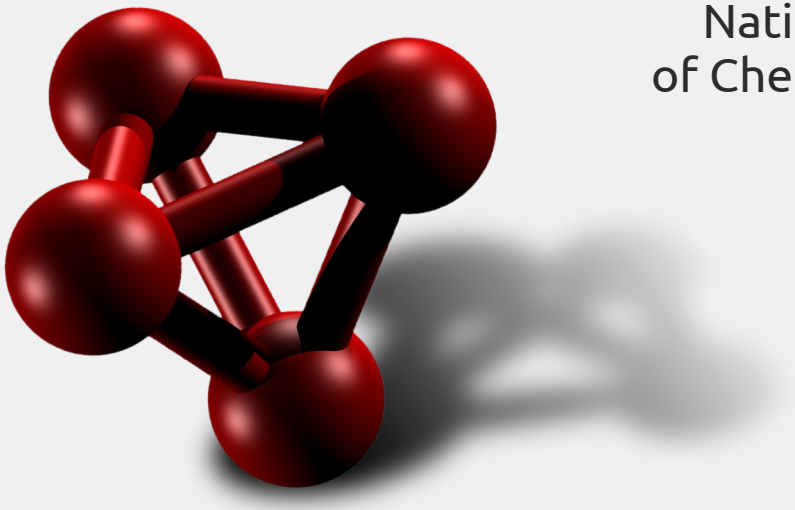
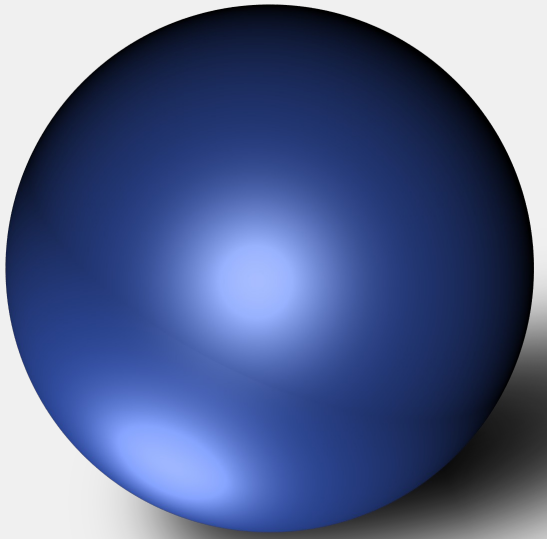
ESPResSo Summer School 2012

ESPResSo++ and AdResS

Staš Bevc
National Institute
of Chemistry, Slovenia





Acknowledgements

- Special thanks to the ESPResSo++ developer team

Current developers:

Torsten Stuehn (Max Planck Institute for Polymer Research, Germany)
Vitalii Starchenko (Max Planck Institute for Polymer Research, Germany)
Konstantin Koschke (Max Planck Institute for Polymer Research, Germany)
Livia Moreira (Max Planck Institute for Polymer Research, Germany)
Raffaello Potestio (Max Planck Institute for Polymer Research, Germany)
Karsten Kreis (Max Planck Institute for Polymer Research, Germany)

Former developers:

Thomas Brandes (Fraunhofer Institute SCAI, Germany)
Dirk Reith (Fraunhofer Institute SCAI, Germany)
Jonathan Halverson (Brookhaven National Laboratory, USA)
Axel Arnold (Institute for Computational Physics, Uni-Stuttgart, Germany)
Olaf Lenz (Institute for Computational Physics, Uni-Stuttgart, Germany)
Christoph Junghans (Los Alamos National Laboratory, USA)
Victor Ruehle (University of Cambridge, UK)

