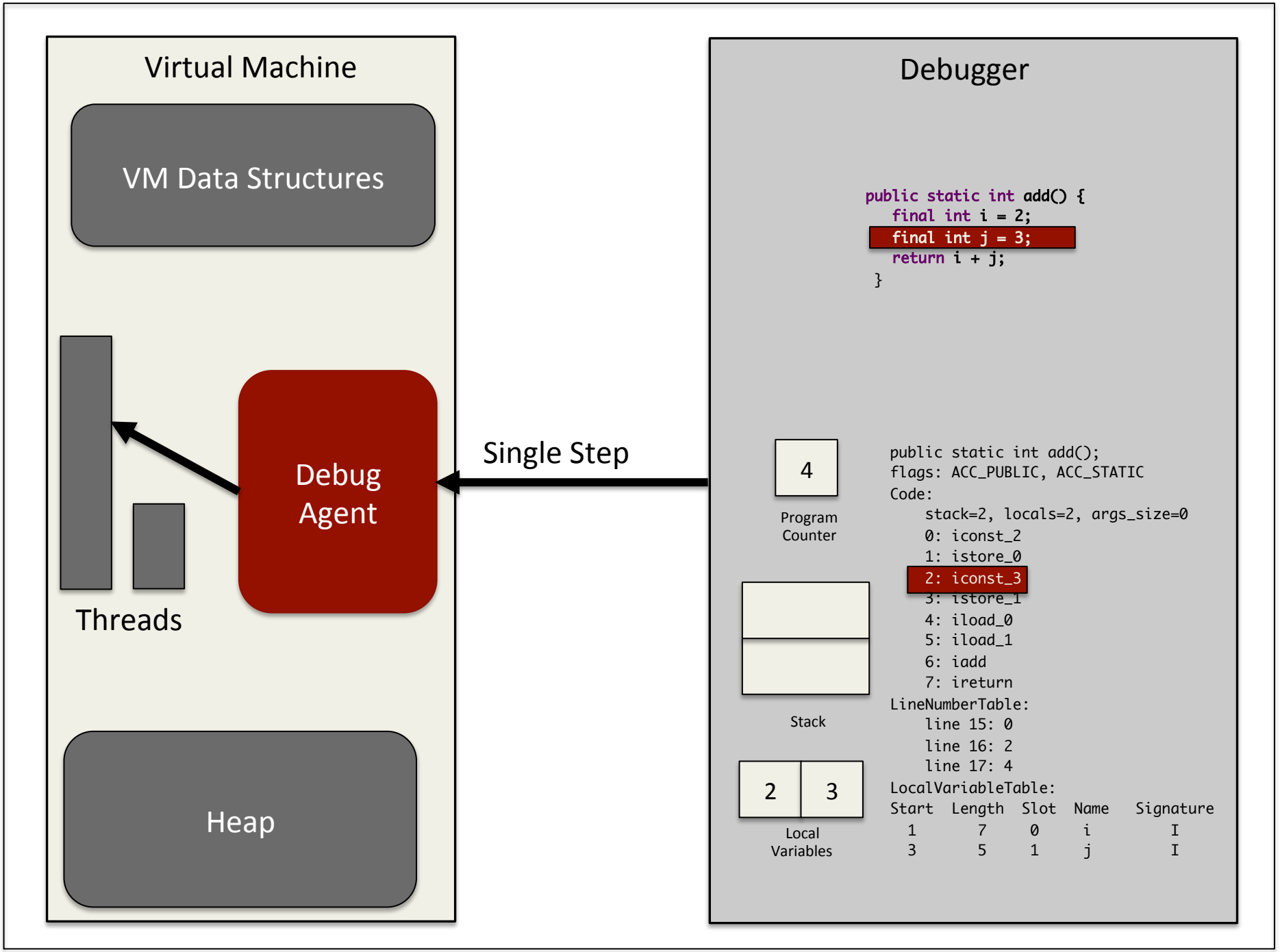

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

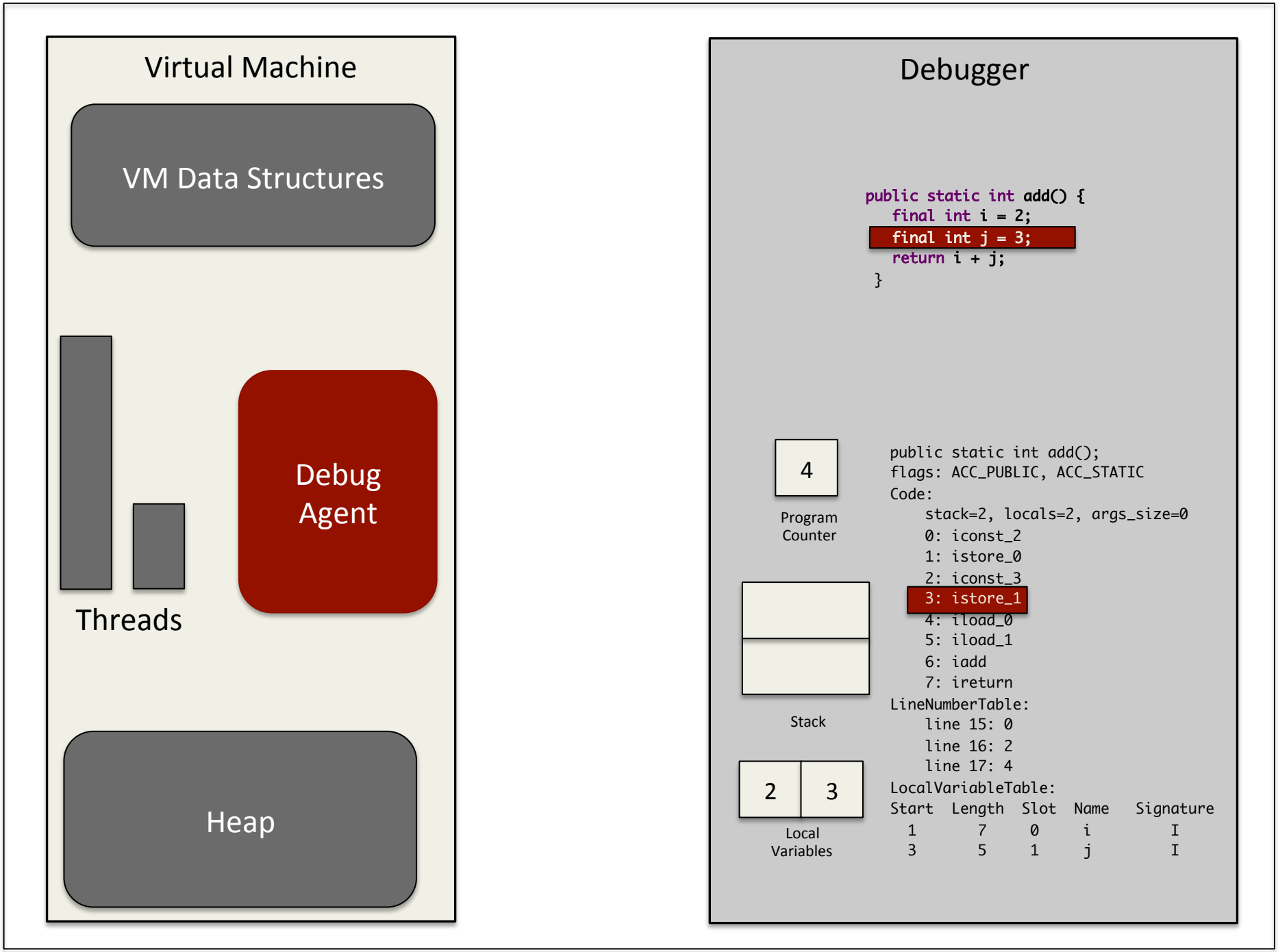

