Scriptable Markdown pretty-printing with GraalVM

Pascal Maissen

pascal.maissen@unifr.ch

20.11.2018

Seminar Software Composition, University of Bern

Motivation

Implement a pretty printer for markdown in Java, which is scriptable with JavaScript (rather than having a configuration file).

Use the functionality of GraalVM

- to build a native image which runs faster
- to provide an API for a scripting language to customize the pretty printer

Introduction

- What is GraalVM?
- What is a native image?
- GraalVM Polyglot API
- Markdown language

GraalVM

- Universal virtual machine for running different programming languages
  - **JavaScript**, Python, Ruby, R
  - JVM-based: **Java**, Scala, Kotlin,
  - LLVM-based: C and C++
- Objectives:
  - Run Java faster
  - Make an application extensible, e.g. Java pretty printer is scriptable with JavaScript

Native image

- GraalVM can build a native image
- Full ahead-of-time compilation
- The built native binary contains the program in machine code and can be directly executed (on Linux and macOS)
- No JVM startup cost and a smaller memory footprint

Polyglot API

- GraalVM allows to embed and run code from other languages
- Zero overhead interoperability between languages
- Use advantages of a language, e.g. if a language has good predefined functions
- Use the best language for the given task

Markdown (1)

- Lightweight markup language in plain text syntax
- Can be converted to HTML
- Common use case: README files on GitHub

Markdown (2)

```
# Header 1

Some normal text, *italics* or **bold**.

A list:

 - Item 1
 - Item 2

## Header 2

```
A code block
of 2 lines
```

Here's a link to [a website](http://foo.bar)
```

Header 1

Some normal text, *italics* or **bold**.

A list:

