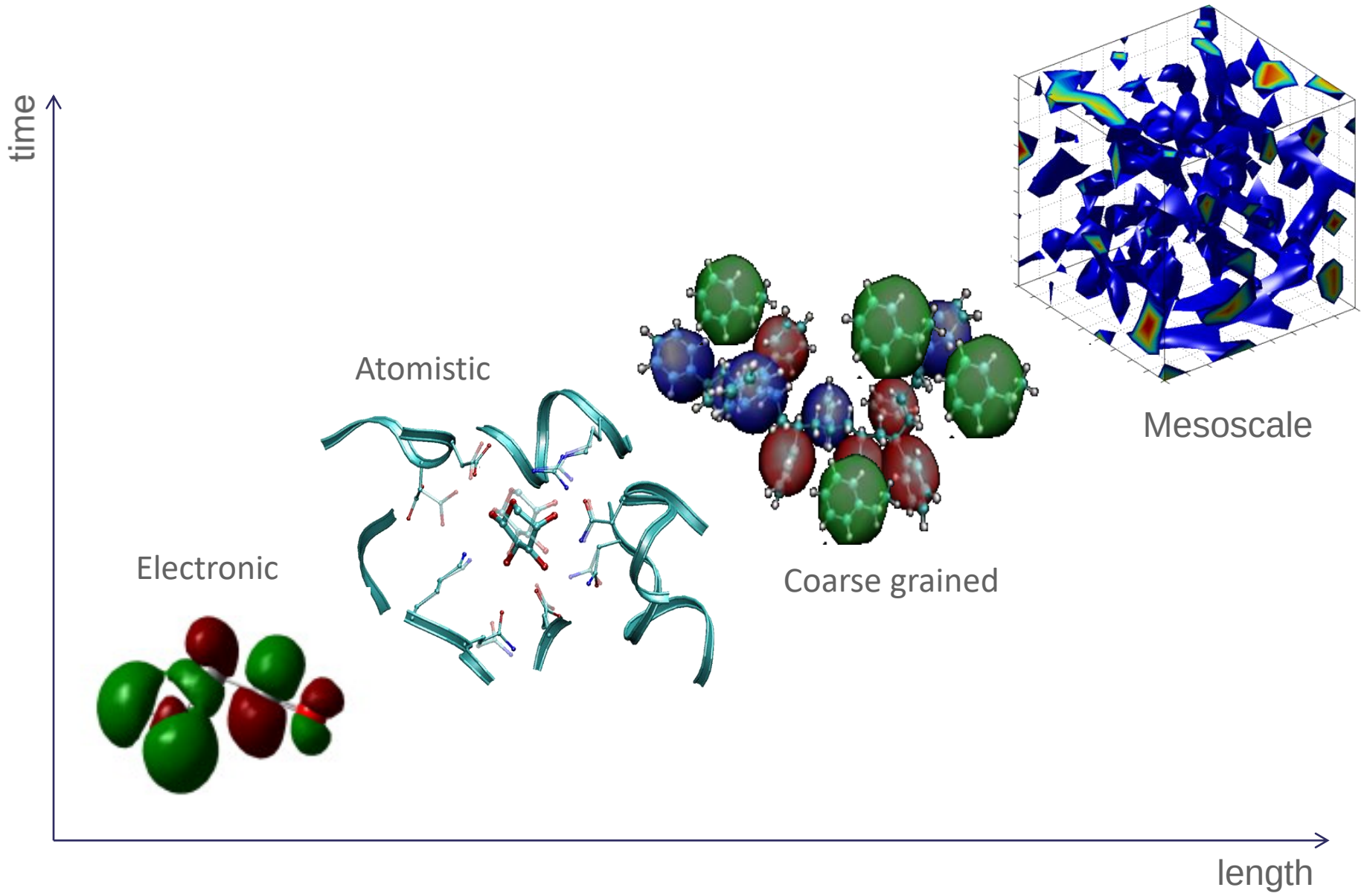
Science and Technology Facilities Council

