



engineering Python-to-Fortran bindings
in C++, for use in Julia and Matlab

Sylwester Arabas¹, Zach D'Aquino², Jeff Curtis², Nicole Riemer², Matt West³
& [Py]PartMC contributors



agh.edu.pl



FOSDEM '24, ULB, Brussels



atmos.illinois.edu

¹Physics & Applied CS, AGH University of Krakow, Poland (agh.edu.pl)

²Atmospheric Sciences, University of Illinois at Urbana-Champaign (atmos.illinois.edu)

³Mechanical Science & Engineering, University of Illinois at Urbana-Champaign (mechse.illinois.edu)



engineering **Python-to-Fortran** bindings
in **C++**, for use in **Julia** and **Matlab**

Sylwester Arabas¹, Zach D'Aquino², Jeff Curtis², Nicole Riemer², Matt West³
& [Py]PartMC contributors



agh.edu.pl



FOSDEM '24, ULB, Brussels



atmos.illinois.edu

¹Physics & Applied CS, AGH University of Krakow, Poland (agh.edu.pl)

²Atmospheric Sciences, University of Illinois at Urbana-Champaign (atmos.illinois.edu)

³Mechanical Science & Engineering, University of Illinois at Urbana-Champaign (mechse.illinois.edu)

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary



<https://lagrange.mechse.illinois.edu/partmc/>



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2
- ▶ developed at Univ. Illinois Urbana-Champaign (v1.0.0 back in 2007)



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2
- ▶ developed at Univ. Illinois Urbana-Champaign (v1.0.0 back in 2007)
- ▶ "box model" (process studies) with a coupler to WRF (weather prediction model)



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2
- ▶ developed at Univ. Illinois Urbana-Champaign (v1.0.0 back in 2007)
- ▶ "box model" (process studies) with a coupler to WRF (weather prediction model)
- ▶ **simulating air pollution evolution through particle coagulation, condensation, chemical reactions, ...**



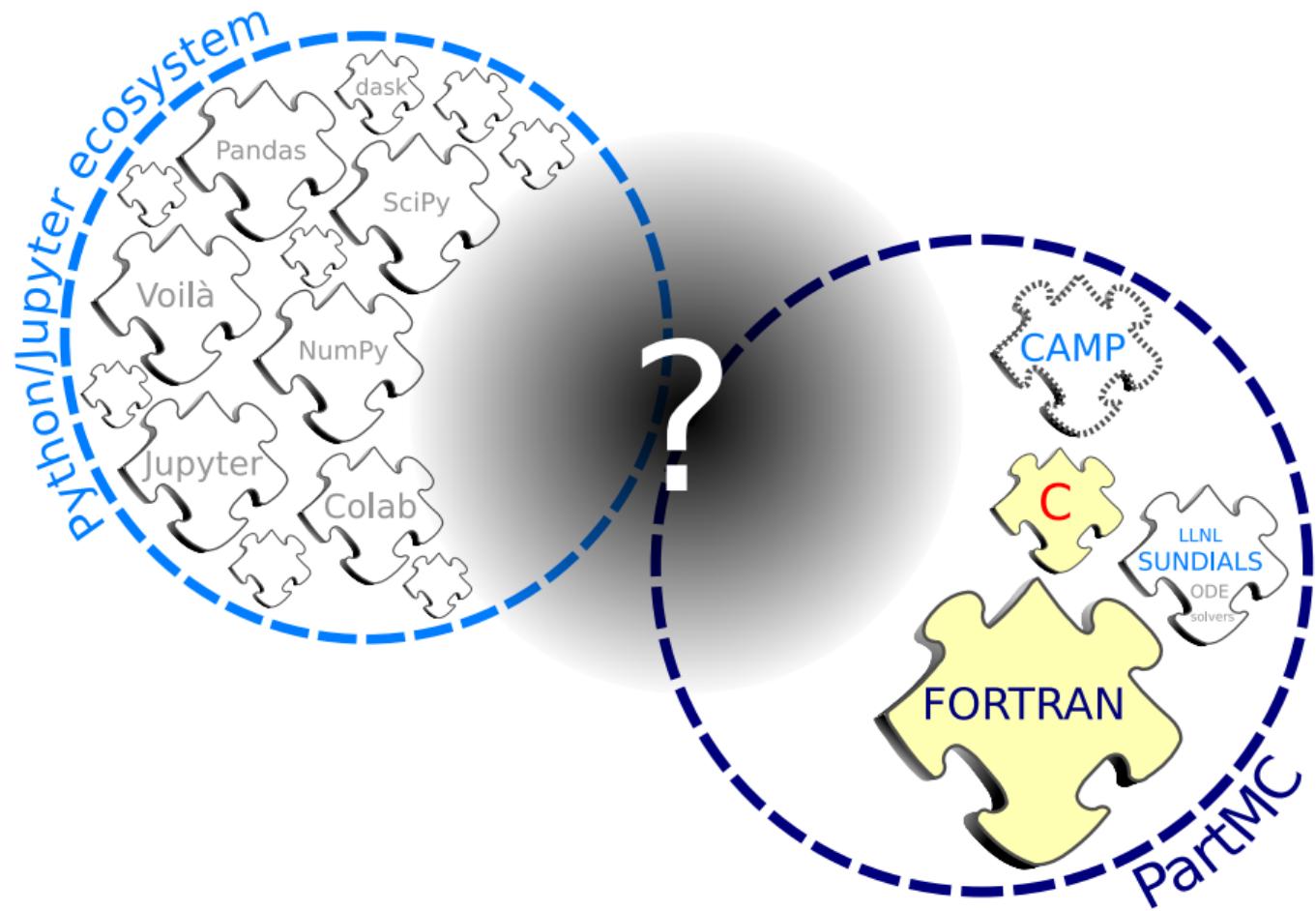
<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2
- ▶ developed at Univ. Illinois Urbana-Champaign (v1.0.0 back in 2007)
- ▶ "box model" (process studies) with a coupler to WRF (weather prediction model)
- ▶ simulating air pollution evolution through particle coagulation, condensation, chemical reactions, ...
- ▶ **object-oriented architecture, F90, extensive automated test suite**



<https://lagrange.mechse.illinois.edu/partmc/>

- ▶ Monte-Carlo aerosol dynamics simulation package
- ▶ open source, GPLv2
- ▶ developed at Univ. Illinois Urbana-Champaign (v1.0.0 back in 2007)
- ▶ "box model" (process studies) with a coupler to WRF (weather prediction model)
- ▶ simulating air pollution evolution through particle coagulation, condensation, chemical reactions, ...
- ▶ object-oriented architecture, F90, extensive automated test suite
- ▶ **usage poses challenges, e.g., to students intending to use it from Jupyter notebooks** (dependencies, compilation, updates, automation usually through shell, multi-text-file i/o, output analysis requiring bringing in Fortran, ...)



plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

project goals

- ▶ **lower the entry threshold for installing and setting up of PartMC**
down to `pip install PyPartMC`, i.e., no manual dependency installation,
no compilation, user doesn't even need to know FORTRAN is involved

project goals

- ▶ **lower the entry threshold for installing and setting up of PartMC**
down to `pip install PyPartMC`, i.e., no manual dependency installation,
no compilation, user doesn't even need to know FORTRAN is involved
- ▶ ensure the same experience on Linux, macOS & Windows

project goals

- ▶ **lower the entry threshold for installing and setting up of PartMC**
down to `pip install PyPartMC`, i.e., no manual dependency installation,
no compilation, user doesn't even need to know FORTRAN is involved
- ▶ ensure the same experience on Linux, macOS & Windows
- ▶ **lower the entry threshold for usage with Jupyter-based example notebooks**

project goals

- ▶ **lower the entry threshold for installing and setting up of PartMC**
down to `pip install PyPartMC`, i.e., no manual dependency installation,
no compilation, user doesn't even need to know FORTRAN is involved
- ▶ ensure the same experience on Linux, macOS & Windows
- ▶ **lower the entry threshold for usage with Jupyter-based example notebooks**
- ▶ streamline the dissemination of paper-result reproducers (peer review)

status of the project: v1.0 in Dec 2023 (started 2021)

SoftwareX 

ORIGINAL SOFTWARE PUBLICATION

PyPartMC: A Pythonic interface to a particle-resolved, Monte Carlo aerosol simulation framework

Zachary D'Aquino • Sylwester Arabas • Jeffrey H. Curtis • Akshunna Vaishnav • Nicole Riemer   • Matthew West

Open Access • Published: December 23, 2023 • DOI: <https://doi.org/10.1016/j.softx.2023.101613>





[Help](#) [Sponsors](#) [Log in](#) [Register](#)

PyPartMC 1.0.2

 Latest version

```
pip install PyPartMC
```



Released: Jan 31, 2024



7/22

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

pybind11

master ▾ 27 Branches 58 Tags ⌂ ⌂ Go to file Go to file ⌂ Code ⌂ ...

