Anchoring a CMS Ecosystem with QCSchema: Status, Uses, & Lessons Learned

LORI A. BURNS

SHERRILL GROUP, GEORGIA TECH

MOLSSI WORKSHOP ON JULIA FOR COMPUTATIONAL MOLECULAR AND MATERIALS SCIENCE, CARNEGIE MELLON UNIVERSITY

22 OCTOBER 2024

REFACTORING AN ECOSYSTEM

prefer loose coupling and high cohesion

REFACTORING AN ECOSYSTEM

prefer loose coupling and high cohesion

QC PROGRAMS

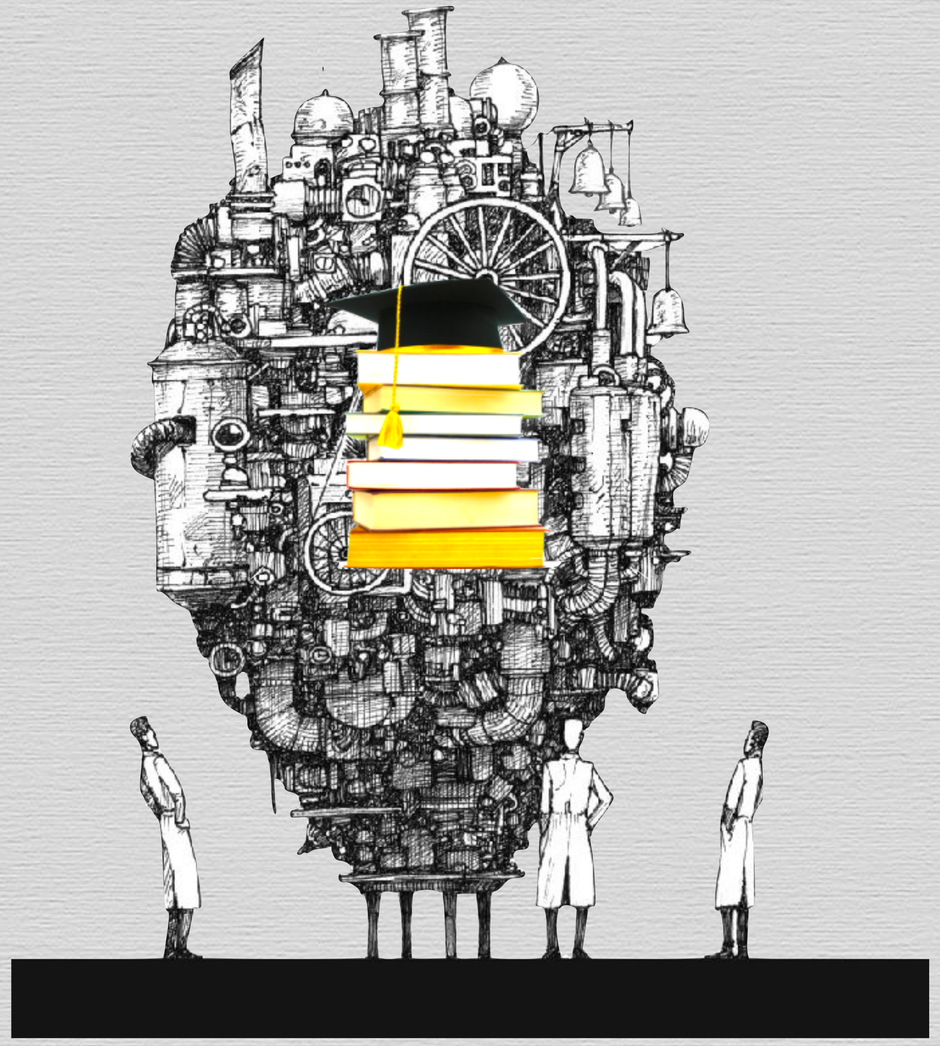
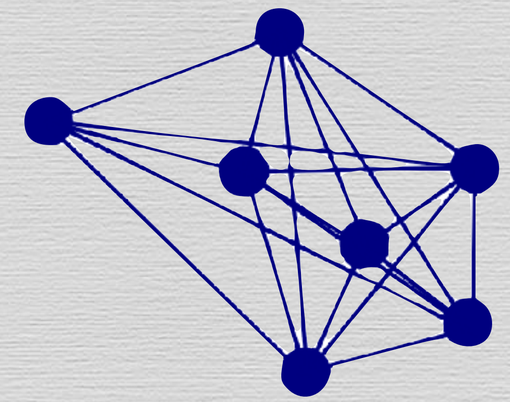


image from https://patientsafe.wordpress.com/2016/05/29/the-healthcare-safety-machine-is-broken/

REFACTORING AN ECOSYSTEM

prefer loose coupling and high cohesion

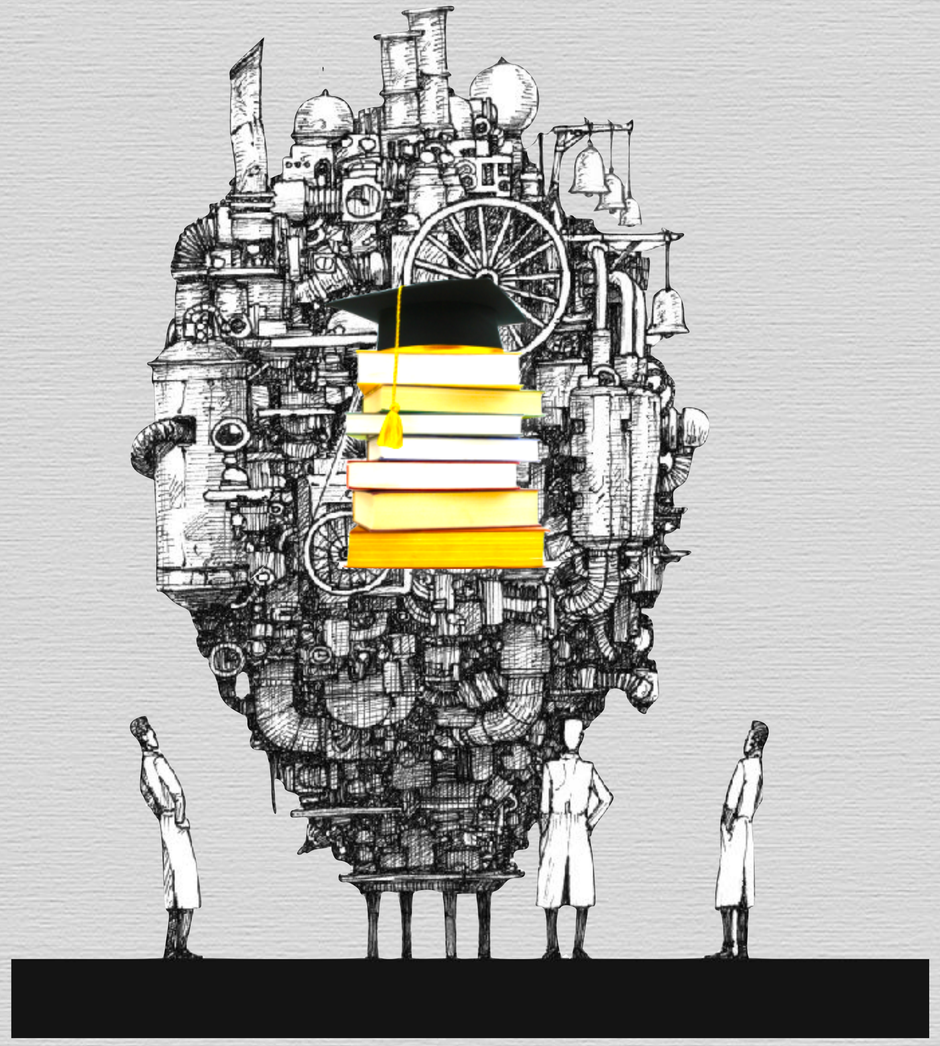
STRONG COUPLING



LOW COHESION

- difficult to maintain, test, read
- Swiss army knife modules

QC PROGRAMS

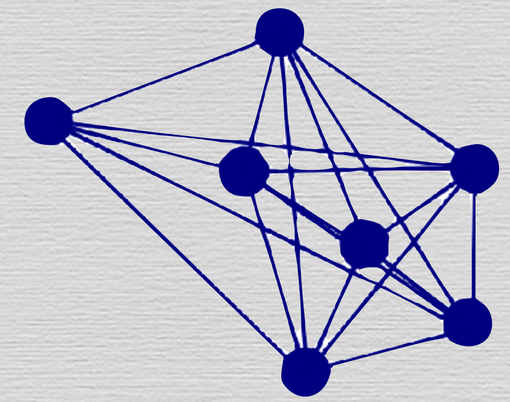


Slide LHS taken from Managing Complexity and Modular Code Development by Radovan Bast

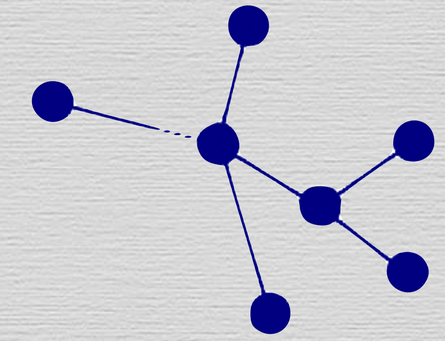
REFACTORING AN ECOSYSTEM

prefer loose coupling and high cohesion

STRONG COUPLING



LOOSE COUPLING



- easier to understand
- easier to compose

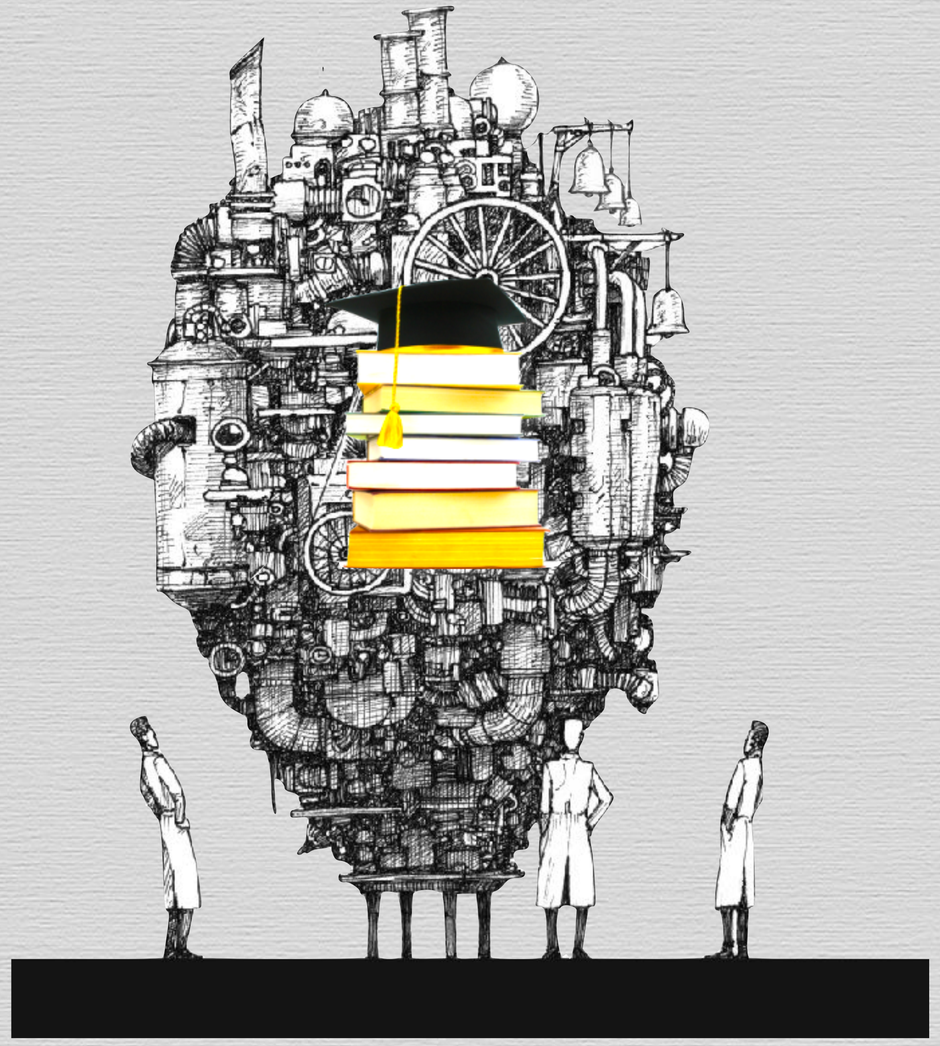
LOW COHESION

- difficult to maintain, test, read
- Swiss army knife modules

HIGH COHESION

- robust, reuseable, understandable
- stable APIs
- do one thing only and do it well (e.g., Unix command line)

QC PROGRAMS



Slide LHS taken from Managing Complexity and Modular Code Development by Radovan Bast

QC PROGRAMS

drivers of whole calcs
geometry optimization/IRC • MD driver
crystal lattice energies • torsion driver

processors of whole calcs
finite diff. deriv. • vibrational analysis
many-body decomposition

wfn processors, Molden file comm.
orbital analysis • multipole analysis
natural population analysis

user interface

workhorses
SCF methods • post-SCF methods
method support libraries

separable method components
dispersion corr. • basis set effects

post-SCF, SCF wfn comm.
DMRG/multiref. PT2 • arb.-order CC/CI
active-space CC & CR-CC • ADC mtds
GW • linear-response TDDFT

SCF-time (part of wfn), API comm.
eff. core (ECP) & frag. (EFP) potentials
relativistic contrib. • solvation corr.
polarizable embedding

tools
DFT quadrature grids • orbital opt.
symmetry analysis • k-point grid gen.

resources, API or file copy/paste
physical constants • periodic table
basis sets • norm-conserving PP

intensive libs, fast API comm.
$2e^-$ & property integrals
density functionals • grid evaluations
tensor manipulations

non-chem. intensive libs, fast API
linear algebra • binary file format
parallelism frameworks • FFT

software engineering tools
compilers • build systems
version control • testing frameworks
continuous integration services

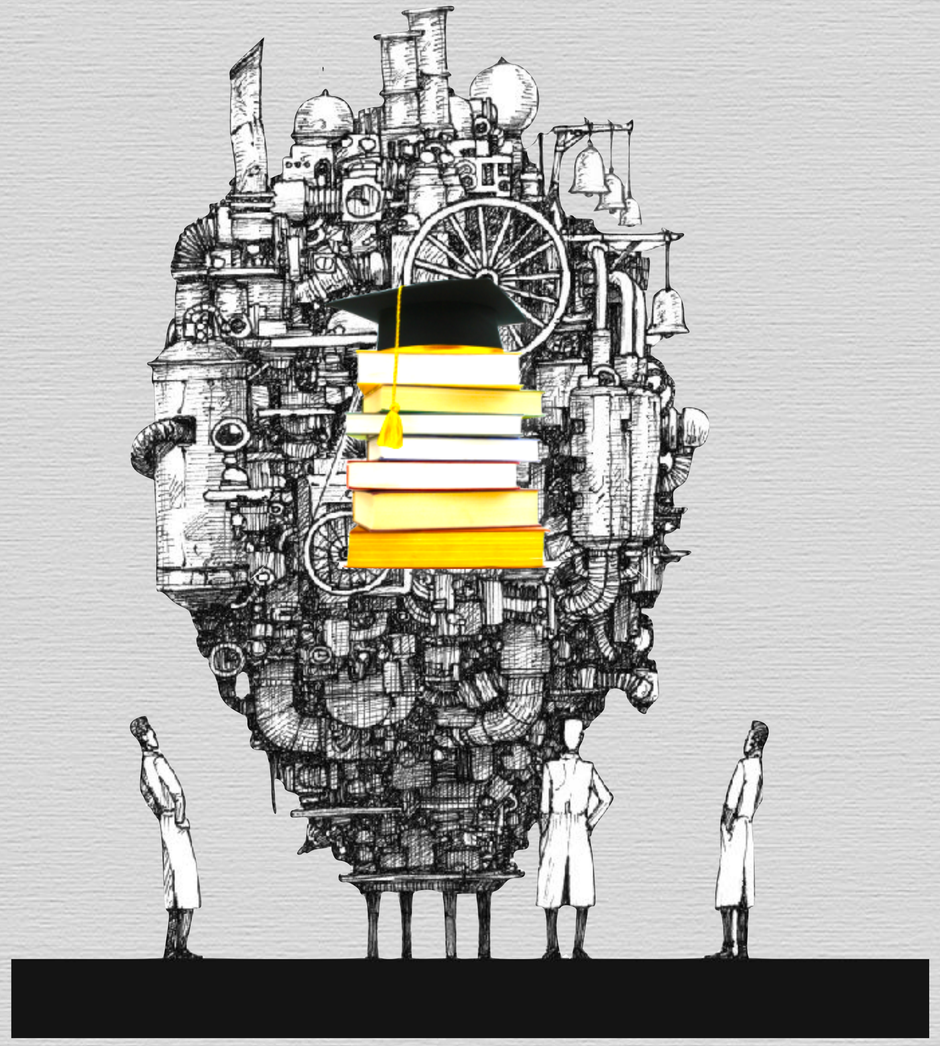


figure from Electronic Structure (2024) chapter (in press)
Blum, Windus, Burns, Oliveira, Pritchard, Slipchenko, et al.

drivers of whole calcs
geometry optimization/IRC • MD driver
crystal lattice energies • torsion driver

processors of whole calcs
finite diff. deriv. • vibrational analysis
many-body decomposition

wfn processors, Molden file comm.
orbital analysis • multipole analysis
natural population analysis

user interface

workhorses
SCF methods • post-SCF methods
method support libraries

separable method components
dispersion corr. • basis set effects

post-SCF, SCF wfn comm.
DMRG/multiref. PT2 • arb.-order CC/CI
active-space CC & CR-CC • ADC mtds
GW • linear-response TDDFT

SCF-time (part of wfn), API comm.
eff. core (ECP) & frag. (EFP) potentials
relativistic contrib. • solvation corr.
polarizable embedding

tools
DFT quadrature grids • orbital opt.
symmetry analysis • k-point grid gen.

resources, API or file copy/paste
physical constants • periodic table
basis sets • norm-conserving PP

intensive libs, fast API comm.
$2e^-$ & property integrals
density functionals • grid evaluations
tensor manipulations

non-chem. intensive libs, fast API
linear algebra • binary file format
parallelism frameworks • FFT

software engineering tools
compilers • build systems
version control • testing frameworks
continuous integration services

maximal observed monolithification

QC PROGRAMS

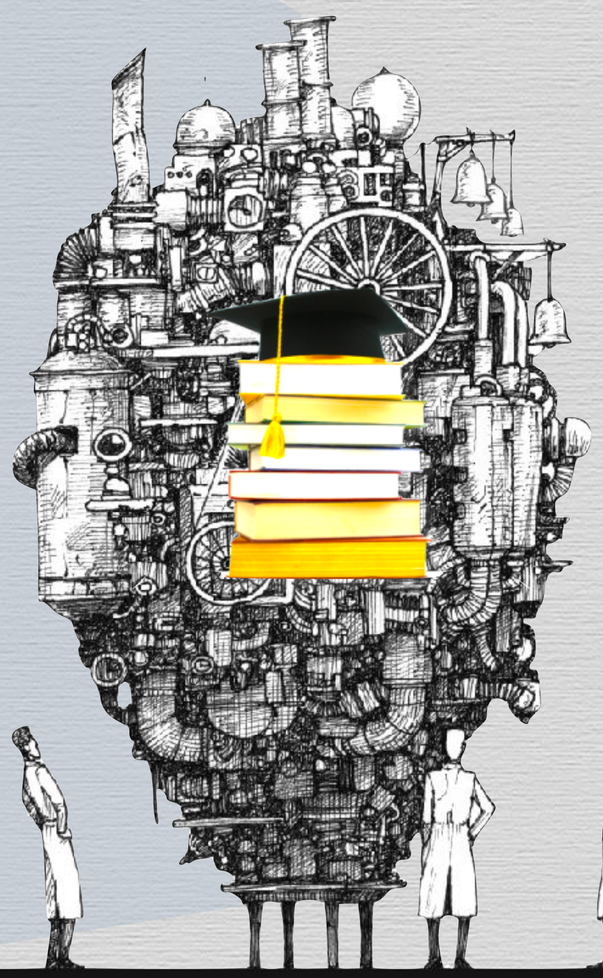


figure from Electronic Structure (2024) chapter (in press)
Blum, Windus, Burns, Oliveira, Pritchard, Slipchenko, et al.

QC PROGRAMS

drivers of whole calcs
geometry optimization/IRC • MD driver
crystal lattice energies • torsion driver

processors of whole calcs
finite diff. deriv. • vibrational analysis
many-body decomposition

wfn processors, Molden file comm.
orbital analysis • multipole analysis
natural population analysis

user interface

workhorses
SCF methods • post-SCF methods
method support libraries

separable method components
dispersion corr. • basis set effects

post-SCF, SCF wfn comm.
DMRG/multiref. PT2 • arb.-order CC/CI
active-space CC & CR-CC • ADC mtds
GW • linear-response TDDFT

SCF-time (part of wfn), API comm.
eff. core (ECP) & frag. (EFP) potentials
relativistic contrib. • solvation corr.
polarizable embedding

tools
DFT quadrature grids • orbital opt.
symmetry analysis • k-point grid gen.

resources, API or file copy/paste
physical constants • periodic table
basis sets • norm-conserving PP

intensive libs, fast API comm.
$2e^-$ & property integrals
density functionals • grid evaluations
tensor manipulations

non-chem. intensive libs, fast API
linear algebra • binary file format
parallelism frameworks • FFT

software engineering tools
compilers • build systems
version control • testing frameworks
continuous integration services

maximal observed monolithification

minimal desired monolithification

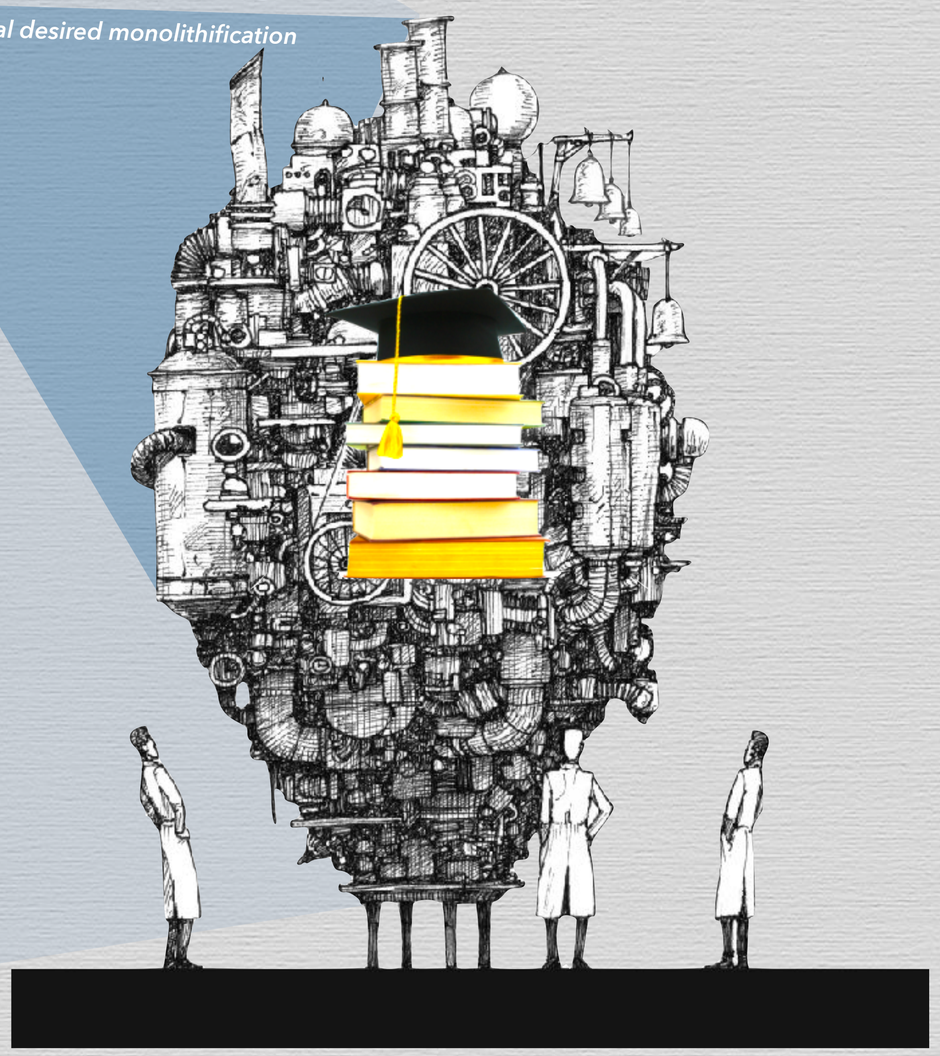
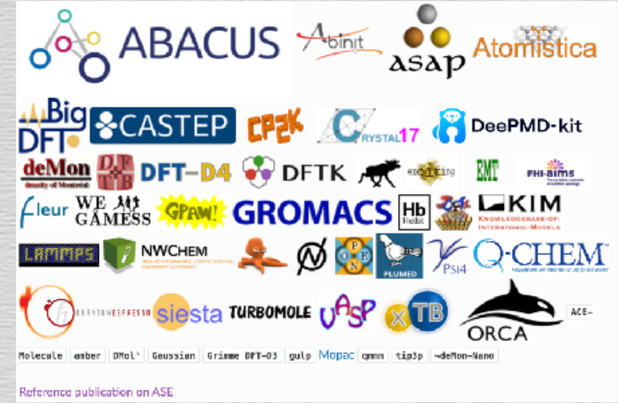
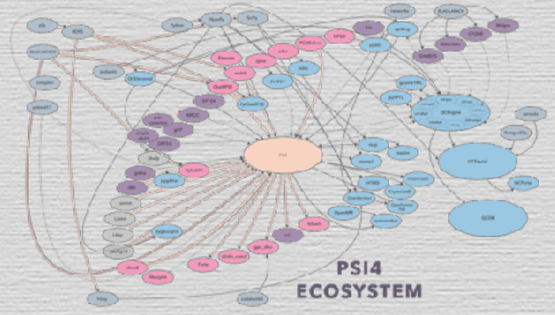


figure from Electronic Structure (2024) chapter (in press)
Blum, Windus, Burns, Oliveira, Pritchard, Slipchenko, et al.





drivers of whole calcs
geometry optimization/IRC • MD driver
crystal lattice energies • torsion driver

processors of whole calcs
finite diff. deriv. • vibrational analysis
many-body decomposition

wfn processors, Molden file comm.
orbital analysis • multipole analysis
natural population analysis

user interface

workhorses
SCF methods • post-SCF methods
method support libraries

separable method components
dispersion corr. • basis set effects

post-SCF, SCF wfn comm.
DMRG/multiref. PT2 • arb.-order CC/CI
active-space CC & CR-CC • ADC mtds
GW • linear-response TDDFT

SCF-time (part of wfn), API comm.
eff. core (ECP) & frag. (EFP) potentials
relativistic contrib. • solvation corr.
polarizable embedding

tools
DFT quadrature grids • orbital opt.
symmetry analysis • k-point grid gen.

resources, API or file copy/paste
physical constants • periodic table
basis sets • norm-conserving PP

intensive libs, fast API comm.
$2e^-$ & property integrals
density functionals • grid evaluations
tensor manipulations

non-chem. intensive libs, fast API
linear algebra • binary file format
parallelism frameworks • FFT

software engineering tools
compilers • build systems
version control • testing frameworks
continuous integration services

maximal observed monolithification

minimal desired monolithification

QC PROGRAMS

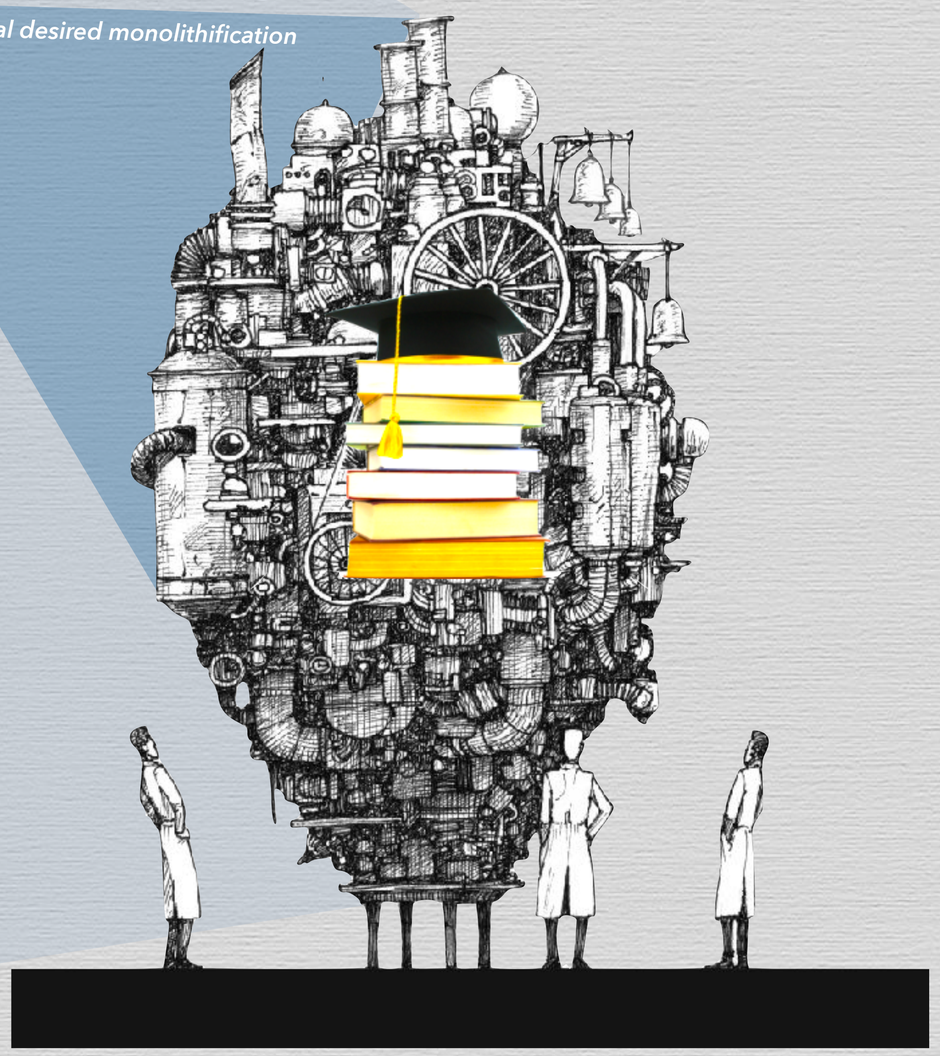
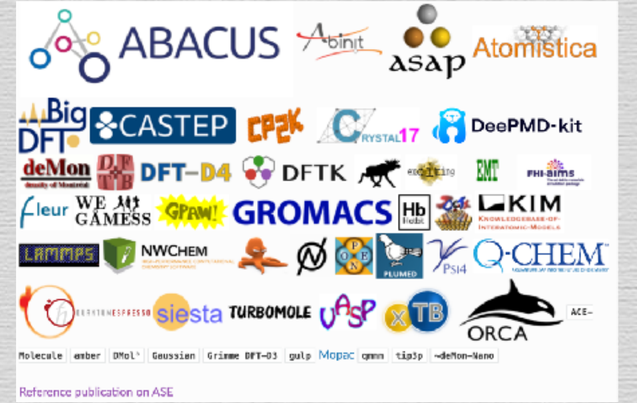
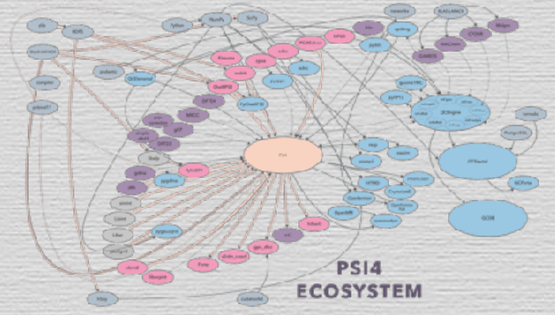


figure from Electronic Structure (2024) chapter (in press)
Blum, Windus, Burns, Oliveira, Pritchard, Slipchenko, et al.

QC PROGRAMS





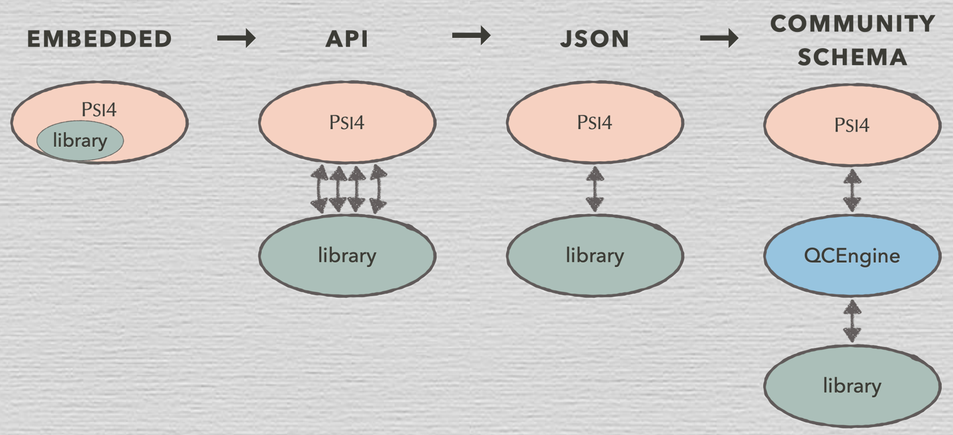


figure from Electronic Structure (2024) chapter (in press)
Blum, Windus, Burns, Oliveira, Pritchard, Slipchenko, et al.

drivers of whole calcs
geometry optimization/IRC • MD driver
crystal lattice energies • torsion driver

processors of whole calcs
finite diff. deriv. • vibrational analysis
many-body decomposition

wfn processors, Molden file comm.
orbital analysis • multipole analysis
natural population analysis

user interface

workhorses
SCF methods • post-SCF methods
method support libraries

separable method components
dispersion corr. • basis set effects

post-SCF, SCF wfn comm.
DMRG/multiref. PT2 • arb.-order CC/CI
active-space CC & CR-CC • ADC mtds
GW • linear-response TDDFT

SCF-time (part of wfn), API comm.
eff. core (ECP) & frag. (EFP) potentials
relativistic contrib. • solvation corr.
polarizable embedding

tools
DFT quadrature grids • orbital opt.
symmetry analysis • k-point grid gen.

resources, API or file copy/paste
physical constants • periodic table
basis sets • norm-conserving PP

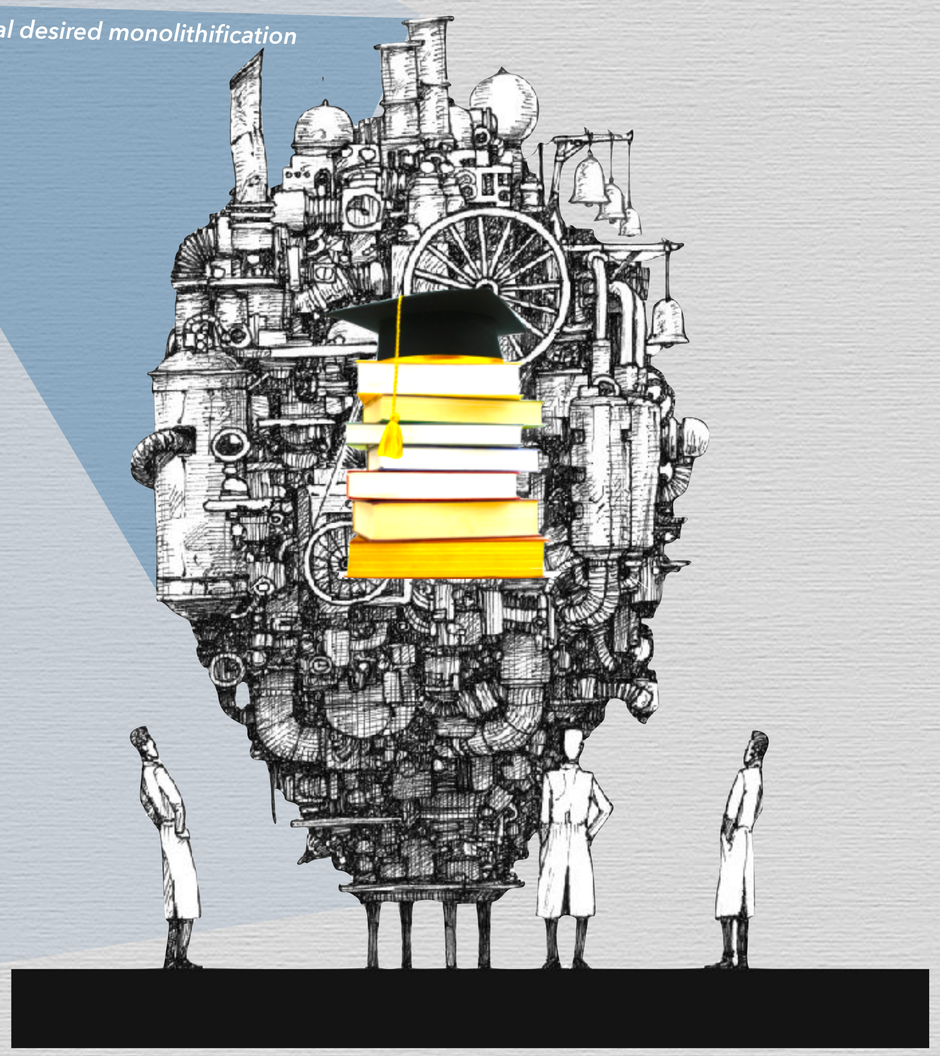
intensive libs, fast API comm.
$2e^-$ & property integrals
density functionals • grid evaluations
tensor manipulations

non-chem. intensive libs, fast API
linear algebra • binary file format
parallelism frameworks • FFT

software engineering tools
compilers • build systems
version control • testing frameworks
continuous integration services

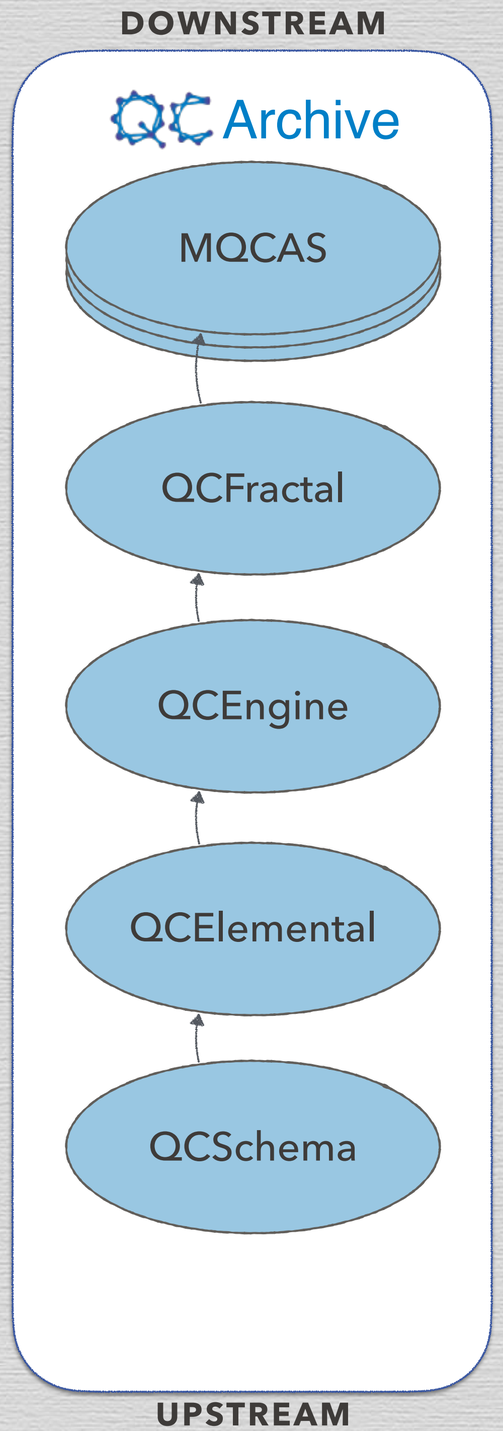
maximal observed monolithification

minimal desired monolithification



QCArchive Stack

qcarchive.molssi.org/



- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

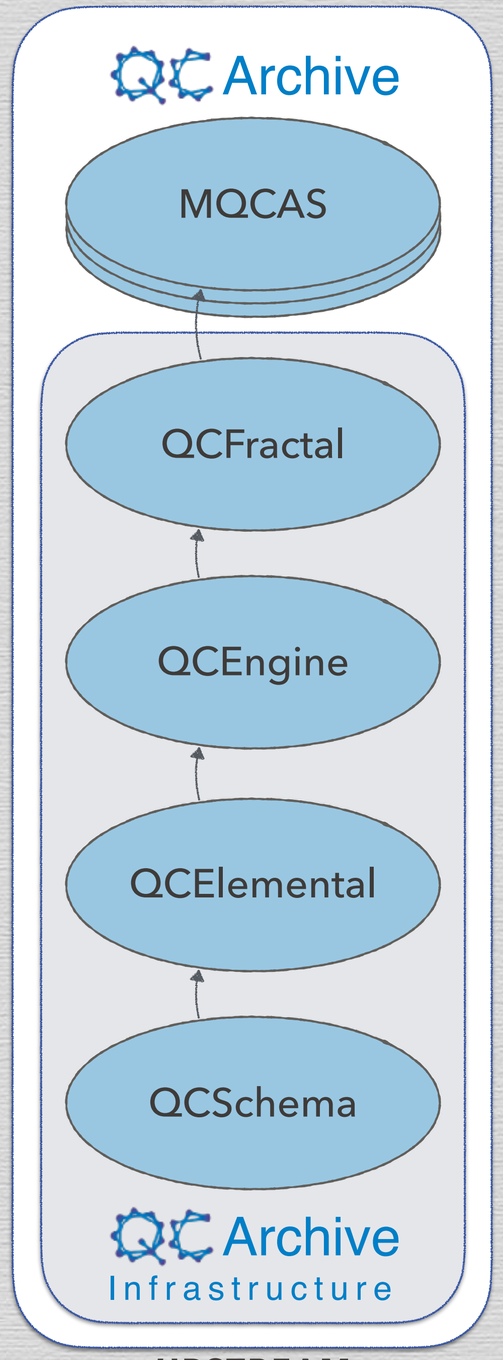
- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

QCArchive Stack

qcarchive.molssi.org/

DOWNSTREAM



UPSTREAM

- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

github.com/MolSSI/QCFractal

pip install qcfractal

conda install qcfractal -c conda-forge

github.com/MolSSI/QCEngine

pip install qcengine

conda install qcengine -c conda-forge

github.com/MolSSI/QCElemental

pip install qcelemental

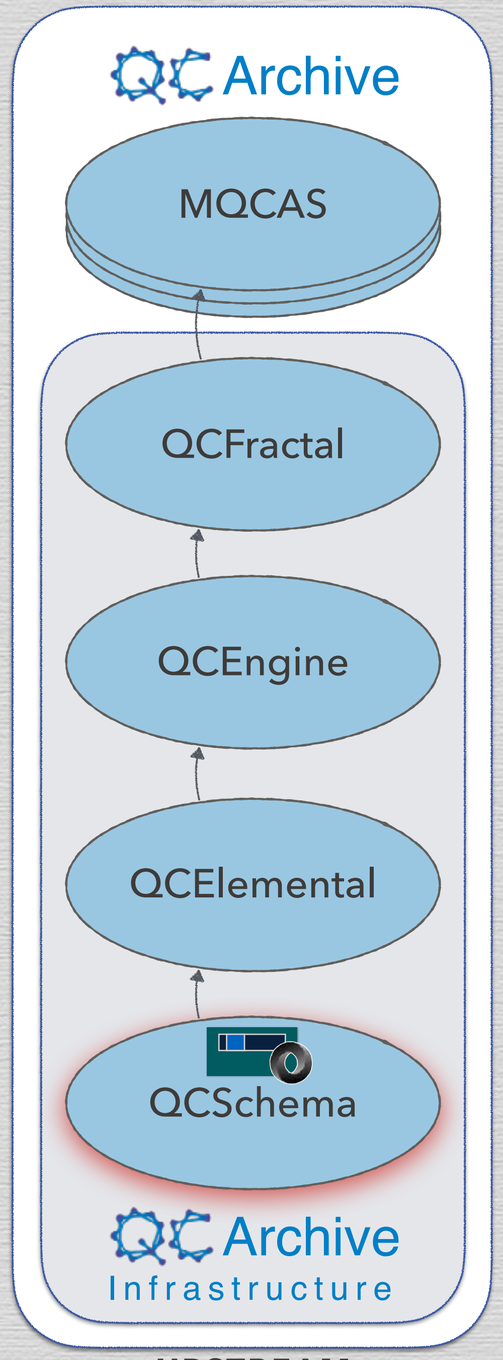
conda install qcelemental -c conda-forge

github.com/MolSSI/QCSchema

QCArchive STACK

qcarchive.molssi.org/

DOWNSTREAM



UPSTREAM

- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

github.com/MolSSI/QCFractal

pip install qcfractal

conda install qcfractal -c conda-forge

github.com/MolSSI/QCEngine

pip install qcengine

conda install qcengine -c conda-forge

github.com/MolSSI/QCElemental

pip install qcelemental

conda install qcelemental -c conda-forge

github.com/MolSSI/QCSchema

Don't visit this repository, as it's out-of-date.

