scientist.com

The RDKit is getting Rust-y…

Javier Pineda, PhD

11 September 2024

Overview

1. Scientist.com

   Company context and structure search requirements

2. The Rust-y RDKit

   Low-level Rust bindings for the RDKit C++

3. Cheminée

   Using the Rust-y RDKit for a stand-alone structure search application

4. Future Directions

Overview

1. Scientist.com

   Company context and structure search requirements

2. The Rust-y RDKit

   Low-level Rust bindings for the RDKit C++

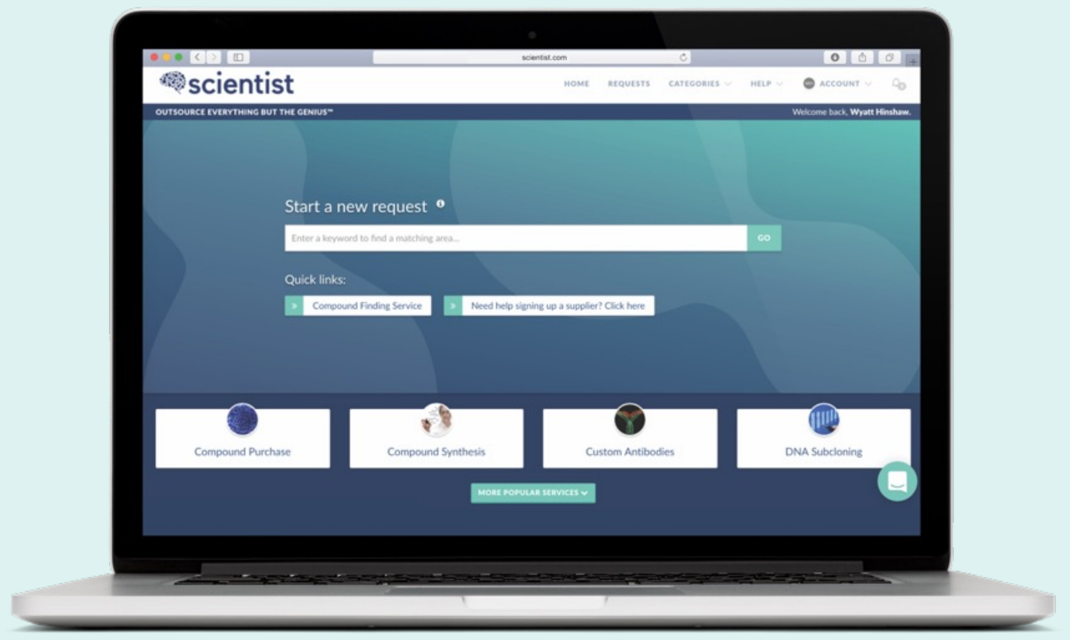
3. Cheminée

   Using the Rust-y RDKit for a stand-alone structure search application

4. Future Directions

Scientist.com is an R&D marketplace for researchers

Large and Small Research Organizations



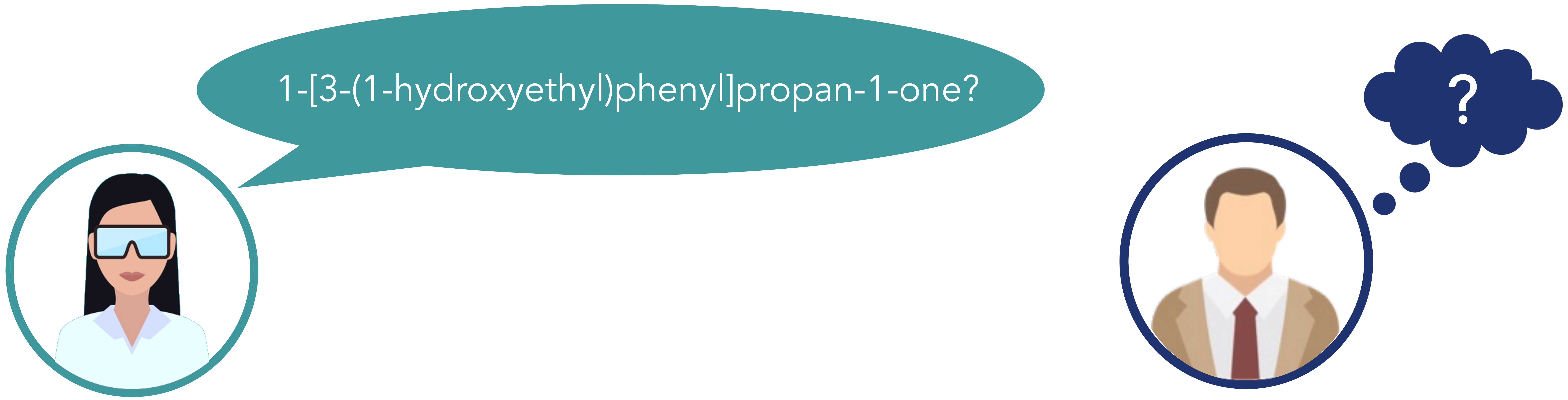
Scientist.com Marketplace

Global Suppliers of Life Science Products & Services

Human Biological Samples • Compound Synthesis • Antibody Services • Protein Sciences • Toxicology
In Vitro Assays • DNA/RNA Services • Assay Development • Molecular Pharmacology
Bioanalysis • Cell Culture and Charaterization • Data Sciences • CMC • Proteomics

Compound searches require structure searches

- ✓ 10+ million compounds from dozens of distributors of compounds
- ✓ Potential for scale up (e.g. more distributors, virtual compounds)
- ✓ We want something simple to spin up locally for our software dev team