Introduction to ChemShell

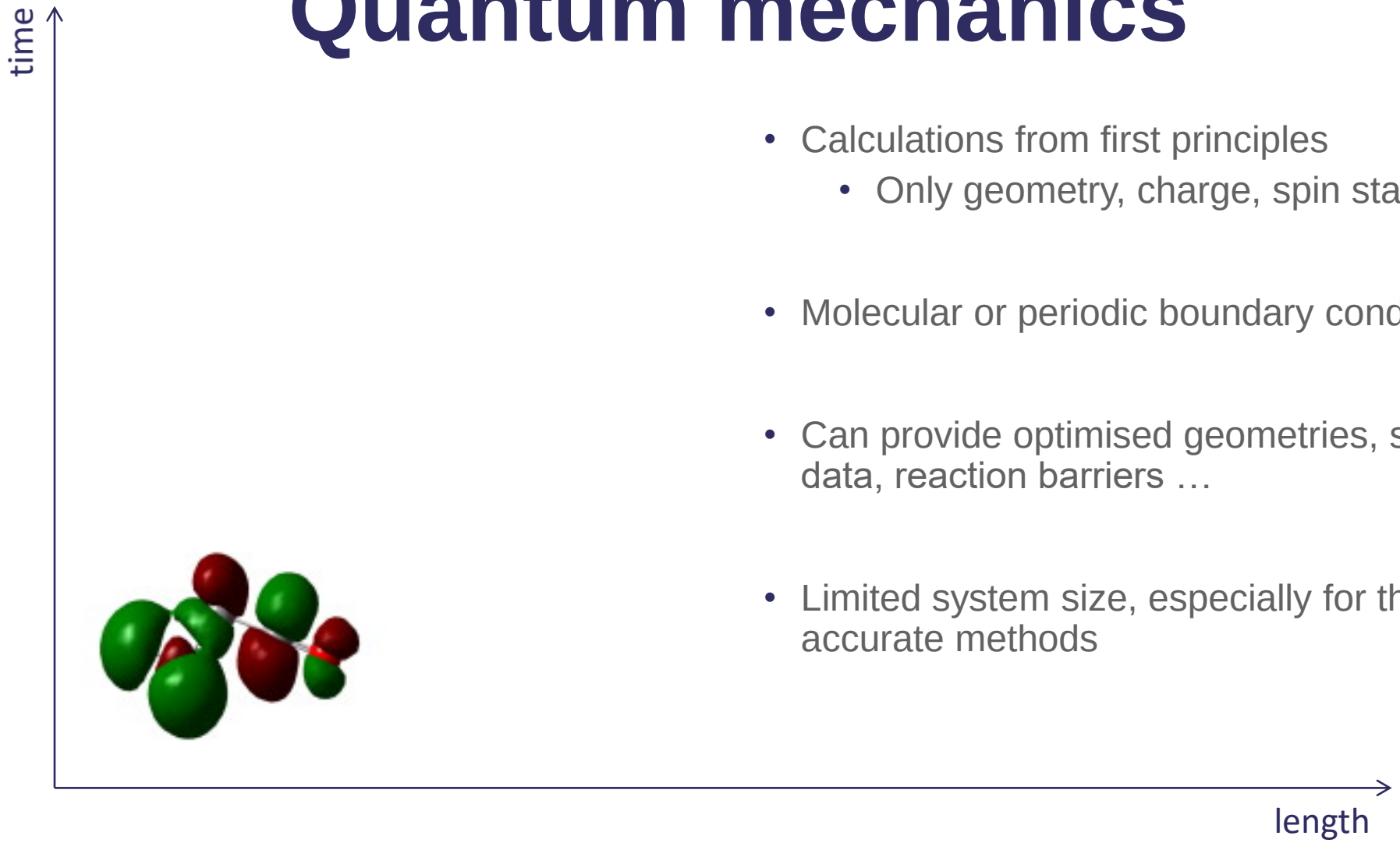
Tom Keal
STFC Daresbury Laboratory

CCPBioSim Training Week 2021
1 October 2021



Science and Technology Facilities Council

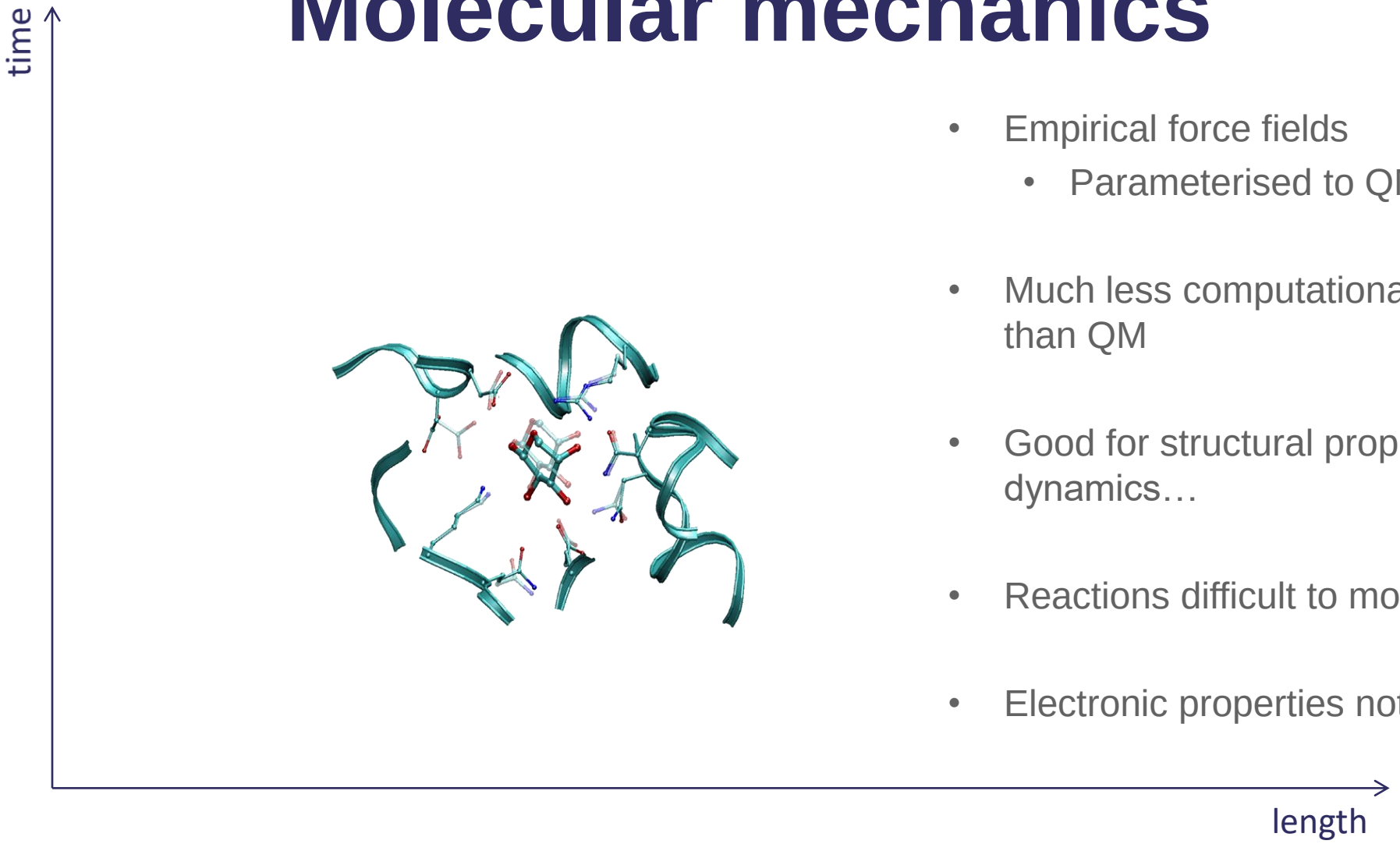
Quantum mechanics



- Calculations from first principles
  - Only geometry, charge, spin state required
- Molecular or periodic boundary conditions
- Can provide optimised geometries, spectroscopic data, reaction barriers …
- Limited system size, especially for the most accurate methods

Science and Technology Facilities Council

Molecular mechanics

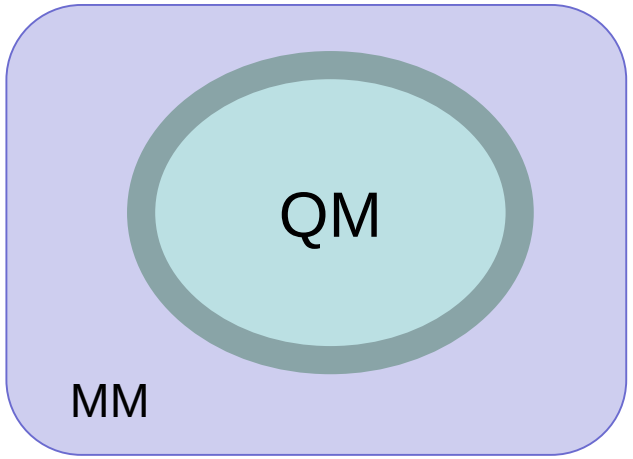


- Empirical force fields
  - Parameterised to QM results
- Much less computationally demanding than QM
- Good for structural properties, dynamics…
- Reactions difficult to model
- Electronic properties not available

Science and Technology Facilities Council

QM/MM methods

- **Overcome the QM size limit by combining quantum and classical approaches**
  - QM where an electronic description is required
  - MM for the environment





Nobel Prize for Chemistry 2013

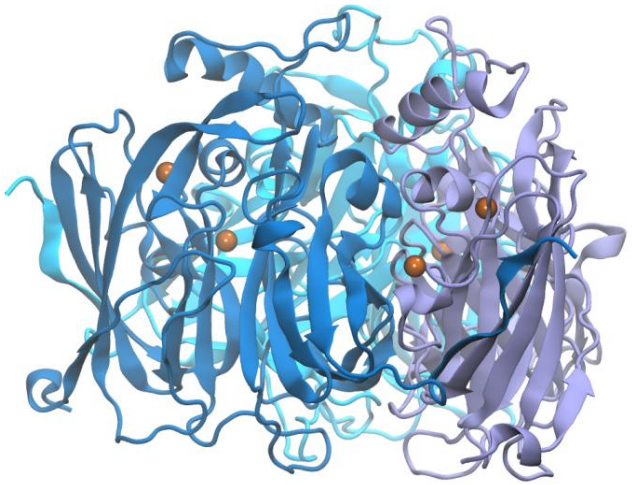
Science and Technology Facilities Council