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

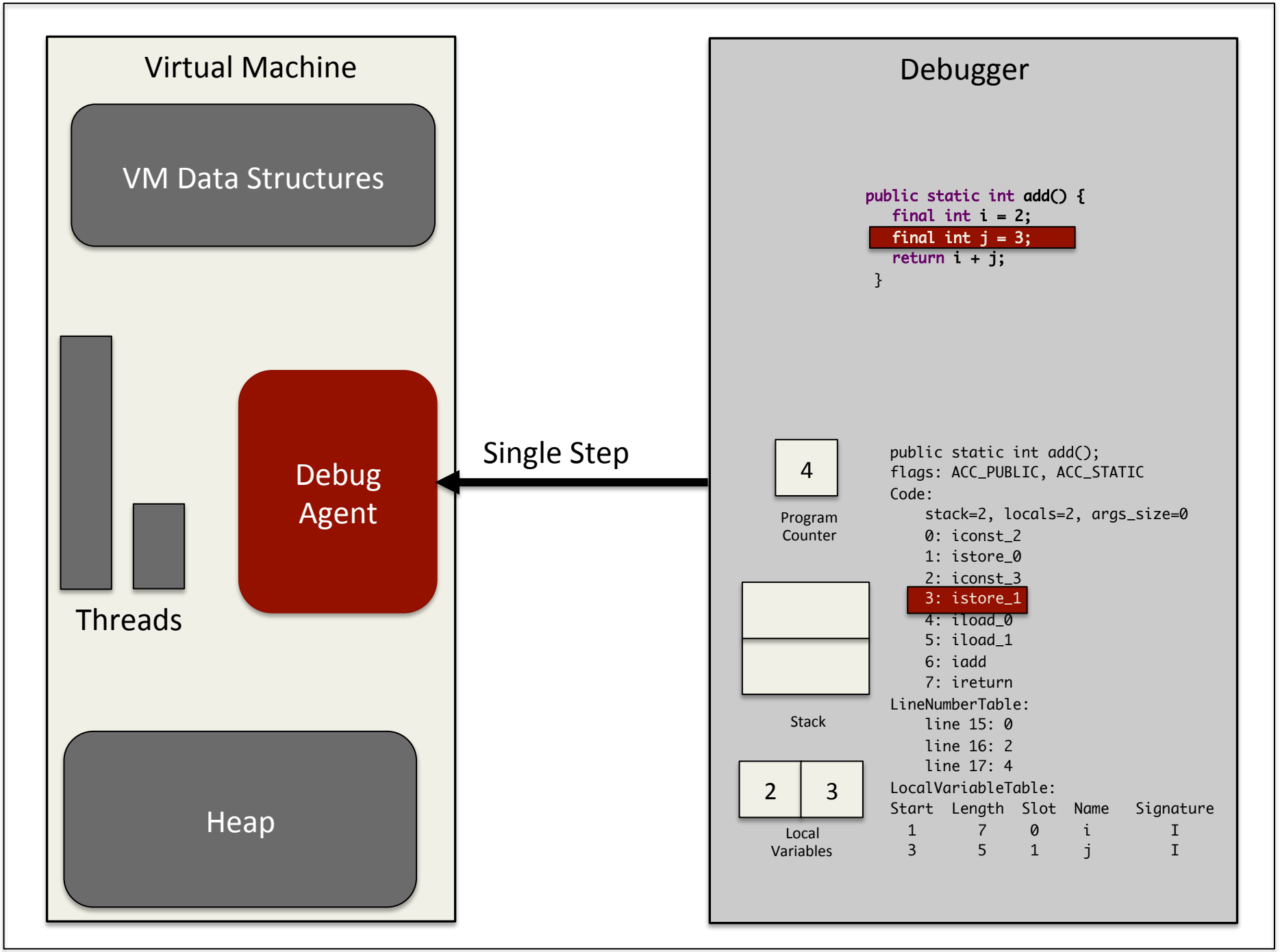

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Breakpoint Hit
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

