UNIVERSITY OF Southampton

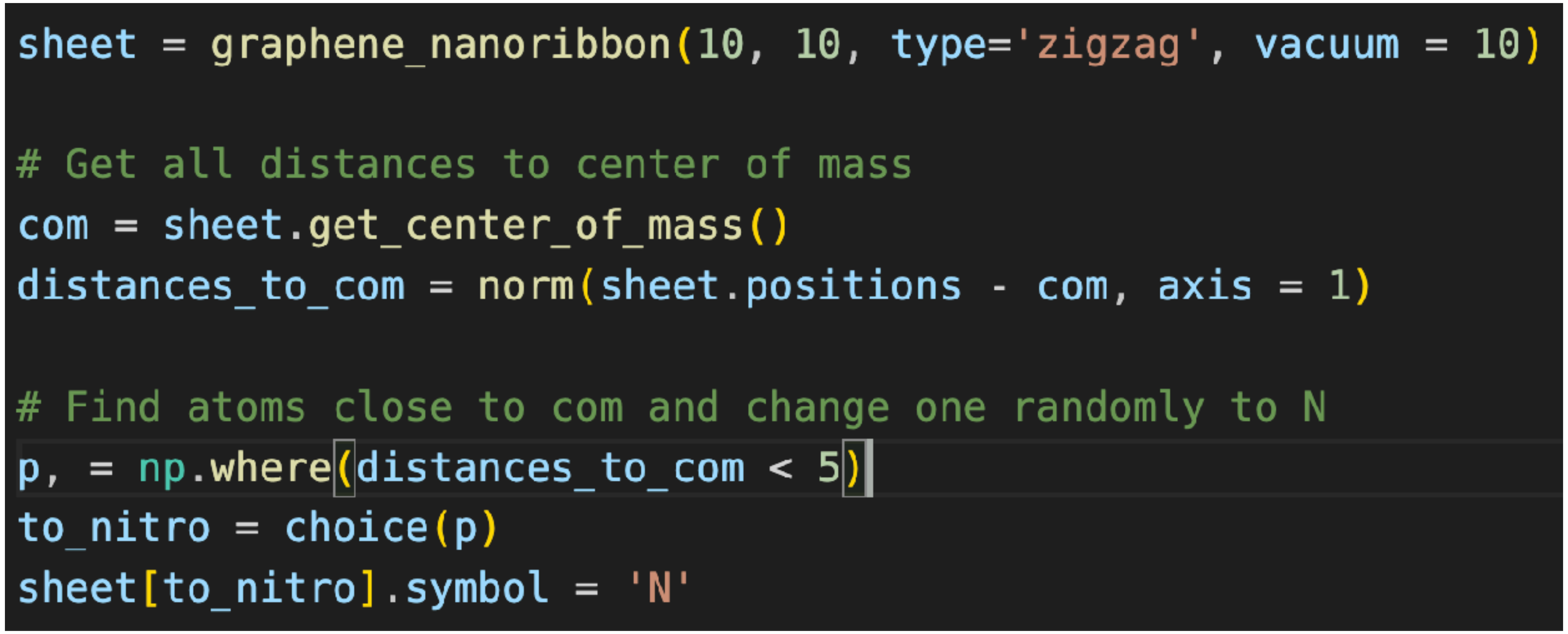
Python for computational chemistry

General capabilities, with a ONETEP example

Tom Demeyere

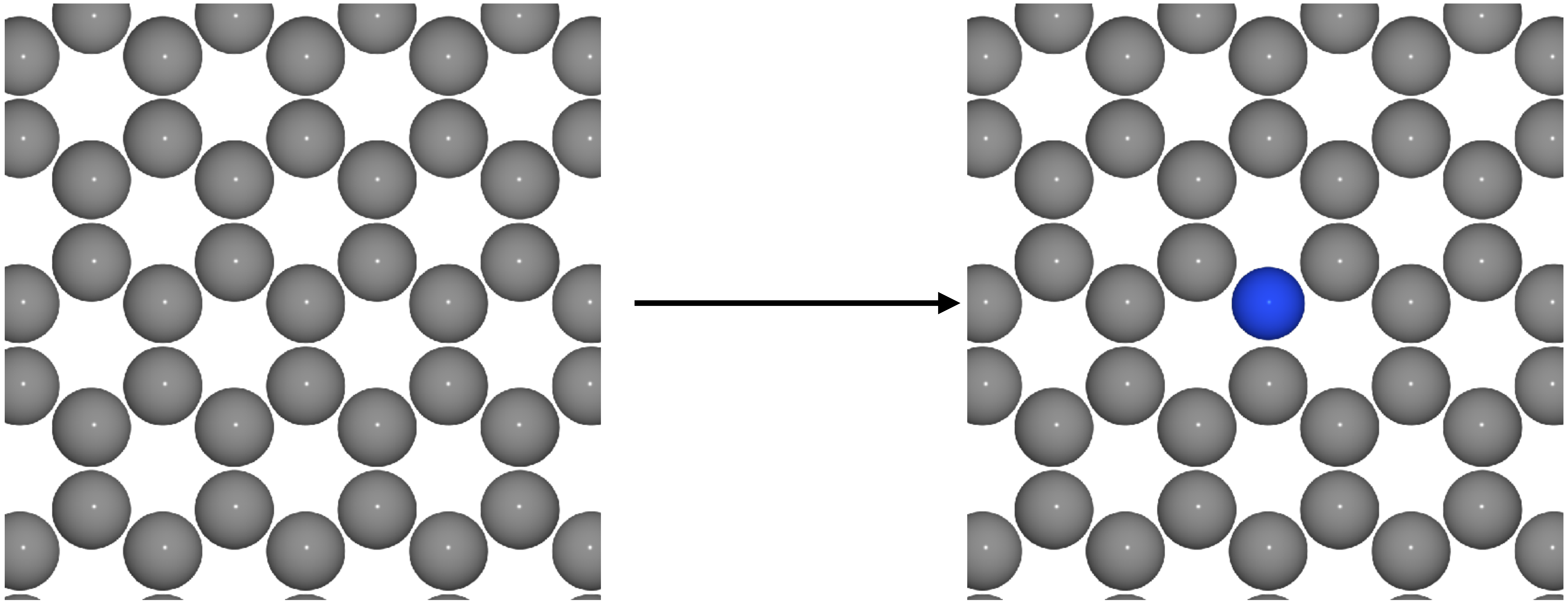
Example: doped graphene nanoribbon

Let’s create a simple graphene nanoribbon sheet with
10 Å of vacuum on each side

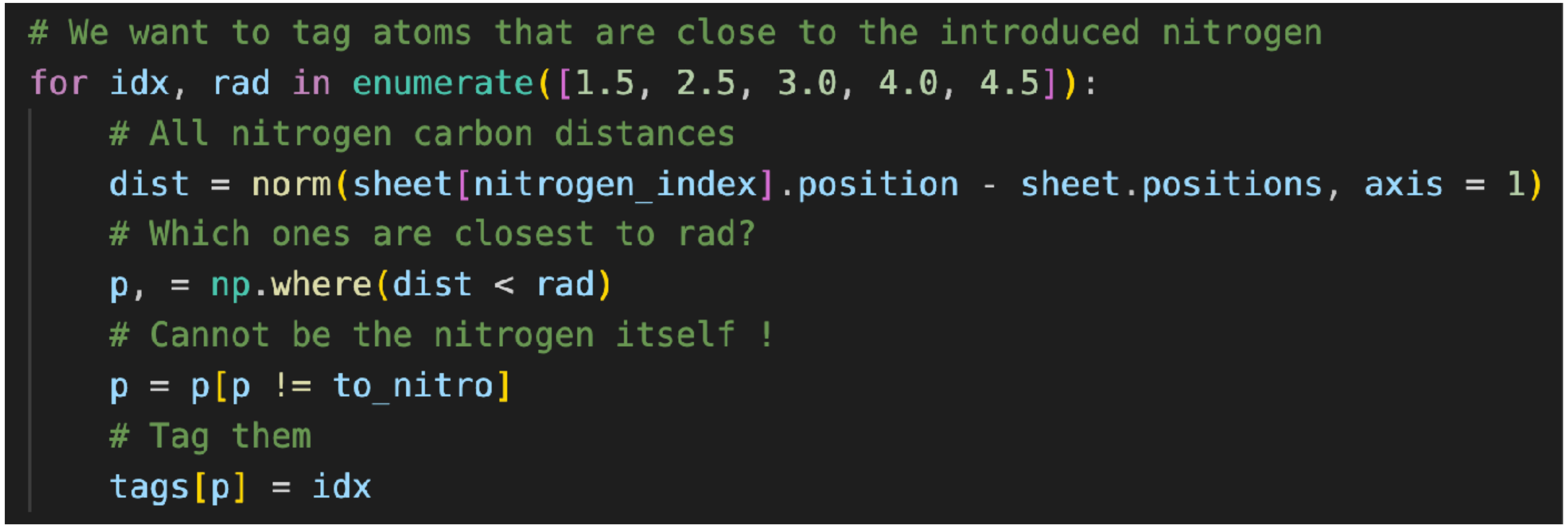


```
sheet = graphene_nanoribbon(10, 10, type='zigzag', vacuum = 10)

# Get all distances to center of mass
com = sheet.get_center_of_mass()
distances_to_com = norm(sheet.positions - com, axis = 1)

# Find atoms close to com and change one randomly to N
p, = np.where(distances_to_com < 5)
to_nitro = choice(p)
sheet[to_nitro].symbol = 'N'
```

