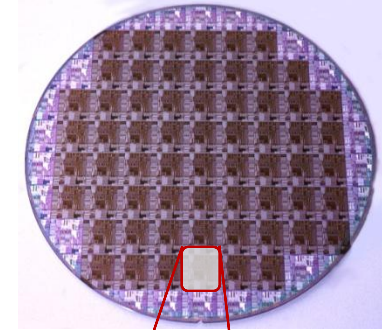
A Python software framework for the design of photonic integrated circuits

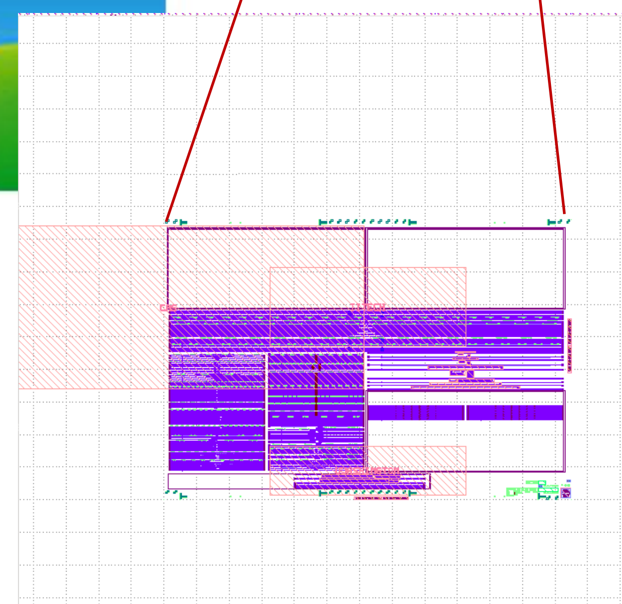
Emmanuel Lambert – Martin Fiers

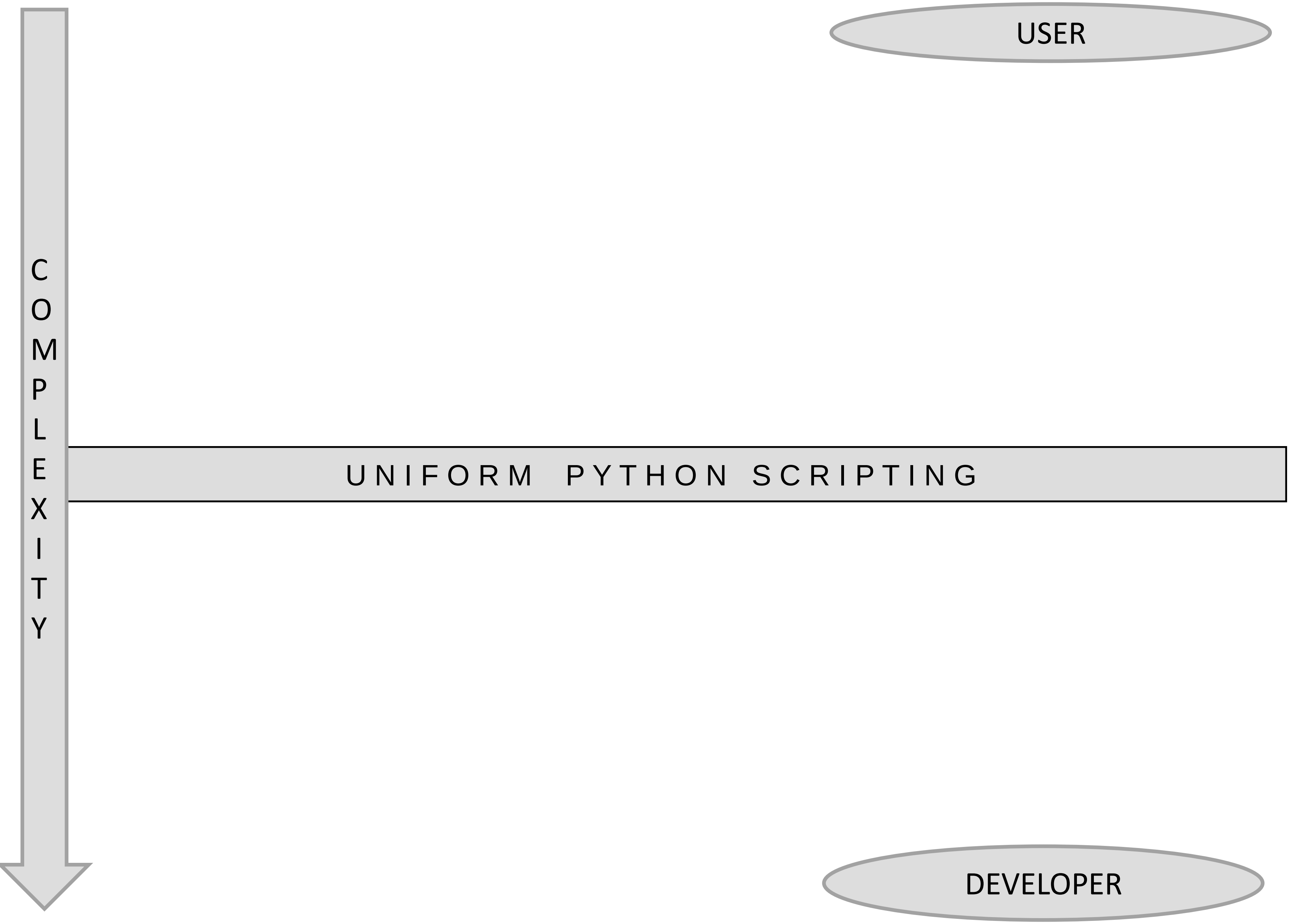
Department of Information Technology
Photonics Research Group
Sint-Pietersnieuwstraat 41
9000 Ghent
Belgium
http://photonics.intec.ugent.be