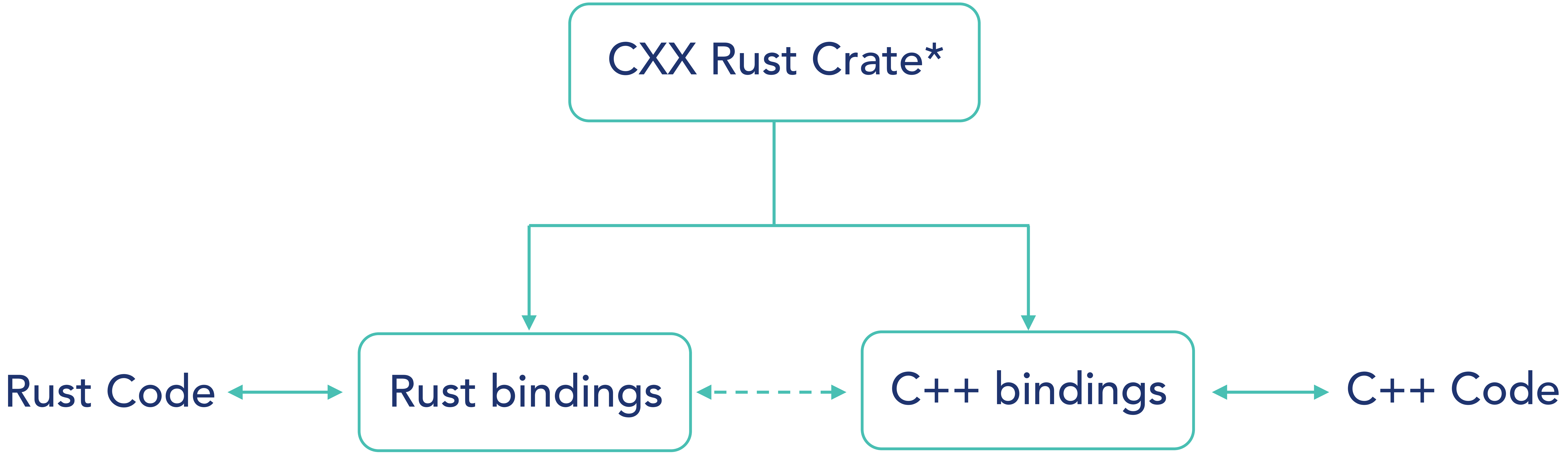
Overview

1. Scientist.com
   Company context and structure search requirements
2. The Rust-y RDKit
   Low-level Rust bindings for the RDKit C++
3. Cheminée
   Using the Rust-y RDKit for a stand-alone structure search application
4. Future Directions

Why Rust?

	C++	Python	Rust
Fast	Yes	No	Yes
Memory Safe	If you code right	Garbage collector	Yes
Thread Safe	If you code right	If you code right	Yes
User Friendly	No	Yes	Kind of

How should we implement the RDKit in Rust?



*https://cxx.rs

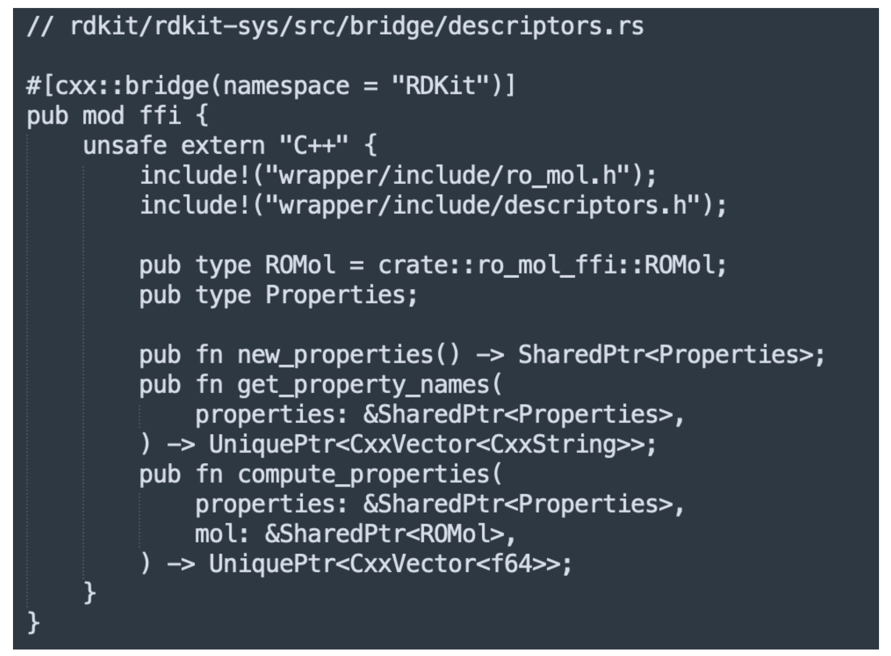
How is this different from the RDKit CFFI?

	RDKit CFFI*	Rust CXX
Pros	Already tied to RDKit User friendly Agnostic of end language	Directly call C++ code from Rust No serialization/deserialization Unlimited access to functionality Better handling of exceptions
Cons	Indirect bindings Serialization/deserialization Limited access to functionality	Limited to Rust Not yet implemented for RDKit

*C Foreign Function Interface

Creating Rust bindings can be cumbersome

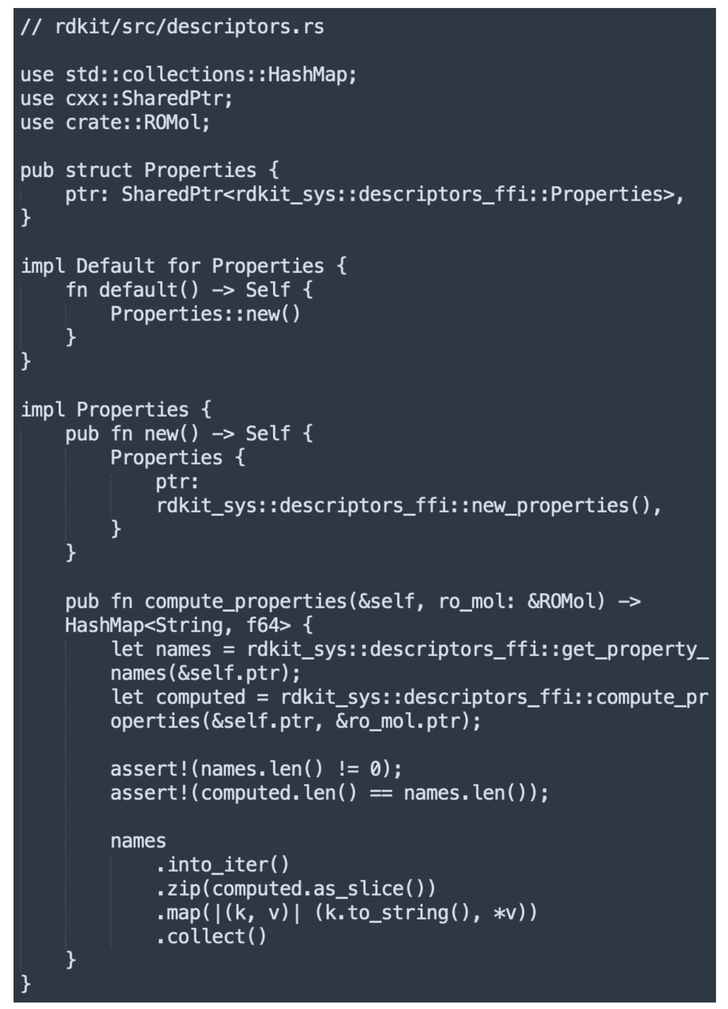
CXX Bridge*



```rust
// rdkit/rdkit-sys/src/bridge/descriptors.rs

#[cxx::bridge(namespace = "RDKit")]
pub mod ffi {
    unsafe extern "C++" {
        include!("wrapper/include/ro_mol.h");
        include!("wrapper/include/descriptors.h");

        pub type ROMol = crate::ro_mol_ffi::ROMol;
        pub type Properties;

        pub fn new_properties() -> SharedPtr<Properties>;
        pub fn get_property_names(
            properties: &SharedPtr<Properties>,
        ) -> UniquePtr<CxxVector<CxxString>>;
        pub fn compute_properties(
            properties: &SharedPtr<Properties>,
            mol: &SharedPtr<ROMol>,
        ) -> UniquePtr<CxxVector<f64>>;
    }
}
```