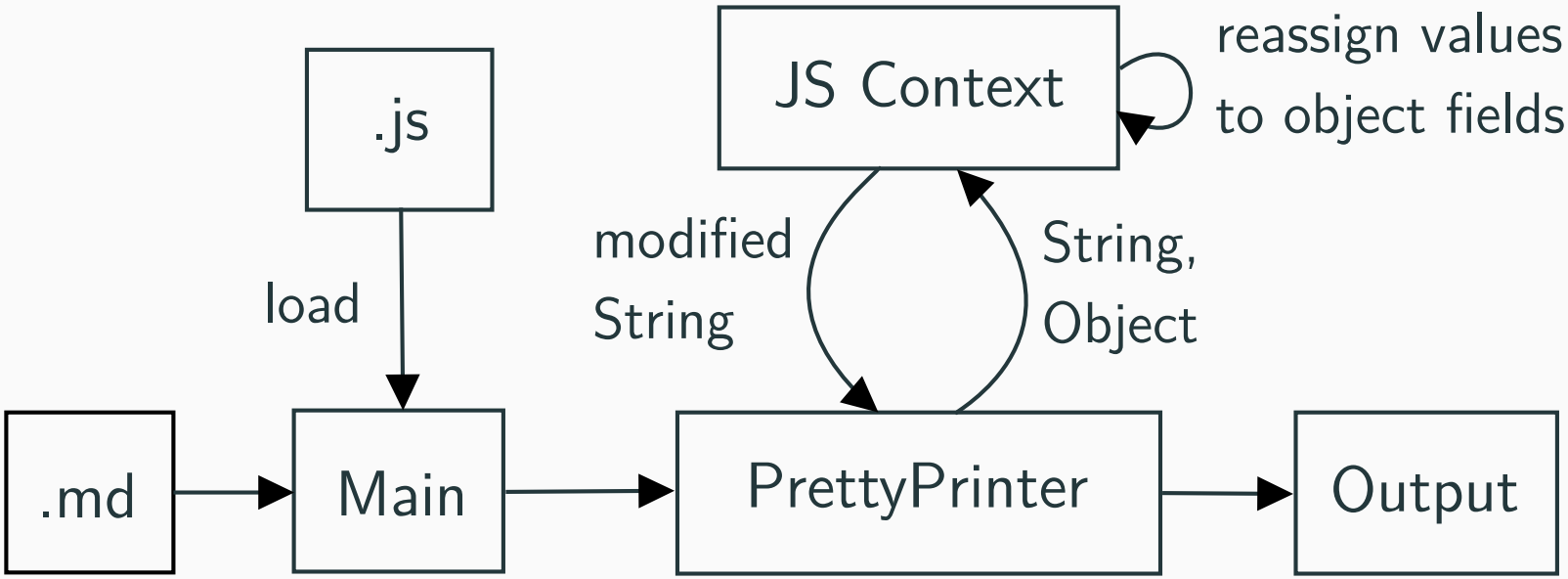
- Item 1
- Item 2

Header 2

```
A code block
of 2 lines
```

Here's a link to a website

Overview


.js
JS Context
reassign values
to object fields
load
modified
String
String,
Object
.md
Main
PrettyPrinter
Output

Code Snippets (1)

Java:

```
String someText = "## My Header";
// JavaScript polyglot context
Context context = Context.create();
context.eval("js", javaScript);
// calls the function "processHeader" in JS
Value v = context.getBindings("js")
          .getMember("processHeader")
          .execute(someText);
```

Code Snippets (2)

JavaScript:

```
function processHeader(params) {
    let result = "";
    // Transform the String passed from Java
    for(let i=0; i<params.length; i++) {
        result += params[i].toUpperCase() + " ";
    }
    return result;
}
```

Code Snippets (3)

Java:

```
String result = "";
// Check if returned value is a String
if (v.isString()) {
    result = v.asString();
} else {
    result = someText;
}
System.out.println(result);
```

