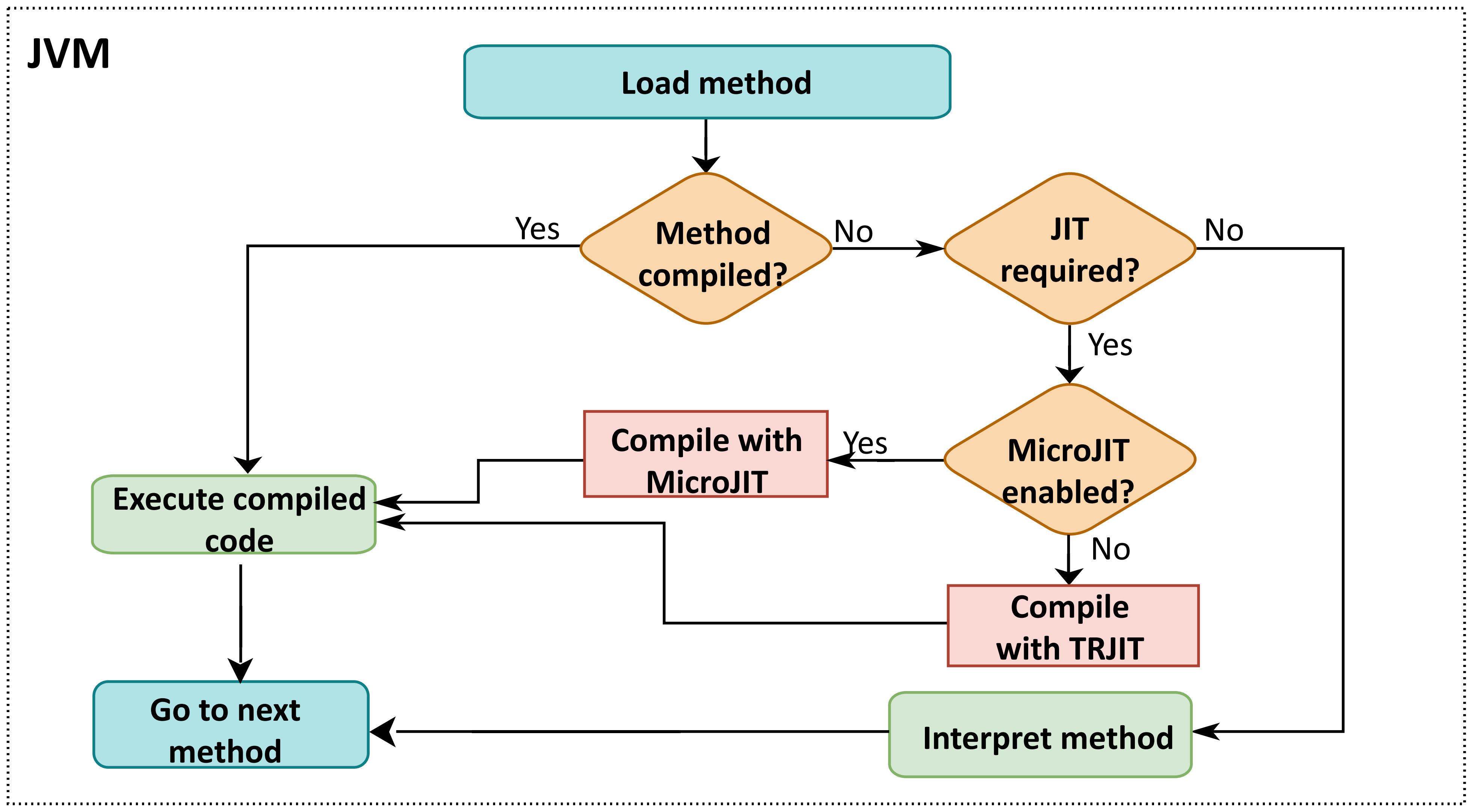
Implementation of Array Bytecodes in MicroJIT