Based on the distance, we randomly introduce a nitrogen atoms

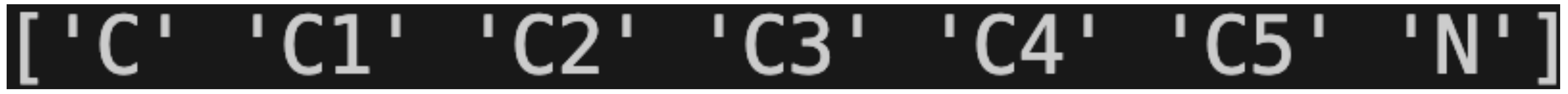


We number the carbon atoms depending on how close they are to the previously introduced nitrogen



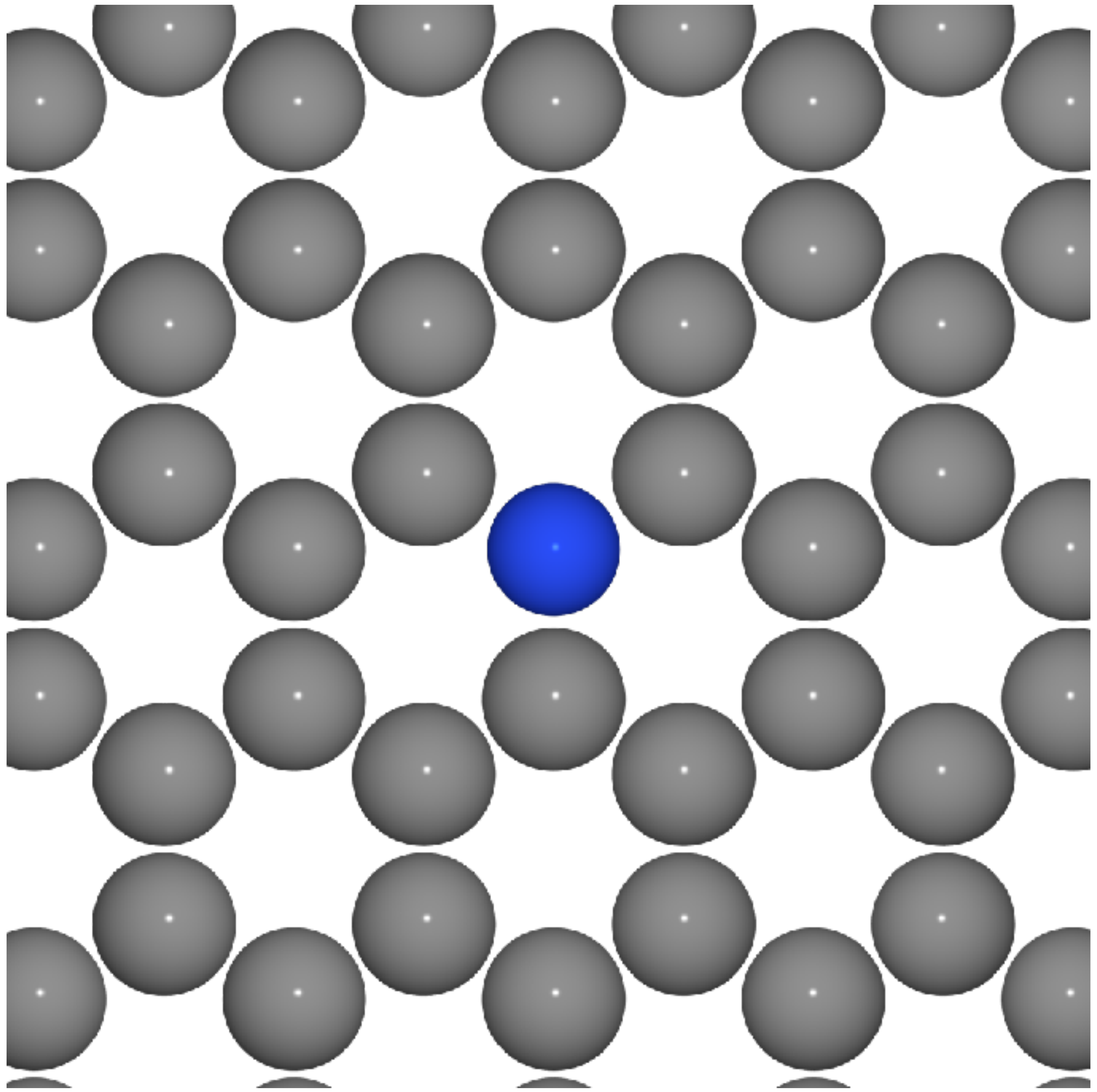
```python
# We want to tag atoms that are close to the introduced nitrogen
for idx, rad in enumerate([1.5, 2.5, 3.0, 4.0, 4.5]):
    # All nitrogen carbon distances
    dist = norm(sheet[nitrogen_index].position - sheet.positions, axis = 1)
    # Which ones are closest to rad?
    p, = np.where(dist < rad)
    # Cannot be the nitrogen itself !
    p = p[p != to_nitro]
    # Tag them
    tags[p] = idx
```

The aim is to generate these species, that we can feed to ONETEP.

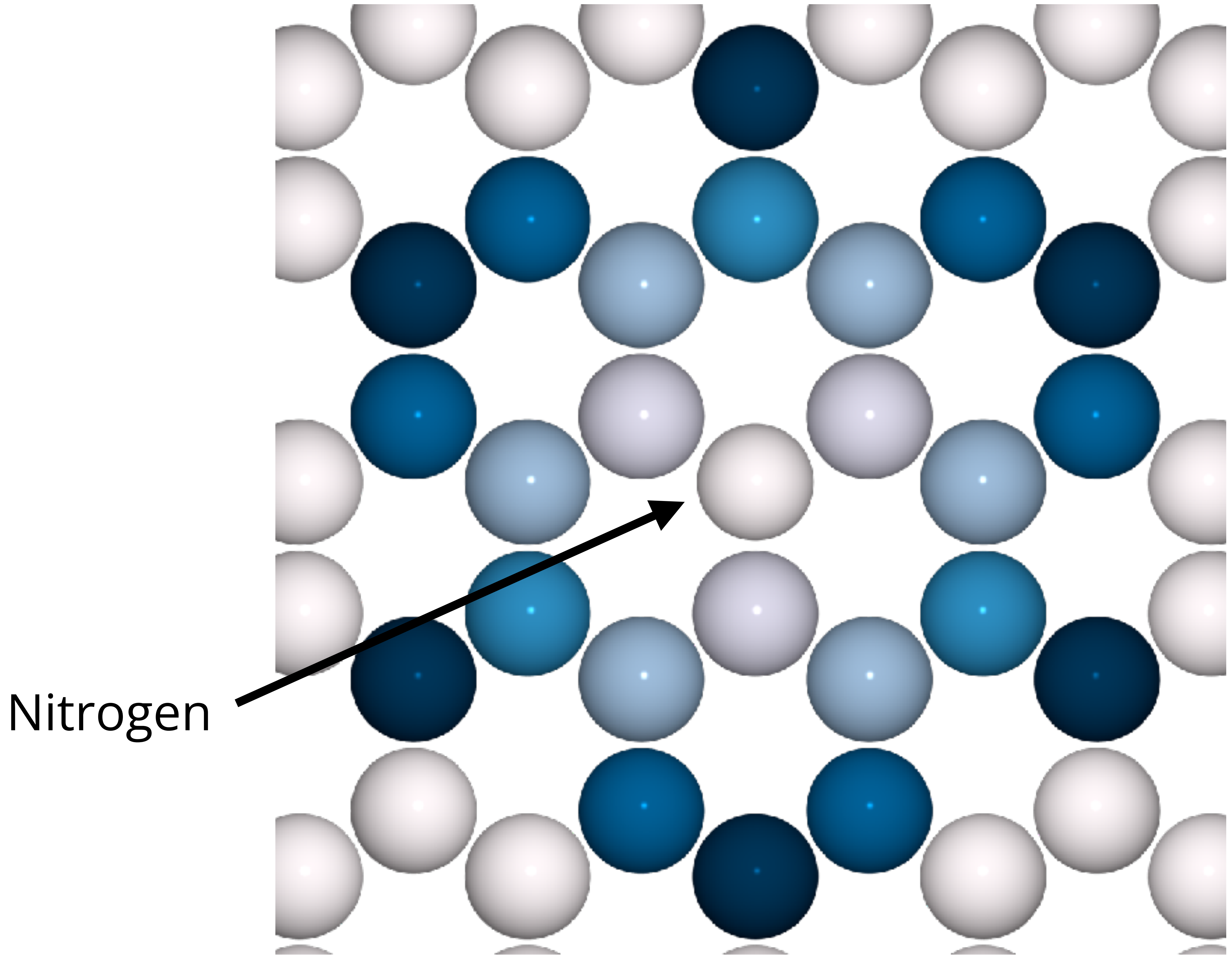


```
['C' 'C1' 'C2' 'C3' 'C4' 'C5' 'N']
```