README License Security ⌂

pybind11

pybind11 — Seamless operability between C++11 and Python

docs passing docs stable chat on gitter Discussions Ask CI passing build passing

latest packaged version 2.11.1 pypi v2.11.1 conda-forge v2.11.0 python 3.6 | 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12

[Setup tools example](#) • [SciKit-build example](#) • [CMake example](#)

pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent [Boost.Python](#) library by David Abrahams: to minimize boilerplate code in traditional extension modules by inferring type information using compile-time introspection.

About

Seamless operability between C++11 and Python

[pybind11.readthedocs.io/](#)

#python #bindings

Readme View license Security policy Activity Custom properties

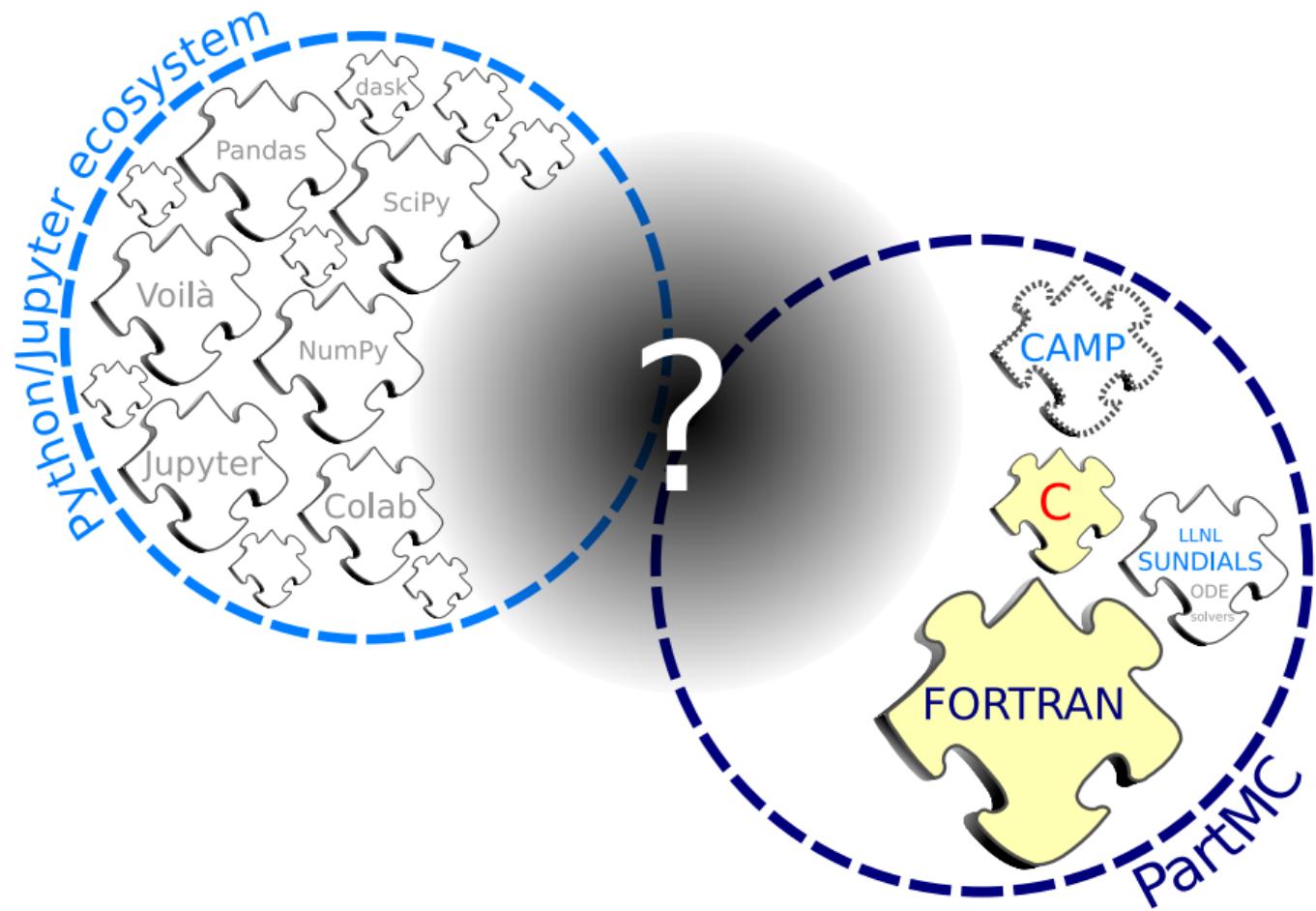
14.3k stars 250 watching 2.1k forks

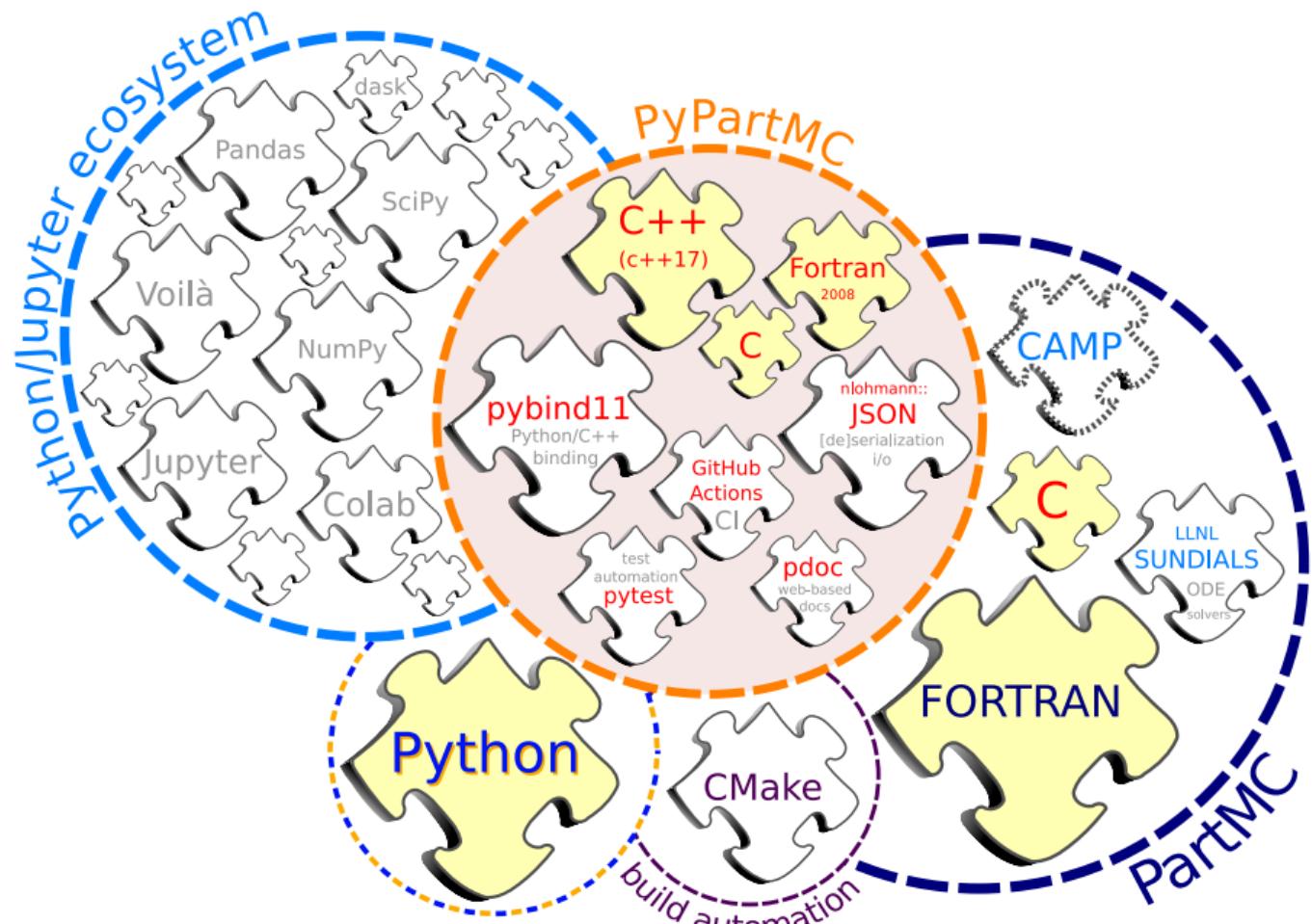
Report repository

Releases 21

Version 2.11.1 Latest on Jul 17, 2023 + 20 releases

Contributors 337





developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types),
Python bindings generated using **pybind11**

developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types),
Python bindings generated using **pybind11**
- ▶ three-language build automation with CMake, test automation with pytest, CI workflows

developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC internals (derived types), Python bindings generated using **pybind11**
- ▶ three-language build automation with CMake, test automation with pytest, CI workflows
- ▶ **JSON-based reimplementation of PartMC "spec-file" i/o module**
(unmodified code of PartMC uses original API)
 - ~~ minimising effort to accommodate future additions to PartMC

developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC internals (derived types), Python bindings generated using **pybind11**
- ▶ three-language build automation with CMake, test automation with pytest, CI workflows
- ▶ JSON-based reimplementation of PartMC "spec-file" i/o module
(unmodified code of PartMC uses original API)
~~ minimising effort to accommodate future additions to PartMC
- ▶ freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector

developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC internals (derived types), Python bindings generated using **pybind11**
- ▶ three-language build automation with CMake, test automation with pytest, CI workflows
- ▶ JSON-based reimplementation of PartMC "spec-file" i/o module
(unmodified code of PartMC uses original API)
~~ minimising effort to accommodate future additions to PartMC
- ▶ freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector
- ▶ **dependency version pinning with git submodules:** PartMC (F), CAMP (C/F), json (C++), pybind11 (C++), json-fortran (F), netCDF (C/F), SUNDIALS (F/C), SuiteSparse (C), ... & backports of C++20 features to C++17 (multilinx!): span, string_view, optional

developer perspective

- ▶ written in C/Fortran/C++ as **C++ bindings** to PartMC internals (derived types), Python bindings generated using **pybind11**
- ▶ three-language build automation with CMake, test automation with pytest, CI workflows
- ▶ JSON-based reimplementation of PartMC "spec-file" i/o module
(unmodified code of PartMC uses original API)
~~ minimising effort to accommodate future additions to PartMC
- ▶ freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector
- ▶ dependency version pinning with git submodules: PartMC (F), CAMP (C/F), json (C++), pybind11 (C++), json-fortran (F), netCDF (C/F), SUNDIALS (F/C), SuiteSparse (C), ... & backports of C++20 features to C++17 (multilinx!): span, string_view, optional
- ▶ **all dependencies (incl. Fortran and C++ runtimes) statically linked (single-file install)**

user perspective: Fortran (PartMC)

c: Fortran code

```
program main
  use pmc_spec_file
  use pmc_aero_data
  use pmc_aero_mode
  use pmc_aero_dist
  use pmc_aero_state

  implicit none

  type(spec_file_t) :: f_aero_data, f_aero_dist
  type(aero_data_t) :: aero_data
  type(aero_dist_t) :: aero_dist
  type(aero_state_t) :: aero_state