Shubham Verma, Kenneth B. Kent
Faculty of Computer Science, University of New Brunswick
Marius Pirvu
IBM Canada
{sverma1, ken}@unb.ca
mpirvu@ca.ibm.com

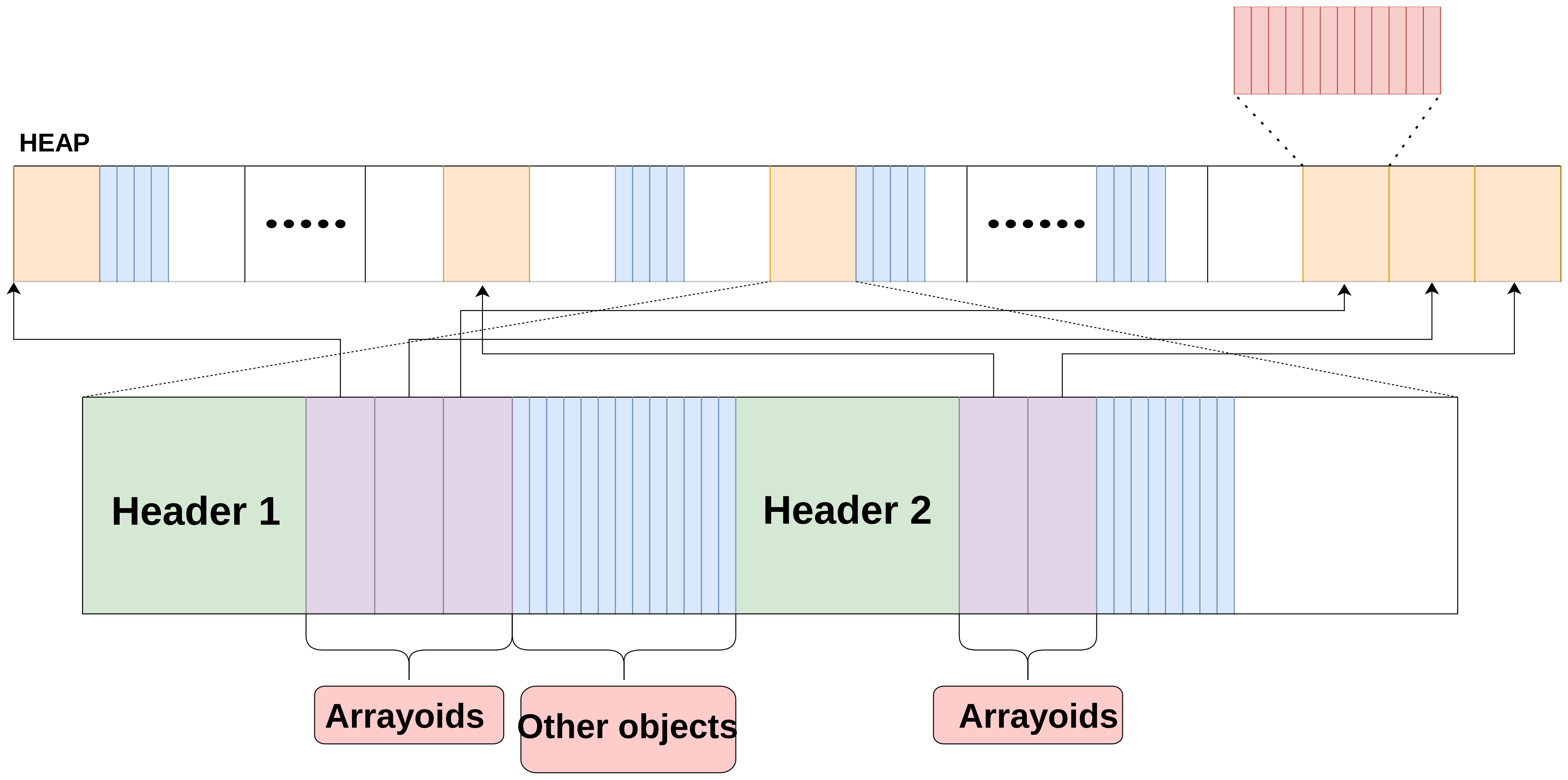
Background

- Java programs are executed on JVMs after being translated into bytecodes, which can be interpreted or compiled.
- JIT compilers boost the performance of interpreted Java programs on the otherwise slow JVM.
- Testarossa JIT (TRJIT) is the default optimizing compiler in Eclipse OpenJ9 JVM.



Arrays and Arraylets

- Our goal is to enable non-contiguous memory allocation in MicroJIT arrays.
- Small arrays are allocated contiguously; large arrays are split into arraylets and allocated across regions.