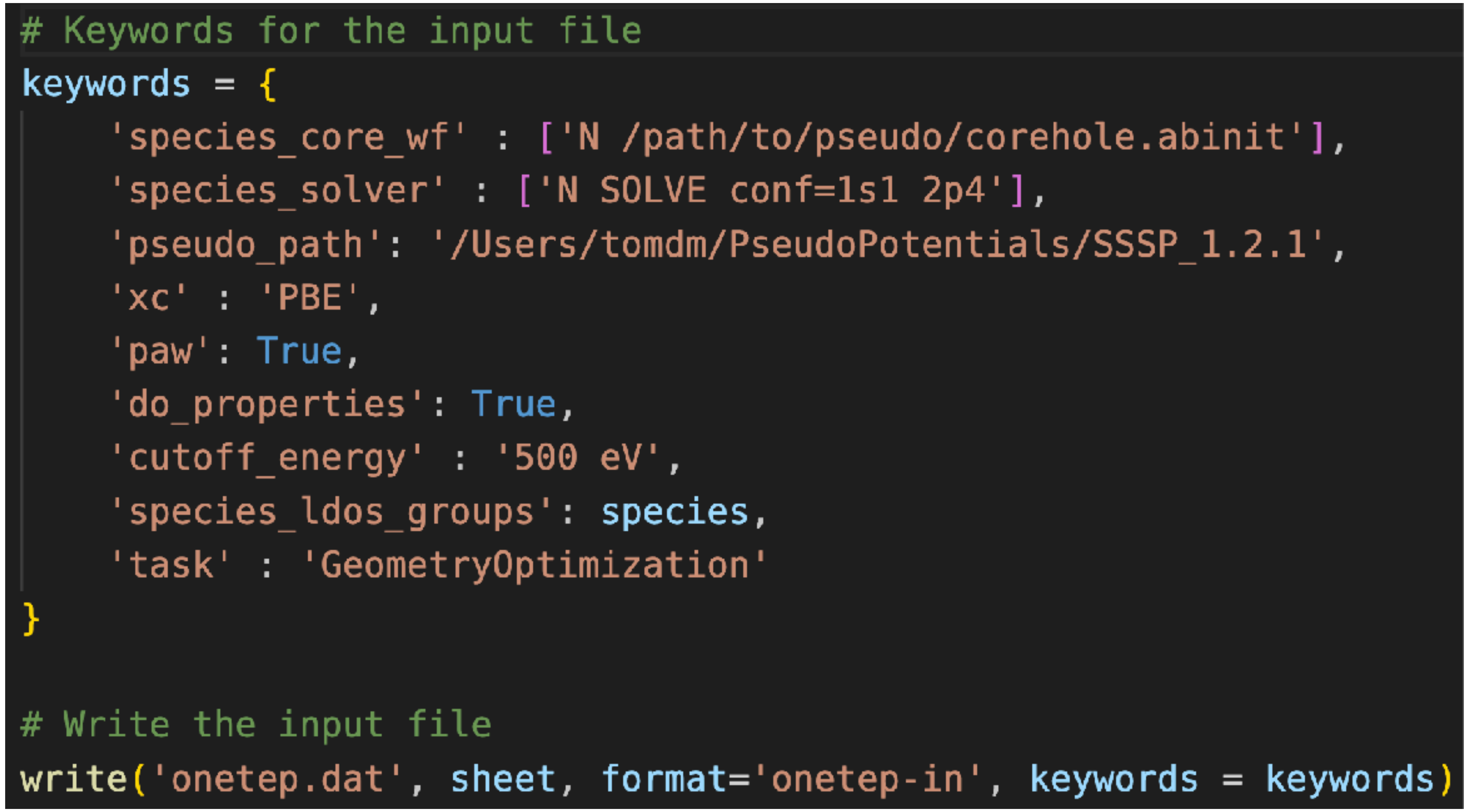
Normal colouring



Colouring based on tag number

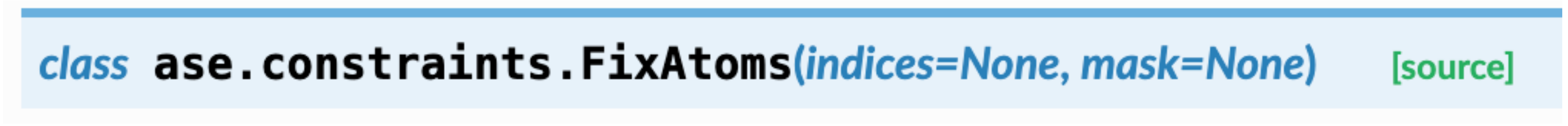


Finally, we write the onetep input file using the specified parameters

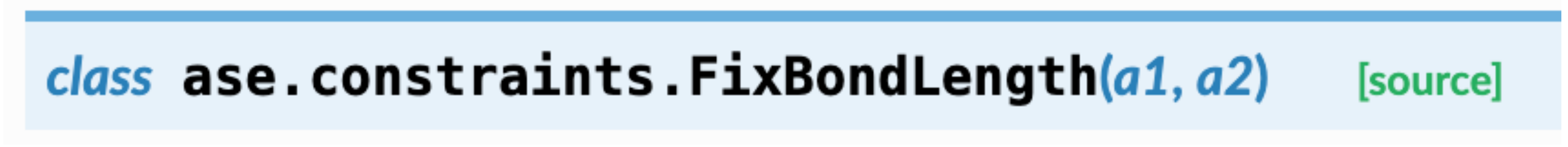


```python
# Keywords for the input file
keywords = {
    'species_core_wf' : ['N /path/to/pseudo/corehole.abinit'],
    'species_solver' : ['N SOLVE conf=1s1 2p4'],
    'pseudo_path': '/Users/tomdm/PseudoPotentials/SSSP_1.2.1',
    'xc' : 'PBE',
    'paw': True,
    'do_properties': True,
    'cutoff_energy' : '500 eV',
    'species_ldos_groups': species,
    'task' : 'GeometryOptimization'
}

# Write the input file
write('onetep.dat', sheet, format='onetep-in', keywords = keywords)
```

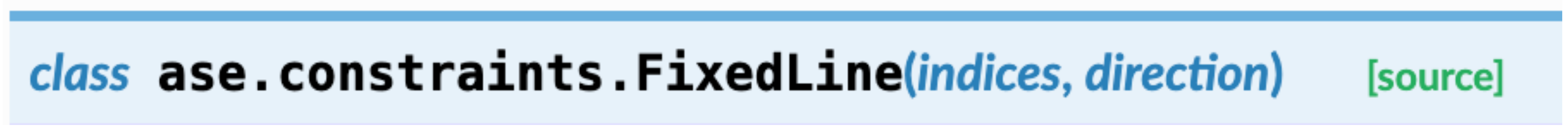
ONETEP will then calculate the LDOS for every carbon subgroups.



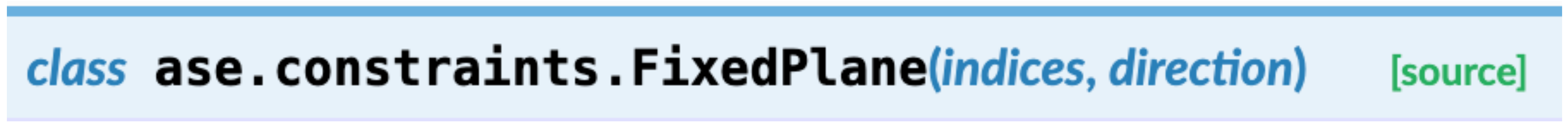
class ase.constraints.FixAtoms(*indices=None, mask=None*) [source]



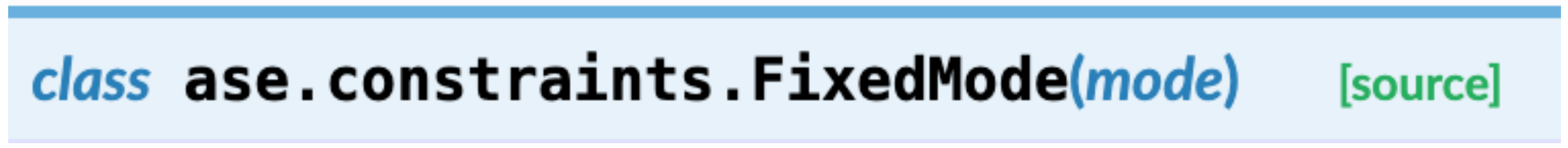
class ase.constraints.FixBondLength(*a1, a2*) [source]



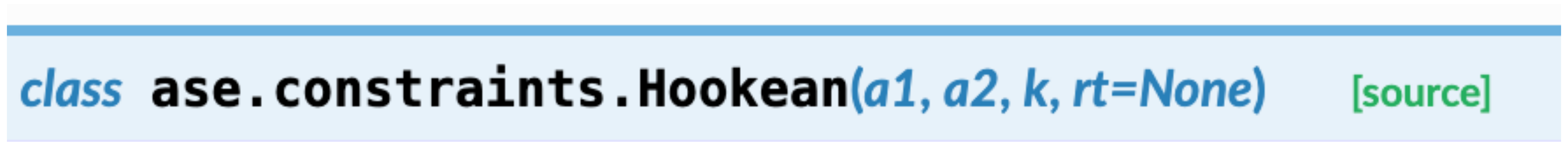
class ase.constraints.FixedLine(*indices, direction*) [source]



class ase.constraints.FixedPlane(*indices, direction*) [source]