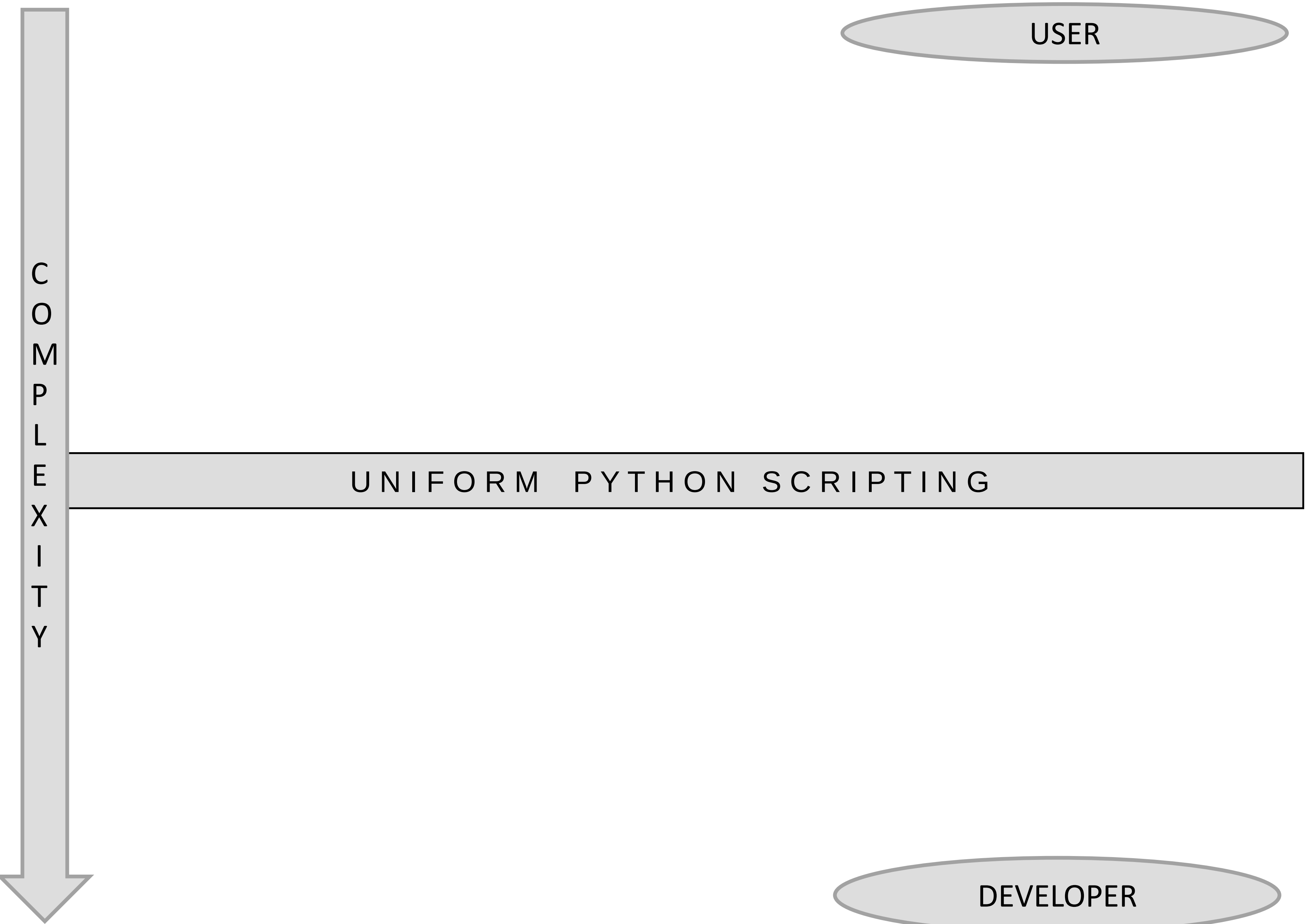
Photonics Research

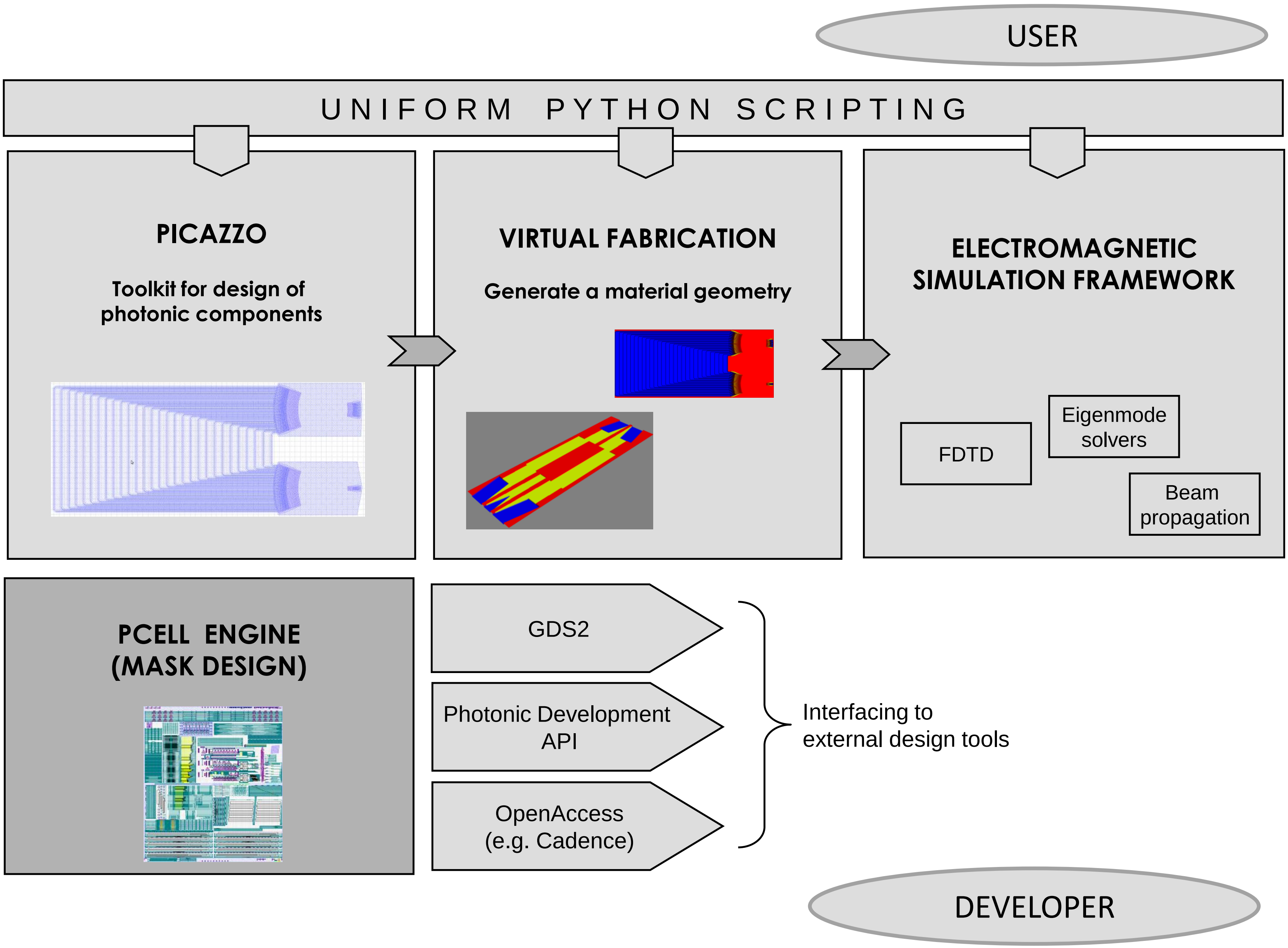
Setting the scene…






USER
COMPLEXITY
UNIFORM PYTHON SCRIPTING
DEVELOPER


USER
COMPLEXITY
UNIFORM PYTHON SCRIPTING
DEVELOPER

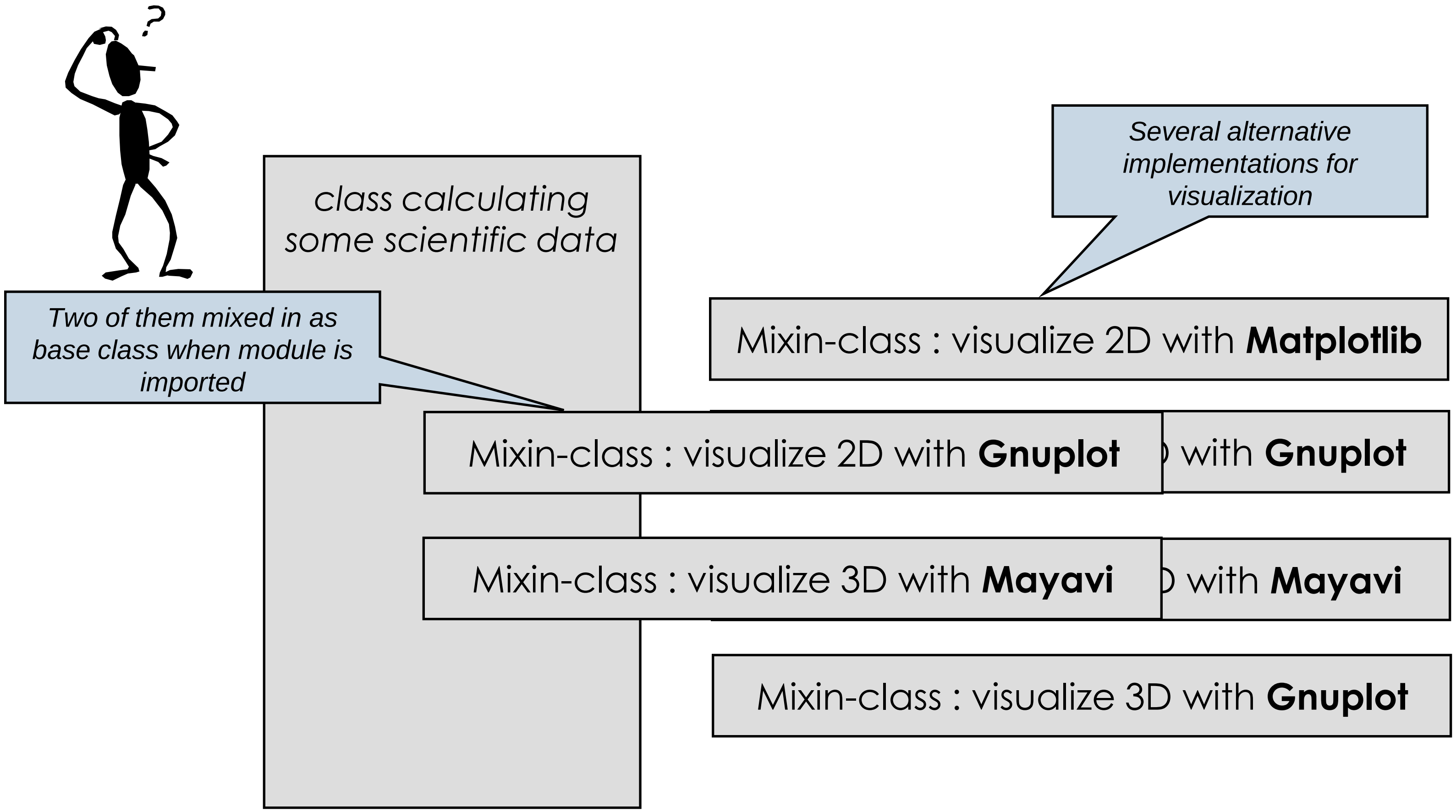

USER
UNIFORM PYTHON SCRIPTING
PICAZZO
Toolkit for design of photonic components
VIRTUAL FABRICATION
Generate a material geometry
ELECTROMAGNETIC SIMULATION FRAMEWORK
FDTD
Eigenmode solvers
Beam propagation
PCELL ENGINE (MASK DESIGN)
GDS2
Photonic Development API
OpenAccess (e.g. Cadence)
Interfacing to external design tools
DEVELOPER

- **Architecture**
- **Technical implementation**
  - Virtual fabrication
  - Interface with FDTD simulator

Main architectural concept :

separation of concerns through **Mixins**

What is a mixin ? Let’s illustrate it ...



How we do it in Python...

```
class __Visualize__(object) :
    def plot_2d(self):
        raise NotImplementedError("Abstract class")

    def plot_3d(self):
        raise NotImplementedError("Abstract class")
```

```
class Visualize2DGnuplot(object) :
        def plot_2d(self):
            ...
```

```
class Visualize2DMatplotlib(object) :
        def plot_2d(self):
            ...
```