- Arraylets are composed of a metadata spine and leaves, with arrayoids serving as pointers to the leaves.



MicroJIT

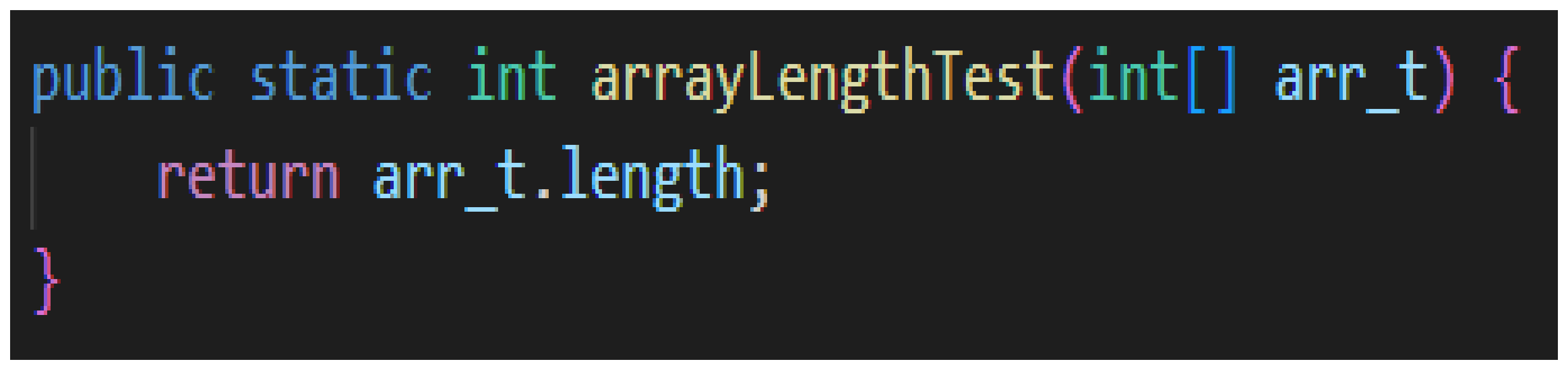
- TRJIT provides optimized code, it increases overhead during runtime and start-up times.
- MicroJIT is a lightweight JIT compiler aimed at faster start-up times, with a template-based structure that eliminates intermediary phases, reducing memory footprint.

Motivation

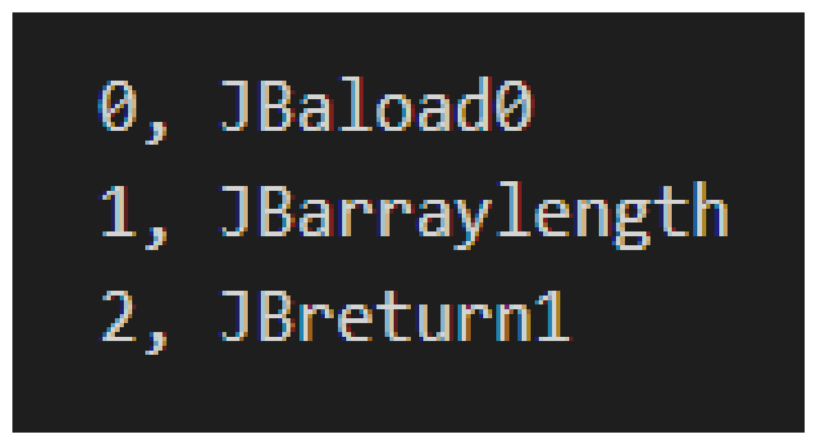
- MicroJIT lacks support for array bytecodes, requiring switching back to the interpreter for execution, which hinders performance. Expanding bytecode support can improve JVM performance.

