Oxidizing Python: writing extensions in Rust

Lab for Data Intensive Biology

UC DAVIS VETERINARY MEDICINE

Luiz Carlos Irber Júnior

lcirberjr@ucdavis.edu

Department of Population Health and Reproduction, University of California, Davis, USA

@luizirber

luizirber/2018-python-rust

GORDON AND BETTY MOORE FOUNDATION

Introduction

Python has a mature ecosystem for **extensions using C/C++**, with **Cython** being part of the standard toolset for **scientific programming**. Even so, C/C++ still have **many drawbacks**, ranging from smaller annoyances (like **library packaging**, **versioning** and **build systems**) to **serious** one like **buffer overflows** and **undefined behavior** leading to **security issues**. **Rust** is a system programming language trying to **avoid** many of the **C/C++ pitfalls**, on top of providing a **good development workflow** and **memory safety guarantees**. This work presents a way to **write extensions in Rust** and use them in Python, using **sourmash** [Brown and Irber, 2016] as an example.

sourmash implements **MinHash** [Broder, 1997], a method for estimating the **similarity of two or more datasets**, and expanding on the work pioneered by **Mash** [Ondov et al, 2016]. It is available as a CLI and a Python library.

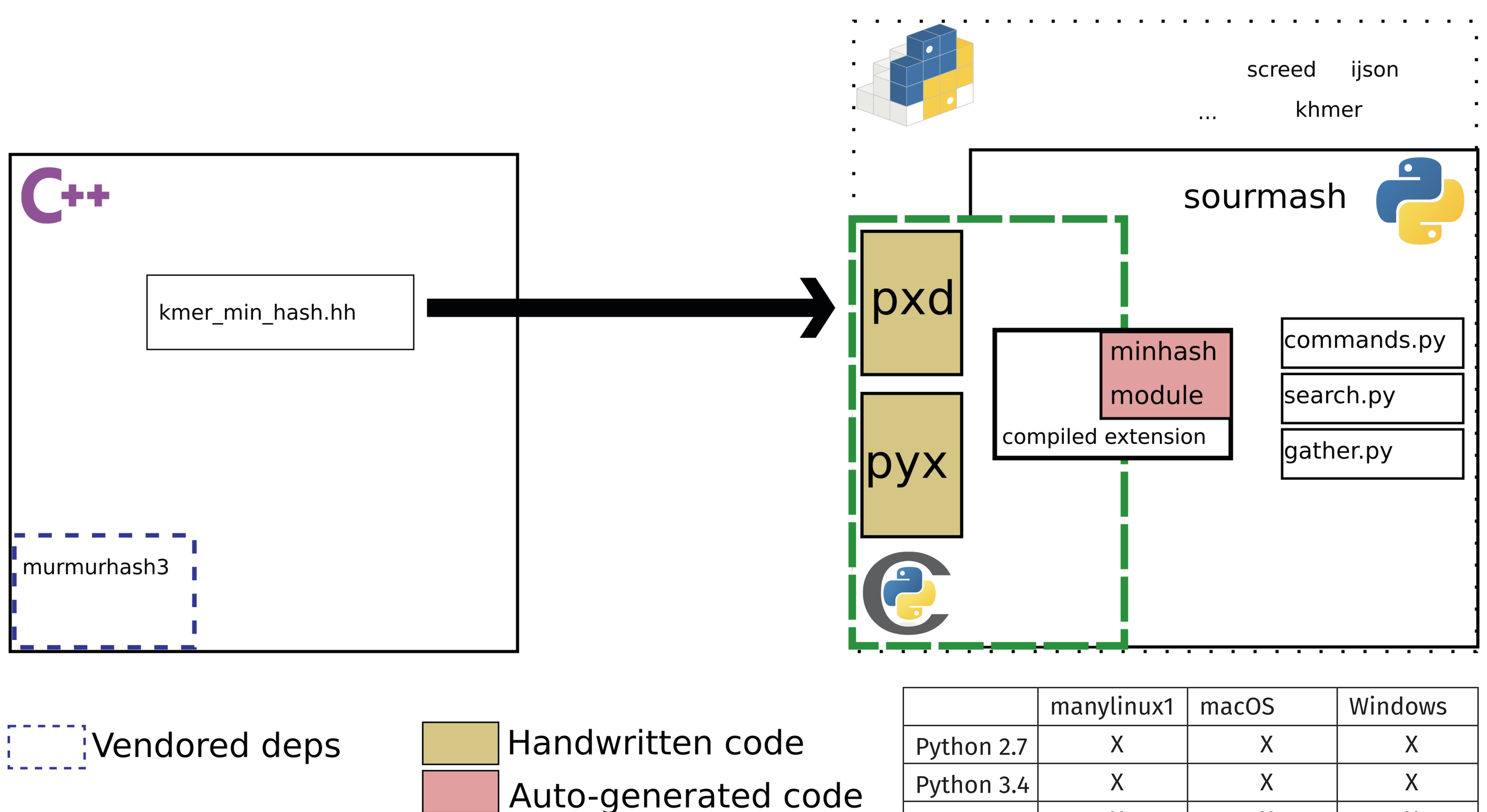
Why Rust?