  integer, parameter :: n_part = 100
  integer :: n_part_add
  real(kind=dp), dimension(n_part) :: num_concs, masses

  call spec_file_open("aero_data.dat", f_aero_data)
  call spec_file_read_aero_data(f_aero_data, aero_data)
  call spec_file_close(f_aero_data)

  call spec_file_open("aero_dist.dat", f_aero_dist)
  call spec_file_read_aero_dist(f_aero_dist, aero_data, aero_dist)
  call spec_file_close(f_aero_dist)

  call aero_state_zero(aero_state)
  call fractal_set_spherical(aero_data%fractal)
  call aero_state_set_weight(aero_state, aero_data, &
    AERO_STATE_WEIGHT_NUMMASS_SOURCE)
  call aero_state_set_n_part_ideal(aero_state, dble(n_part))
  call aero_state_add_aero_dist_sample(aero_state, aero_data, &
    aero_dist, 1d0, 0d0, .true., .true., n_part_add)

  num_concs = aero_state_num_concs(aero_state, aero_data)
  masses = aero_state_masses(aero_state, aero_data)
  print *, dot_product(num_concs, masses), "# kg/m3"
end
```

d: aero_dist.dat file (for Fortran code)

```
mode_name cooking
mass_frac cooking_comp.dat
diam_type geometric
mode_type log_normal
num_conc 3.2e9 # (#/m^3)
geom_mean_diam 8.64e-9 # (m)
log10_geom_std_dev 0.28

mode_name diesel
mass_frac diesel_comp.dat
diam_type geometric
mode_type log_normal
num_conc 2.9e9 # (#/m^3)
geom_mean_diam 5e-8
log10_geom_std_dev 0.24
```

e: cooking_comp.dat file (for Fortran code)

```
#      proportion
OC          1
```

f: diesel_comp.dat file (for Fortran code)

```
#      proportion
OC          0.3
BC          0.7
```

user perspective: Python (PyPartMC)

a: Python code (with embedded data)

```
import numpy as np

import PyPartMC as ppmc
from PyPartMC import si

aero_data = ppmc.AeroData({
    # [density, ions in solution, molecular weight, kappa]
    {"OC": [1000 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0.001]},
    {"BC": [1800 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0]}, 
})

aero_dist = ppmc.AeroDist(
    aero_data,
    [
        {
            "cooking": {
                "mass_frac": [{"OC": [1]}],
                "diam_type": "geometric",
                "mode_type": "log_normal",
                "num_conc": 3200 / si.cm**3,
                "geom_mean_diam": 8.64 * si.nm,
                "log10_geom_std_dev": 0.28,
            }
        },
        {
            "diesel": {
                "mass_frac": [{"OC": [0.3]}, {"BC": [0.7]}],
                "diam_type": "geometric",
                "mode_type": "log_normal",
                "num_conc": 2900 / si.cm**3,
                "geom_mean_diam": 50 * si.nm,
                "log10_geom_std_dev": 0.24,
            }
        },
    ],
)

n_part = 100
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source")
aero_state.dist_sample(aero_dist)
print(np.dot(aero_state.masses, aero_state.num_concs), "# kg/m3")
```

user perspective: Python (PyPartMC) & Julia (via PyCall.jl)

a: Python code (with embedded data)

```
import numpy as np

import PyPartMC as ppmc
from PyPartMC import si

aero_data = ppmc.AeroData(
    #           density, ions in solution, molecular weight, kappa
    {"OC": [1000 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0.001]},
    {"BC": [1800 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0]},
)

aero_dist = ppmc.AeroDist(
    aero_data,
    [
        {
            "cooking": {
                "mass_frac": [{"OC": [1]}],
                "diam_type": "geometric",
                "mode_type": "log_normal",
                "num_conc": 3200 / si.cm**3,
                "geom_mean_diam": 8.64 * si.nm,
                "log10_geom_std_dev": 0.28,
            }
        },
        {
            "diesel": {
                "mass_frac": [{"OC": [0.3]}, {"BC": [0.7]}],
                "diam_type": "geometric",
                "mode_type": "log_normal",
                "num_conc": 2900 / si.cm**3,
                "geom_mean_diam": 50 * si.nm,
                "log10_geom_std_dev": 0.24,
            }
        },
    ],
)

n_part = 100
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source")
aero_state.dist_sample(aero_dist)
print(np.dot(aero_state.masses, aero_state.num_concs), "# kg/m3")
```