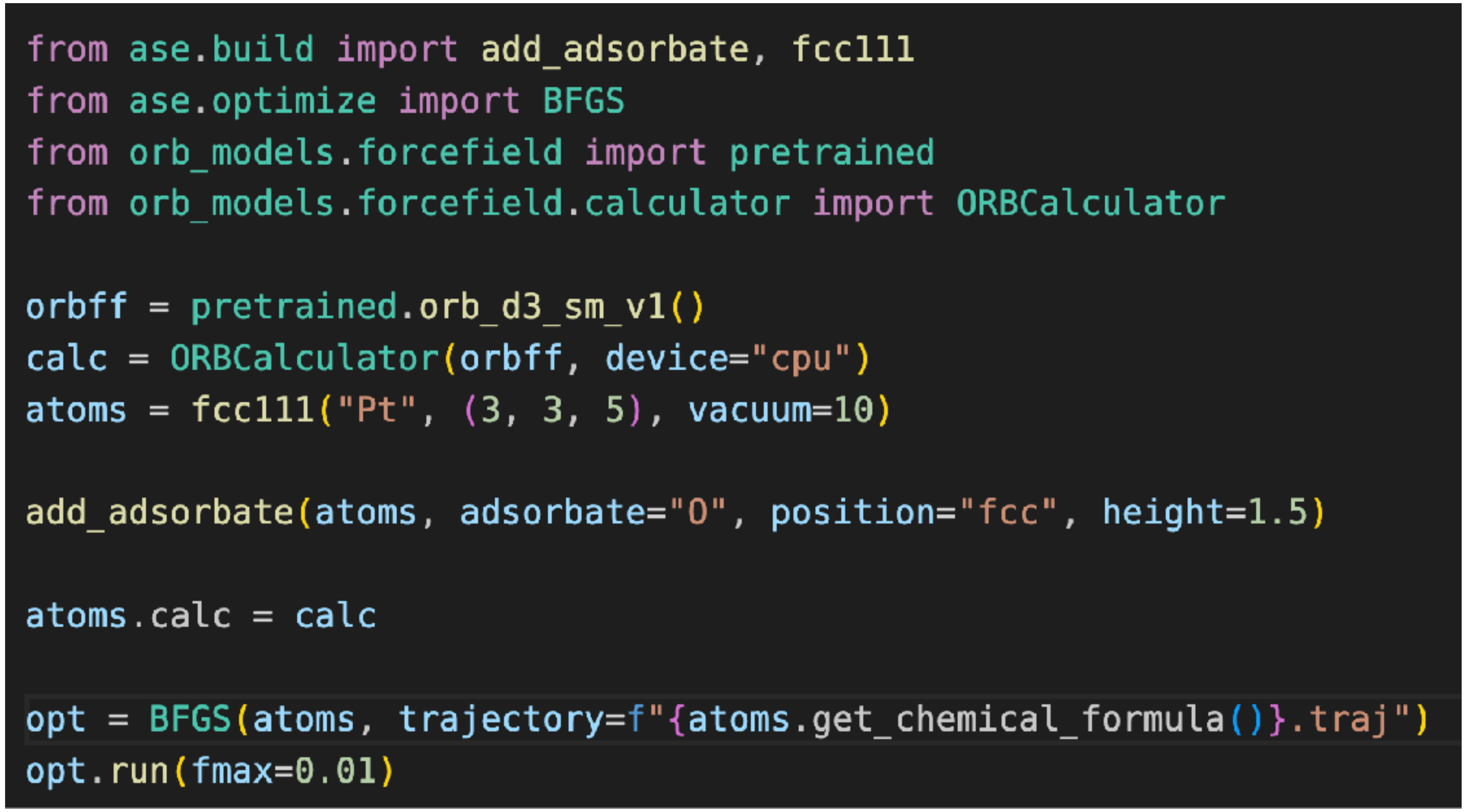
class ase.constraints.FixedMode(*mode*) [source]



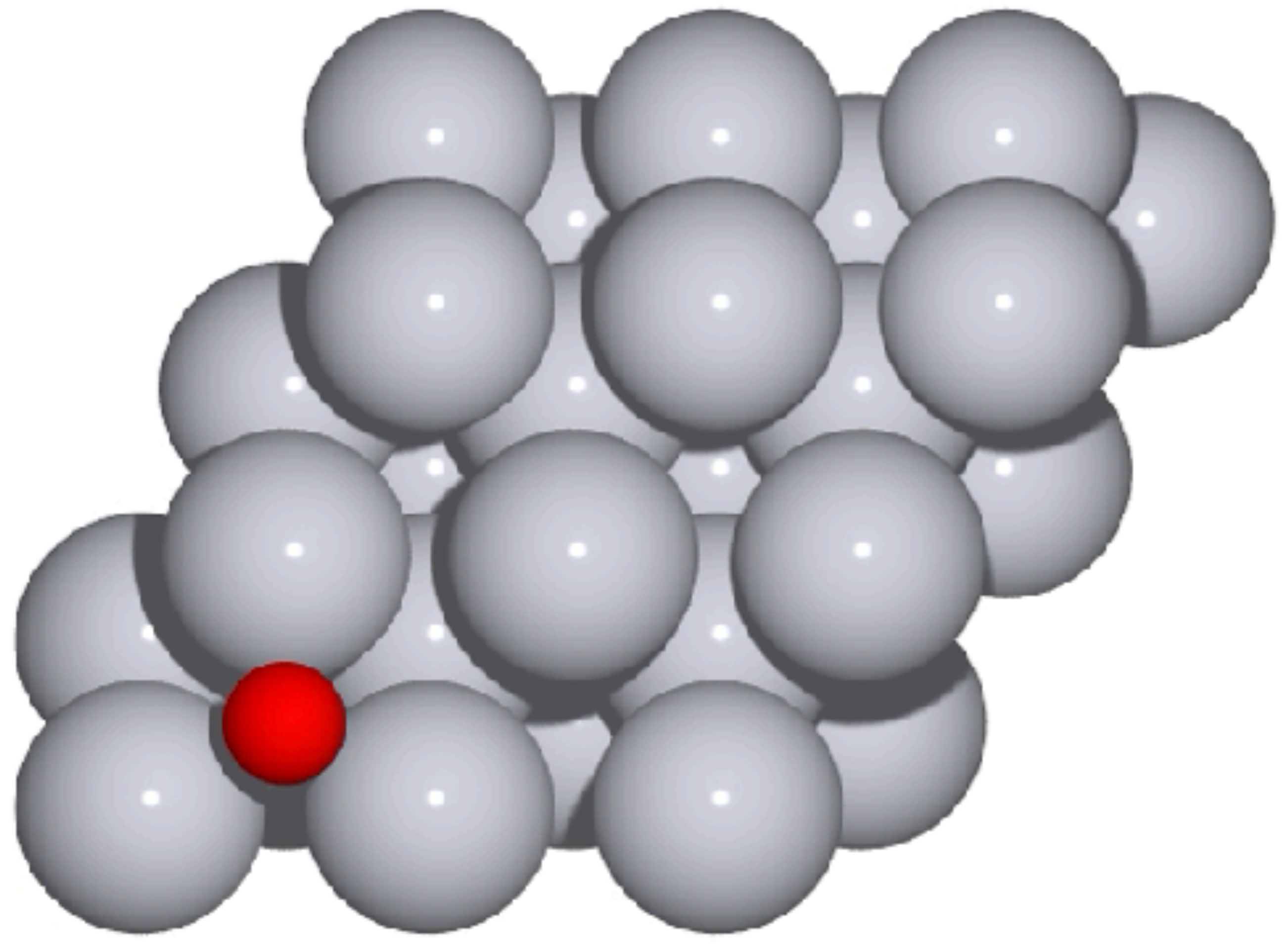
class ase.constraints.Hookean(*a1, a2, k, rt=None*) [source]

Orbital Materials

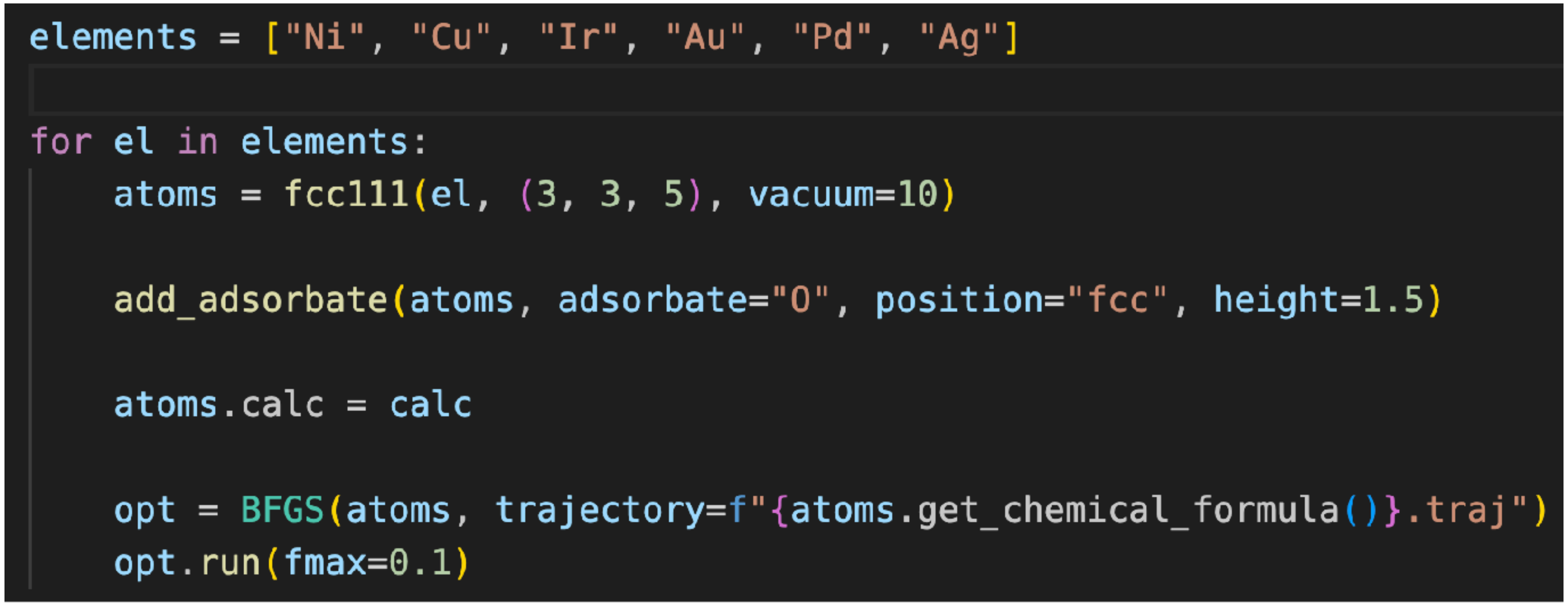
ORB, currently ranked 1st on the benchmark of Machine Learned models can be used with ASE to pre-optimize systems.



