

# HOTPATH VM

An Effective JIT Compiler for Resource-constrained Devices

based on the paper by Andreas Gal, Christian W. Probst and Michael Franz, VEE '06

# INTRODUCTION

# INTRODUCTION

- Most important factor: speed

# INTRODUCTION

- Most important factor: speed
- Overlooked: power consumption, memory footprint

# INTRODUCTION

- Most important factor: speed
- Overlooked: power consumption, memory footprint
- Embedded JIT compilers (simpler algorithms)

JAMVVM

# JAMVM

- GNU classpath-based

# JAMVM

- GNU classpath-based
- Own intermediate representation

# JAMVM

- GNU classpath-based
- Own intermediate representation
- HotpathVM as add-on

# JAMVM

- GNU classpath-based
- Own intermediate representation
- HotpathVM as add-on
  - Integrated w/ 20 lines of code

# HOTPATHVM

# HOTPATHVM

- 150 kB memory footprint

# HOTPATHVM

- 150 kB memory footprint
- Searching for hot code by tracing

# HOTPATHVM

- 150 kB memory footprint
- Searching for hot code by tracing
  - Cold code for interpreter

# HOTPATHVM

- 150 kB memory footprint
- Searching for hot code by tracing
  - Cold code for interpreter
- Optimizing and compiling traces

# TRACE SELECTION

# TRACE SELECTION

- Problem: bytecode is not purely sequential

# TRACE SELECTION

- Problem: bytecode is not purely sequential
- Solution: Software Trace Scheduling

# TRACE SELECTION

- Problem: bytecode is not purely sequential
- Solution: Software Trace Scheduling
- Limited to loops

# IDENTIFYING LOOP HEADERS

# IDENTIFYING LOOP HEADERS

- Backward jump → record destination

# IDENTIFYING LOOP HEADERS

- Backward jump → record destination
  - Assumption: bytecode is forward

# IDENTIFYING LOOP HEADERS

- Backward jump → record destination
  - Assumption: bytecode is forward
- Counter for backward jumps

# IDENTIFYING LOOP HEADERS

- Backward jump → record destination
  - Assumption: bytecode is forward
- Counter for backward jumps
  - Saved in intermediate representation

# IDENTIFYING LOOP HEADERS

- Backward jump → record destination
  - Assumption: bytecode is forward
- Counter for backward jumps
  - Saved in intermediate representation
- Record after threshold is exceeded

# TRACE EXAMPLE

# TRACE EXAMPLE

```
public static void main(String args[]) {  
    int i, k = 0;  
    for (i = 0; i < 1000, ++i)  
        ++k;  
    System.out.println(k);  
}
```

# TRACE EXAMPLE

```
public static void main(String args[]) {  
    int i, k = 0;  
    for (i = 0; i < 1000, ++i)  
        ++k;  
    System.out.println(k);  
}
```

```
    iconst_0  
    istore_2  
    iconst_0  
    istore_1  
A: iload_1  
    sipush 1000  
    if_icmpge B  
    iinc 2,1  
    iinc 1,1  
    goto A  
B: getstatic System.out  
    iload_2  
    invokevirtual println(int)  
    return
```

# TRACE EXAMPLE

```
public static void main(String args[]) {  
    int i, k = 0;  
    for (i = 0; i < 1000, ++i) hotspot  
        ++k;  
    System.out.println(k);  
}
```

```
    iconst_0  
    istore_2  
    iconst_0  
    istore_1  
A: iload_1  
   sipush 1000  
   if_icmpge B  
   iinc 2,1  
   iinc 1,1  
   goto A  
B: getstatic System.out  
   iload_2  
   invokevirtual println(int)  
   return
```

# TRACE EXAMPLE

```
public static void main(String args[]) {  
    int i, k = 0;  
    for (i = 0; i < 1000, ++i)  
        ++k;  
    System.out.println(k);  
}
```

*hotspot*

```
iconst_0  
istore_2  
iconst_0  
istore_1
```

```
A: iload_1  
   sipush 1000  
   if_icmpge B  
   iinc 2,1  
   iinc 1,1  
   goto A
```

*hotpath*

```
B: getstatic System.out  
   iload_2  
   invokevirtual println(int)  
   return
```

# RECORDING TRACES

# RECORDING TRACES

- Meta block after each instruction

# RECORDING TRACES

- Meta block after each instruction
- Second stop-loss threshold

# RECORDING TRACES

- Meta block after each instruction
- Second stop-loss threshold
  - No overhead without tracer

# RECORDING TRACES

- Meta block after each instruction
- Second stop-loss threshold
  - No overhead without tracer
- Guards for branches

# GUARDS

# GUARDS

- Ensures following the hotpath

# GUARDS

- Ensures following the hotpath
  - Returns control to VM on failure

# GUARDS

- Ensures following the hotpath
  - Returns control to VM on failure
- Reset values from intermediate representation

# BRANCHES

# BRANCHES

- Multiple hot paths in loop

# BRANCHES

- Multiple hot paths in loop
- Lazy recording

# BRANCHES

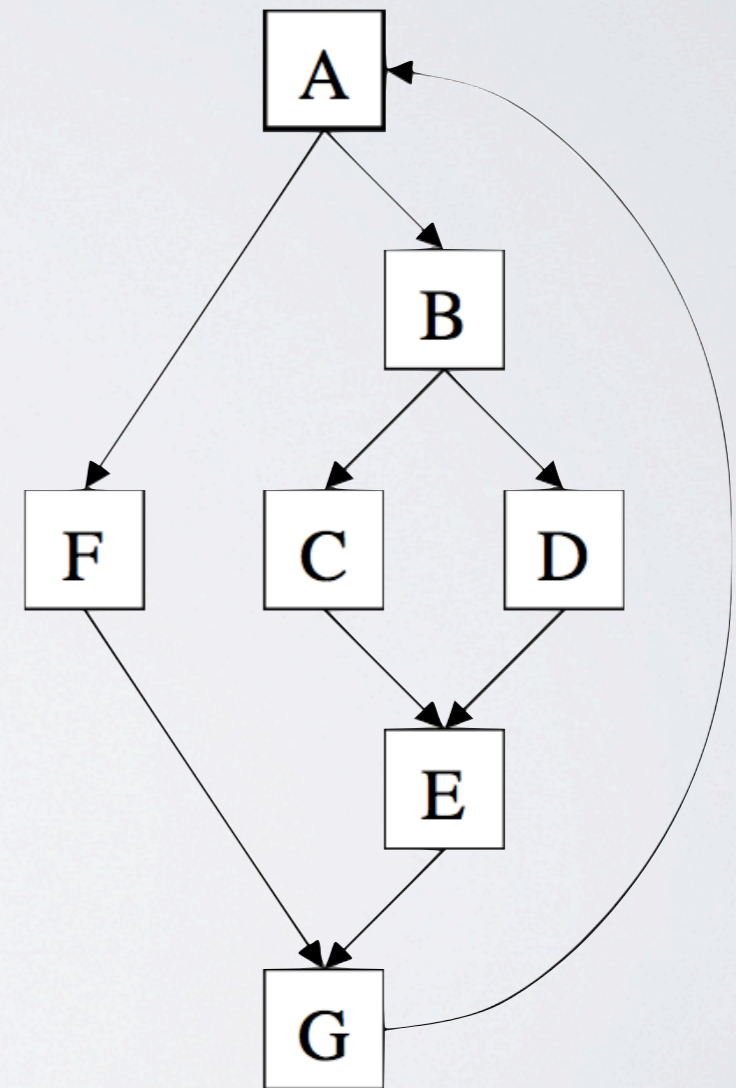
- Multiple hot paths in loop
- Lazy recording
- Secondary traces (like Dynamo)

# BRANCHES

- Multiple hot paths in loop
- Lazy recording
- Secondary traces (like Dynamo)
  - Updates guard instruction

# BRANCHES

- Multiple hot paths in loop
- Lazy recording
- Secondary traces (like Dynamo)
  - Updates guard instruction