b: Julia code (with embedded data)

```
using Pkg
Pkg.add("PyCall")

using PyCall
ppmc = pyimport("PyPartMC")
si = ppmc["si"]

aero_data = ppmc.AeroData(
    #           density, ions in solution, molecular weight, kappa
    Dict("OC"=>(1000 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0.001)),
    Dict("BC"=>(1800 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0))
)

aero_dist = ppmc.AeroDist(aero_data, (
    Dict(
        "cooking" => Dict(
            "mass_frac" => (Dict("OC" => (1,)),),
            "diam_type" => "geometric",
            "mode_type" => "log_normal",
            "num_conc" => 3200 / si.cm^3,
            "geom_mean_diam" => 8.64 * si.nm,
            "log10_geom_std_dev" => .28,
        ),
        "diesel" => Dict(
            "diesel" => Dict(
                "mass_frac" => (Dict("OC" => (.3,)), Dict("BC" => (.7,))),
                "diam_type" => "geometric",
                "mode_type" => "log_normal",
                "num_conc" => 2900 / si.cm^3,
                "geom_mean_diam" => 50 * si.nm,
                "log10_geom_std_dev" => .24,
            )
        )
    )
))

n_part = 100
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source")
aero_state.dist_sample(aero_dist)
print(aero_state.masses*aero_state.num_concs, "# kg/m3")
```

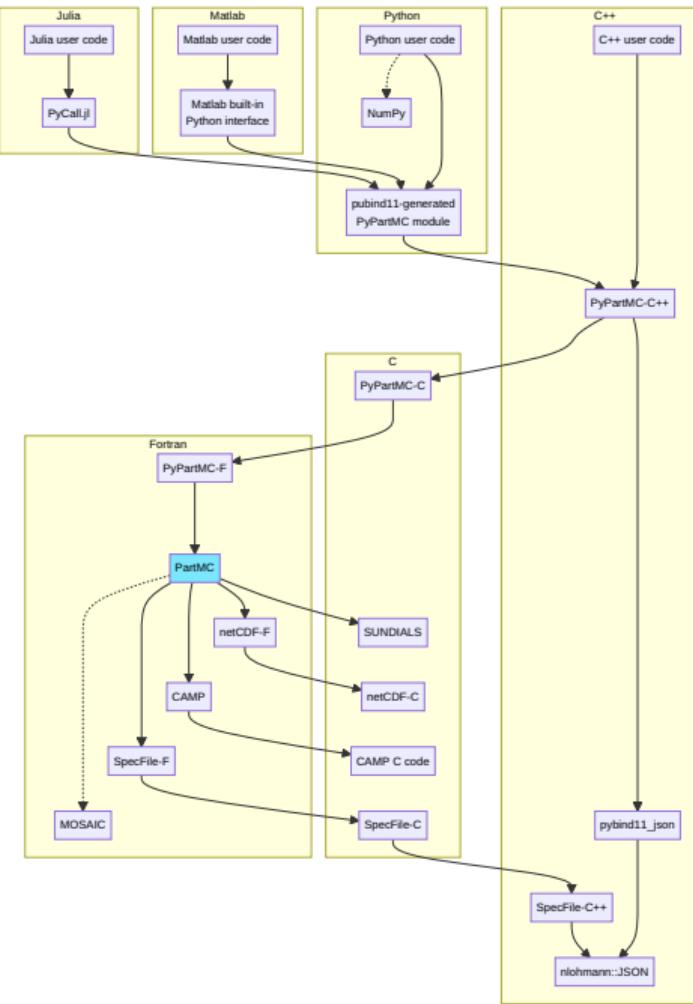
user perspective: Matlab (built-in Python bridge)