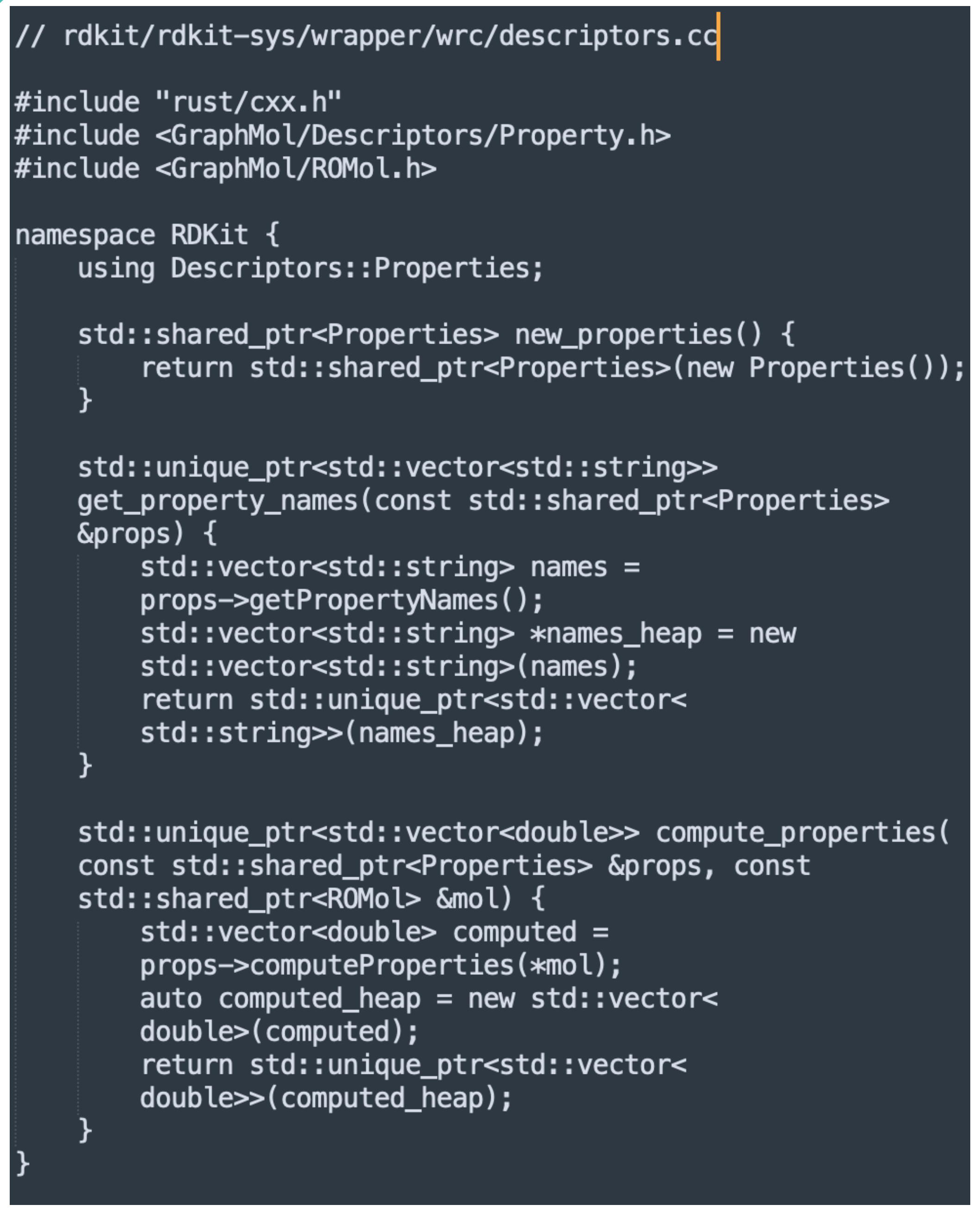
Higher-level

Rust code

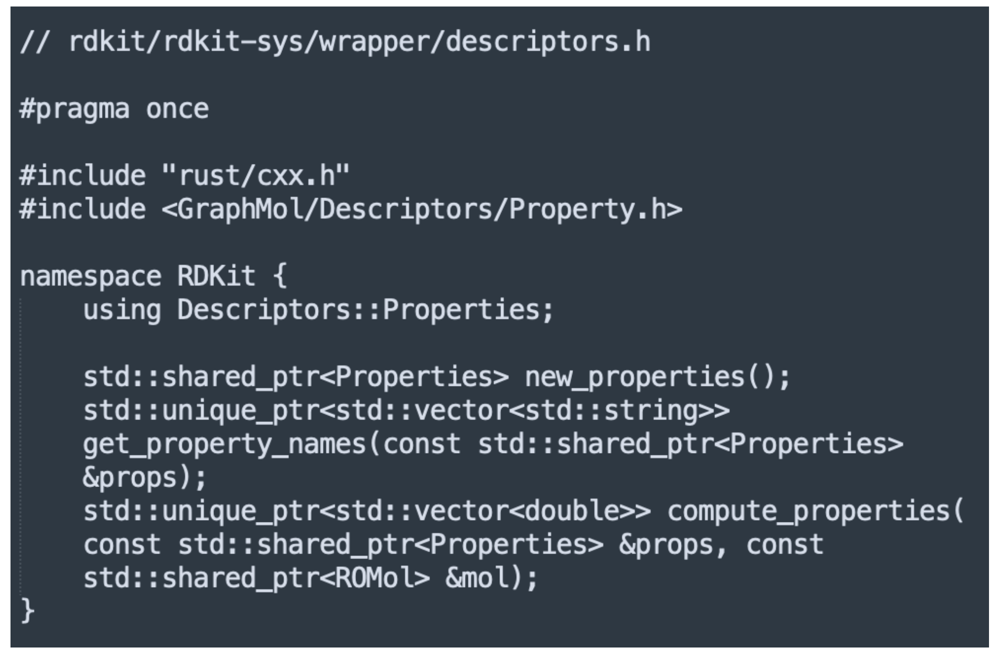


```rust
// rdkit/src/descriptors.rs

use std::collections::HashMap;
use cxx::SharedPtr;
use crate::ROMol;

pub struct Properties {
    ptr: SharedPtr<rdkit_sys::descriptors_ffi::Properties>,
}

impl Default for Properties {
    fn default() -> Self {
        Properties::new()
    }
}

impl Properties {
    pub fn new() -> Self {
        Properties {
            ptr:
            rdkit_sys::descriptors_ffi::new_properties(),
        }
    }

    pub fn compute_properties(&self, ro_mol: &ROMol) ->
    HashMap<String, f64> {
        let names = rdkit_sys::descriptors_ffi::get_property_
        names(&self.ptr);
        let computed = rdkit_sys::descriptors_ffi::compute_pr
        operties(&self.ptr, &ro_mol.ptr);

        assert!(names.len() != 0);
        assert!(computed.len() == names.len());

        names
            .into_iter()
            .zip(computed.as_slice())
            .map(|(k, v)| (k.to_string(), *v))
            .collect()
    }
}
```

Rust Bindings



```cpp
// rdkit/rdkit-sys/wrapper/wrc/descriptors.cc

#include "rust/cxx.h"
#include <GraphMol/Descriptors/Property.h>
#include <GraphMol/ROMol.h>

namespace RDKit {
    using Descriptors::Properties;

    std::shared_ptr<Properties> new_properties() {
        return std::shared_ptr<Properties>(new Properties());
    }

    std::unique_ptr<std::vector<std::string>>
    get_property_names(const std::shared_ptr<Properties>
    &props) {