ChemShell QM/MM package

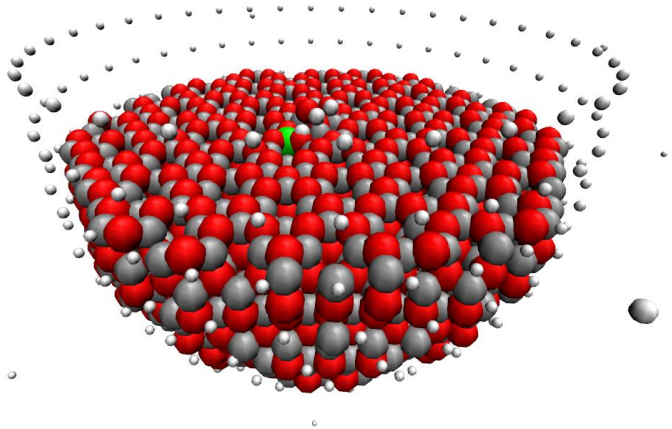
ChemShell

- Scriptable computational chemistry environment
- Flexible modular design
- Interfaces to a range of QM and MM codes

Biomolecular modelling



Materials chemistry

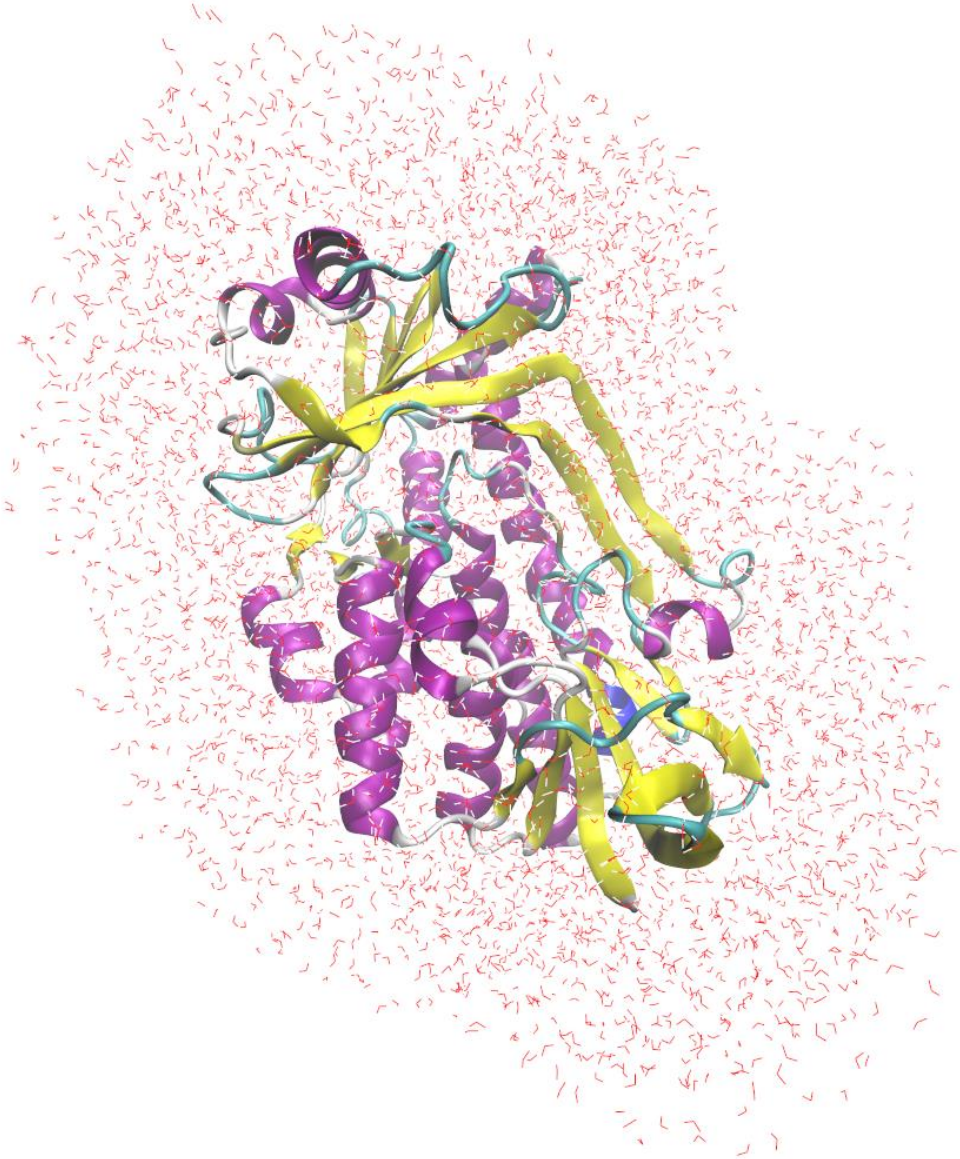


UKRI Science and Technology Facilities Council

www.chemshell.org

Biomolecular modelling with ChemShell

- **Solvation shell around protein - not periodic**
  - Use molecular QM codes
- **Internal MM module**
  - Import standard force fields
- **QM region typically up to 300 atoms**

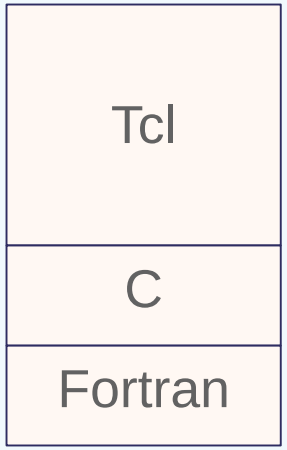


Science and Technology Facilities Council

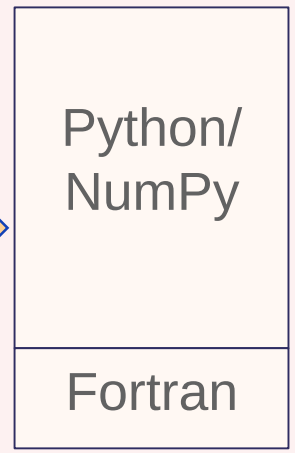
ChemShell versions

Tcl-ChemShell

- The original version (1990s onwards)
  - User interface controlled by Tcl scripts
- Proprietary licence (500 GBP for academic groups, free for UK-based groups)



Py-ChemShell



- EPSRC funded redevelopment (2014 onwards)
  - New embedding models
  - Complete rewrite of the codebase
  - New Python-based UI – easier for new users

open source™

- Reduce barriers to uptake, especially outside UK
- Encourage external developers

Science and Technology Facilities Council

JCTC 2019, 15, 1317

DOI: 10.1021/acs.jctc.8b01036

Science and Technology Facilities Council

JCTC Journal of Chemical Theory and Computation

Article

Cite This: J. Chem. Theory Comput. 2019, 15, 1317–1328

pubs.acs.org/JCTC

Open-Source, Python-Based Redevelopment of the ChemShell Multiscale QM/MM Environment