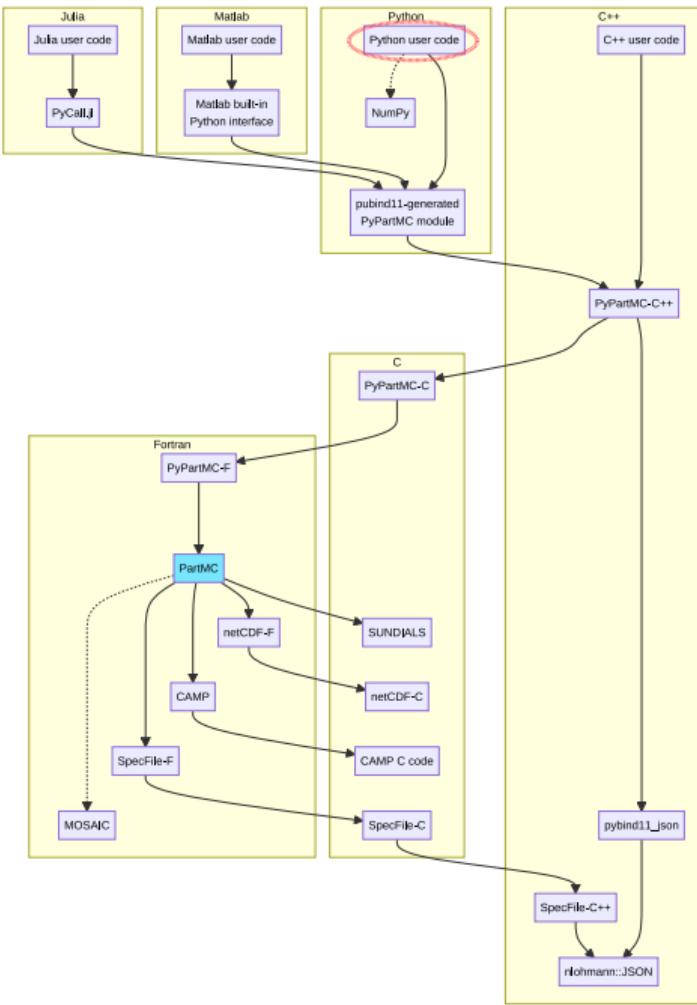
```
ppmc = py.importlib.import_module('PyPartMC');
si = py.importlib.import_module('PyPartMC').si;

aero_data = ppmc.AeroData(py.tuple({ ...
    py.dict(pyargs("OC", py.tuple({1000 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0.001}))), ...
    py.dict(pyargs("BC", py.tuple({1800 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0})))) ...
}));

aero_dist = ppmc.AeroDist(aero_data, py.tuple({ ...
    py.dict(pyargs( ...
        "cooking", py.dict(pyargs( ...
            "mass_frac", py.tuple({py.dict(pyargs("OC", py.tuple({1}))}), ...
            "diam_type", "geometric", ...
            "mode_type", "log_normal", ...
            "num_conc", 3200 / si.cm^3, ...
            "geom_mean_diam", 8.64 * si.nm, ...
            "log10_geom_std_dev", .28 ...
        )) ...
    )), ...
    py.dict(pyargs( ...
        "diesel", py.dict(pyargs( ...
            "mass_frac", py.tuple({ ...
                py.dict(pyargs("OC", py.tuple({.3}))), ...
                py.dict(pyargs("BC", py.tuple({.7}))), ...
            }), ...
            "diam_type", "geometric", ...
            "mode_type", "log_normal", ...
            "num_conc", 2900 / si.cm^3, ...
            "geom_mean_diam", 50 * si.nm, ...
            "log10_geom_std_dev", .24 ...
        )) ...
    )))
  }));
});

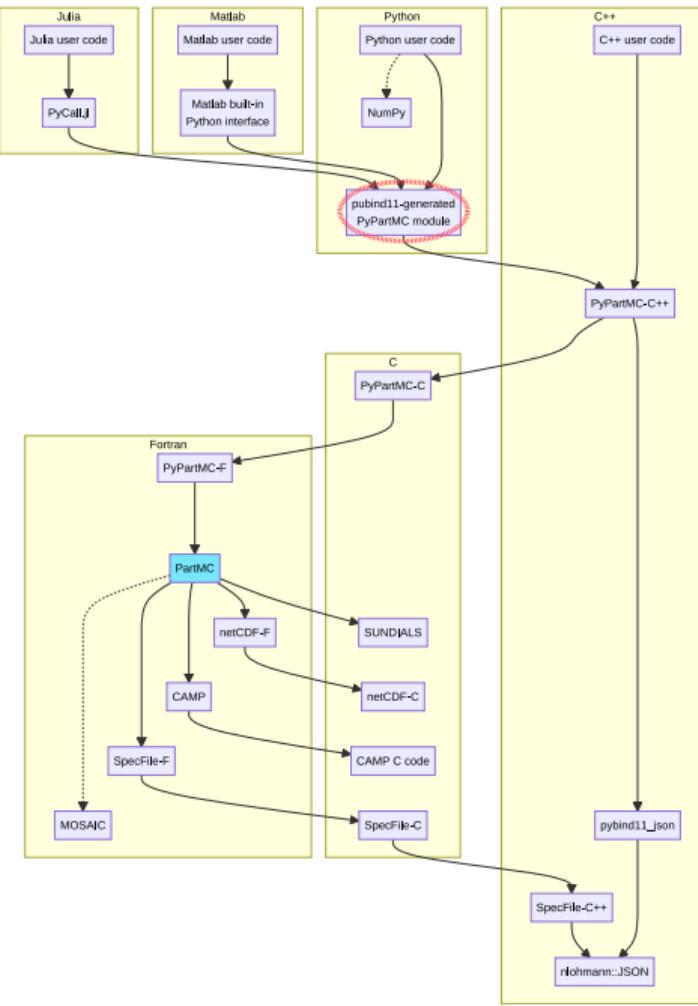
n_part = 100;
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source");
aero_state.dist_sample(aero_dist);
masses = cell(aero_state.masses());
num_concs = cell(aero_state.num_concs);
fprintf('%g # kg/m3\n', dot([masses{:}], [num_concs{:}]));
```





```
import PyPartMC as ppmc
from PyPartMC import si

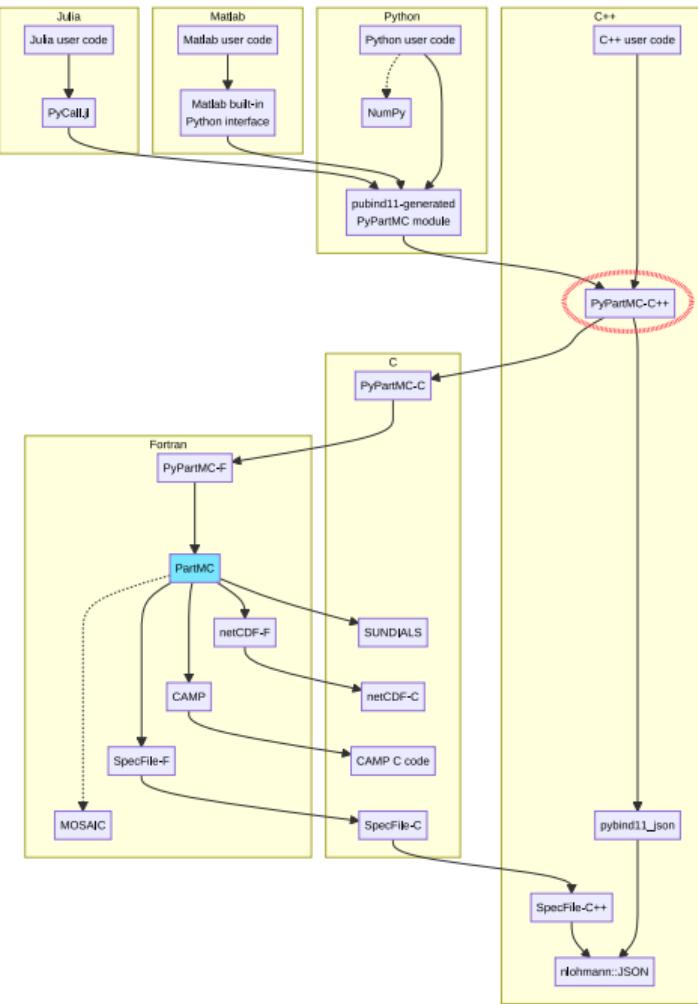
aero_data = ppmc.AeroData(
    # [density, ions in solution, molecular weight, kappa]
    {"OC": [1000 * si.kg/si.m**3, 0, 1e-3 * si.kg/si.mol, 0.001]},
    {"BC": [1800 * si.kg/si.m**3, 0, 1e-3 * si.kg/si.mol, 0]},  
    ))
```



```
import os
from contextlib import contextmanager
from pathlib import Path

# https://github.com/diegofering/cmake-build-extension/blob/master/src/cmake_build_extension
@contextmanager
def _build_extension_env():
    cookies = []
    # https://docs.python.org/3/whatsnew/3.8.html#bpo-36085-whatsnew
    if hasattr(os, "add_dll_directory"):
        basepath = os.path.dirname(os.path.abspath(__file__))
        dlspath = os.path.join(basepath, "..")
        os.environ["PATH"] = dlspath + os.pathsep + os.environ["PATH"]
        for path in os.environ.get("PATH", "").split(os.pathsep):