QUANTUM CHEMISTRY SCHEMA

github.com/MolSSI/QCSchema

- **COMMUNICATION** channel between all pieces of the ecosystem.
- **COMMUNITY** project useful for many aspects of quantum chem.
- **JSON** nominally, but any key/value/array language like MessagePack/BSON/XML/YAML ok.
- **TIMELINE** stable since 2019. Most changes for QCA. More community participation desired but little momentum. v2 soon.
- **SCHEMA** defined in Python with Pydantic in QCElemental module, then exported to JSON.

QUANTUM CHEMISTRY SCHEMA

github.com/MolSSI/QCSchema

- **COMMUNICATION** channel between all pieces of the ecosystem.
- **COMMUNITY** project useful for many aspects of quantum chem.
- **JSON** nominally, but any key/value/array language like MessagePack/BSON/XML/YAML ok.
- **TIMELINE** stable since 2019. Most changes for QCA. More community participation desired but little momentum. v2 soon.
- **SCHEMA** defined in Python with Pydantic in QCElemental module, then exported to JSON.

Molecule

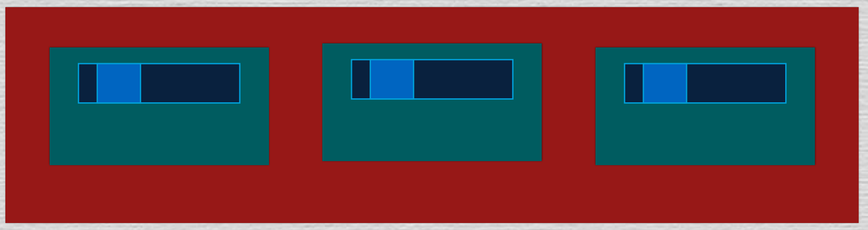
```
{
   "geometry": [0, 0, 0, 0, 0, 1],
   "symbols": ["He", "He"],
    ...
}
```

QUANTUM CHEMISTRY SCHEMA

github.com/MolSSI/QCSchema

- **COMMUNICATION** channel between all pieces of the ecosystem.
- **COMMUNITY** project useful for many aspects of quantum chem.
- **JSON** nominally, but any key/value/array language like MessagePack/BSON/XML/YAML ok.
- **TIMELINE** stable since 2019. Most changes for QCA. More community participation desired but little momentum. v2 soon.
- **SCHEMA** defined in Python with Pydantic in QCElemental module, then exported to JSON.

OptimizationResult



Molecule

```
{
    "geometry": [0, 0, 0, 0, 0, 1],
    "symbols": ["He", "He"],
     ...
}
```

AtomicInput

```
{
    "molecule": {
        "geometry": [0, 0, 0, 0, 0, 1],
        "symbols": ["He", "He"]
    },
    "driver": "energy",
    "model": {
        "method": "CCSD(T)",
        "basis": "aug-cc-pVDZ",
    },
    "keywords": {},
}
```

AtomicResult

```
{
    ... AtomicInput ...
    "provenance": {
        "creator": "My QM Program",
        "version": "1.1rc1",
    },
    "properties": {
        "calcinfo_nalpha": 5,
        "scf_total_energy": -5.433191881443323,
        "nuclear_repulsion_energy": 2.11670883436,
        "scf_iterations": 8.0,
    },
    "error": null,
    "return_result": -6.5432123456,
    "success": true,
    "stdout": "...",
    "wavefunction": "..."
}
```

QCSchema Molecule

Cartesian, AU-based topology specification

Molecule (abr.)

schema_version: 2 *(used w/QCSk v1)*
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.

QCSchema Molecule

Cartesian, AU-based topology specification

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info

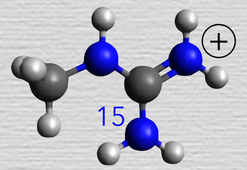
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr

atomic_numbers: per atom numbers

QCSchema Molecule

Cartesian, AU-based topology specification

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.



Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info

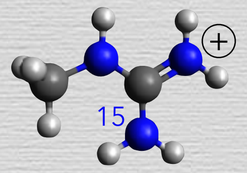
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses

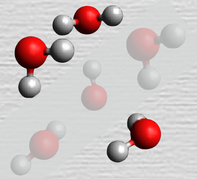
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #

QCSchema Molecule

Cartesian, AU-based topology specification

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.





Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info

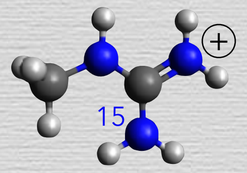
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual

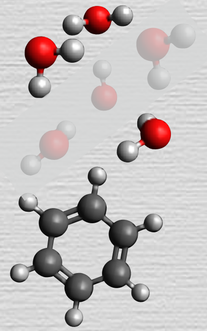
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult

QCSchema Molecule

Cartesian, AU-based topology specification

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.





Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info

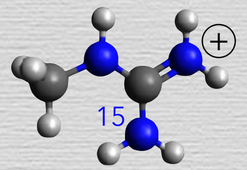
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual

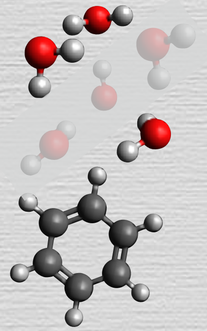
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group

QCSchema Molecule

Cartesian, AU-based topology specification

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden)
- tried to be as **expressive** as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc.





Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group

connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCSchema Molecule

JSON Schema for Molecule (abr.)

```
#> qcelemental.models.Molecule.schema_json()

{'$schema': 'http://json-schema.org/draft-04/schema#',
 'additionalProperties': False,
 'description': 'The physical Cartesian representation of the molecular system.\n'
 'properties': {'atomic_numbers': {'description': 'An optional ordered 1-D array-like object of atomic numbers of
shape (nat,). Index matches the 0-indexed indices of all other per-atom settings '
                                                 'like :attr:`~qcelemental.models.Molecule.symbols`
and :attr:`~qcelemental.models.Molecule.real`. Values are inferred from the '
                                                 ':attr:`~qcelemental.models.Molecule.symbols` list if not explicitly
set. Ghostedness should be indicated through '
                                                 ':attr:`~qcelemental.models.Molecule.real` field, not zeros here.',
                                     'items': {'multipleOf': 1.0, 'type': 'number'},
                                     'shape': ['nat'],
                                     'title': 'Atomic Numbers',
                                     'type': 'array'},
                 'fix_symmetry': {'description': 'Maximal point group symmetry
which :attr:`~qcelemental.models.Molecule.geometry` should be treated. Lowercase.',
                                  'title': 'Fix Symmetry',
                                  'type': 'string'},
                 'geometry': {'description': 'The ordered array for Cartesian XYZ atomic coordinates [a0]. Atom
ordering is fixed; that is, a consumer who shuffles atoms must not reattach the input '
                                                 '(pre-shuffling) molecule schema instance to any output (post-shuffling)
per-atom results (e.g., gradient). Index of the first dimension matches the '
                                                 '0-indexed indices of all other per-atom settings
like :attr:`~qcelemental.models.Molecule.symbols` and :attr:`~qcelemental.models.Molecule.real`.\n'
                                                 'Serialized storage is always flat, (3*nat,), but QCSchema implementations
will want to reshape it. QCElemental can also accept array-likes which can be '
                                                 'mapped to (nat,3) such as a 1-D list of length 3*nat, or the serialized
version of the array in (3*nat,) shape; all forms will be reshaped to (nat,3) for '
                                                 'this attribute.',
                                      'items': {'type': 'number'},
                                      'items': {'type': 'number'},
                                      'shape': ['nat', 3],
                                      'title': 'Geometry',
                                      'type': 'array',
                                      'units': 'a0'}},
...
```

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group

connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCElemental implements QCSchema

with strong validation through pydantic

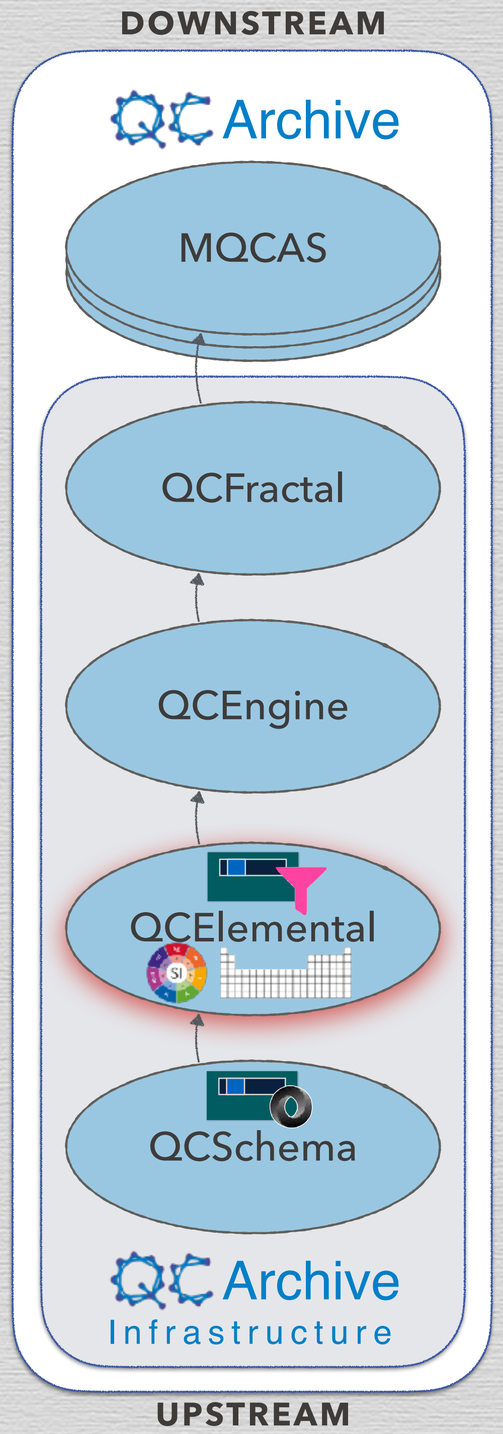
SIX FLAWED MOLECULES

```
> Molecule(geometry=[5, 6], symbols=['O'])
ValueError: cannot reshape array of size 2 into shape (3)

> Molecule(geometry=[4, 4, 4], symbols=['Z'])
NotAnElementError: Z

> Molecule(geometry=[0, 1, 2, 3, 4, 5], symbols=['H', 'H', 'H'])
ValidationError: dropped atoms! nat = 3 != 2

> Molecule(geometry=[0, 0.001, 0, 0, 0, 0], symbols=['N', 'N'])
ValidationError: Following atoms are too close: [(0, 1, 0.001)]

> Molecule(geometry=[0, 0, 0], symbols=['He'], molecular_charge=0, molecular_multiplicity=2)
ValidationError: Inconsistent or unspecified chg/mult: sys chg: 0, frag chg: [None], sys mult: 2,
frag mult: [None]

> Molecule(geometry=[0, 0, 0], symbols=['He'], fragments=[[0], [1]])
ValidationError: dropped atoms! nat = 2 != 1
```

THAT RAISE HELPFUL ERRORS IN QCElemental

QCArchive Stack

qcarchive.molssi.org/



- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

QCElemental provides datasets

NIST CODATA, covalent and vdW radii, Periodic Table

- **STRUCTURED DATA** processed into light Python API directly from NIST structured data, if available.
- **PRECISION** original significant figures value always available.
- **VERSIONED** when reproducibility demands old versions, accessing both helps transitions.
- **EXPORTS** to C and Fortran headers.
- **CONVERSION** with `pint` module, internally consistent unit conversions available.
- **STORAGE/USER UNITS** maintain AU in QCSchema, while presenting results in customary units.

```
> import qcelemental as qcel
> qcel.constants.Hartree_energy_in_eV
27.21138602
> pc = qcel.constants.get('hartree ENERGY in ev', return_tuple=True)
> pc.label
'Hartree energy in eV'
> pc.data
Decimal('27.21138602')
> pc.units
'eV'
> pc.comment
'uncertainty=0.000 000 17'
> qcel.constants.conversion_factor("bohr", "miles")
3.2881547429884475e-14
```

```
> import qcelemental as qcel
> qcel.periodictable.to_E('KRYPTON')
'Kr'
> qcel.periodictable.to_element(36)
'Krypton'
> qcel.periodictable.to_Z('kr84')
36
> qcel.periodictable.to_A('Kr')
84
> qcel.periodictable.to_mass('Kr86')
85.9106106269
```

QCSchema Molecule

Cartesian, AU-b

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden
- tried to be as expressive as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc. Nevertheless, missed:
  - periodic box vectors **NYI, see end**
  - per-atom spin info **NYI, interested?**
  - **others?**
- **FULL** access to data in shaped NumPy arrays with periodic table default lookups
- **CONCISE** serialization of only supplied fields
- helper functions for alignment, extraction, & export to other formats
- validation code also used by Psi4, so well **TESTED**
- **JSON** schema exported from Pydantic model

```python
# create mol from arrays
from qcelemental.models import Molecule
dimer = Molecule(symbols=["Ne", "Ne"], geometry=[0,0,0,0,0,3])
print(dimer)
```

```
Molecule(name='Ne2', formula='Ne2', hash='5ca593f')
```

```python
# data available in arrays from periodic table lookup
dimer.masses
```

```
array([19.99244018, 19.99244018])
```

```python
# or create mol from string: XYZ or Psi4
neoh = Molecule.from_data("""
0 0 0 0
2H 2 0 0  # nuclide
--
@Ne 5 0 0  # ghost
units au
""")
print(neoh)
```

```
Molecule(name='HNeO', formula='HNeO', hash='47ba02b')
```

```python
# data reflects input string
neoh.mass_numbers
```

```
array([16,  2, 20], dtype=int16)
```

```python
# AU internals easy to convert
neoh.geometry * qcel.constants.conversion_factor("bohr", "nm")
```

```
array([[0.        , 0.        , 0.        ],
       [0.10583544, 0.        , 0.        ],
       [0.26458861, 0.        , 0.        ]])
```

```python
# export into other formats
txt, info = neoh.to_string("qchem", return_data=True, prec=4)
print(txt, info["keywords"])
```

```
$molecule
0 2
--
0 2
O                              0.0000             0.0000             0.0000
H                              2.0000             0.0000             0.0000
--
0 1
@Ne                            5.0000             0.0000             0.0000
$end
 {'no_reorient': False, 'input_bohr': 'True'}
```

```python
# full or consise serialization to Py, JSON, msgpack
pprint.pprint(neoh.dict())
```

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group

connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCSchema Molecule

Cartesian, AU-b

- stores fully specified (no variables) **Cartesian** coordinates in Bohr, consistent with QCSchema using only **atomic units**, but easy to convert. (Zmat abilities hidden
- tried to be as expressive as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc. Nevertheless, missed:
  - periodic box vectors NYI, see end
  - per-atom spin info NYI, interested?
  - others?
- **FULL** access to data in shaped NumPy arrays with periodic table default lookups
- **CONCISE** serialization of only supplied fields
- helper functions for alignment, extraction, & export to other formats
- validation code also used by Psi4, so well **TESTED**
- **JSON** schema exported from Pydantic model

```python
# create mol from arrays
from qcelemental.models import Molecule
dimer = Molecule(symbols=["Ne", "Ne"], geometry=[0,0,0,0,0,3])
print(dimer)
```

```
Molecule(name='Ne2', formula='Ne2', hash='5ca593f')
```

```python
# data available in arrays from periodic table lookup
dimer.masses
```

```
array([19.99244018, 19.99244018])
```

```python
# or create mol from string: XYZ or Psi4
neoh = Molecule.from_data("""
0 0 0 0
2H 2 0 0  # nuclide
--
@Ne 5 0 0  # ghost
units au
""")
print(neoh)
```

```
Molecule(name='HNeO', formula='HNeO', hash='47ba02b')
```

```python
# data reflects input string
neoh.mass_numbers
```

```
array([16,  2, 20], dtype=int16)
```

```python
# AU internals easy to convert
neoh.geometry * qcel.constants.conversion_factor("bohr", "nm")
```

```
array([[0.        , 0.        , 0.        ],
       [0.10583544, 0.        , 0.        ],
       [0.26458861, 0.        , 0.        ]])
```

```python
# export into other formats
txt, info = neoh.to_string("qchem", return_data=True, prec=4)
print(txt, info["keywords"])
```

```
$molecule
0 2
--
0 2
O                              0.0000             0.0000             0.0000
H                              2.0000             0.0000             0.0000
--
0 1
@Ne                            5.0000             0.0000             0.0000
$end
 {'no_reorient': False, 'input_bohr': 'True'}
```

```python
# full or consise serialization to Py, JSON, msgpack
pprint.pprint(neoh.dict())
```

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group
validated: passed physics checks
connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCSCHE

Cartesian, AU-b

- stores fully specified (no variables) **CARTESIAN** coordinates in Bohr, consistent with QCSCHEMA using only **ATOMIC UNITS**, but easy to convert. (Zmat abilities hidden
- tried to be as expressive as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc. Nevertheless, missed:
  - periodic box vectors **NYI, see end**
  - per-atom spin info **NYI, interested?**
  - **others?**
- **FULL** access to data in shaped NUMPY arrays with periodic table default lookups
- **CONCISE** serialization of only supplied fields
- helper functions for alignment, extraction, & export to other formats
- validation code also used by PSI4, so well **TESTED**
- **JSON** schema exported from PYDANTIC model

```
@Ne 5 0 0  # ghost
units au
""")
print(neoh)
```

```
Molecule(name='HNeO', formula='HNeO', hash='47ba02b')
```

```
# data reflects input string
neoh.mass_numbers
```

```
array([16,  2, 20], dtype=int16)
```

```
# AU internals easy to convert
neoh.geometry * qcel.constants.conversion_factor("bohr", "nm")
```

```
array([[0.        , 0.        , 0.        ],
       [0.10583544, 0.        , 0.        ],
       [0.26458861, 0.        , 0.        ]])
```

```
# export into other formats
txt, info = neoh.to_string("qchem", return_data=True, prec=4)
print(txt, info["keywords"])
```

```
$molecule
0 2
--
0 2
O                          0.0000            0.0000            0.0000
H                          2.0000            0.0000            0.0000
--
0 1
@Ne                        5.0000            0.0000            0.0000
$end
 {'no_reorient': False, 'input_bohr': 'True'}
```

```
# full or consise serialization to Py, JSON, msgpack
pprint.pprint(neoh.dict())
```

```
{'extras': {},
 'fix_com': False,
 'fix_orientation': False,
 'fragment_charges': [0.0, 0.0],
 'fragment_multiplicities': [2, 1],
 'fragments': [array([0, 1], dtype=int32), array([2], dtype=int32)],
 'geometry': array([[0., 0., 0.],
       [2., 0., 0.],
       [5., 0., 0.]]),
 'mass_numbers': array([16,  2, 20], dtype=int16),
 'masses': array([15.99491462,  2.01410178, 19.99244018]),
 'molecular_charge': 0.0,
 'molecular_multiplicity': 2,
 'name': 'HNeO',
 'provenance': {'creator': 'QCElemental',
                'routine': 'qcelemental.molparse.from_string',
                'version': '0.28.0'},
 'real': array([ True,  True, False]),
 'schema_name': 'qcschema_molecule',
 'schema_version': 2,
 'symbols': array(['O', 'H', 'Ne'], dtype='<U2'),
 'validated': True}
```

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group
validated: passed physics checks
connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCSCHE

Cartesian, AU-b

- stores fully specified (no variables) **CARTESIAN** coordinates in Bohr, consistent with QCSCHEMA using only **ATOMIC UNITS**, but easy to convert. (Zmat abilities hidden
- tried to be as expressive as possible, incorporating isotopes, fragments, user labeling, intentionality of supplied frame, connectivity, etc. Nevertheless, missed:
  - periodic box vectors **NYI, see end**
  - per-atom spin info **NYI, interested?**
  - **others?**
- **FULL** access to data in shaped NUMPY arrays with periodic table default lookups
- **CONCISE** serialization of only supplied fields
- helper functions for alignment, extraction, & export to other formats
- validation code also used by PSI4, so well **TESTED**
- **JSON** schema exported from PYDANTIC model

★ provide validation & helper function to save repetition in user code stack

```
@Ne 5 0 0  # ghost
units au
""")
print(neoh)
```

```
Molecule(name='HNeO', formula='HNeO', hash='47ba02b')
```

```
# data reflects input string
neoh.mass_numbers
```

```
array([16,  2, 20], dtype=int16)
```

```
# AU internals easy to convert
neoh.geometry * qcel.constants.conversion_factor("bohr", "nm")
```

```
array([[0.        , 0.        , 0.        ],
       [0.10583544, 0.        , 0.        ],
       [0.26458861, 0.        , 0.        ]])
```

```
# export into other formats
txt, info = neoh.to_string("qchem", return_data=True, prec=4)
print(txt, info["keywords"])
```

```
$molecule
0 2
--
0 2
O                          0.0000            0.0000            0.0000
H                          2.0000            0.0000            0.0000
--
0 1
@Ne                        5.0000            0.0000            0.0000
$end
 {'no_reorient': False, 'input_bohr': 'True'}
```

```
# full or consise serialization to Py, JSON, msgpack
pprint.pprint(neoh.dict())
```

```
{'extras': {},
 'fix_com': False,
 'fix_orientation': False,
 'fragment_charges': [0.0, 0.0],
 'fragment_multiplicities': [2, 1],
 'fragments': [array([0, 1], dtype=int32), array([2], dtype=int32)],
 'geometry': array([[0., 0., 0.],
       [2., 0., 0.],
       [5., 0., 0.]]),
 'mass_numbers': array([16,  2, 20], dtype=int16),
 'masses': array([15.99491462,  2.01410178, 19.99244018]),
 'molecular_charge': 0.0,
 'molecular_multiplicity': 2,
 'name': 'HNeO',
 'provenance': {'creator': 'QCElemental',
                'routine': 'qcelemental.molparse.from_string',
                'version': '0.28.0'},
 'real': array([ True,  True, False]),
 'schema_name': 'qcschema_molecule',
 'schema_version': 2,
 'symbols': array(['O', 'H', 'Ne'], dtype='<U2'),
 'validated': True}
```

Molecule

schema_version: 2 *(used w/QCSk v1)*
provenance: creator info
id: tracker for databases
extras: free-form storage
name: user label
comment: further user text
symbols: elemental symb. for atoms
geometry: Cartesian coord. in Bohr
molecular_charge: net elst. charge
molecular_multiplicity: total mult.
masses: per atom atomic masses
real: per atom real vs. ghost/virtual
atom_labels: further per atom labels
atomic_numbers: per atom numbers
mass_numbers: per atom isotope #
fragments: group atoms into fragmts
fragment_charges: partitioned chg
fragment_multiplicities: ptn'd mult
fix_com: conserve input COM
fix_orientation: conserve input frame
fix_symmetry: use lesser point group
validated: passed physics checks
connectivity: bond info
identifiers: registration/classification

Identifiers (abr.)

molecular_formula
smiles
inchi
canonical_isomeric_smiles
canonical_smiles
pubchem_cid
pubchem_conformerid

QCSchema BasisSet & App BSE

https://www.basissetexchange.org/

BasisSet

schema_version: 1
name: BSE standard name, e.g., cc-pVDZ
description: text of characteristics
nbf: basis function count
atom_map: center & mol. correspondence
center_data: merged for all atoms/centers
{center_id:

BasisCenter

electron_shells: elec. shells for this center
[

ElectronShell

angular_momentum: *d*=[2]; *sp*=[0,1]
harmonic_type: shell repr. for AM≥2
- **spherical**
- **cartesian**

exponents: exp. for contracted shell
coefficients: nested list of coeff. Outer generalized; inner segmented.
]

ecp_electrons: #e^- replaced by field pots
ecp_potentials: field pots for this center
[

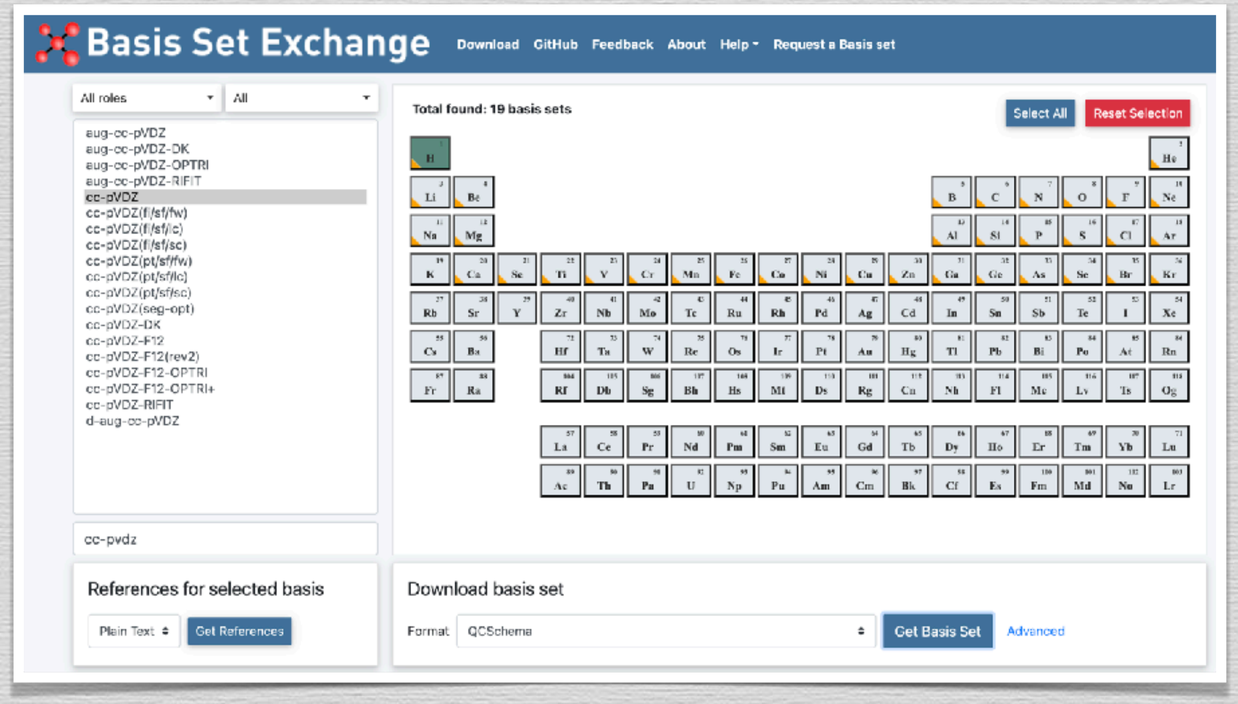
ECPPotential

angular_momentum: *d*=[2]; *sp*=[0,1]
ecp_type:
- **scalar**
- **spinorbit**

r_exponents: exp. of *r* term
gaussian_exponents: exp. of e^{-r^2} term
coefficients: nested list of coeff. Outer generalized; inner segmented.
]
}

QCSchema BASISSET & APP BSE

https://www.basissetexchange.org/



Ben Pritchard
MolSSI

Doaa Altarawy
MolSSI

Susi Lehtola
Helsinki

BSE OUTPUT QCSCHEMA FORMAT


```
{ "schema_name": "qcschema_basis",
  "schema_version": 1,
  "name": "cc-pVDZ",
  "description": "cc-pVDZ",
  "center_data": {
    "h_cc-pVDZ": {
      "electron_shells": [
        { "angular_momentum": [0],
          "exponents": [
            "1.301000E+01",
            "1.962000E+00",
            "4.446000E-01",
            "1.220000E-01"],
          "coefficients": [
            [ "1.968500E-02",
              "1.379770E-01",
              "4.781480E-01",
              "5.012400E-01"],
            [ "0.000000E+00",
              "0.000000E+00",
              "0.000000E+00",
              "1.000000E+00"]
          ],
          "harmonic_type": "cartesian"
        },
        { "angular_momentum": [1],
          "exponents": [
            "7.270000E-01"],
          "coefficients": [
            ["1.0000000"]],
          "harmonic_type": "cartesian"
        }
      ]
    }
  }
}
```


BasisSet

schema_version: 1
name: BSE standard name, e.g., cc-pVDZ
description: text of characteristics
nbf: basis function count
atom_map: center & mol. correspondence
center_data: merged for all atoms/centers
{center_id:

BasisCenter

electron_shells: elec. shells for this center
[

ElectronShell

angular_momentum: *d*=[2]; *sp*=[0,1]
harmonic_type: shell repr. for AM≥2
- **spherical**
- **cartesian**

exponents: exp. for contracted shell
coefficients: nested list of coeff. Outer generalized; inner segmented.
]

ecp_electrons: #e^- replaced by field pots
ecp_potentials: field pots for this center
[

ECPPotential

angular_momentum: *d*=[2]; *sp*=[0,1]
ecp_type:
- **scalar**
- **spinorbit**

r_exponents: exp. of *r* term
gaussian_exponents: exp. of e^{-r^2} term
coefficients: nested list of coeff. Outer generalized; inner segmented.
]
}

QCSchema BASISSET & APP BSE

https://www.basissetexchange.org/

- used by QCSchema **AtomicResult.wavefunction** for AO output in CCA normalization, but not yet used for input wavefunction (diff. normalizations).
- actual BSE schema is close but not quite QCSchema **BasisSet**
- used by BSE for uncentered and publication coefficient and shell-level representations of basis sets (BSE)

LINKING QCSCHEMA MOLECULE & BASISSET

```
import json
import qcelemental as qcel
import basis_set_exchange as bse

mol = qcel.models.Molecule.from_data("""
C 0 0 0
O 0 0 1.2
H 0 -1.0 -0.3
H 0  1.0 -0.3
""")
print(mol)
#> Molecule(name='CH2O', formula='CH2O', hash='2af8547')

# get all elements from BSE
sbas = bse.get_basis("cc-pvdz", elements=mol.symbols.tolist(), fmt="qcschema")
# prepare schema specialized for molecule
bas = qcel.models.BasisSet(**json.loads(sbas),
          atom_map=["c_cc-pVDZ", "o_cc-pVDZ", "h_cc-pVDZ", "h_cc-pVDZ"])
print(f"{bas.name=} {bas.nbf=}")
#> bas.name='cc-pVDZ' bas.nbf=26
```

BasisSet

schema_version: 1
name: BSE standard name, e.g., cc-pVDZ
description: text of characteristics
nbf: basis function count
atom_map: center & mol. correspondence
center_data: merged for all atoms/centers
{center_id:

BasisCenter

electron_shells: elec. shells for this center
[

ElectronShell

angular_momentum: *d*=[2]; *sp*=[0,1]
harmonic_type: shell repr. for AM≥2
- **spherical**
- **cartesian**

exponents: exp. for contracted shell
coefficients: nested list of coeff. Outer generalized; inner segmented.
]

ecp_electrons: #e^- replaced by field pots
ecp_potentials: field pots for this center
[

ECPPotential

angular_momentum: *d*=[2]; *sp*=[0,1]
ecp_type:
- **scalar**
- **spinorbit**

r_exponents: exp. of *r* term
gaussian_exponents: exp. of e^{-r^2} term
coefficients: nested list of coeff. Outer generalized; inner segmented.
]
}

QCSchema ATOMIC

analytical single-point computations

- **AtomicInput** designed as a job directive to express any **SINGLE-GEOMETRY** analytic derivative or property calc. (as defined by `driver`) among CMS workhorses
- in practice, it can hide more complicated computations (e.g., finite difference frequencies), and advanced Result schema often seemingly inherit from **AtomicResult**
- note that QCSchema controls **LAYOUT** and "manages" values for **Molecule** and `driver` with model and enum respectively. All other values are free-form strings in CMS native syntax. See "Data Layouts" slide later.
- **AtomicResult** **APPENDS** to **AtomicInput** with output, simple scalar and array results, and `wavefunction` results.
- `provenance` tracks who wrote the data – program version, module, compute information
- note **FLAT** structure where Result inherits from Input
- overwhelmingly the most used of the calculation schema. These are stored by the **MILLIONS** at MolSSI
- Called "Atomic" for "fundamental"

What best describes this – Atomic, SinglePoint, other?

QCSchema ATOMIC

analytical single-point computations

- **AtomicInput** designed as a job directive to express any **SINGLE-GEOMETRY** analytic derivative or property calc. (as defined by `driver`) among CMS workhorses
- in practice, it can hide more complicated computations (e.g., finite difference frequencies), and advanced Result schema often seemingly inherit from **AtomicResult**
- note that QCSchema controls **LAYOUT** and "manages" values for **Molecule** and `driver` with model and enum respectively. All other values are free-form strings in CMS native syntax. See "Data Layouts" slide later.
- **AtomicResult** **APPENDS** to **AtomicInput** with output, simple scalar and array results, and `wavefunction` results.
- `provenance` tracks who wrote the data – program version, module, compute information
- note **FLAT** structure where Result inherits from Input
- overwhelmingly the most used of the calculation schema. These are stored by the **MILLIONS** at MolSSI
- Called "Atomic" for "fundamental"

What best describes this – Atomic, SinglePoint, other?

AtomicInput

schema_version: 1

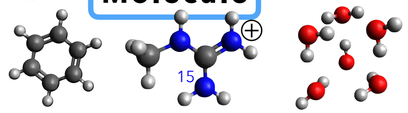
provenance: creator, version, & RT info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

molecule: **Molecule**



protocols: customize return layout

AtomicResultProtocols

driver: single-point derivative

- energy
- gradient
- Hessian
- properties

E *G* *H*

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

QCSchema ATOMIC

analytical single-point computations

- **AtomicInput** designed as a job directive to express any **SINGLE-GEOMETRY** analytic derivative or property calc. (as defined by `driver`) among CMS workhorses
- in practice, it can hide more complicated computations (e.g., finite difference frequencies), and advanced Result schema often seemingly inherit from **AtomicResult**
- note that QCSchema controls **LAYOUT** and "manages" values for **Molecule** and `driver` with model and enum respectively. All other values are free-form strings in CMS native syntax. See "Data Layouts" slide later.
- **AtomicResult** **APPENDS** to **AtomicInput** with output, simple scalar and array results, and `wavefunction` results.
- `provenance` tracks who wrote the data – program version, module, compute information
- note **FLAT** structure where Result inherits from Input
- overwhelmingly the most used of the calculation schema. These are stored by the **MILLIONS** at MolSSI
- Called "Atomic" for "fundamental"

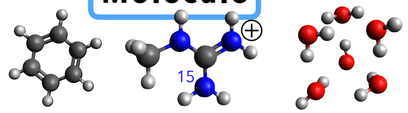
What best describes this – Atomic, SinglePoint, other?

AtomicResultProtocols

stdout: save text output
wavefunction: save orbital info
native_files: save raw prog. files
error_correction: auto-edit&rerun

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



protocols: customize return layout
AtomicResultProtocols
driver: single-point derivative

- energy
- gradient
- Hessian
- properties

E *G* *H*

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

QCSchema ATOMIC

analytical single-point computations

- **AtomicInput** designed as a job directive to express any **SINGLE-GEOMETRY** analytic derivative or property calc. (as defined by `driver`) among CMS workhorses
- in practice, it can hide more complicated computations (e.g., finite difference frequencies), and advanced Result schema often seemingly inherit from **AtomicResult**
- note that QCSchema controls **LAYOUT** and "manages" values for **Molecule** and `driver` with model and enum respectively. All other values are free-form strings in CMS native syntax. See "Data Layouts" slide later.
- **AtomicResult** **APPENDS** to **AtomicInput** with output, simple scalar and array results, and `wavefunction` results.
- `provenance` tracks who wrote the data – program version, module, compute information
- note **FLAT** structure where Result inherits from Input
- overwhelmingly the most used of the calculation schema. These are stored by the **MILLIONS** at MolSSI
- Called "Atomic" for "fundamental"

What best describes this – Atomic, SinglePoint, other?

AtomicResult

AtomicInput

schema_version: 1

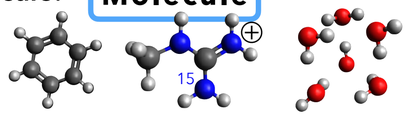
provenance: creator, version, & RT info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

molecule: **Molecule**



protocols: customize return layout

AtomicResultProtocols

driver: single-point derivative

- energy
- gradient
- Hessian
- properties

E *G* *H*

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

success: True

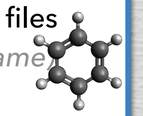
provenance: *(overwrites input)*

`xtb, 1.0.0, psinet, CPU E5-2630`

stdout/stderr: program's text output for ppl

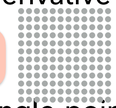
native_files: requested raw program files

molecule: *(overwrites inp w/result frame)*



return_result: single-point derivative

E · *G* *H*



properties: key results for single-point

AtomicResultProperties

wavefunction: orbital-level results

WavefunctionProperties

QCSchema ATOMIC

analytical single-point computations


```
'properties': {'calcinfo_nalpha': 10,
               'calcinfo_natom': 2,
               'calcinfo_nbasis': 28,
               'calcinfo_nbeta': 10,
               'calcinfo_nmo': 28,
               'nuclear_repulsion_energy': 16.666666666666668,
               'return_energy': -256.977621071098,
               'return_gradient': array([[0,  0, -6.54470201e-05],
                                         [0,  0,  6.54470212e-05]]),
               'return_hessian': None,
               'scf_dipole_moment': array([0,  0, 4.97379915e-14]),
               'scf_iterations': 5,
               'scf_one_electron_energy': -398.5650589792168,
               'scf_total_energy': -256.977621071098,
               'scf_total_gradient': array([[0,  0, -6.54470201e-05],
                                            [0,  0,  6.54470212e-05]]),
               'scf_total_hessian': None,
               'scf_two_electron_energy': 124.92077124145213},

'extras': {'qcvars': {'CURRENT DIPOLE': array([0.00000000e+00, 0.00000000e+00, 4.97379915e-14]),
                      'CURRENT ENERGY': -256.977621071098,
                      'CURRENT GRADIENT': array([[0,  0, -6.54470201e-05],
                                                 [0,  0,  6.54470212e-05]]),
                      'CURRENT REFERENCE ENERGY': -256.977621071098,
                      'DD SOLVATION ENERGY': 0.0,
                      'HF KINETIC ENERGY': 256.97713564856093,
                      'HF POTENTIAL ENERGY': -513.9547567196589,
                      'HF TOTAL ENERGY': -256.977621071098,
                      'HF TOTAL GRADIENT': array([[0,  0, -6.54470201e-05],
                                                  [0,  0,  6.54470212e-05]]),
                      'HF VIRIAL RATIO': 2.000001888971701,
                      'NUCLEAR REPULSION ENERGY': 16.666666666666668,
                      'ONE-ELECTRON ENERGY': -398.5650589792168,
                      'PCM POLARIZATION ENERGY': 0.0,
                      'PE ENERGY': 0.0,
                      'SCF DIPOLE': array([0.00000000e+00, 0.00000000e+00, 4.97379915e-14]
                      'SCF ITERATION ENERGY': -256.977621071098,
                      'SCF ITERATIONS': 5.0,
                      'SCF TOTAL ENERGIES': array([-256.97759612, -256.97762005, -256.9776
                                                   -256.97762107, -256.97762107, -256.9776
                      'SCF TOTAL ENERGY': -256.977621071098,
                      'SCF TOTAL GRADIENT': array([[0,  0, -6.54470201e-05],
                                                   [0,  0,  6.54470212e-05]]),
                      'TWO-ELECTRON ENERGY': 124.92077124145213}},
```

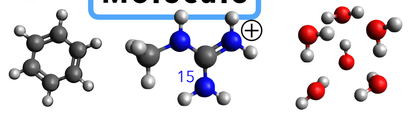

AtomicResultProperties (abr.)

calcinfo_natom
calcinfo_nmo
nuclear_repulsion_energy
return_energy: *E*
return_gradient: *G*
scf_total_*egh*: ref. energy, gradient, etc.
***mtd*_correlation_energy:** post-scf energy
***mtd*_iterations:** count to convergence
***mtd*_mpole_moment:** dipole etc. array

AtomicResult

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



protocols: customize return layout

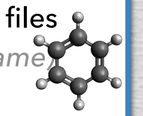
AtomicResultProtocols

driver: single-point derivative
- energy
- gradient
- Hessian
- properties

E *G* *H*

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

success: True
provenance: *(overwrites input)*
`xtb, 1.0.0, psinet, CPU E5-2630`
stdout/stderr: program's text output for ppl
native_files: requested raw program files
molecule: *(overwrites inp w/result frame)*



return_result: single-point derivative

E *G* *H*

properties: key results for single-point

AtomicResultProperties

wavefunction: orbital-level results

WavefunctionProperties

QCSchema ATOMIC

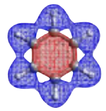
analytical single-point computations

- **AtomicInput** designed as a job directive to express any **SINGLE-GEOMETRY** analytic derivative or property calc. (as defined by `driver`) among CMS workhorses
- in practice, it can hide more complicated computations (e.g., finite difference frequencies), and advanced Result schema often seemingly inherit from **AtomicResult**
- note that QCSchema controls **LAYOUT** and "manages" values for **Molecule** and `driver` with model and enum respectively. All other values are free-form strings in CMS native syntax. See "Data Layouts" slide later.
- **AtomicResult APPENDS** to **AtomicInput** with output, simple scalar and array results, and `wavefunction` results.
- `provenance` tracks who wrote the data – program version, module, compute information
- note **FLAT** structure where Result inherits from Input
- overwhelmingly the most used of the calculation schema. These are stored by the **MILLIONS** at MolSSI
- Called "Atomic" for "fundamental"

What best describes this – Atomic, SinglePoint, other?