You Lu,[†] Matthew R. Farrow,[‡] Pierre Fayon,[†,⊥] Andrew J. Logsdail,[‡,§] Alexey A. Sokol,[‡] C. Richard A. Catlow,[‡,§,∥] Paul Sherwood,[†] and Thomas W. Keal*[,†]

[†]Scientific Computing Department, STFC Daresbury Laboratory, Keckwick Lane, Daresbury, Warrington WA4 4AD, United Kingdom

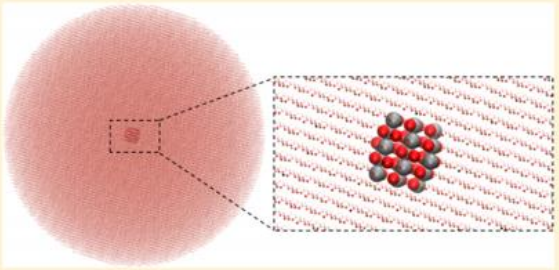
[‡]Kathleen Lonsdale Materials Chemistry, Department of Chemistry, University College London, 20 Gordon Street, London WC1H 0AJ, United Kingdom

[§]Cardiff Catalysis Institute, School of Chemistry, Cardiff University, Cardiff CF10 3AT, United Kingdom

[∥]UK Catalysis Hub, Research Complex at Harwell, STFC Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Oxon OX11 0QX, United Kingdom

Supporting Information

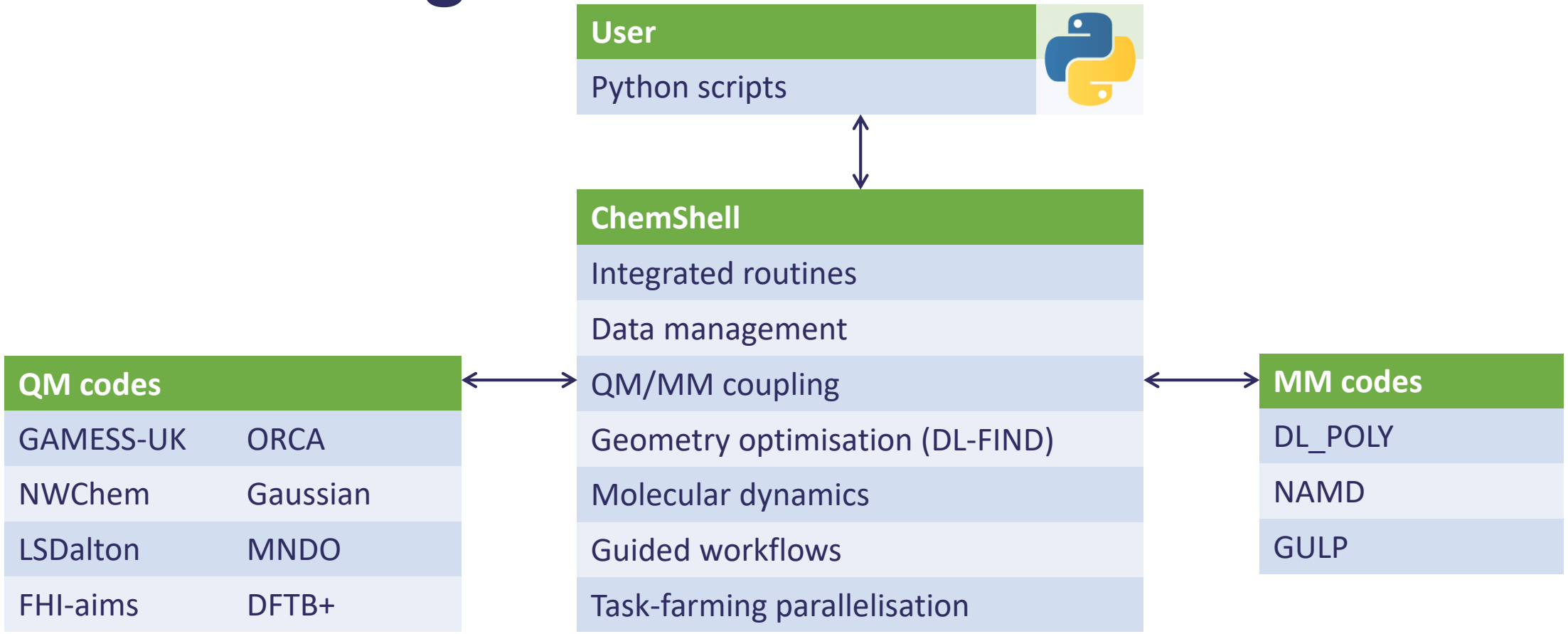
ABSTRACT: ChemShell is a scriptable computational chemistry environment with an emphasis on multiscale simulation of complex systems using combined quantum mechanical and molecular mechanical (QM/MM) methods. Motivated by a scientific need to efficiently and accurately model chemical reactions on surfaces and within microporous solids on massively parallel computing systems, we present a major redevelopment of the ChemShell code, which provides a modern platform for advanced QM/MM embedding models. The new version of ChemShell has been re-engineered from the ground up with a new QM/MM driver module, an improved parallelization framework, new interfaces to high performance QM and MM programs, and a user interface written in the Python programming language. The redeveloped package is capable of performing QM/MM calculations on systems of significantly increased size, which we illustrate with benchmarks on zirconium dioxide nanoparticles of over 160000 atoms.



Using ChemShell

Science and Technology Facilities Council

Modular design



Science and Technology Facilities Council

A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

Let's go over this line by line…

Science and Technology Facilities Council

Initialisation

```
from chemsh import *
```

- This line initialises ChemShell (chemsh) within the Python environment
- If you are unable to run ChemShell commands, it is probably because this line has been missed out

A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

ChemShell data storage

```
water = Fragment(coords="water.xyz")
```

- All data stored in ChemShell '**objects**' (Python classes)

- Object types include:
  - Fragment - molecular structure
    - Can create manually or from XYZ, PDB, JSON, etc.
    - Contains other objects with further info (point charges, shells, …)
    - Periodic systems supported (Cell object)
  - Result – contains results from a calculation (energy, gradients, Hessian matrix, …)
  - Field – data on a grid (e.g. for electrostatic potentials)

- Stored in memory unless explicitly saved (unlike Tcl-ChemShell)

A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

QM code interfaces

```
my_theory = NWChem(frag=water, method="hf", basis="3-21g")
```

- ChemShell '**theories**' are interfaces providing access to third party codes. QM interfaces include:
  - NWChem (PNNL, USA) is the main target for Py-ChemShell development
  - GAMESS-UK – developed at Daresbury
  - ORCA, good for biomolecules
  - DFTB+, fast semi-empirical DFT method

- `frag=water` – use the Fragment object defined in the previous line

- Option keywords are standardised as much as possible between interfaces:
  - `method='dft', functional='b3lyp'`
  - `charge=1, mult=2, scftype='uhf'`
  - `basis='6-31g'` (uses external code libraries)