            if path and Path(path).is_absolute() and Path(path).is_dir():
                cookies.append(os.add_dll_directory(path))
    try:
        yield
    finally:
        for cookie in cookies:
            cookie.close()

with _build_extension_env():
    import PyPartMC
    from PyPartMC import *
    from _PyPartMC import (
        __all__,
        __version__,
        __versions_of_build_time_dependencies__,
    )
```



```
#include "pybind11/pybind11.h"
#include "nlohmann/json.hpp"
#include "pybind11_json/pybind11_json.hpp"

[...]

#include "aero_data.hpp"

[...]

namespace py = pybind11;

[...]

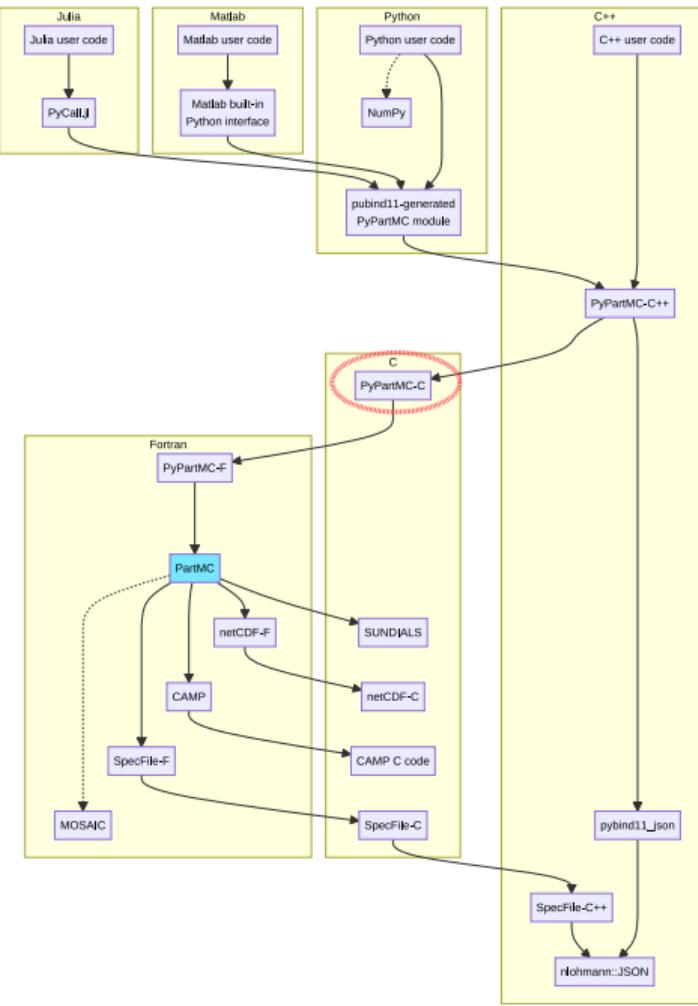
PYBIND11_MODULE(_PyPartMC, m) {

    [...]

    py::class_<
        AeroData,
        std::shared_ptr<AeroData>
    >(m, "AeroData")
        .def(py::init<const nlohmann::json&>())

    [...]

    m.attr("__all__") = py::make_tuple(
        "__version__",
        "AeroData",
        [...]
    );
}
```



```
#pragma once

#include "pmc_resource.hpp"
#include "gimmicks.hpp"

[...]

extern "C" void f_aero_data_ctor(void *ptr) noexcept;
extern "C" void f_aero_data_dtor(void *ptr) noexcept;
extern "C" void f_aero_data_from_json(const void *ptr) noexcept;

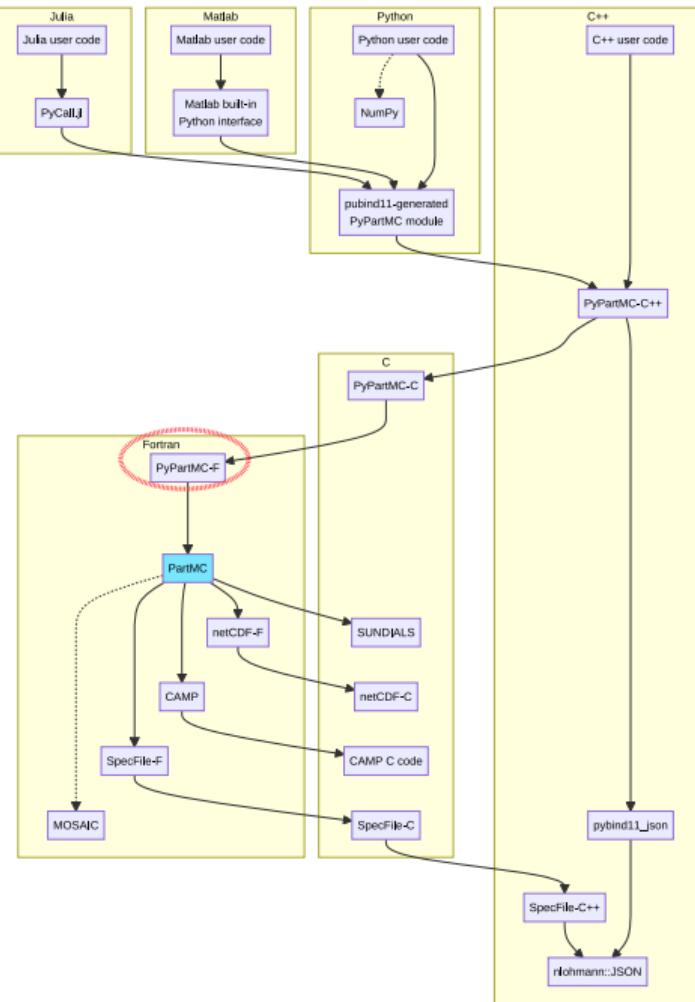
[...]

struct AeroData {
    PMCRResource ptr;

    AeroData(const nlohmann::json &json) :
        ptr(f_aero_data_ctor, f_aero_data_dtor)
    {
        if (!InputGimmick::unique_keys(json))
            throw std::runtime_error("Species names must be unique");

        GimmickGuard<InputGimmick> guard(json);
        f_aero_data_from_json(this->ptr.f_arg());
    }

    [...]
};
```



```

module PyPartMC_aero_data
use iso_c_binding
use pmc_aero_data
implicit none

contains

subroutine f_aero_data_ctor(ptr_c) bind(C)
    type(aero_data_t), pointer :: ptr_f => null()
    type(c_ptr), intent(out) :: ptr_c

    allocate(ptr_f)
    call fractal_set_spherical(ptr_f%fractal)
    ptr_c = c_loc(ptr_f)
end subroutine

subroutine f_aero_data_dtor(ptr_c) bind(C)
    type(aero_data_t), pointer :: ptr_f => null()
    type(c_ptr), intent(in) :: ptr_c

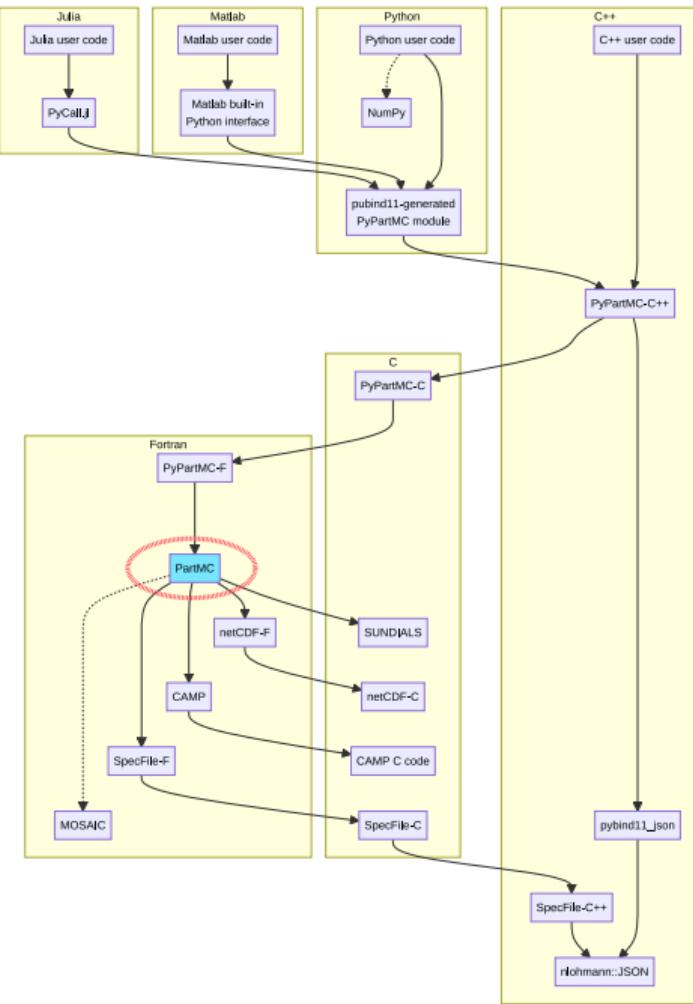
    call c_f_pointer(ptr_c, ptr_f)
    deallocate(ptr_f)
end subroutine

subroutine f_aero_data_from_json(ptr_c) bind(C)
    type(aero_data_t), pointer :: ptr_f => null()
    type(c_ptr), intent(in) :: ptr_c
    type(spec_file_t) :: file
    call c_f_pointer(ptr_c, ptr_f)
    call spec_file_read_aero_data(file, ptr_f)
end subroutine

[...]

end module

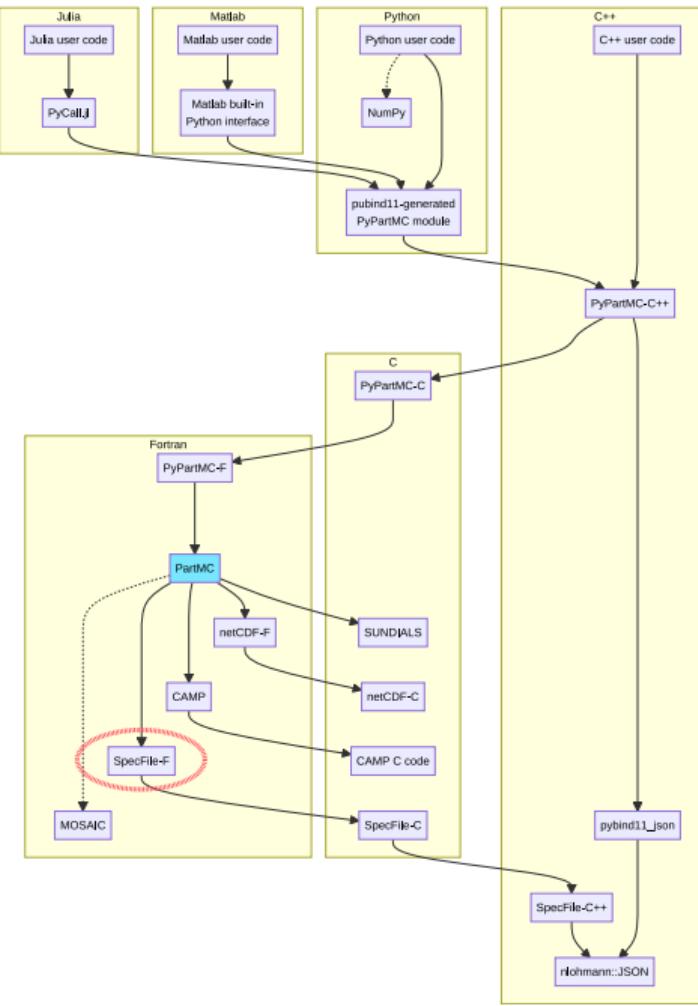
```



