ITALIAN NATIONAL AGENCY FOR
NEW TECHNOLOGIES, ENERGY AND
SUSTAINABLE ECONOMIC DEVELOPMENT

Particle and radiation transport simulations: tackling user needs via software architecture layer abstractions

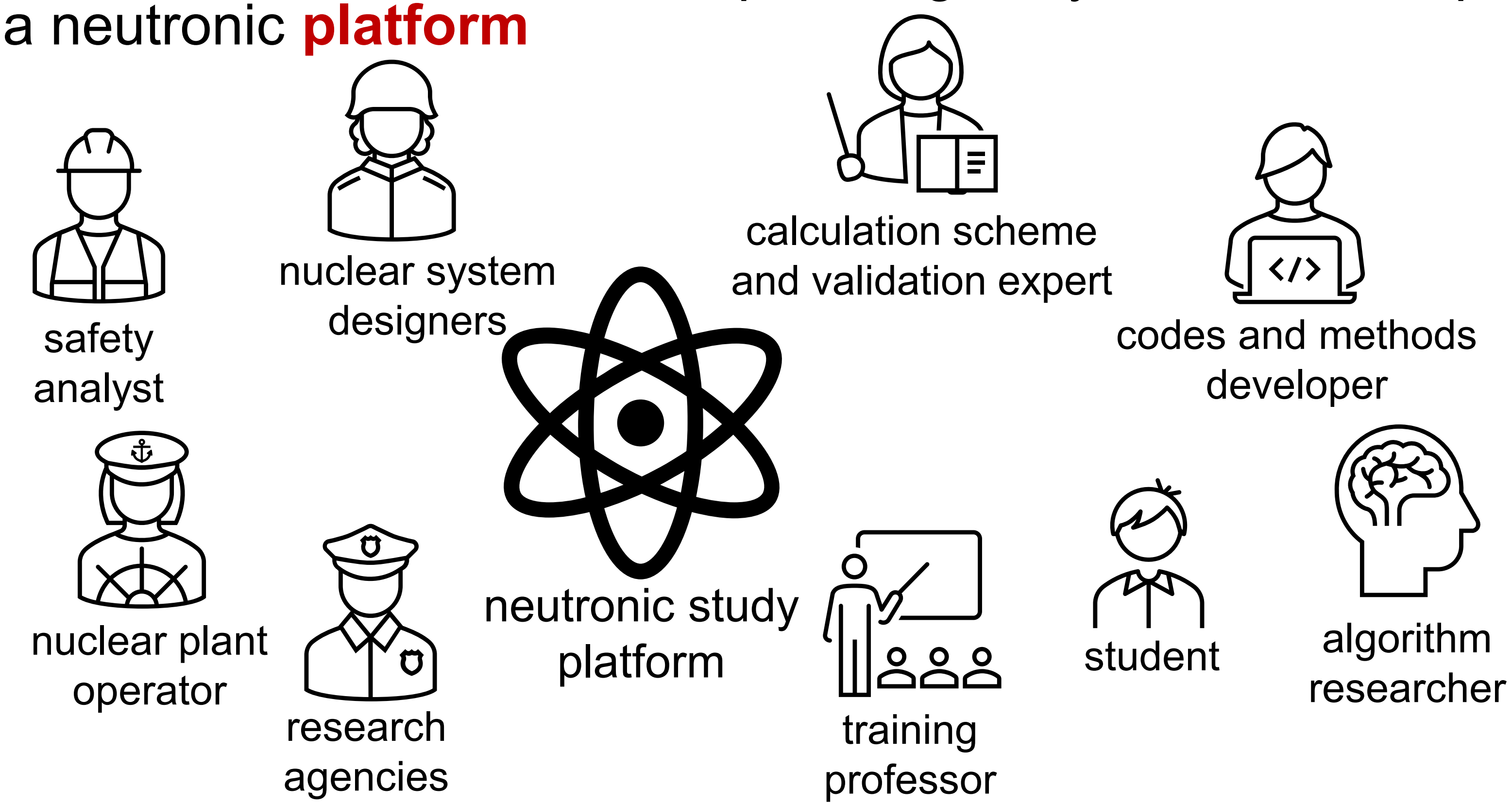
Shielding aspects of Accelerators, Targets and Irradiation Facility 16

INFN LNF - 28th May 2024

Alberto Previti : `alberto.previti@enea.it`

Actors in particle and radiation transport

- There are a multitude of neutronic **codes** either deterministic or stochastic
- … but industrial and research needs require to go beyond the concept of neutronic code to provide a neutronic **platform**



➢**This presentation aims at highlighting the effect of software architecture choices and their impact on the user needs**