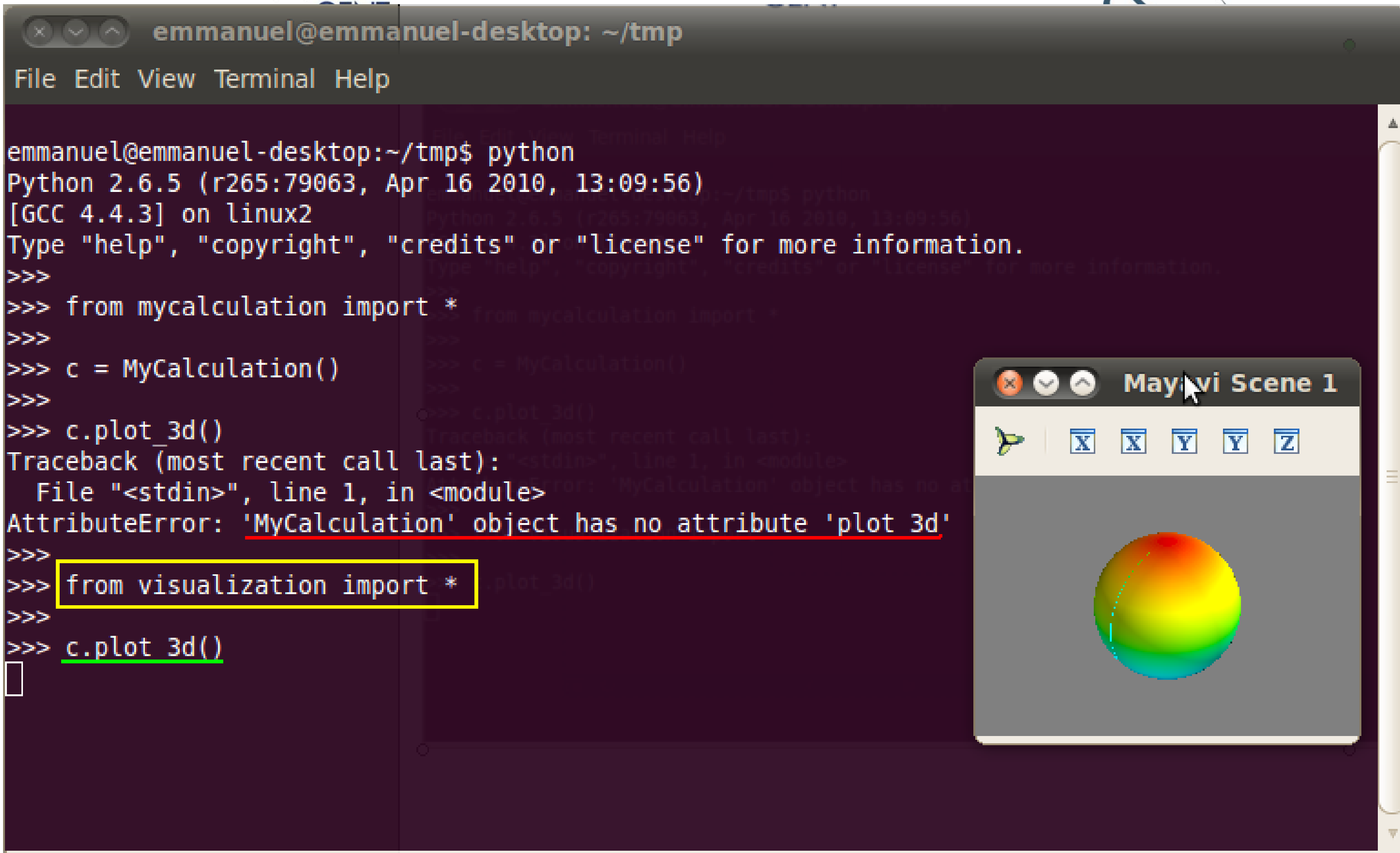
```
class Visualize3DGnuplot(object) :
        def plot_3d(self):
            ...
```

```
class Visualize3DMayavi(object) :
        def plot_2d(self):
            ...
```

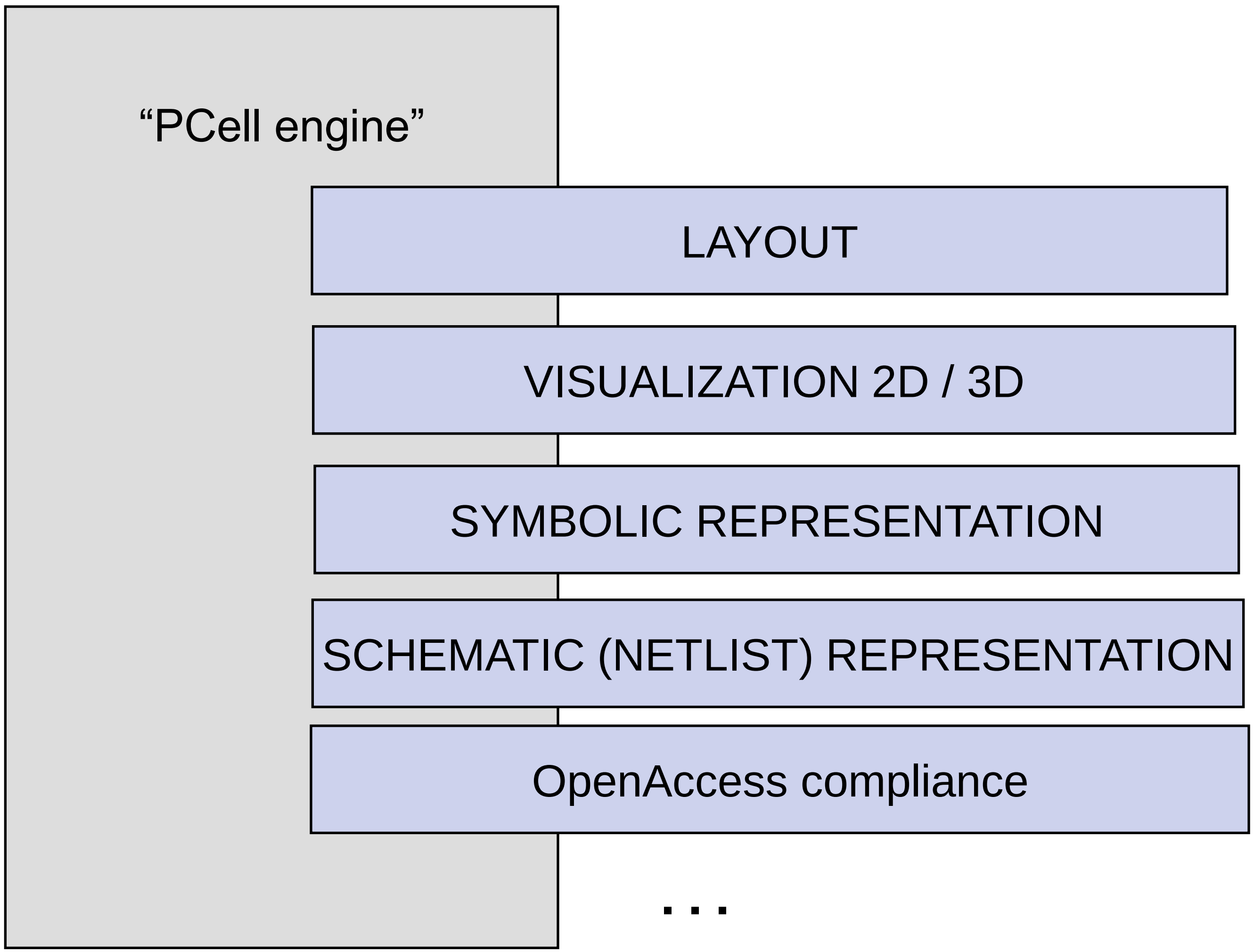
```
class MyCalculation(MixinBowl):
    #core functionality only


    def calculate(self):
        ....


    def get_data(self):

        ...

        return (X,Y,Z)
```

in the __init__.py file of the package :

```
MyCalculation.mixin(Visualize2DGnuplot)
MyCalculation.mixin(Visualize3DMayavi)
```


emmanuel@emmanuel-desktop: ~/tmp
File Edit View Terminal Help
emmanuel@emmanuel-desktop:~/tmp$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from mycalculation import *
>>>
>>> c = MyCalculation()
>>>
>>> c.plot_3d()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'MyCalculation' object has no attribute 'plot_3d'
>>>
>>> from visualization import *
>>>
>>> c.plot_3d()
Mayavi Scene 1
X
X
Y
Y
Z

Applied to our framework ...