untouched PartMC code!

```
! Copyright (C) 2005-2012, 2016, 2021 Nicole Riemer and Matthew West
! Licensed under the GNU General Public License version 2 or (at your
! option) any later version. See the file COPYING for details.
```

```
[...]  
module pmc_aero_data  
use pmc_spec_file  
[...]  
contains  
[...]  
subroutine spec_file_read_aero_data(file, aero_data)  
[...]  
    type(spec_file_t), intent(inout) :: file  
    type(aero_data_t), intent(inout) :: aero_data  
    real(kind=dp), allocatable :: species_data(:, :)  
[...]  
    call spec_file_read_real_named_array(file, 0, species_name, species_data)  
[...]  
end subroutine spec_file_read_aero_data  
[...]  
end module pmc_aero_data
```

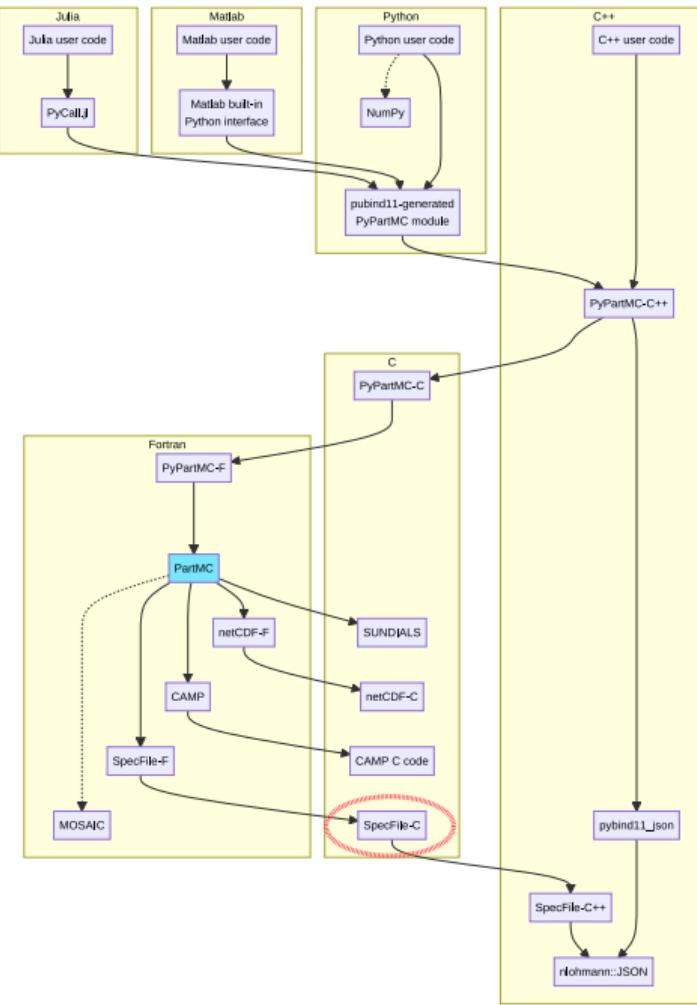


```

module pmc_spec_file
[...]
interface
[...]
subroutine c_spec_file_read_real_named_array_data( &
    row, &
    names_data, names_size, &
    vals_data, vals_size &
) bind(C)
    import c_double
    character, intent(in) :: names_data
    real(c_double), intent(out) :: vals_data
    integer, intent(in) :: row, vals_size, names_size
end subroutine

[...]
end interface
[...]
subroutine spec_file_read_real_named_array(file, max_lines, names, vals)
[...]
do row = 1, n_rows
[...]
    call c_spec_file_read_real_named_array_data( &
        row, &
        names(row), name_size, &
        vals_row(1), size(vals, 2) &
    )
[...]
end do
[...]
end subroutine
[...]
end module

```

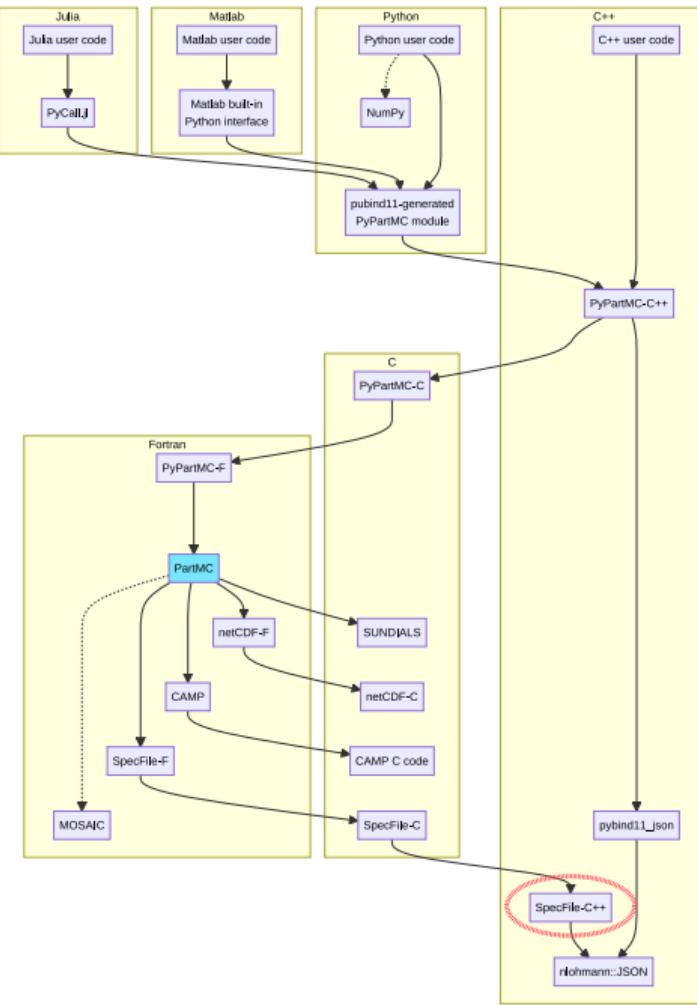


```

#include "gimmicks.hpp"
[...]
void spec_file_read_real_named_array_data(
    const unsigned int row,
    char *name_data,
    int *name_size,
    const tcb::span<double> &vals
) noexcept {
    auto i = 0u, n_numeric_array_entries = gimmick_ptr()->n_numeric_array_entries();
    for (auto it = gimmick_ptr()->begin(); i < n_numeric_array_entries; ++i, ++it)
    {
        assert(it->is_object());
        if (i == row-1) {
            assert(it->size() == 1);
            for (auto &entry : it->items()) {
                for (auto c=0u; c < entry.key().size(); ++c)
                    name_data[c] = entry.key()[c];
                *name_size = entry.key().size();
            }
            [...]
        }
    }
}

extern "C"
void c_spec_file_read_real_named_array_data(
    const int *row,
    char *name_data,
    int *name_size,
    double *vals_data,
    const int *vals_size
) noexcept {
    spec_file_read_real_named_array_data(
        *row,
        name_data, name_size,
        tcb::span<double>(vals_data, *vals_size)
    );
}
[...]