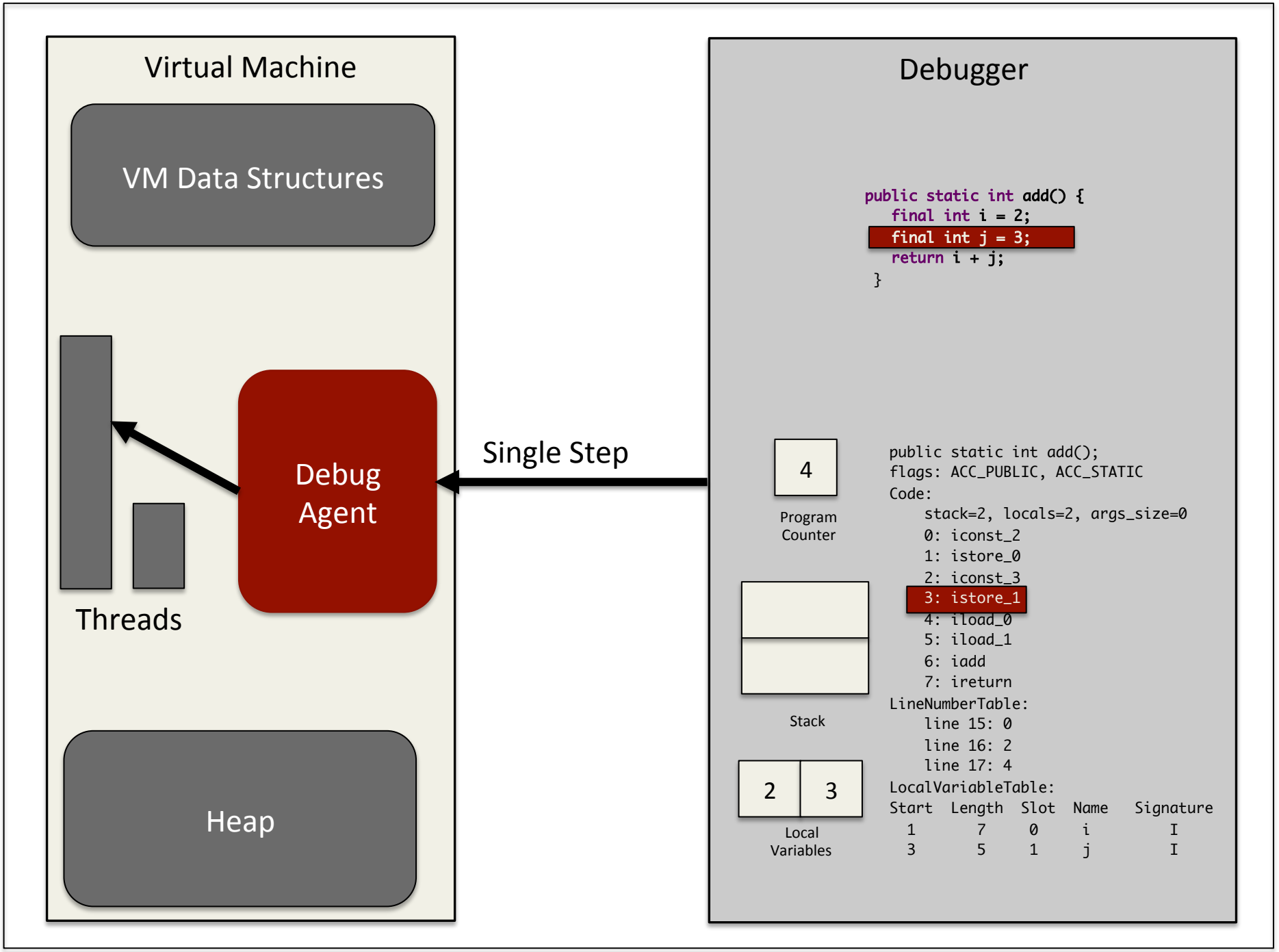

Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

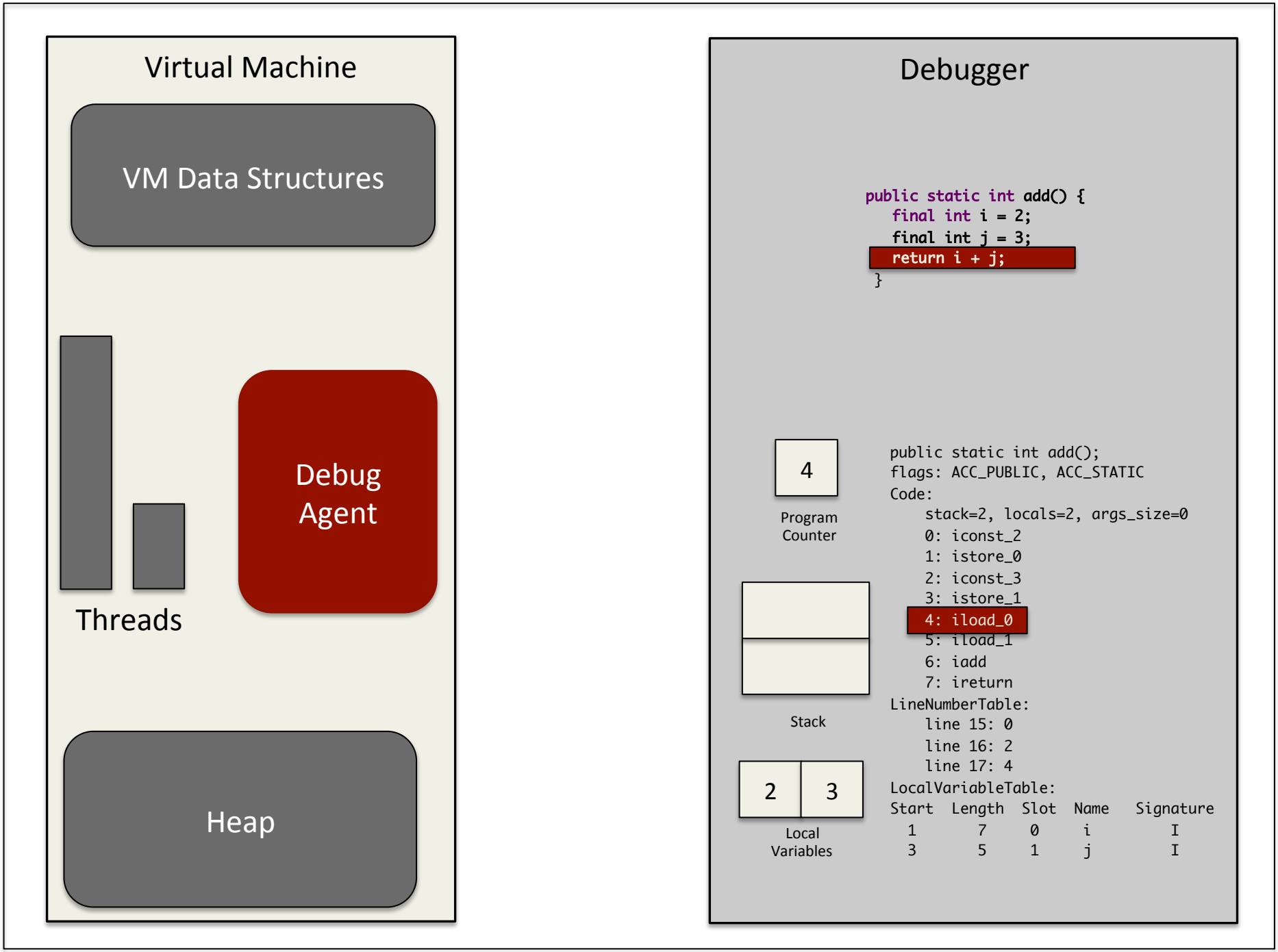

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Single Step
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