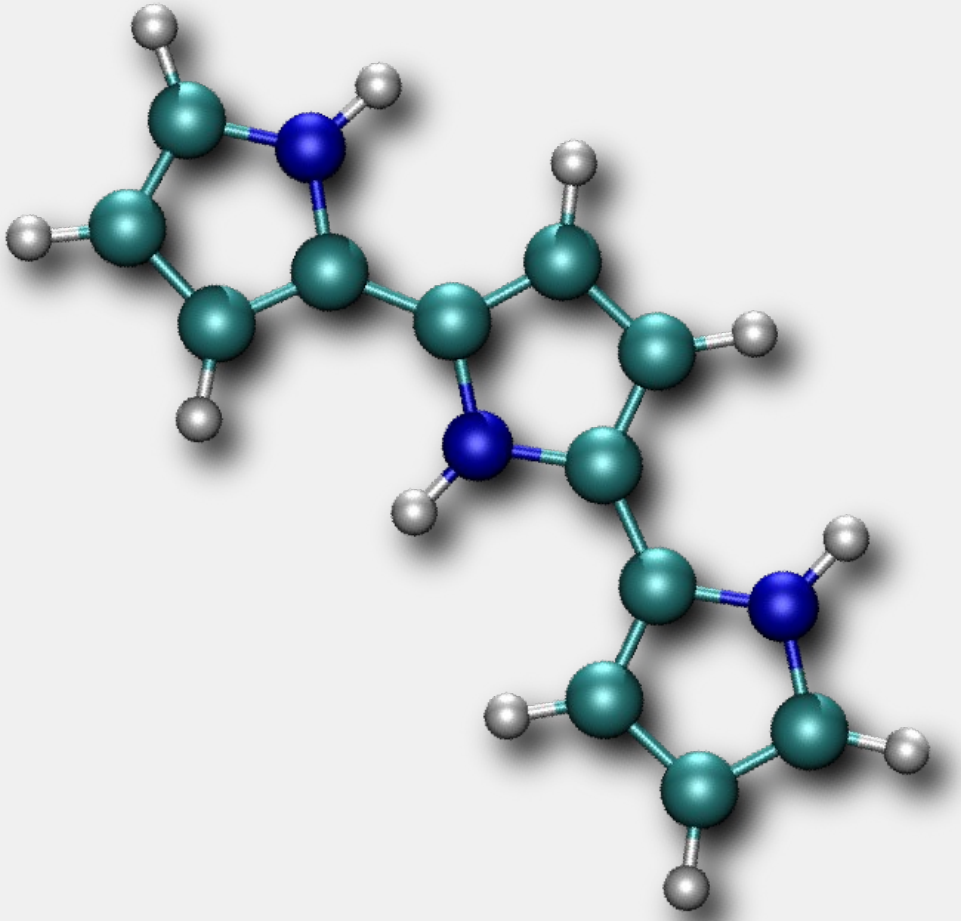
Outline

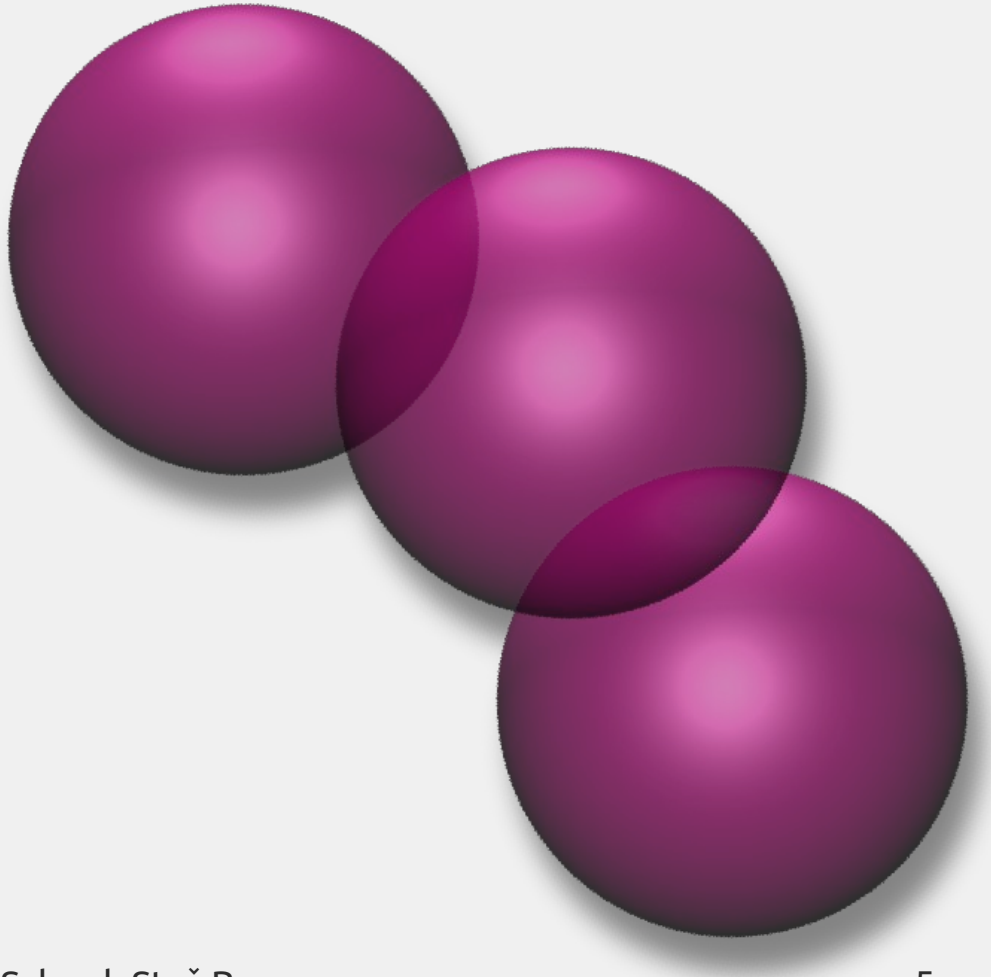
- Introduction to AdResS
- AdResS implementation in ESPResSo++
- Tetrahedral molecule example

