OpenMC's Nuclear Data Pipeline and HDF5 Format

Paul K. Romano
Computational Scientist, Argonne National Laboratory

IAEA Consultancy Meeting on Model Code Output & Application Nuclear Data
March 16, 2021

OpenMC

- Community-developed, open source Monte Carlo particle transport code
  - **Code:** https://github.com/openmc-dev/openmc
  - **Docs:** https://docs.openmc.org
  - **Forum:** https://openmc.discourse.group
- Neutrons, photons, thick-target bremsstrahlung
- Constructive solid geometry or CAD-based geometry
- Development team places particular emphasis on ease of use, scalability, and community building
- Input scripts (Python API) generate XML files, HDF5 output files

Motivation for a new format

Until 2016, OpenMC relied directly on ACE format data. Decided to move away from ACE for a variety of reasons:

- Archaic format, binaries not cross-platform
- Not easily extensible
- Not designed for multi-temperature problems

Moving to an **HDF5**-based format helps solve many of these problems:

- Binary files containing filesystem-like hierarchy
- HDF5 library enables applications to read/write data portably
- Bindings for many languages (C, C++, Fortran, Java, Python)

To support the new format, a data interface was developed within OpenMC's Python API.

OpenMC: Python API

- In addition to the underlying transport solver, OpenMC features a rich, extensible Python API
- Tightly couples input generation, simulation execution, and postprocessing/data analysis
- Leverages existing Python scientific computing ecosystem, namely NumPy, SciPy, h5py, and Pandas

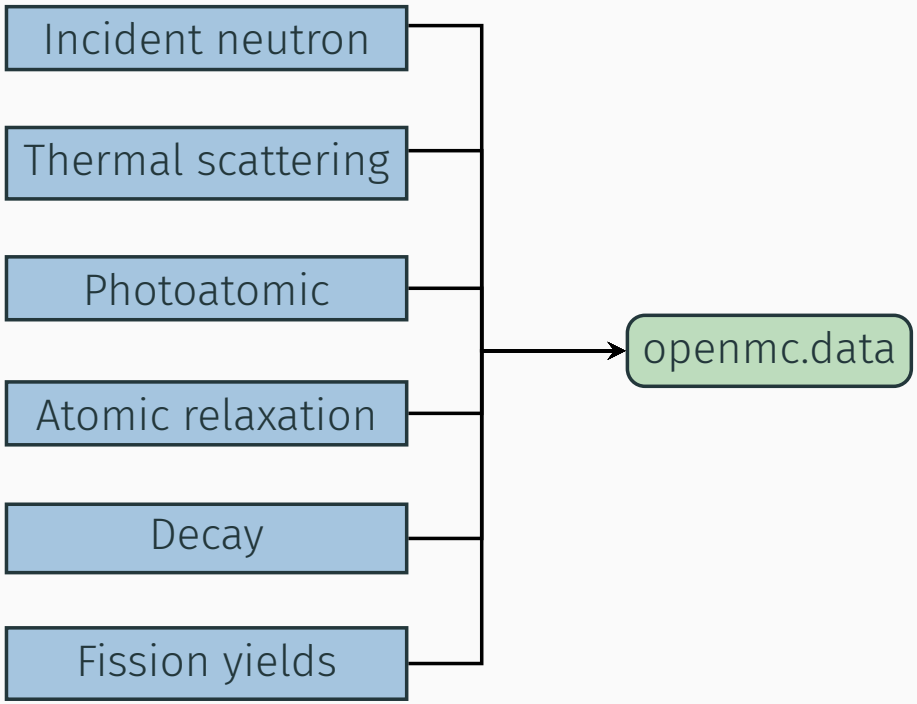
Python API Components

- Input generation classes (produce XML files)
- Output processing and analysis
- Multigroup cross section generation (`openmc.mgxs`)
- Depletion and transmutation solver (`openmc.deplete`)
- Bindings to C/C++ interface (`openmc.lib`)
- Nuclear data interface (`openmc.data`)