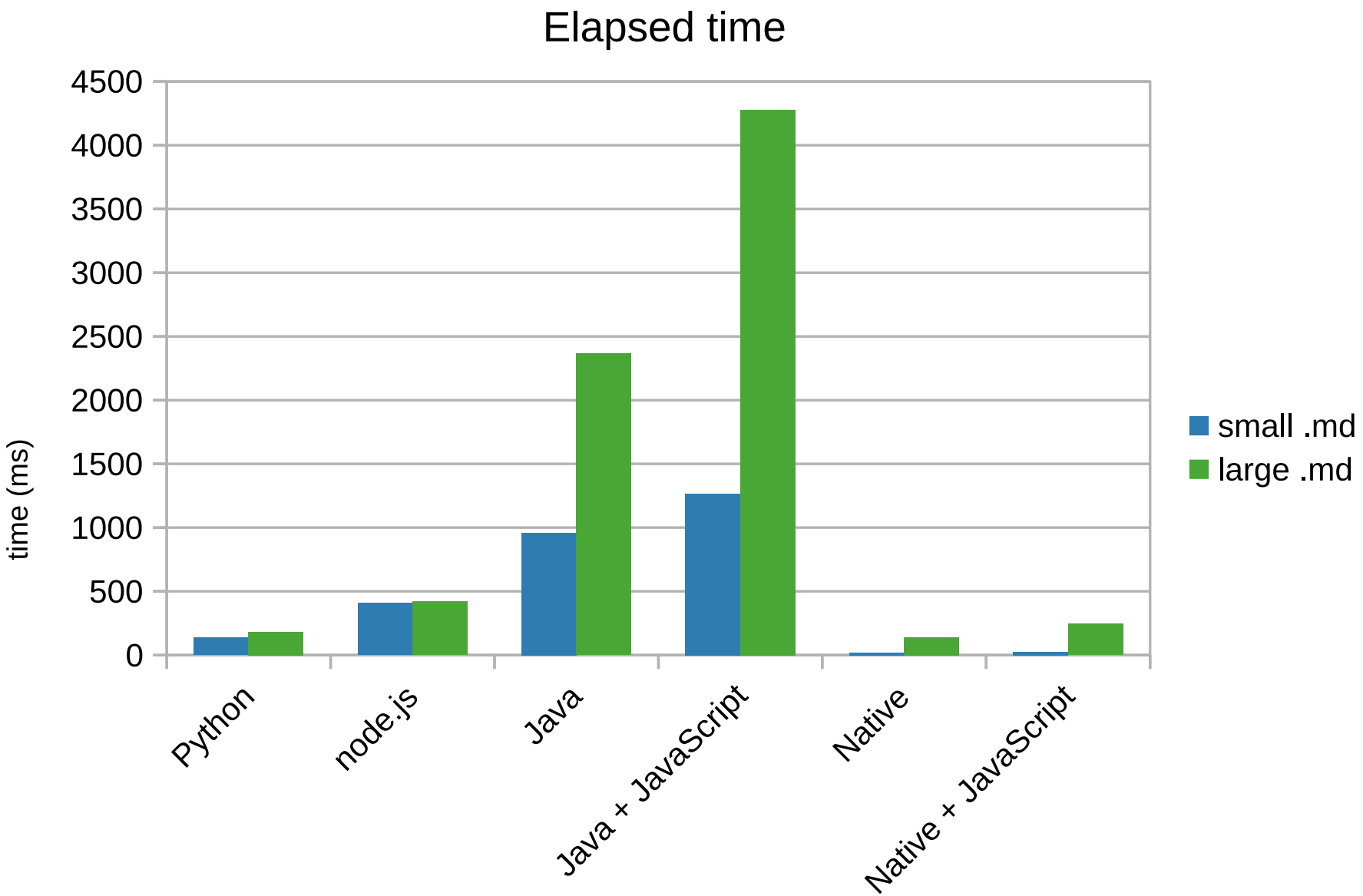
```
# #  M Y  H E A D E R
```