WavefunctionProperties (abr.)

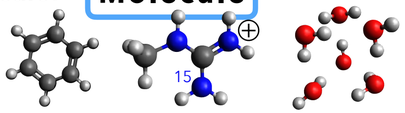
basis: **BasisSet**
restricted: alpha == beta
orbitals_a/b
density_a/b
fock_a/b
eigenvalues_a/b
occupations_a/b



AtomicResult

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



protocols: customize return layout

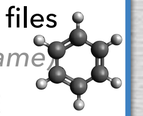
AtomicResultProtocols

driver: single-point derivative
- energy
- gradient
- Hessian
- properties

E G H

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

success: True
provenance: *(overwrites input)*
`xtb, 1.0.0, psinet, CPU E5-2630`
stdout/stderr: program's text output for ppl
native_files: requested raw program files
molecule: *(overwrites inp w/result frame)*



return_result: single-point derivative

E G H

properties: key results for single-point

AtomicResultProperties

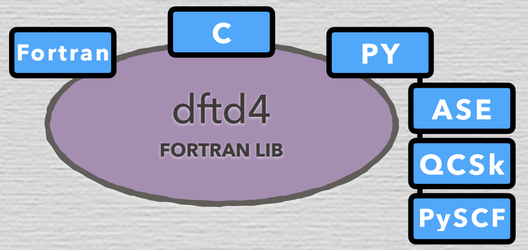
wavefunction: orbital-level results

WavefunctionProperties

APP: AUXILIARY API

with or without QCElemental

- Independent of the QCArchive Infrastructure software, one can use **AtomicInput**/**Result** as the vehicle for a one-shot interface, in addition to a generic **API**.
- Five programs have direct QCSchema **INTERFACES** (and three procedures)
- It is convenient to bring in **QCELEMENTAL** as a dependency to officially form the models, but it isn't necessary.



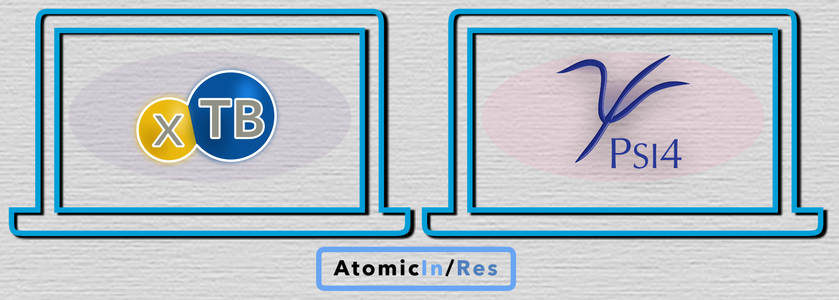
PROGRAMS RUNNING QCSchema

```
import dftd4  # or dftd3

atres = dftd4.qcschema.run_qcschema(atin)
```

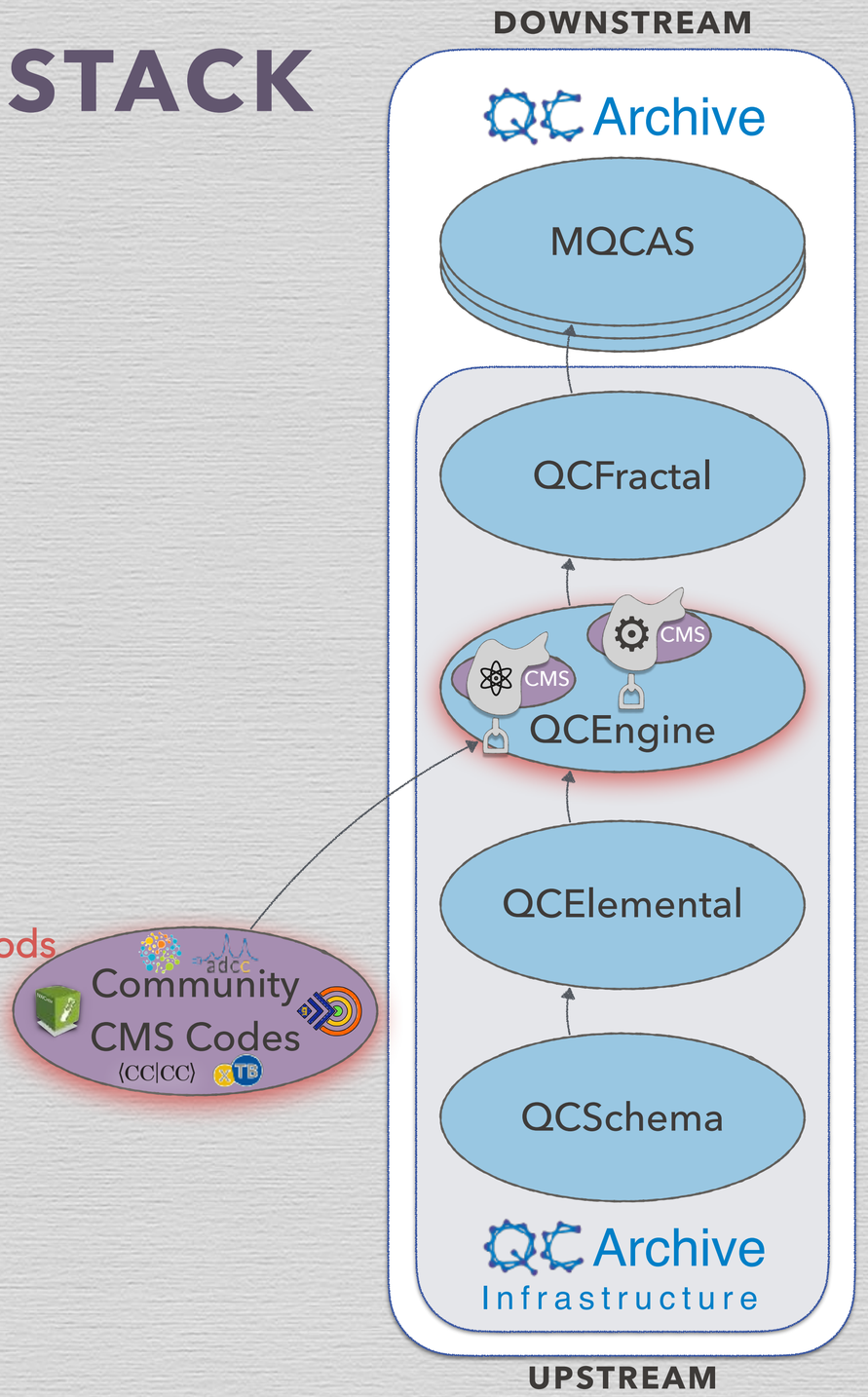
```
import xtb

atres = xtb.qcschema.harness.run_qcschema(atin)
```

```
import tcpb

atres = tcpb._tcpb_client.compute(atin)
```

```
import psi4

atres = psi4.schema_wrapper.run_qcschema(atin)
```



QCArchive Stack



- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

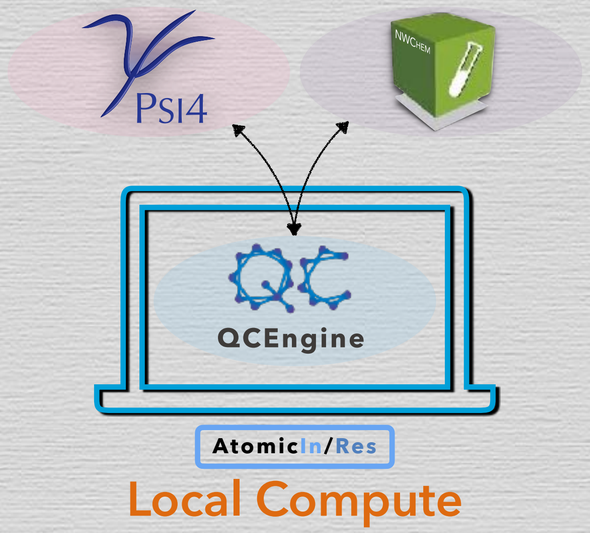
- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

- the difficult part – coded QC methods
- structured output uncommon
- DSL input; API/schema uncommon

QCEngine: Python QCSchema Runner

github.com/MolSSI/QCEngine



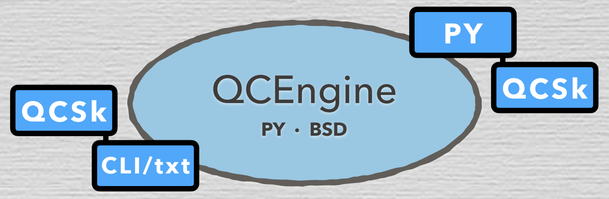
CMDLINE

```
                    cfour
> qcengine run molpro atomicinput
                    psi4
                    dftd4
```

PYAPI

```
                                cfour
qcengine.compute(atomicinput, "molpro", task_config({"nnodes": 4, "memory": 10})
                                psi4
                                dftd4
```

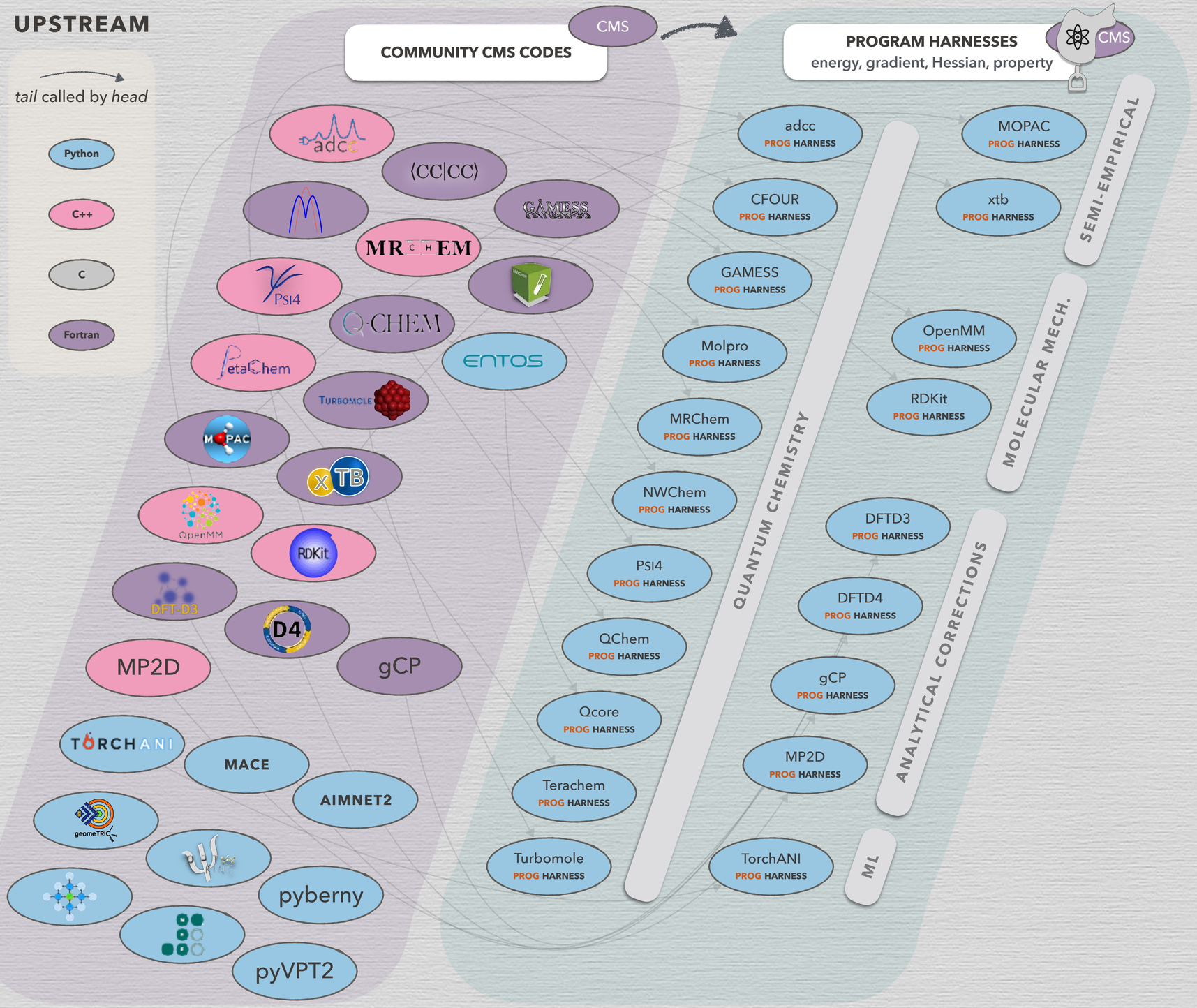
sh> `qcengine run cmscode atomicinput.json`



py> `qcengine.compute("cmscode", atomicinput)`

QCEngine: INTERNAL MAP

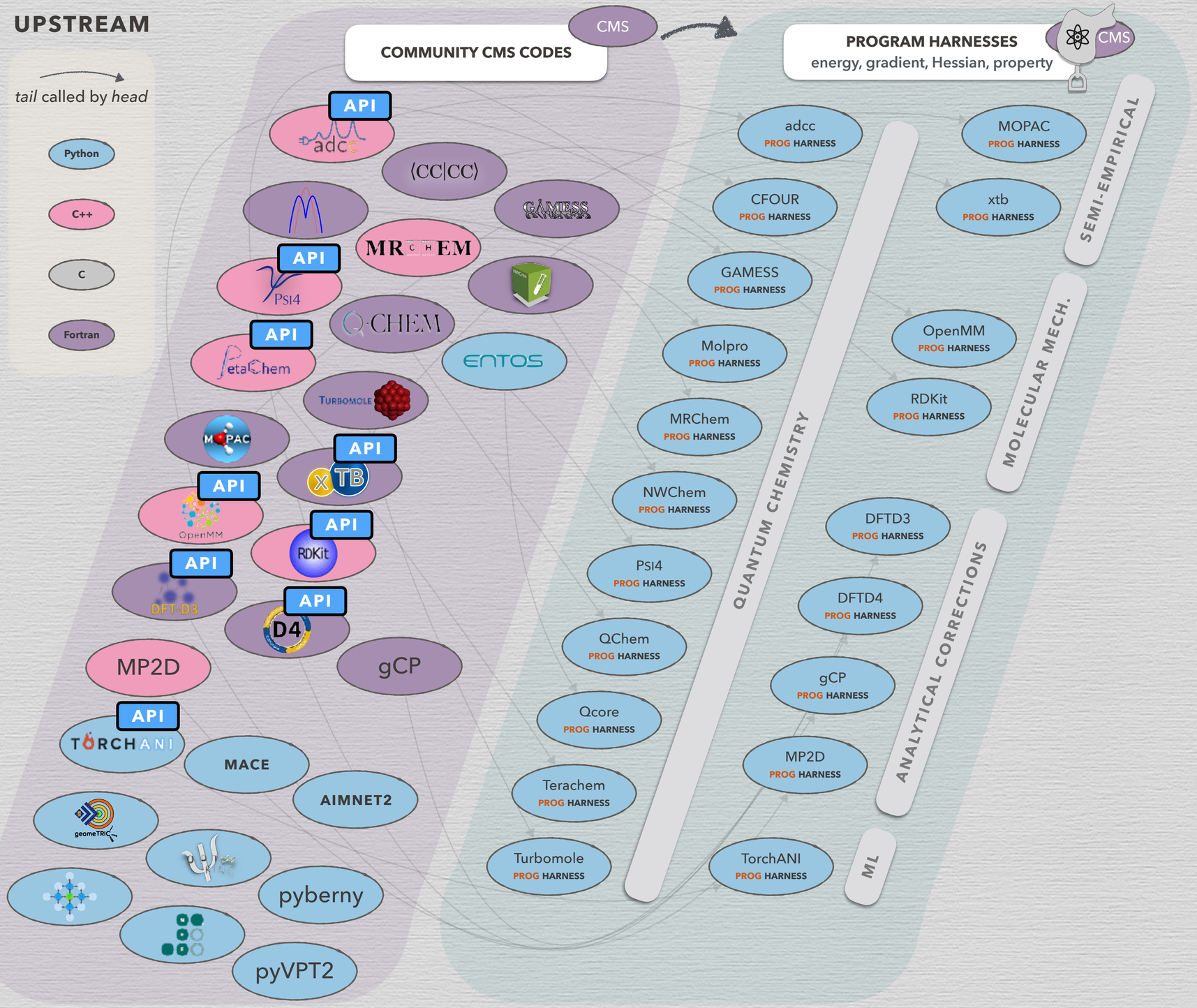
github.com/MolSSI/QCEngine



- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

QCEngine: INTERNAL MAP

github.com/MolSSI/QCEngine

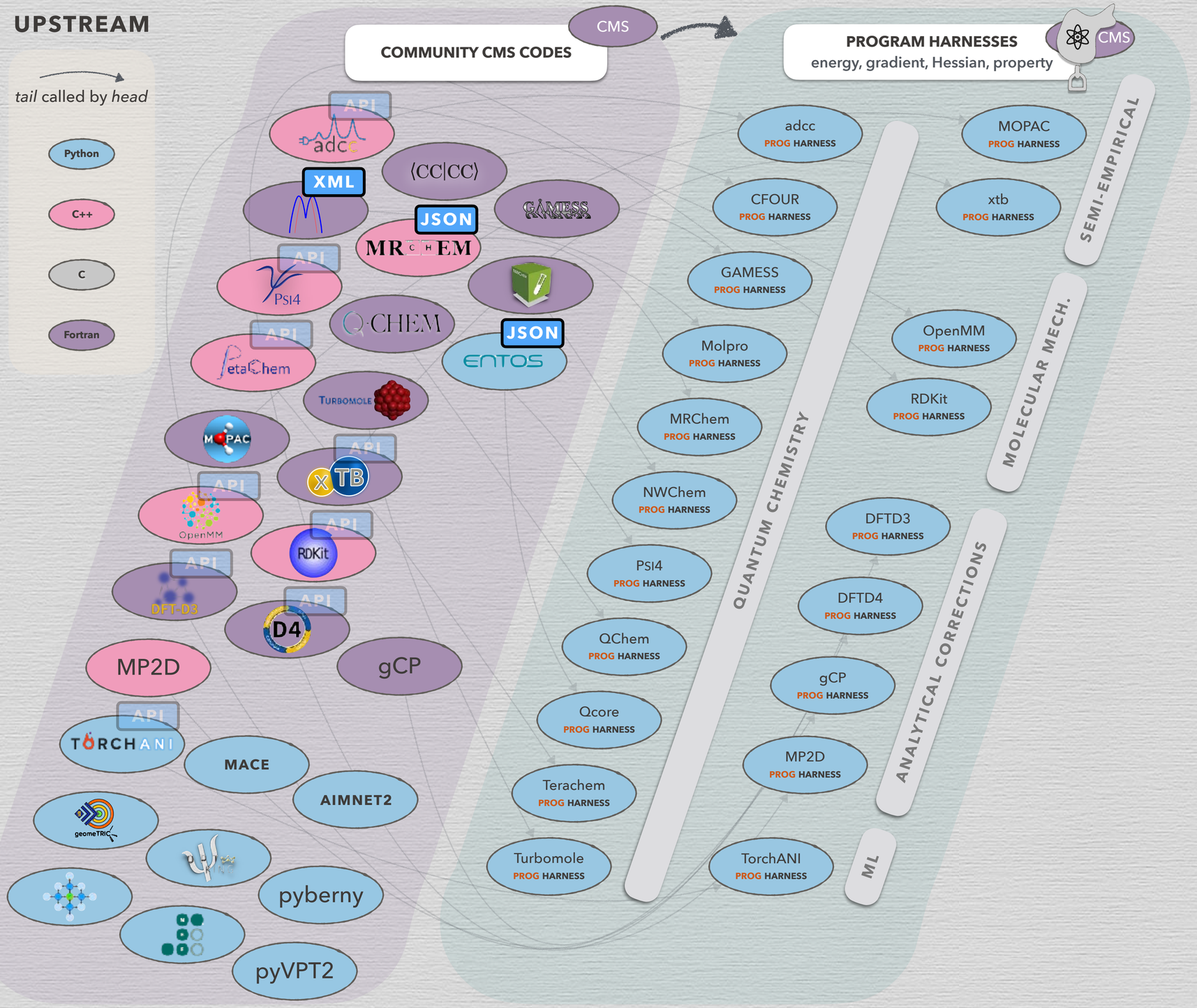


- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

DOWNSTREAM

QCEngine: INTERNAL MAP

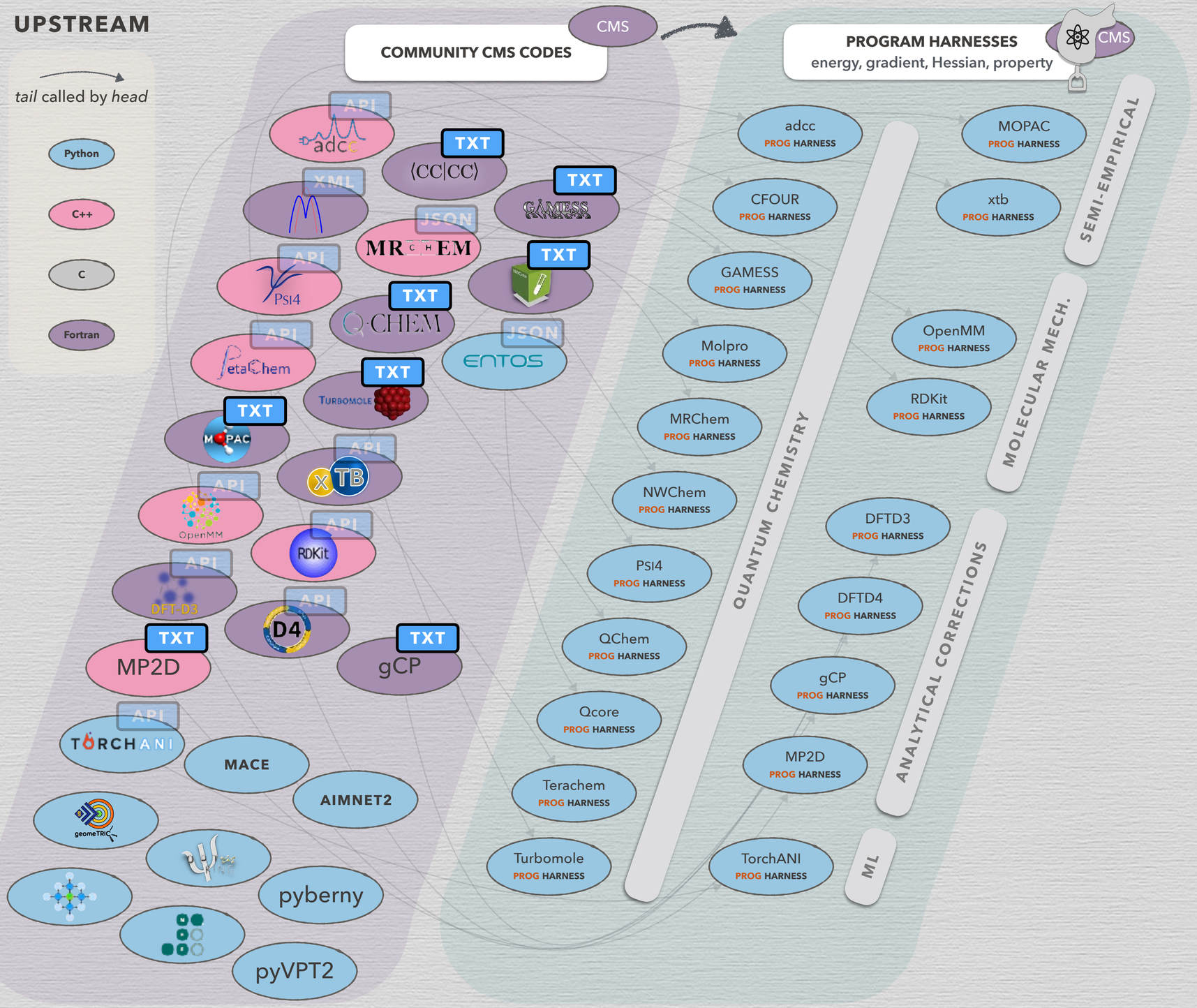
github.com/MolSSI/QCEngine



- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

QCEngine: INTERNAL MAP

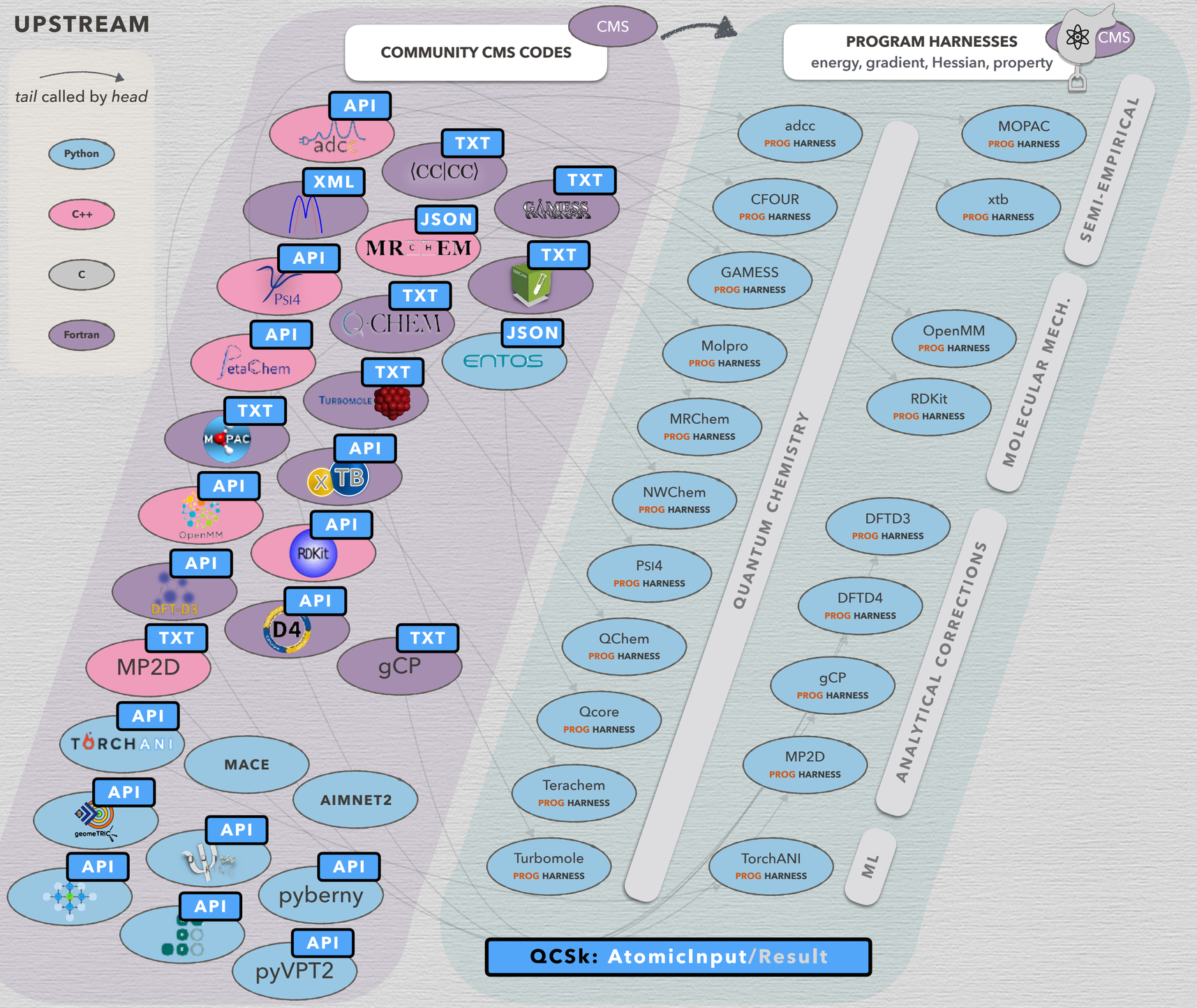
github.com/MolSSI/QCEngine



- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

QCEngine: INTERNAL MAP

github.com/MolSSI/QCEngine



- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

SAMPLE QCEngine HARNESS: MOLPRO

usually a single file

SAMPLE QCENGINE HARNESS: MOLPRO

usually a single file

```python
class MolproHarness(ProgramHarness):
    _defaults: Dict[str, Any] = {
        "name": "Molpro",
        "scratch": True,
        "thread_safe": False,
        "thread_parallel": True,
        "node_parallel": False,
        "managed_memory": True}
    _dft_functionals: Set[str] = {"B86MGC", "B86R", "B86", "B88C"}  # ...

    def found(self, raise_error: bool = False) -> bool:
        return which("molpro", return_bool=True, raise_error=raise_error,
            raise_msg="Please install via https://www.molpro.net/"

    def get_version(self) -> str:
        self.found(raise_error=True)
       if which("molpro") not in self.version_cache:
            success, output = execute([which_prog, "version.inp", "-d", ".", "-W", "."],
                infiles={"version.inp": ""}, outfiles=["version.out", "version.xml"])
            # exctract version from XML and return

    def compute(self, input_data: "AtomicInput", config: "TaskConfig") -> "AtomicResult":
        self.found(raise_error=True)
        if parse_version(self.get_version()) < parse_version("2018.1"):
            raise TypeError("Molpro version '{}' not supported".format(self.get_version()))

        # Set up the job, run, parse if succeeded, error if not
        job_inputs = self.build_input(input_data, config)
        exe_success, proc = self.execute(job_inputs)
        if exe_success:
            return self.parse_output(proc["outfiles"], input_data)
        else:
            return UnknownError(proc["stderr"])

    def execute():  # args omitted
        # Run the Molpro program
        exe_success, proc = execute(
            commands,
            infiles=infiles, outfiles=outfiles, as_binary=as_binary,
            scratch_name=scratch_name,
            scratch_directory=inputs["scratch_directory"],
            scratch_messy=scratch_messy, timeout=timeout)
        return exe_success, proc
```

SAMPLE QCENGINE HARNESS: MOLPRO

usually a single file

```python
class MolproHarness(ProgramHarness):
    _defaults: Dict[str, Any] = {
        "name": "Molpro",
        "scratch": True,
        "thread_safe": False,
        "thread_parallel": True,
        "node_parallel": False,
        "managed_memory": True}
    _dft_functionals: Set[str] = {"B86MGC", "B86R", "B86", "B88C"}  # ...

    def found(self, raise_error: bool = False) -> bool:
        return which("molpro", return_bool=True, raise_error=raise_error,
            raise_msg="Please install via https://www.molpro.net/"

    def get_version(self) -> str:
        self.found(raise_error=True)
       if which("molpro")
            success, outpu
                infiles={'
            # exctract ver

    def compute(self, inpu
        self.found(raise_e
        if parse_version(s
            raise TypeErro

        # Set up the job,
        job_inputs = self.
        exe_success, proc
        if exe_success:
            return self.pa
        else:
            return Unknown

    def execute():  # args
        # Run the Molpro p
        exe_success, proc
            commands,
            infiles=infile
            scratch_name=s
            scratch_direct
            scratch_messy=
        return exe_success
```

```python
    def build_input(
        self, input_model: "AtomicInput", config: "TaskConfig") -> Dict[str, Any]:
        # Memory is in megawords per core for Molpro
        memory_mw_core = int(config.memory * (1024**3) / 8e6 / config.ncores)
        input_file = [append("memory,{},M".format(memory_mw_core))]

        # Write the geom & basis set
        xyz_block = input_model.molecule.to_string(dtype="molpro", units="Bohr")
        input_file.append(xyz_block)
        input_file.append("basis=}" + f"default,{input_model.model.basis}" + "}")

        # Write method & driver
        unrestricted = input_model.keywords.get("reference", "").lower() == "unrestricted"
        if input_model.model.method.upper() in self._dft_functionals:  # DFT case
            energy_call = [f"{{{'UKS' if unrestricted else 'RKS'},{input_model.model.method}}}"]
        input_file.extend(energy_call)
        if input_model.driver == "gradient":
            input_file.append("{force}")

        input_file = "\n".join(input_file)

        return {
            "commands": ["molpro", "dispatch.mol", "-d", ".", "-W", ".", "-n", str(config.ncores)],
            "infiles": {"dispatch.mol": input_file},
            "scratch_directory": config.scratch_directory,
            "input_result": input_model.copy(deep=True),
        }
```

SAMPLE QCENGINE HARNESS: MOLPRO

usually a single file

```python
class MolproHarness(ProgramHarness):
    _defaults: Dict[str, Any] = {
        "name": "Molpro",
        "scratch": True,
        "thread_safe": False,
        "thread_parallel": True,
        "node_parallel": False,
        "managed_memory": True}
    _dft_functionals: Set[str] = {"B86MGC", "B86R", "B86

    def found(self, raise_error: bool = False) -> bool:
        return which("molpro", return_bool=True, raise_e
            raise_msg="Please install via https://www.mo

    def get_version(self) -> str:
        self.found(raise_error=True)
       if which("molpro")
            success, outpu
                infiles={"
            # exctract ver

    def compute(self, inpu
        self.found(raise_e
        if parse_version(s
            raise TypeErro

        # Set up the job,
        job_inputs = self.
        exe_success, proc
        if exe_success:
            return self.pa
        else:
            return Unknown

    def execute():  # args
        # Run the Molpro p
        exe_success, proc
            commands,
            infiles=infile
            scratch_name=s
            scratch_direct
            scratch_messy=
        return exe_success
```

```python
    def build_input(
        self, input_model: "At
        # Memory is in megawor
        memory_mw_core = int(c
        input_file = [append("

        # Write the geom & bas
        xyz_block = input_mode
        input_file.append(xyz_
        input_file.append("bas

        # Write method & drive
        unrestricted = input_m
        if input_model.model.m
            energy_call = [f"{
        input_file.extend(ener
        if input_model.driver
            input_file.append(

        input_file = "\n".join

        return {
            "commands": ["molp
            "infiles": {"dispa
            "scratch_directory
            "input_result": in
        }
```

```python
    def parse_output(self, outfiles: Dict[str, str], input_model: "AtomicInput") -> "AtomicResult":
        tree = ET.ElementTree(ET.fromstring(outfiles["dispatch.xml"]))
        root = tree.getroot()
        properties = {}

        molpro_variable_map = {
            "_ENUC": "nuclear_repulsion_energy",
            "_DFTFUN": "scf_xc_energy",
            "_NELEC": ["calcinfo_nalpha", "calcinfo_nbeta"]}

        name_space = {"molpro_uri": "http://www.molpro.net/schema/molpro-output"}
        molecule = root.find("molpro_uri:job/molpro_uri:molecule", name_space)
        molecule_method = molecule.attrib["method"]
        molecule_final_energy = float(molecule.attrib["energy"])
        for variable in molecule.findall("molpro_uri:variables/molpro_uri:variable", name_space):
            pass  # extract from XML and save to properties
        basis_set = root.find("molpro_uri:job/molpro_uri:molecule/molpro_uri:basisSet", name_space)
        nbasis = int(basis_set.attrib["length"])
        properties["calcinfo_nbasis"] = nbasis

        # Grab the method from input
        method = input_model.model.method.upper()

        # Initialize output_data by copying over input_model.dict()
        output_data = input_model.dict()

        # Determining return_result
        if input_model.driver == "energy":
            output_data["return_result"] = final_energy
        elif input_model.driver == "gradient":
            output_data["return_result"] = properties.pop("gradient")

        # Final output_data assignments needed for the AtomicResult object
        output_data["properties"] = properties
        output_data["stdout"] = outfiles["dispatch.out"]
        output_data["success"] = True

        return AtomicResult(**output_data)
```

QCSchema defines data layouts

and text advice, nothing more; QCEngine will not standardize input

DOMAIN-SPECIFIC ASCII

all-electron restricted-open-shell CCSD/aug-cc-pVDZ energy of NH_2 molecule

⟨CC|CC⟩

```
N   0.000   0.000 -0.146
H   0.000 -1.511   1.014
H   0.000   1.511   1.014

*CFOUR(REFERENCE=ROHF
BASIS=AUG-PVDZ,CALC_LEVEL=CCSD
COORDINATES=CARTESIAN,UNITS=BOHR
CHARGE=0,MULTIPLICITY=2)
```

GAMESS

```
$ccinp ncore=0 $end
$basis gbasis=accd $end
$contrl cctyp=ccsd coord=prinaxis
 icharg=0 ispher=1 mult=2
 runtyp=energy scftyp=rohf
 units=bohr $end
$data

C1
N 7  0.000  0.000 -0.146
H 1  0.000 -1.511  1.014
H 1  0.000  1.511  1.014
$end
```

Psi4

```
molecule {
0 2
N   0.000   0.000 -0.146
H   0.000 -1.511   1.014
H   0.000   1.511   1.014
units au
}

set reference rohf

energy('ccsd/aug-cc-pvdz')
```

NWChem

```
geometry units bohr
N   0.000   0.000 -0.146
H   0.000 -1.511   1.014
H   0.000   1.511   1.014
end
charge 0
basis spherical
  h library aug-cc-pvdz
  n library aug-cc-pvdz
end
scf
  rohf
  nopen 1
end
tce
  ccsd
end
task tce energy
```

QCSchema: AtomicInput

QCSchema defines data layouts

and text advice, nothing more; QCEngine will not standardize input

DOMAIN-SPECIFIC ASCII

all-electron restricted-open-shell CCSD/aug-cc-pVDZ energy of NH_2 molecule

⟨CC|CC⟩

```
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014

*CFOUR(REFERENCE=ROHF
BASIS=AUG-PVDZ,CALC_LEVEL=CCSD
COORDINATES=CARTESIAN,UNITS=BOHR
CHARGE=0,MULTIPLICITY=2)
```

GAMESS

```
$ccinp ncore=0 $end
$basis gbasis=accd $end
$contrl cctyp=ccsd coord=prinaxis
 icharg=0 ispher=1 mult=2
 runtyp=energy scftyp=rohf
 units=bohr $end
$data

C1
N 7  0.000  0.000 -0.146
H 1  0.000 -1.511  1.014
H 1  0.000  1.511  1.014
$end
```

Psi4

```
molecule {
0 2
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
units au
}

set reference rohf

energy('ccsd/aug-cc-pvdz')
```

NWChem

```
geometry units bohr
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
end
charge 0
basis spherical
  h library aug-cc-pvdz
  n library aug-cc-pvdz
end
scf
  rohf
  nopen 1
end
tce
  ccsd
end
task tce energy
```

DATA LAYOUT TRANSLATION

QCSchema: AtomicInput

```
{ 'molecule':     ,
  'driver':         ,
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {


                        }}
```

```
{ 'molecule':     ,
  'driver':         ,
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {


                        }}
```

```
{ 'molecule':     ,
  'driver':         ,
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {


                        }}
```

```
{ 'molecule':     ,
  'driver':         ,
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {


                        }}
```

QCSchema defines data layouts

and text advice, nothing more; QCEngine will not standardize input

DOMAIN-SPECIFIC ASCII

all-electron restricted-open-shell CCSD/aug-cc-pVDZ energy of NH_2 molecule

⟨CC|CC⟩

```
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014

*CFOUR(REFERENCE=ROHF
BASIS=AUG-PVDZ,CALC_LEVEL=CCSD
COORDINATES=CARTESIAN,UNITS=BOHR
CHARGE=0,MULTIPLICITY=2)
```

GAMESS

```
$ccinp ncore=0 $end
$basis gbasis=accd $end
$contrl cctyp=ccsd coord=prinaxis
 icharg=0 ispher=1 mult=2
 runtyp=energy scftyp=rohf
 units=bohr $end
$data

C1
N 7  0.000  0.000 -0.146
H 1  0.000 -1.511  1.014
H 1  0.000  1.511  1.014
$end
```

PSI4

```
molecule {
0 2
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
units au
}

set reference rohf

energy('ccsd/aug-cc-pvdz')
```

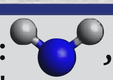
NWChem

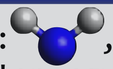
```
geometry units bohr
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
end
charge 0
basis spherical
  h library aug-cc-pvdz
  n library aug-cc-pvdz
end
scf
  rohf
  nopen 1
end
tce
  ccsd
end
task tce energy
```

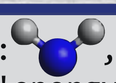
DATA LAYOUT TRANSLATION

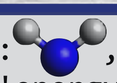
MOL & DRIVER STANDARDIZATION

QCSchema: AtomicInput

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {

                          }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {

                          }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {

                          }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method':       ,
    'basis':              },
  'keywords': {

                          }}
```

QCSchema defines data layouts

and text advice, nothing more; QCEngine will not standardize input

DOMAIN-SPECIFIC ASCII

all-electron restricted-open-shell CCSD/aug-cc-pVDZ energy of NH_2 molecule

⟨CC|CC⟩

```
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014

*CFOUR(REFERENCE=ROHF
BASIS=AUG-PVDZ,CALC_LEVEL=CCSD
COORDINATES=CARTESIAN,UNITS=BOHR
CHARGE=0,MULTIPLICITY=2)
```

GAMESS

```
$ccinp ncore=0 $end
$basis gbasis=accd $end
$contrl cctyp=ccsd coord=prinaxis
 icharg=0 ispher=1 mult=2
 runtyp=energy scftyp=rohf
 units=bohr $end
$data

C1
N 7  0.000  0.000 -0.146
H 1  0.000 -1.511  1.014
H 1  0.000  1.511  1.014
$end
```

PSI4

```
molecule {
0 2
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
units au
}

set reference rohf

energy('ccsd/aug-cc-pvdz')
```

NWChem

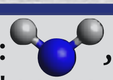
```
geometry units bohr
N  0.000  0.000 -0.146
H  0.000 -1.511  1.014
H  0.000  1.511  1.014
end
charge 0
basis spherical
  h library aug-cc-pvdz
  n library aug-cc-pvdz
end
scf
  rohf
  nopen 1
end
tce
  ccsd
end
task tce energy
```

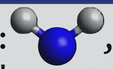
DATA LAYOUT TRANSLATION

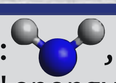
MOL & DRIVER STANDARDIZATION

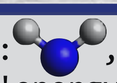
BASIS & KEYWORDS STANDARDIZATION

QCSchema: AtomicInput

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'aug-pvdz'    },
  'keywords': {
    'reference': 'rohf'
                           }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method':'ccsd' ,
    'basis': 'accd'        },
  'keywords': {
    'contrl__ispher': 1,
    'contrl__scftyp': 'rohf',
    'ccinp__ncore': 0      }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'aug-cc-pvdz'},
  'keywords': {
    'reference': 'rohf'
                           }}
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'aug-cc-pvdz'},
  'keywords': {
    'basis__spherical': True,
    'scf__rohf': True,
    'qc_module': 'tce'     }}
```

QCSchema defines data layouts

and text advice, nothing more; QCEngine will not standardize input

INTEROPERABILITY BEYOND QCENGINE

all-ele … c-pVDZ energy of NH_2 molecule

DOMAIN-SPECIFIC ASCII

⟨CC|CC⟩

```
N   0.000   0.000 -0.146
H   0.000  -1.511   1.014
H   0.000   1.511   1.014

*CFOUR(REFERENCE=ROHF
BASIS=AUG-PVDZ,CALC_LEVEL=CCSD
COORDINATES=CARTESIAN,UNITS=BOHR
CHARGE=0,MULTIPLICITY=2)
```

```
$ccinp r
$basis g
$contrl
 icharg=
 runtyp=
 units=b
$data

C1
N 7  0.0
H 1  0.0
H 1  0.0
$end
```

Psi4

```
geometry units bohr
N   0.000   0.000 -0.146
H   0.000  -1.511   1.014
H   0.000   1.511   1.014
end
charge 0
basis spherical
  h library aug-cc-pvdz
  n library aug-cc-pvdz
end
scf
  rohf
  nopen 1
end
tce
  ccsd
end
task tce energy
```

Capabilities of QCDB, including details of overlapping modules. "✓" runs analytically. "!" produces nonstandard results.