        std::vector<std::string> names =
        props->getPropertyNames();
        std::vector<std::string> *names_heap = new
        std::vector<std::string>(names);
        return std::unique_ptr<std::vector<
        std::string>>(names_heap);
    }

    std::unique_ptr<std::vector<double>> compute_properties(
    const std::shared_ptr<Properties> &props, const
    std::shared_ptr<ROMol> &mol) {
        std::vector<double> computed =
        props->computeProperties(*mol);
        auto computed_heap = new std::vector<
        double>(computed);
        return std::unique_ptr<std::vector<
        double>>(computed_heap);
    }
}
```



```cpp
// rdkit/rdkit-sys/wrapper/descriptors.h

#pragma once

#include "rust/cxx.h"
#include <GraphMol/Descriptors/Property.h>

namespace RDKit {
    using Descriptors::Properties;

    std::shared_ptr<Properties> new_properties();
    std::unique_ptr<std::vector<std::string>>
    get_property_names(const std::shared_ptr<Properties>
    &props);
    std::unique_ptr<std::vector<double>> compute_properties(
    const std::shared_ptr<Properties> &props, const
    std::shared_ptr<ROMol> &mol);
}
```

C++

Bindings

*In this case, just for chemical descriptor methods

What about automated Rust bindings to RDKit?

- ✓ An attractive idea, considering only two of us are working on this
- ✓ AUTOCXX Rust crate* is already built to automate Rust-to-C++ bindings

Automated bindings!

Did not work reliably

*https://docs.rs/autocxx

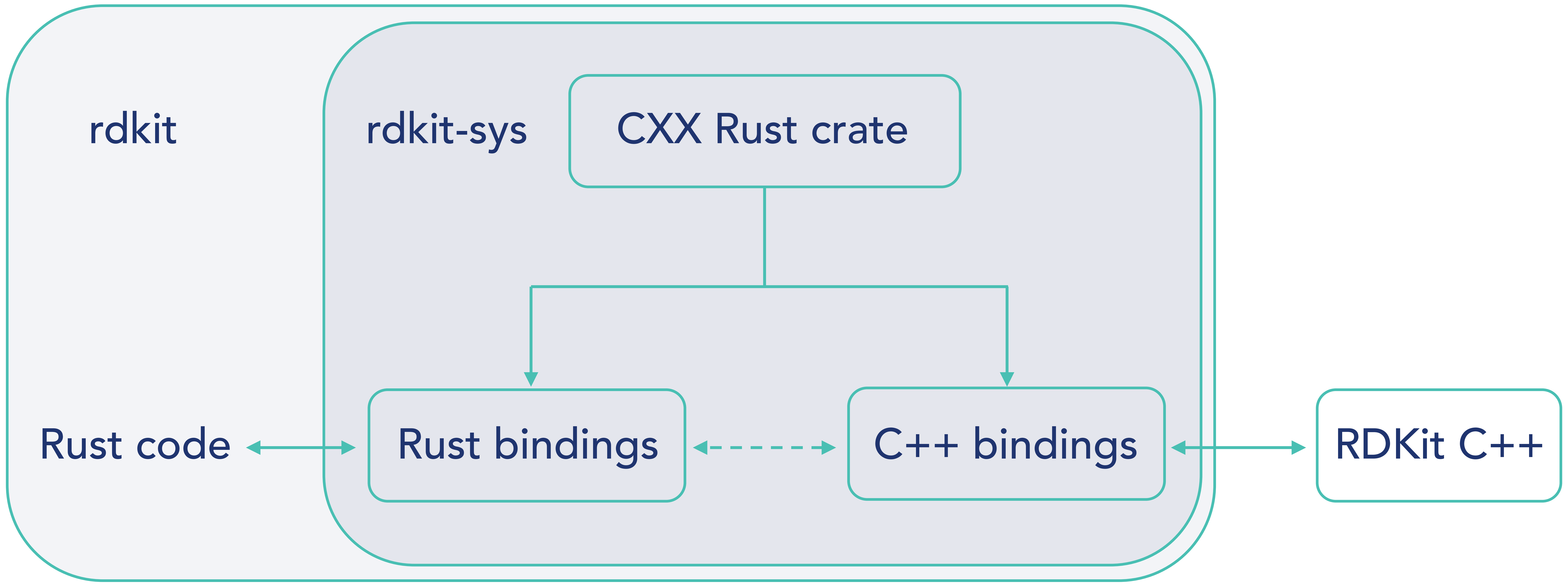
Back to manual RDKit Rust bindings!