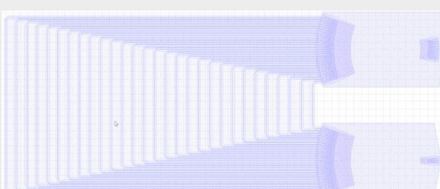


Implementation of the virtual fabrication

UNIFORM PYTHON SCRIPTING

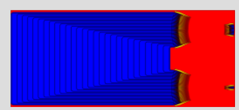
PICAZZO

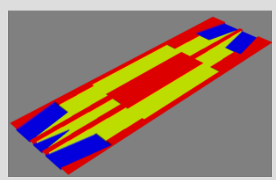
Toolkit for design of photonic components



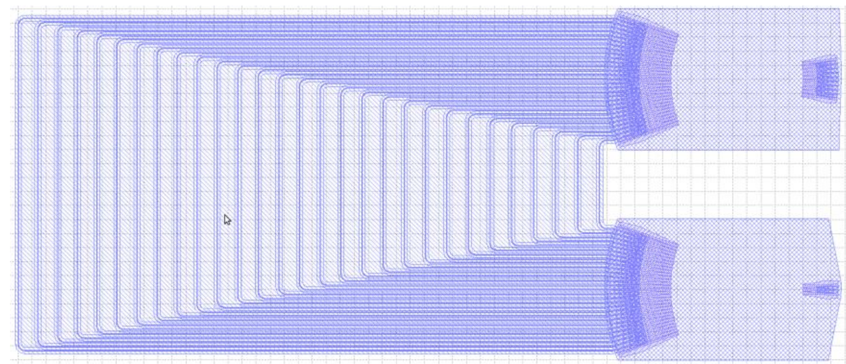
VIRTUAL FABRICATION

Generate a material geometry



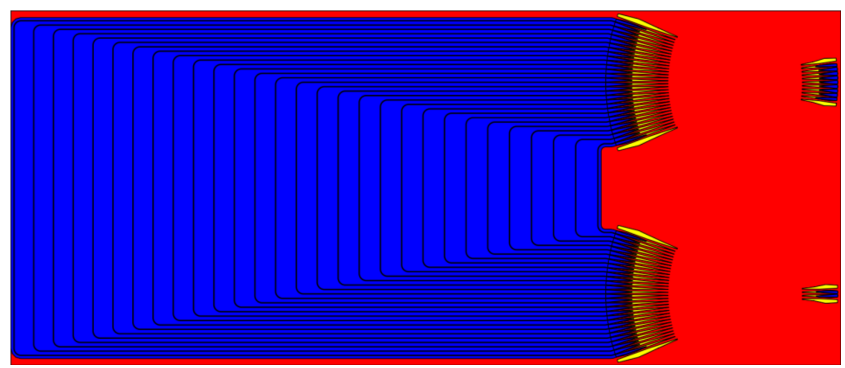


Mask layout =
a collection of shapes
on different layers



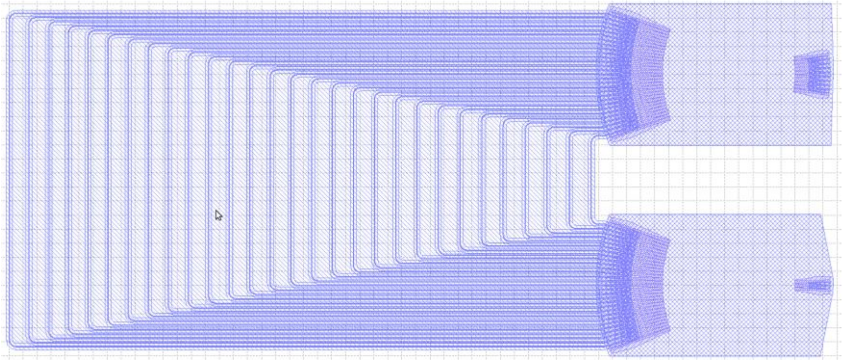
Virtual fabrication =

Can be expressed as
an algorithm with logical operations on subsets of the shapes
(AND, OR, XOR, NOT)



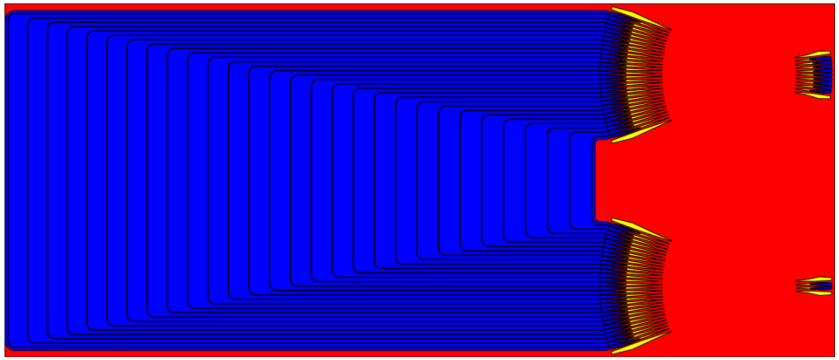
<u>Challenge</u> :