# STOP CONDITIONS

# STOP CONDITIONS

- Successful recording

# STOP CONDITIONS

- Successful recording
- Aborted recording on rare events

# STOP CONDITIONS

- Successful recording
- Aborted recording on rare events
  - Exceptions

# STOP CONDITIONS

- Successful recording
- Aborted recording on rare events
  - Exceptions
  - Native method invocation

# COMPILING TRACES

# COMPILING TRACES

- Compiling complete traces only

# COMPILING TRACES

- Compiling complete traces only
- Assumption: trace is executed repeatedly

# COMPILING TRACES

- Compiling complete traces only
- Assumption: trace is executed repeatedly

# COMPILING TRACES

- Compiling complete traces only
- Assumption: trace is executed repeatedly
- Stack Deconstruction

# COMPILING TRACES

- Compiling complete traces only
- Assumption: trace is executed repeatedly
- Stack Deconstruction
- Code Analysis

# COMPILING TRACES

- Compiling complete traces only
- Assumption: trace is executed repeatedly
- Stack Deconstruction
- Code Analysis
- Code Generation

# STACK DECONSTRUCTION

# STACK DECONSTRUCTION

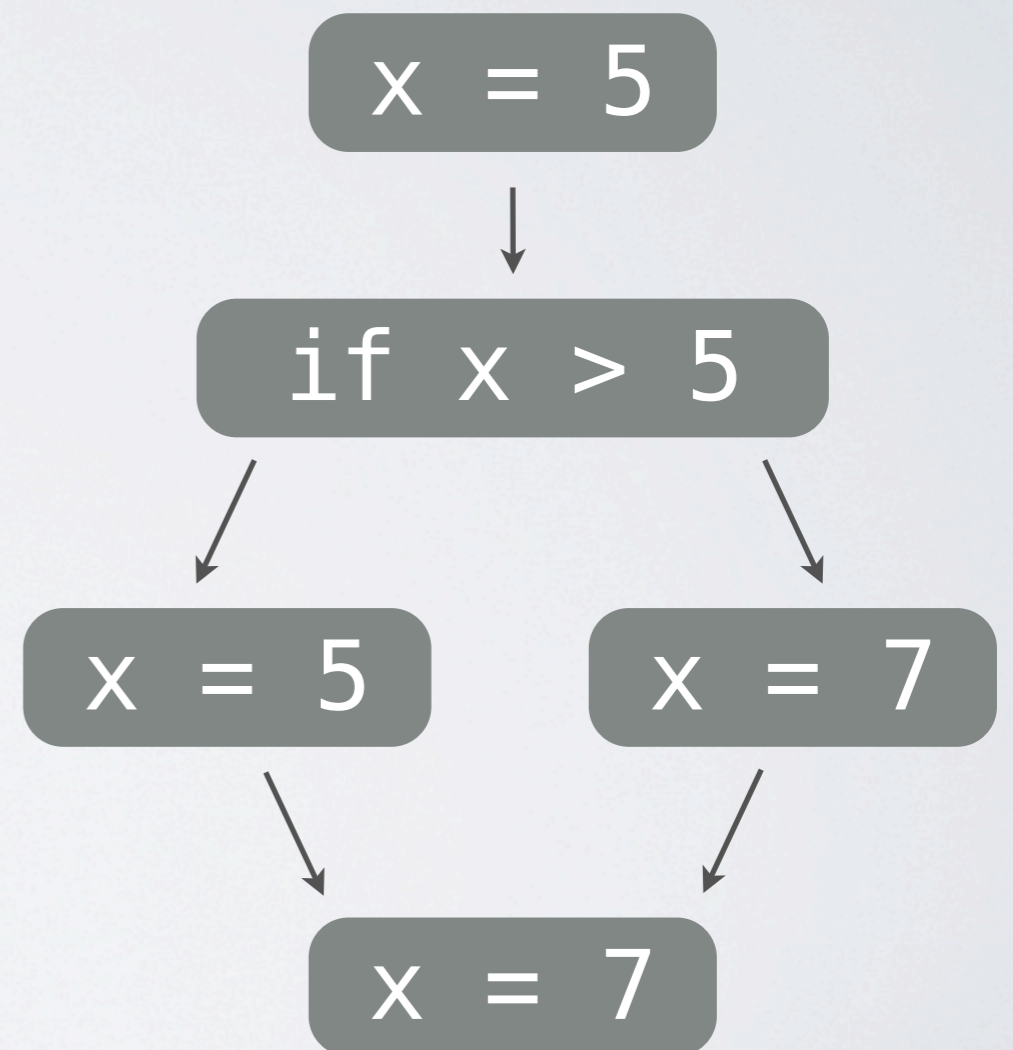
- Generates SSA-based IR

# STACK DECONSTRUCTION

- Generates SSA-based IR
- $\varphi$  pseudo instruction

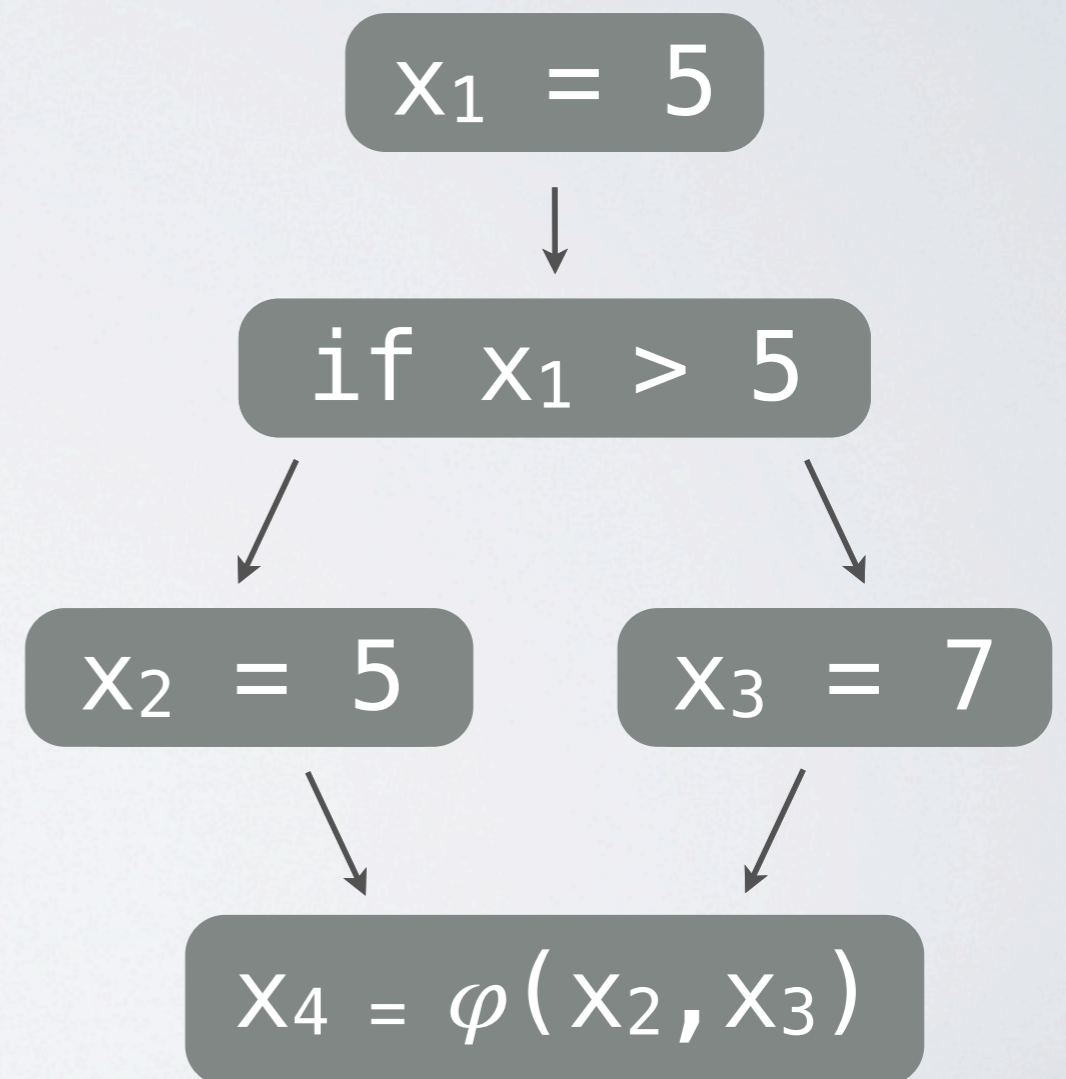
# STACK DECONSTRUCTION

- Generates SSA-based IR
- $\varphi$  pseudo instruction



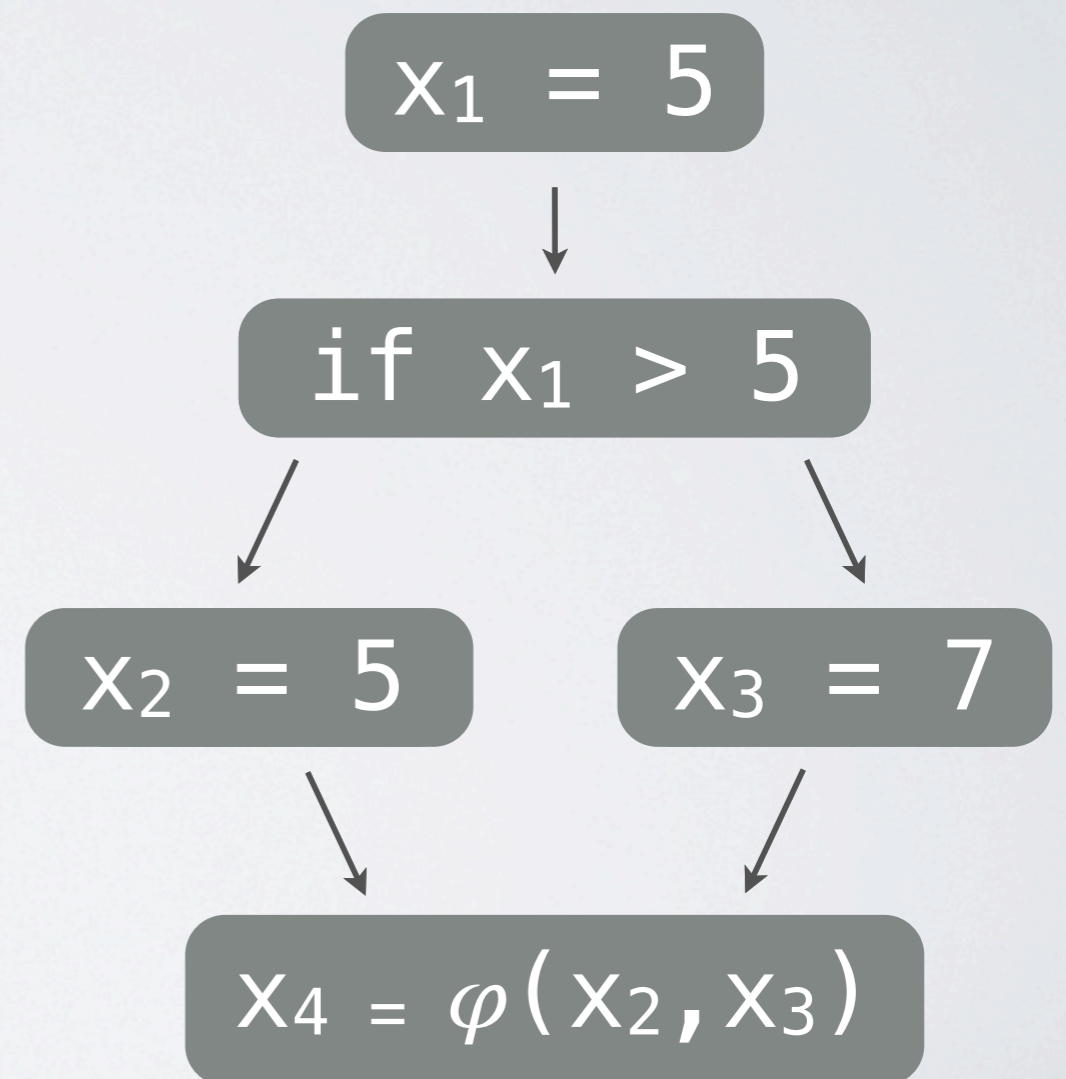
# STACK DECONSTRUCTION

- Generates SSA-based IR
- $\varphi$  pseudo instruction



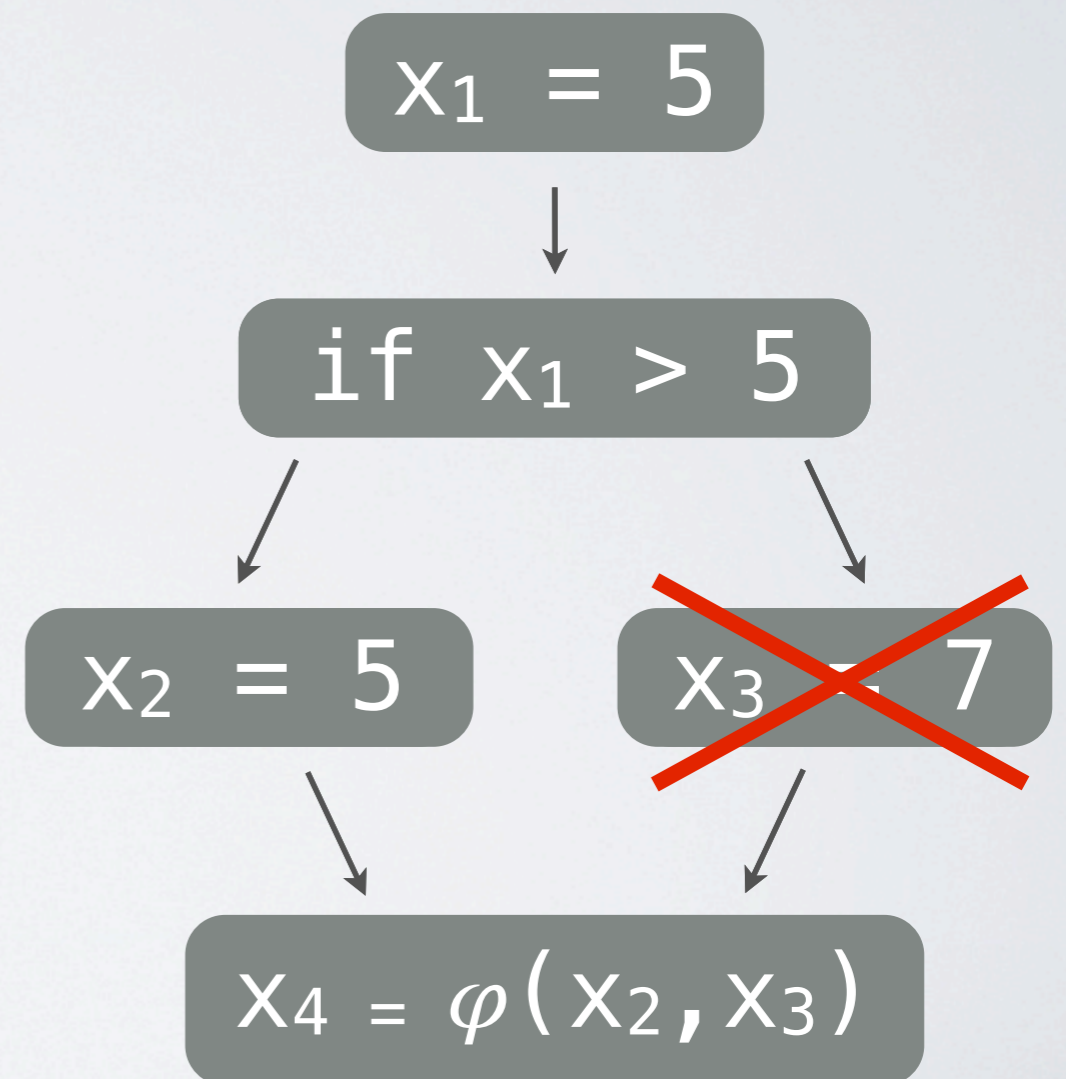
# STACK DECONSTRUCTION

- Generates SSA-based IR
- $\varphi$  pseudo instruction
- Flags loop invariant code