Mol*	
from_smiles	atom_with_idx
to_smiles	clean_up
from_molblock	detect_chemistry_problems
to_molblock	update_property_cache
fingerprint	substruct_match
add_hs	descriptors
remove_hs	tautomer_canonicalization

Atom*
get_symbol
is_aromatic
get_formal_charge
hybridization
explicit_hs
total_valence
atomic_num

*A few of the bindings we've created

Say hello to the Rust-y RDKit*



*Repo: https://github.com/rdkit-rs/rdkit

Crate: https://docs.rs/rdkit

Overview

1. Scientist.com

   Company context and structure search requirements

2. The Rust-y RDKit

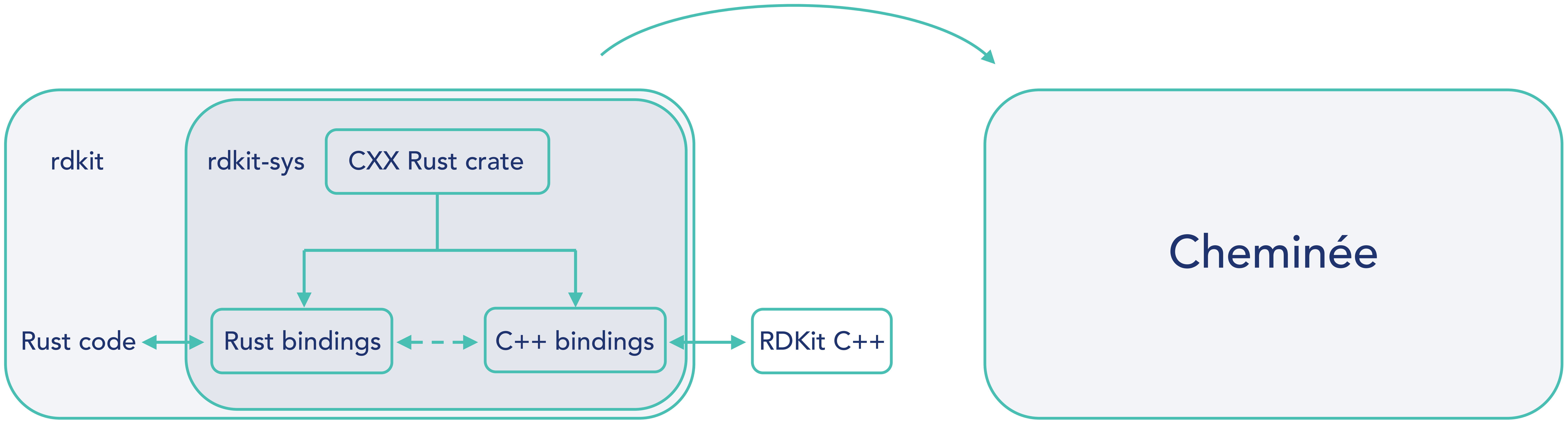
   Low-level Rust bindings for the RDKit C++

3. Cheminée

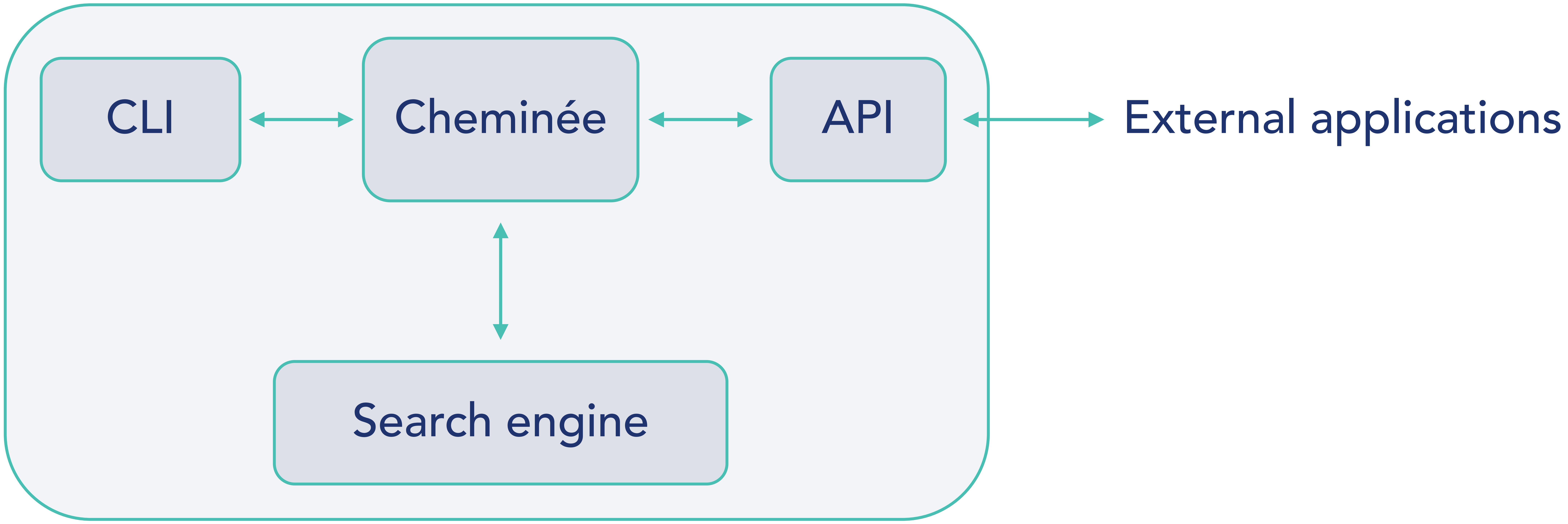
   Using the Rust-y RDKit for a structure search application