transform a mask layout



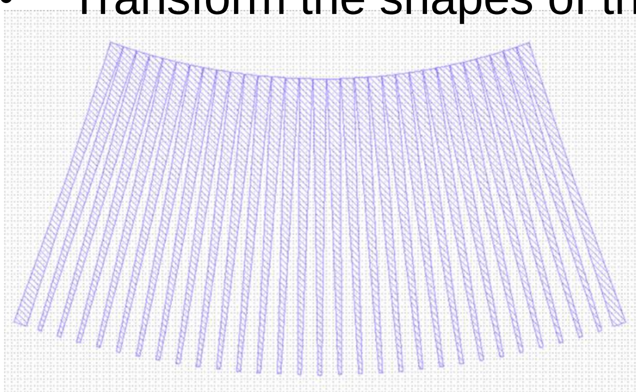
into

a materials geometry



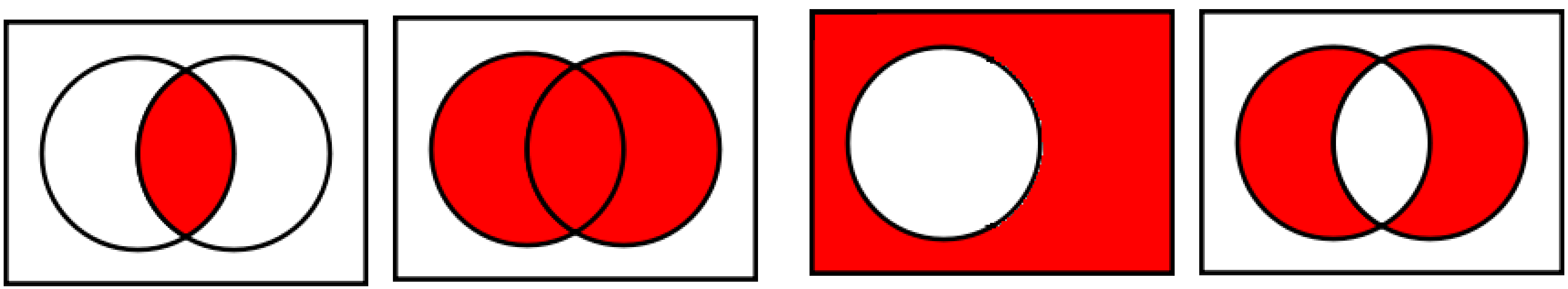
simulating the physical fabrication processes

- Geometrical Python packages **Shapely** and **Descartes**
- Transform the shapes of the mask layout into **Shapely polygons** (per layer)



```
shapely.geometry.polygon.Polygon

shapely.geometry.multipolygon.MultiPolygon
```

- Apply the **logical operations at polygon level** through Shapely functions :



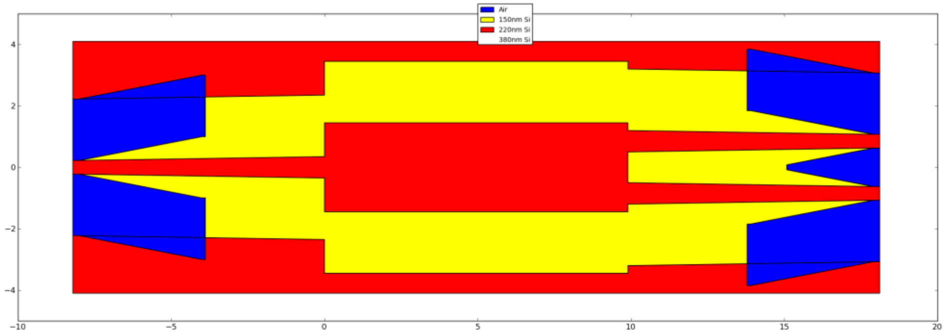
AND	OR	NOT	XOR
`Shapely :` **`intersection`**	`Shapely:` **`union`**	`Shapely:` **`difference`** `with the canvas`	`Shapely :` **`symmetric_difference`**

Advantages of implementation with Shapely vs more brute-force approaches:

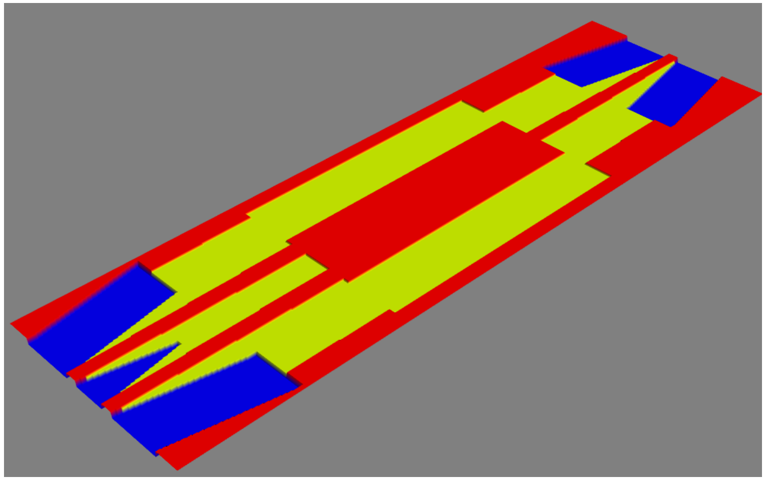
- **High accuracy** :
  - “analytical” information about the geometry is maintained throughout the algorithm
  - Allows interfacing with other tools (such as simulators) without loss of detail

- **Great performance** :
  - Very fast
  - Consumes very little memory

- **Descartes essential for 2D visualization with Matplotlib** (direct plotting of Shapely polygons)



- **3D visualisation with Mayavi surface plot (to be improved)**

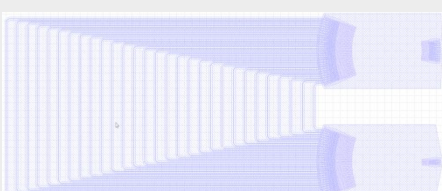


Interfacing to the Meep FDTD simulator

UNIFORM PYTHON SCRIPTING

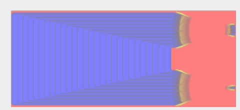
PICAZZO

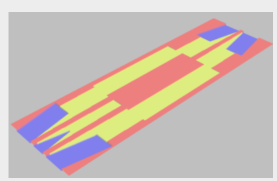
Toolkit for design of photonic components



VIRTUAL FABRICATION

Generate a material geometry





ELECTROMAGNETIC SIMULATION FRAMEWORK

FDTD

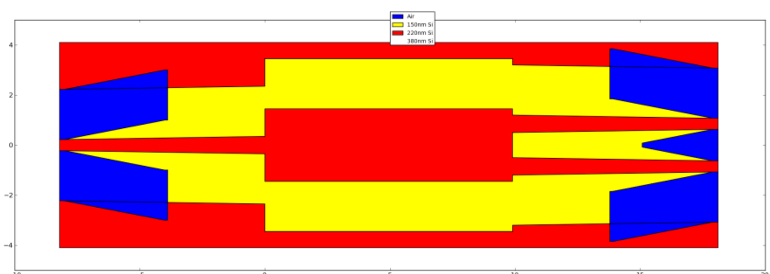
Eigenmode solvers

Beam Propagation

- Meep is a popular open source EM FDTD simulator developed by **MIT**
- It allows scripting through **Scheme** and **C++**