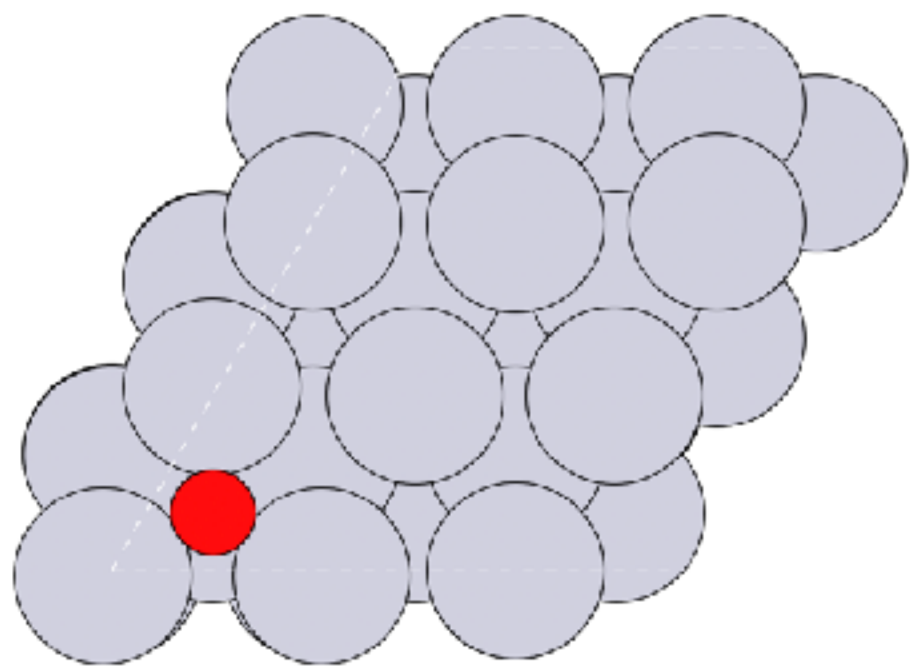
```python
from ase.build import add_adsorbate, fcc111
from ase.optimize import BFGS
from orb_models.forcefield import pretrained
from orb_models.forcefield.calculator import ORBCalculator

orbff = pretrained.orb_d3_sm_v1()
calc = ORBCalculator(orbff, device="cpu")
atoms = fcc111("Pt", (3, 3, 5), vacuum=10)

add_adsorbate(atoms, adsorbate="O", position="fcc", height=1.5)

atoms.calc = calc

opt = BFGS(atoms, trajectory=f"{atoms.get_chemical_formula()}.traj")
opt.run(fmax=0.01)
```

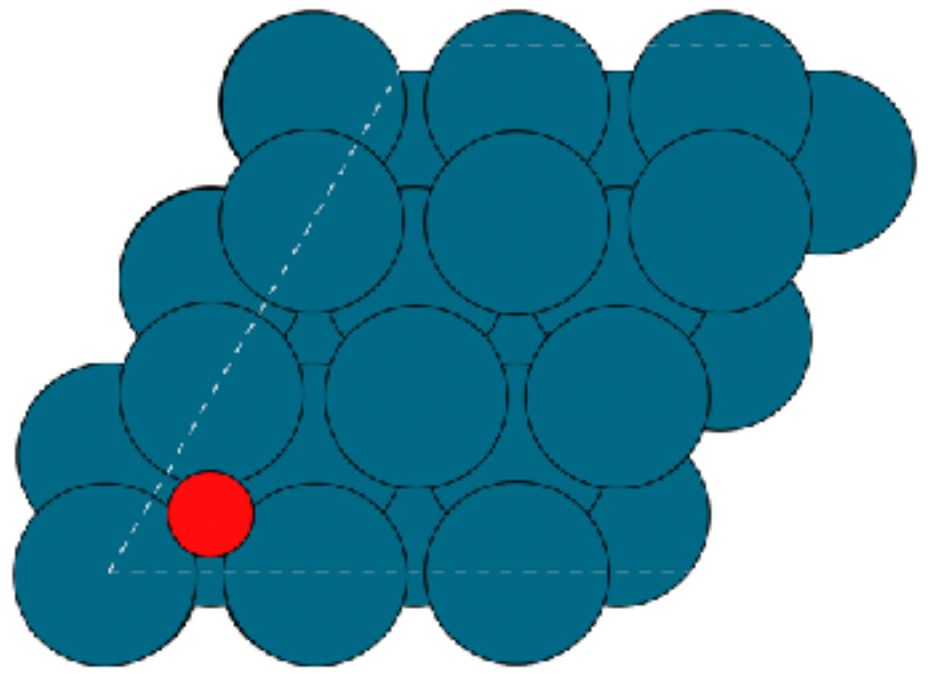


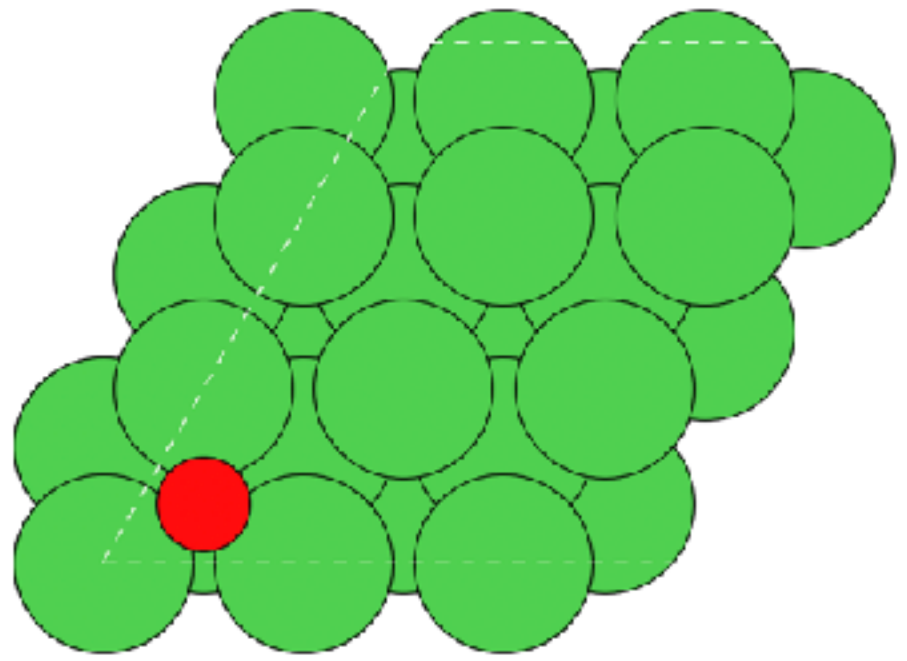
Changing the script to run the model for various elemental surfaces:

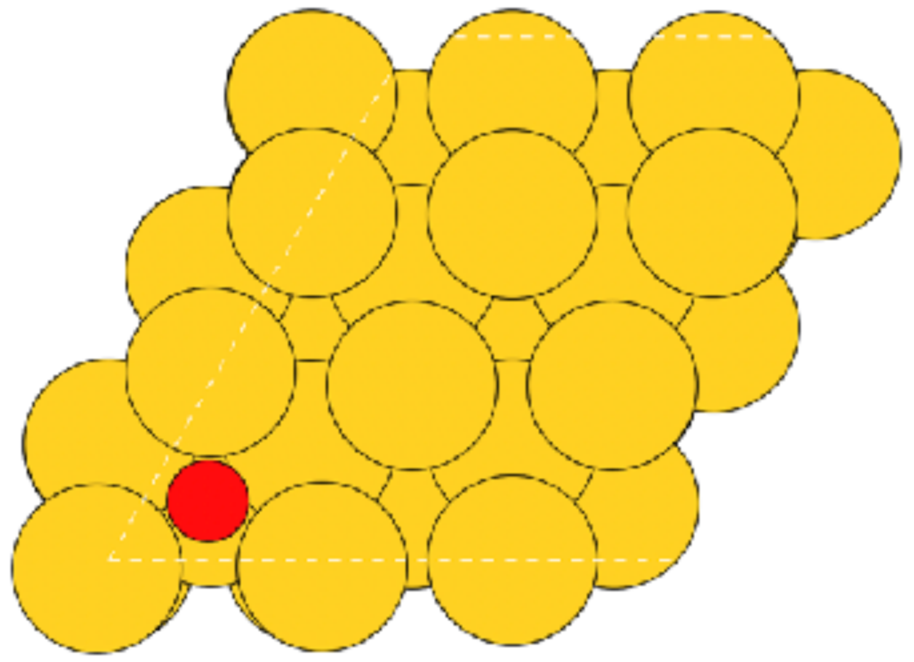


```python
elements = ["Ni", "Cu", "Ir", "Au", "Pd", "Ag"]

for el in elements:
    atoms = fcc111(el, (3, 3, 5), vacuum=10)

    add_adsorbate(atoms, adsorbate="O", position="fcc", height=1.5)

    atoms.calc = calc

    opt = BFGS(atoms, trajectory=f"{atoms.get_chemical_formula()}.traj")
    opt.run(fmax=0.1)
```