Science and Technology Facilities Council

A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

Defining and running your calculation

```
my_calc = SP(theory=my_theory, gradients=True)
```

- SP (single point) is a ChemShell '**task**'
  - Defines the calculation that you want to perform (single point energy and gradient)
  - `my_theory` is the NWChem configuration defined on the previous line

```
my_calc.run()
```

- Execute the calculation defined above
- NWChem is now run either by a system call or as a directly-linked library

A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

Getting the result

```
print("Energy = ", my_calc.result.energy)
```

- The result object can be accessed as `my_calc.result`
  - `my_calc.result.energy` is a variable containing the energy

- Here, we simply print out the energy using Python's print command:

```
Energy = [-75.58528779]
```

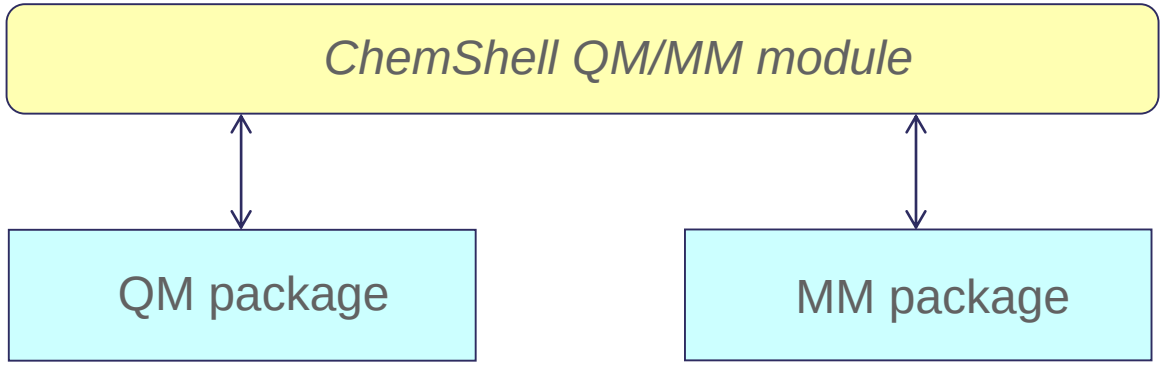
A simple (Py-)ChemShell script

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = SP(theory=my_theory, gradients=True)

my_calc.run()

print("Energy = ", my_calc.result.energy)
```

QM/MM calculations with ChemShell

Science and Technology Facilities Council

QM/MM calculations



- QM/MM calculations are controlled by the QMMM theory:

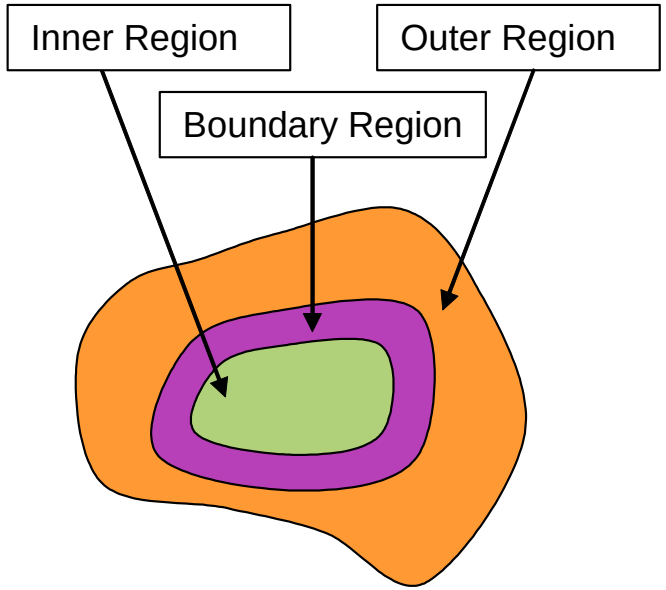
```
my_qmmm = QMMM(frag=butanol, qm_region=range(5),
               qm=NWChem(method="hf", basis="6-31g"),
               mm=GULP(ff=butanol.ff, molecule=True),
               embedding="electrostatic", coupling="covalent")