4. Future Directions

Cheminée is our structure search engine



Cheminée has its own CLI, API, and search engine



Cheminée has several endpoints

API

Indexing

- Create/delete index
- Bulk compound index
- Bulk compound delete

Text and Descriptor Search

- Basic search
- Descriptor search

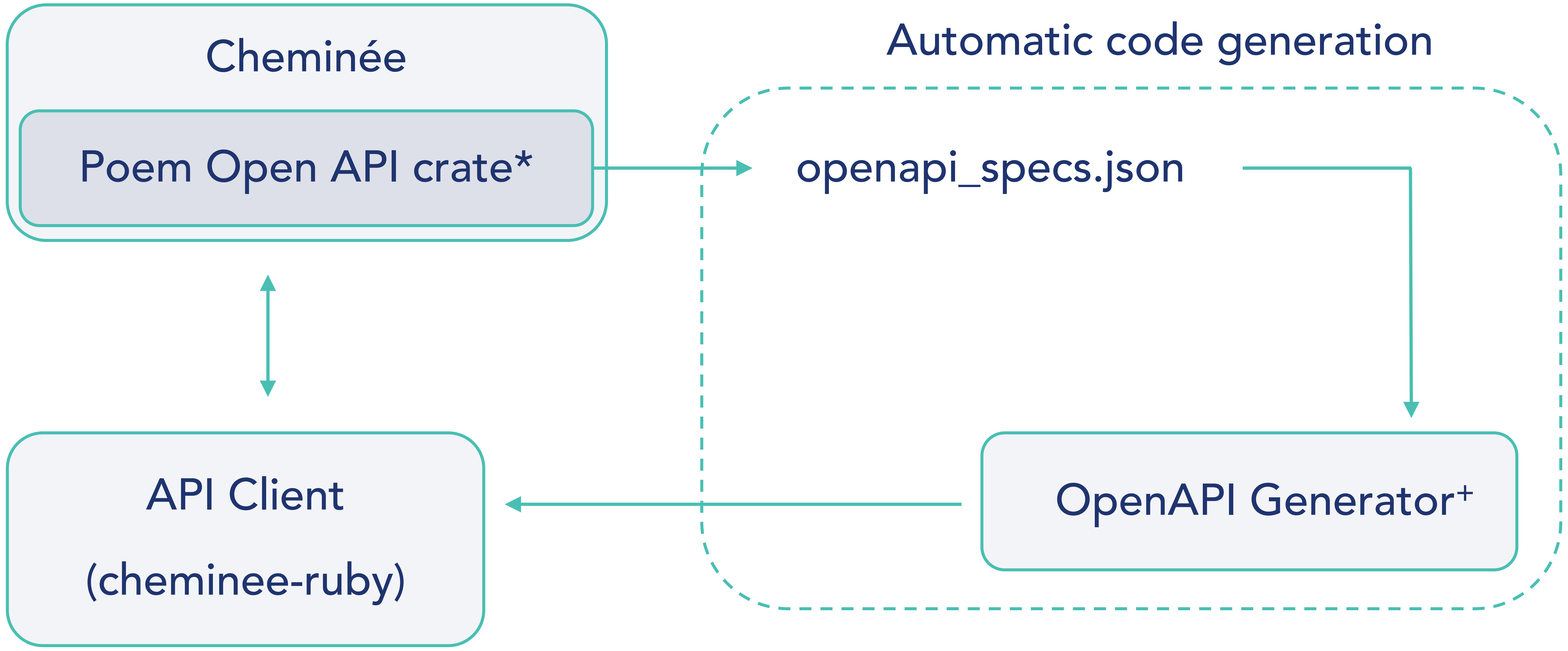
Compound Processing

- Standardization
- Smiles-to-molblock
- Molblock-to-smiles

Structure Search

- Exact structure
- Substructure
- Superstructure
- Similarity (in progress)

The Cheminée API works smarter, not harder



*https://docs.rs/poem-openapi

+https://github.com/OpenAPITools/openapi-generator

Cheminée does not use the RDKit Postgres cartridge

- ✓ Direct incorporation of the Tantivy indexing/search engine Rust crate
- ✓ Enables straight-forward multi-threaded indexing and searches
- ✓ Allows easier set-up within our software development stack
- ✓ Main drawback is not having Boost serialization of molecules → we try to avoid unnecessary re-parsing of molecules

Current schema breakdown

SMILES

- Standardized/canonicalized

Descriptors

- Integer fields (e.g. NumAtoms, NumHBD)
- Float fields (e.g. exactmw, TPSA)