Python ecosystem for computational chemistry

QuAcc

PySAGES

ACAT

organometallics solids surfaces main group

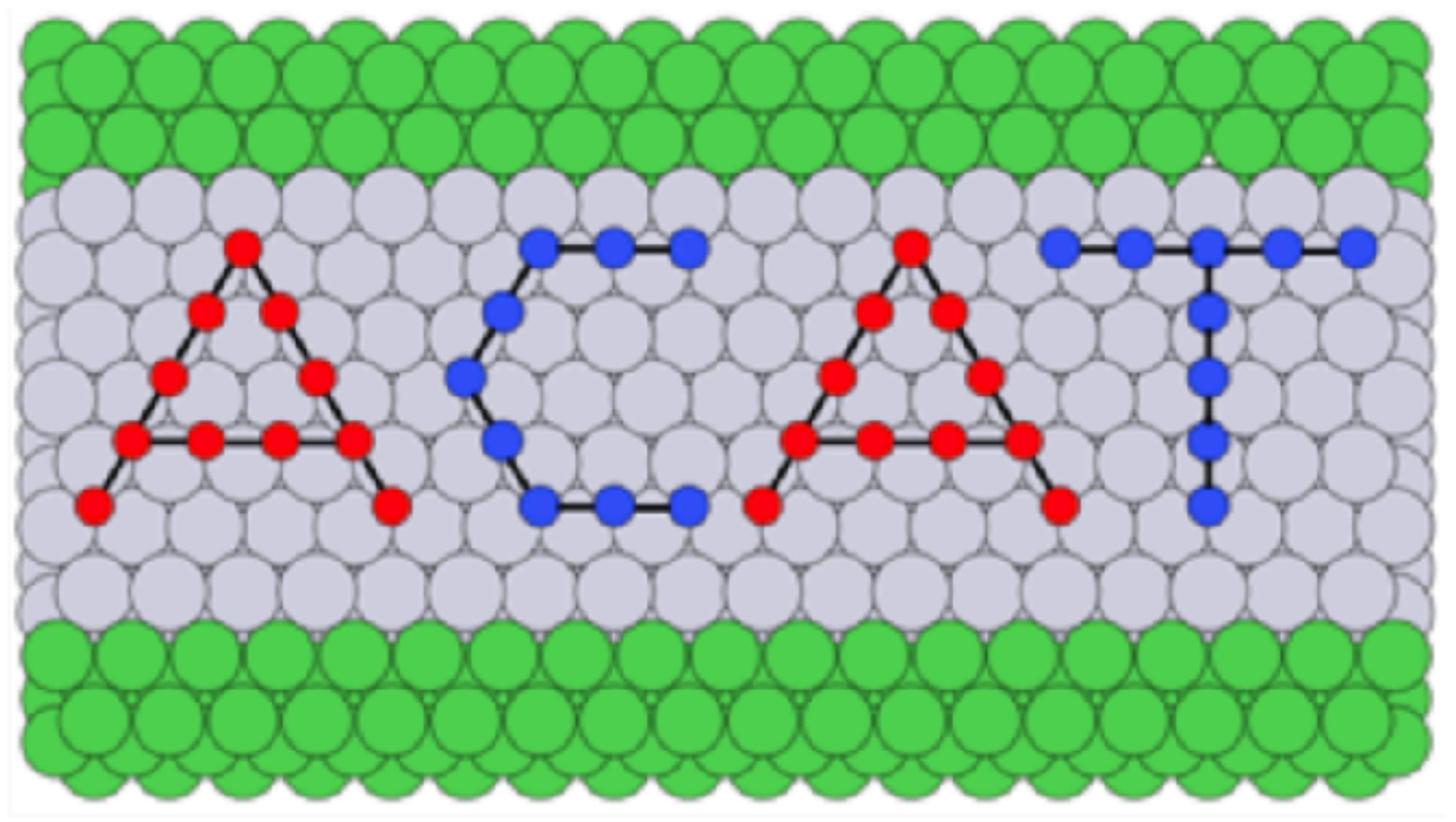
D4

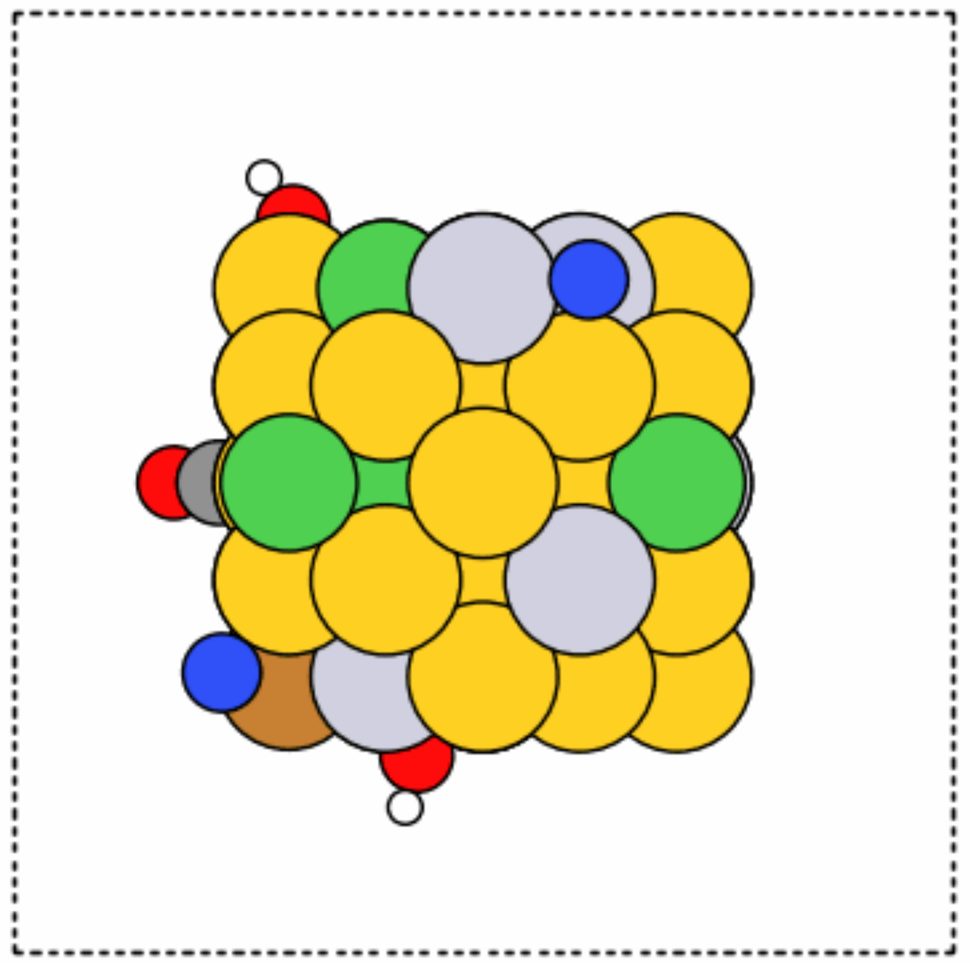
pymatgen

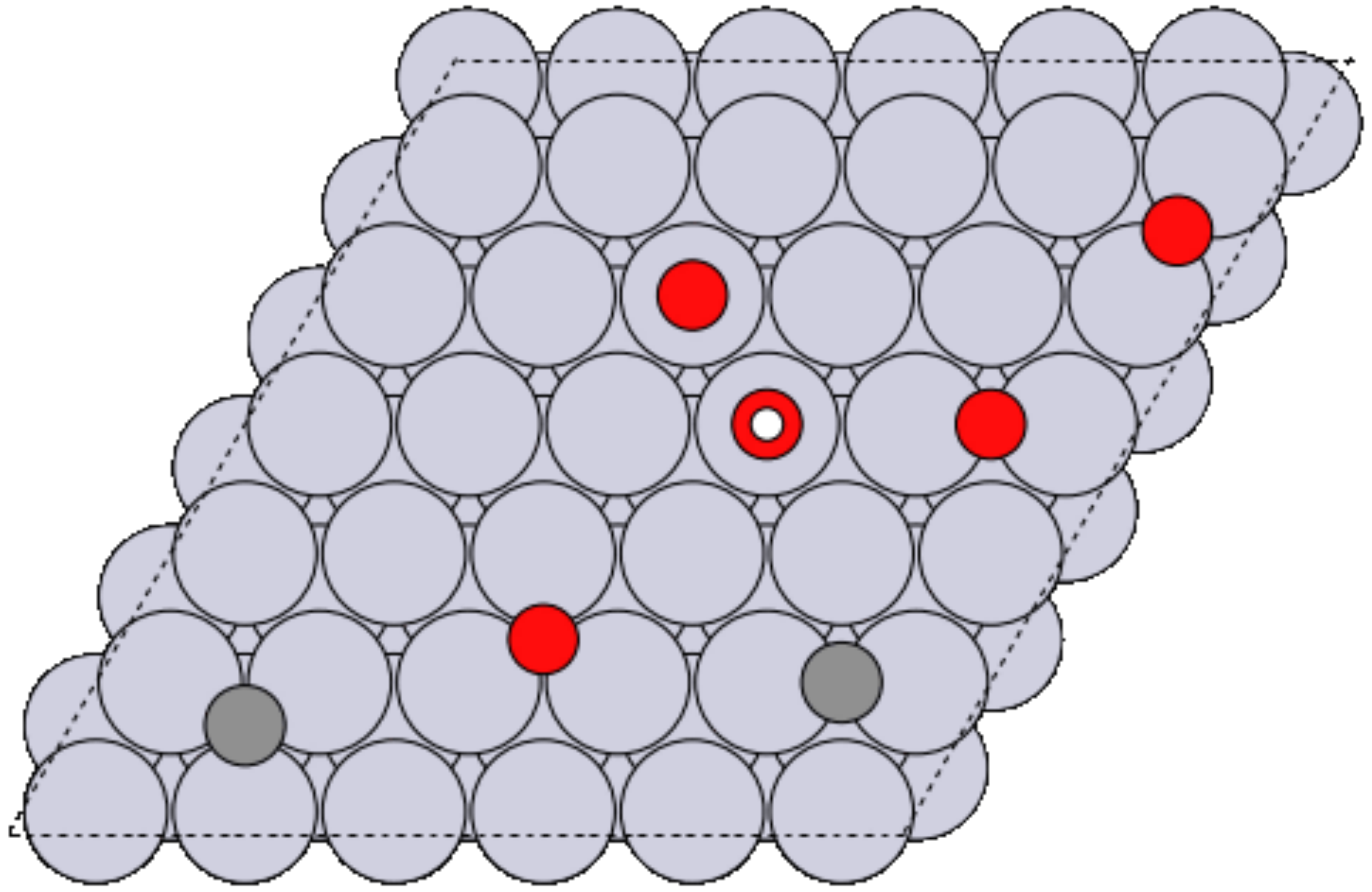
Sella

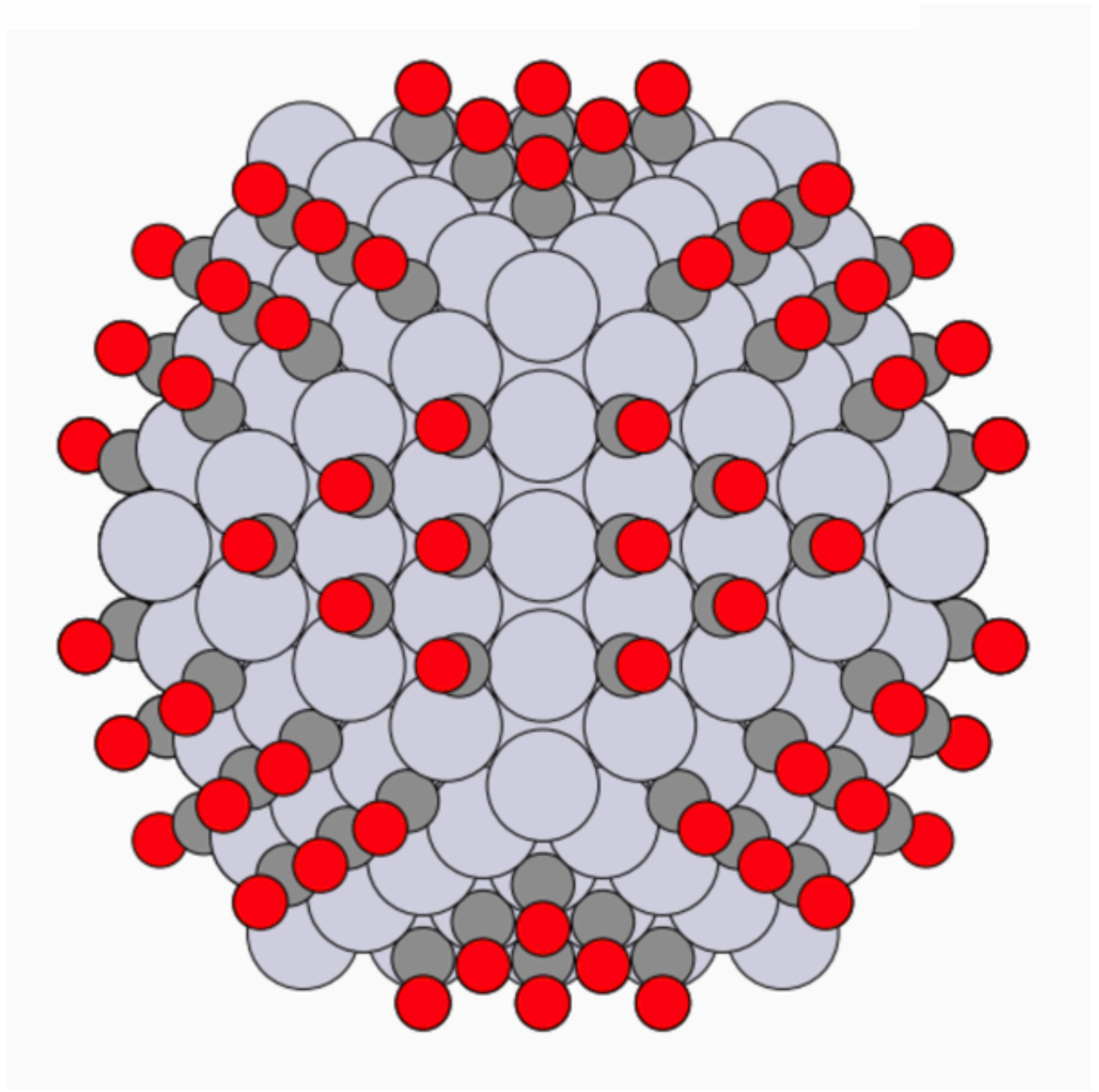
abTEM

Phonopy