Demo

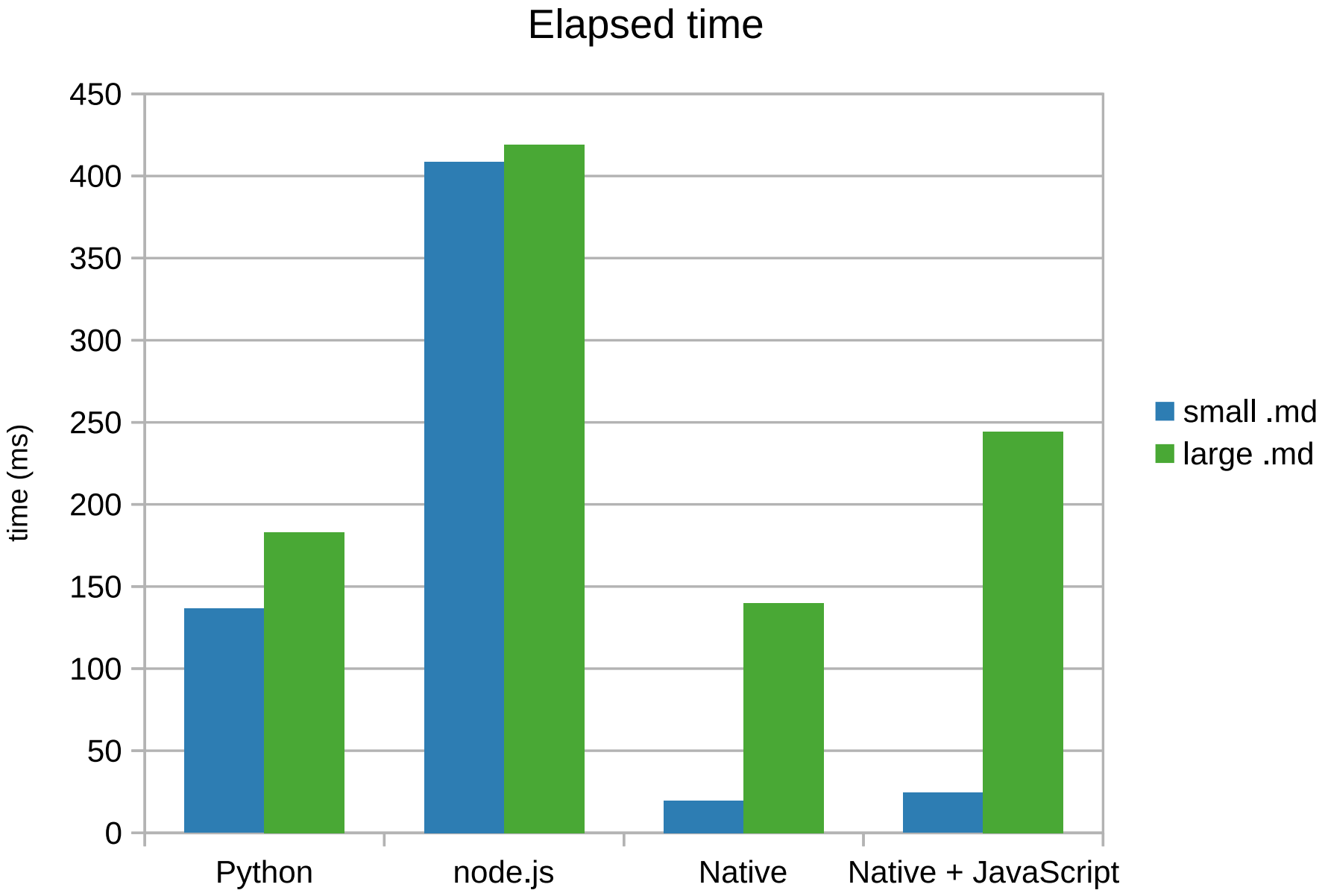
Performance (1)