Introduction to AdResS

Introduction to AdResS

- Motivation





Introduction to AdResS

- Motivation

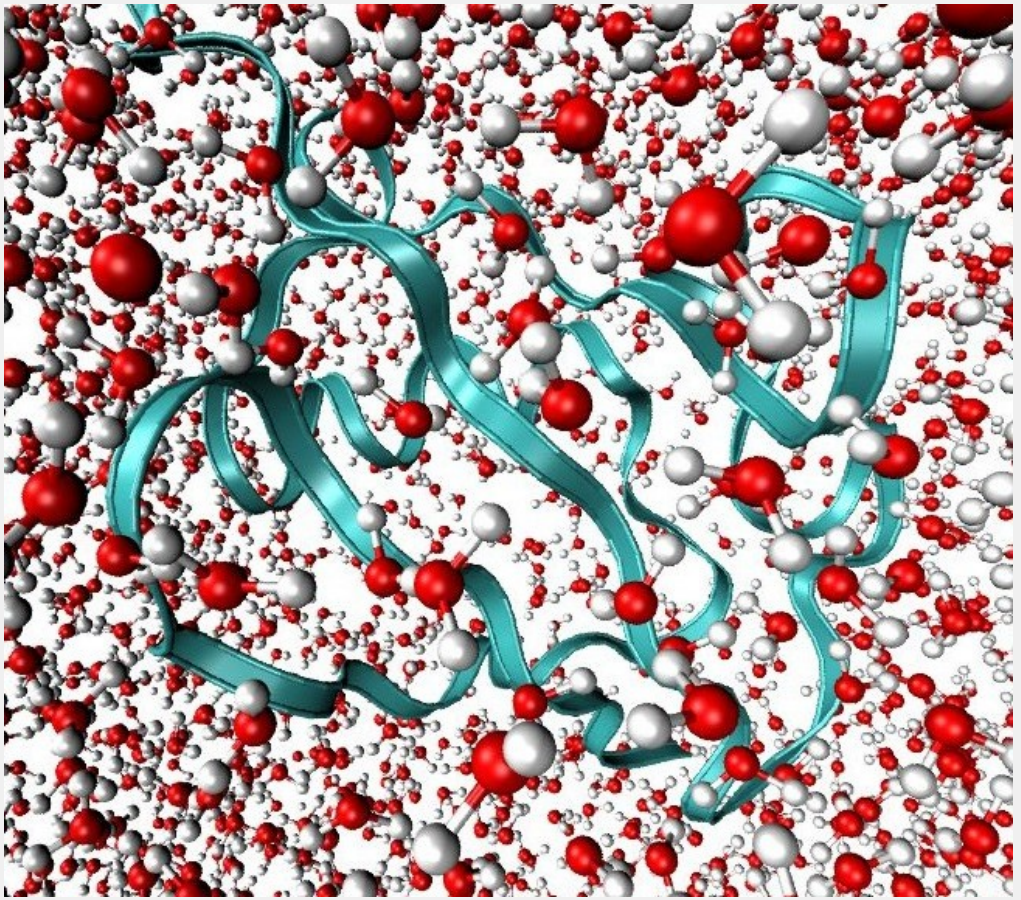
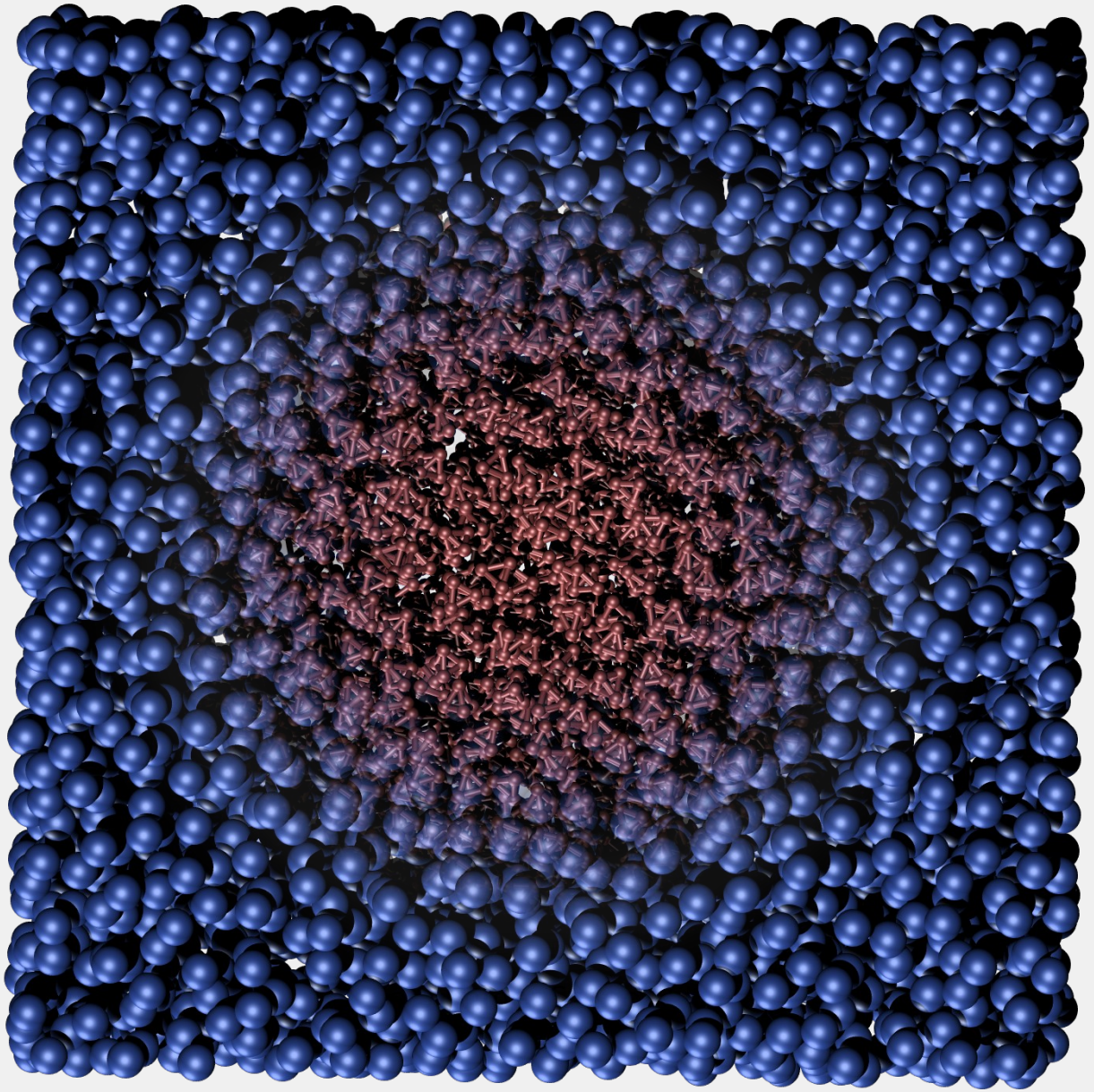