Fingerprint

- Stored as bytes field

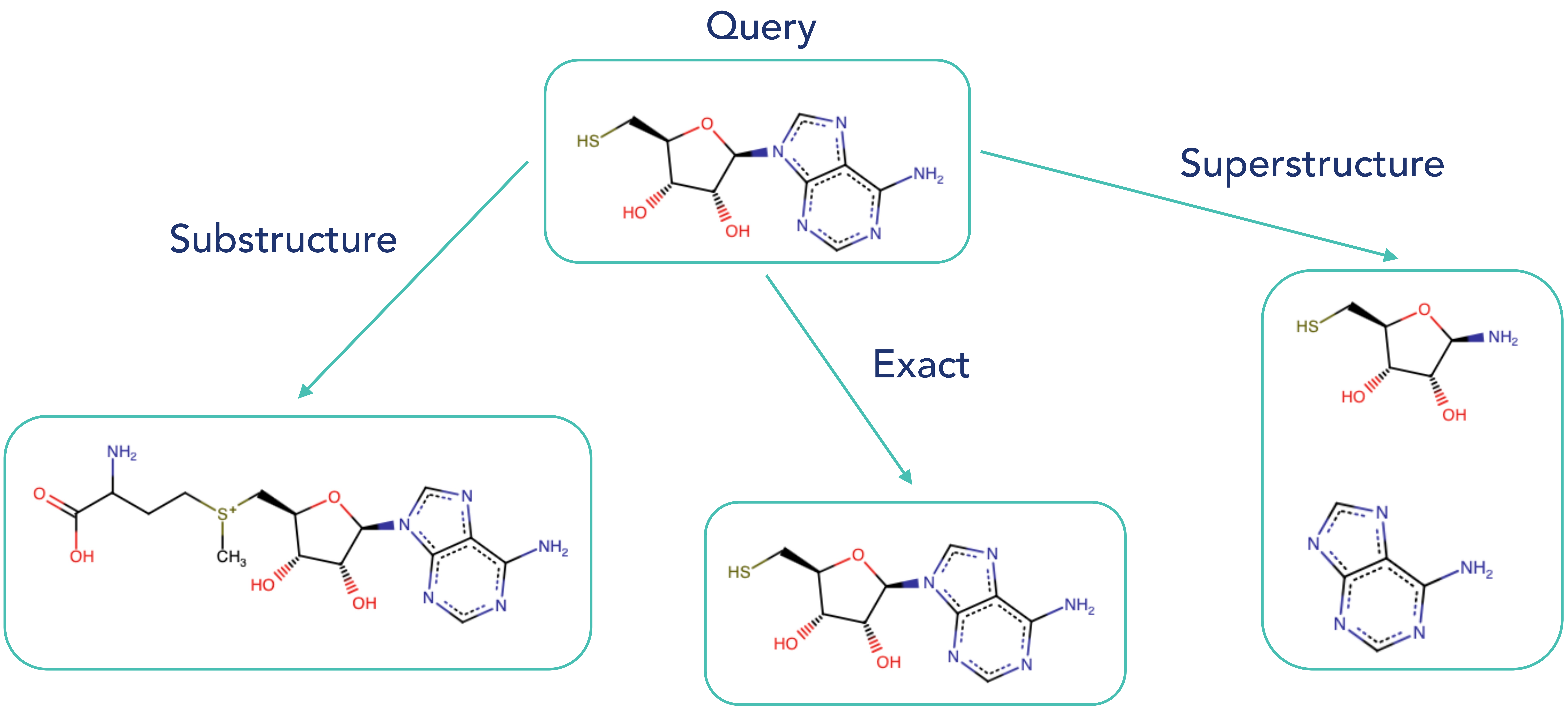
Miscellaneous metadata

- Stored as nested json field
- Very flexible
- E.g. { "orgs": [1, 2, 3], "amounts_mg": [1, 5, 10], "scaffolds": [10, 12, 13] }

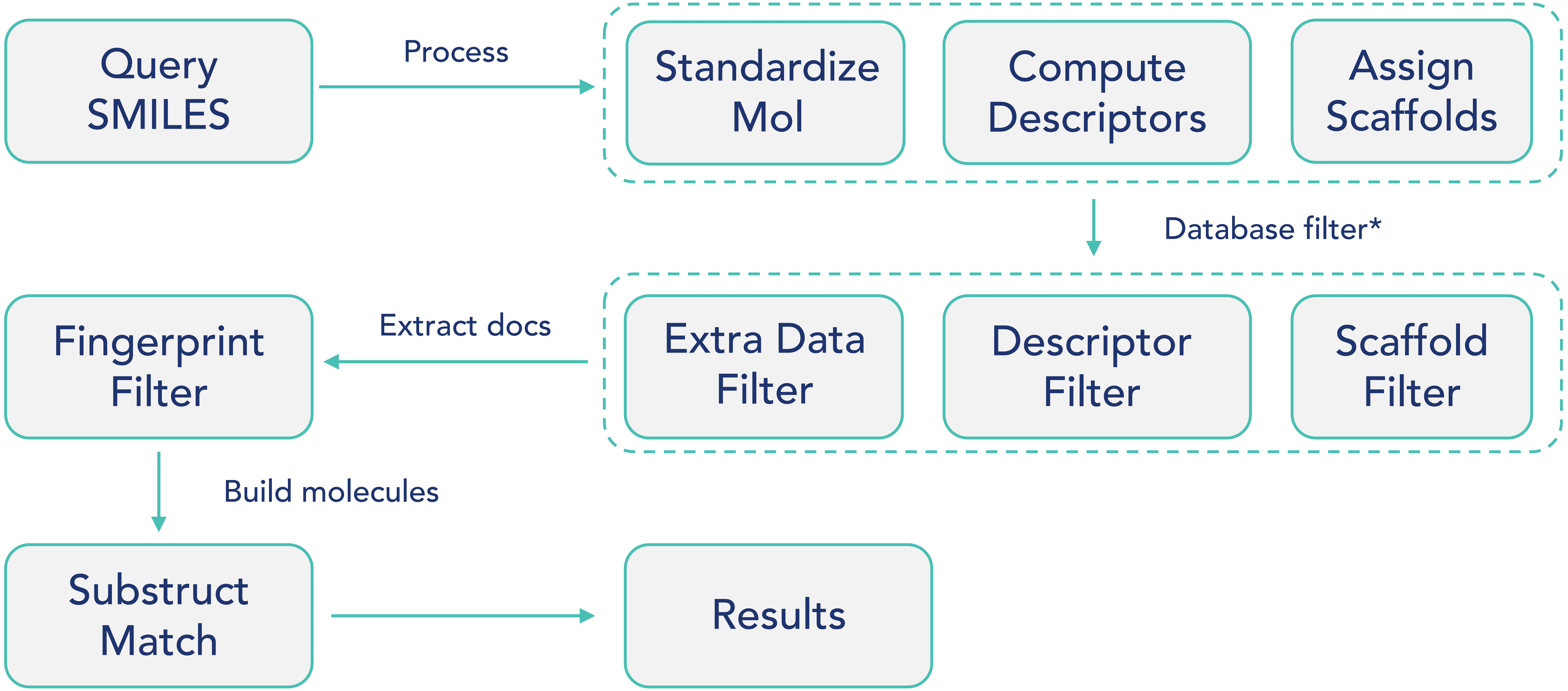
Cheminée indexes scaffolds

- ✓ Currently not using a graph database architecture (potential improvement for the future)
- ✓ Instead, we use an empirical set of 1000 scaffolds to speed up structure searches
    - ❑ Run ScaffoldNetwork on 100K randomized PubChem compounds
    - ❑ Rank scaffolds according to prevalence
- ✓ For every indexed compound, assign scaffolds from the empirical set

Structure searches



Substructure search breakdown



*Tautomers processed in parallel

Cheminée can likely get faster

- ✓ Indexing
  - ❑ ~2 million compounds per hour (with automated multithreading)
- ✓ Substructure searches*
  - ❑ ~10-150 milliseconds
- ✓ Superstructure searches*
  - ❑ ~10-150 milliseconds