□	qc_module ↓	QCDB Capabilities																	
□	scf_reference →	Restricted (RHF)						Unrestricted (UHF)						Restricted Open (ROHF)					
name ↓ →	□	E		G		H		E		G		H		E		G		H	
□	freeze_core[1] →	A	F	A	F	A	F	A	F	A	F	A	F	A	F	A	F	A	F
mp2	CFOUR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	!	!
	CAMESS-DDI	✓	✓	✓	✓			✓	✓	✓	✓								
	CAMESS-IMS	✓	✓	✓	✓														
	CAMESS-SERIAL	✓	✓	✓	✓			✓	✓	✓	✓			✓	✓	!	!		
	NWChem-DIRECTMP2	✓	✓																
	NWChem-MP2GRAD	✓	✓	✓	✓			✓	✓	✓	✓								
	NWChem-TCE	✓	✓					✓	✓					!	!	!	!		
	Psi4-FNOCC	✓	✓																
	Psi4-OCC	✓	✓	✓				✓	✓	✓				✓	✓				
ccsd(t)	CFOUR-ECC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	CFOUR-NCC	✓	✓	✓	!														
	CFOUR-VCC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	CAMESS	✓	✓																
	NWChem-CC	✓	✓																
	NWChem-TCE	✓	✓					✓	✓					!	!	!			
	Psi4-CCENERGY	✓	✓					✓	✓					✓	✓				
	Psi4-FNOCC	✓	✓																
	Psi4-MRCC	✓	✓					✓	✓					✓	✓				

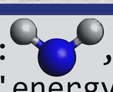
[1] Active orbital values to the right: all-electron A and frozen-core F.

One can add enough detail to **AtomicInput** and extract enough detail from **AtomicResult.properties** to check these programs, methods, and derivatives (and many others) match.

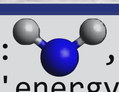
DATA LAYOUT TRANSLATIO … ATION

MOL & DRIVER STANDARDIZATIO

QCSchema: AtomicInput

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'aug-pvdz'    },
  'keywords': {
    'reference': 'rohf'
                                  }}
```

```
{ 'molec
  'drive
  'model
    'me
    'ba
  'keyw
    'co
    'co
    'cc
```

```
{ 'molecule':  ,
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'aug-cc-pvdz'},
  'keywords': {
    'basis__spherical': True,
    'scf__rohf': True,
    'qc_module': 'tce'    }}
```

APP: Psi4, pyVPT2, opt

using QCSchema for generic access to single-points

- Greatest use of QCENGINE is **ABSTRACTING** access to single-point computations.
- Even though PSI4 had internal interfaces for dispersion codes at one time, it's easier to use QCENGINE as an **INTERMEDIARY**.
- Geometry **OPTIMIZERS** can work with abstract gradients rather than within specific codes.
- **PYVPT2**, an anharmonic vibrational frequency project can use arbitrary sources for Hessians

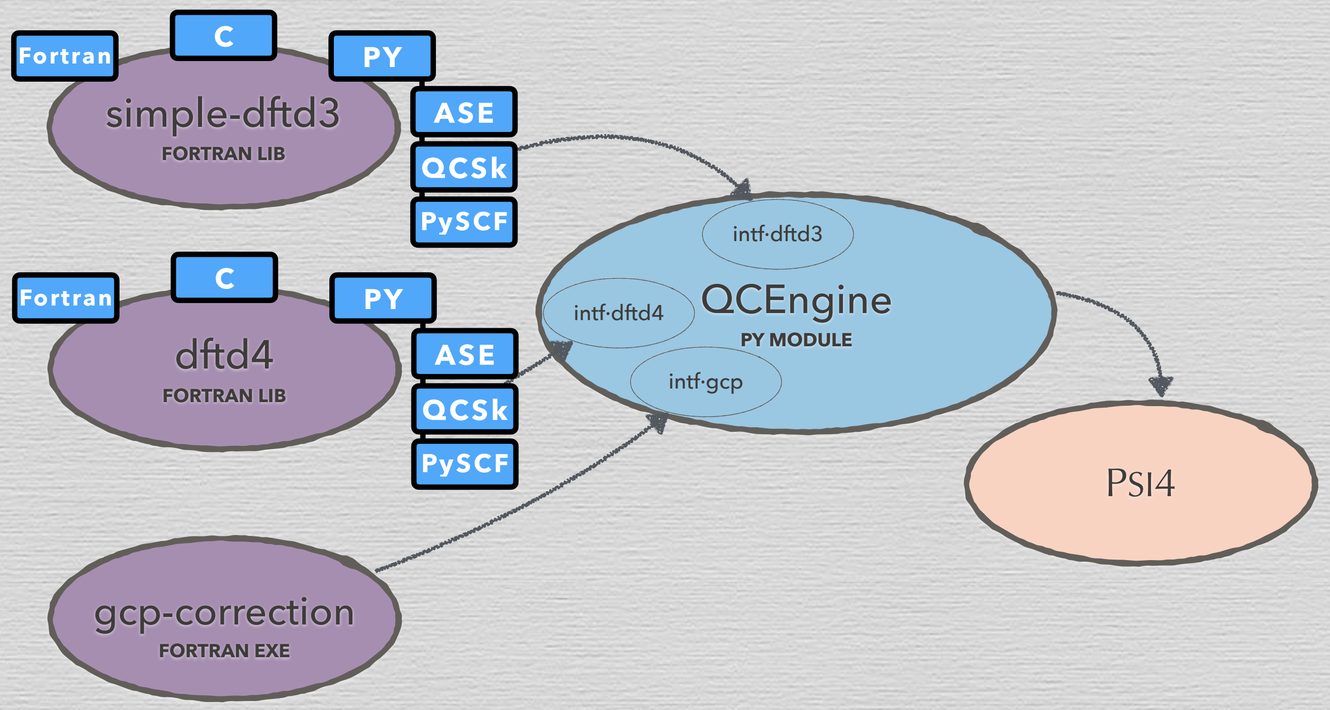


TABLE I. PYVPT2 harmonic and fundamental frequencies (cm^{-1}) obtained with different QC programs, for the H_2O molecules at the CCSD(T)/aug-cc-pVTZ level of theory

	ω			ν		
Mode	PSI4	CFOUR	NWCHEM	PSI4	CFOUR	NWCHEM
1	1646.0	1646.0	1646.0	1595.7	1595.8	1595.8
2	3810.6	3810.5	3810.6	3638.2	3638.0	3637.1
3	3919.7	3919.6	3919.6	3733.3	3733.1	3733.0

P. M. Nelson and C. D. Sherrill, pyVPT2, in review.

APP: QISKIT & ACCELERATED DFT

transporting single-point records with AtomicInput/Result

APP: QISKIT & ACCELERATED DFT

transporting single-point records with AtomicInput/Result

- IBM **QISKIT** for Quantum Computing
- QCSchema used to **TRANSFER** data between the classical computation (i.e., quantum chemistry) & Qiskit Nature
- **EXTENDED** schema with ERIs

Qiskit | Ecosystem

Qiskit Nature 0.7.2

```
from qiskit_nature.second_q.drivers import PySCFDriver

driver = PySCFDriver(atom="H 0 0 0; H 0 0 0.735", basis="sto3g",
    charge=0, spin=0, unit=DistanceUnit.ANGSTROM,)
problem = driver.run()

print(problem)
#> <qiskit_nature.second_q...ElectronicStructureProblem object>

qcschema = driver.to_qcschema()
```

https://qiskit-community.github.io/qiskit-nature/tutorials/08_qcschema.html

APP: QISKIT & ACCELERATED DFT

transporting single-point records with AtomicInput/Result

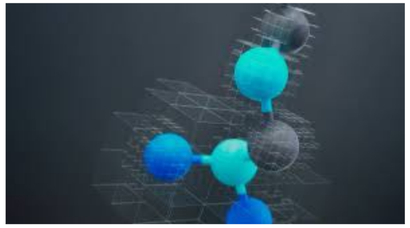
- IBM **QISKIT** for Quantum Computing
- QCSchema used to **TRANSFER** data between the classical computation (i.e., quantum chemistry) & Qiskit Nature
- **EXTENDED** schema with ERIs

Qiskit | Ecosystem

Qiskit Nature 0.7.2

```python
from qiskit_nature.second_q.drivers import PySCFDriver

driver = PySCFDriver(atom="H 0 0 0; H 0 0 0.735", basis="sto3g",
    charge=0, spin=0, unit=DistanceUnit.ANGSTROM,)
problem = driver.run()

print(problem)
#> <qiskit_nature.second_q...ElectronicStructureProblem object>

qcschema = driver.to_qcschema()
```

- Microsoft **ACCELERATED DFT** GPU SaaS
- QCSchema & QCElemental used for **I/O** standardization & serialization
- QCEngine used for DFT **EXECUTION** abstraction
- **MODIFIED** schema usage to accept an XYZ file as an alternative to **Molecule** model
- **PARTITIONED** schema so that large native files like checkpoint remain in the cloud, with a reference stored in the schema response



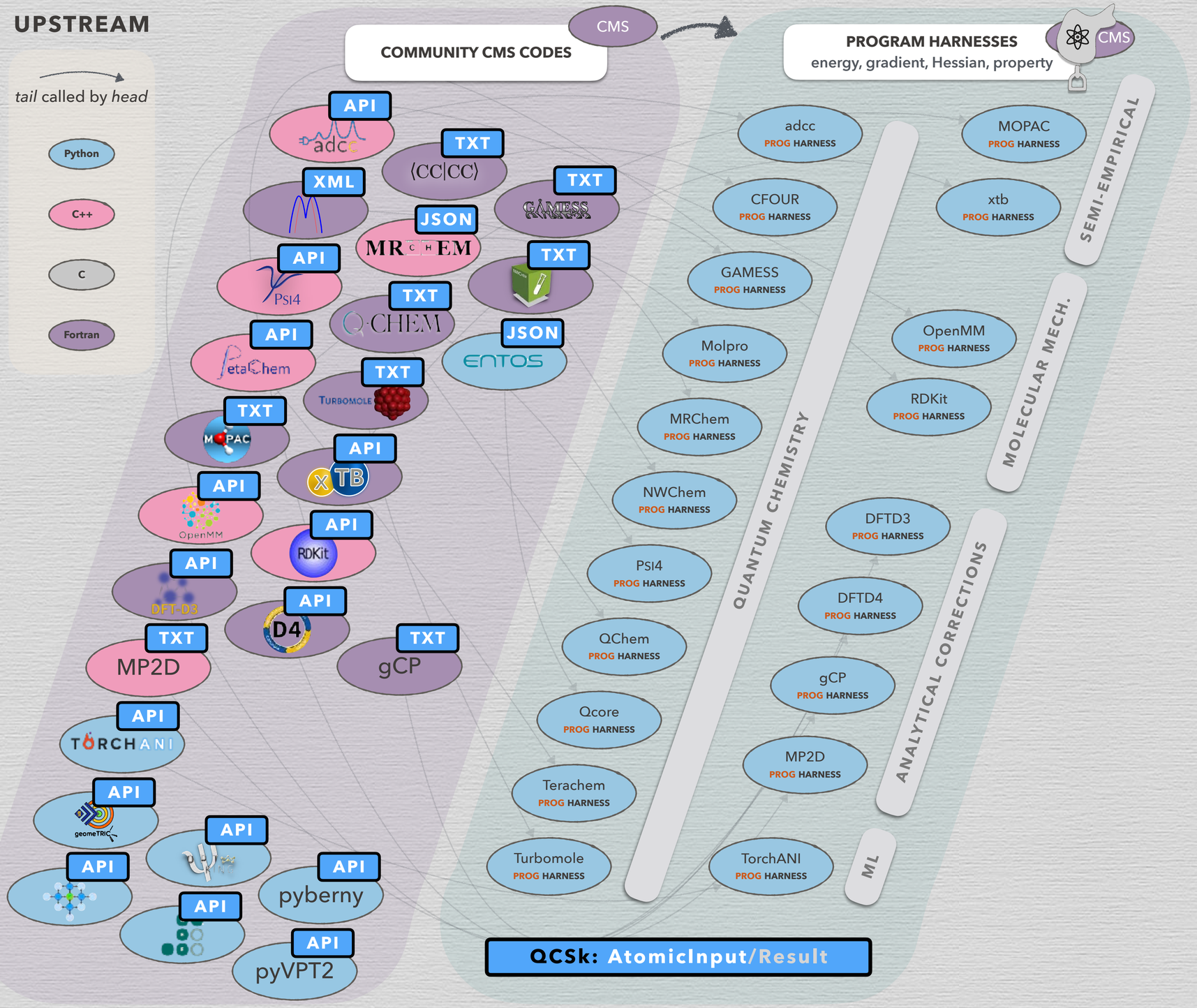
```python
from azure.quantum import Workspace

workspace = Workspace.from_connection_string("access_key")
target = workspace.get_targets("microsoft.dft")
job = target.submit(
    input_data=Path("./ubiquitin.qcschema").read_text(),
    input_data_format="microsoft.qc-schema.v1",
    name="ubiquitin-spe")

job.wait_until_completed()
output = job.get_results()
```

https://arxiv.org/pdf/2406.11185v1

https://qiskit-community.github.io/qiskit-nature/tutorials/08_qcschema.html

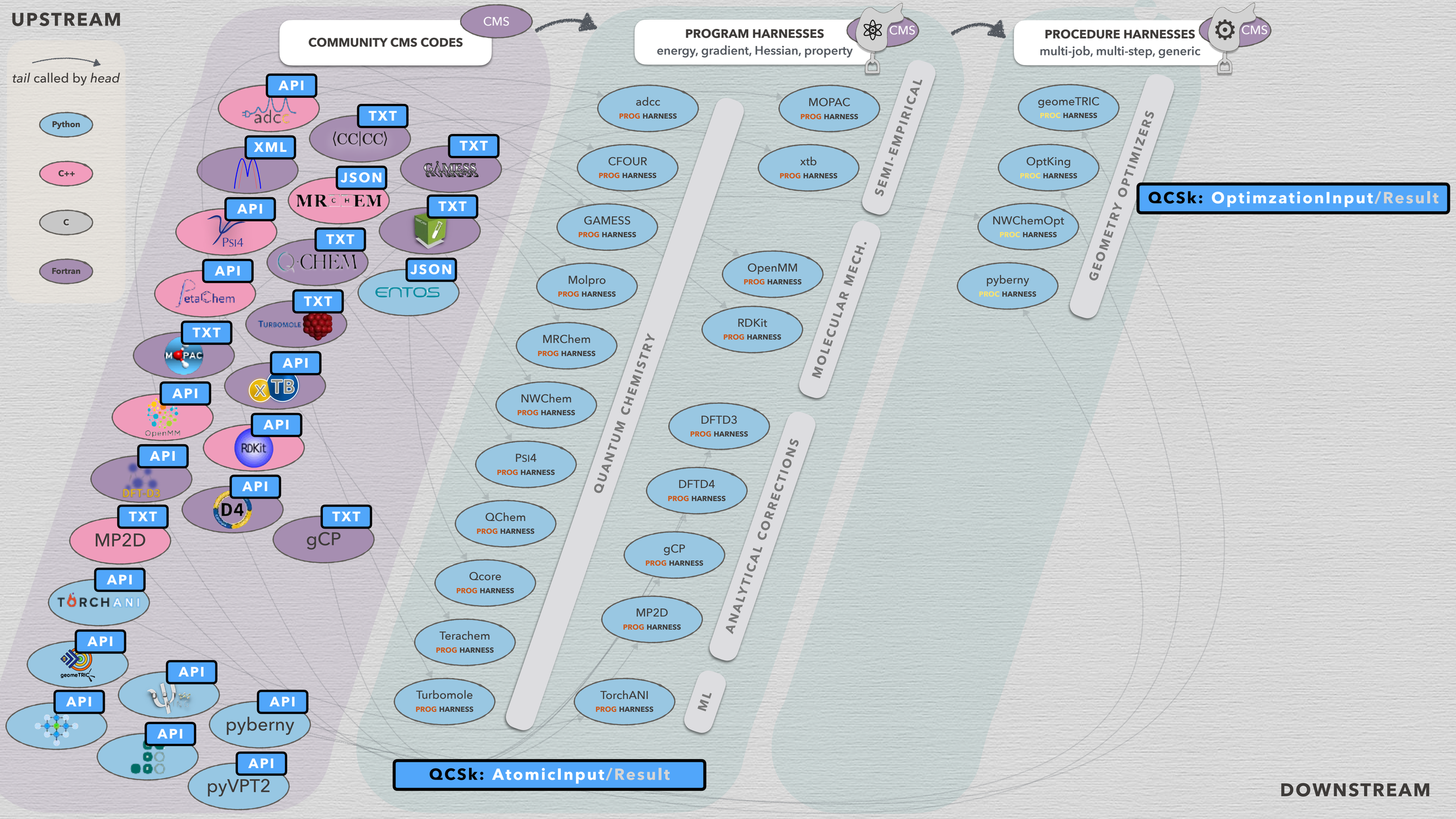
QCEngine: INTERNAL MAP

github.com/MolSSI/QCEngine



- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. So far, mostly optimizers: geometric, optking, pyberny, NWChem (int.).
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety of ways from API to structured data to regex. Former preferred for numerical precision.

DOWNSTREAM


UPSTREAM
tail called by head
Python
C++
C
Fortran
COMMUNITY CMS CODES
CMS
API
adcc
TXT
CC
XML
TXT
GAMESS
JSON
MRCHEM
API
PSI4
TXT
TXT
Q-CHEM
JSON
ENTOS
API
PetaChem
TXT
TURBOMOLE
TXT
MOPAC
API
xTB
API
OpenMM
API
RDKit
API
DFT-D3
API
D4
TXT
MP2D
TXT
gCP
API
TORCHANI
API
geomeTRIC
API
API
API
pyberny
API
pyVPT2
PROGRAM HARNESSES
energy, gradient, Hessian, property
CMS
adcc PROG HARNESS
CFOUR PROG HARNESS
GAMESS PROG HARNESS
Molpro PROG HARNESS
MRChem PROG HARNESS
NWChem PROG HARNESS
PSI4 PROG HARNESS
QChem PROG HARNESS
Qcore PROG HARNESS
Terachem PROG HARNESS
Turbomole PROG HARNESS
QUANTUM CHEMISTRY
MOPAC PROG HARNESS
xtb PROG HARNESS
SEMI-EMPIRICAL
OpenMM PROG HARNESS
RDKit PROG HARNESS
MOLECULAR MECH.
DFTD3 PROG HARNESS
DFTD4 PROG HARNESS
gCP PROG HARNESS
MP2D PROG HARNESS
ANALYTICAL CORRECTIONS
TorchANI PROG HARNESS
ML
QCSk: AtomicInput/Result
PROCEDURE HARNESSES
multi-job, multi-step, generic
CMS
geomeTRIC PROC HARNESS
OptKing PROC HARNESS
NWChemOpt PROC HARNESS
pyberny PROC HARNESS
GEOMETRY OPTIMIZERS
QCSk: OptimzationInput/Result
DOWNSTREAM

QCSchema Optimization

driving geometry optimizations

- **OptimizationInput** & **OptimizationResult** are job directives for an abstract generic optimizer calling generic QC programs for gradient computations.
- Geometry optimizations are the classic case of **SEPARABLE** code (next geometry step vs. gradient supplier) that traditionally have been tied within QC programs for user convenience.
- QCEngine can use **STANDALONE** CMS optimizers but also has abstracted in-program optimizers (NWChem; soon, Gaussian), though restricted to use native gradients
- **INPUT** keywords are not standardized
- **OUTPUT** results are not standardized, though we hope to add **OptimizationProperties** soon for summary data like RMS gradient
- **ERROR CORRECTION** in QCEngine has to potential to be particularly useful for optimizations
- Note no room for optimization program

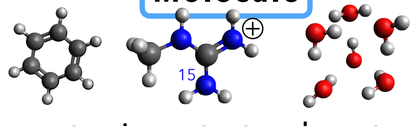
RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

OptimizationInput

schema_version: 1

initial_molecule: **Molecule**



keywords: knobs in the geo opt program

`{coordsys: tric, program:` Psi4 MOPAC `}`

input_specification: how to run *gradients* for opt

QCSchema Optimization

driving geometry optimizations

- **OptimizationInput** & **OptimizationResult** are job directives for an abstract generic optimizer calling generic QC programs for gradient computations.
- Geometry optimizations are the classic case of **SEPARABLE** code (next geometry step vs. gradient supplier) that traditionally have been tied within QC programs for user convenience.
- QCEngine can use **STANDALONE** CMS optimizers but also has abstracted in-program optimizers (NWChem; soon, Gaussian), though restricted to use native gradients
- **INPUT** keywords are not standardized
- **OUTPUT** results are not standardized, though we hope to add **OptimizationProperties** soon for summary data like RMS gradient
- **ERROR CORRECTION** in QCEngine has to potential to be particularly useful for optimizations
- Note no room for optimization program

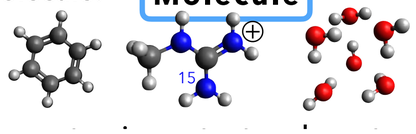
RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

OptimizationInput

schema_version: 1

initial_molecule: **Molecule**



keywords: knobs in the geo opt program

`{coordsys: tric, program: }`

input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage

driver: single-point derivative

- **energy**
- **gradient** *E* *G* *H*
- **Hessian**
- **properties**

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

QCSchema Optimization

driving geometry optimizations

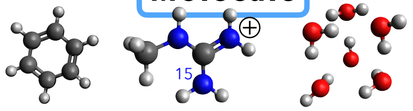
- **OptimizationInput** & **OptimizationResult** are job directives for an abstract generic optimizer calling generic QC programs for gradient computations.
- Geometry optimizations are the classic case of **SEPARABLE** code (next geometry step vs. gradient supplier) that traditionally have been tied within QC programs for user convenience.
- QCEngine can use **STANDALONE** CMS optimizers but also has abstracted in-program optimizers (NWChem; soon, Gaussian), though restricted to use native gradients
- **INPUT** keywords are not standardized
- **OUTPUT** results are not standardized, though we hope to add **OptimizationProperties** soon for summary data like RMS gradient
- **ERROR CORRECTION** in QCEngine has to potential to be particularly useful for optimizations
- Note no room for optimization program

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

OptimizationInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
initial_molecule: **Molecule**



protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program
`{coordsys: tric, program:` `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1
extras: free-form storage
driver: single-point derivative
- **energy**
- **gradient**
- **Hessian**
- **properties**

E G H

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

QCSchema Optimization

driving geometry optimizations

- **OptimizationInput** & **OptimizationResult** are job directives for an abstract generic optimizer calling generic QC programs for gradient computations.
- Geometry optimizations are the classic case of **SEPARABLE** code (next geometry step vs. gradient supplier) that traditionally have been tied within QC programs for user convenience.
- QCEngine can use **STANDALONE** CMS optimizers but also has abstracted in-program optimizers (NWChem; soon, Gaussian), though restricted to use native gradients
- **INPUT** keywords are not standardized
- **OUTPUT** results are not standardized, though we hope to add **OptimizationProperties** soon for summary data like RMS gradient
- **ERROR CORRECTION** in QCEngine has to potential to be particularly useful for optimizations
- Note no room for optimization program

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

OptimizationResult

OptimizationInput

schema_version: 1

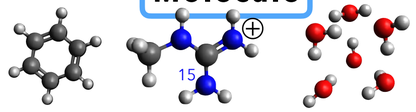
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program:` }

input_specification: how to run *gradients* for opt

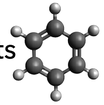
QCInputSpecification

stdout: Opt program's text output for humans

stderr: anything in error channel

extras: *(merges with input)*

final_molecule: optimized geom. for results



energies: energy for opt steps

[*E* ·]

trajectory: full results for opt steps

[**AtomicResult**]

OPTIMIZERS

use QCSchema as API

- **GEOMETRIC** by Lee-Ping Wang features full and constrained optimizations, transition states, and conical intersections. NO CI for the QCEngine API
- **OPTKING** by Rollin King features full and constrained optimizations, transition states, and IRC. No IRC for the QCEngine API
- Both have a QCSchema interface to **OptimizationInput**/**Result**, which uses QCEngine to compute gradients
- Alternatively, both also have QCEngine harnesses to drive the optimizer, which in turn calls QCEngine for the gradients

OptimizationInput

```
optin = {
  "keywords": {
    "program": "psi4",
    # constraints for geomeTRIC
   "constraints":{"set":[{"type":"distance","indices":[1,2],"value": 2.4}]}},
  "initial_molecule": {
    "geometry": [0, 0, 0.28, 0, 1.43, 0.84, 0, -1.43, 0.84],
    "symbols": ["O", "H", "H"]},
  "input_specification": {
    "model": {"method": "hf", "basis": "sto-3g"},
    "keywords": {"d_convergence": 10, "scf_type": "pk"}}
}

inmol = qcelemental.models.Molecule(**optin["initial_molecule"])
print(inmol.nuclear_repulsion_energy())
#> 10.7681
```

GEOMETRIC RUNNING QCSchema

```
import geometric

optres = geometric.run_json.geometric_run_json(optin)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 9.3595  # w/constraint
```

QCEngine RUNNING GEOMETRIC

```
import qcengine

optres = qcengine.compute_procedure(optin, "geometric", return_dict=True)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 9.3595
```

OPTKING RUNNING QCSchema

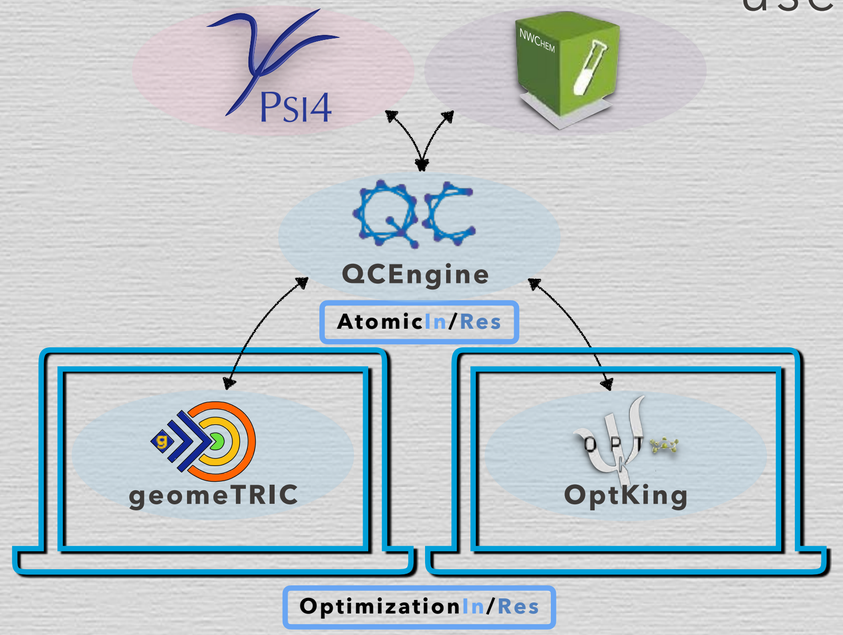
```
import optking

optres = optking.optimize_qcengine(optin)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 8.9061
```

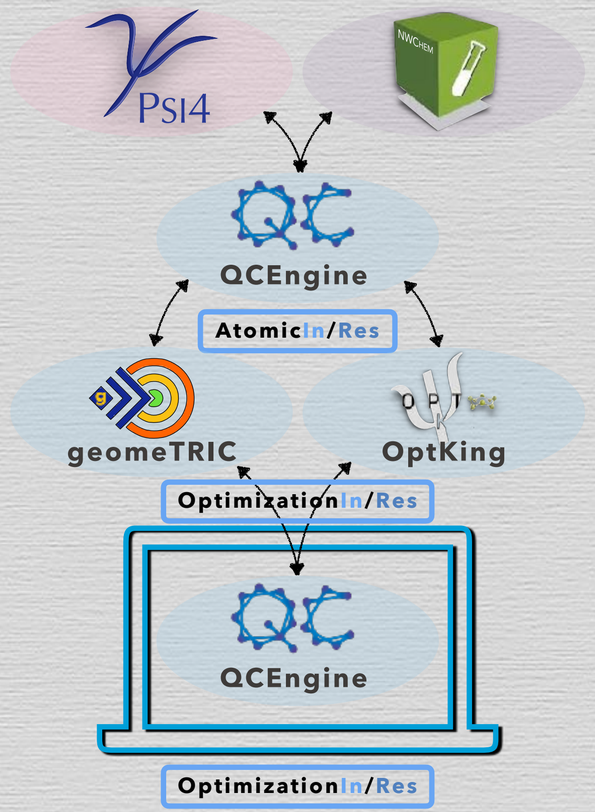
QCEngine RUNNING OPTKING

```
import qcengine

optres = qcengine.compute_procedure(optin, "optking", return_dict=True)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 8.9061
```

OPTIMIZERS

use QCSchema as API





GEOMETRIC RUNNING QCSchema

```
import geometric

optres = geometric.run_json.geometric_run_json(optin)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 9.3595  # w/constraint
```

QCEngine RUNNING GEOMETRIC

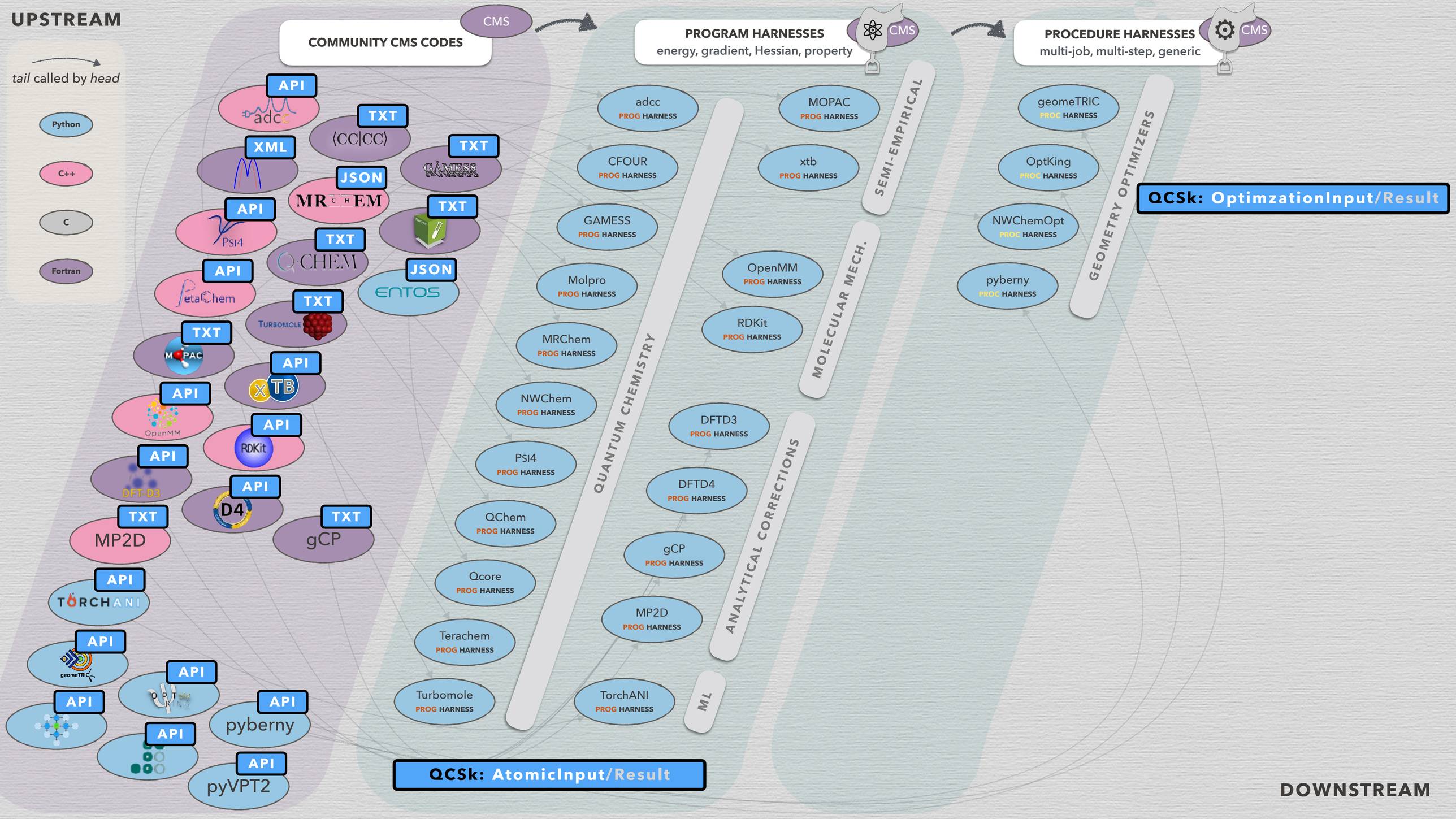
```
import qcengine

optres = qcengine.compute_procedure(optin, "geometric", return_dict=True)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 9.3595
```

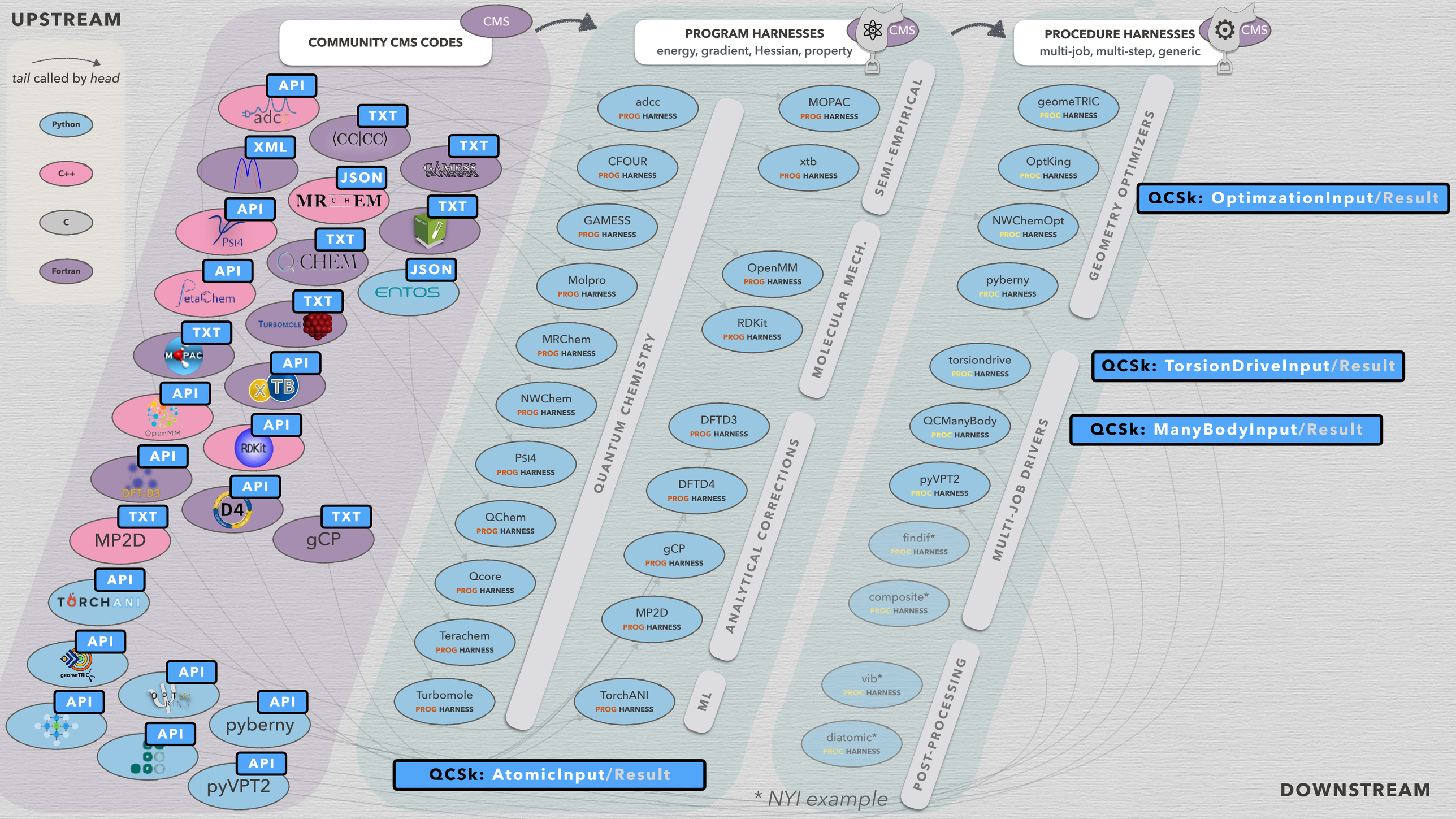
OPTKING RUNNING QCSchema