Bricks of a nuclear code

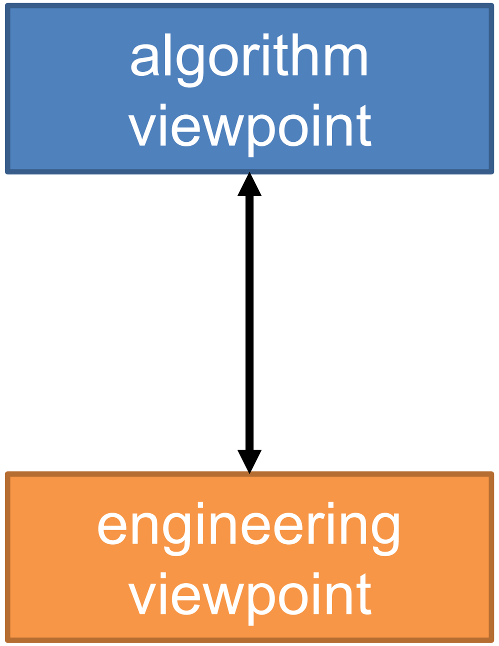
- **Geometry**
  - volumes of the system
  - regions on which the results are collected
- **Materials**
  - isotopic vector
  - nuclear data
- **Source**
  - intensity
  - location
  - type
- **Solver**
  - numerical methods to the get solution
  - projection/restriction to compute results
- **I/O**
  - get input data
  - provide results

➢ **A calculation platform employs one or more nuclear codes as components**

Backend vs Frontend

Distinct software components involved in calculation platforms:

- **backend**
  - calculation kernel
  - manages solvers and numerical methods
  - provides solution methodologies for physical phenomena
  - manages multiple geometries (e.g. computation and homogenization)
  - allows implementing calculation protocols
- **frontend**
  - modeling/engineering layer providing building blocks for each application
  - interpreter of user needs and translation to actual calculations
  - provides pre/post processing tools
  - allows interactively drive back-end based on
    - user calculation requests
    - outcome of calculations during processing



➢**Building a calculation platform requires multiple competencies**

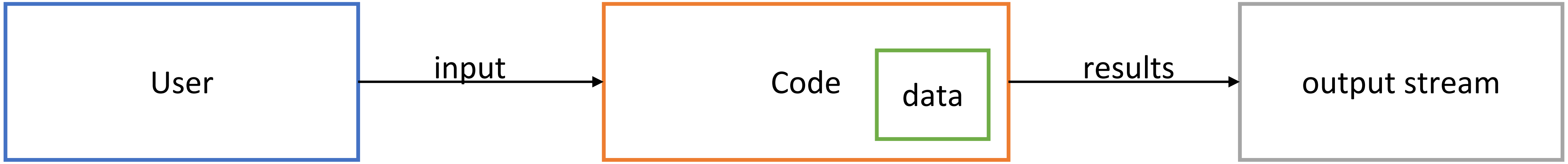
Monolithic vs Library

- Scientific code:
  - **monolithic** → code process internally an input deck
  - **library platform** → basically as an ensemble of pure functions working on objects

- Need to build complex yet powerful calculations schemes
  - **separate responsibilities**
    - low level numeric/mathematic functionalities (e.g. C++/Fortran)
    - high level engineering/physics functionalities (e.g. Python)
  - **programmable** platform
    - ease to build new schemes and studies
    - ease to post-processing
    - multi-physics possible

Objects property and life cycle (1)

- **Static/monolithic approach**



  - data masked to the user → no introspection
  - internal state modified → no multiple executions possible in the same thread
  - fixed output stream → interactive execution not easily achievable

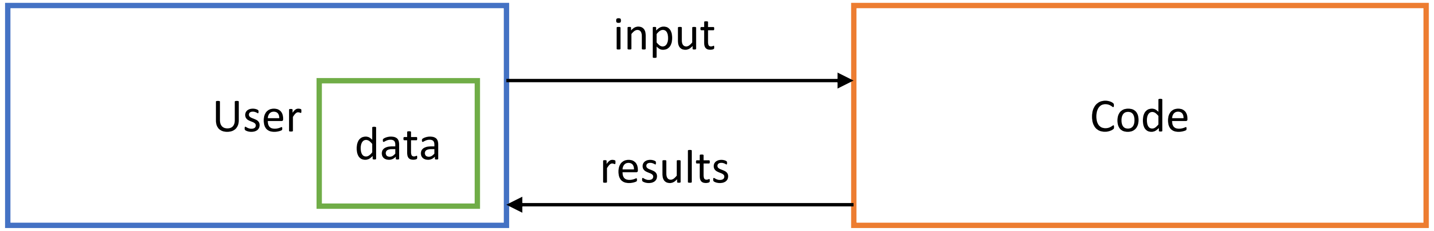
➢ difficult to build a platform

- Example: flux calculation
  - User defines a flux problem→ code stores the modeling and data structures internally
  - User launches a flux calculation → code perform the calculation and show outputs in listing

  ➢ If no convergence, the User needs to manually analyze the listing, stop the execution, and relaunch everything from scratch after modifying the input deck → **manual processing**

Objects property and life cycle (2)

- **Library/pure approach**



  - data owned by the user → introspection possible
  - internal state not present → easily thread safe
  - output stream decided by the user → interactive choices possible