# STACK DECONSTRUCTION

- Generates SSA-based IR
- $\varphi$  pseudo instruction
- Flags loop invariant code



# CODE ANALYSIS

# CODE ANALYSIS

- Constant Propagation

# CODE ANALYSIS

- Constant Propagation
- Loop Peeling

# CODE ANALYSIS

- Constant Propagation
- Loop Peeling
- Common subexpression elimination

# CODE ANALYSIS

- Constant Propagation
- Loop Peeling
- Common subexpression elimination
- Invariant code motion

# CODE GENERATION

# CODE GENERATION

- Generating code backwards

# CODE GENERATION

- Generating code backwards
  - Enables to emit special machine code

# CODE GENERATION

- Generating code backwards
  - Enables to emit special machine code
- Constant Folding

# SIDE EXITS

# SIDE EXITS

- Leaving the hot path

# SIDE EXITS

- Leaving the hot path
- Generator generates compensation code

# SIDE EXITS

- Leaving the hot path
- Generator generates compensation code
- Special case: inline methods

# SIDE EXITS

- Leaving the hot path
- Generator generates compensation code
- Special case: inline methods
  - Writing back current state

# WALKTHROUGH

# WALKTHROUGH

Java Code

# WALKTHROUGH

Java Code

*compilation*  
→

# WALKTHROUGH



# WALKTHROUGH

Bytecode

# WALKTHROUGH



# WALKTHROUGH



# WALKTHROUGH



- Flag invariant code

# WALKTHROUGH

SSA

# WALKTHROUGH

SSA

*analysis*  
*optimization*  
→

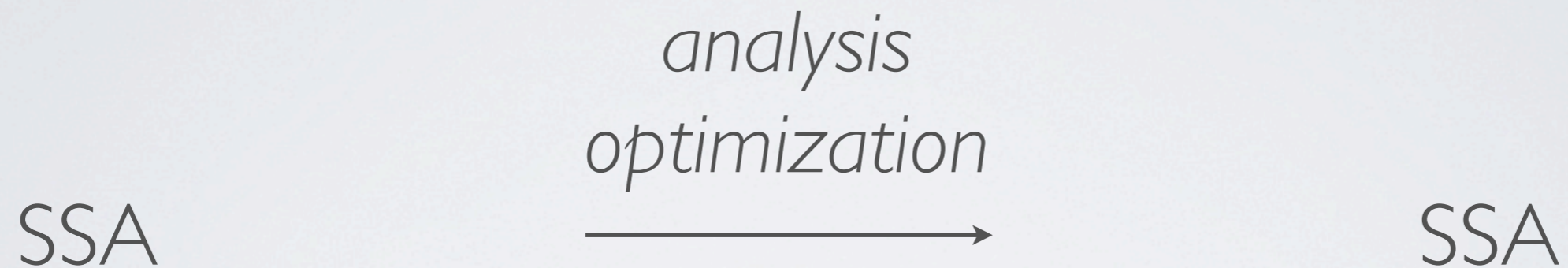
# WALKTHROUGH

SSA

*analysis*  
*optimization*  
→

SSA

# WALKTHROUGH



- Constant Propagation
- Loop Peeling
- Common subexpression elimination
- Invariant code motion