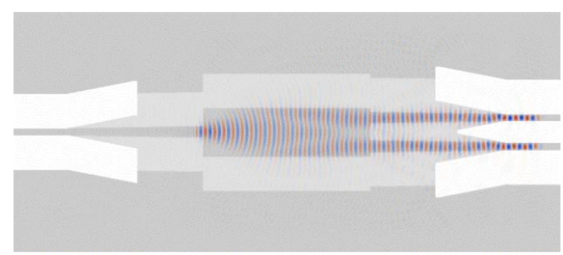
<u>Challenge</u> : seamless integration

Material geometry from virtual fabrication



Python

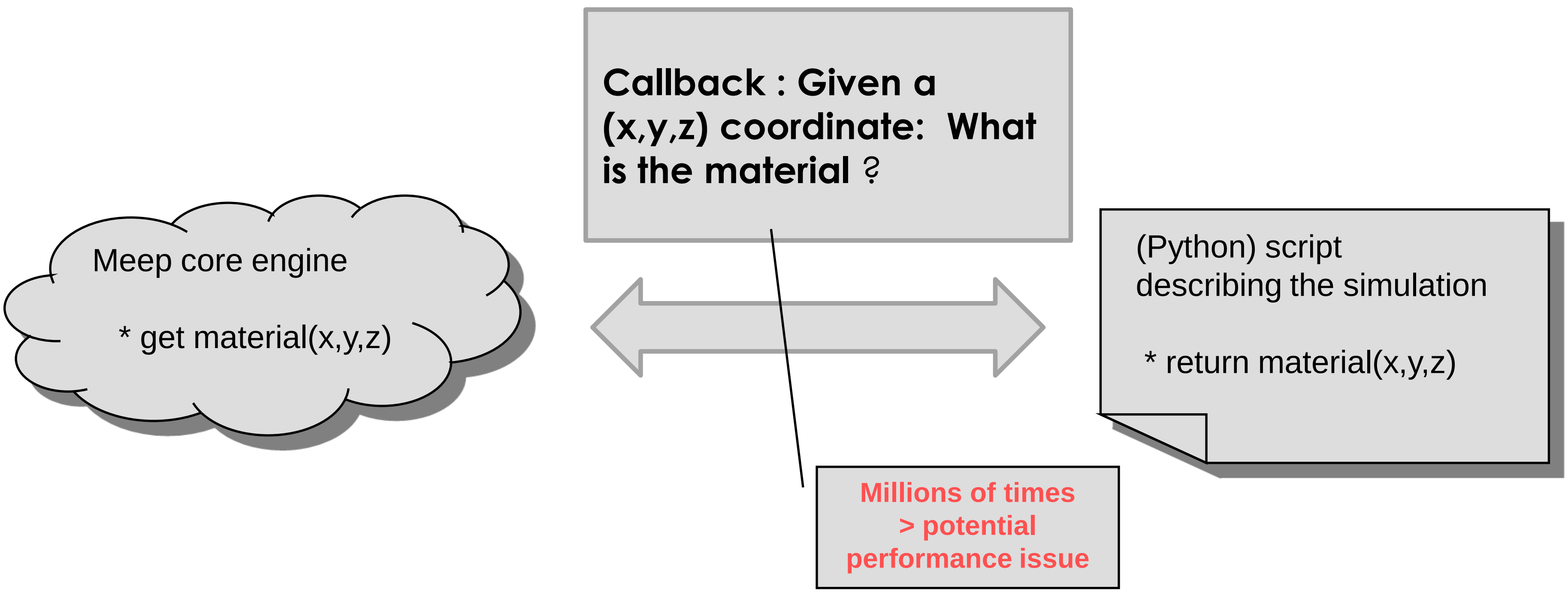
FDTD simulation



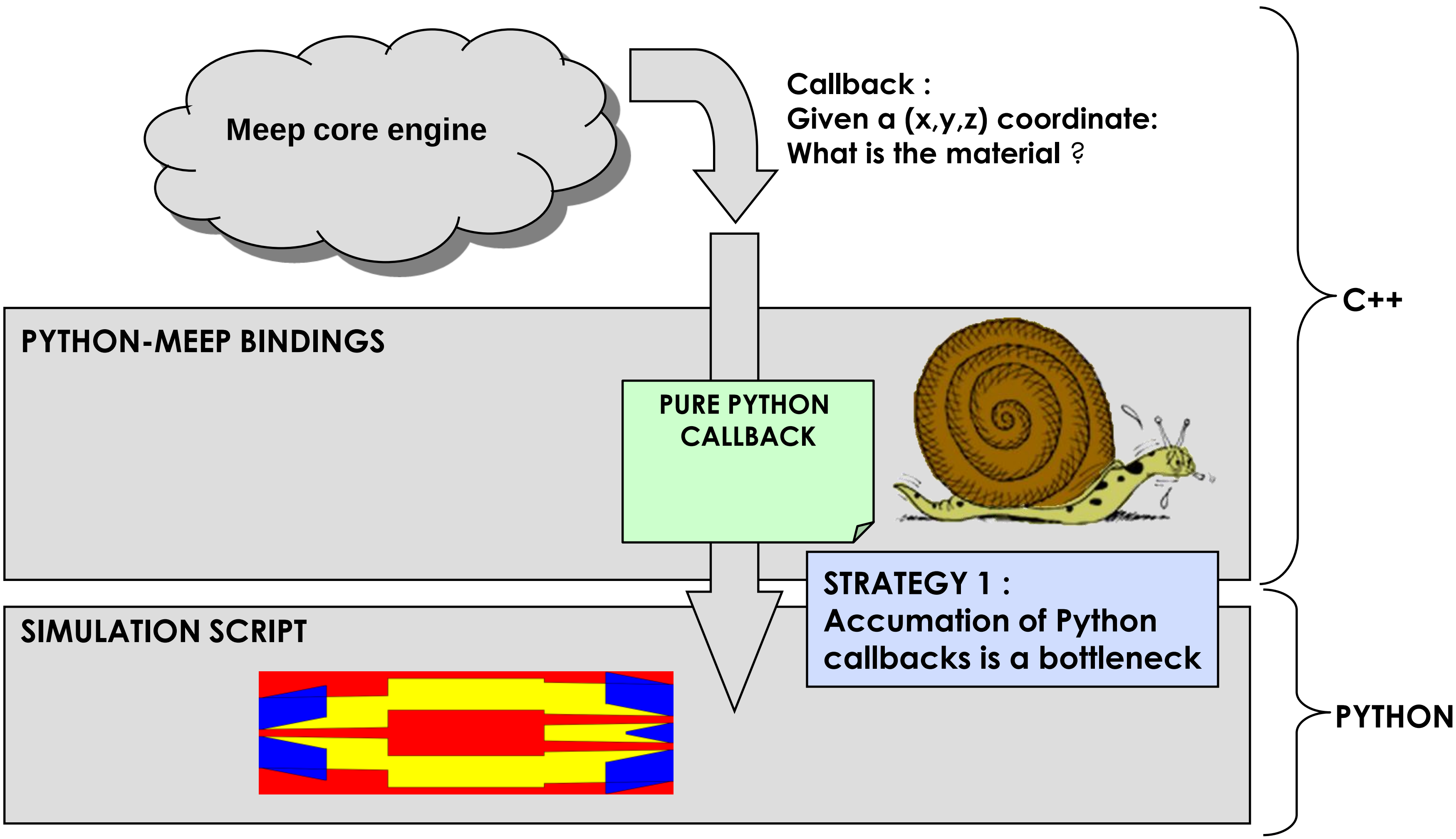
Scheme / C++

Step towards this goal :

- **Python bindings for Meep**, based on SWIG
- Released as open source on Launchpad in December 2009 ("Python-Meep")



Approaches for interfacing the material data with the Meep callback:



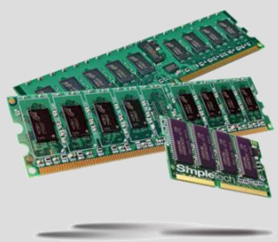
Approaches for interfacing the material data with the Meep callback:

Meep core engine

Callback :
Given a (x,y,z) coordinate:
What is the material ?

C++

PYTHON-MEEP BINDINGS



NUMPY MATRIX