➢ programmable calculations possible

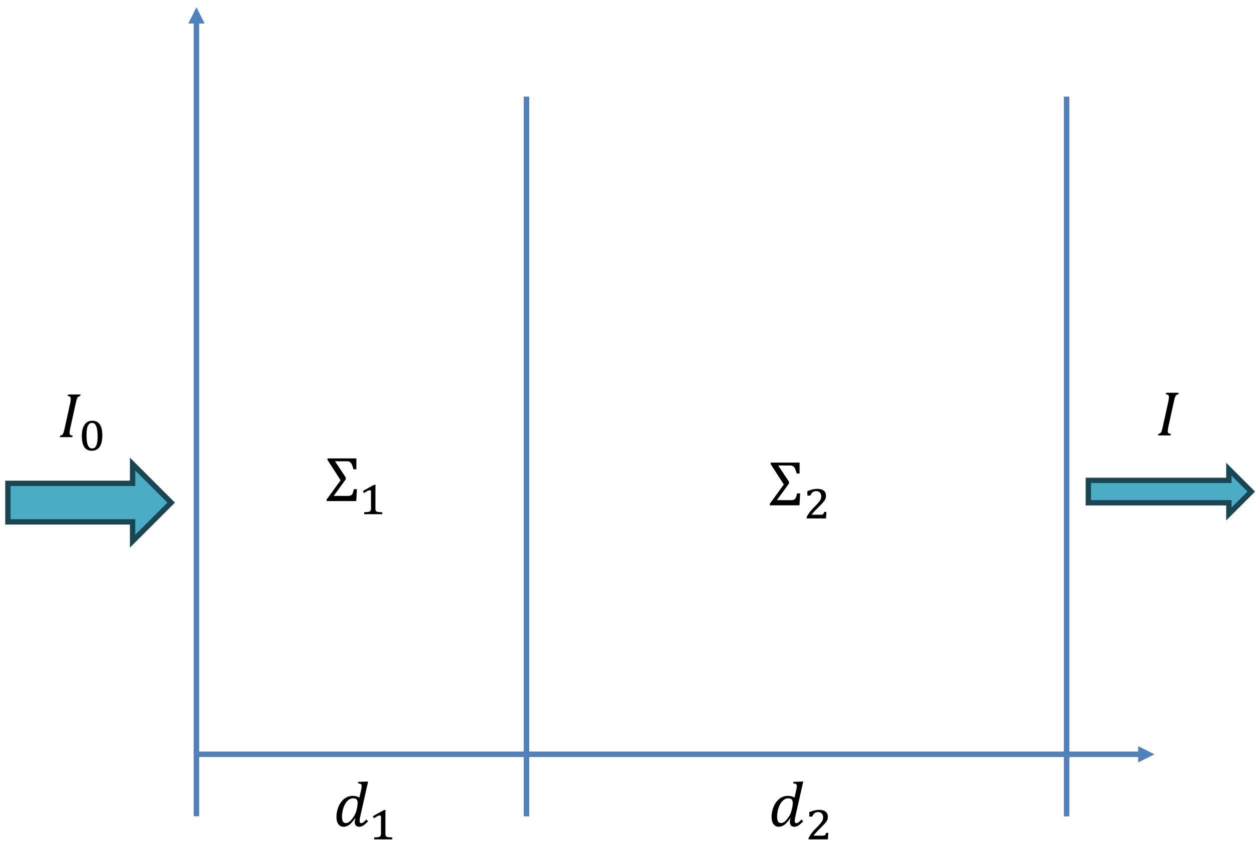
- Example: flux calculation
  - User defines a flux problem→ code returns the modeling and data objects
  - User launches a flux calculation → code manages input objects, performs the calculation and returns an outcome objects

  ➢ If no convergence, the User may analyze the results and perform adjustments dynamically in a **programmable** way → **automatic processing** possible in addition to manual.

A test case: attenuation problem

Attenuation problem in 2 slabs

- geometry: d_1, d_2
- materials: Σ_1, Σ_2
- source: I_0
- result: I
- solver: $I = I_0 e^{-(\Sigma_1 d_1 + \Sigma_2 d_2)}$
  - semi-analytic solution is possible in this case



Sample card-based input deck

```
3.1e-3
6.82e-2
2.1
1.2
57.33
```

materials

geometry

source

Different implementations

1. "Classic" Fortran implementation `[fortran2]`
2. Direct translation in C++ `[cpp2b]`
3. Python bindings to provide nicer user interfaces
4. Issues of the internal states and of the I/O stream
5. Refactoring into a pure C++ library `[cpp3]`
6. Python bindings of the code to construct a platform
7. Composition of elements via Python objects
8. Backward compatibility of the card-based input deck

Test cases available at https://github.com/alberto743/satif16

Concluding remarks

- Algorithm developers should consider and incorporate the user needs in building a calculation platform
- Separation of responsibilities between frontend and backend allows presenting the engineering viewpoint, while maintaining the algorithm and modeling complexity to a dedicated layer
- Designing clever object lifecycle is of paramount importance to obtain programmable platforms
- Allocated data structures should belong to the user to allow introspection and reusability and to avoid race conditions and side effects
- Building a transport solver pure (re-entrant) library may require deep refactoring efforts

Thank you for your attention