```
import optking

optres = optking.optimize_qcengine(optin)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 8.9061
```

QCEngine RUNNING OPTKING

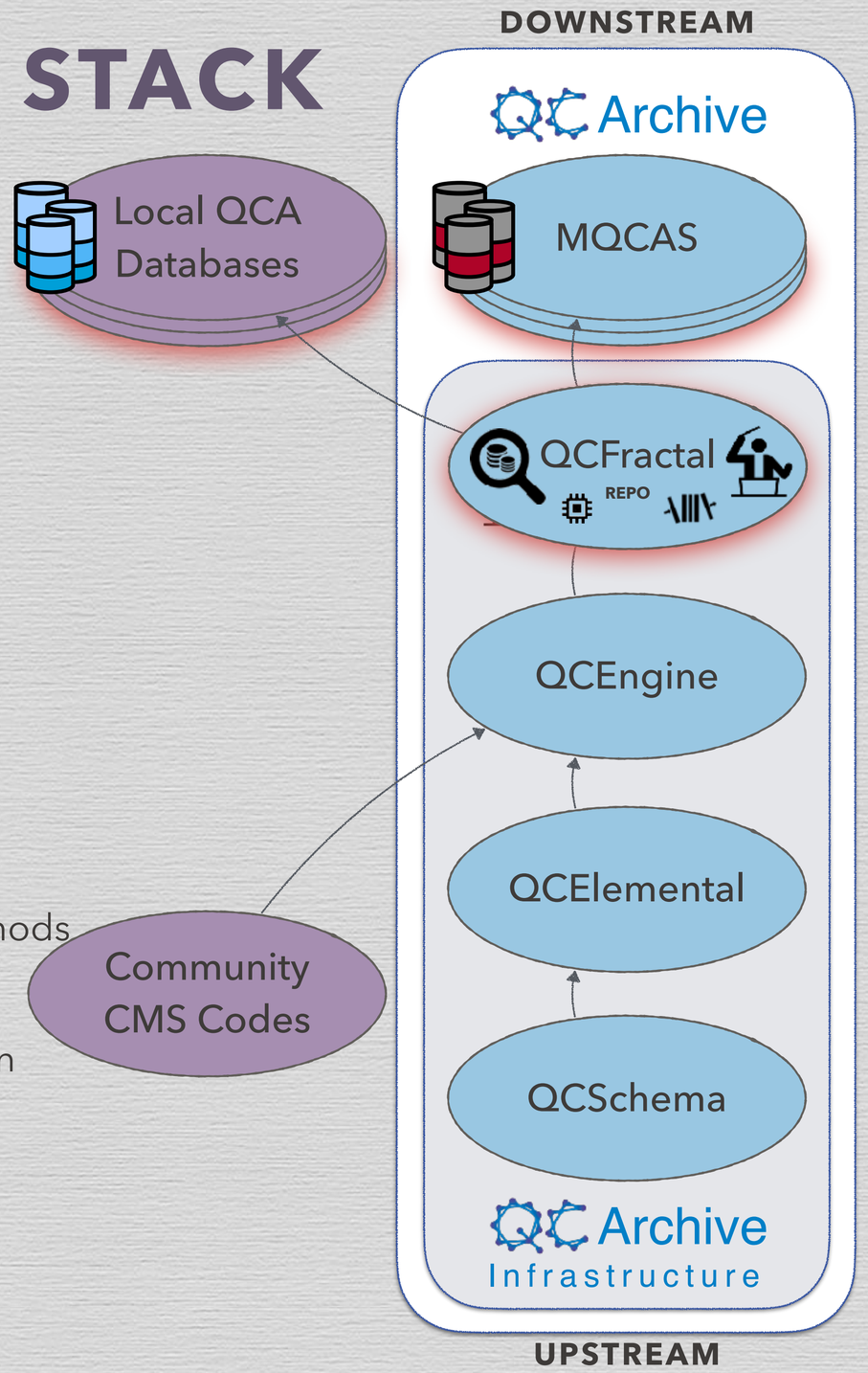
```
import qcengine

optres = qcengine.compute_procedure(optin, "optking", return_dict=True)

print(optres["trajectory"][-1]["properties"]["nuclear_repulsion_energy"])
#> 8.9061
```


UPSTREAM
tail called by head
Python
C++
C
Fortran
COMMUNITY CMS CODES
CMS
API
adcc
TXT
CC
XML
TXT
GAMESS
JSON
MRCHEM
API
PSI4
TXT
TXT
Q-CHEM
JSON
ENTOS
API
PetaChem
TXT
TURBOMOLE
TXT
MOPAC
API
xTB
API
OpenMM
API
RDKit
API
DFT-D3
API
D4
TXT
MP2D
TXT
gCP
API
TORCHANI
API
geomeTRIC
API
OptKing
API
API
pyberny
API
pyVPT2
PROGRAM HARNESSES
energy, gradient, Hessian, property
CMS
adcc PROG HARNESS
CFOUR PROG HARNESS
GAMESS PROG HARNESS
Molpro PROG HARNESS
MRChem PROG HARNESS
NWChem PROG HARNESS
PSI4 PROG HARNESS
QChem PROG HARNESS
Qcore PROG HARNESS
Terachem PROG HARNESS
Turbomole PROG HARNESS
QUANTUM CHEMISTRY
MOPAC PROG HARNESS
xtb PROG HARNESS
SEMI-EMPIRICAL
OpenMM PROG HARNESS
RDKit PROG HARNESS
MOLECULAR MECH.
DFTD3 PROG HARNESS
DFTD4 PROG HARNESS
gCP PROG HARNESS
MP2D PROG HARNESS
ANALYTICAL CORRECTIONS
TorchANI PROG HARNESS
ML
QCSk: AtomicInput/Result
PROCEDURE HARNESSES
multi-job, multi-step, generic
CMS
geomeTRIC PROC HARNESS
OptKing PROC HARNESS
NWChemOpt PROC HARNESS
pyberny PROC HARNESS
GEOMETRY OPTIMIZERS
QCSk: OptimzationInput/Result
DOWNSTREAM


UPSTREAM
tail called by head
Python
C++
C
Fortran
COMMUNITY CMS CODES
CMS
PROGRAM HARNESSES
energy, gradient, Hessian, property
CMS
PROCEDURE HARNESSES
multi-job, multi-step, generic
CMS
API
adcc
TXT
CC
XML
TXT
GAMESS
JSON
MRCHEM
API
PSI4
TXT
Q-CHEM
TXT
API
PetaChem
JSON
ENTOS
TXT
TURBOMOLE
TXT
MOPAC
API
XTB
API
OpenMM
API
RDKit
API
DFT-D3
API
D4
TXT
MP2D
TXT
gCP
API
TORCHANI
API
geometRIC
API
OptKing
API
API
pyberny
API
pyVPT2
adcc PROG HARNESS
CFOUR PROG HARNESS
GAMESS PROG HARNESS
Molpro PROG HARNESS
MRChem PROG HARNESS
NWChem PROG HARNESS
PSI4 PROG HARNESS
QChem PROG HARNESS
Qcore PROG HARNESS
Terachem PROG HARNESS
Turbomole PROG HARNESS
QUANTUM CHEMISTRY
MOPAC PROG HARNESS
xtb PROG HARNESS
SEMI-EMPIRICAL
OpenMM PROG HARNESS
RDKit PROG HARNESS
MOLECULAR MECH.
DFTD3 PROG HARNESS
DFTD4 PROG HARNESS
gCP PROG HARNESS
MP2D PROG HARNESS
ANALYTICAL CORRECTIONS
TorchANI PROG HARNESS
ML
geomeTRIC PROC HARNESS
OptKing PROC HARNESS
NWChemOpt PROC HARNESS
pyberny PROC HARNESS
GEOMETRY OPTIMIZERS
torsiondrive PROC HARNESS
QCManyBody PROC HARNESS
pyVPT2 PROC HARNESS
findif* PROC HARNESS
composite* PROC HARNESS
MULTI-JOB DRIVERS
vib* PROC HARNESS
diatomic* PROC HARNESS
POST-PROCESSING
QCSk: OptimzationInput/Result
QCSk: TorsionDriveInput/Result
QCSk: ManyBodyInput/Result
QCSk: AtomicInput/Result
* NYI example
DOWNSTREAM

QCArchive Stack

- database, permissioned query
- fully powerful as MQCAS but locally controlled



- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- database storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

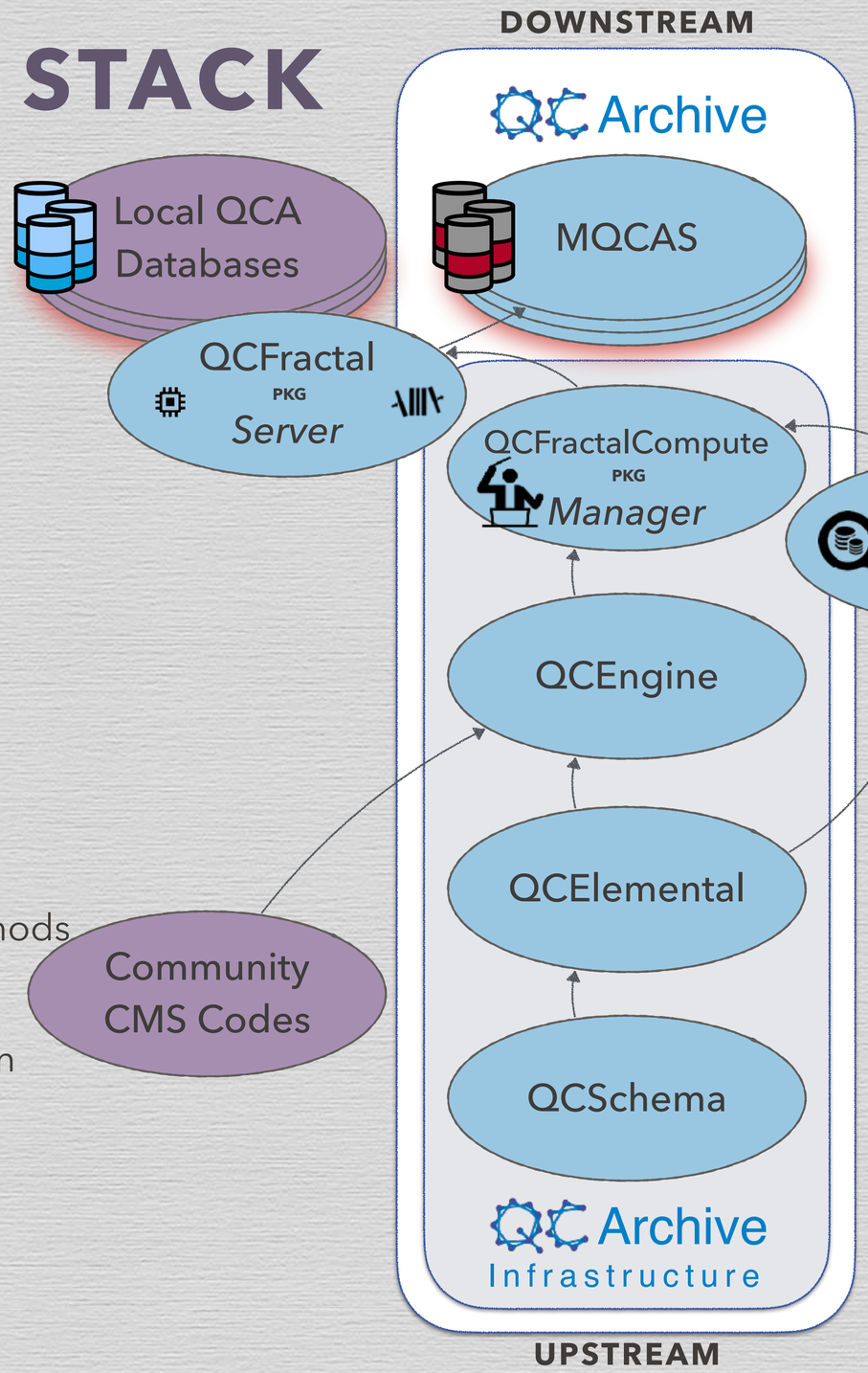
- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

- the difficult part – coded QC methods
- structured output uncommon
- DSL input; API/schema uncommon

QCArchive Stack

- database, permissioned query
- fully powerful as MQCAS but locally controlled



- database, universally queryable for CMS results
- MolSSI QCArchive Server ("em-quacks")

- batch (parallel) compute setup & management
- storage & query of QC results

- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

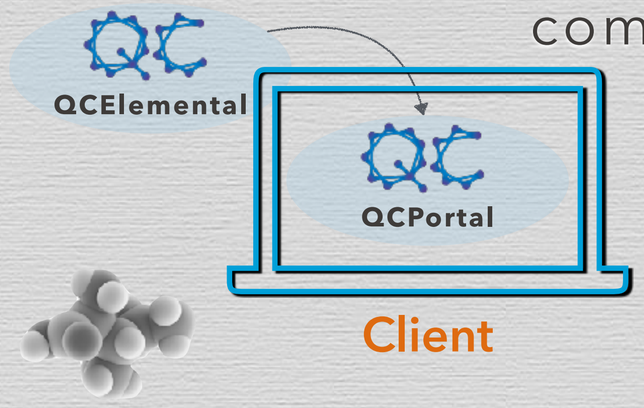
- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

- the difficult part – coded QC methods
- structured output uncommon
- DSL input; API/schema uncommon

QCArchive Overview

compute, analyze, share quantum chemistry data

qcarchive.molssi.org

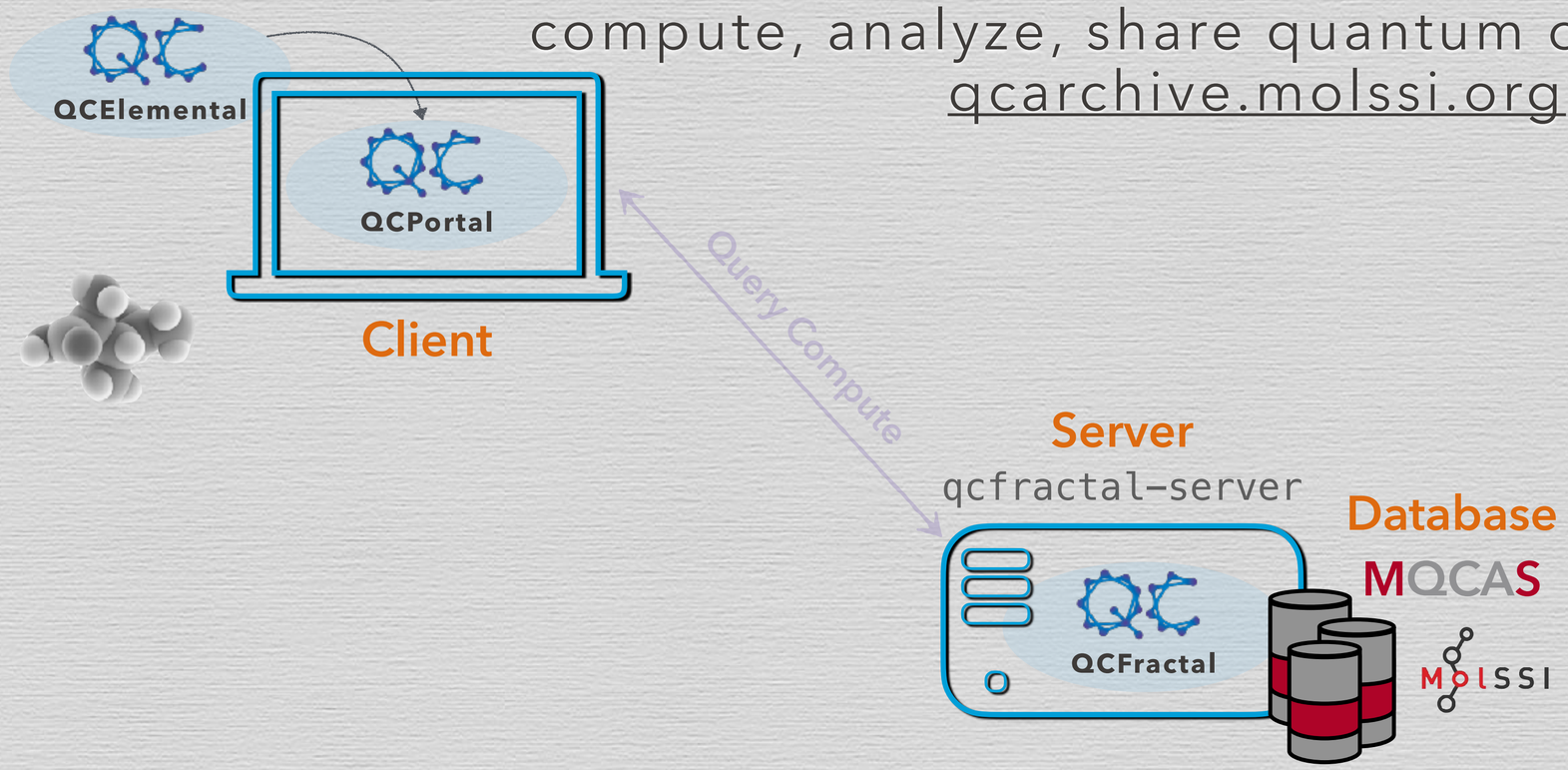


- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org

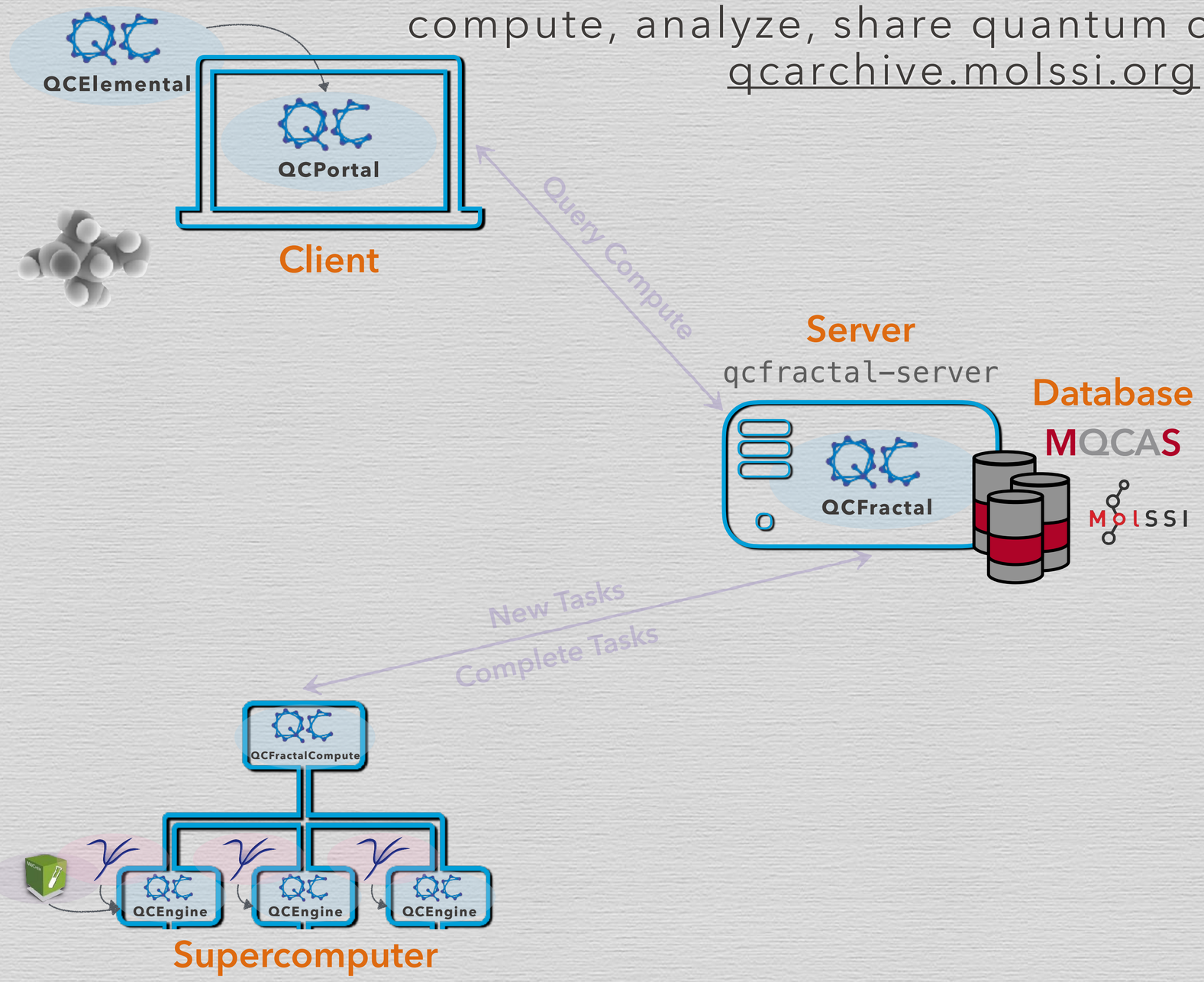


- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

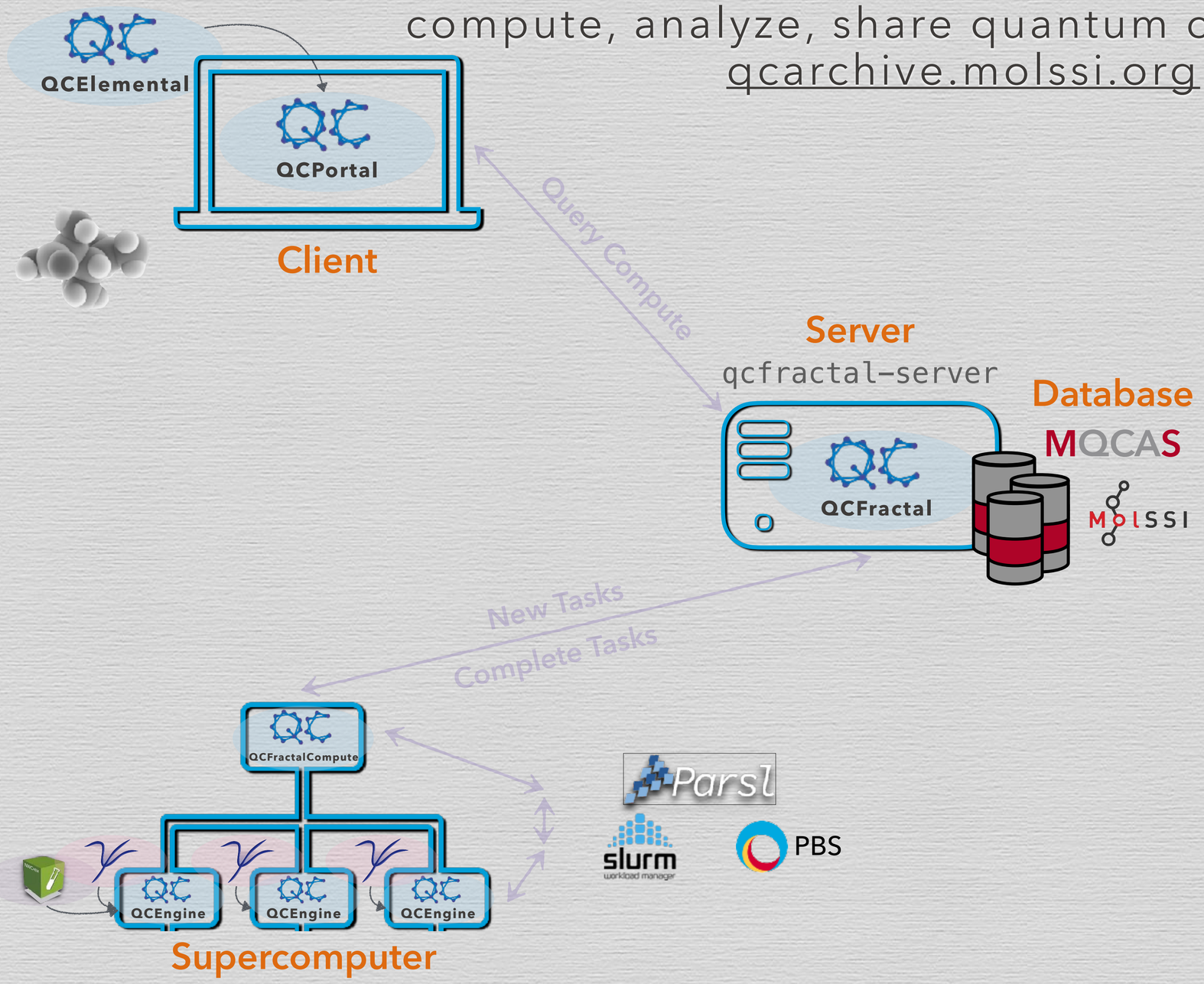
`qcfractal-compute-manager` **Manager**

worker **Compute Nodes**

QCArchive Overview

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



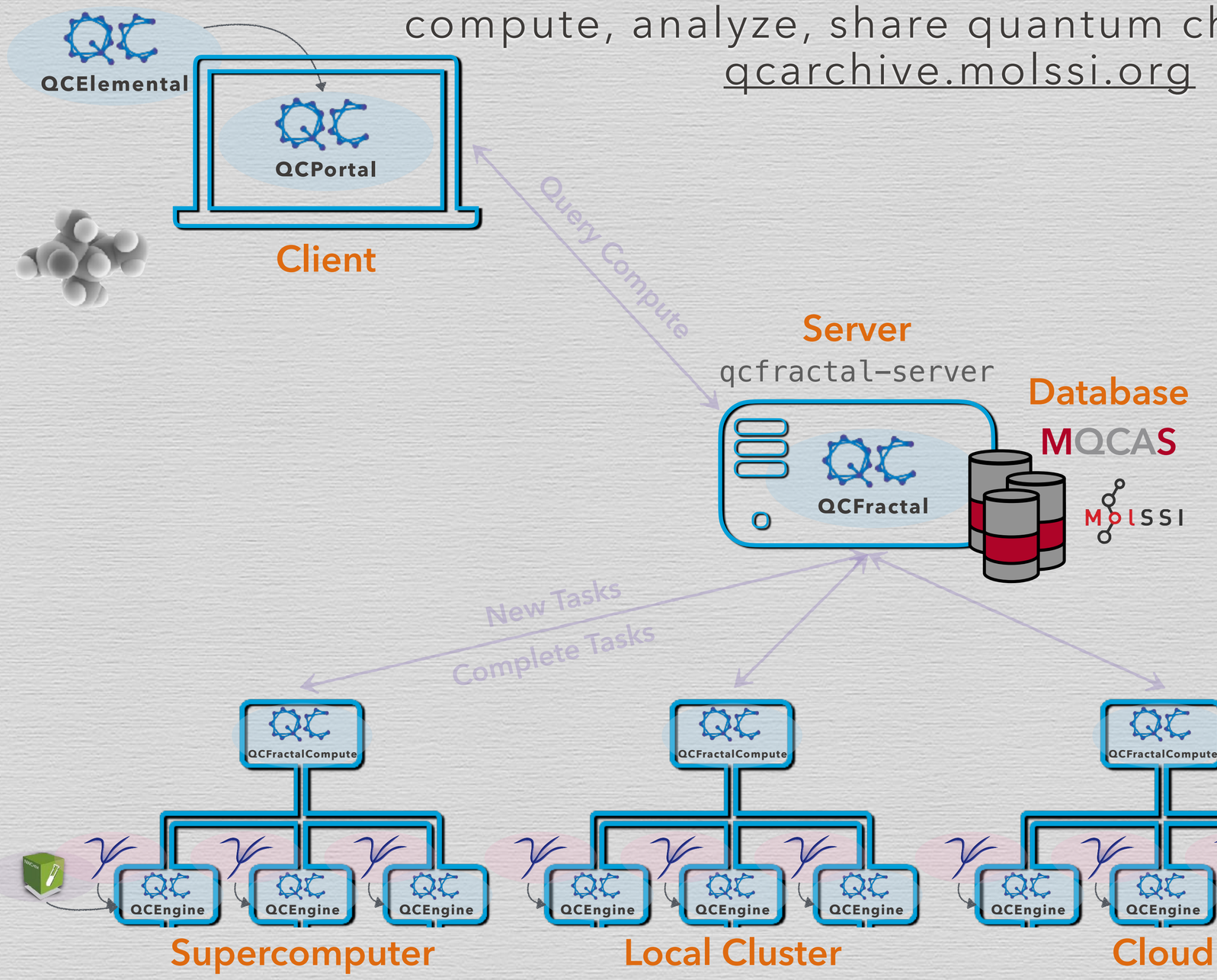
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



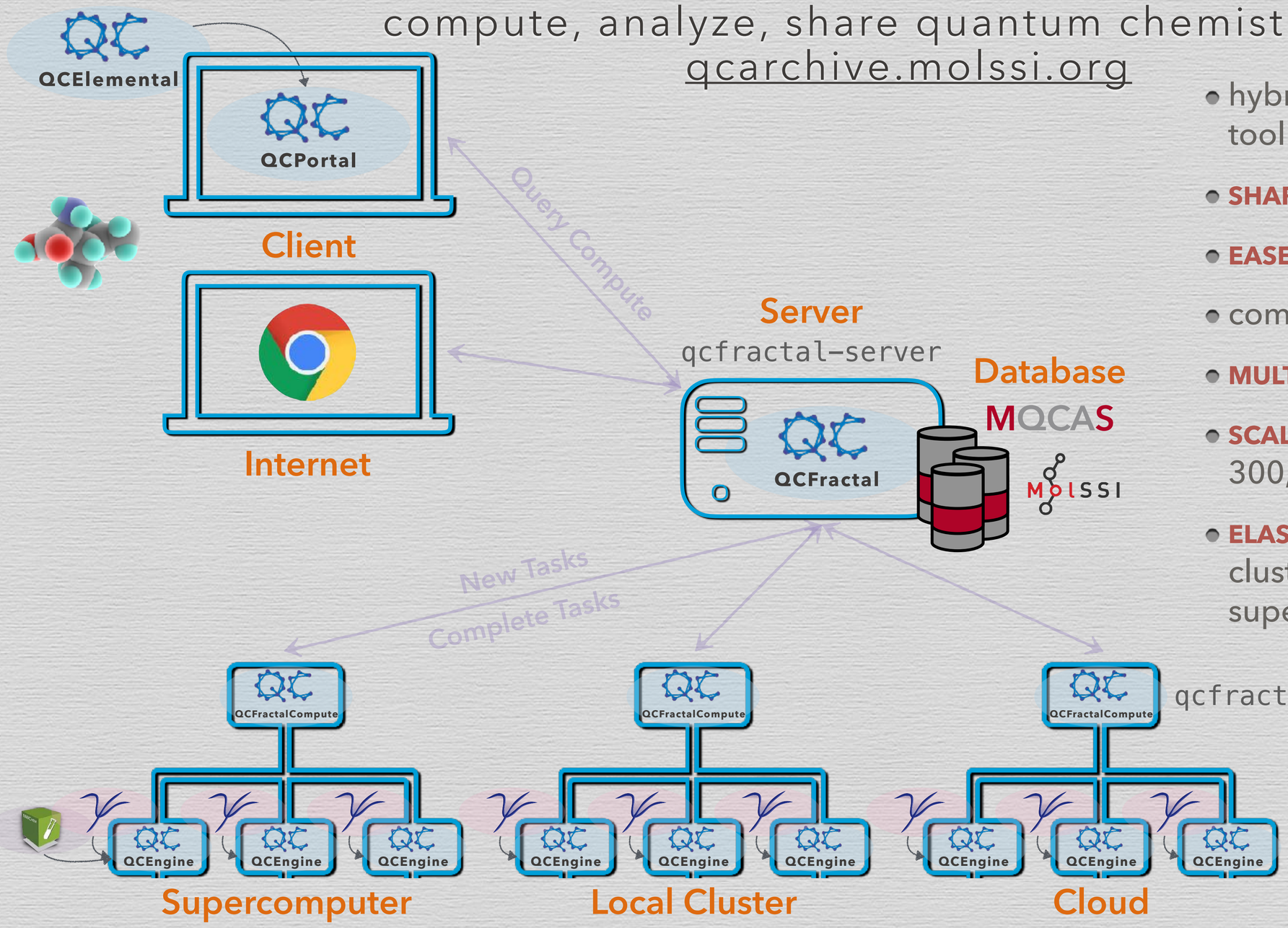
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



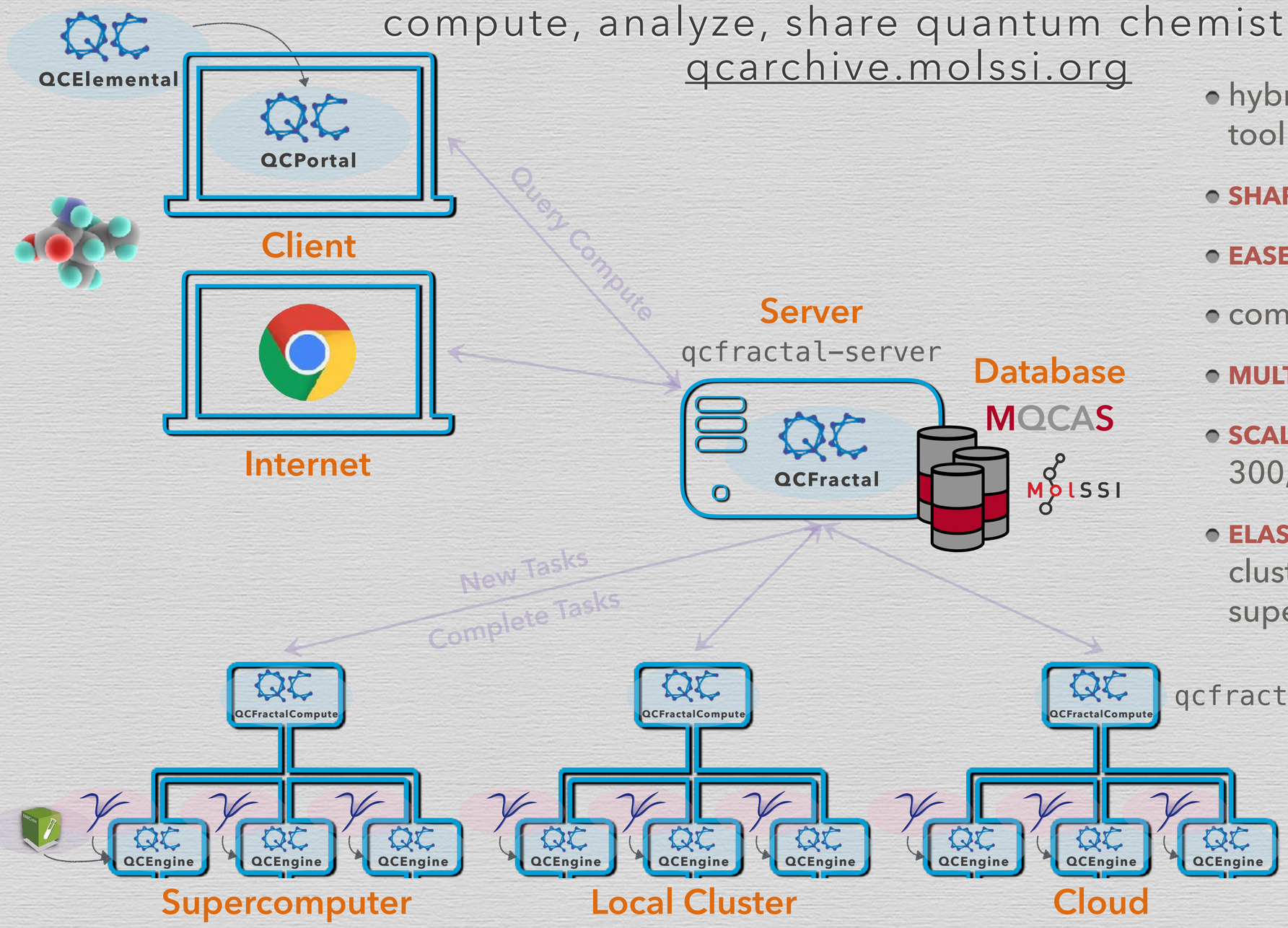
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org

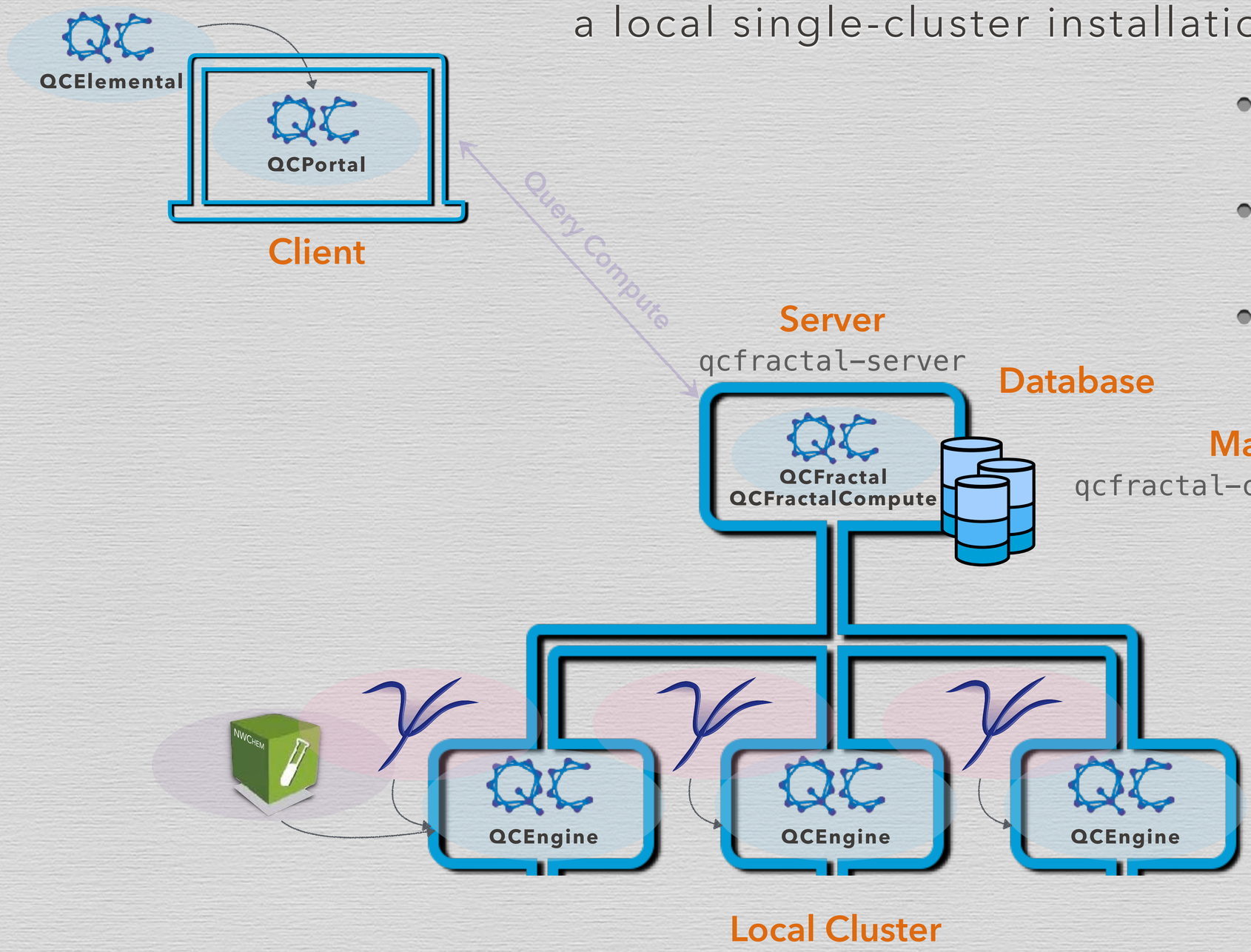


- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive, Smaller

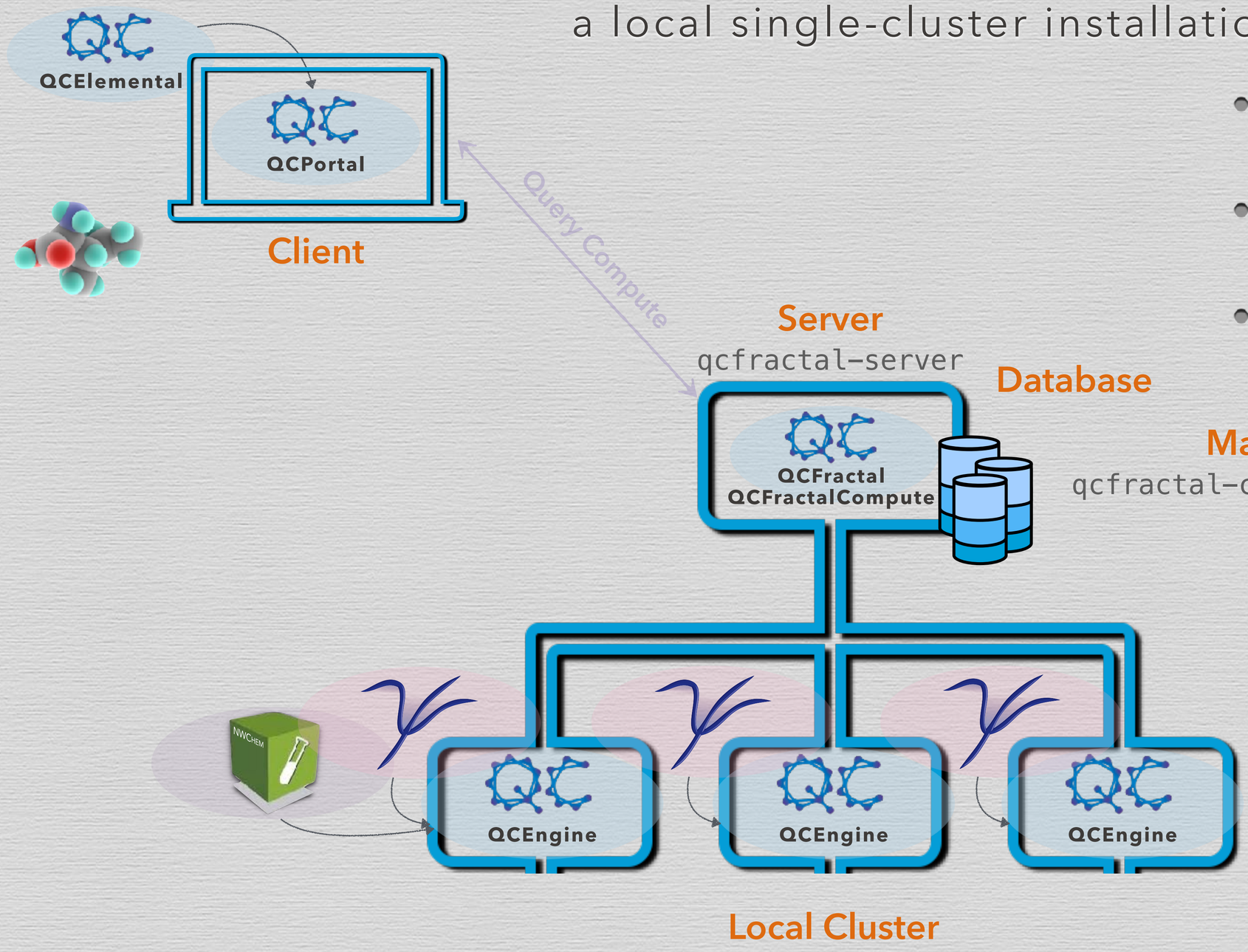
a local single-cluster installation



- **QCARCHIVE INFRASTRUCTURE** software stack unchanged
- **COLOCATE** QCFractal and database for task submission and orchestration
- **LOCAL** database fully powerful as MQCAS but locally controlled

QCArchive, smaller

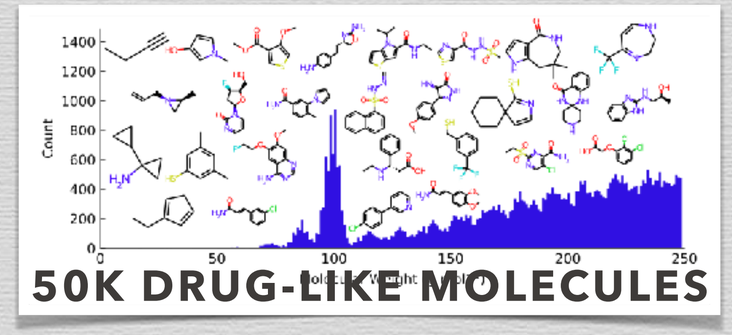
a local single-cluster installation



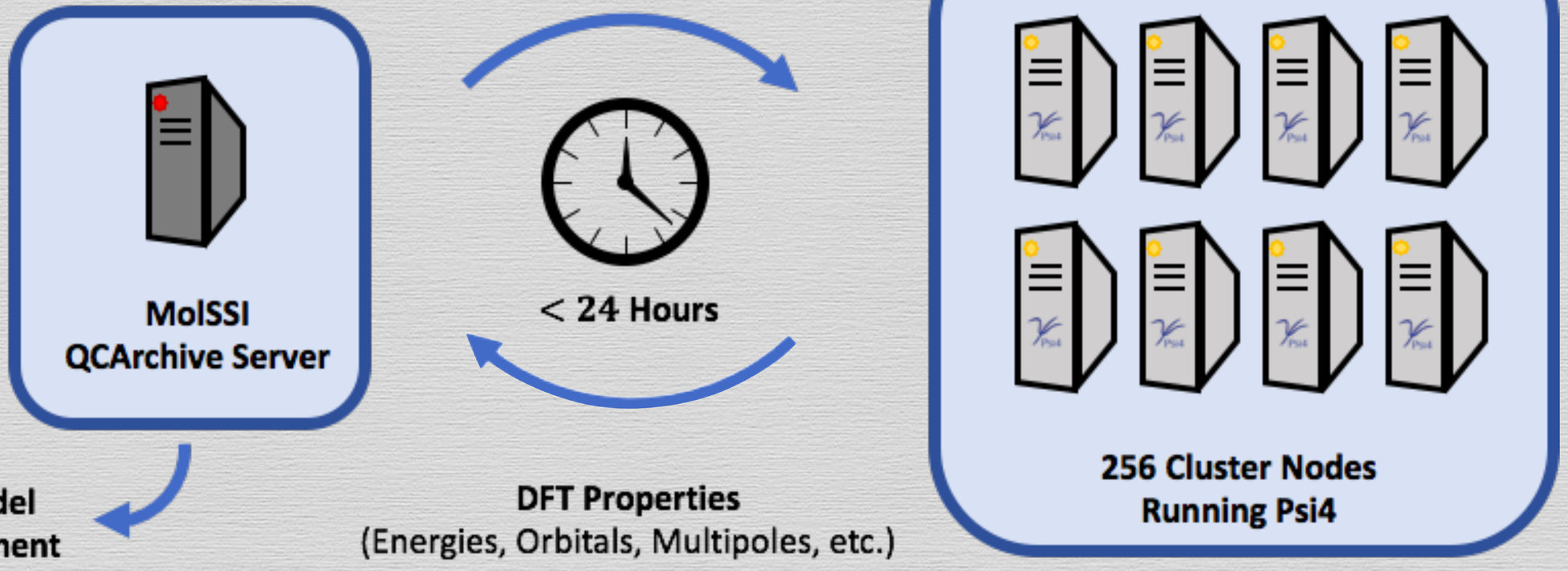
- **QCARCHIVE INFRASTRUCTURE** software stack unchanged
- **COLOCATE** QCFractal and database for task submission and orchestration
- **LOCAL** database fully powerful as MQCAS but locally controlled

QCArchive for Throughput & Reproducibility

HPC message-passing neural network computations preserved on public server



50K DRUG-LIKE MOLECULES



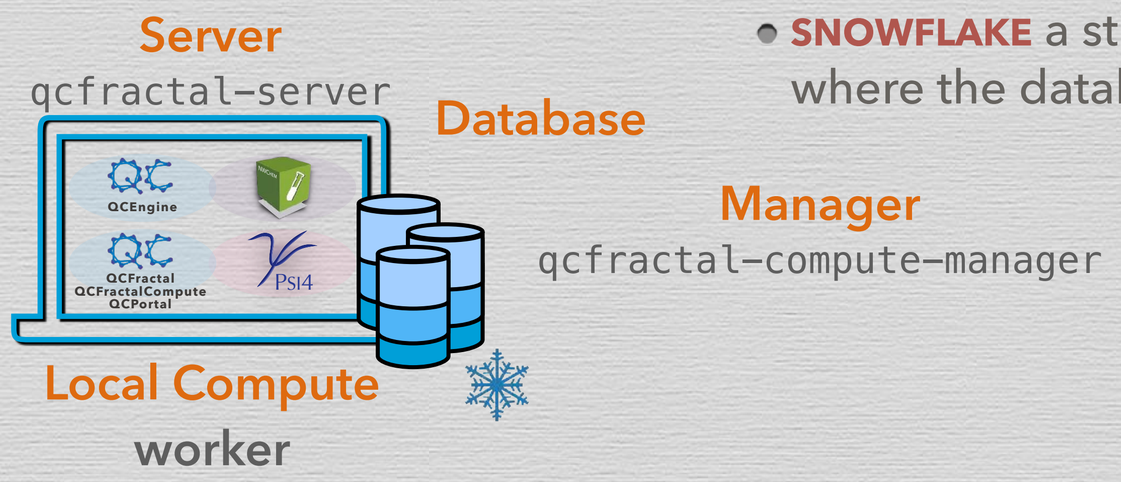
Zach Glick
GaTech

Z. L. Glick, A. Koutsoukas, D. L. Cheney, and C. D. Sherrill, J. Chem. Phys 143 224013 (2021).

QCArchive, smallest

a local single-box installation

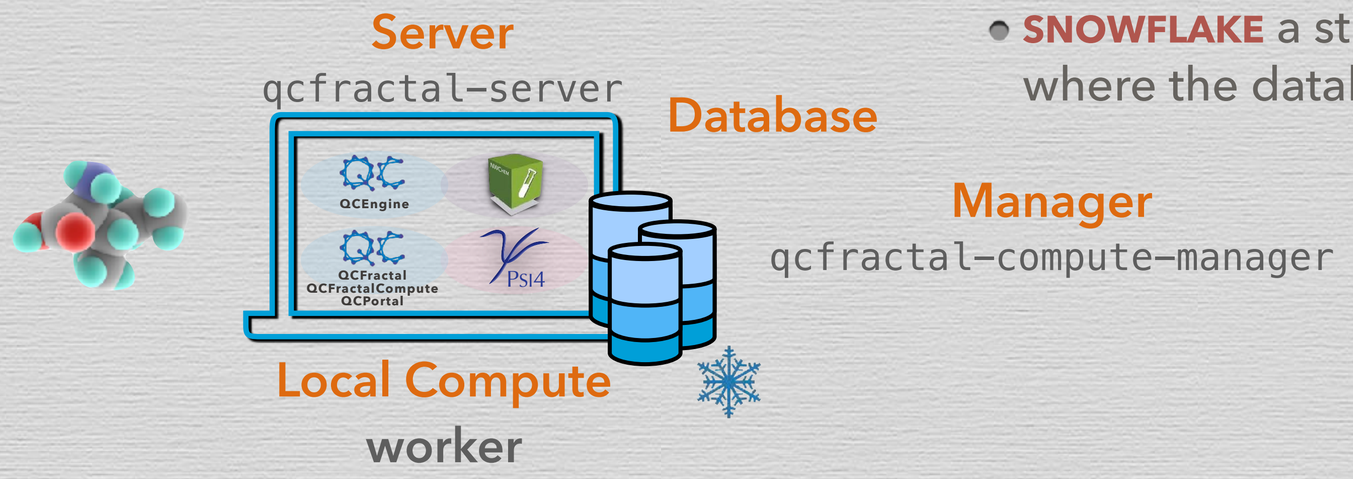
- **QCARCHIVE INFRASTRUCTURE** software stack unchanged
- **COLOCATE** QCFractal, database and QC codes for self-contained setup
- **SNOWFLAKE** a still-simpler mode where the database does not persist



QCArchive, smallest

a local single-box installation

- **QCARCHIVE INFRASTRUCTURE** software stack unchanged
- **COLOCATE** QCFractal, database and QC codes for self-contained setup
- **SNOWFLAKE** a still-simpler mode where the database does not persist



APP: Psi4 DISTRIBUTED DRIVER

post-processing flexibility vs. automation: an unwelcome choice