my_calc = SP(theory=my_qmmm, gradients=True)
```

Types of QM/MM

- **Additive scheme:** **E(MM, outer) + E(QM, inner) + E(QM/MM)**
  - No MM description of inner region required
  - Electrostatic coupling can be complex with link atoms
  - Can be applied to most systems e.g. solvation, enzymes, zeolites
  - Standard approach in ChemShell for biomolecules

- **Subtractive scheme:** **E(MM, full) + E(QM, inner) – E(MM, inner)**
  - Can treat polarisation of both regions at force field level
  - Full set of MM parameters required for inner region
  - Requires accurate forcefields to match potential energy surfaces
  - Eg: ONIOM method, Sauer QM-Pot (zeolites)

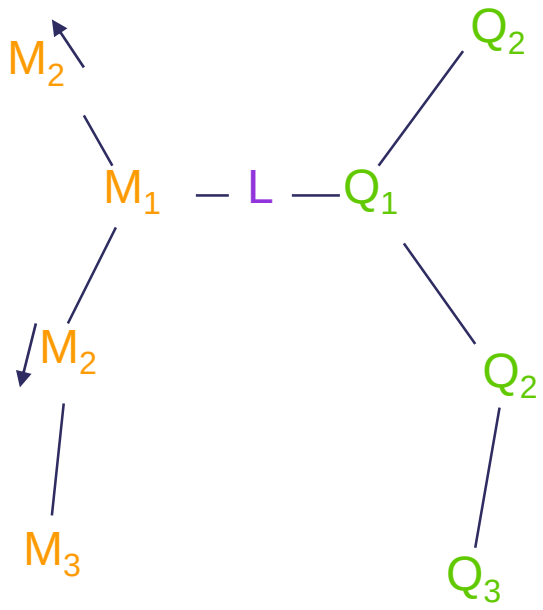
Termination of the QM region



- **Link atom scheme**
  - Terminating (link) atoms are invisible to MM calculations
    - Hydrogen, pseudo-halogen, etc.
  - For covalent systems, e.g. enzymes, zeolites
  - Choose bond to break carefully

- **Boundary region approaches**
  - Boundary atoms are present in both QM and MM calculations
  - Ionic systems: use pseudopotentials only on boundary atoms in QM calc

Link atoms charge shift boundary adjustment



- Some of the classical centres will lie close to link atom (L)
  - Artefacts can result if charge at the M_1 centre is included in Hamiltonian
- Adjustment:
  - Delete charge on M_1
  - Add an equal fraction of $q(M_1)$ to all atoms M_2
  - Add correcting dipole to M_2 sites (implemented as a pair of charges)
  - charge and dipole of classical system preserved
- Requested using `coupling='covalent'`

Science and Technology Facilities Council

Levels of QM/MM embedding

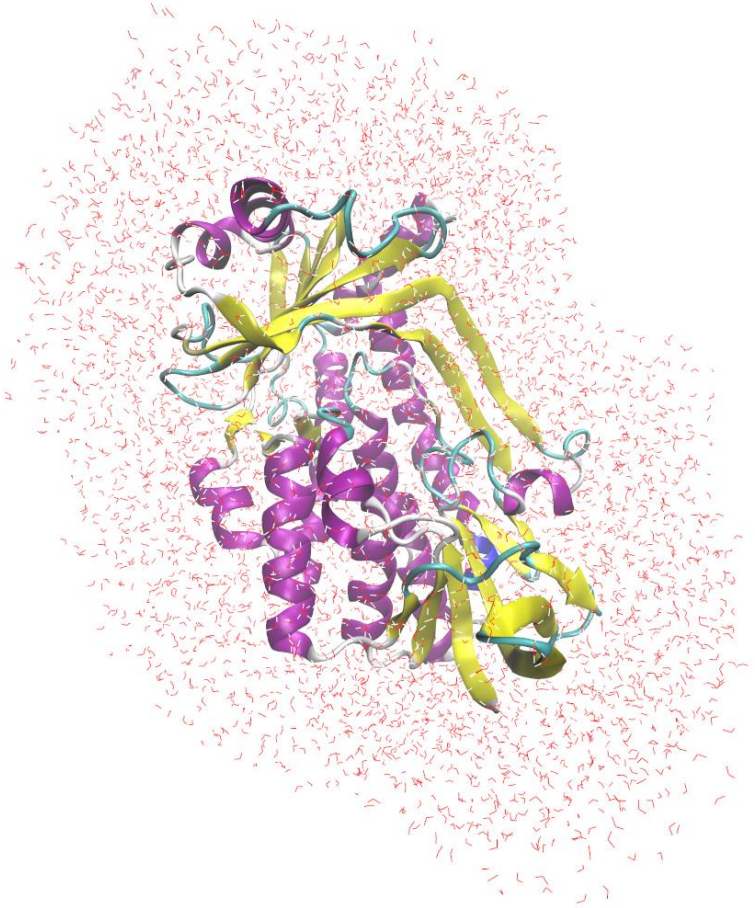
- **Mechanical embedding**
  - *in vacuo* QM calculation coupled classically to MM via point charges at QM nuclear sites
  - Straightforward, separable QM and MM calculations