# WALKTHROUGH

SSA

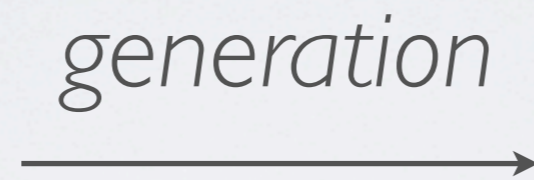
# WALKTHROUGH

SSA



# WALKTHROUGH

SSA



Machine code

# WALKTHROUGH

SSA



Machine code

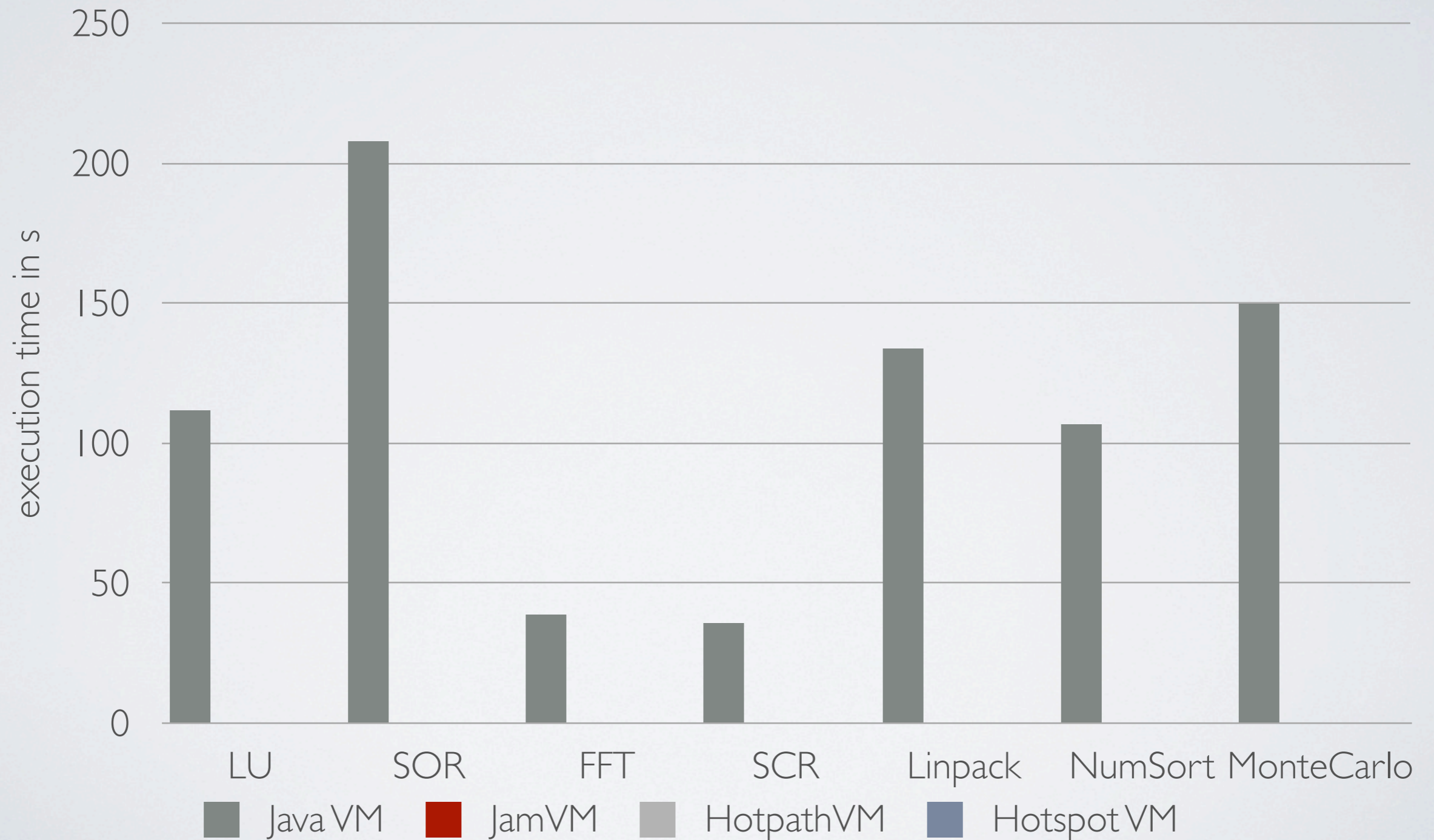
- Constant Folding

# BENCHMARKS

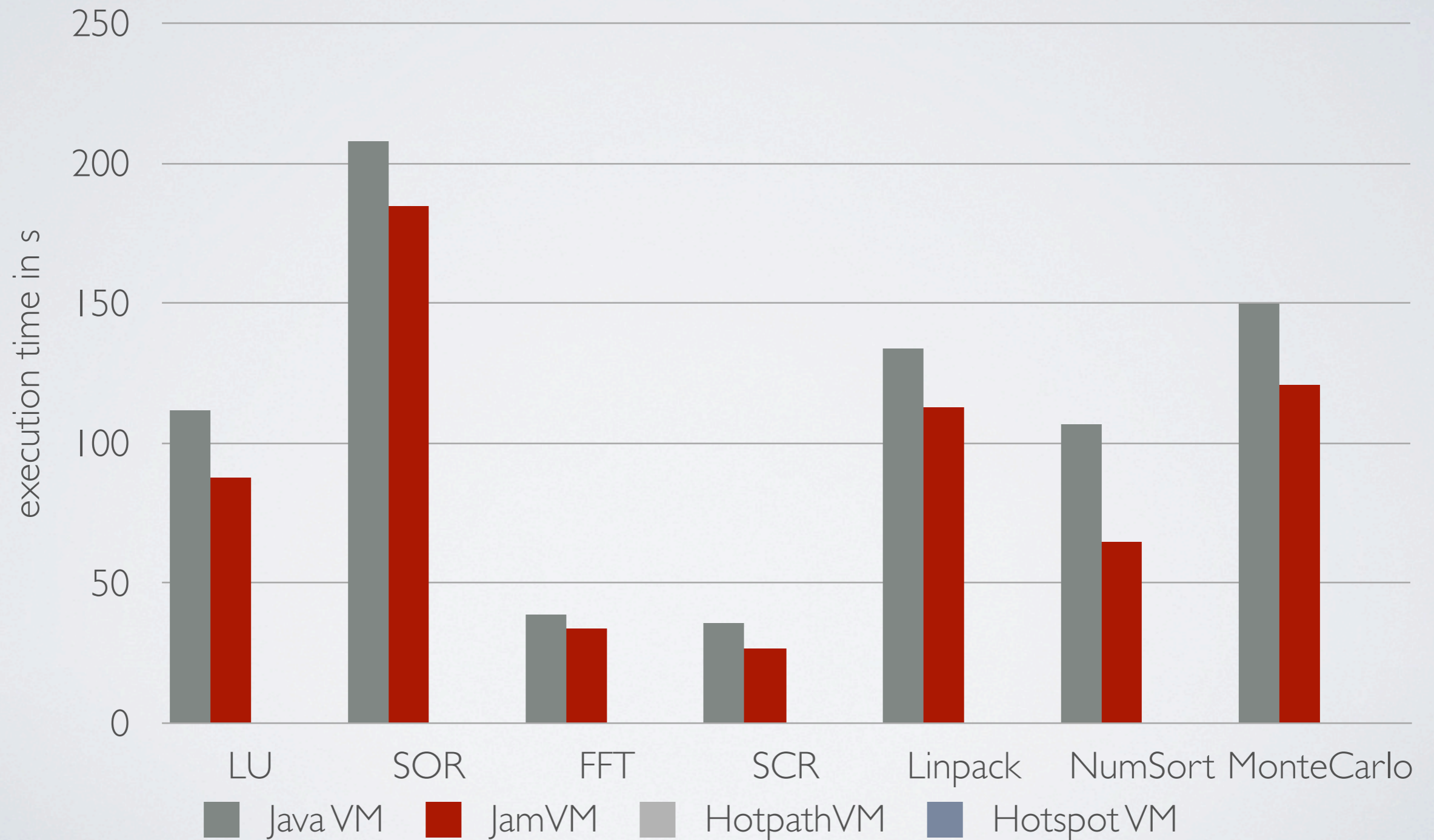