image source: Internet

Introduction to AdResS

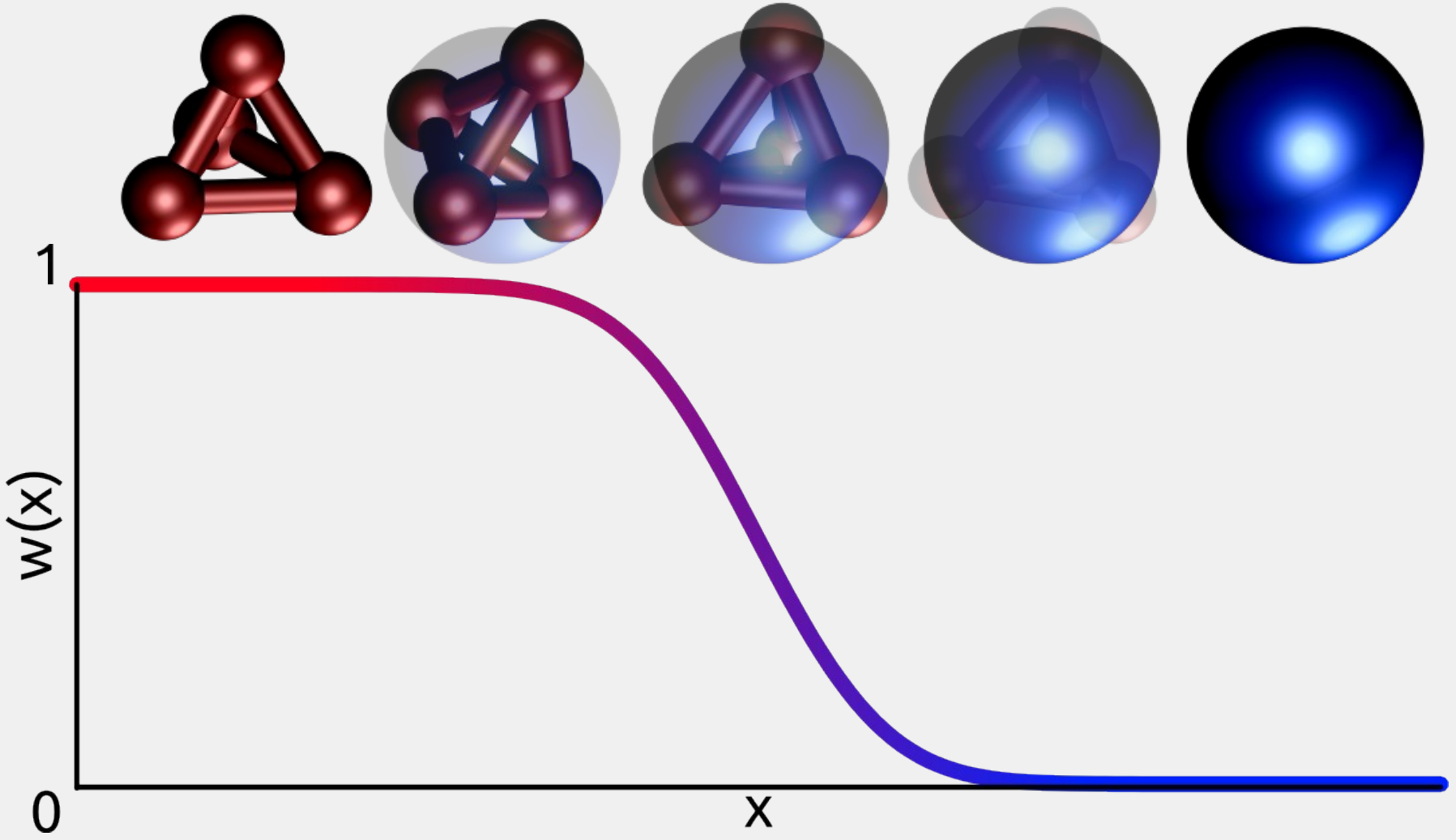


Introduction to AdResS

- Force calculation:

$$\boldsymbol{F}_{ij} = w_i w_j \boldsymbol{F}_{ij}^{AT} + \left(1 - w_i w_j\right) \boldsymbol{F}_{ij}^{CG}$$

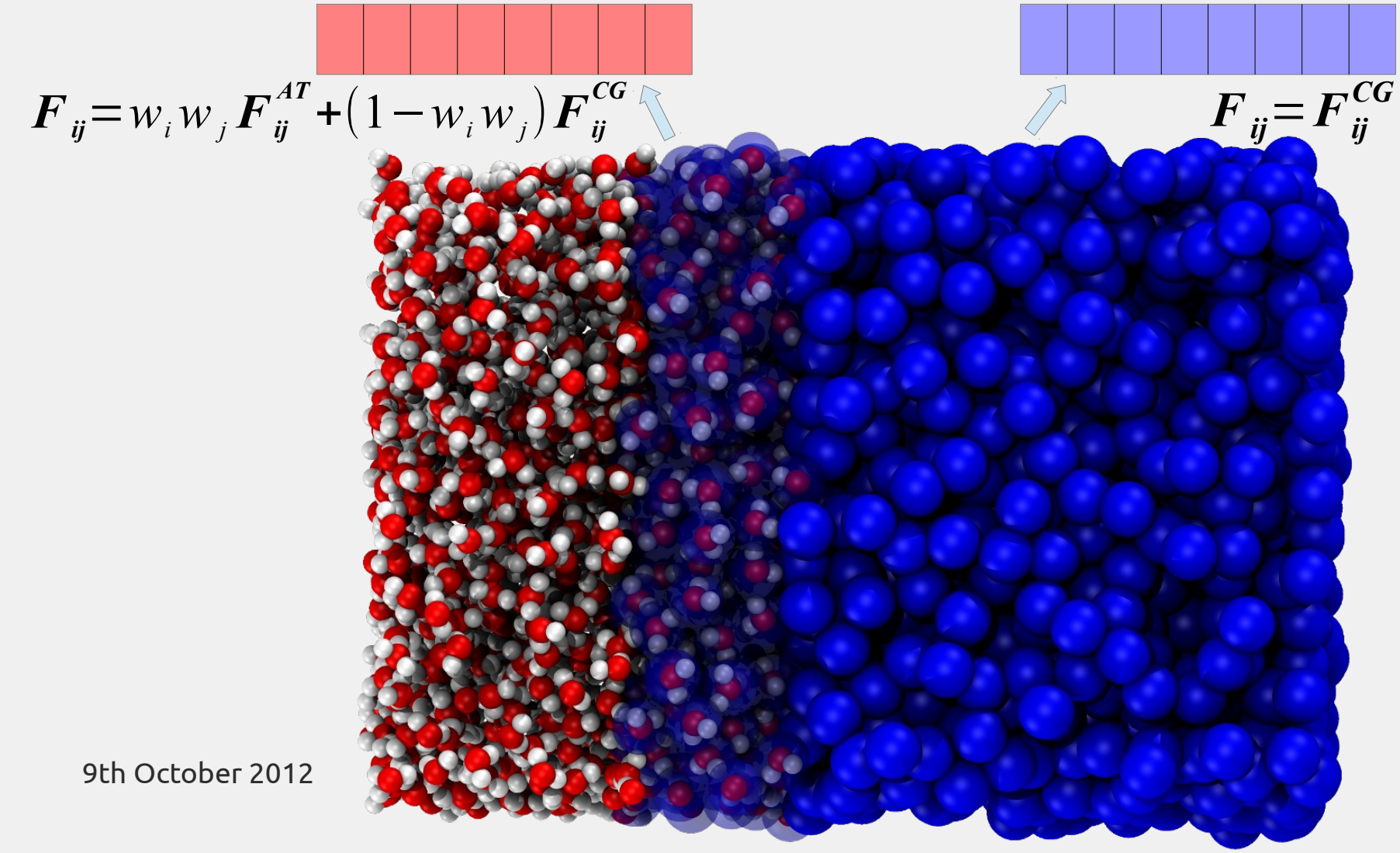
Introduction to AdResS



AdResS implementation in ESPResSo++

Implementation in ESPResSo++

- General idea: we build two Verlet lists



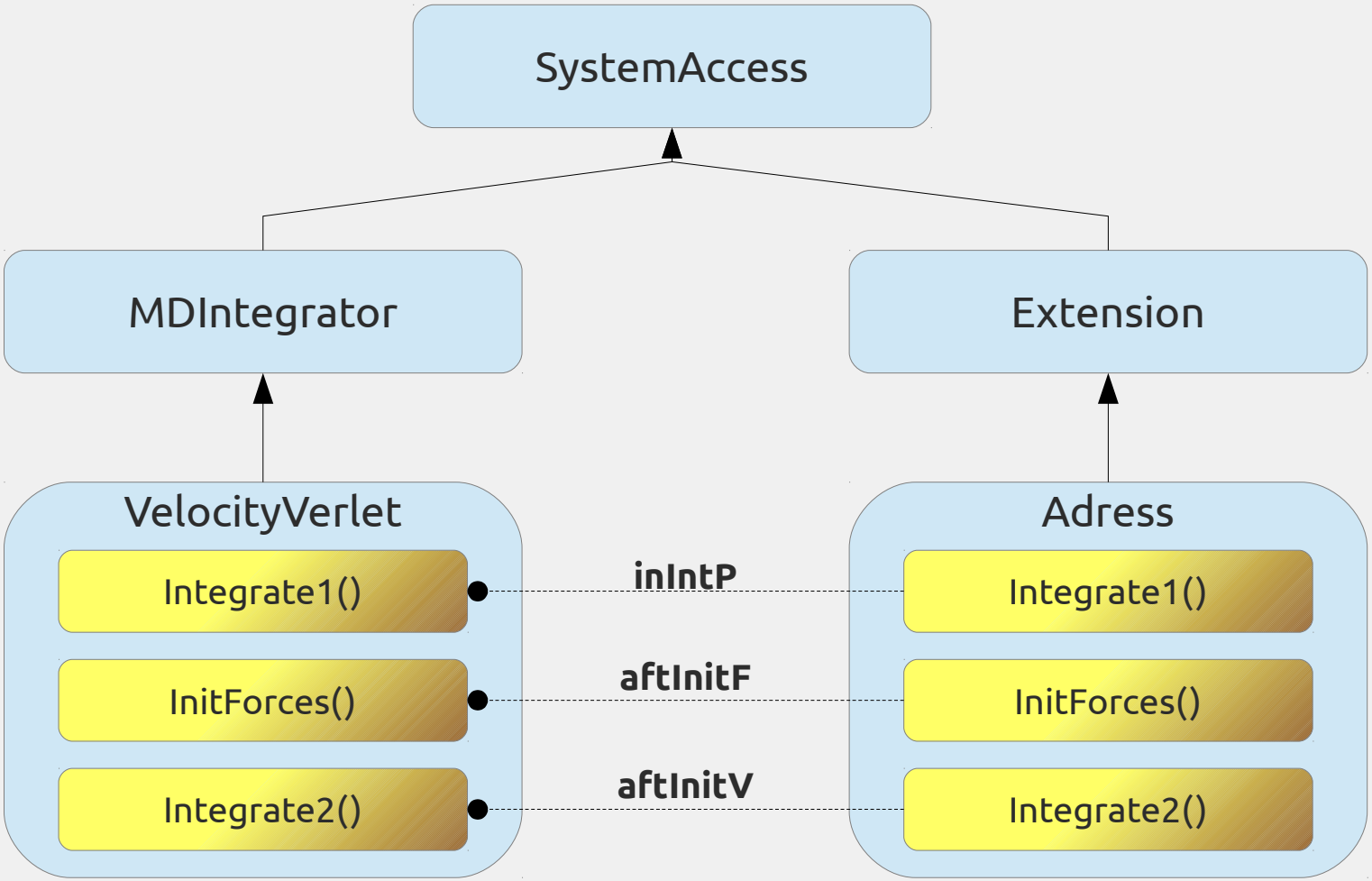
Implementation in ESPResSo++

- General idea: atomistic particles act as an additional property of the coarse grained particles
  - there is a mapping from a CG particle to a list of AT particles (via the TupleList)
    `std::map<Particle*, std::vector<Particle*> >`
  - atomistic particles are always present

Implementation in ESPResSo++

- AdResS is an extension of the integrator
  - but changes to other code and new classes were also introduced
- The new (atomistic) particles have to be integrated
  - we connect to 3 integrator signals:
    - inIntP
    - afterInitF
    - afterIntV