```
psi4.energy("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule= )
```

read: *water dimer at double-triple-zeta extrapolated MP2 with a coupled-cluter correction, counterpoise corrected*

Local Compute

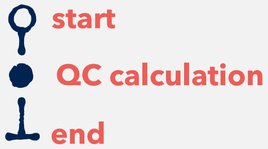
- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by sum of all calcs since run sequentially.
- **RETRIEVAL** from filesystem difficult since many calcs in aggregated outfile.
- **FLEXIBILITY** limited to Psi4 only.

Psi4 distributed driver

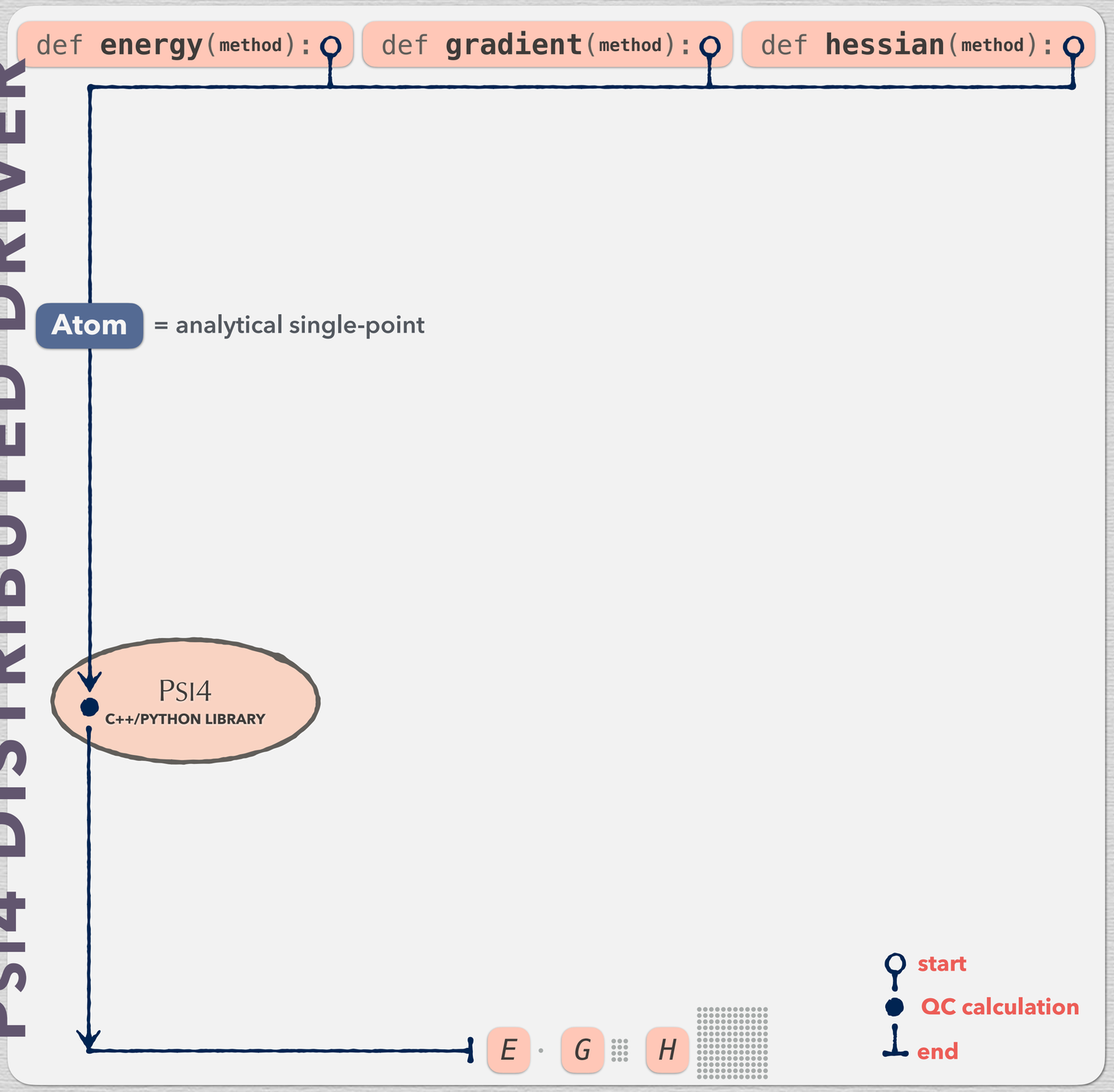
```
def energy(method):
```

```
def gradient(method):
```

```
def hessian(method):
```



PSI4 DISTRIBUTED DRIVER



PSI4 DISTRIBUTED DRIVER

```
def energy(method):
```

```
def gradient(method):
```

```
def hessian(method):
```

Atom = analytical single-point

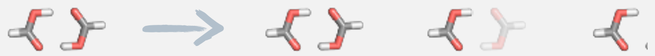
PSI4
C++/PYTHON LIBRARY

E · G H

start
QC calculation
end

```
class ManyBodyComputer ():
```

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

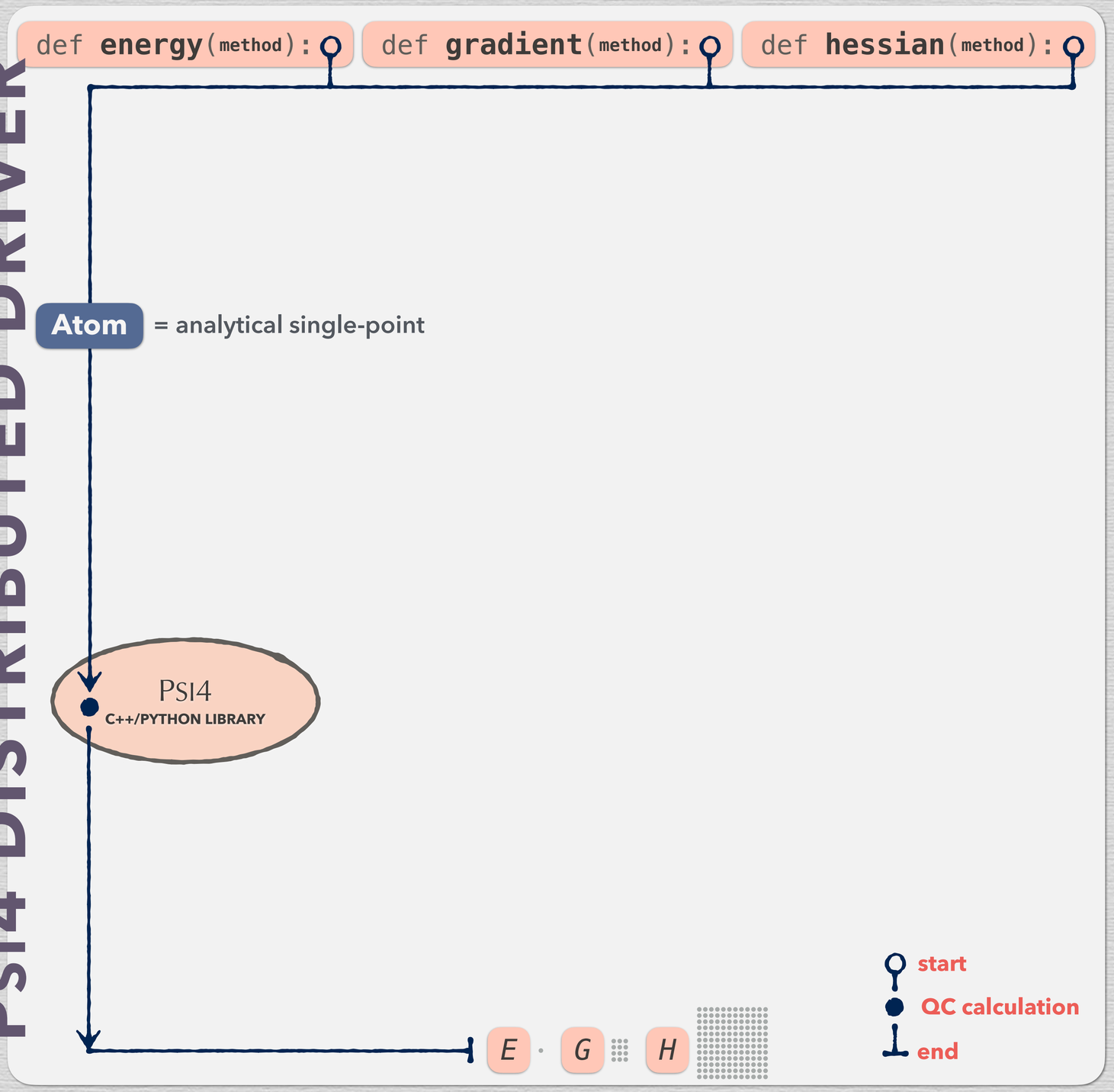


```
for frag in fragments: return qcschema
```

QCSchema

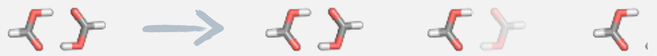
ASM Assemble n-body & interaction results from fragments.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.



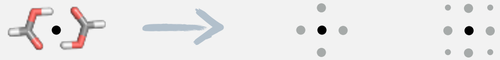
```
for frag in fragments: return qcschema
```

QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.

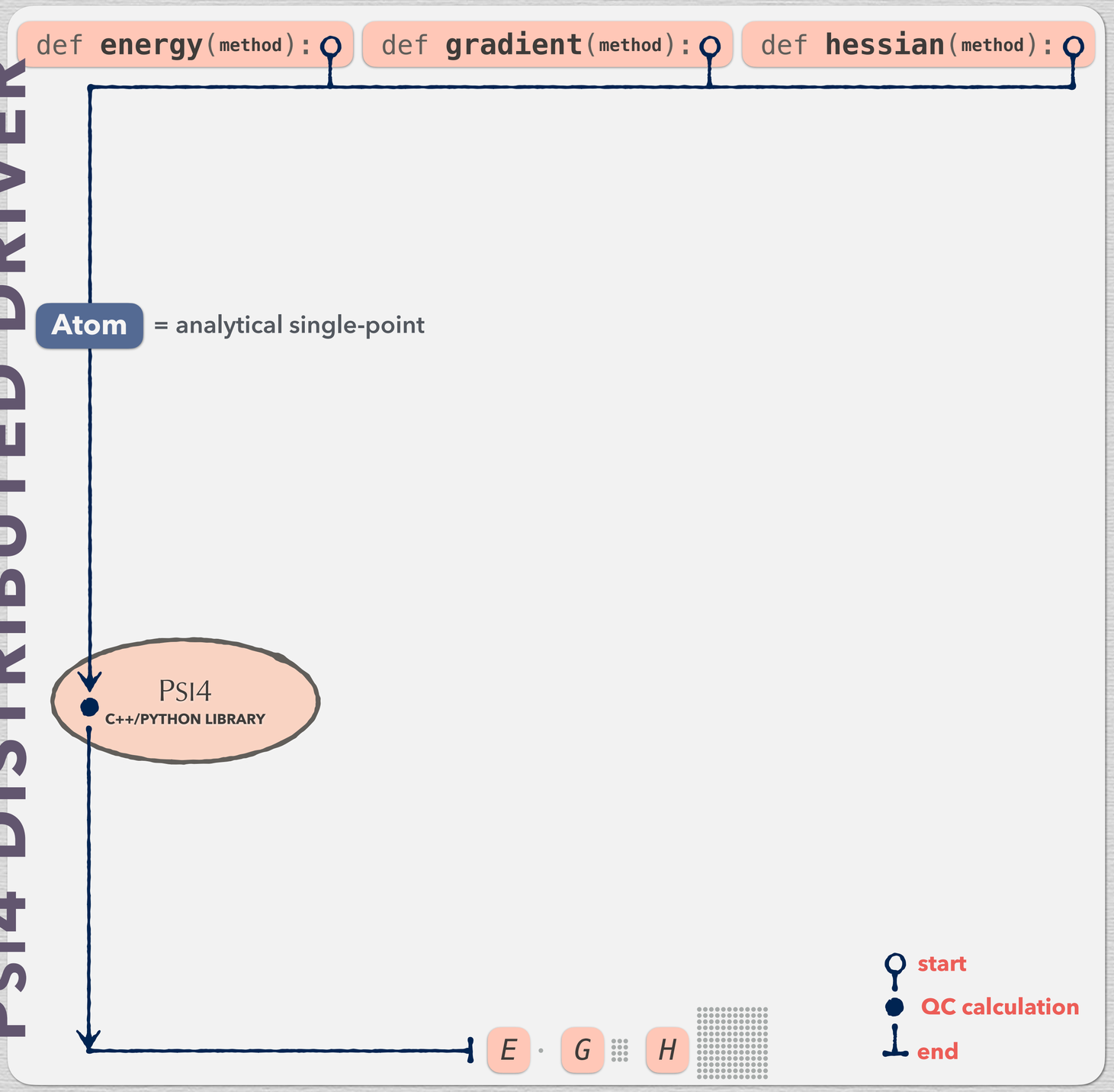


```
for disp in displacements: return qcschema
```

QCSchema

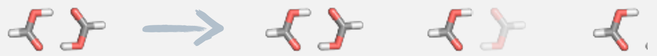
ASM Assemble derivative results from displacements.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.



```
for frag in fragments: return qcschema
```

QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.

```
for disp in displacements: return qcschema
```

QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

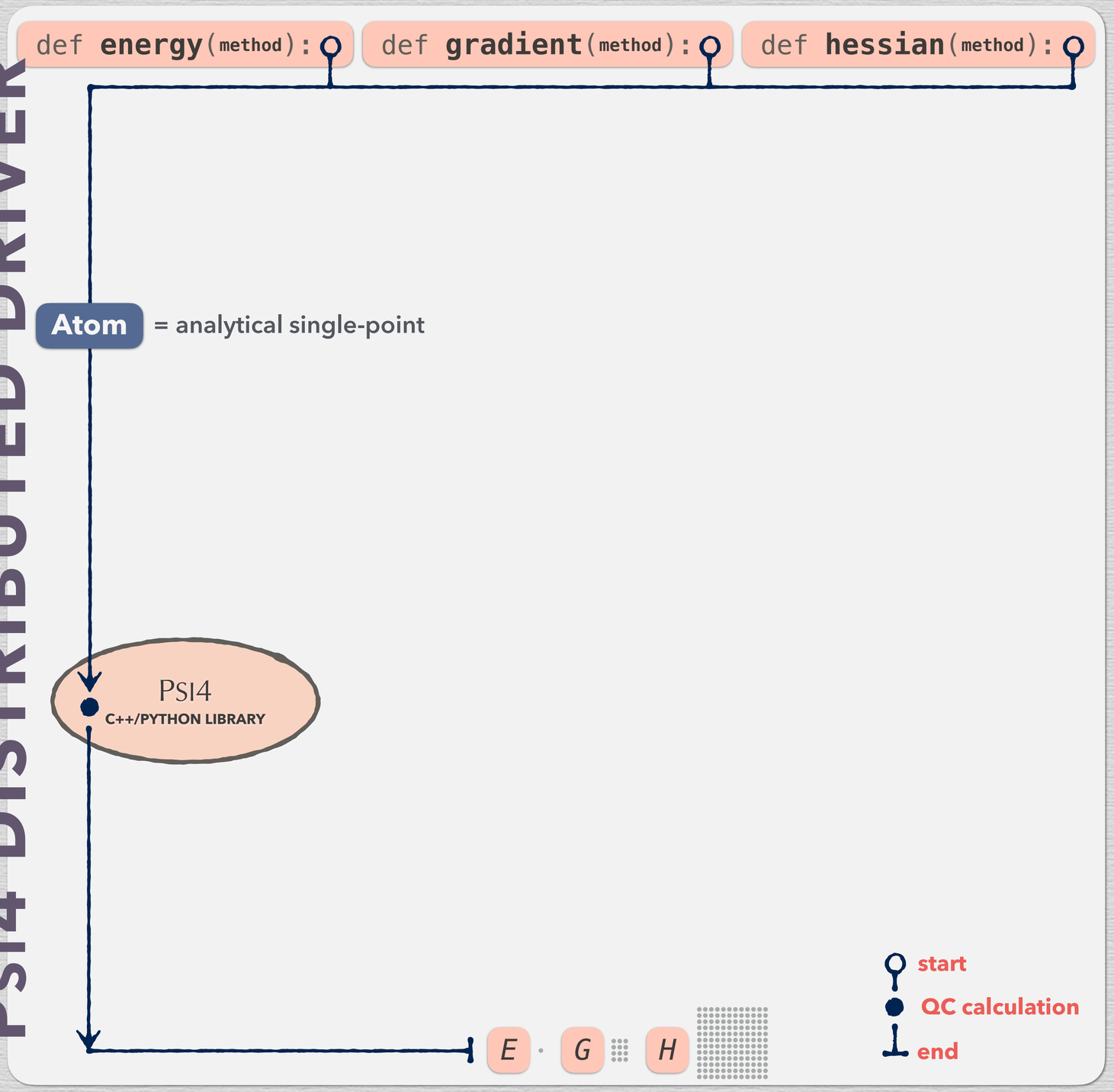
PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

```
mp2/cc-pv[tq]z  →  MP2 TOTAL ENERGY/cc-pVTZ
                   MP2 TOTAL ENERGY/cc-pVQZ
for mc in modelchems: return qcschema
```

QCSchema

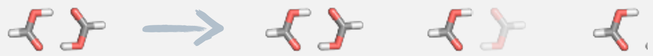
ASM Assemble extrapolations & total results from modelchems.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

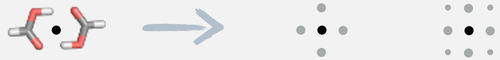


`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



`for disp in displacements: return qcschema` QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema` QCSchema

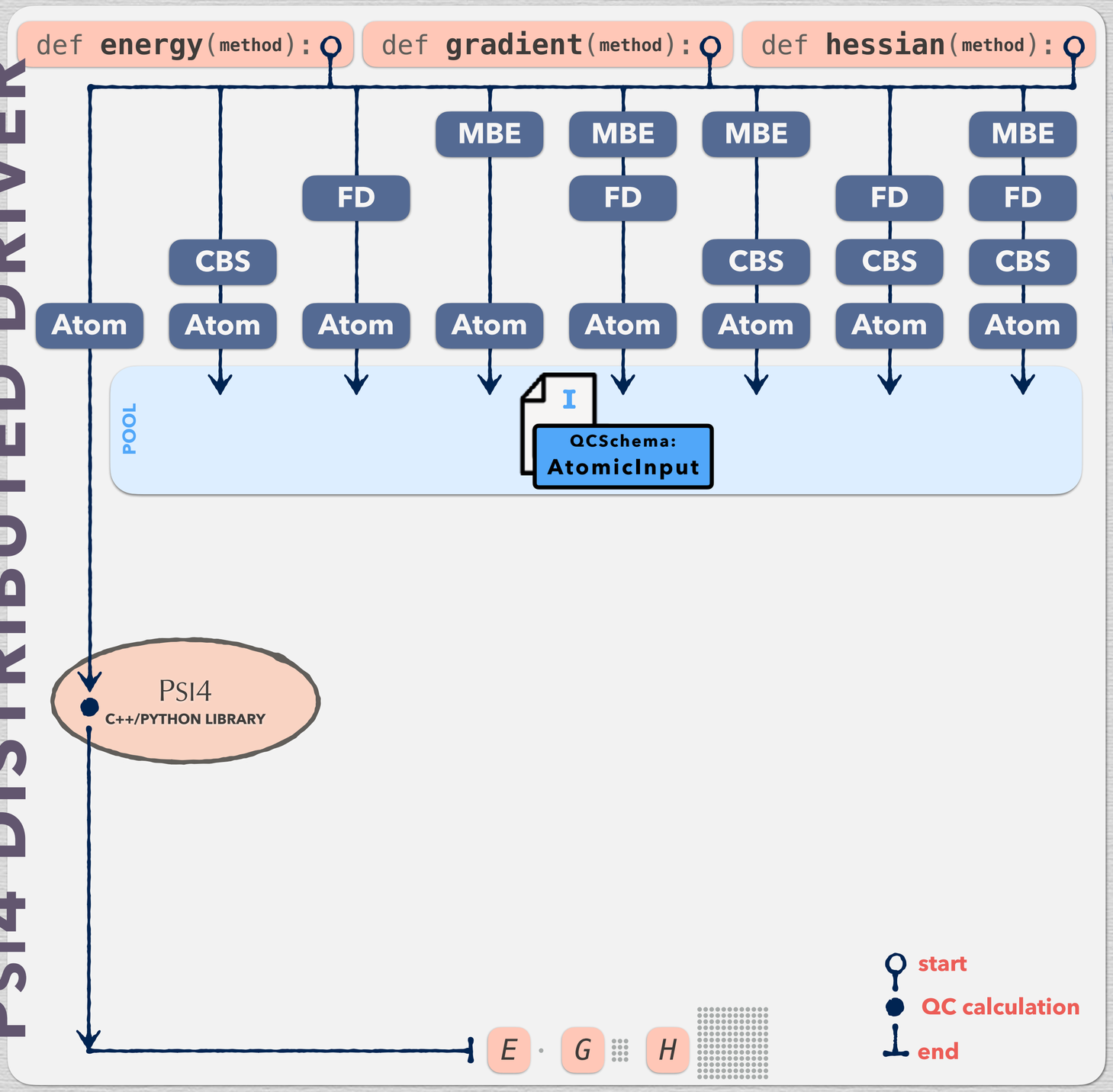
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

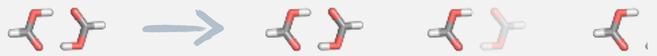
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

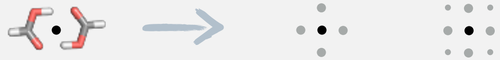


`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



`for disp in displacements: return qcschema` QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema` QCSchema

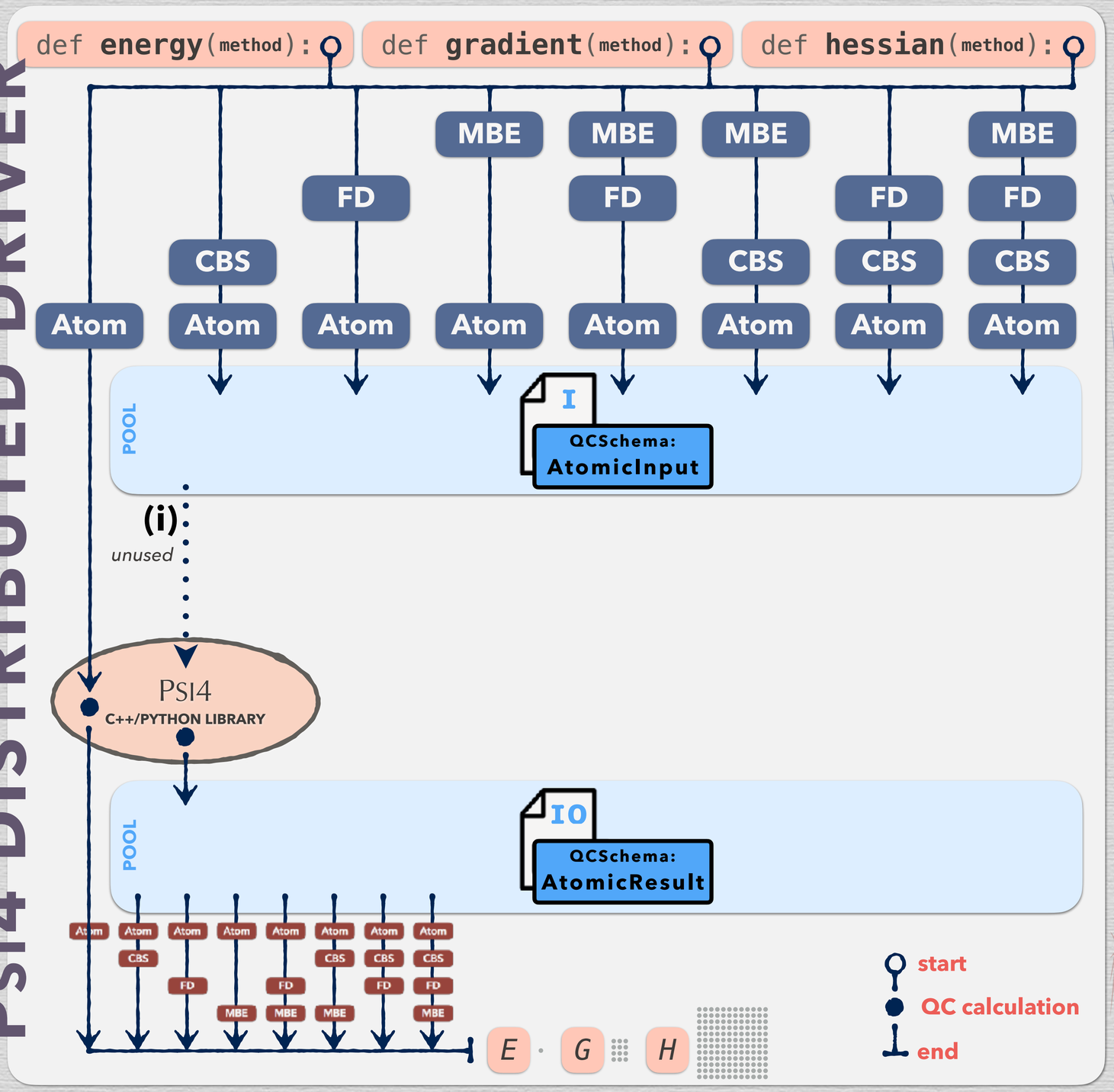
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

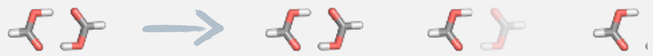
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

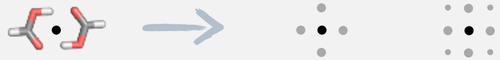


`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



`for disp in displacements: return qcschema` QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema` QCSchema

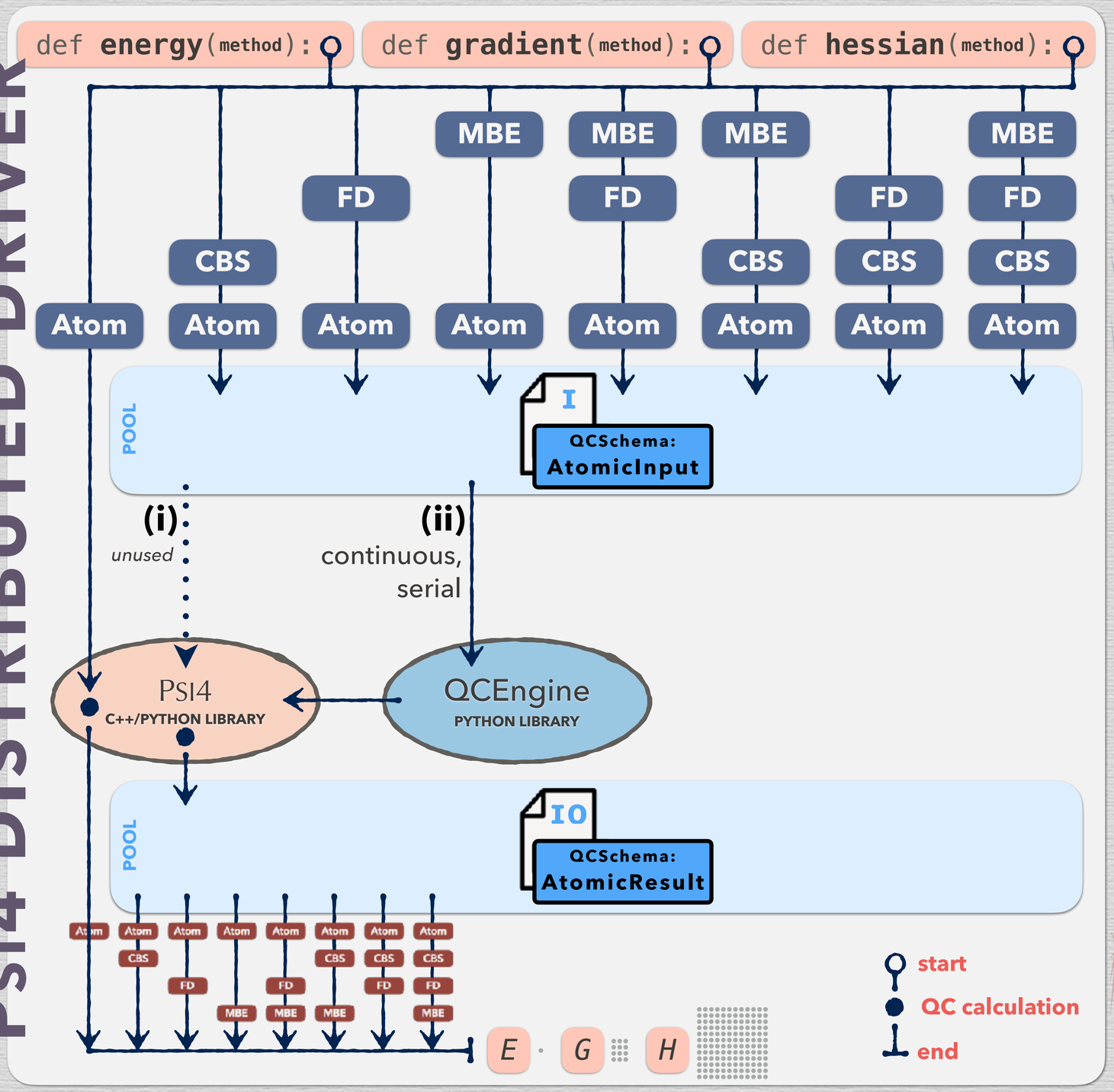
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

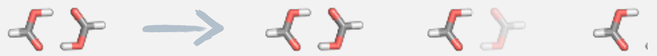
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

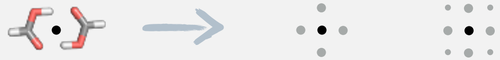


`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



`for disp in displacements: return qcschema` QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema` QCSchema

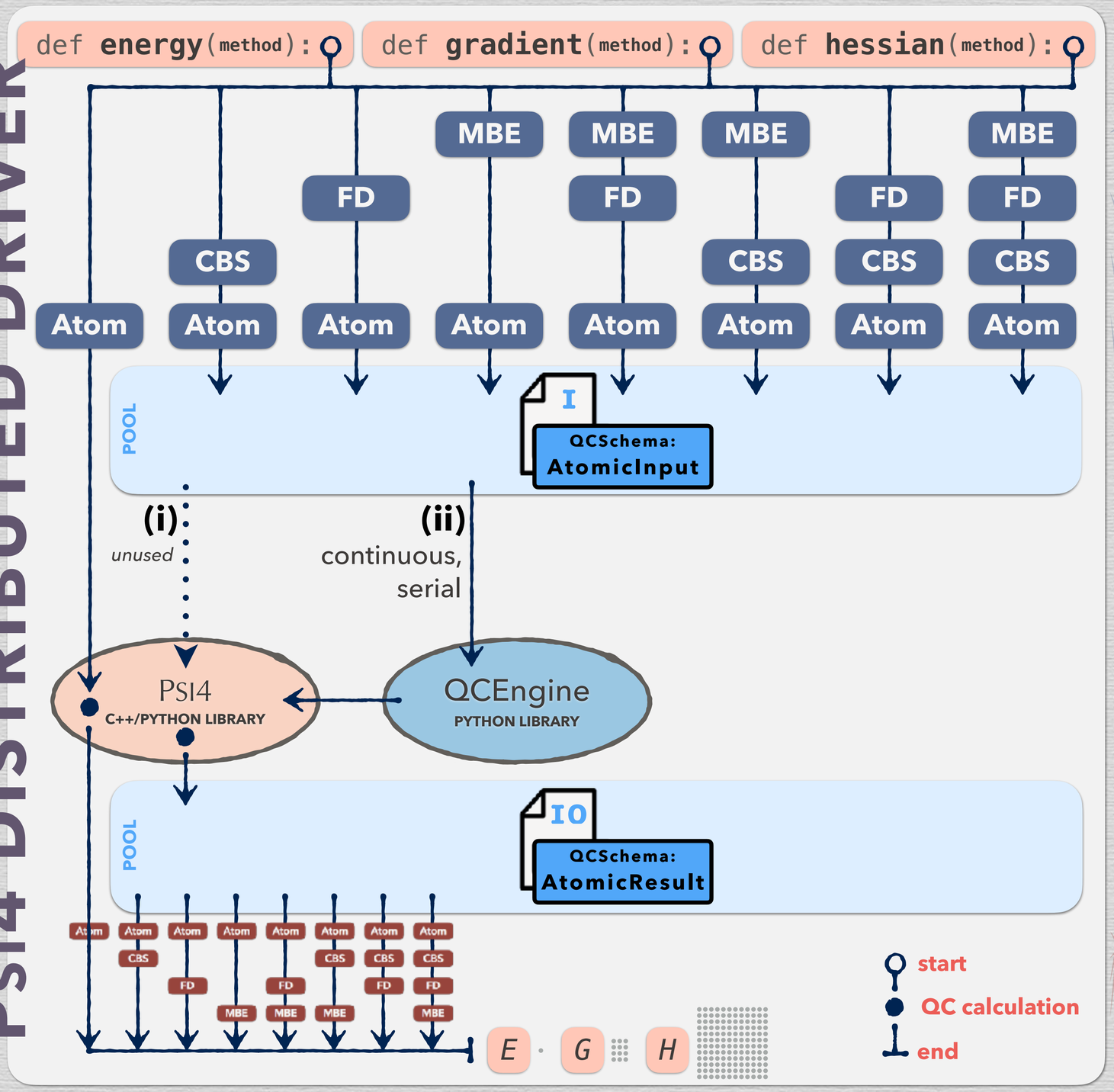
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

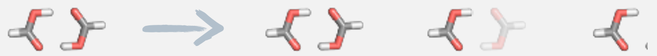
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.



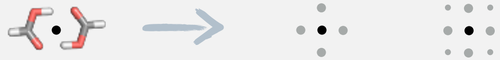
```
for frag in fragments: return qcschema
```

QCSchema

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



```
for disp in displacements: return qcschema
```

QCSchema

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

```
mp2/cc-pv[tq]z  →  MP2 TOTAL ENERGY/cc-pVTZ
                   MP2 TOTAL ENERGY/cc-pVQZ
for mc in modelchems: return qcschema
```

QCSchema

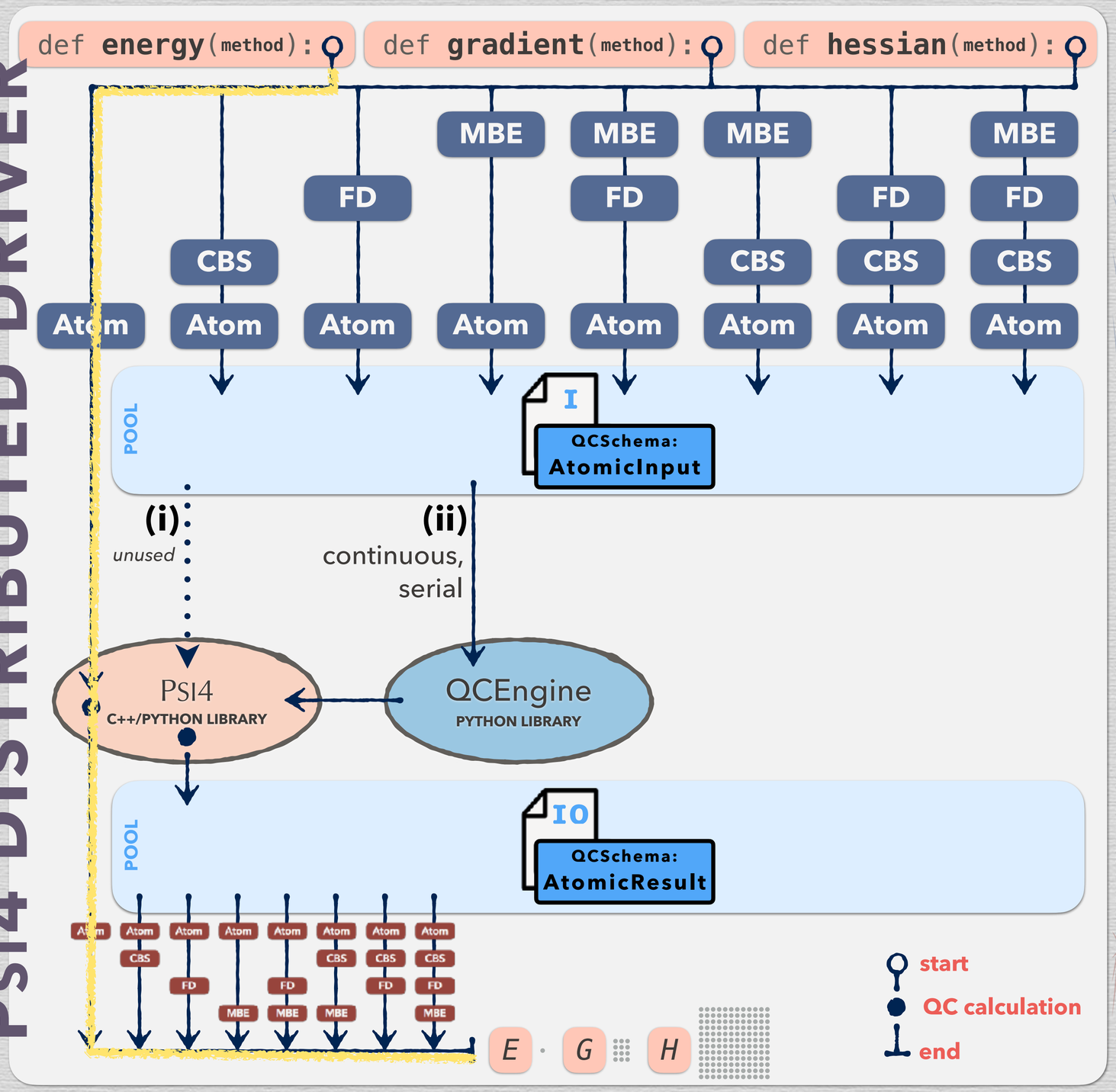
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

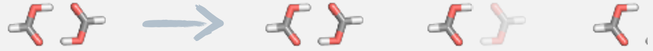
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.



`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

```
psi4.energy("mp2/cc-pvdz")
```

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema` QCSchema

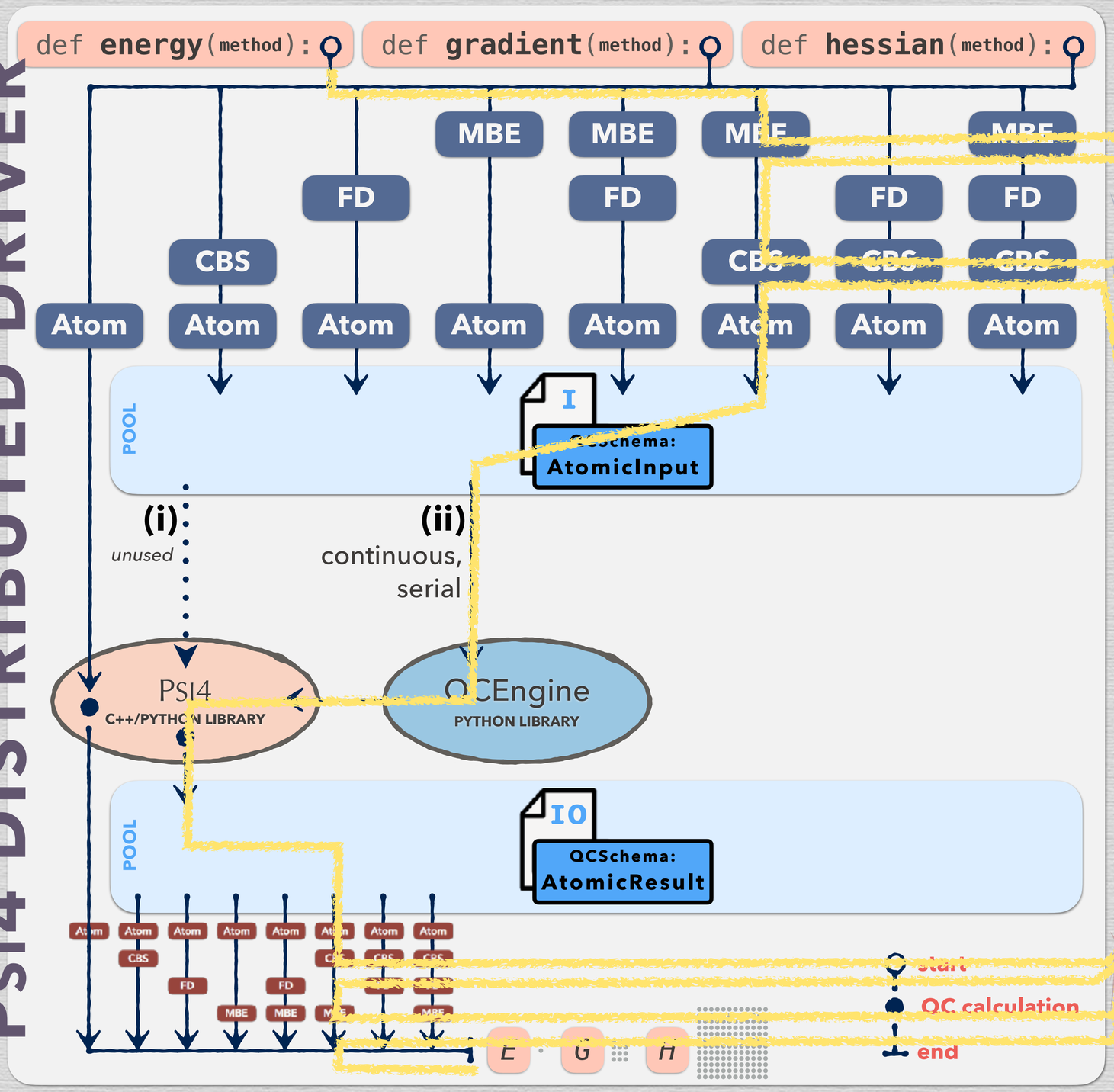
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

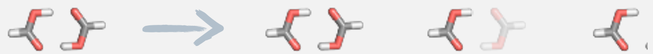
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.



for frag in fragments: return qcschema

ASM Assemble n-body & interaction results from fragments.