with a discretised representation of the materials

SIMULATION SCRIPT

STRATEGY 2 :

- **Inaccurate (discretization)**
- **(very) high memory usage**

PYTHON

Approaches for interfacing the material data with the Meep callback:

Meep core engine

Callback :
Given a (x,y,z) coordinate:
What is the material ?

C++

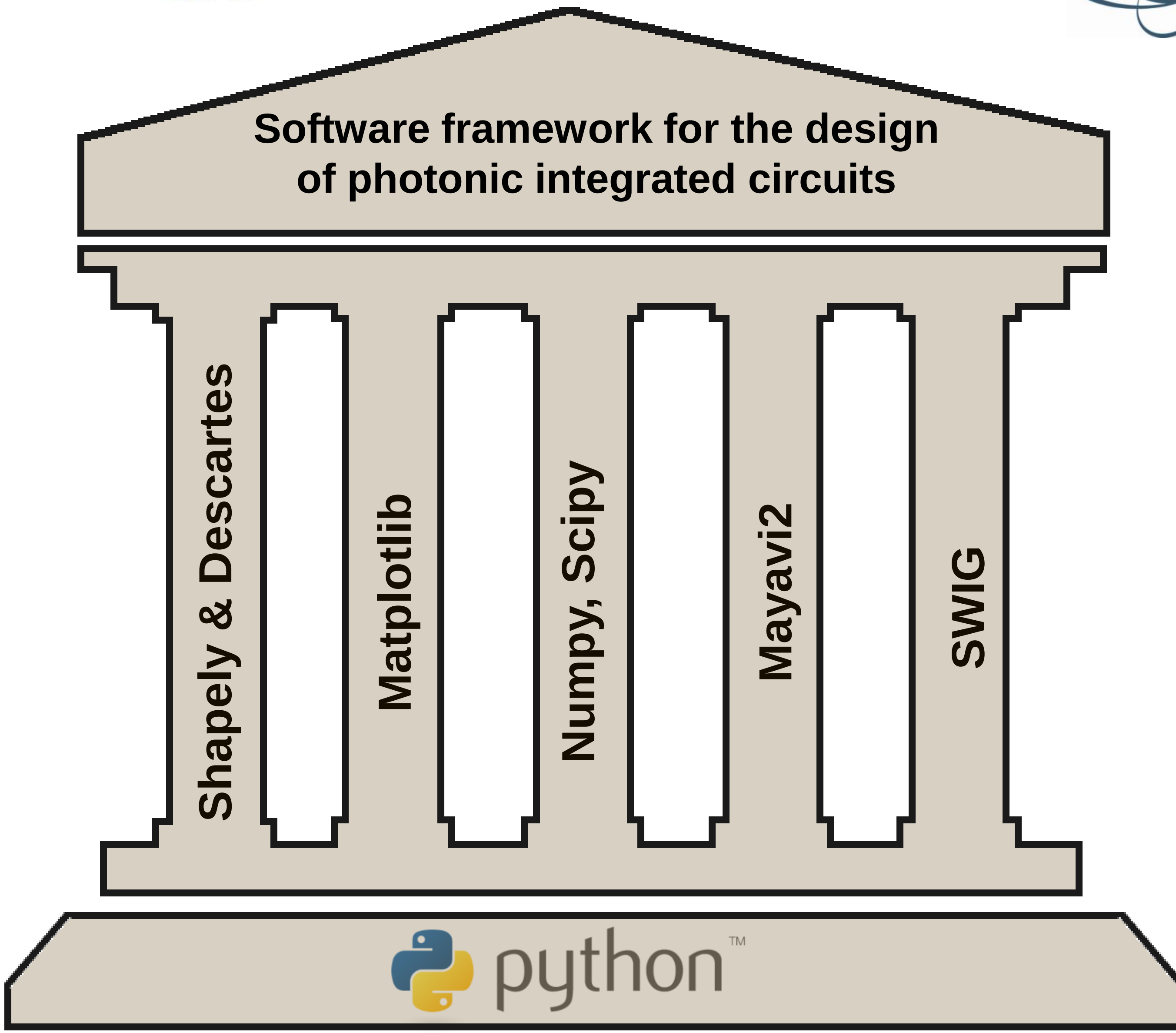
PYTHON-MEEP BINDINGS

POLYGON REPRESENTATION of the materials.
Given a point (x,y,z), which polygon is the point in ?
• winding number OR
• ray-casting algorithm

SIMULATION SCRIPT

STRATEGY 3 :
- **very accurate**
- **low memory requirements**

PYTHON


Software framework for the design
of photonic integrated circuits
Shapely & Descartes
Matplotlib
Numpy, Scipy
Mayavi2
SWIG
python™