Implementation in ESPResSo++



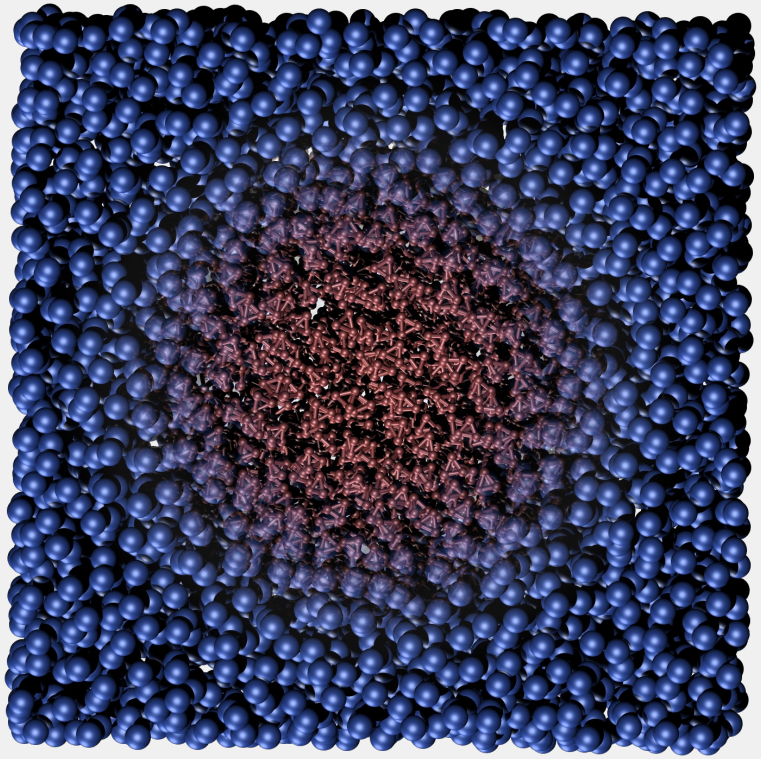
Implementation in ESPResSo++

- New classes
  - VerletListAdress
  - VerletListAdressInteractionTemplate
  - DomainDecompostionAdress
  - FixedTupleList
  - FixedPair/Triple/QuadrupleListAdress
- New functions and data structures
  - Storage
  - Particle

Implementation in ESPResSo++

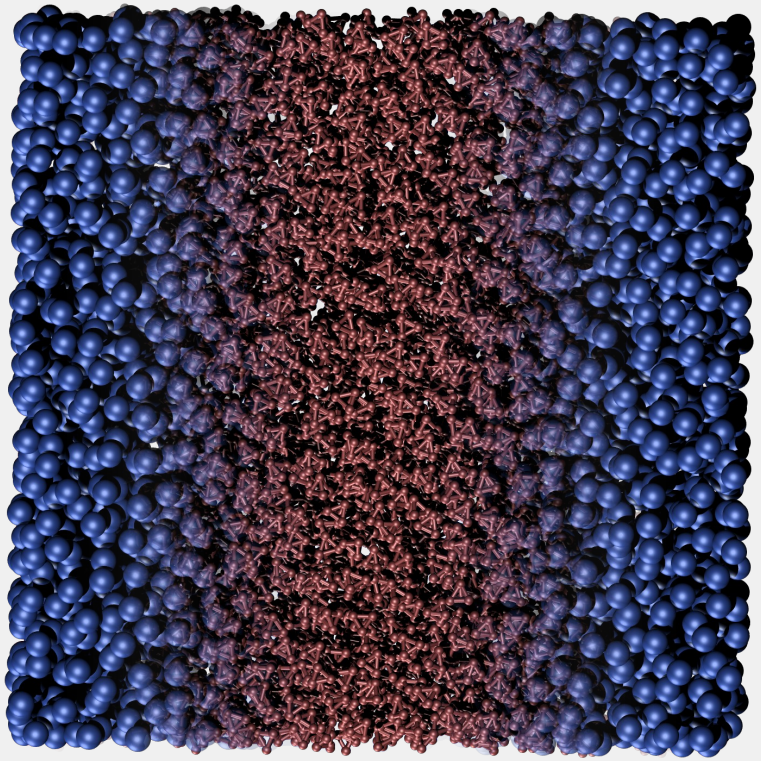
- Spherical geometry by default, but can be planar with minor code changes
- For spherical

```
Real3D dist = pos() - ref();
real distsq = dist.sqr();
```



- For planar

```
real dist = pos()[0] - ref()[0];
real distsq = dist.sqr();
```



Implementation in ESPResSo++

- What about the time?
- Is there any speedup with AdResS?

Implementation in ESPResSo++

- What about the time?
- Is there any speedup with AdResS?

- The force calculation is faster

Implementation in ESPResSo++

- What about the time?
- Is there any speedup with AdResS?

- The force calculation is faster
- But in total it is slower

Tetrahedral molecule example

Tetrahedral example

```
# AdResS domain decomposition
system.storage = espresso.storage.DomainDecompositionAdress(system, nodeGrid,
cellGrid)

# prepare AT particles
allParticlesAT = []
allParticles = []
tuples = []
for pidAT in range(num_particles):
    #print pidAT,
    allParticlesAT.append([pidAT, # add here these particles just temporarily!
                           Real3D(x[pidAT], y[pidAT], z[pidAT]),
                           Real3D(vx[pidAT], vy[pidAT], vz[pidAT]),
                           Real3D(fx[pidAT], fy[pidAT], fz[pidAT]),
                           1, 1.0, 1]) # type, mass, is AT particle
```