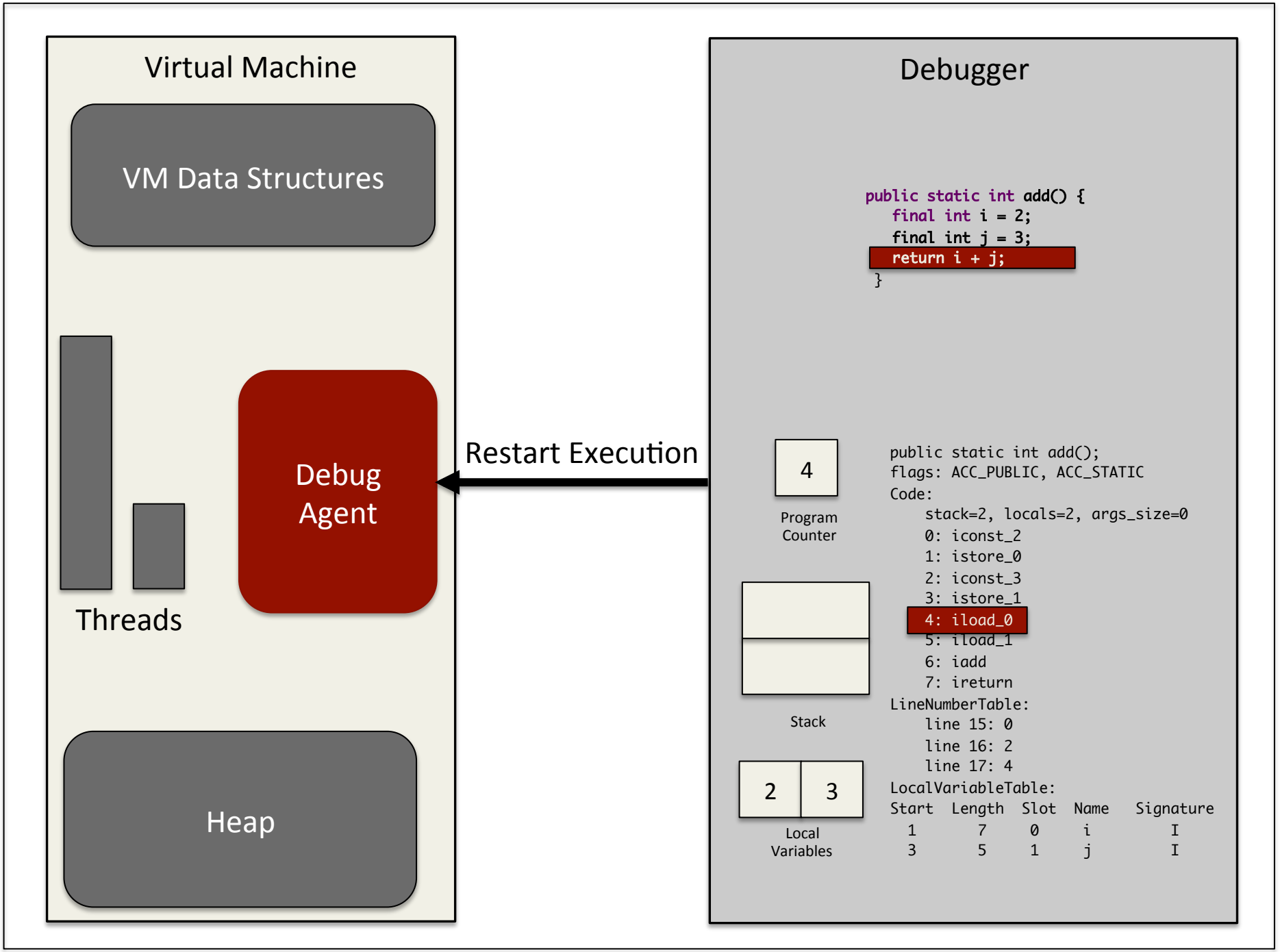

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Single Step
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