*Using a test dataset of 5 million compounds

Overview

1. Scientist.com

   Company context and structure search requirements

2. The Rust-y RDKit

   Low-level Rust bindings for the RDKit C++

3. Cheminée

   Using the Rust-y RDKit for a stand-alone structure search application

4. Future Directions

Improvements in the queue

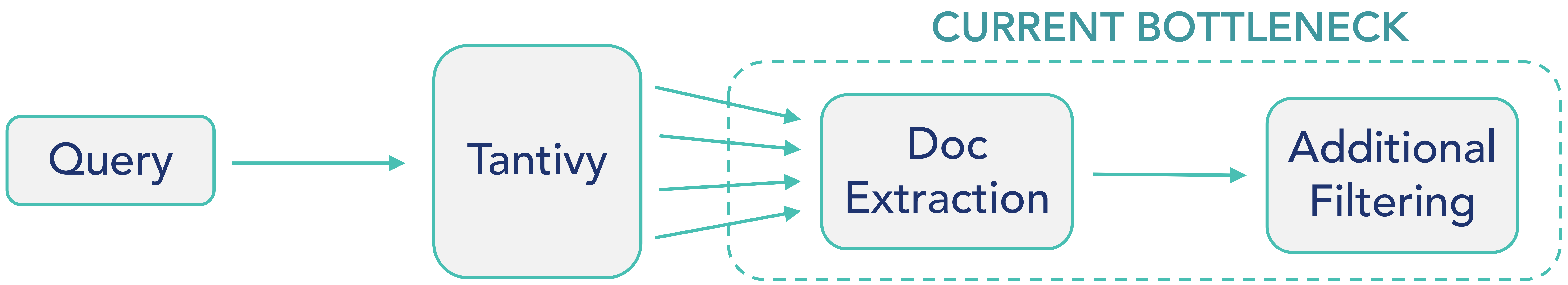
✓ Faster indexing

❑ Scaffold assignment involves 1000 substructure matches per compound

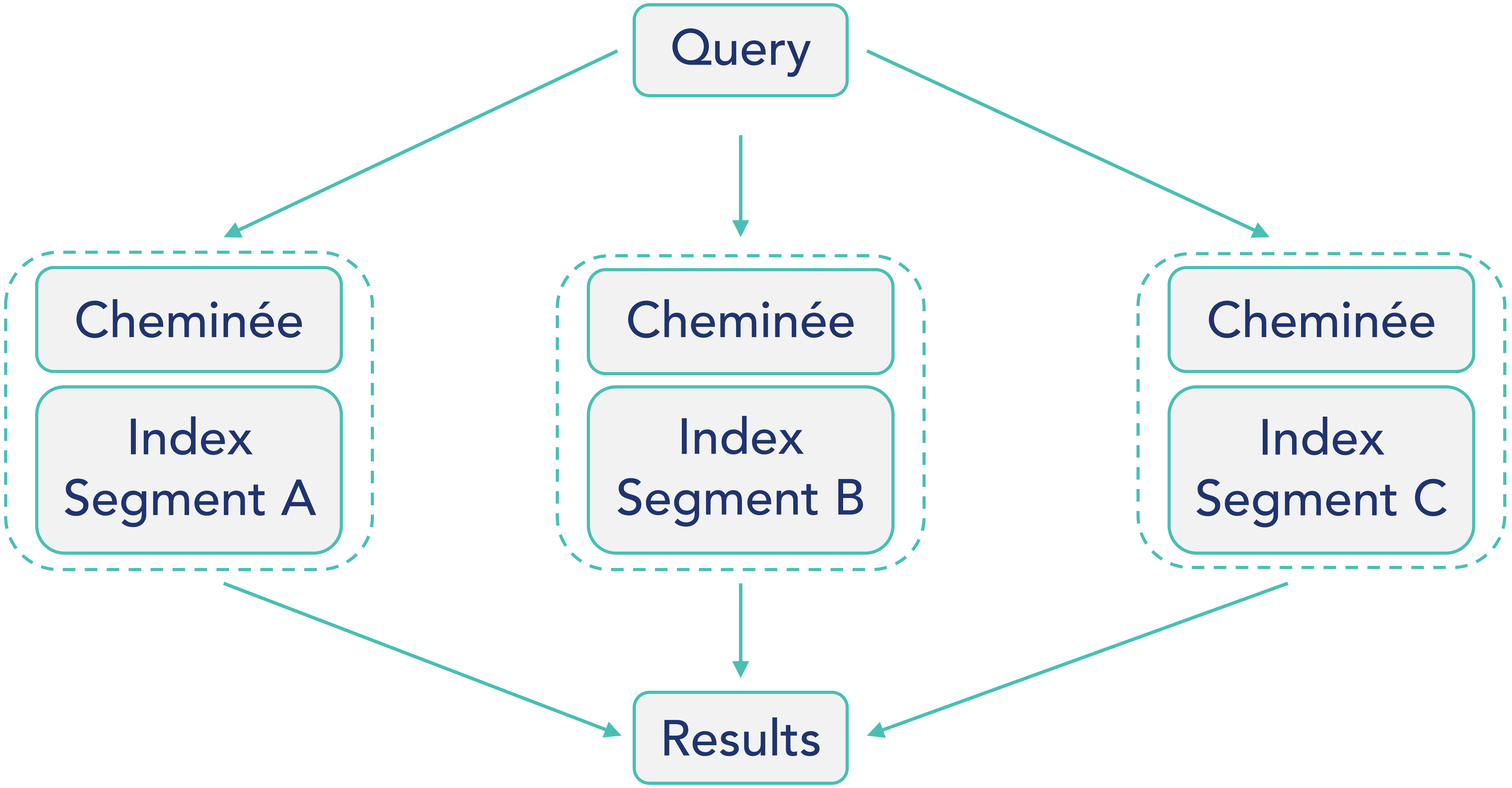
- Probably unnecessary brute force…

✓ Faster searching

❑ Push more filtering into Tantivy before doc extraction



Future direction: A horizontally scaled Cheminée*



*Especially for very large libraries

Acknowledgments

- ☆ Xavier Lange
- ☆ Scientist.com
- ☆ RDKit developers
- ☆ RDKit meeting organizers

Contributions are welcome!

The Rust-y RDKit

- ✓ Check out our repo: https://github.com/rdkit-rs/rdkit
- ✓ Check out the bindings: https://docs.rs/rdkit

Cheminée

- ✓ Check out our repo: https://github.com/rdkit-rs/cheminee
- ✓ Spin up our docker container image for some quick testing*

```
*docker run --rm -dt -p 4001:4001 --name cheminee ghcr.io/rdkit-rs/cheminee:0.1.31
```