Tetrahedral example

```
# create CG particles from center of mass
for pidCG in range(num_particlesCG):
    cmp = [0,0,0]
    cmv = [0,0,0]
    tmptuple = [pidCG+num_particles]
    # com calculation
    for pidAT in range(4):
        pid = pidCG*4+pidAT
        tmptuple.append(pid)
        pos = (allParticlesAT[pid])[1]
        vel = (allParticlesAT[pid])[2]
        for i in range(3):
            cmp[i] += pos[i] # masses are 1.0 so we skip
multiplication
            cmv[i] += vel[i]
    for i in range(3):
        cmp[i] /= 4.0 # 4.0 is the mass of molecule
        cmv[i] /= 4.0
```

Tetrahedral example

```
# create CG particles from center of mass
for pidCG in range(num_particlesCG):

    …

    allParticles.append([pidCG+num_particles, # CG particle first!
                         Real3D(cmp[0], cmp[1], cmp[2]), # pos
                         Real3D(cmv[0], cmv[1], cmv[2]), # vel
                         Real3D(0, 0, 0), # f
                         0, 4.0, 0]) # type, mass, is not AT particle

    for pidAT in range(4):
        pid = pidCG*4+pidAT
        allParticles.append([pid, # now the AT particles can be added
                             (allParticlesAT[pid])[1], # pos
                             (allParticlesAT[pid])[2], # vel
                             (allParticlesAT[pid])[3], # f
                             (allParticlesAT[pid])[4], # type
                             (allParticlesAT[pid])[5], # mass
                             (allParticlesAT[pid])[6]]) # is AT particle

    tuples.append(tmptuple)
```

Tetrahedral example

```
# create CG particles from center of mass
for pidCG in range(num_particlesCG):

    …

    allParticles.append([pidCG+num_particles, # CG particle first!
                         Real3D(cmp[0], cmp[1], cmp[2]), # pos
                         Real3D(cmv[0], cmv[1], cmv[2]), # vel
                         Real3D(0, 0, 0), # f
                         0, 4.0, 0]) # type, mass, is not AT particle

    for pidAT in range(4):
        pid = pidCG*4+pidAT
        allParticles.append([pid, # now the AT particles can be added
                             (allParticlesAT[pid])[1], # pos
                             (allParticlesAT[pid])[2], # vel
                             (allParticlesAT[pid])[3], # f
                             (allParticlesAT[pid])[4], # type
                             (allParticlesAT[pid])[5], # mass
                             (allParticlesAT[pid])[6]]) # is AT particle

    tuples.append(tmptuple)
```

Tetrahedral example

```
# add particles
system.storage.addParticles(allParticles, "id", "pos", "v", "f", "type", "mass",
"adrat")

# add tuples
ftpl = espresso.FixedTupleList(system.storage)
ftpl.addTuples(tuples)
system.storage.setFixedTuples(ftpl)

# add bonds between AT particles
fpl = espresso.FixedPairListAdress(system.storage, ftpl)
fpl.addBonds(bonds)
```

Tetrahedral example

```
# AdResS Verlet list
vl = espresso.VerletListAdress(system, cutoff=rc+skin,
adrcut=rc+skin,
                               dEx=ex_size, dHy=hy_size,
                               adrCenter=[18.42225, 18.42225, 18.42225])

# non-bonded potentials
# LJ Capped WCA between AT and tabulated Morse between CG particles
interNB = espresso.interaction.VerletListAdressLennardJonesCapped(vl, ftpl)
potWCA  = espresso.interaction.LennardJonesCapped(epsilon=1.0, sigma=1.0, shift=True,
caprad=0.27, cutoff=rca)
potMorse = espresso.interaction.Tabulated(itype=2, filename=tabMorse, cutoff=rc) # CG
interNB.setPotentialAT(type1=1, type2=1, potential=potWCA) # AT
interNB.setPotentialCG(type1=0, type2=0, potential=potMorse) # CG
system.addInteraction(interNB)
```

Tetrahedral example

```
# bonded potentials
# FENE and LJ potential between AT particles
potFENE = espresso.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
potLJ = espresso.interaction.LennardJones(epsilon=1.0, sigma=1.0, shift=True, cutoff=rca)
interFENE = espresso.interaction.FixedPairListFENE(system, fpl, potFENE)
interLJ = espresso.interaction.FixedPairListLennardJones(system, fpl, potLJ)
system.addInteraction(interFENE)
system.addInteraction(interLJ)
```

Tetrahedral example

```
# VV integrator
integrator = espresso.integrator.VelocityVerlet(system)
integrator.dt = timestep

# add AdResS extension
adress = espresso.integrator.Adress(system)
integrator.addExtension(adress)

# add Langevin thermostat extension
langevin =
espresso.integrator.LangevinThermostat(system)
langevin.gamma = gamma
langevin.temperature = temp
langevin.adress = True # enable AdResS!
integrator.addExtension(langevin)
```

Tetrahedral example

- Things to try out
  - Run the simulation with different explicit and hybrid sizes (ex_size, hy_size)
    - Also try CG and AA
  - Compare the times it takes to finish the simulation
  - Modify the code to get 1-dimensional splitting