- Performance tests:
  - Execution time
  - Used memory
- 2 different files:
  - Small .md file (≈100 lines)
  - Large .md file (≈2000 lines)
- Test device:
  - Intel Core i5-5200U @ 4x 2.7GHz
  - 8 GB RAM
  - SSD

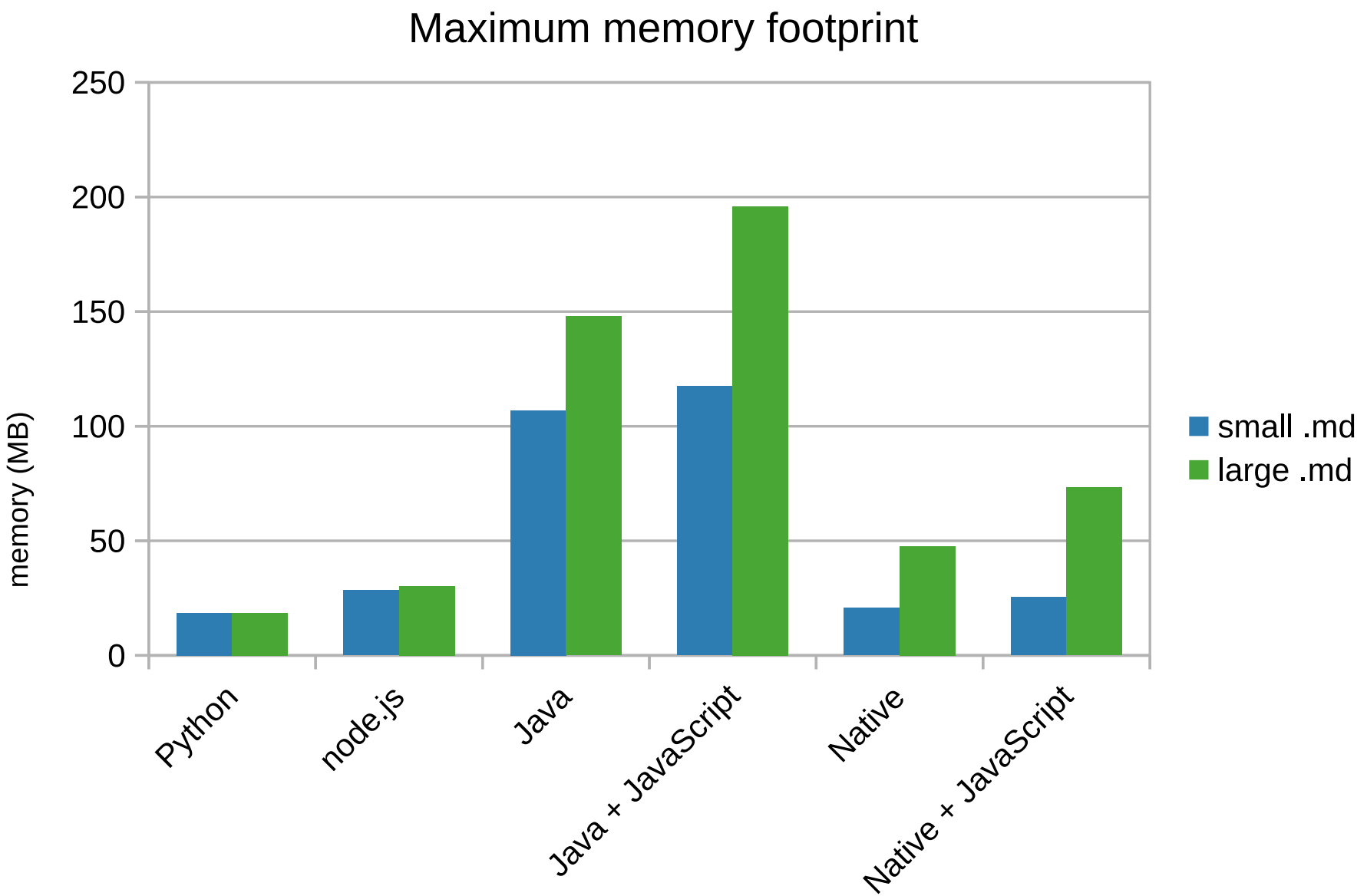
Performance (2)



Performance (3)



Performance (4)



Performance (5)

- Native image executes almost instantaneous for small files (typically .md files are not too large)
- For small files the native image is around 50× faster than Java, mostly due to boot up time of the JVM
- 100 ms faster than the python implementation for small files
- **Use case:** format on save in an IDE, 100 ms each time are already a lot

Security feature of GraalVM

```
Context context = Context.create();

Timer timer = new Timer(true);
timer.schedule(new TimerTask() {
    @Override
    public void run() {
        context.close(true);
    }
}, 5000);

try {
    context.eval("js", "while(true)");
    assert false;
} catch (PolyglotException e) {
    assert e.isCancelled();
}
```

Issues and limitations

- Native image can't handle Java objects passed to JavaScript (open issue)
  $\longrightarrow$ but there is an interface called `ProxyObject` to make host (Java) objects behave like guest (JavaScript) objects
- Native image build time varied from 5 min up to 40 min, both on idle identical machine
- Atlassian commonmark parser has limited functionality: no tables, no alternate header etc.
- Compromise between customization and visitor pattern integrity
- Native image has a size of 70 MB

Future work

- Write documentation
- Further code improvements and cleanup
- Explore other functionality of GraalVM

Summary

- Native image is much faster
- Scriptable pretty printer, high customization if desired
- Pretty Printer can have a scripting interface for any supported language

References

prettymd:

`github.com/boris-spas/prettymd`

GraalVM:

`www.graalvm.org`

Atlassian commonmark:

`github.com/atlassian/commonmark-java`

Python pretty printer:

`github.com/mrcoles/readmd`

Node.js pretty printer:

`github.com/jonschlinkert/prettify-markdown`