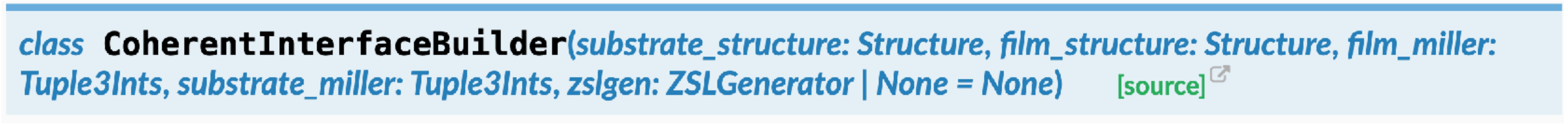




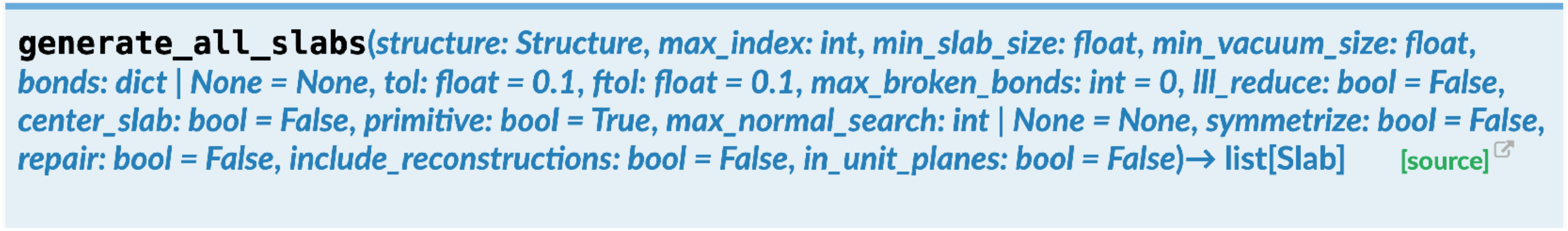




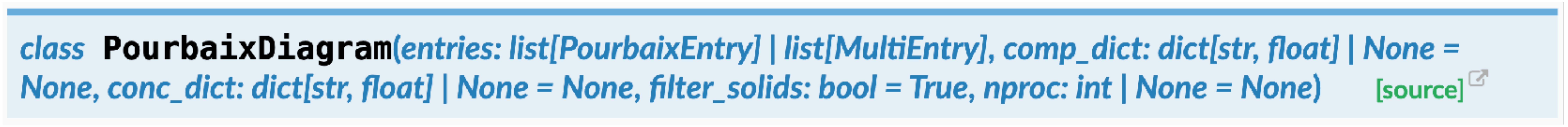
pymatgen



```
class CoherentInterfaceBuilder(substrate_structure: Structure, film_structure: Structure, film_miller: Tuple3Ints, substrate_miller: Tuple3Ints, zslgen: ZSLGenerator | None = None)    [source]
```