```
psi4.energy("mp2/cc-pvdz")
```

```
psi4.energy("mp2/cc-pv[dt]z", bsse_type="cp")
```

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

for me in modelchem: return qcschema

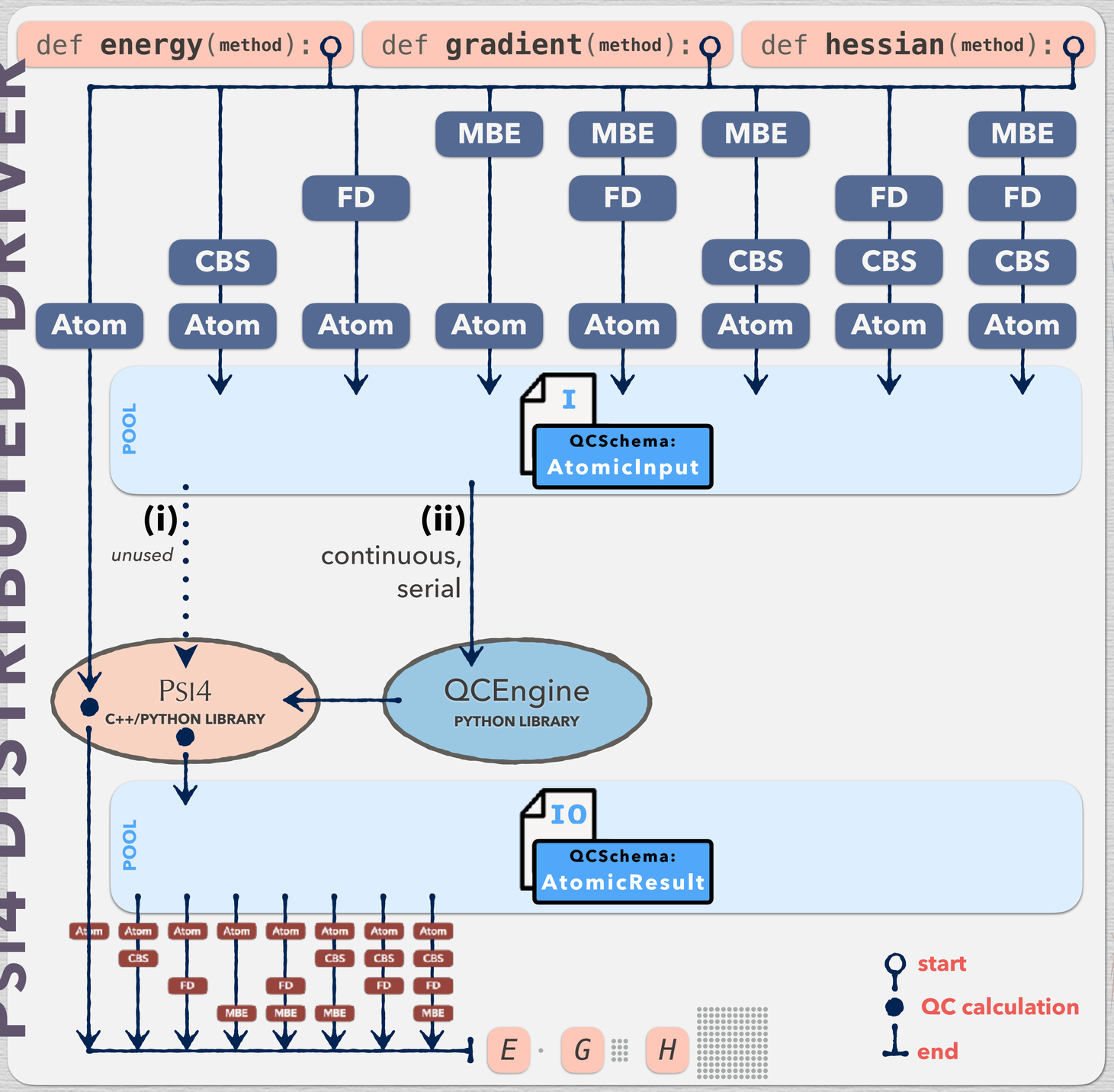
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. return qcschema

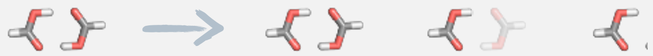
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

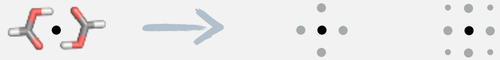


```
for frag in fragments: return qcschema
```

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



```
for disp in displacements: return qcschema
```

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

```
mp2/cc-pv[tq]z  →  MP2 TOTAL ENERGY/cc-pVTZ
                   MP2 TOTAL ENERGY/cc-pVQZ
for mc in modelchems: return qcschema
```

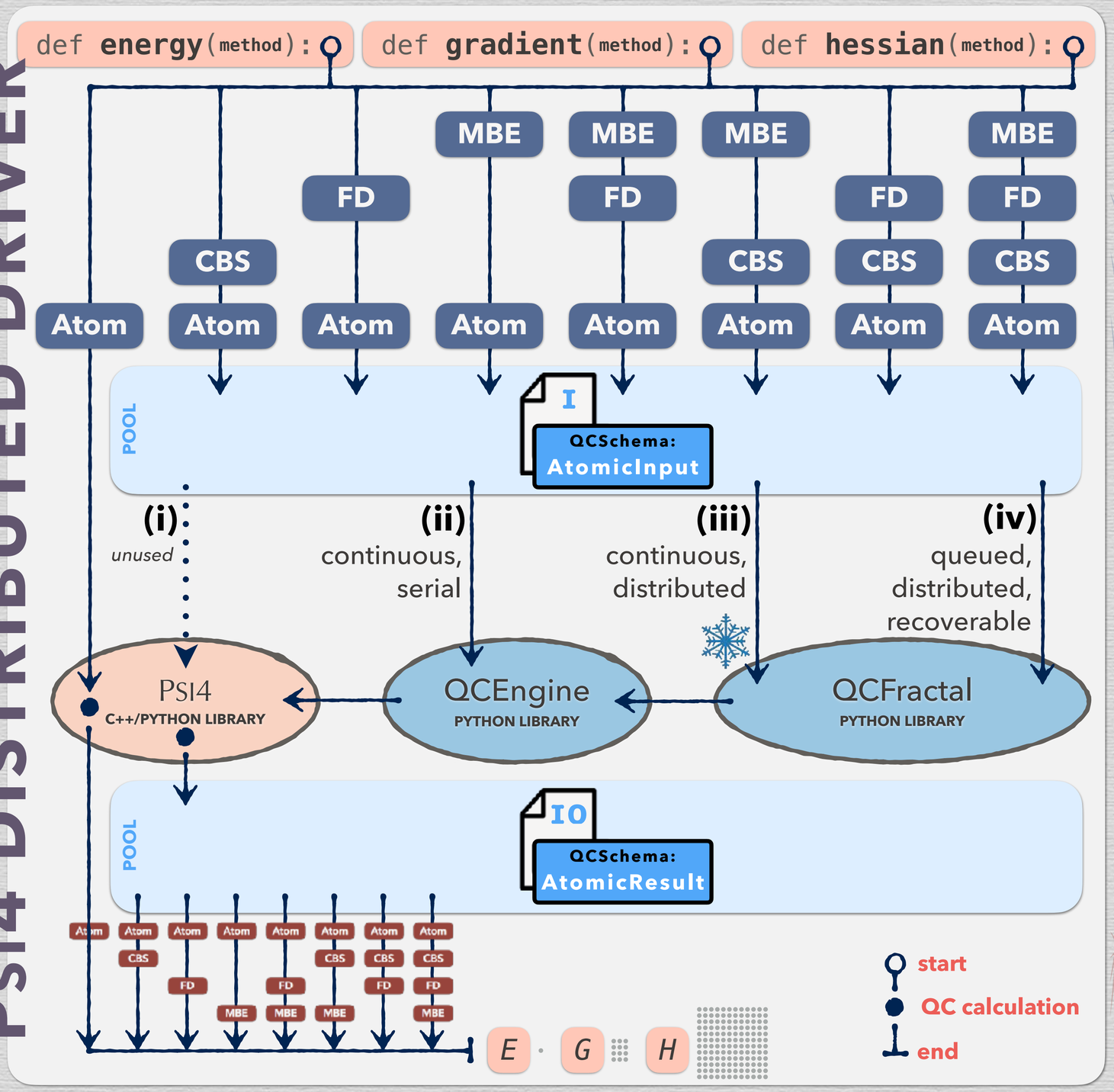
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

PLAN **molecule** & **method** unchanged. `return qcschema`

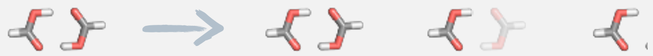
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

PLAN Separate **molecule** into subsystems. CP, noCP, VMFC basis. **method** unchanged.

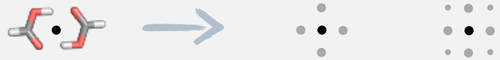


`for frag in fragments: return qcschema`

ASM Assemble n-body & interaction results from fragments.

class FiniteDifferenceComputer ():

PLAN Displace **molecule** according to stencil. Reference **molecule** & **method** unchanged.



`for disp in displacements: return qcschema`

ASM Assemble derivative results from displacements.

class CompositeComputer ():

PLAN Separate **method** into method, basis, & extrapolations. **molecule** unchanged.

`mp2/cc-pv[tq]z` → `MP2 TOTAL ENERGY/cc-pVTZ`, `MP2 TOTAL ENERGY/cc-pVQZ`

`for mc in modelchems: return qcschema`

ASM Assemble extrapolations & total results from modelchems.

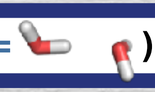
class AtomicComputer ():

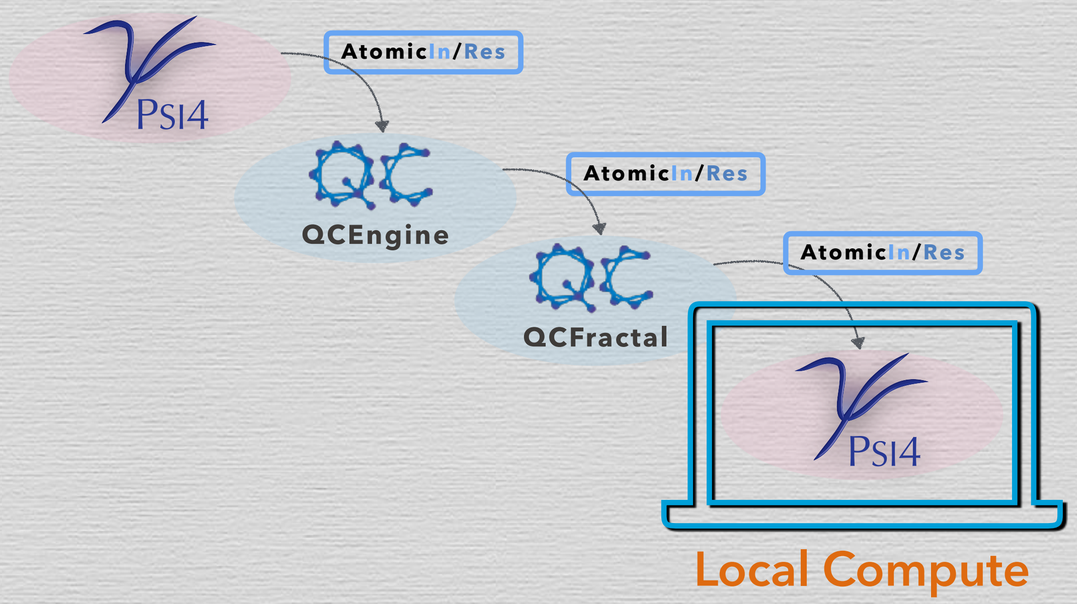
PLAN **molecule** & **method** unchanged. `return qcschema`

ASM Return analytic energy, gradient, or Hessian.

Psi4 DISTRIBUTED DRIVER, QCArchive

distributed, searchable, and error-resistant

```
psi4.energy("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=  )
```



(4) **Psi4** driver (core) runs single-point jobs

(5) **QCEngine** recieves jobs as **AtomicResult QCSchema**

(3) **QCEngine** recieves jobs as **AtomicInput QCSchema**

(6) **QCFractal** stores jobs in DB & retrives on query

(2) **QCFractal** orchestrates job running among attached resources

(7) **Psi4** driver assembles results into usual API

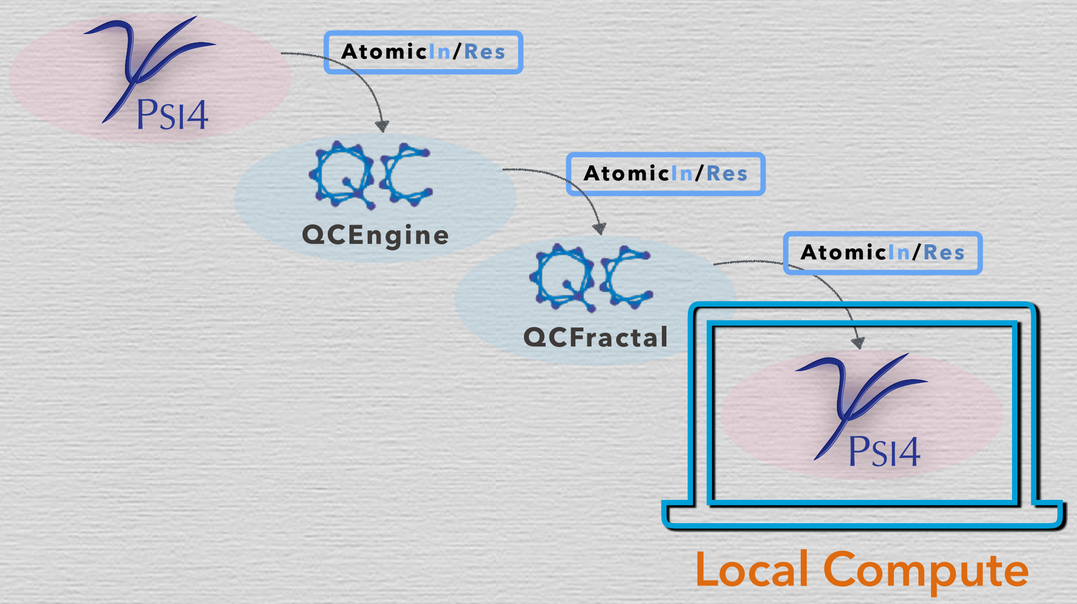
(1) **Psi4** driver (post-proc. manager) plans single-point jobs from API input

- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by single longest calc since QCFractal runs parallel in queue.
- **RETRIEVAL** from QCArchive database with query.
- **FLEXIBILITY** limited to Psi4 only.

Psi4 DISTRIBUTED DRIVER, QCArchive

distributed, searchable, and error-resistant

```
psi4.energy("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=
```



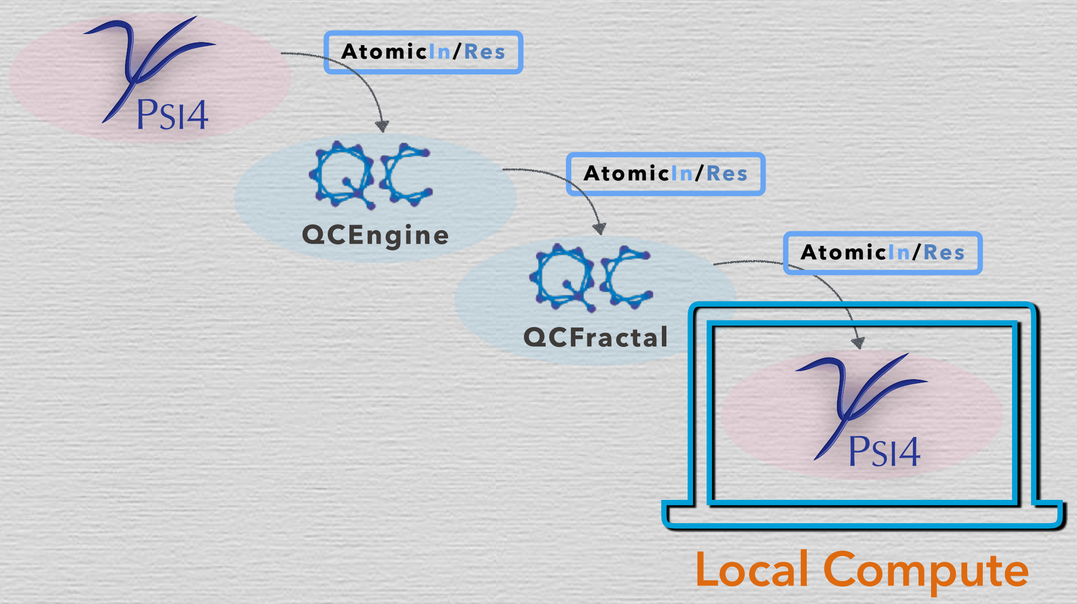

```
import psi4
from qcportal import PortalClient

client = PortalClient("http://localhost:7777")

plan = psi4.energy(
        "MP2/cc-pV[DT]Z + d:CCSD(T)/cc-pVDZ",
        bsse_type="cp",
        return_plan=True,
        molecule=  )
plan.compute(client)

# re-run file after jobs complete for final processing

ene = plan.get_psi4_results(client)
print(ene)  # CP IE
```

- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by single longest calc since QCFractal runs parallel in queue.
- **RETRIEVAL** from QCArchive database with query.
- **FLEXIBILITY** limited to Psi4 only.

Psi4 DISTRIBUTED DRIVER, QCArchive

distributed, searchable, and error-resistant

```
psi4.energy("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=  )
```




```
import psi4
from qcportal import PortalClient

client = PortalClient("http://localhost:7777")

plan = psi4.energy(
        "MP2/cc-pV[DT]Z + d:CCSD(T)/cc-pVDZ",
        bsse_type="cp",
        return_plan=True,
        molecule=  )
plan.compute(client)

# re-run file after jobs complete for final processing

ene = plan.get_psi4_results(client)
print(ene)  # CP IE
```

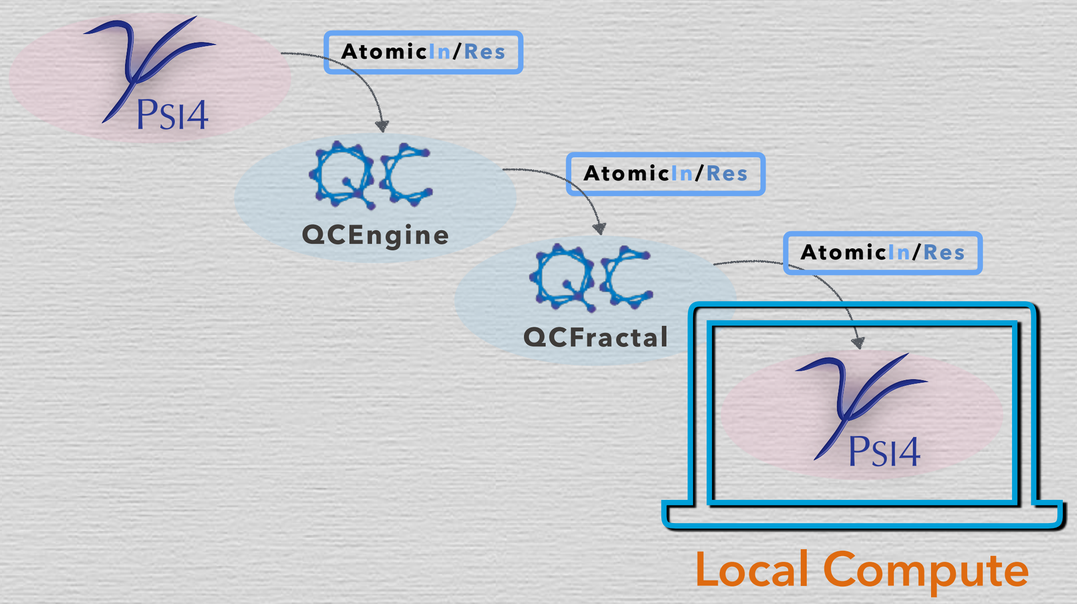
```
plan = psi4.energy("CCSD(T)/cc-pVDZ",
                   return_plan=True,
                   molecule=  )
plan.compute(client)  # free! calcs in database

ene, wfn = plan.get_psi4_results(client, return_wfn=True)
print(ene)  # CCSD(T) energy
```

- **SPECIFICATION** through single file. Procedure automated, so low ri
- **RATE-LIMITED** by single longest calc since QCFractal runs parallel i
- **RETRIEVAL** from QCArchive database with query.
- **FLEXIBILITY** limited to Psi4 only.

Psi4 DISTRIBUTED DRIVER, QCArchive

distributed, searchable, and error-resistant

```
psi4.energy("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=  )
```




```
import psi4
from qcportal import PortalClient

client = PortalClient("http://localhost:7777")

plan = psi4.energy(
        "MP2/cc-pV[DT]Z + d:CCSD(T)/cc-pVDZ",
        bsse_type="cp",
        return_plan=True,
        molecule=  )
plan.compute(client)

# re-run file after jobs complete for final processing

ene = plan.get_psi4_results(client)
print(ene)  # CP IE
```

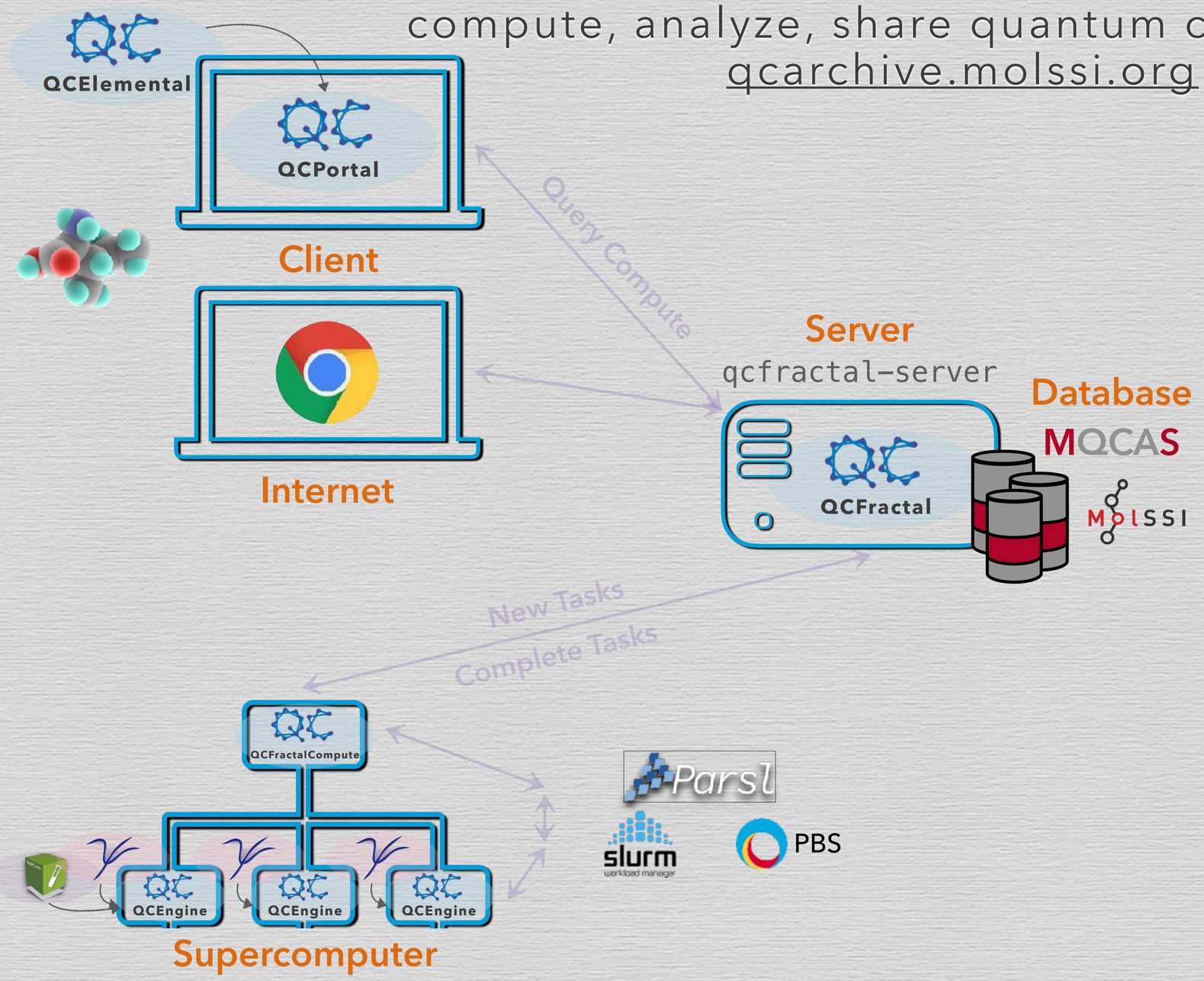
```
plan = psi4.energy("CCSD(T)/cc-pVDZ",
                   return_plan=True,
                   molecule=  )
plan.compute(client)  # free! calcs in database