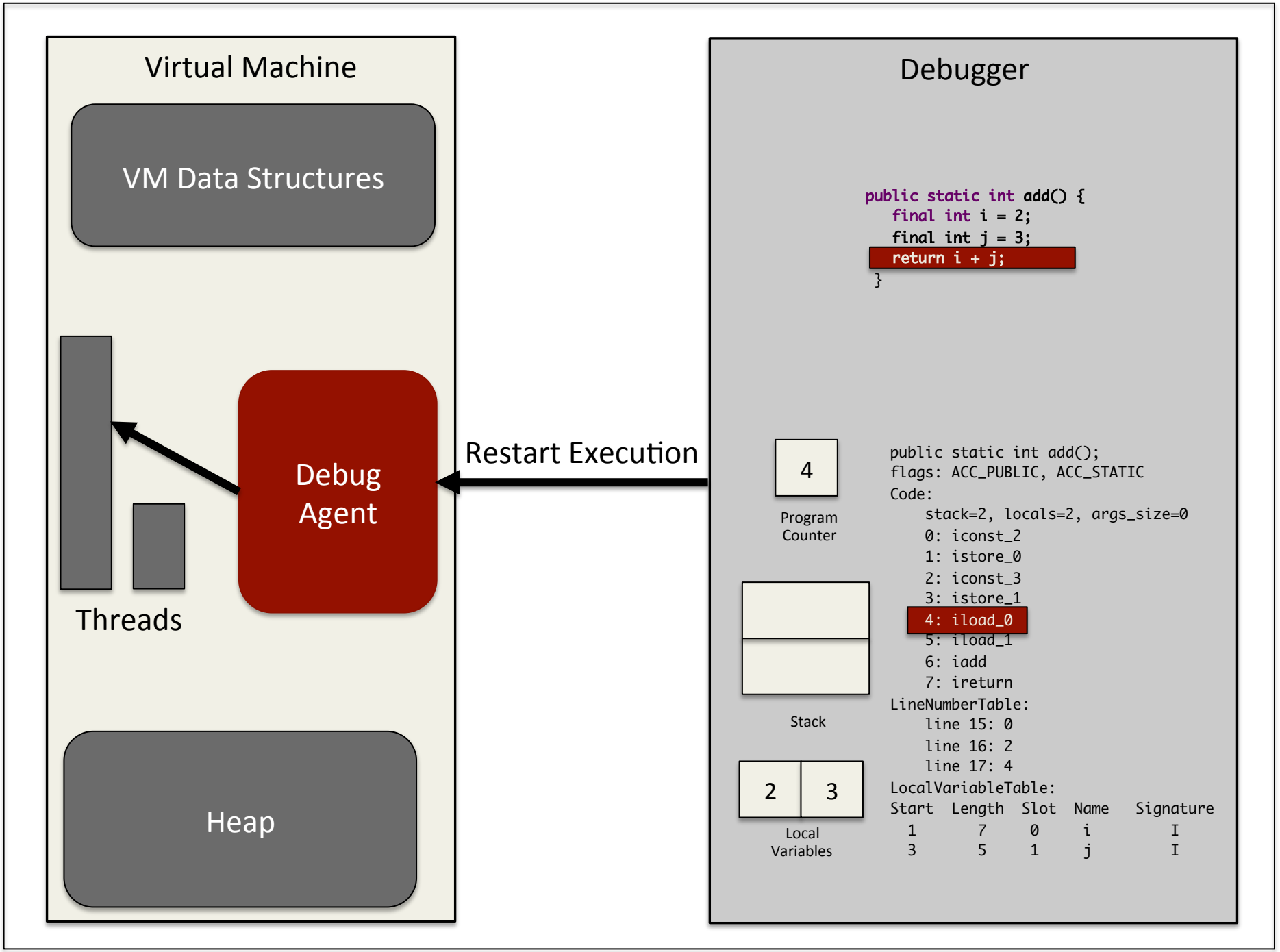

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

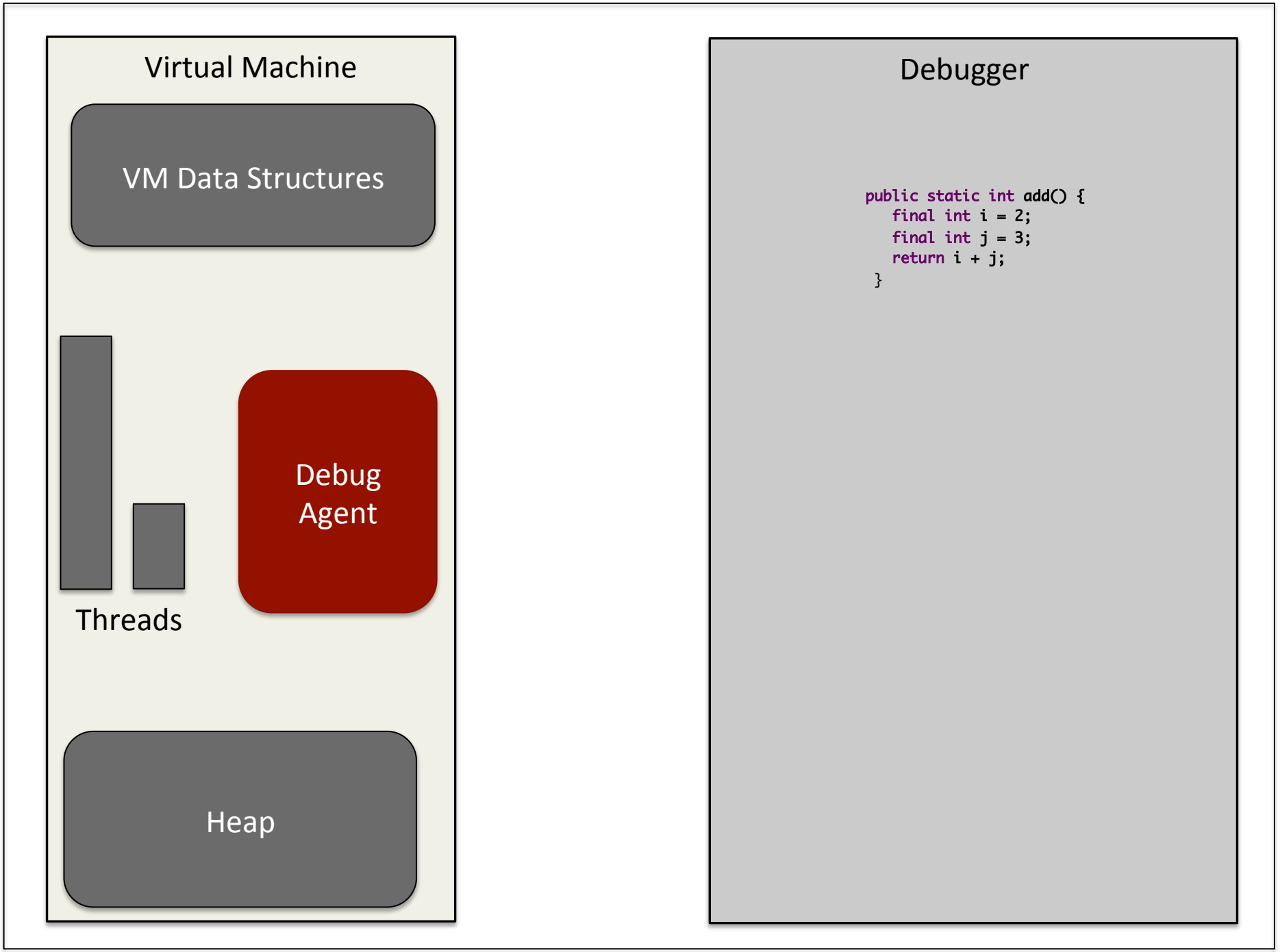

Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Single Step
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Single Step
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Restart Execution
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Threads
Debug Agent
Heap
Restart Execution
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}
4
Program Counter
Stack
2
3
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Virtual Machine
VM Data Structures
Debug Agent
Threads
Heap
Debugger
public static int add() {
final int i = 2;
final int j = 3;
return i + j;
}

Missing Events

- Not all interesting operations have events
  - Object allocation
  - Operand stack or local variable activity
- Interface allows for bytecode modification
  - Rewrite the class file before the VM loads it
  - Reload a class to insert new code
- Instrumentation happens during class loading
  - After loading but before linking
  - Rewritten bytecode is verified

Bytecode Instrumentation

- Can insert very specific data gathering
  - Generate conditionally events
  - May rely on static analysis
- Profiling code is pure Java bytecode
  - New bytecode becomes part of the program
  - JIT compiler optimizes profiling code
- No native code involved in runtime profiling

Java-Level Agent Data

- May now have two methods of profiling
  - Instrumented bytecode at the Java level
  - JVMTI agent works in native code
- May need communication between the two
  - Can use JNI to access Java-level data
- May output two streams to external tool
  - Less communication inside the VM
  - More complexity in the tool interface

Instrumentation Limitations

- Bytecode instrumentation best for small changes
  - Add a static call to profiling code
  - Log program state at key points during execution
- Hard to make more elaborate changes
  - Can only modify classes, not create them
  - Cross-class dependencies hard to manage
  - Instrumentation code normally written in C
- Custom classloaders are much more flexible

Bytecode Rewriting