# BENCHMARKS



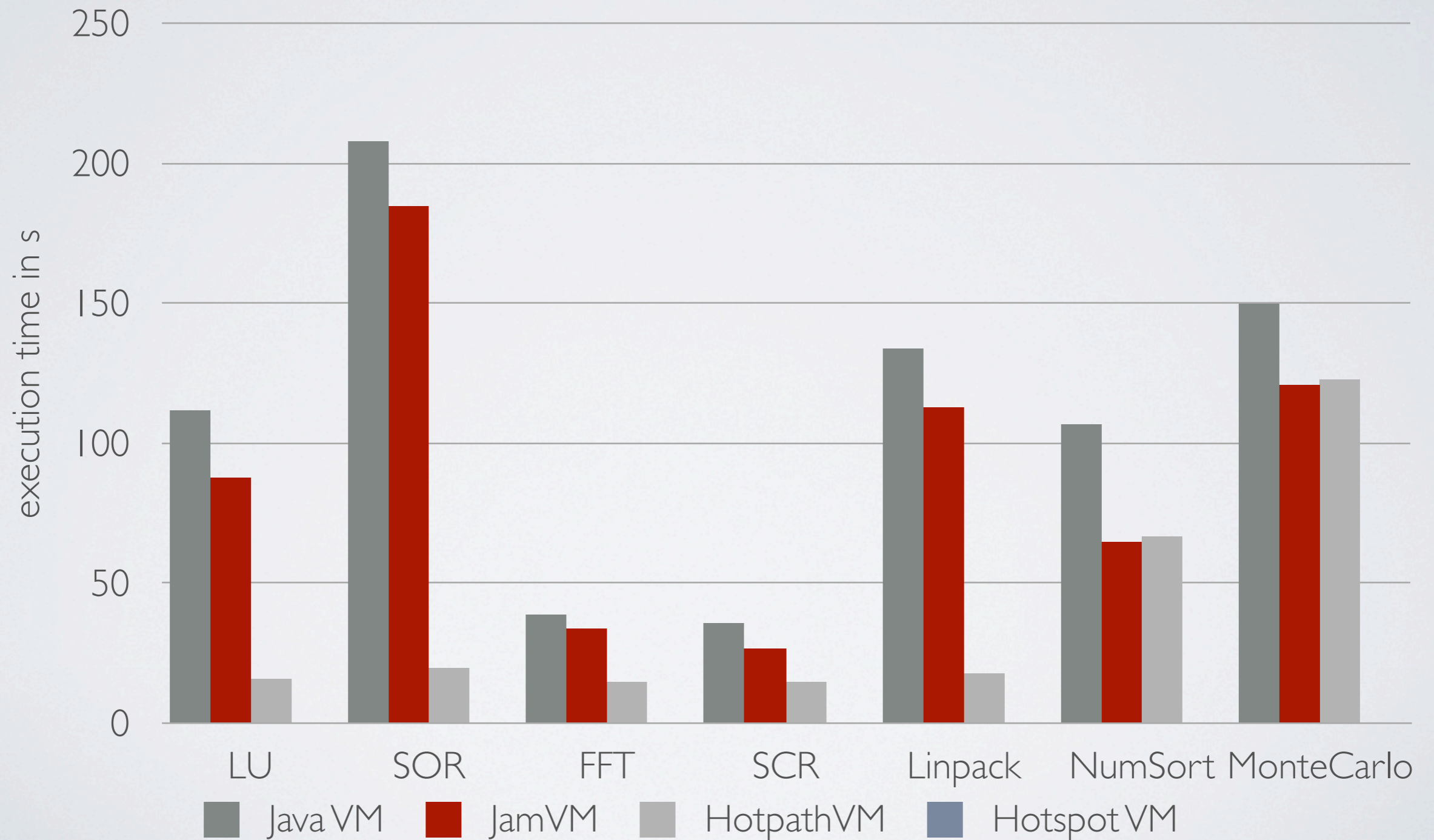
# BENCHMARKS



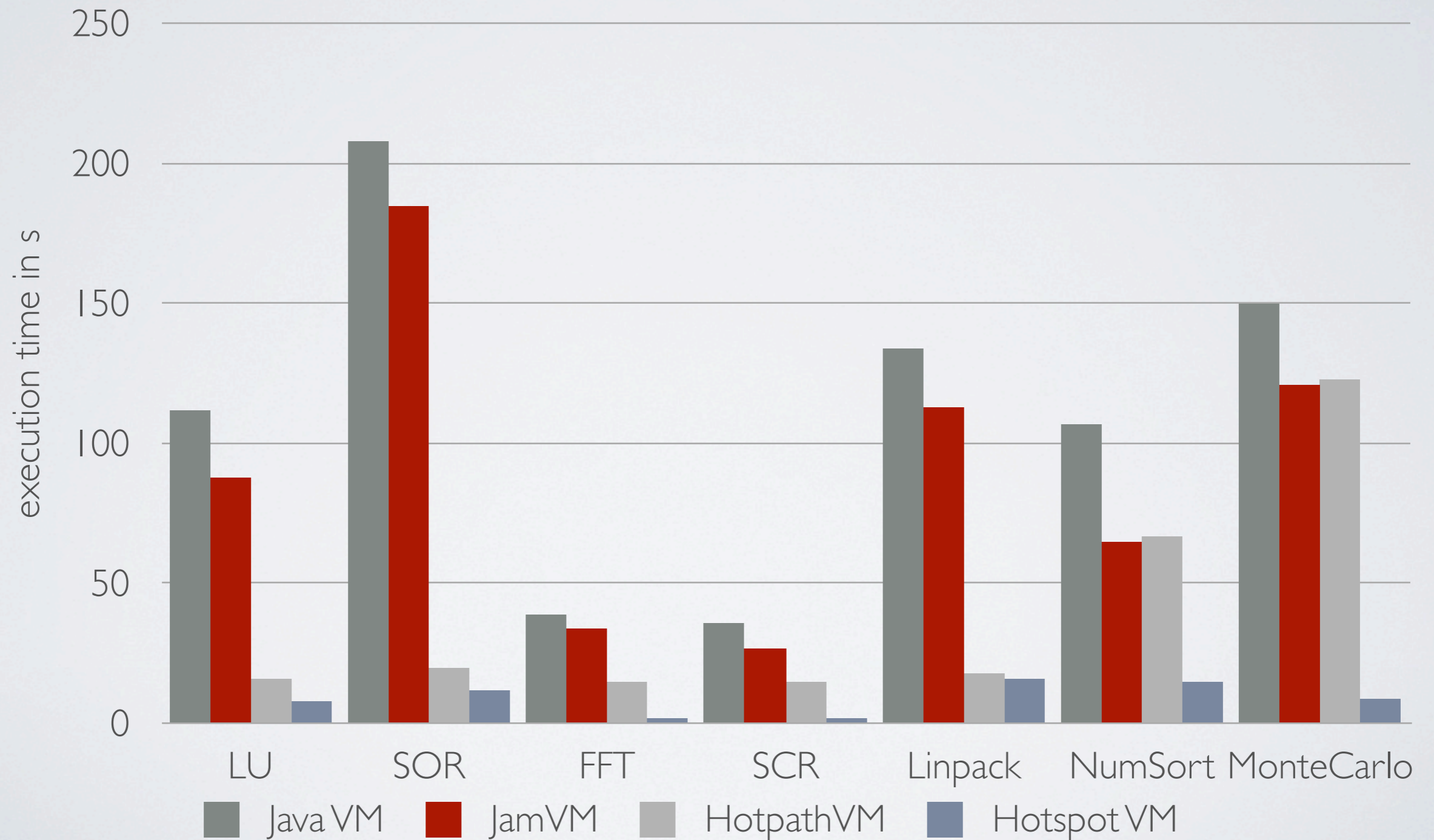
# BENCHMARKS



# BENCHMARKS

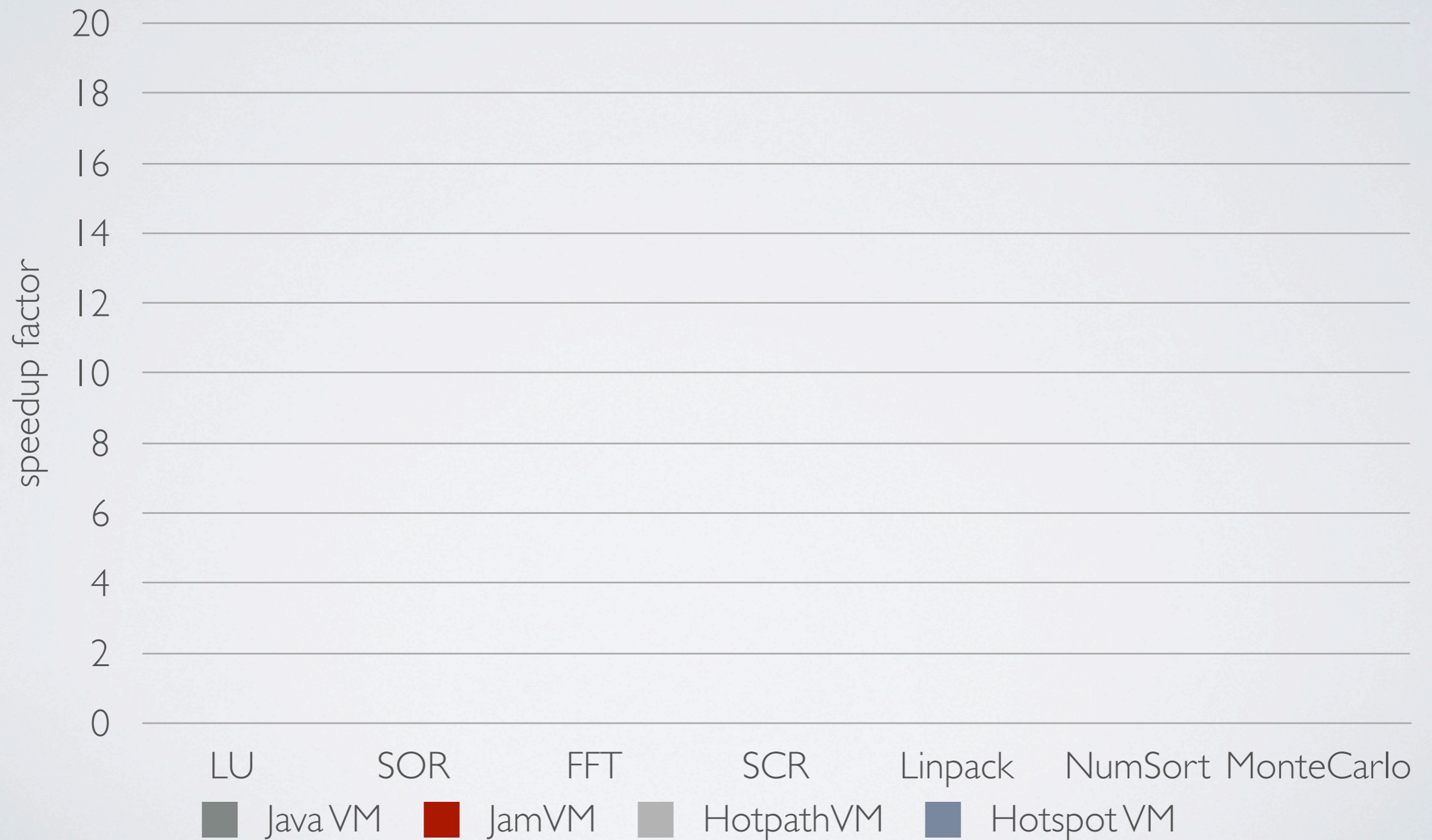


# BENCHMARKS

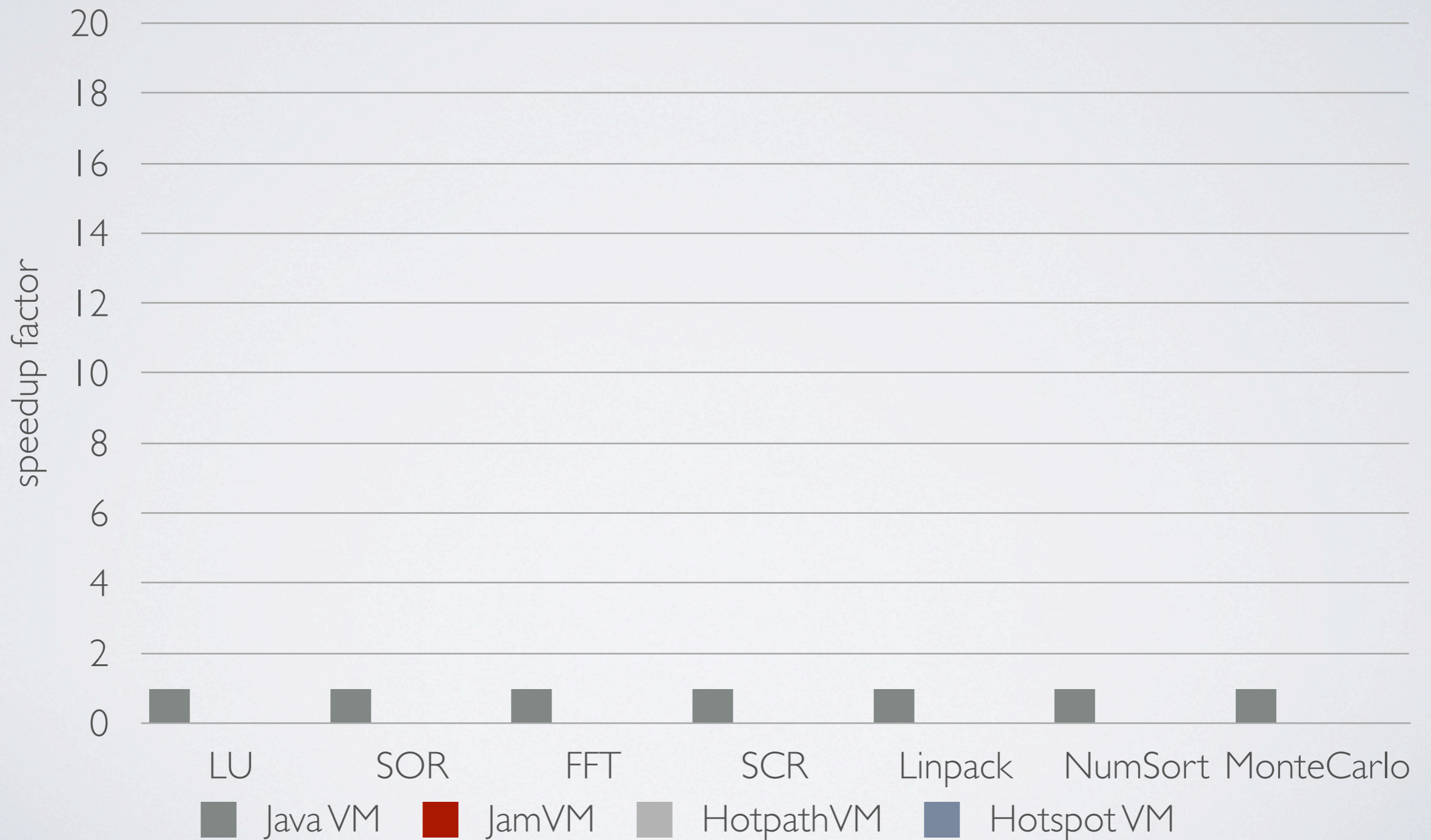