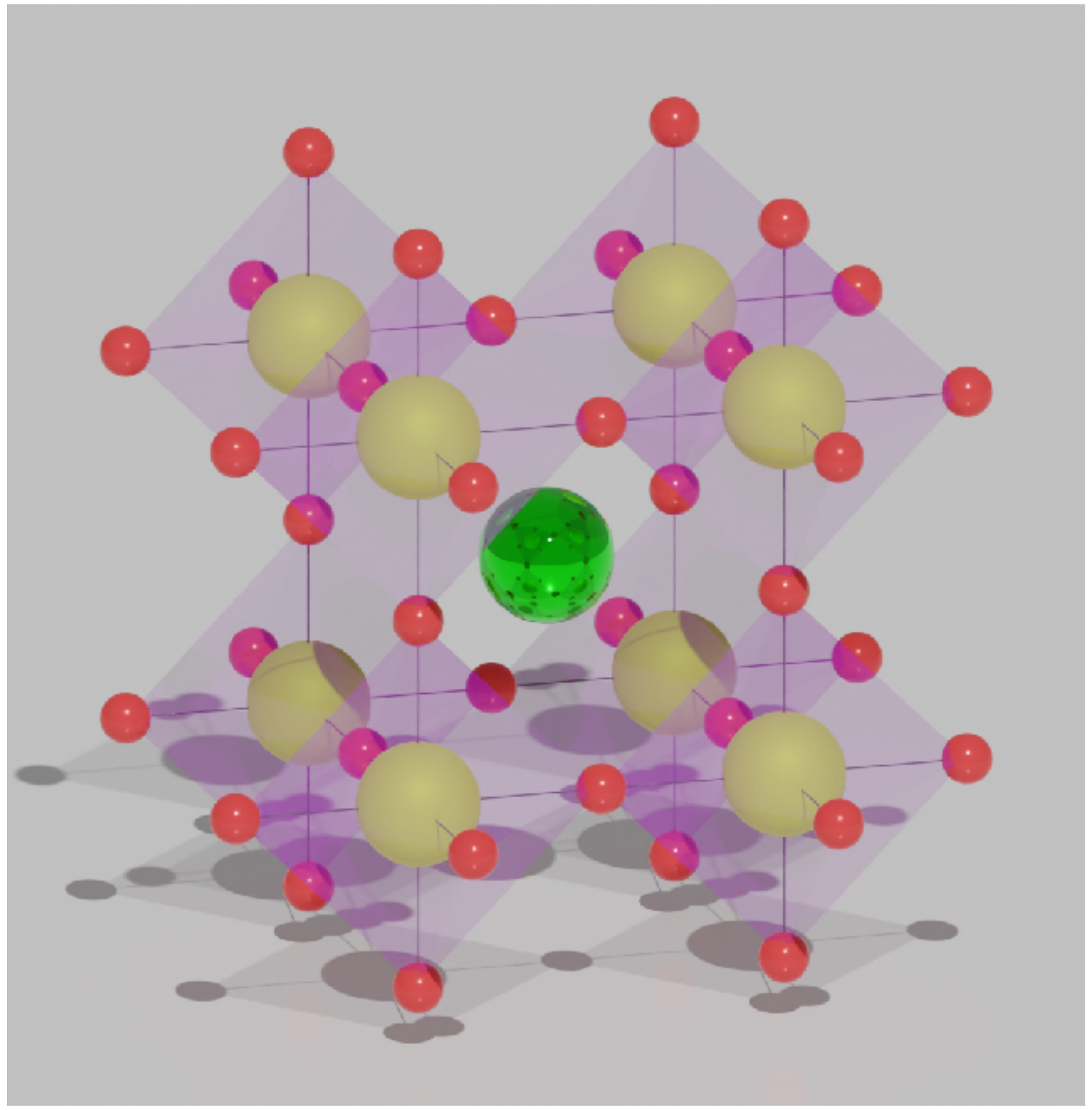


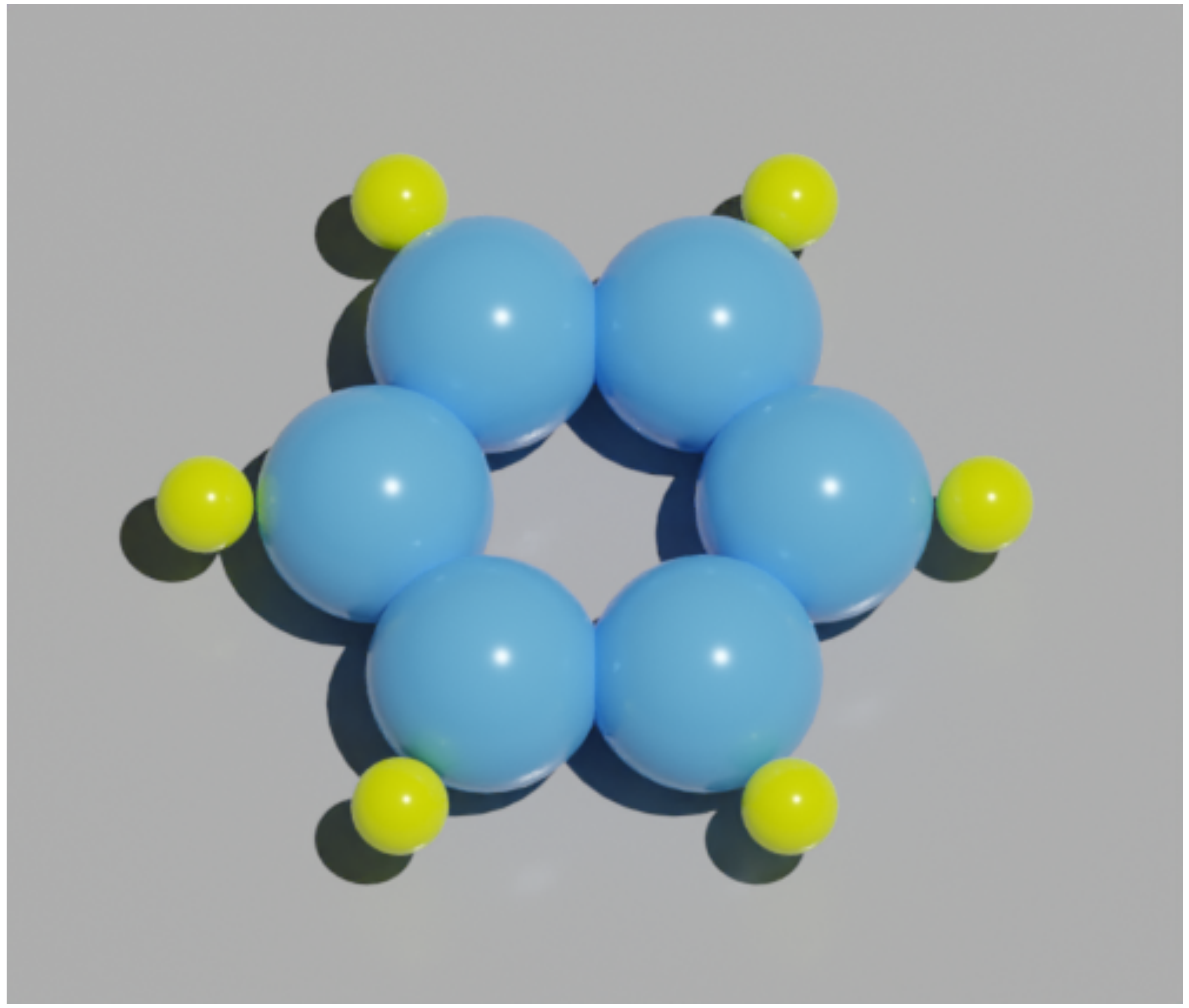
```
generate_all_slabs(structure: Structure, max_index: int, min_slab_size: float, min_vacuum_size: float, bonds: dict | None = None, tol: float = 0.1, ftol: float = 0.1, max_broken_bonds: int = 0, lll_reduce: bool = False, center_slab: bool = False, primitive: bool = True, max_normal_search: int | None = None, symmetrize: bool = False, repair: bool = False, include_reconstructions: bool = False, in_unit_planes: bool = False)→ list[Slab]    [source]
```

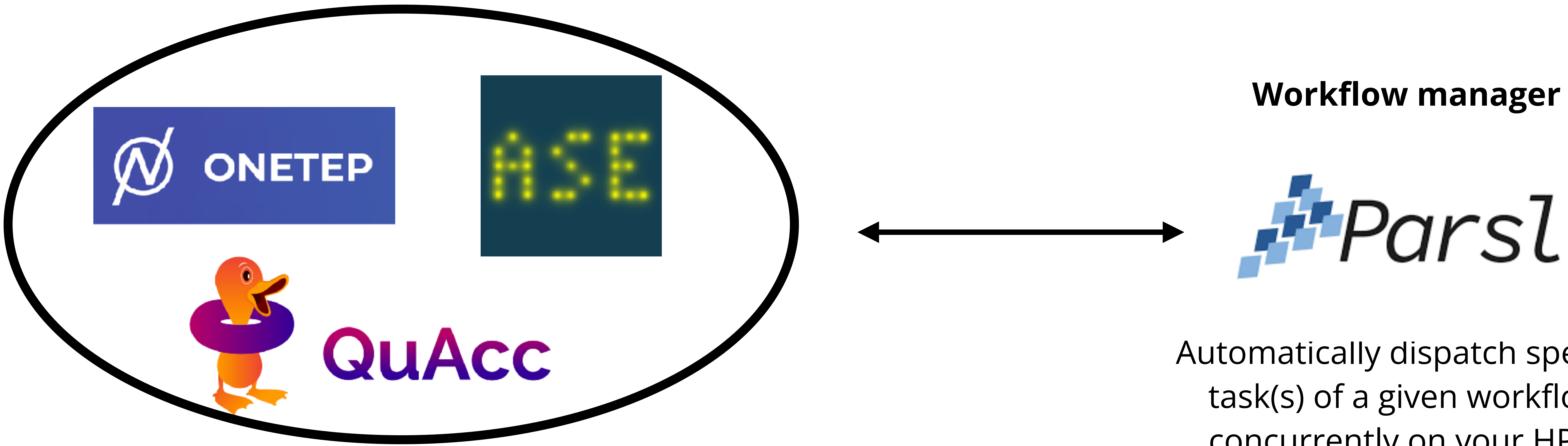


```
class PourbaixDiagram(entries: list[PourbaixEntry] | list[MultiEntry], comp_dict: dict[str, float] | None = None, conc_dict: dict[str, float] | None = None, filter_solids: bool = True, nproc: int | None = None)    [source]
```

beautiful-atoms Public

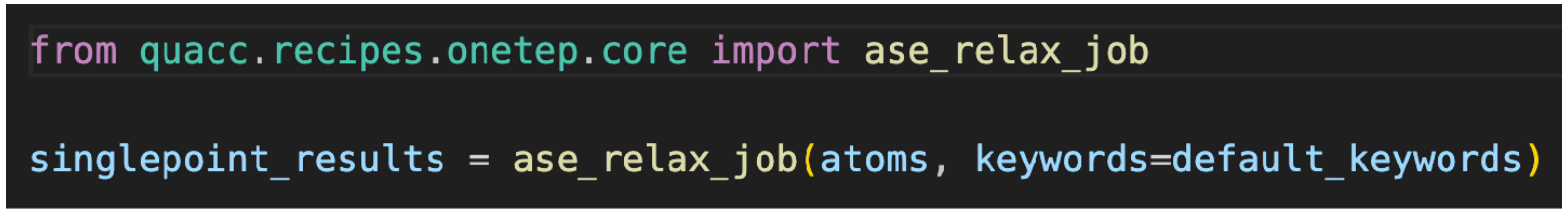






Automatically dispatch specific task(s) of a given workflow concurrently on your HPC.

With Quacc, running a ONETEP calculation is reduced to a simple line:



```
from quacc.recipes.onetep.core import ase_relax_job

singlepoint_results = ase_relax_job(atoms, keywords=default_keywords)
```