While Rust doesn't aim at being a scientific language, its focus on being a **general purpose language** allows a phenomenon similar to what happened with Python, where people from many areas **pushed the language in different directions** (system scripting, web development, numerical programming...) creating an environment where developers **can combine it all in their systems**.

Rust brings many **best practices** to the default experience: **integrated package management with Cargo** (supporting **documentation**, **testing** and **benchmarking**). Some of them are not viable in C/C++ due to the **widespread adoption of both languages** and **backward compatibility guarantees**, but due to Rust being developed initially to be **integrated incrementally** in the Firefox **browser engine** it tries to keep **as much compatibility** as possible with C/C++.

Rust also has a **minimal runtime** (like C, unlike Python), making it a good candidate for **embedding** into other software or even for situations where **strict control of resources** is required (microcontrollers and **embedded systems**, for example).

Current implementation



	manylinux1	macOS	Windows
Python 2.7	X	X	X
Python 3.4	X	X	X
Python 3.5	X	X	X
Python 3.6	X	X	X
...	X	X	X

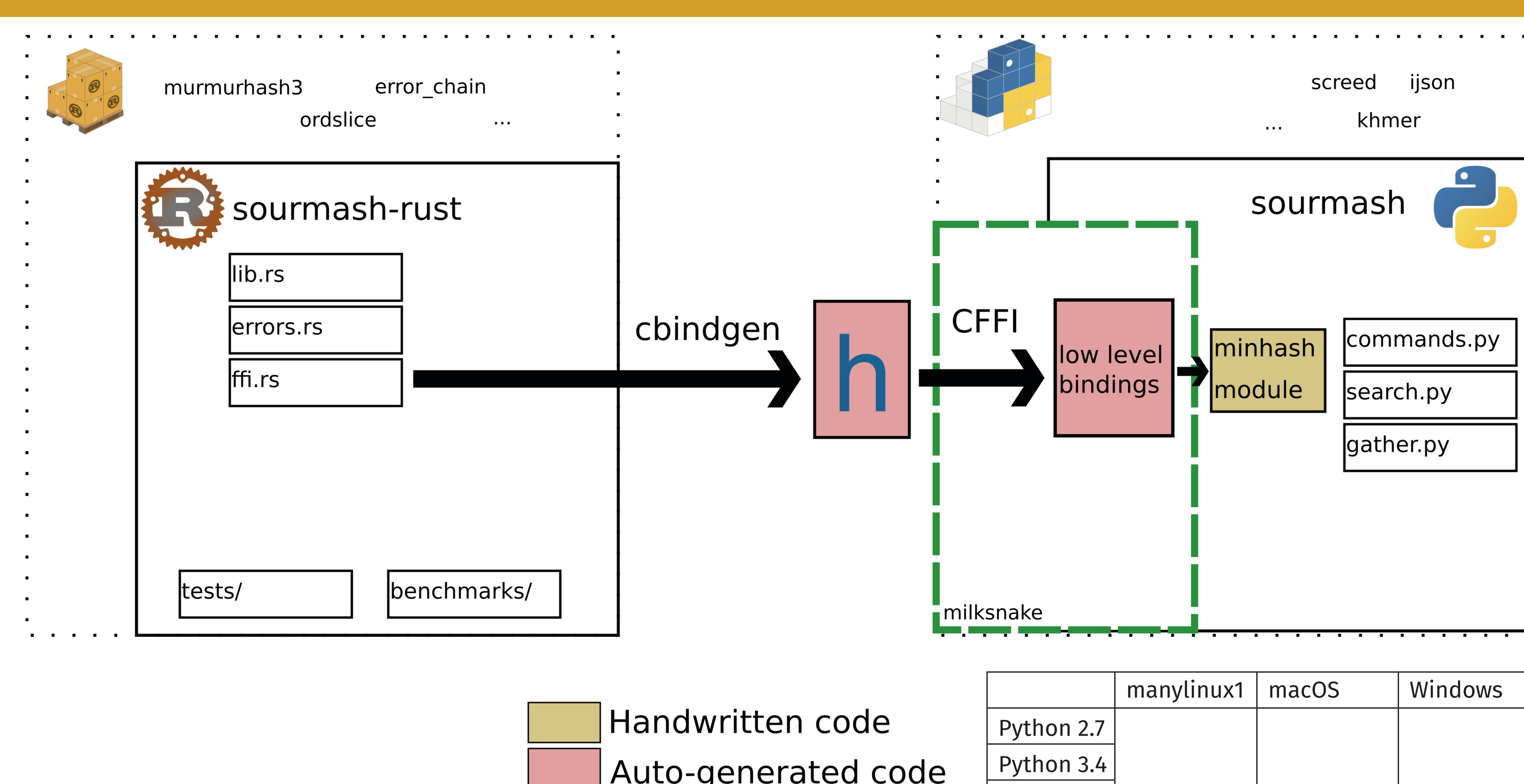
PROS

- Cython is a superset of Python
- Mature codebases for example usage and best practices
- Lower overhead to call C/C++ code
- NumPy integration
- Nice gradual path to migrate performance-intensive code from Python to C/C++

CONS

- Cython C++ integration has some corner cases and missing features
- Need to rewrite header declarations (pxd file)
- Errors can be cryptic (do they happen at the C/C++, Cython or Python level?)
- Many C/C++ build system combinations
- Vendored dependencies (no package mgmt)
- One wheel per OS and Python version

Rust implementation



<table>
<tr><th></th><th>manylinux1</th><th>macOS</th><th>Windows</th></tr>
<tr><td>Python 2.7</td><td rowspan="5">X</td><td rowspan="5">X</td><td rowspan="5">X</td></tr>
<tr><td>Python 3.4</td></tr>
<tr><td>Python 3.5</td></tr>
<tr><td>Python 3.6</td></tr>
<tr><td>...</td></tr>
</table>

PROS

- Cargo and crates.io for package management
- FFI interface is reusable in other languages