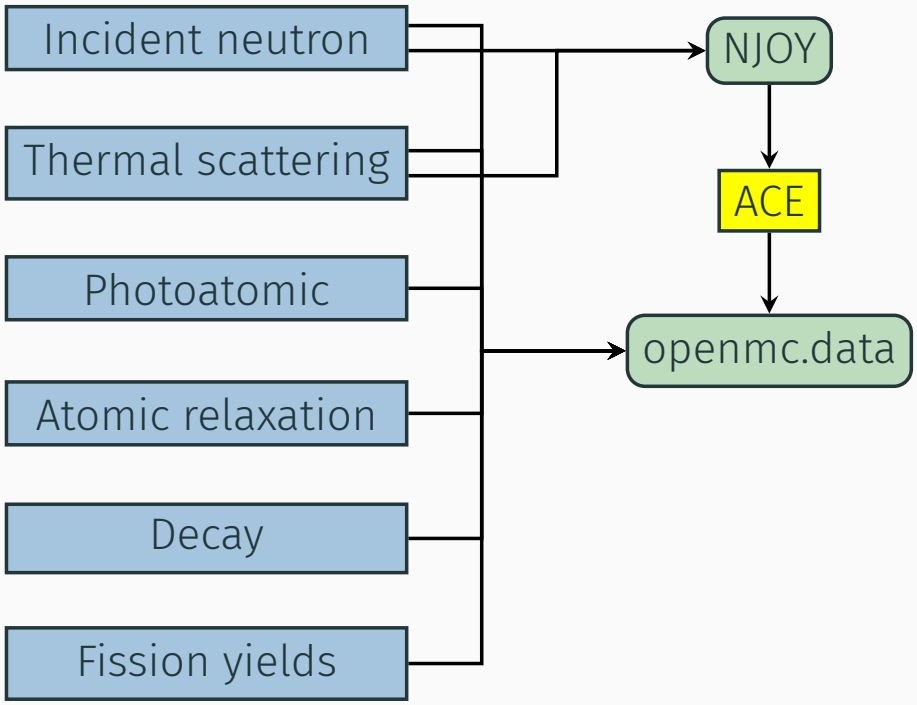
Python API example (Jezebel)

```
from openmc import *

# Define material
plutonium = Material()
plutonium.add_nuclide('Pu239', 3.7047e-2)
plutonium.add_nuclide('Pu240', 1.7512e-3)
plutonium.add_nuclide('Pu241', 1.1674e-4)
plutonium.add_element('Ga', 1.3752e-3)
Materials([plutonium]).export_to_xml()

# Create a sphere of plutonium
surf = Sphere(r=6.3849, boundary_type='vacuum')
main_cell = Cell(fill=plutonium, region=-surf)
Geometry([main_cell]).export_to_xml()

# Define settings for the simulation
settings = Settings()
settings.particles = 10000
settings.batches = 1000
settings.inactive = 50
center = (0., 0., 0.)
settings.source = Source(space=stats.Point(center))
settings.export_to_xml()

# And run!
run()
```

