```
#---------------------------------------
# PYTHON VARIABLES
#---------------------------------------

from math import pi, sqrt, log, exp
import numpy as np

L_0 = 2.*pi          # laser wavelength (lambda)
t_0 = L_0            # optical cycle

dx = 0.0125*L_0         # cells per wavelength
dr = 0.05*L_0
Lx = 224.0 * L_0      # grid size
Lr = 160.0 * L_0
nx = Lx/dx
nr = Lr/dr

t_sim = 700.0 * t_0
t_fwhm = 10.6*t_0       # c*tau/lambda

nppc = 16             # number of particles-per-cell
lambda0 = 0.8e-6

diag_every = 15*t_0 / dt

Radius_plasma =144*L_0

def n_He(x,r):
    if x > 50*L_0:
        return 1.5*(r<Radius_plasma)*0.5*(0.85**2)*(2.14e18)/(1.1*10e21)
    else:
        return 0

def ne(x,r):
    if x > 50*L_0:
        return 1.5*(r<Radius_plasma)*(0.85**2)*(2.14e18)/(1.1*10e21)
    else:
        return 0

r_n = 0.08*pi #d=80nm, r = 0.085/2*2*pi

def nanoparticle(x,r):
    if (x-250*2*pi)**2+(r)**2 < r_n**2:
       return 40
    else:
       return 0
```

```
#————————————————
# SMILEY NAMELIST
#————————————————

Main(
    geometry = "AMcylindrical",
    number_of_AM = 2,
    #patch_arrangement = "linearized_XY",
    timestep_over_CFL = 0.95,
    simulation_time = t_sim,
    cell_length  = [dx, dr],
    grid_length = [ Lx, Lr ],
    number_of_patches = [512, 128],
    EM_boundary_conditions = [
        ["PML","PML"],
        ["PML","PML"],
    ],
    solve_poisson = False,
    print_every = 100,
    reference_angular_frequency_SI = 2*pi*3e8/lambda0,
    random_seed = smilei_mpi_rank
)

MovingWindow(
    time_start = 200.*t_0,
    velocity_x = 0.9985
)

Species(
    name = 'He_ions',
    atomic_number = 2,
    position_initialization = 'regular',
    momentum_initialization = 'cold',
    particles_per_cell = nppc,
    mass = 4.*1836.0,
    charge = 2.0,
    time_frozen = t_sim,
    number_density = n_He,
    boundary_conditions = [
        ["remove", "remove"],
        ["reflective", "remove"],
    ],
)
```

```
Species(
    name = 'aluminium',
    ionization_model = 'tunnel',
    ionization_electrons = 'electronAl',
    atomic_number = 13,
    position_initialization = 'regular',
    momentum_initialization = 'cold',
    particles_per_cell = 100*nppc,
    mass = 26.*1836.0,
    charge = 0.0,
        time_frozen = t_sim,
    number_density = nanoparticle,
    boundary_conditions = [
        ["remove", "remove"],
        ["reflective", "remove"],
    ],
)

Species(
    name = 'electron',
    position_initialization = 'regular',
    momentum_initialization = 'cold',
    particles_per_cell = nppc,
    mass = 1.0,
    charge = -1.0,
    number_density = ne,
    mean_velocity = [0.0, 0.0, 0.0],
    boundary_conditions = [
        ["remove", "remove"],
        ["reflective", "remove"],
    ],
)

Species(
    name = 'electronAl',
    position_initialization = 'regular',
    momentum_initialization = 'cold',
    particles_per_cell = 0,
    mass = 1.0,
    charge = -1.0,
    number_density = 0.0,
        mean_velocity = [0.0, 0.0, 0.0],
    boundary_conditions = [
        ["remove", "remove"],
        ["reflective", "remove"],
    ],
)
```

```
LaserGaussianAM(
    box_side          = "xmin",
    a0                = 2.1,
    omega                         = 1,
    focus             = [33000.0, 0], # [295.0 * L_0, 0.5 * Ly]
    waist             = 53 * L_0,  # at 1/e electric field ... w_0
    polarization_phi = pi / 2.0,
    time_envelope    = tgaussian(fwhm=t_fwhm, center=2.0*t_fwhm)
)

Checkpoints(
    # restart_dir = "dump1",
    dump_step = 5000,
    #dump_minutes = 240.,
    exit_after_dump = False,
    keep_n_dumps = 2,
)

DiagProbe(
    every = 15 * t_0 / dt,
    origin    = [0., -nr*dr,0.],
    corners  = [ [nx*dx,-nr*dr,0.], [0,nr*dr,0.] ],
    number    = [nx, 2*nr],
    fields = ['Ex','Ey','Rho_electron','Rho_electronAl', 'Ez']
)

DiagFields(
    every = 15 * t_0 / dt,
    fields = ["Et_mode_1"]
)

def my_filter(particles):
        if Main.iteration * Main.timestep < 220*2*pi:
                return (particles.y < 1.)*(particles.x < 1.)*(particles.y > -1.)
                        *(particles.z < 1.)*(particles.z > -1.)
        elif Main.iteration * Main.timestep > 500*2*pi:
                return np.sqrt(1.0 + particles.px**2 + particles.py**2
                        + particles.pz**2) > 2
        else:
                return (particles.y > -350.)*(particles.y < 350.)
                        *(particles.z > -350.)*(particles.z < 350.)
```

```
DiagParticleBinning(
    deposited_quantity = "weight",
    every = diag_every,
    time_average = 1,
    species = ["electron"],
    axes = [ ["ekin", 0, 150, 500] ]
)

DiagParticleBinning(
    deposited_quantity = "weight",
    every = diag_every,
    time_average = 1,
    species = ["electronAl"],
    axes = [ ["ekin", 0, 150, 500] ]
)


DiagParticleBinning(
    deposited_quantity = "weight",
    every = diag_every,
    time_average = 1,
    species = ["electron"],
    axes = [ ["x", 1850, 6300, 500]]
)


DiagParticleBinning(
    deposited_quantity = "weight",
    every = diag_every,
    time_average = 1,
    species = ["electronAl"],
    axes = [ ["x", 1850, 6300, 500]]
)


DiagTrackParticles(
    species = "electron",
    every = 15 * t_0 / dt,
    filter = my_filter,
    attributes = ["x", "y", "z", "px", "py", "pz", "w"]
)

DiagTrackParticles(
    species = "electronAl",
    every = 15 * t_0 / dt,
    filter = my_filter,
    attributes = ["x", "y","z", "px", "py", "pz", "w"]
)
```