# BENCHMARKS

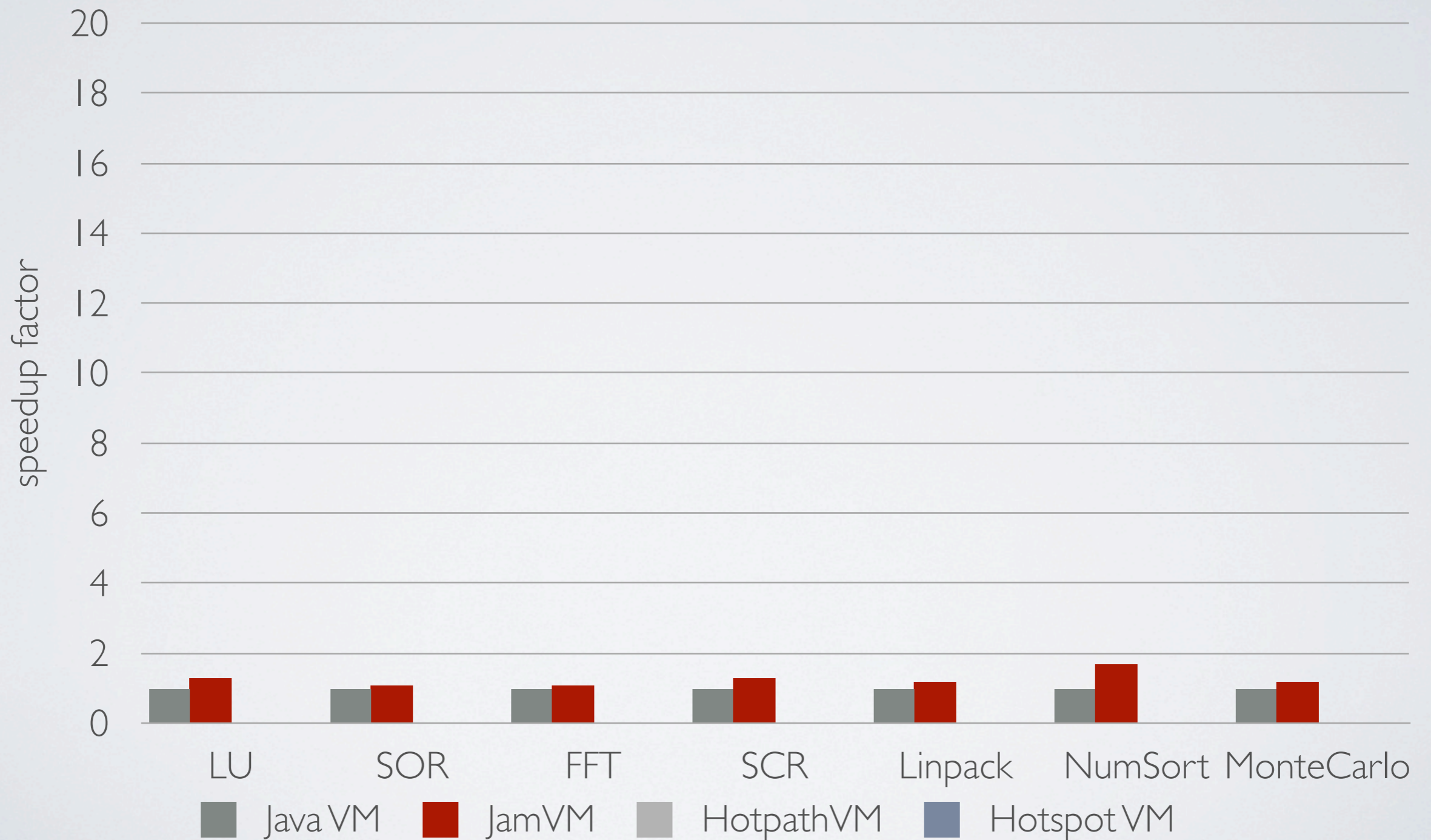
# BENCHMARKS



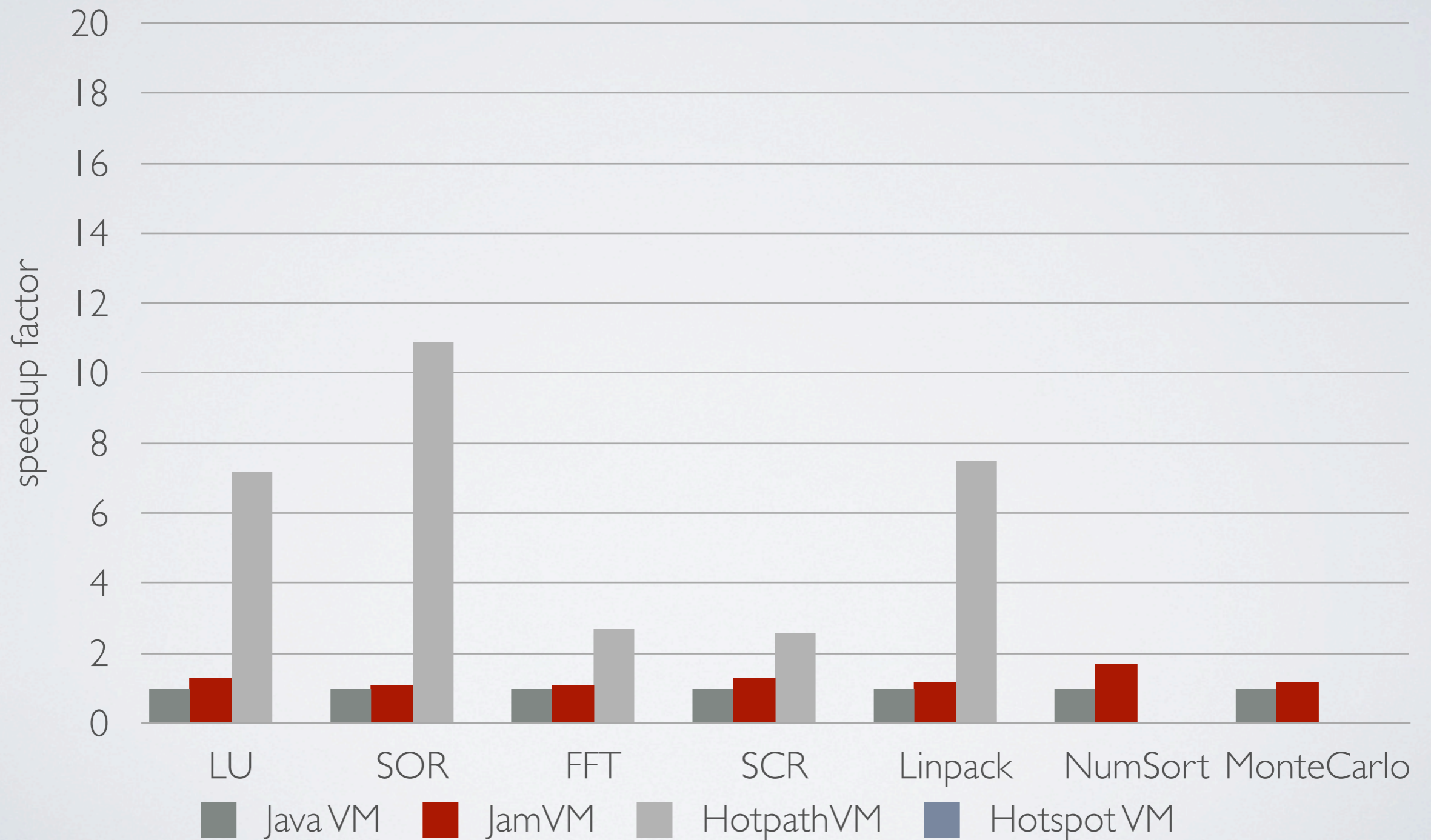
# BENCHMARKS



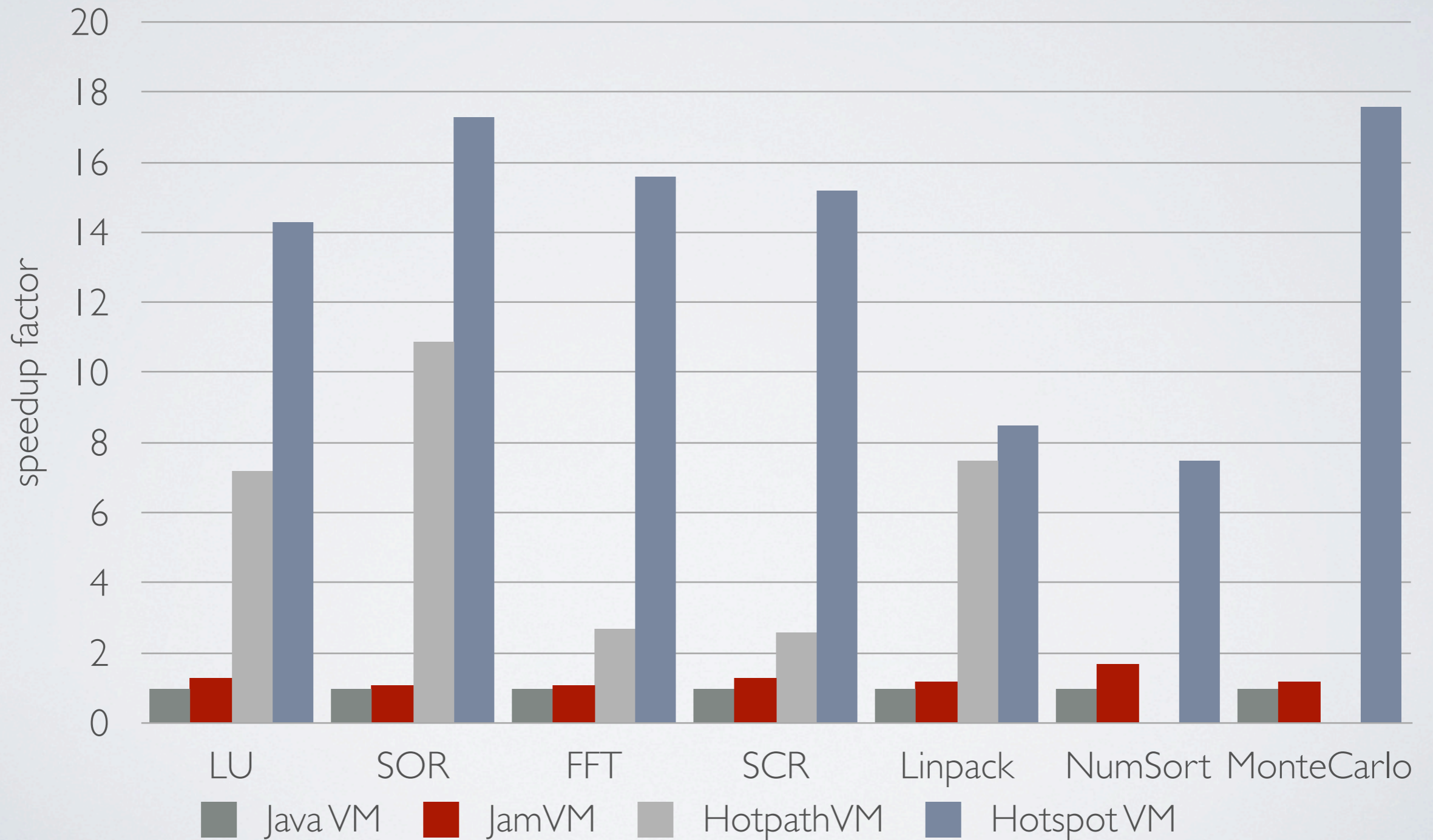
# BENCHMARKS



# BENCHMARKS



# BENCHMARKS



# OUTLOOK

# OUTLOOK

- Trace Merging

# OUTLOOK

- Trace Merging
- Support additional architectures (ARM)

# OUTLOOK

- Trace Merging
- Support additional architectures (ARM)
- Use outside embedded environment

# HOTPATH VM

An Effective JIT Compiler for Resource-constrained Devices