Implementation

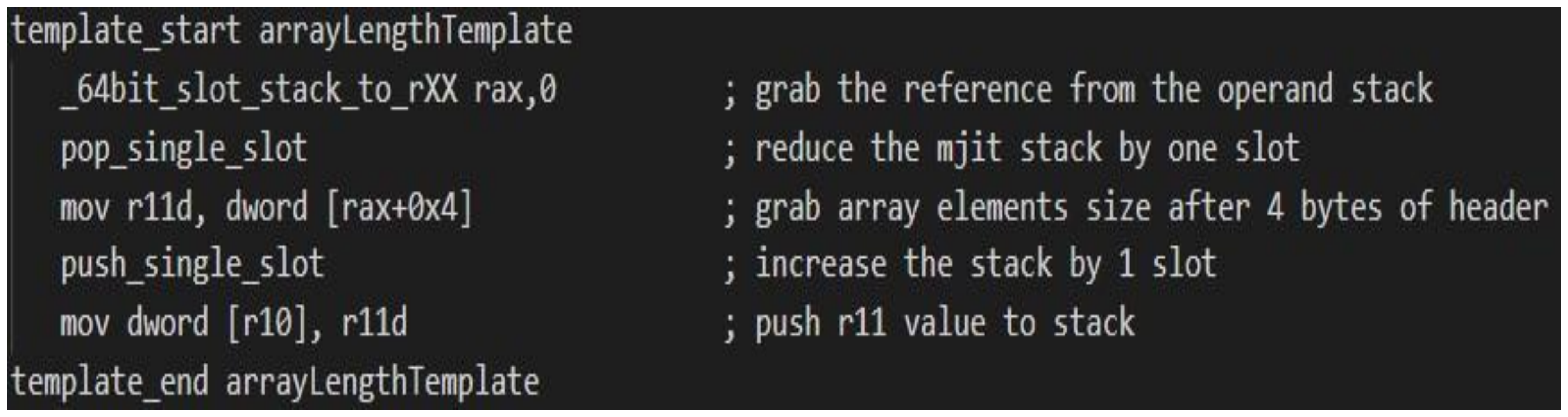
- Templates for different array bytecodes consist of predefined assembly sequence, written in Netwide Assembler (NASM).
- Continuous arrays use fast path; Discontinuous use helper function that allocates memory in different regions.
- To check the correctness of the bytecodes, Junit regression test framework is used.



```
public static int arrayLengthTest(int[] arr_t) {
    return arr_t.length;
}
```



```
0, JBaload0
1, JBarraylength
2, JBreturn1
```



```
template_start arrayLengthTemplate
   _64bit_slot_stack_to_rXX rax,0          ; grab the reference from the operand stack
   pop_single_slot                         ; reduce the mjit stack by one slot
   mov r11d, dword [rax+0x4]               ; grab array elements size after 4 bytes of header
   push_single_slot                        ; increase the stack by 1 slot
   mov dword [r10], r11d                   ; push r11 value to stack
template_end arrayLengthTemplate
```

Evaluation Plan

- We intend to evaluate and compare the performance of the JVM, featuring array implementation in MicroJIT, against the interpreter and TRJIT, both with and without optimization.
- Our primary emphasis is on evaluating key metrics, including compile time, execution time, and memory footprint.

CAS–Atlantic