```

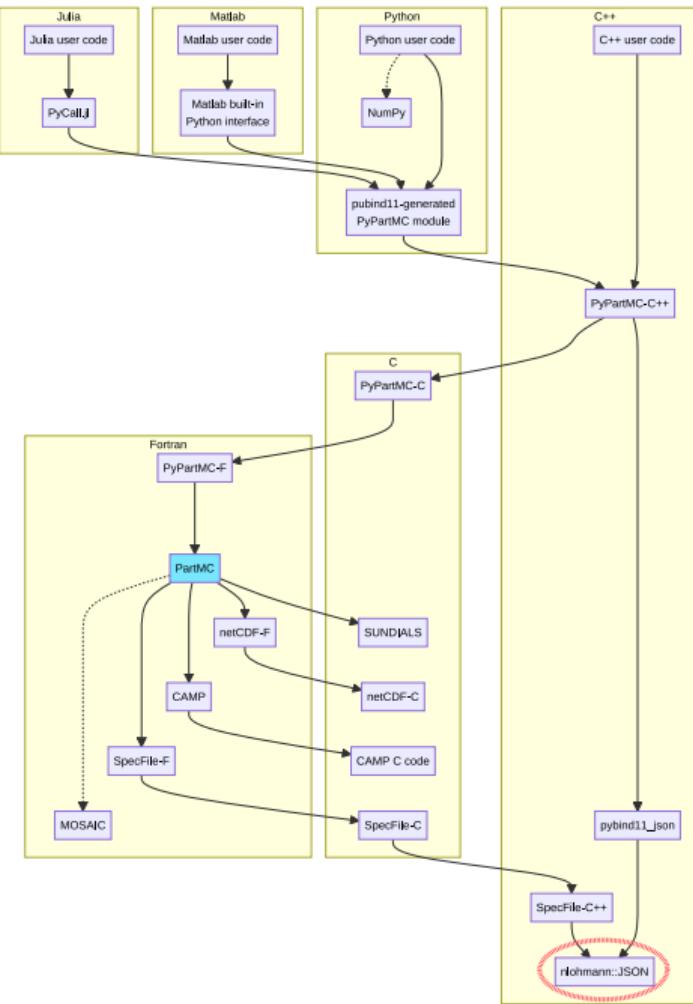


```

#include "gimmicks.hpp"
[...]
void spec_file_read_real_named_array_data(
    const unsigned int row,
    char *name_data,
    int *name_size,
    const tcb::span<double> &vals
) noexcept {
    auto i = 0u, n_numeric_array_entries = gimmick_ptr()->n_numeric_array_entries();
    for (auto it = gimmick_ptr()->begin(); i < n_numeric_array_entries; ++i, ++it)
    {
        assert(it->is_object());
        if (i == row-1) {
            assert(it->size() == 1);
            for (auto &entry : it->items()) {
                for (auto c=0u; c < entry.key().size(); ++c)
                    name_data[c] = entry.key()[c];
                *name_size = entry.key().size();
            }
            [...]
        }
    }
}

extern "C"
void c_spec_file_read_real_named_array_data(
    const int *row,
    char *name_data,
    int *name_size,
    double *vals_data,
    const int *vals_size
) noexcept {
    spec_file_read_real_named_array_data(
        *row,
        name_data, name_size,
        tcb::span<double>(vals_data, *vals_size)
    );
}
[...]

```



json.nlohmann.me

JSON for Modern C++

JSON for Modern C++

What if JSON was part of modern C++?

3.11.3

The screenshot shows a browser window displaying the PyPartMC API documentation at <https://open-atmos.github.io/PyPartMC/>. The left sidebar contains an 'Index' section and lists 'Functions' and 'Classes'. The main content area is titled 'Module PyPartMC' and describes PyPartMC as a Python interface to PartMC. It lists three functions: 'condense_equilib_particles', 'pow2_above', and 'run_part'. Below these are detailed descriptions and code snippets for each function. The 'AeroData' class is also briefly mentioned.

Index

Functions

- condense_equilib_particles
- pow2_above
- run_part

Classes

- AeroData
- AeroState
- EnvState
- GasData
- GasState
- RunPartOpt
- Scenario

Module PyPartMC

PyPartMC is a Python interface to PartMC.

Functions

```
def condense_equilib_particles(arg0: EnvState, arg1: AeroData,  
                                arg2: AeroState)
```

Call condense_equilib_particle() on each particle in the aerosol to ensure that every particle has its water content in equilibrium.

```
def pow2_above(arg0: int) -> int
```

Return the least power-of-2 that is at least equal to n.

```
def run_part(arg0: Scenario, arg1: EnvState, arg2: AeroData, arg3: AeroState,  
            arg4: GasData, arg5: GasState, arg6: RunPartOpt)
```

Do a particle-resolved Monte Carlo simulation.

Classes

```
class AeroData(...)
```

Aerosol material properties and associated data.

The data in this structure is constant, as it represents physical quantities that cannot

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

PyPartMC: demo

<https://github.com/open-atmos/PyPartMC>

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary



what PyPartMC achieves:



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
 - ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)
- ▶ leverages Python binary dissemination system for PartMC and dependencies (static linkage)



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)
- ▶ leverages Python binary dissemination system for PartMC and dependencies (static linkage)
- ▶ **encapsulates simulation setup/input within one single-language file (e.g., for paper review)**



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)
- ▶ leverages Python binary dissemination system for PartMC and dependencies (static linkage)
- ▶ encapsulates simulation setup/input within one single-language file (e.g., for paper review)
- ▶ allows to extend PartMC simulation logic with Python code (e.g., optics with PyMieScatt)



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)
- ▶ leverages Python binary dissemination system for PartMC and dependencies (static linkage)
- ▶ encapsulates simulation setup/input within one single-language file (e.g., for paper review)
- ▶ allows to extend PartMC simulation logic with Python code (e.g., optics with PyMieScatt)
- ▶ **streamlined workflows for generating simulation ensembles (no need for input text files!)**



what PyPartMC achieves:

- ▶ single-command (pip) install on Windows, Linux & macOS (source-only for ARM)
- ▶ access to unmodified PartMC internals from Python, Julia, Matlab... and C++
- ▶ potential for use of PartMC in test suites of other Python packages (already in PySDM)
- ▶ leverages Python binary dissemination system for PartMC and dependencies (static linkage)
- ▶ encapsulates simulation setup/input within one single-language file (e.g., for paper review)
- ▶ allows to extend PartMC simulation logic with Python code (e.g., optics with PyMieScatt)
- ▶ streamlined workflows for generating simulation ensembles (no need for input text files!)
- ▶ offering users (students) a single-language familiar environment (Colab, ARM JupyterHub)



take-home messages & fun facts:



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...
- ▶ static linkage: on the one hand essential (lack of standarised Fortran ABI);
on the other hand blocks Conda packaging (policy)



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...
- ▶ static linkage: on the one hand essential (lack of standardised Fortran ABI);
on the other hand blocks Conda packaging (policy)
- ▶ git[hub] submodules instrumental for handling 10+ Fortran, C and C++ dependencies



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...
- ▶ static linkage: on the one hand essential (lack of standardised Fortran ABI);
on the other hand blocks Conda packaging (policy)
- ▶ git[hub] submodules instrumental for handling 10+ Fortran, C and C++ dependencies
- ▶ no universal binaries for macOS yet ([gfortran help welcome!](#))



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...
- ▶ static linkage: on the one hand essential (lack of standardised Fortran ABI);
on the other hand blocks Conda packaging (policy)
- ▶ git[hub] submodules instrumental for handling 10+ Fortran, C and C++ dependencies
- ▶ no universal binaries for macOS yet (gfortran help welcome!)
- ▶ **kudos to Mathworks for github.com/matlab-actions**



take-home messages & fun facts:

- ▶ pybind11 as a viable tool for interfacing Fortran and Python
(especially given integration with CMake which handles Fortran well)
- ▶ Python's "glue language" role leveraged: Julia, Matlab, ...
- ▶ static linkage: on the one hand essential (lack of standardised Fortran ABI);
on the other hand blocks Conda packaging (policy)
- ▶ git[hub] submodules instrumental for handling 10+ Fortran, C and C++ dependencies
- ▶ no universal binaries for macOS yet (gfortran help welcome!)
- ▶ kudos to Mathworks for github.com/matlab-actions
- ▶ SoftwareX review: actually also concerned code/installation

acknowledgements



acknowledgements



Thank you for your attention!

pypi.org/p/PyPartMC

github.com/open-atmos/PyPartMC

doi:10.1016/j.softx.2023.101613