- Auto-generated C header (cbindgen) and low level bindings (CFFI)
- Works for PyPy too
- One wheel per OS (universal)

CONS

- Fewer projects using Rust extensions
- FFI overhead when calling C code
- No gradual transition from Python to Rust code
- Fewer bioinformatics libraries available
- No NumPy integration
- Low level abstraction ("what C can represent")

Future Work

This proof of concept focused on **replacing the C++ parts** with **Rust**, but while all the **sourmash tests are passing** there are many **improvements** to be done. The **performance** in most benchmarks is **very close** to the C++ implementation, but since this wasn't the initial goal of the experiment there are **many opportunities** to make it **faster**.

Another goal is to be able to use the **core functionality** of sourmash in **browsers**. A **previous experiment** focused on implementing a compatible package in **JavaScript**, but it lead to split codebases and **increased maintenance burden**. The Rust implementation make it possible to **target WebAssembly** and generate a **JavaScript package** wrapping it, with the added benefit of avoiding some JavaScript **shortcomings** (like **64-bit integers** support).

The Rust library implements **basic compatibility** with **Finch sketches** [Bovee and Greenfield, 2018], **allowing sharing data** between both **MinHash implementations**. Many of the other **sourmash methods** (**search**, **gather**) are not available in Rust yet, but this already allows using **other MinHash sketches with them**.

References

Broder, Andrei Z. 1997. "**On the Resemblance and Containment of Documents.**" In Compression and Complexity of Sequences 1997. Proceedings, 21–29. IEEE. http://ieeexplore.ieee.org/abstract/document/666900/.

Ondov, Brian D., Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. 2016. "**Mash: Fast Genome and Metagenome Distance Estimation Using MinHash.**" Genome Biology 17: 132. doi:10.1186/s13059-016-0997-x.

Bovee, Roderick, and Nick Greenfield. 2018. "**Finch: A Tool Adding Dynamic Abundance Filtering to Genomic MinHashing.**" The Journal of Open Source Software. doi: 10.21105/joss.00505.

Titus Brown, C., and Luiz Irber. 2016. "**sourmash: A Library for MinHash Sketching of DNA.**" The Journal of Open Source Software 1 (5). doi: 10.21105/joss.00027.