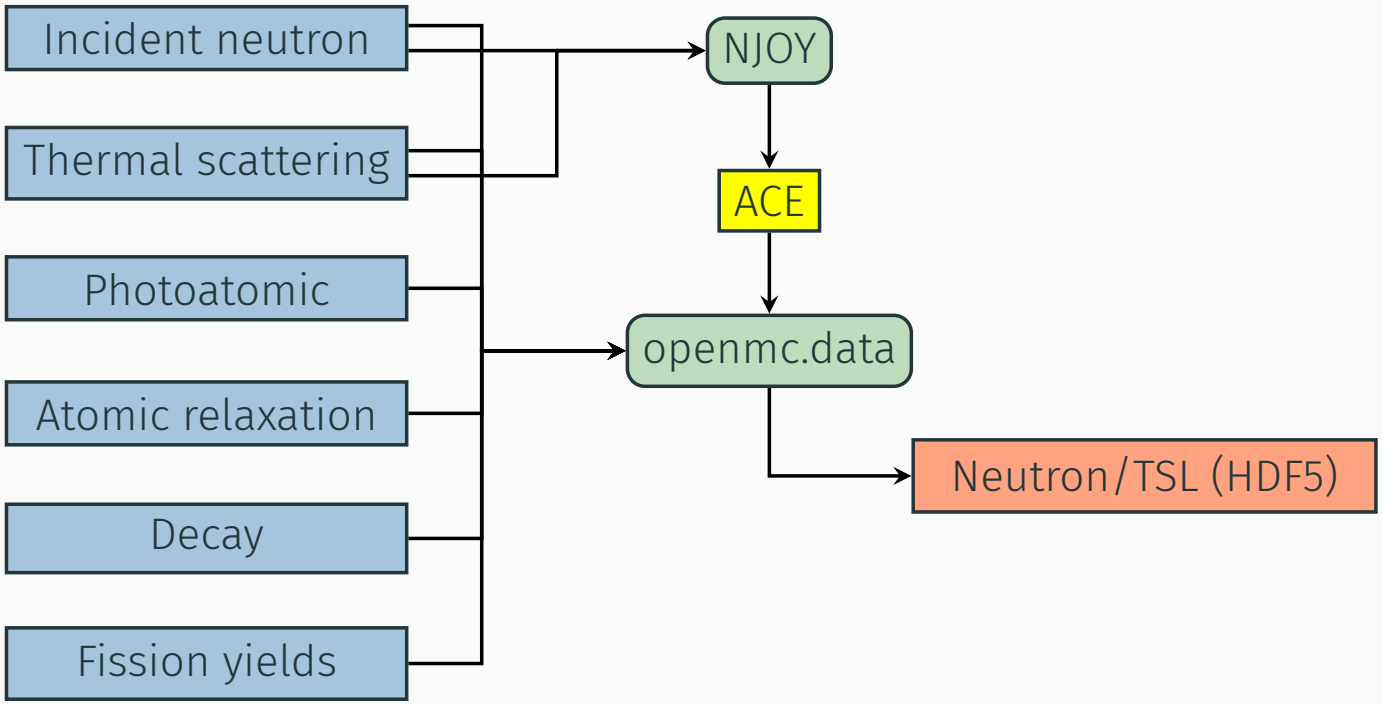
OpenMC Data Pipeline



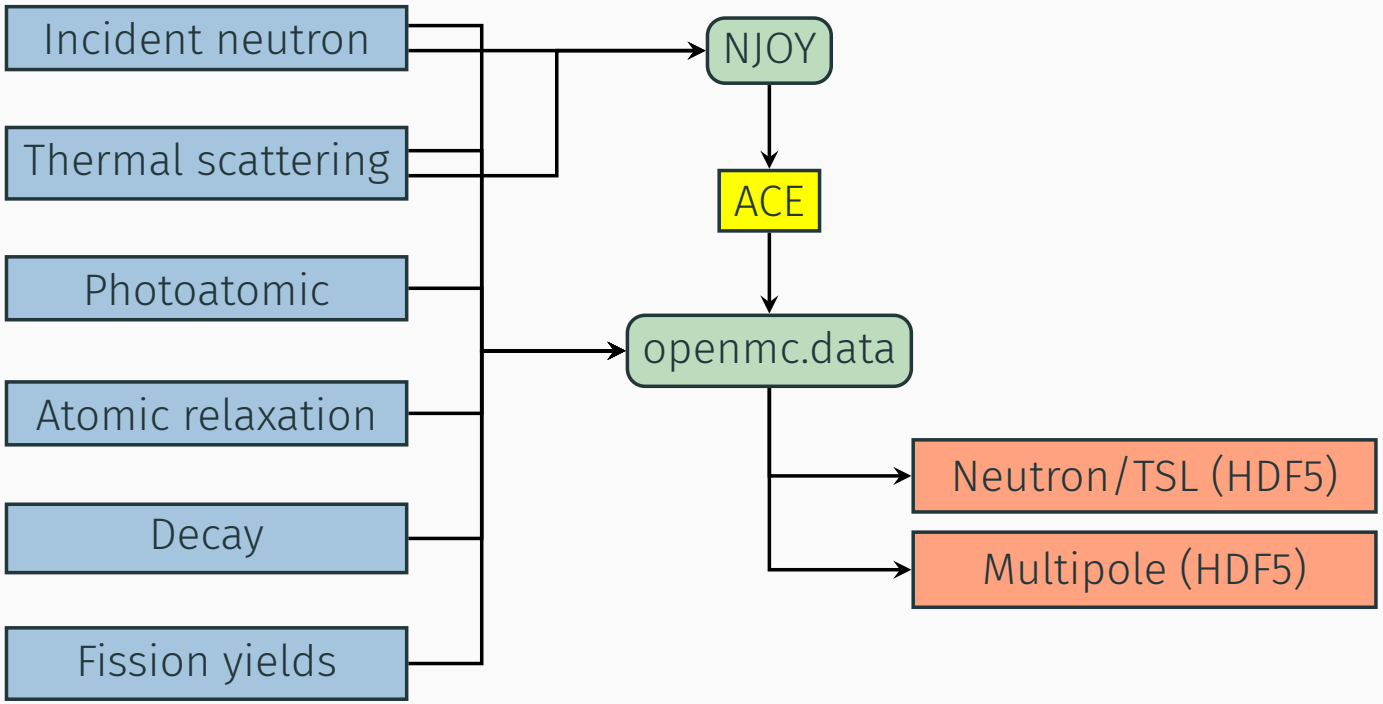
OpenMC Data Pipeline



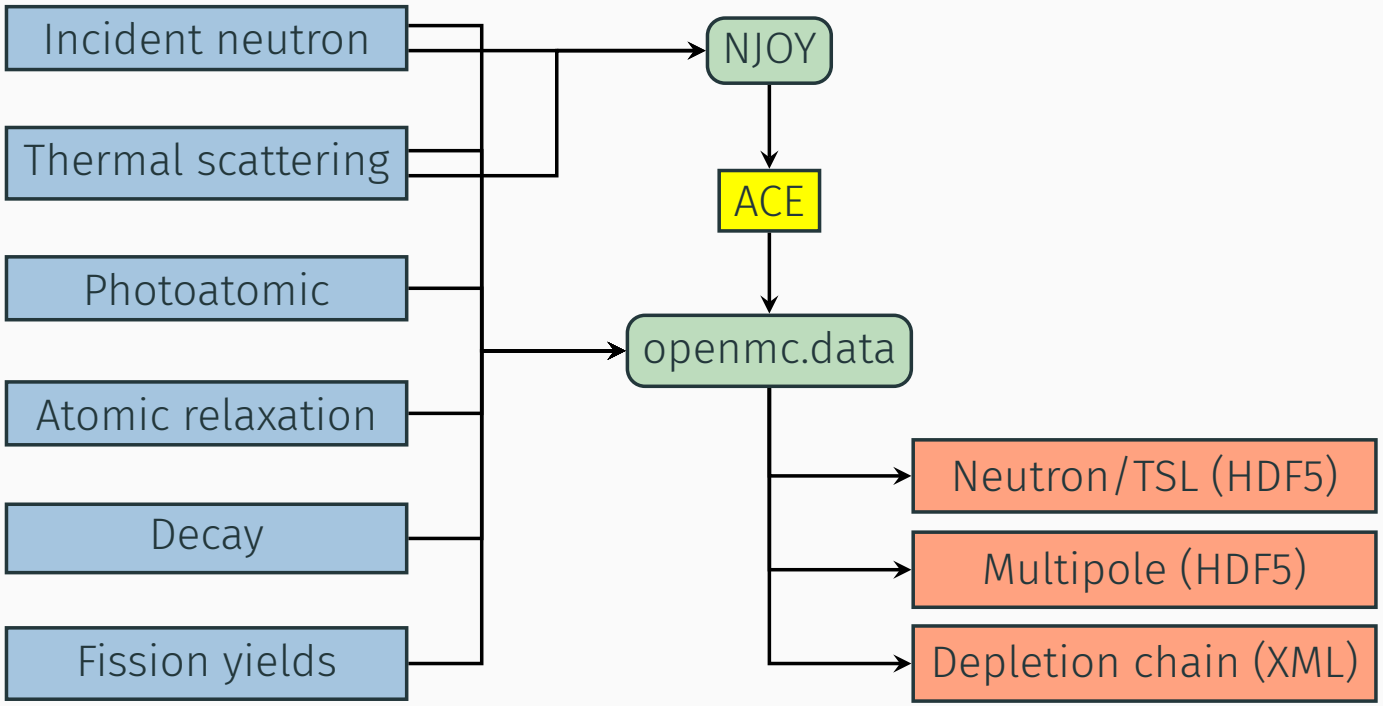
OpenMC Data Pipeline



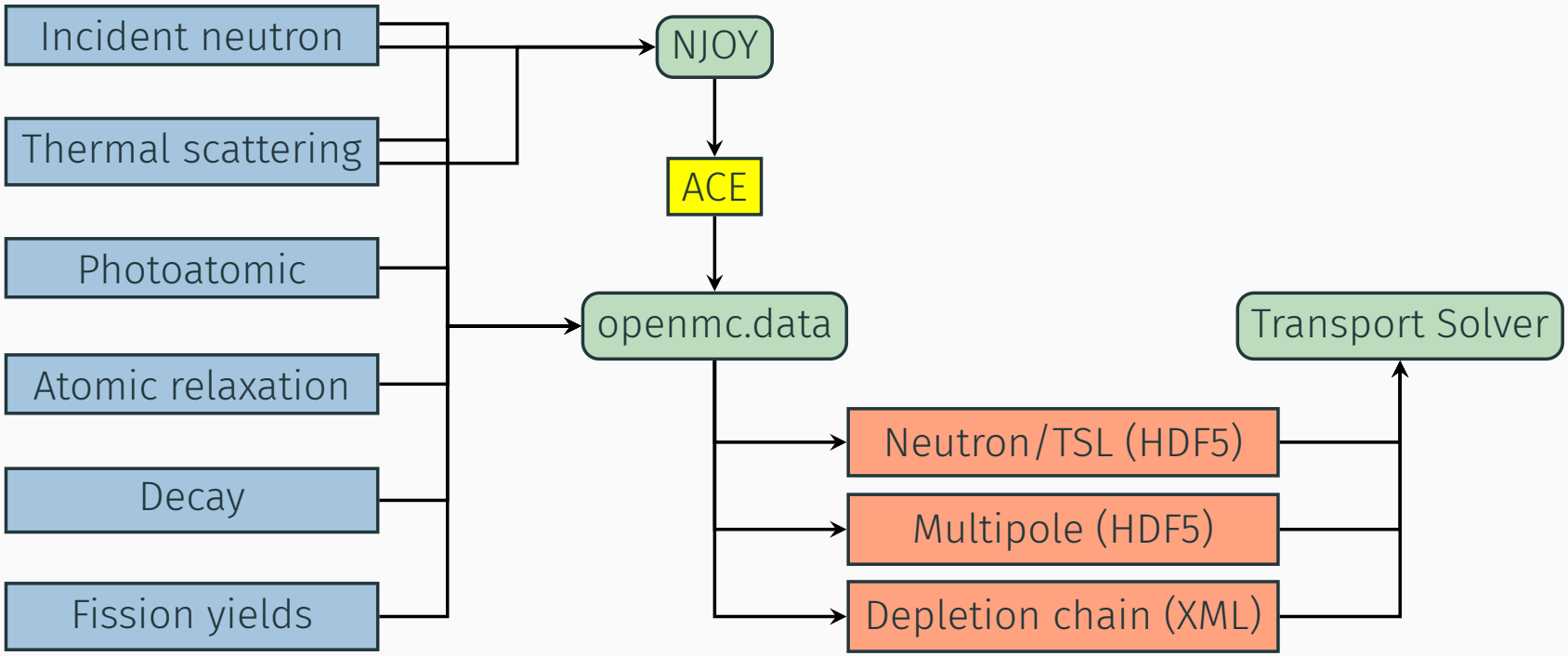
OpenMC Data Pipeline



OpenMC Data Pipeline



OpenMC Data Pipeline



Data sources

Most ENDF sublibraries have a corresponding class in the data API:

ENDF Sublibrary	Python class
Incident neutron	`openmc.data.IncidentNeutron`
Thermal scattering	`openmc.data.ThermalScattering`
Photoatomic	`openmc.data.IncidentPhoton`
Atomic relaxation	`openmc.data.IncidentPhoton`
Decay	`openmc.data.Decay`
Fission product yields	`openmc.data.FissionProductYields`
Photonuclear	—
Electroatomic	—
Incident charged particle	—

Incident neutron data

Incident neutron class can be created a few ways:

- ENDF data: `IncidentNeutron.from_endf(...)`
- ACE data: `IncidentNeutron.from_ace(...)`
- HDF5 data: `IncidentNeutron.from_hdf5(...)`
- ENDF data (+NJOY): `IncidentNeutron.from_njoy(...)`

Live Demonstration

HDF5 File Format

Composed of groups (“directories”) and datasets (“files”):

```
/Pu239/
/Pu239/energy/0K
/Pu239/energy/294K
/Pu239/energy/...
/Pu239/reactions/
/Pu239/reactions/reaction_002/
/Pu239/reactions/reaction_002/0K/xs
/Pu239/reactions/reaction_002/294K/xs
/Pu239/reactions/reaction_002/product_0/yield
/Pu239/reactions/reaction_002/distribution_0/angle/...
...
/Pu239/urr/
/Pu239/urr/0K/
/Pu239/urr/0K/energy
/Pu239/urr/0K/table
```

Full specification

Adding data beyond ACE

With HDF5, easy to extend format to include data not in ACE:

- Fission energy release
- Decay data
- Windowed multipole (for on-the-fly Doppler broadening)

Python API makes the process seamless

```
u235 = IncidentNeutron.from_ace('U235.ace')
fer = FissionEnergyRelease.from_endf('U235.endf')
u235.fission_energy = fer
```

Temperature dependence

Classes differentiate between temperature-dependent...

- Cross sections
- Unresolved resonance probability tables
- $S(\alpha, \beta)$ tables

...and temperature-independent data:

- Reaction product yields
- Reaction product energy-angle distributions

Can results in large savings in memory and disk space, e.g.:

- JEFF 3.2 ACE files: 38 GB
- JEFF 3.2 HDF5 files: 5 GB

Other resouces/infrastructure

Scripts for generating full libraries:

- https://github.com/openmc-dev/data

Pregenerated HDF5 libraries:

- Various libraries: https://openmc.org
- TENDL: https://tendl.web.psi.ch/tendl_2019/tar.html

Just-in-time library generation:

- https://github.com/openmc-data-storage/openmc_data_downloader
- Jon Shimwell (UKAEA) talk tomorrow at 14:30!

Shared Infrastructure

- As a community, we should be thinking about ways to share the burden of development, testing, etc.
  - Common set of tools, e.g., ENDF parsing/manipulation
  - Best utilize limited human capital

- Good ideal, but hard to achieve in practice...
  - Dependency aversion
  - Resources/funding
  - Redundancy can be beneficial

How about GNDS?

- We don't currently have support in any form for GNDS
- Two paths forward:
  1. Move from HDF5 format to using GNDS directly, add support in transport solver (through MCGIDI?)
  2. Add `IncidentNeutron.from_gnds(...)` method and preserve use of HDF5 format

Thank you!