- Modifying bytecode is trickier than it seems
  - Need to pass verification
  - Gets harder when changes are not local
- Bytecode indices are important
  - Jump instructions
  - Local variable, line number, exception tables
- Adding local variables can cause unexpected issues
  - Some bytecodes have indices built in

Object Allocation

- Normally access by bytecode instrumentation
  - Instrument new, newarray, anewarray instructions
- Some allocations not captured this way
  - Constant interning
  - Reflection
  - Native allocations
- Special event signaled in these cases

Method Entry and Exit

- Capability exists for tracking method calls
  - Method entry and exit events can be generated
  - Callback method determines if they are interesting
- Tracking all method calls can be expensive
  - Well-factored code should consist of small methods
  - Need to generate events on inlined code
- Can use bytecode instrumentation instead
  - Much more selective about what to track

Heap Monitoring

- Interface provides some heap traversal calls
  - Can opt to find only reachable objects
  - Traversal order is not guaranteed
- Heap objects can be tagged on creation
  - Associate a `jlong` value with an object
  - Lets the agent keep track of that object later

Profiling

- Design of the tool similar to the debugger

- Use a different set of capabilities
  - Get monitor status
  - Access heap data
  - Receive object allocation events
  - Receive method enter and exit events
  - Ability to tag objects
  - Ability to instrument bytecode

Profiler Performance

- Some of the capabilities are heavyweight
  - Tracing the heap
  - Tracking allocations
  - Tracing method enter and exit
- Profilers should minimize disruption
- Can decide up-front what to monitor
  - Separate runs to track heap and execution time
  - Tailor the capabilities that the agent requests

Other uses for Tool Interface

- Code coverage
  - Instrument all paths through a method
- Deadlock or livelock analysis
  - Pause threads and check monitor status
- VM monitoring
  - Correlate VM events to OS or hardware counters
- Extreme bytecode rewriting
  - Modify classes that otherwise can't be rewritten