  - No model for polarisation of QM region

- **Electrostatic embedding**
  - MM atoms appear as centres generating electrostatic contribution to QM Hamiltonian
  - QM calculation polarised, MM environment not polarised

- **Polarised (MM) embedding**
  - Shell model potential used in ChemShell
  - Mutual polarisation – iterated to self-consistency
  - Important for ionic systems, e.g. metal oxides

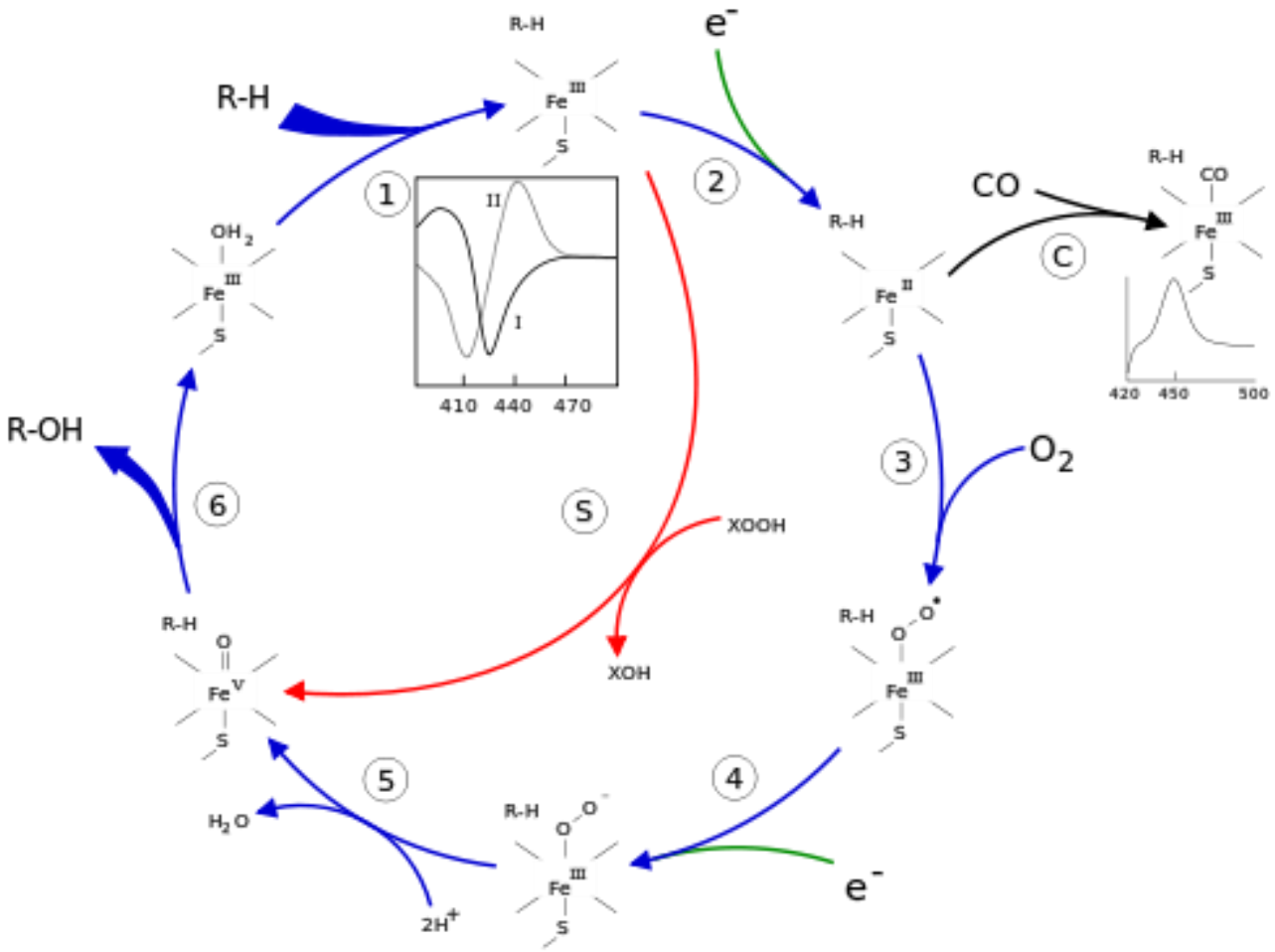
Common choices for biomolecules

- DL_POLY MM module
  - Import CHARMM or AMBER force field
- Wide variety of QM codes used
  - Almost all support electrostatic embedding
- Usual QM/MM approach:
  - Additive method
  - Electrostatic embedding
  - Link atom scheme to handle boundary



Example: cytochrome P450 family

- Extensively studied with QM/MM by a number of groups
- QM/MM used to identify species in catalytic cycle and understand their chemical properties
- Key intermediate Cpd I predicted by QM/MM before discovery

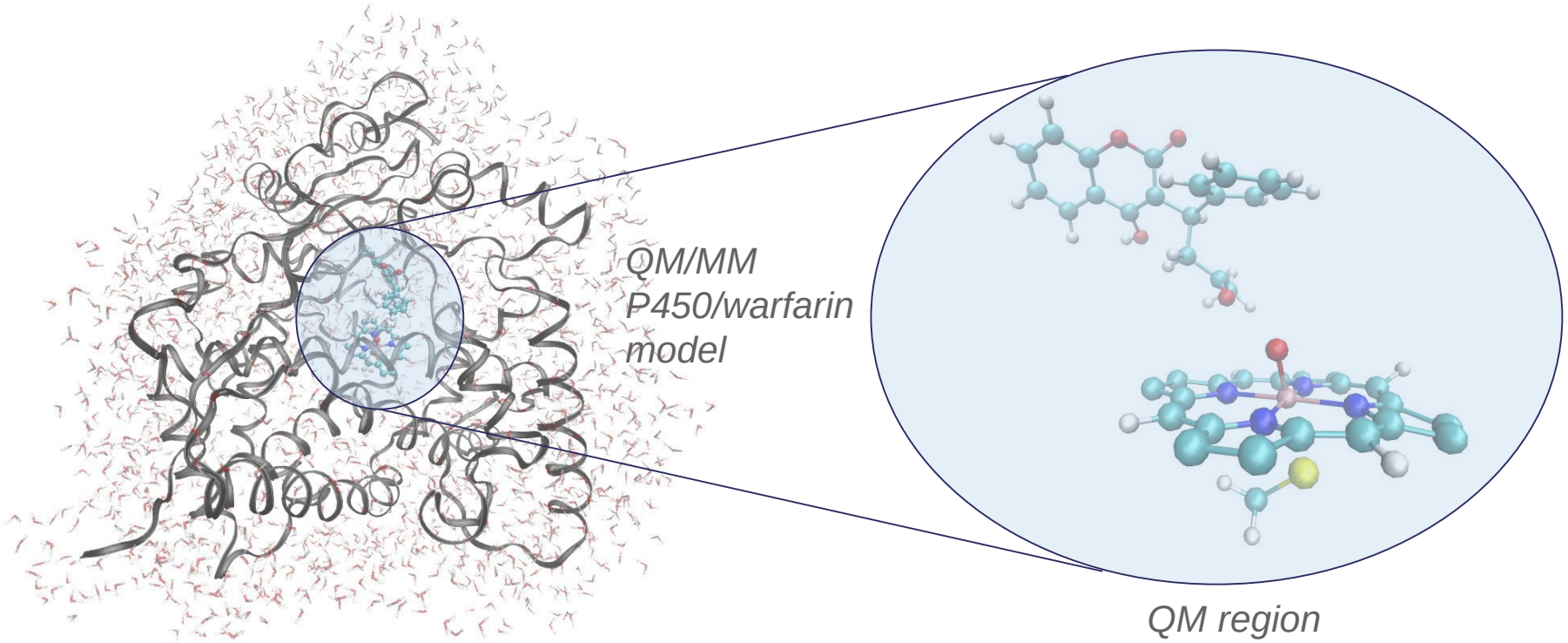


Science and Technology Facilities Council

ChemShell biomolecular modelling workflow

- Solvated and equilibrated using NAMD (see later talks)

- Take a series of “snapshots” (sampled structures from MD run)
  - Minimise using QM/MM (QM = DFT)
  - Further QM/MM optimisations to find barriers, reaction paths, …



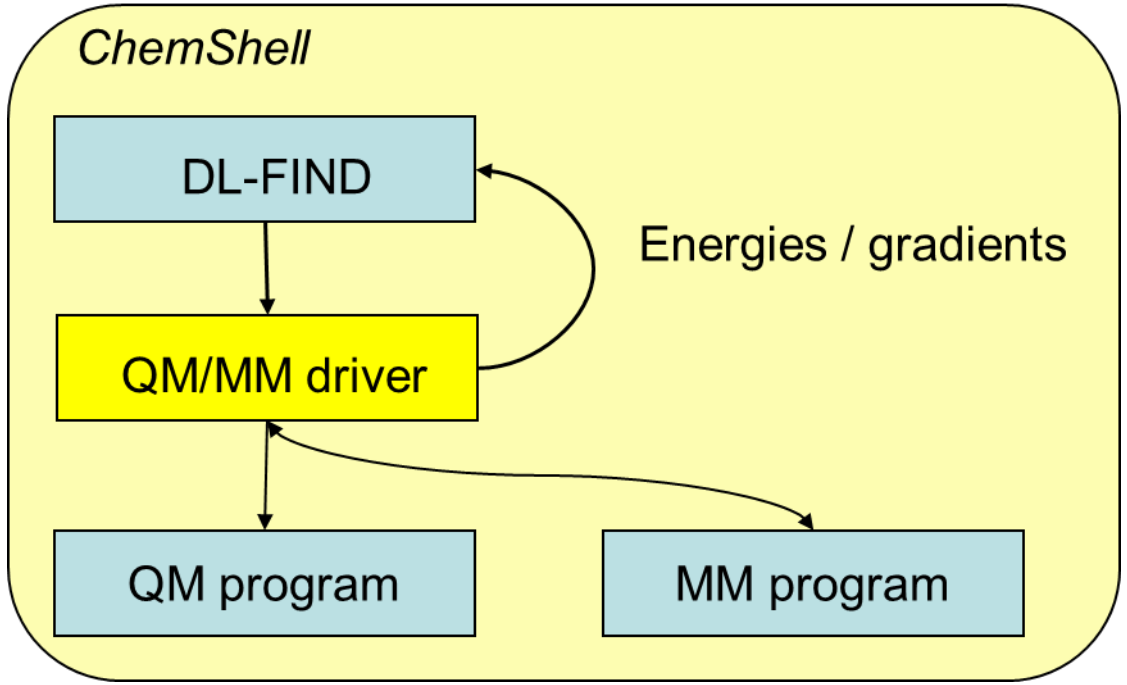
Science and Technology Facilities Council

Geometry optimisation with DL-FIND

Science and
Technology
Facilities Council

DL-FIND

- Open-source geometry optimisation library
- Interface to ChemShell for QM/MM optimisations



Science and Technology Facilities Council

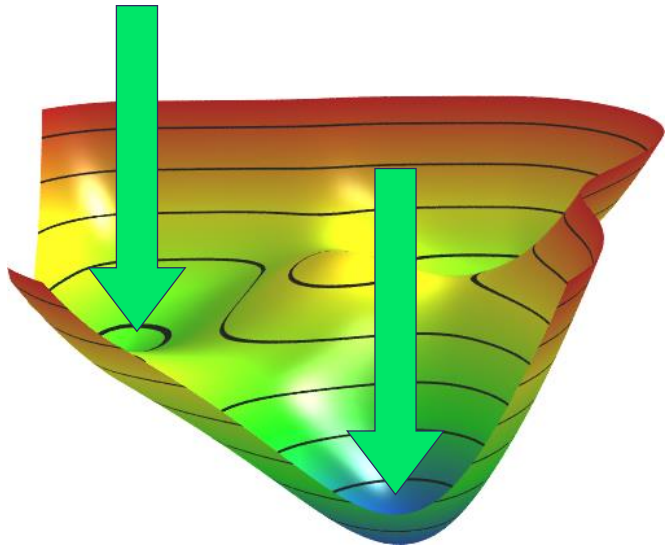
J. Kästner, J.M. Carr, T.W. Keal, W. Thiel, A. Wander, P. Sherwood, *J. Phys. Chem. A,* 2009, **113**, 11856

A simple minimisation example

```
from chemsh import *

water = Fragment(coords="water.xyz")

my_theory = NWChem(frag=water, method="hf", basis="3-21g")

my_calc = Opt(theory=my_theory)

my_calc.run()
```

- `Opt` is the ChemShell task that calls a DL-FIND optimisation
- Otherwise **identical** to the `SP` example
- Default settings used (Cartesian coordinates, 100 cycles maximum, default convergence)

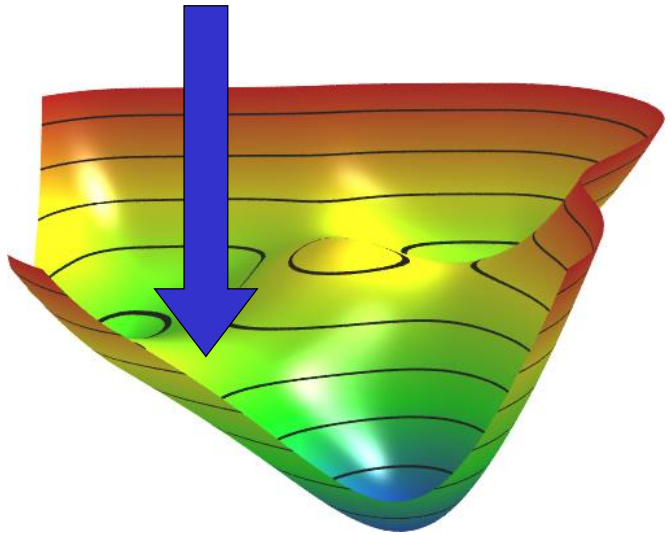
Local minimization algorithms



```
algorithm="sd"/"cg"/"nr"/"lbfgs"/"dyn"
```

- Steepest descent
- Conjugate gradient
- Newton-Raphson/quasi-Newton
  - Choice of initial Hessian approximation and update scheme
- Damped molecular dynamics
- L-BFGS (default)
  - Limited-memory variant of the BFGS quasi-Newton method
  - Full Hessian not stored, uses last *M* steps for updates
  - Suitable for large systems

Transition state optimization



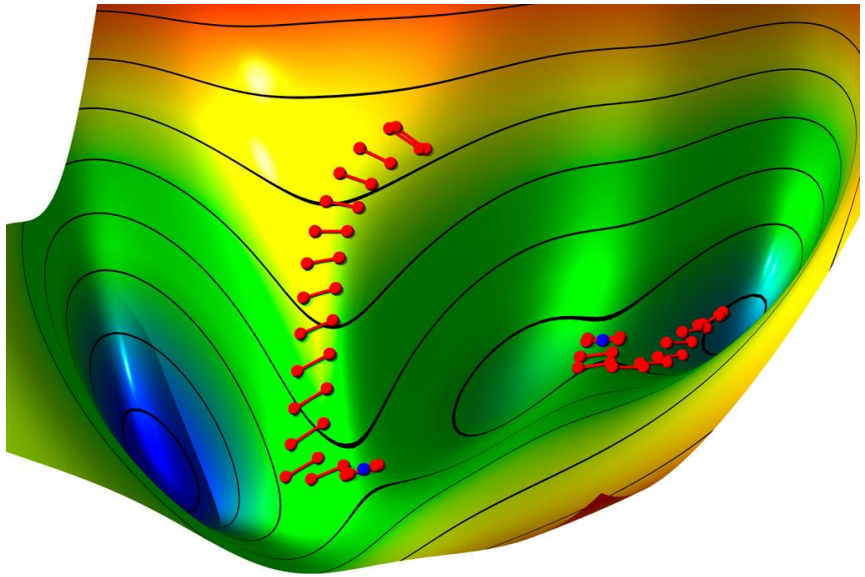
```
algorithm="prfo"
```

- Partitioned rational function optimisation (P-RFO)
  - Maximises in direction corresponding to negative eigenvalue in Hessian, minimises in all other directions.
- Requires a Hessian calculation – only suitable for small systems
- Consider more efficient alternatives!

Dimer method

```
dimer=True
```

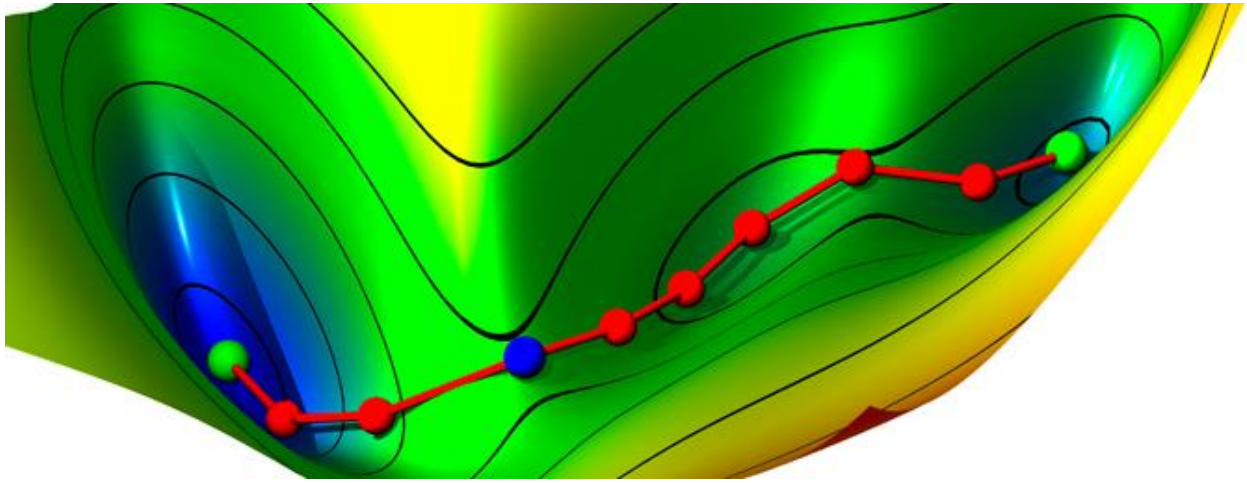
- Dimer: two images of the system, constant distance apart
- Rotation: difference of the forces
  - Aligns dimer with Hessian mode with lowest eigenvalue
- Movement:
  - In dimer direction: maximise
  - Perpendicular to dimer direction: minimise
- Converges to first-order saddle points
- **No Hessian required** so suitable for large systems



Nudged elastic band

```
neb="frozen"/"perpendicular"/"free"
```

- Multiple images, connected by spring forces
  - Specify multiple input fragments using **`coords2=`**
- Converges to the minimum energy path between endpoints
- Climbing image to find transition state
- Usually optimise endpoints first ('frozen' NEB)



- Two stage strategy:
  1. Find a loosely converged transition state with NEB
  2. Converge with dimer method

Science and Technology Facilities Council

Get the code at

www.chemshell.org

Science and Technology Facilities Council