ene, wfn = plan.get_psi4_results(client, return_wfn=True)
print(ene)  # CCSD(T) energy
```

- **SPECIFICATION** through single file. Procedure automated, so low ri
- **RATE-LIMITED** by single longest calc since QCFractal runs parallel i
- **RETRIEVAL** from QCArchive database with query.
- **FLEXIBILITY** limited to Psi4 only.

QCArchive Overview

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



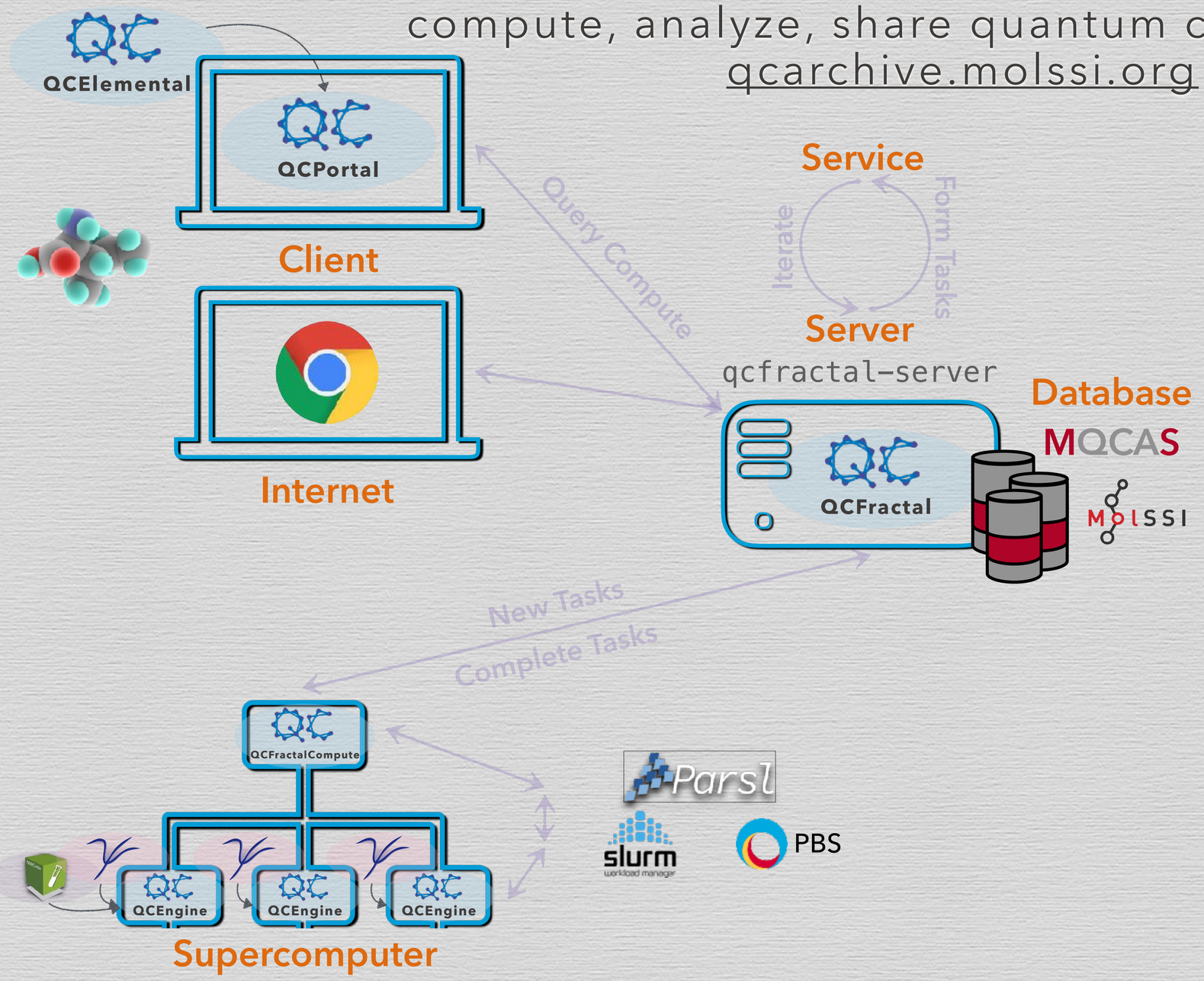
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive Overview

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE** – up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC** – runs on laptops, campus clusters, and leadership-class supercomputers

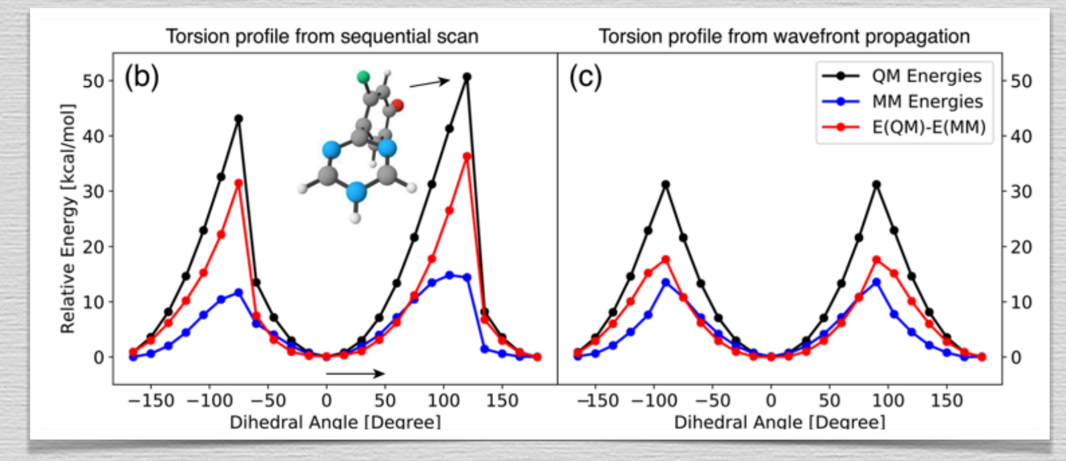
`qcfractal-compute-manager`	Manager
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCSchema TorsionDrive

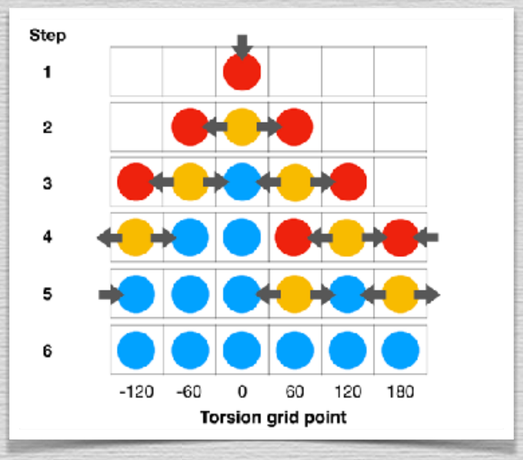
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION



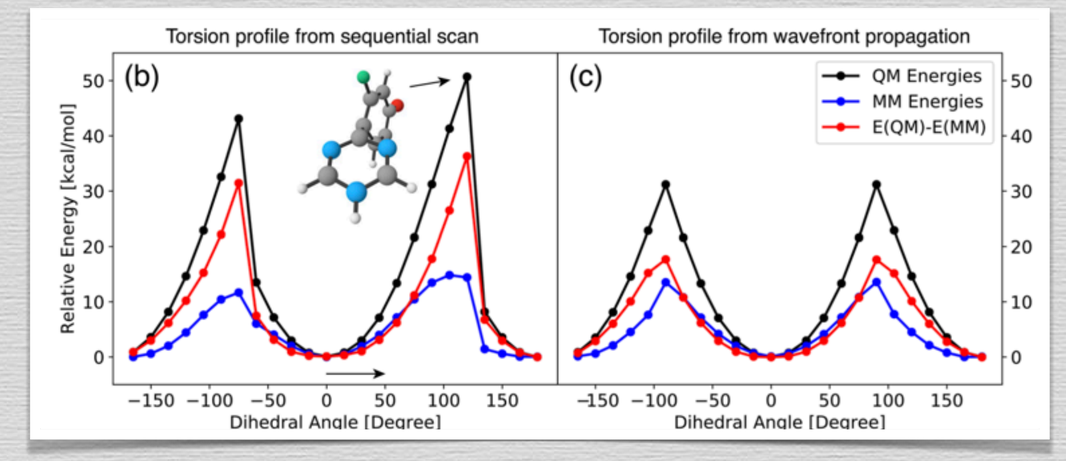
J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

QCSchema TorsionDrive

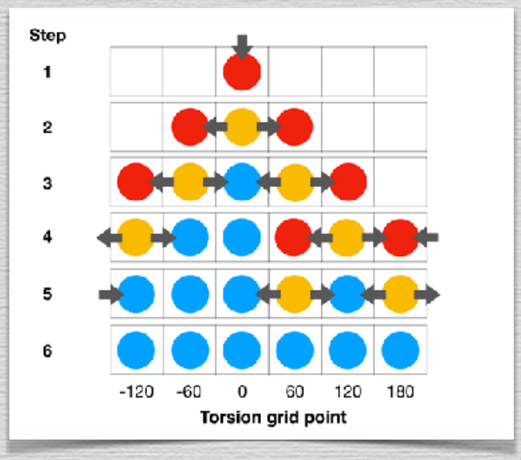
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION



TorsionDriveInput

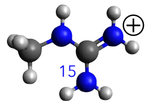
schema_version: 1

provenance: creator name and version info

`QCElemental, 0.28.0`

extras: free-form storage

initial_molecule:



Molecule

keywords: knobs in the TD program

optimization_spec: how to run *geometry opts*

input_specification: how to run *gradients* for opt

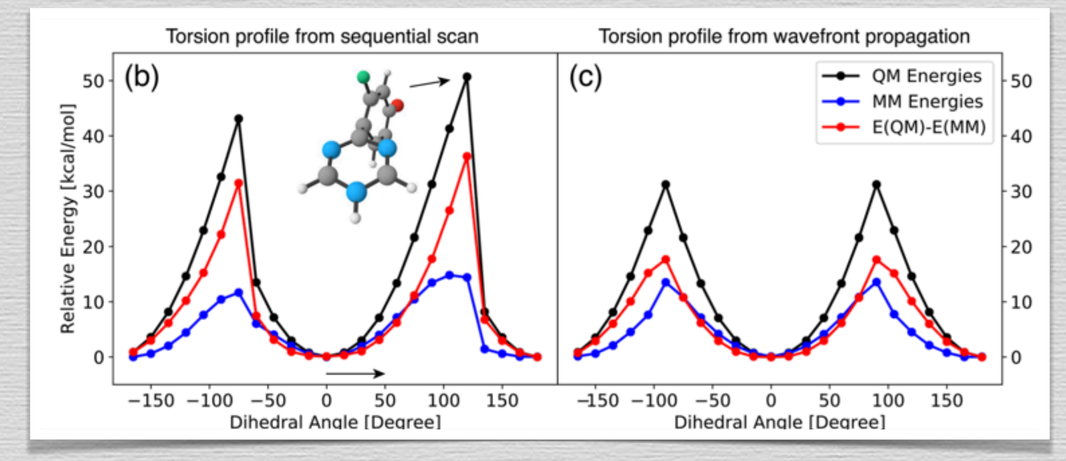
J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

QCSchema TorsionDrive

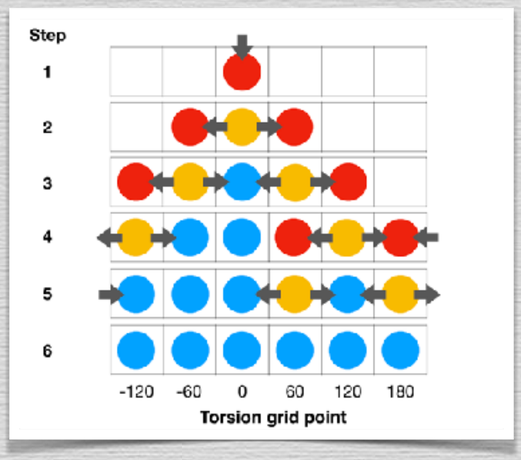
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION



TorsionDriveInput

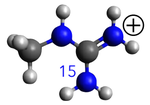
schema_version: 1

provenance: creator name and version info

`QCElemental, 0.28.0`

extras: free-form storage

initial_molecule:



Molecule

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan

dihedral_ranges: limits of scan

grid_spacing: IE dihedral steps of scan

energy_decrease_thresh: point trigger

energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

input_specification: how to run *gradients* for opt

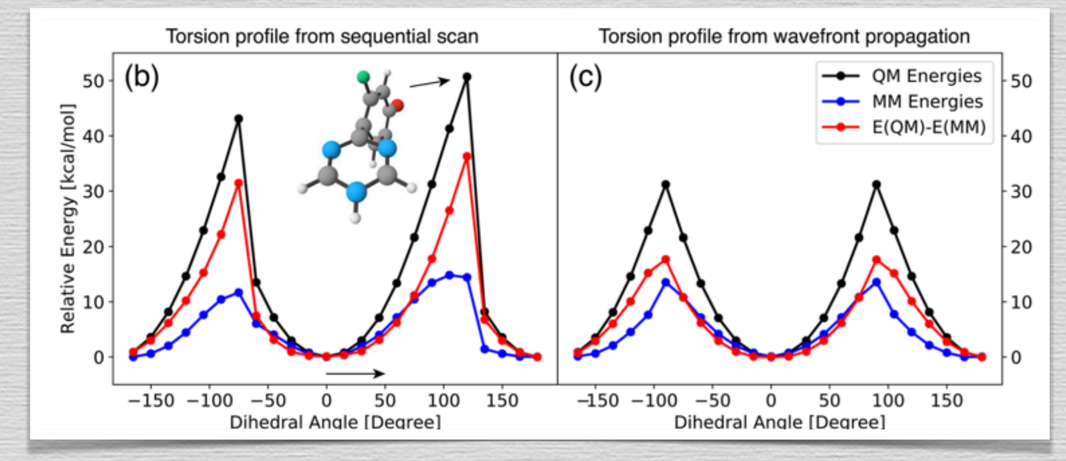
J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

QCSchema TorsionDrive

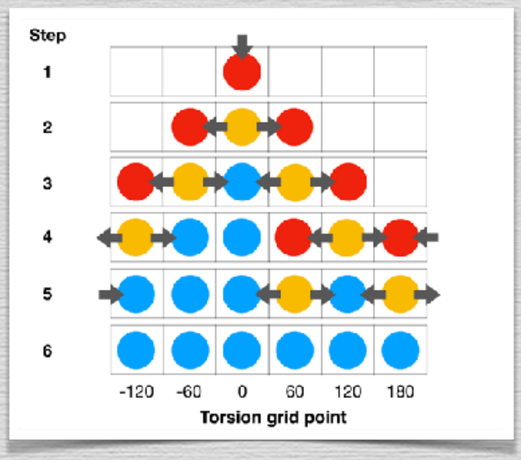
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION



J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

TorsionDriveInput

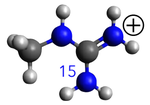
schema_version: 1

provenance: creator name and version info

`QCElemental, 0.28.0`

extras: free-form storage

initial_molecule:



Molecule

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan

dihedral_ranges: limits of scan

grid_spacing: IE dihedral steps of scan

energy_decrease_thresh: point trigger

energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1

procedure: any optimizer in QCEngine

pyberny

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program:` Psi4 `}`

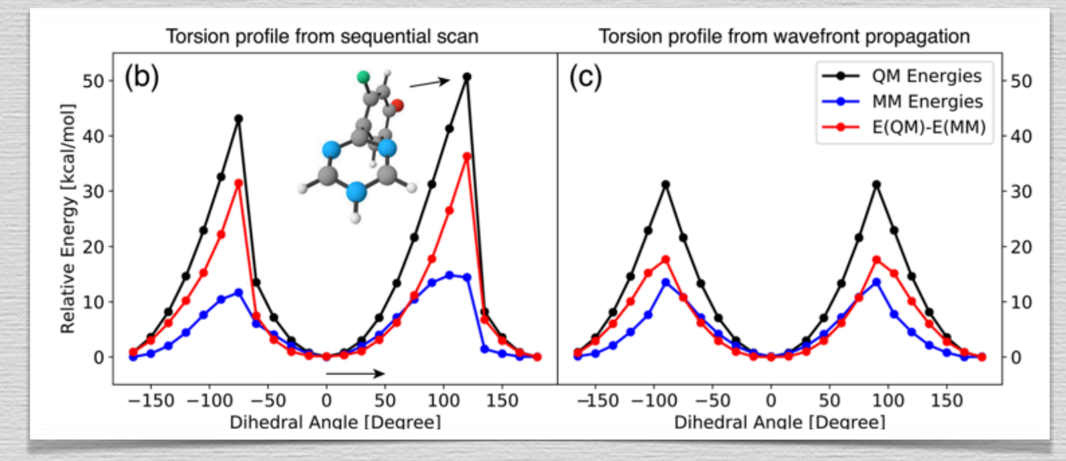
input_specification: how to run *gradients* for opt

QCSchema TorsionDrive

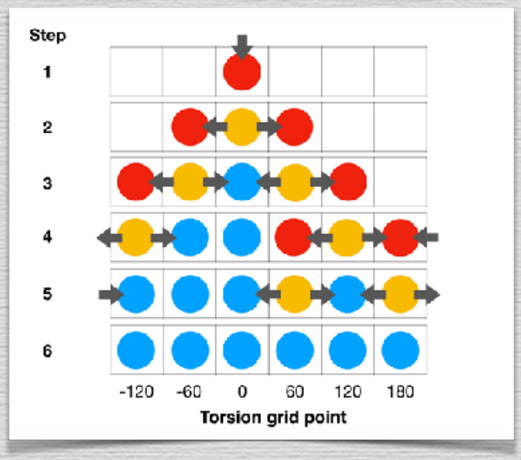
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION



J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

TorsionDriveInput

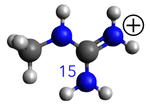
schema_version: 1

provenance: creator name and version info

`QCElemental, 0.28.0`

extras: free-form storage

initial_molecule:



Molecule

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan

dihedral_ranges: limits of scan

grid_spacing: IE dihedral steps of scan

energy_decrease_thresh: point trigger

energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1

procedure: any optimizer in QCEngine

geometTRIC pyberny

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program:` Psi4 NWChem `}`

input_specification: how to run *gradients* for opt

QCInputSpecification

driver: single-point derivative *E* *G* *H*

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

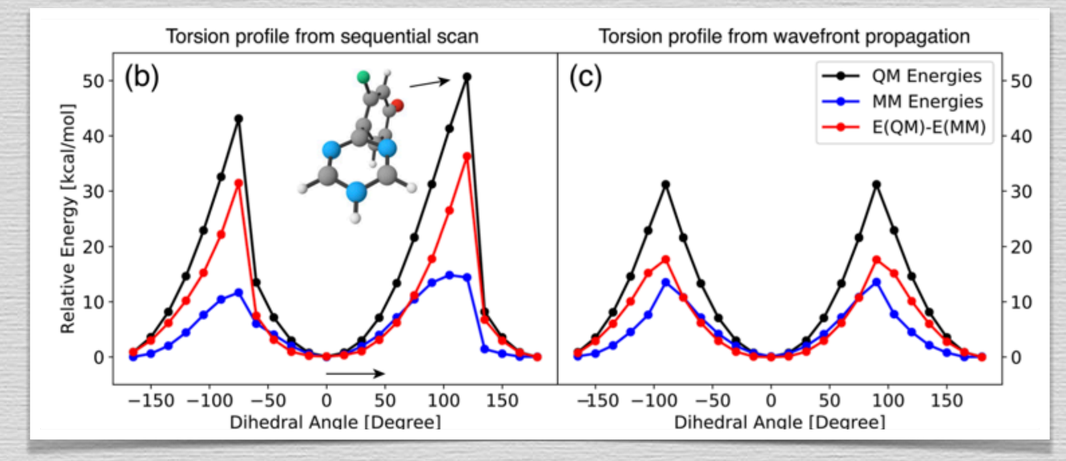
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

QCSchema TorsionDrive

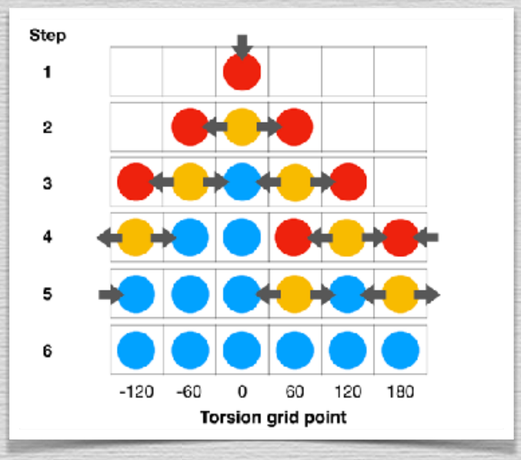
multidimensional optimizations

- dihedral parameters for force fields need full optimizations **CONSTRAINED** to a grid of dihedral values. Typical sequential scans are starting-point-dependent and often asymmetric.
- **TorsionDriveInput** is the first schema type designed for a particular target code, Lee-Ping Wang's TORSIONDRIVE. Thus it has a **CUSTOM** `keywords` schema, not just a dictionary.
- TORSIONDRIVE uses wavefront propagation to opt & re-opt points over 2D or **1D SCANS**.
- output results include the best **OPTIMIZED** structures and energies at each requested angular grid point, as well as the complete opt history that can include multiple opt at each grid pt.
- PES resulting from a TD scan are more symmetric, **RESILIANT** to starting conditions, & lower energy, which should contribute to better force fields
- Note odd way to specify optimization and gradient **PROGRAMS**

PES QUALITY



1D WAVEFRONT PROPAGATION

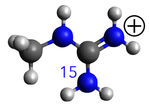


TorsionDriveResult

TorsionDriveInput (abr.)

schema_version: 1

inital_molecule:



Molecule

keywords: knobs in the TD program

TDKeywords

optimization_spec: knobs in the geo opt program

OptimizationSpecification

input_specification: how to run *gradients* opt

QCInputSpecification

success: True

provenance: *(overwrites input)*

`torsiondrive, 1.1.0, psinet, CPU E5-2630`

stdout: TD program's text output for humans

stderr: anything in error channel

extras: *(merges with input)*

final_energies: optimized energy for angles

{dihedral id: *E* }

optimization_history: full results for opt steps

{dihedral id: [**OptimizationResult**] }

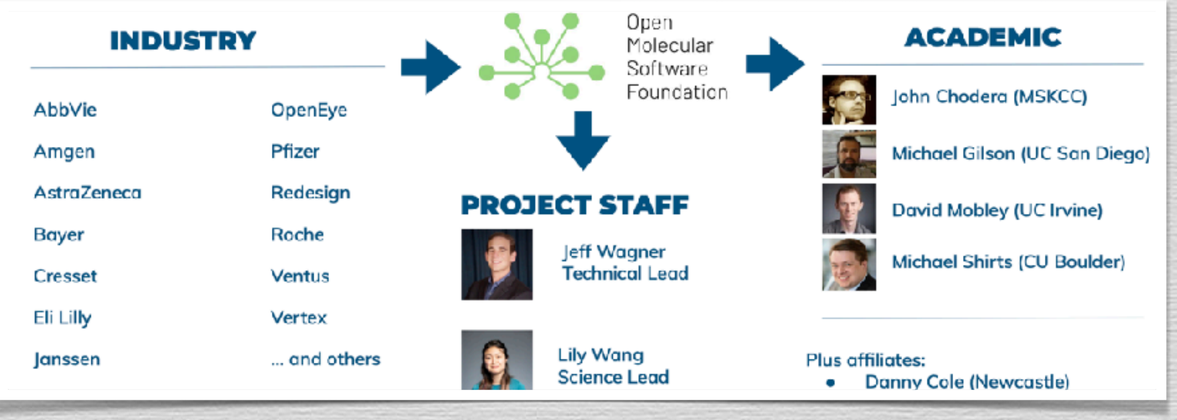
final_molecules: optimized molecules for angles

{dihedral id: **Molecule** }

J. Chem. Phys. 152, 244116 (2020); doi: 10.1063/5.0009232

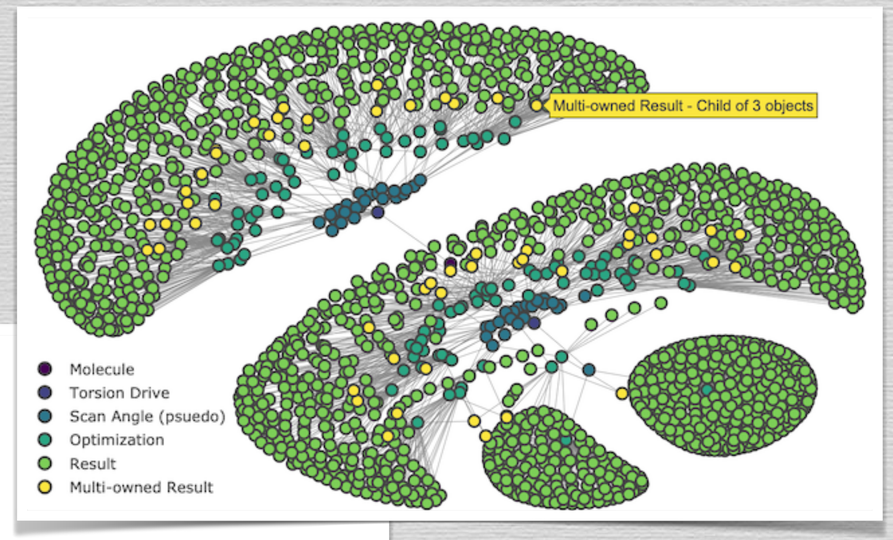
APP: OPENFORCEFIELD

runs on torsiondrives



DATA RELATIONS FOR TORSION DRIVE

UFF/B3LYP-D3 OVER C1CCC(CC1)N-[3,5,6,12]



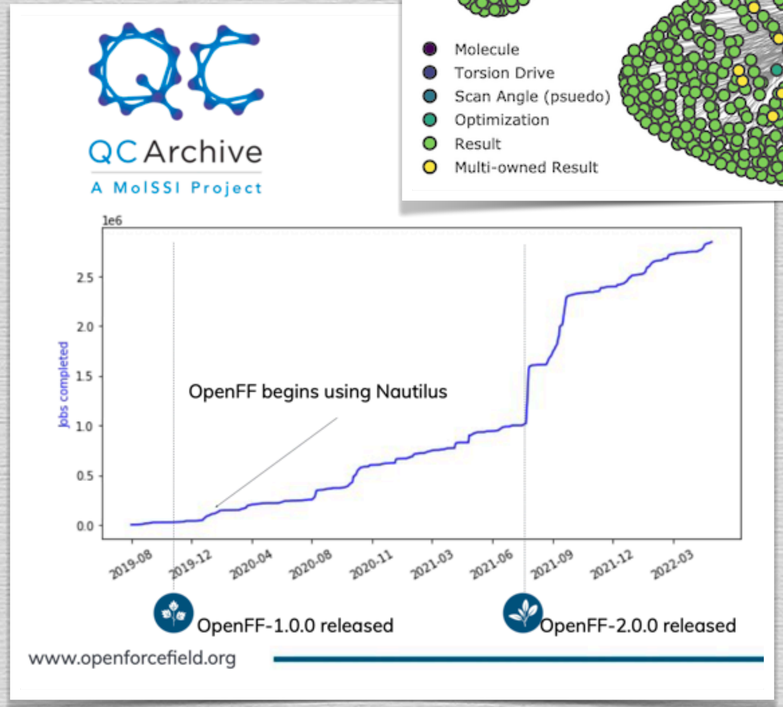
SUBMITTING & QUERYING A TORSIONDRIVE

```python
import qcportal as ptl

client = ptl.FractalClient("https://myserver")

HOOH = ptl.models.Molecule.from_data("""
H 0 0 1.1
O 0 0 0
O 0 1.5 0
H 0 1.5 1.1
""")

torsiondrive_input = ptl.models.TorsionDriveInput(
    initial_molecule=[HOOH],
    keywords={"dihedrals": [[0, 1, 2, 3]], "grid_spacing": [18]},
    optimization_spec={"program": "geometric"},
    qc_spec={"program": "mopac", "method": "PM6", "driver": "gradient"},
)

identifier = client.add_service([torsiondrive_input])
```

```
<ComputeResponse(nsubmitted=1 nexisting=0)>
```

```python
torsiondrive_output = client.query_procedures(identifier.ids)
proc.get_final_energies(18)
```

```
-151.4746215927454
```



NATIONAL RESEARCH PLATFORM CAN RUN ENORMOUS QC WORKLOADS

Slides by:

Pavan Behara
Project Scientist, UCI Center for Neurotherapeutics
Openforcefield@David Mobley's Lab

QCSchema MANYBODY

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

QCSchema ManyBody

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodyInput

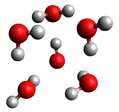
schema_version: 2

provenance: creator name and version info

`QCManyBody, 0.3.0`

id: tracker for databases

extras: free-form storage

molecule:  **Molecule**

specification: *how* to run MBE single-point

ManyBodySpecification

QCSchema MANYBODY

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodySpecification

schema_version: 2

program: any MBE speaking QCSchema

protocols: customize MBE return layout

ManyBodyProtocols

component_results: save cluster data

extras: free-form storage

driver: MBE target derivative

- energy
- gradient
- Hessian

E *G* *H*

keywords: knobs in the MBE program

ManyBodyKeywords

specification: how to run MBE *subsystems*

- single

AtomicSpecification

- mapping

{hi: **AtomicSpecification**,
lo: **AtomicSpecification** }

ManyBodyInput

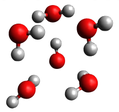
schema_version: 2

provenance: creator name and version info

`QCManyBody, 0.3.0`

id: tracker for databases

extras: free-form storage

molecule:  **Molecule**

specification: *how* to run MBE single-point

ManyBodySpecification

QCSchema ManyBody

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodyKeywords

bsse_type: any combination of
- nocp/mbe
- cp/ssfc
- vmfc

embedding_charges
return_total_data: IE or total
levels:
- {2: hi, 4: lo}
- {1: hi, 2: md, supersystem: lo}

max_nbody: truncate MBE early
supersystem_ie_only: no MBE analysis

ManyBodySpecification

schema_version: 2
program: any MBE speaking QCSchema
protocols: customize MBE return layout

ManyBodyProtocols
component_results: save cluster data

extras: free-form storage
driver: MBE target derivative
- energy
- gradient
- Hessian

E *G* *H*

keywords: knobs in the MBE program

ManyBodyKeywords

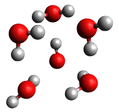
specification: how to run MBE *subsystems*
- single

AtomicSpecification

- mapping

{hi: **AtomicSpecification**,
lo: **AtomicSpecification** }

ManyBodyInput

schema_version: 2
provenance: creator name and version info
`QCManyBody, 0.3.0`
id: tracker for databases
extras: free-form storage
molecule:  **Molecule**

specification: *how* to run MBE single-point

ManyBodySpecification

QCSchema MANYBODY

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

AtomicSpecification

schema_version: 2
program: any SP in QCEngine
protocols: customize return layout

AtomicProtocols

stdout: save text output
wavefunction: save orbital info
native_files: save raw prog. files
error_correction: auto-edit&rerun

extras: free-form storage
driver: single-point derivative
- energy
- gradient
- Hessian
- properties

E G H

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

ManyBodySpecification

schema_version: 2
program: any MBE speaking QCSchema
protocols: customize MBE return layout

ManyBodyProtocols

component_results: save cluster data

extras: free-form storage
driver: MBE target derivative
- energy
- gradient
- Hessian

E G H

keywords: knobs in the MBE program

ManyBodyKeywords

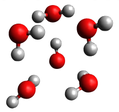
specification: how to run MBE *subsystems*
- single

AtomicSpecification

- mapping

{hi: AtomicSpecification,
lo: AtomicSpecification }

ManyBodyInput

schema_version: 2
provenance: creator name and version info
`QCManyBody, 0.3.0`
id: tracker for databases
extras: free-form storage
molecule:  **Molecule**

specification: *how* to run MBE single-point

ManyBodySpecification

QCSchema ManyBody

driving the many-body expansion

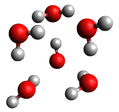
$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodyResult

schema_version: 2
input_data: record of starting conditions

ManyBodyInput

schema_version: 2
provenance: creator name and version info
`QCManyBody, 0.3.0`
id: tracker for databases
extras: free-form storage
molecule:  **Molecule**
specification: *how* to run MBE single-point

ManyBodySpecification

success: True
provenance: creator, version, & RT info
`QCManyBody, 0.4.0, psinet, CPU E5-2630`
id: tracker for databases
stdout: program's text output for humans
stderr: anything in error channel
native_files: requested raw program files
extras: free-form storage
molecule: orientation & frame for results
return_result: MBE derivative

properties: key results for MBE

ManyBodyProperties

cluster_properties: key results for subsystems
{cluster id: **AtomicProperties** }
cluster_results: full results for subsystems
{cluster id: **AtomicResult** }

QCSchema ManyBody

driving the many-body expansion

$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodyProperties (abr.)

calcinfo_nfr: M

return_energy: E

return_gradient: G

bsse_corrected_interaction_egh_through_m_body:
- defined as E^m_{IE}
- defined for all *bsse* as **bsse_type** allows
- defined for all *egh* as **driver** allows
- def. for all *m* as **max_nbody/levels** allows

bsse_corrected_total_egh_through_m_body:
- defined as E^m

bsse_corrected_m_body_contribution_to_egh:
- defined as $E^{(m)}$

ManyBodyResult

schema_version: 2

input_data: record of starting conditions

ManyBodyInput

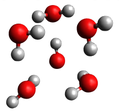
schema_version: 2

provenance: creator name and version info

`QCManyBody, 0.3.0`

id: tracker for databases

extras: free-form storage

molecule:  **Molecule**

specification: *how* to run MBE single-point

ManyBodySpecification

success: True

provenance: creator, version, & RT info

`QCManyBody, 0.4.0, psinet, CPU E5-2630`

id: tracker for databases

stdout: program's text output for humans

stderr: anything in error channel

native_files: requested raw program files

extras: free-form storage

molecule: orientation & frame for results

return_result: MBE derivative

E G H

properties: key results for MBE

ManyBodyProperties

cluster_properties: key results for subsystems

{cluster id: **AtomicProperties** }

cluster_results: full results for subsystems

{cluster id: **AtomicResult** }

QCSchema ManyBody

driving the many-body expansion

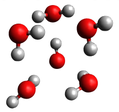
$$E^M = E^1 + E^{(2)} + E^{(3)} + \cdots + E^{(M)},$$
$$= \sum_{I \in \mathcal{P}_1} E^I + \sum_{IJ \in \mathcal{P}_2} E^{(IJ)} + \sum_{IJK \in \mathcal{P}_3} E^{(IJK)} + \cdots,$$

- **RECENT** added in 2024
- Counterpoise, VMFC, and no-CP arbitrary-order manybody expansion (**MBE**) extracted from Psi4 program
- **SUITABLE** for intermolecular fragments, no bond-breaking

ManyBodyResult

schema_version: 2
input_data: record of starting conditions

ManyBodyInput

schema_version: 2
provenance: creator name and version info
`QCManyBody, 0.3.0`
id: tracker for databases
extras: free-form storage
molecule:  **Molecule**
specification: *how* to run MBE single-point

ManyBodySpecification

success: True
provenance: creator, version, & RT info
`QCManyBody, 0.4.0, psinet, CPU E5-2630`
id: tracker for databases
stdout: program's text output for humans
stderr: anything in error channel
native_files: requested raw program files
extras: free-form storage
molecule: orientation & frame for results
return_result: MBE derivative

properties: key results for MBE

ManyBodyProperties

cluster_properties: key results for subsystems
{cluster id: **AtomicProperties** }
cluster_results: full results for subsystems
{cluster id: **AtomicResult** }

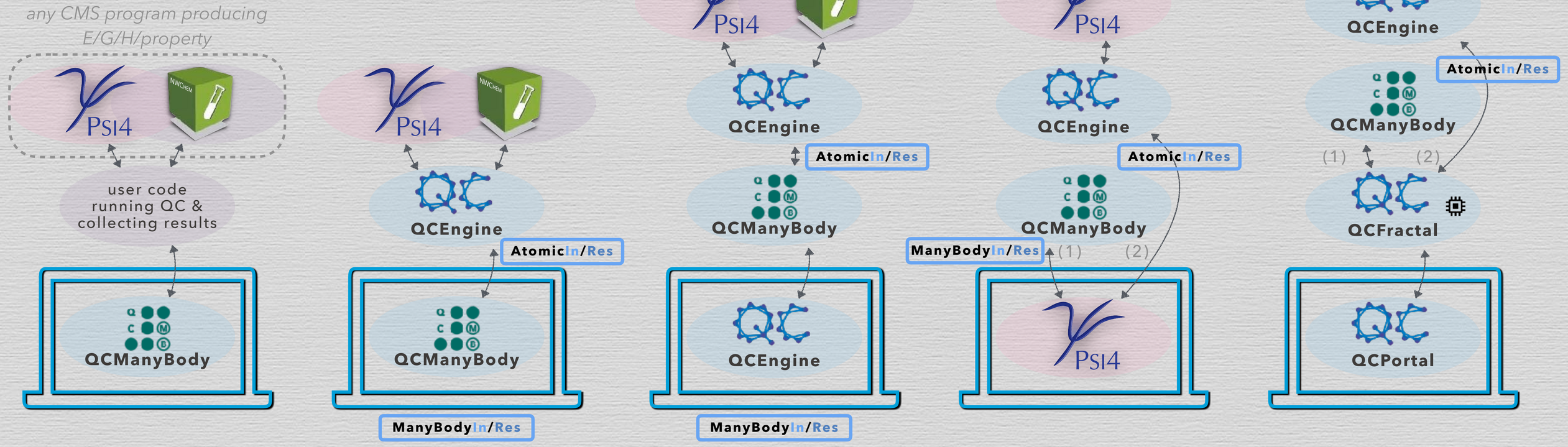
APP: QCMANYBODY

multiple entry points for flexibility

Ben Pritchard
MolSSI

Asim Alenaizan
KFUPM

Daniel Smith
MolSSI → Abiologics



QCManyBody core interface

- user-prov. serial/parallel
- user runs CMS backends

QCManyBody high-level interface

- serial
- multi CMS backends

QCEngine procedure

- serial
- multi CMS backends

PSI4 distributed driver

- serial or parallel
- only Psi4 backend

QCFractal service

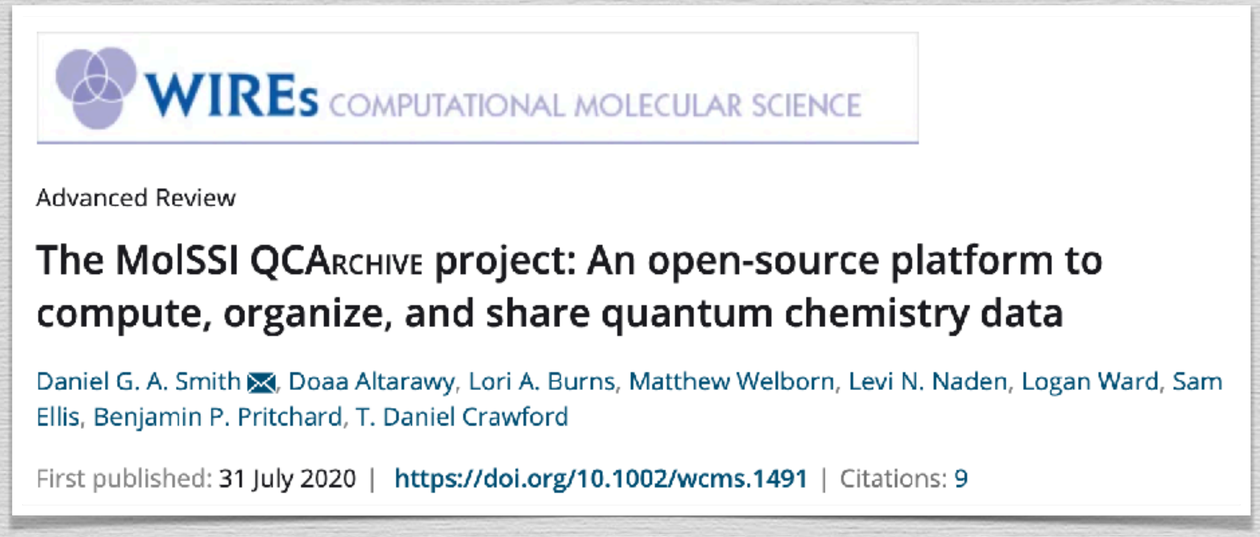
- parallel
- multi CMS backends

WHAT QCARCHIVE DID RIGHT

a lot, in my opinion; DGAS timeline

- The QCFractal/OpenFF collaboration illustrates the most important thing MolSSI did right, imo. They prioritized early on in building a **DEEP** science-results-focused operations stack atop their projects that would exercise and polish the base software (QCSchema and the rest of QCArchive Infrastructure).
- Secondly, they found good **SEPARATIONS OF CONCERN** for the software stack and solidified projects around those missions – define data layout (QCSk), implement schema with resources (QCEl), schema runner (QCEngine), and orchestrate & store (QCFractal) – so that the community can nibble or commit.
- They sped ahead of consensus into practice with QCElemental implementations to evaluate which ideas worked under research conditions.
- They designed schemas are as flat as possible, consistent with reusability, so easier for users to remember and type. QCSchema isn't the broadest schema out there, but it has a track record for **READABILITY** and learnability.
- Previous QCSchema models weren't dated because they've been in recognizable form and **STABLE** with minor refinements since about 2019.

- **Nov 2016** Psi4 v1.1 (May 2017) running pre-QCSchema JSON
- **Jan 2017** MolSSI gets populated
- **Dec 2017** meeting at LBNL to discuss QCSchema
- **Jan 2019** QCElemental starts QCSchema models
- **Mar 2019** ACS announcement of beta QCArchive
- **Jul 2020** QCArchive paper


WIREs COMPUTATIONAL MOLECULAR SCIENCE

Advanced Review

The MolSSI QCARCHIVE project: An open-source platform to compute, organize, and share quantum chemistry data

Daniel G. A. Smith, Doaa Altarawy, Lori A. Burns, Matthew Welborn, Levi N. Naden, Logan Ward, Sam Ellis, Benjamin P. Pritchard, T. Daniel Crawford

First published: 31 July 2020 | https://doi.org/10.1002/wcms.1491 | Citations: 9

WHAT QCARCHIVE DID NOT SO RIGHT

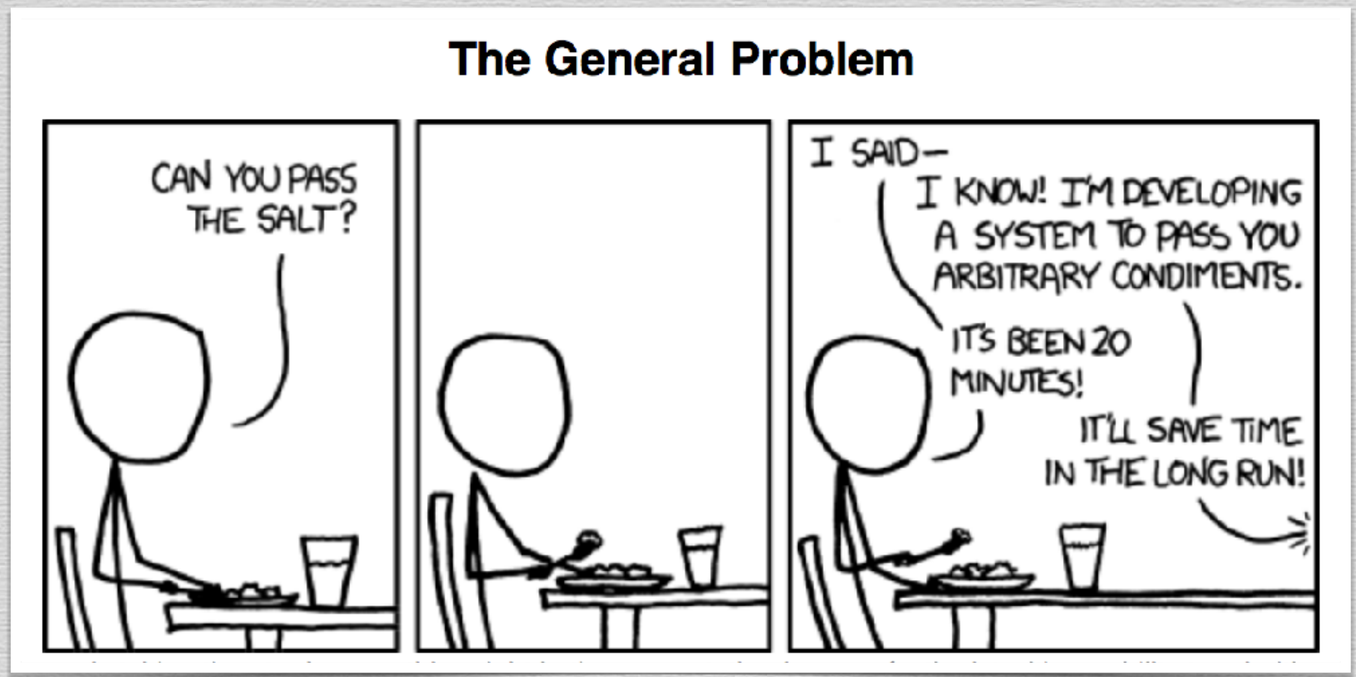
and BPP timeline

- Schema files (like Julia interfaces) are more long-lived than API usage, so it's more **BREAKING** to make changes. Being a foundational dependency of many projects, it was easier to not make changes.
- Regarding schema changes, I think MolSSI expected consensus on contributions. When that didn't ocurr (or there weren't enough opinions), tended to implement a likely candidate in QCElemental for testing and never update QCSchema repository, so **DIVERGENCE**.
- Didn't plan a governance or upgrade **POLICY**
- **RESEARCH** demands some things downstream needs get implemented quickly, lest they be blocked. Research also demands that your dependencies be stable and not interrupted by new nonsense fields.
- Not a fault, just a fact: long-lived branches, like the major QCFractal "next" rewrite from 2021–2023, largely **FREEZE** other parts of the ecosystem until merged.

- **Spring 2020** BPP takes over QCArchive
- **Spring 2021** BPP starts QCFractal "next" overhaul
- **July 2021** Psi4 v1.4 requires QCEngine to run dispersion corrections
- **Nov 2021** QCEngine paper
- **May 2022** Psi4 v1.6 uses QCSchema for every finite difference derivative, counterpoise correction, or basis set extrapolation
- **July 2022** mtg at MQM to discuss QCSchema improvements
- **August 2023** scurry to prep projects for Pydantic v2
- **Sep 2023** QCFractal "next" replaces legacy
  - complete rewrite with better performance
  - greater flexibility incl. software environments
  - able to chage job status, delete records, rerun
- **Fall 2024** prepare for first ever major schema chg.

QCArchive & QCSchema

whether perfectionist, artisan, or fades away remains to be seen



https://xkcd.com/974/

I find that when someone's taking time to do something right in the present, they're a perfectionist with no ability to prioritize, whereas when someone took time to do something right in the past, they're a master artisan of great foresight.

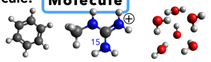
QCSchema v2

v1 has duplicate schema and non-standard names and fields

- QCSchema v1 has been stable 2019–present
- **Pydantic V2** released June 2023
  - model changes are mild to moderate
  - forbidden to *mix* v1/v2 models, so concurrent full-stack (QCElemental, QCEngine, QCFractal, Psi4, OpenFF) required
- the longstanding programs schema (AtomicInput/Result) well-designed and -developed (follows previous dev in psi4), but procedures schemas were ad hoc and thus both repetitive and missing features for uniformity
  - subschemas **OVERLAP**, hindering predictability
  - CMS code specification **SCATTERED** among argument, named field, and key/value
  - input difficult to extract since fields **OVERWRITE**
  - some QOL fields (`extras`, `protocols`, `id`) **MISSING**. Also, some schema names vary from pattern
- All of this calls for a **TECHNICAL** overhaul

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



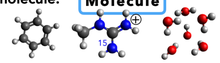
protocols: customize return layout

driver: single-point derivative
- **energy**
- **gradient** E G H
- **Hessian**
- **properties**

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

OptimizationInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
initial_molecule: **Molecule**



protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- **energy**
- **gradient** E G H
- **Hessian**
- **properties**

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

TorsionDriveInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`

extras: free-form storage
initial_molecule: **Molecule**

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan
dihedral_ranges: limits of scan
grid_spacing: IE dihedral steps of scan
energy_decrease_thresh: point trigger
energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1
procedure: any optimizer in QCEngine
pyberny
protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- **energy**
- **gradient** E G H
- **Hessian**
- **properties**

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

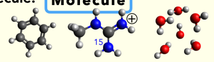
QCSchema v2

v1 has duplicate schema and non-standard names and fields

- QCSchema v1 has been stable 2019–present
- **Pydantic V2** released June 2023
  - model changes are mild to moderate
  - forbidden to *mix* v1/v2 models, so concurrent full-stack (QCElemental, QCEngine, QCFractal, Psi4, OpenFF) required
- the longstanding programs schema (AtomicInput/Result) well-designed and -developed (follows previous dev in psi4), but procedures schemas were ad hoc and thus both repetitive and missing features for uniformity
  - subschemas **OVERLAP**, hindering predictability
  - CMS code specification **SCATTERED** among argument, named field, and key/value
  - input difficult to extract since fields **OVERWRITE**
  - some QOL fields (`extras`, `protocols`, `id`) **MISSING**. Also, some schema names vary from pattern
- All of this calls for a **TECHNICAL** overhaul

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**

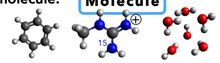


protocols: customize return layout
driver: single-point derivative
- energy
- gradient E G H
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

OptimizationInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
initial_molecule: **Molecule**



protocols: customize opt return layout
keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1
extras: free-form storage
driver: single-point derivative
- energy
- gradient E G H
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

TorsionDriveInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
extras: free-form storage
initial_molecule: **Molecule**

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan
dihedral_ranges: limits of scan
grid_spacing: IE dihedral steps of scan
energy_decrease_thresh: point trigger
energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1
procedure: any optimizer in QCEngine
pyberny
protocols: customize opt return layout
keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1
extras: free-form storage
driver: single-point derivative
- energy
- gradient E G H
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

AtomicInput ≅ **QCInputSpecification** + **Molecule**

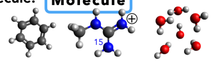
QCSchema v2

v1 has duplicate schema and non-standard names and fields

- QCSchema v1 has been stable 2019–present
- **Pydantic V2** released June 2023
  - model changes are mild to moderate
  - forbidden to *mix* v1/v2 models, so concurrent full-stack (QCElemental, QCEngine, QCFractal, Psi4, OpenFF) required
- the longstanding programs schema (AtomicInput/Result) well-designed and -developed (follows previous dev in psi4), but procedures schemas were ad hoc and thus both repetitive and missing features for uniformity
  - subschemas **OVERLAP**, hindering predictability
  - CMS code specification **SCATTERED** among argument, named field, and key/value
  - input difficult to extract since fields **OVERWRITE**
  - some QOL fields (`extras`, `protocols`, `id`) **MISSING**. Also, some schema names vary from pattern
- All of this calls for a **TECHNICAL** overhaul

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



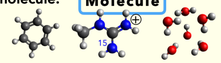
protocols: customize return layout

driver: single-point derivative
- energy
- gradient *E* *G* *H*
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

OptimizationInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
initial_molecule: **Molecule**



protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- energy
- gradient *E* *G* *H*
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

TorsionDriveInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`

extras: free-form storage
initial_molecule: **Molecule**

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan
dihedral_ranges: limits of scan
grid_spacing: IE dihedral steps of scan
energy_decrease_thresh: point trigger
energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1
procedure: any optimizer in QCEngine
pyberny
protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:` Psi4 `}`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- energy
- gradient *E* *G* *H*
- Hessian
- properties

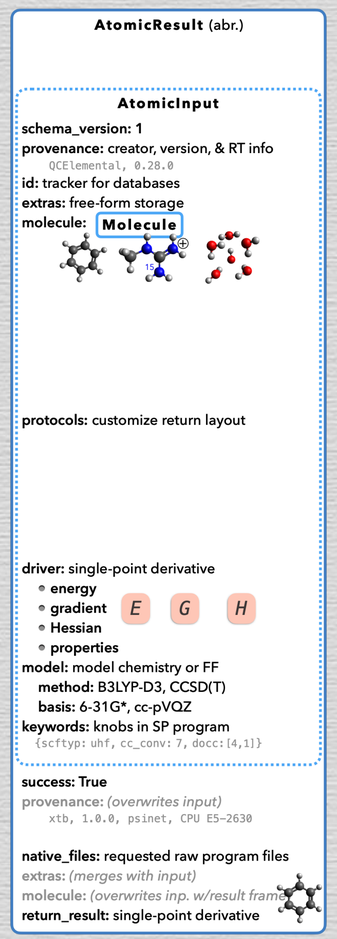
model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

OptimizationInput ≅ **OptimizationSpecification** + **QCInputSpecification** + **Molecule**

QCSchema v2

v1 has duplicate schema and non-standard names and fields

- QCSchema v1 has been stable 2019–present
- **Pydantic V2** released June 2023
  - model changes are mild to moderate
  - forbidden to *mix* v1/v2 models, so concurrent full-stack (QCElemental, QCEngine, QCFractal, Psi4, OpenFF) required
- the longstanding programs schema (AtomicInput/Result) well-designed and -developed (follows previous dev in psi4), but procedures schemas were ad hoc and thus both repetitive and missing features for uniformity
  - subschemas **OVERLAP**, hindering predictability
  - CMS code specification **SCATTERED** among argument, named field, and key/value
  - input difficult to extract since fields **OVERWRITE**
  - some QOL fields (`extras`, `protocols`, `id`) **MISSING**. Also, some schema names vary from pattern
- All of this calls for a **TECHNICAL** overhaul



QCSchema v2

v1 has duplicate schema and non-standard names and fields

- QCSchema v1 has been stable 2019–present
- **Pydantic V2** released June 2023
  - model changes are mild to moderate
  - forbidden to *mix* v1/v2 models, so concurrent full-stack (QCElemental, QCEngine, QCFractal, Psi4, OpenFF) required
- the longstanding programs schema (AtomicInput/Result) well-designed and -developed (follows previous dev in psi4), but procedures schemas were ad hoc and thus both repetitive and missing features for uniformity
  - subschemas **OVERLAP**, hindering predictability
  - CMS code specification **SCATTERED** among argument, named field, and key/value
  - input difficult to extract since fields **OVERWRITE**
  - some QOL fields (`extras`, `protocols`, `id`) **MISSING**. Also, some schema names vary from pattern
- All of this calls for a **TECHNICAL** overhaul

```
sh> qcengine run atin
```

```
sh> qcengine run-procedure tdin
```

```
sh> qcengine run-procedure optin
```

schema_version: 1

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

OptimizationInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
initial_molecule: **Molecule**

protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- energy
- gradient E G H
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

TorsionDriveInput

schema_version: 1
provenance: creator name and version info
`QCElemental, 0.28.0`

extras: free-form storage
initial_molecule: **Molecule**

keywords: knobs in the TD program

TDKeywords

dihedrals: dimensions to scan
dihedral_ranges: limits of scan
grid_spacing: IE dihedral steps of scan
energy_decrease_thresh: point trigger
energy_upper_limit: new point trigger

optimization_spec: how to run *geometry opts*

OptimizationSpecification

schema_version: 1
procedure: any optimizer in QCEngine
pyberny
protocols: customize opt return layout

keywords: knobs in the geo opt program
`{coordsys: tric, program:`
input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage
driver: single-point derivative
- energy
- gradient E G H
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

Schmidt Sciences
Virtual Institute for Scientific Software

Georgia Tech College of Computing
Center for Scientific Software Engineering

QCSchema v2

underway

- Note technical, **NOT FEATURE**, overhaul
- opportunity to work with **GT CSSE** (Center for Scientific Software Engineering) for this semester
- overhaul planned in two phases
  - 0. allow QCSchema v1 & v2 to be imported from QCElemental module for broad **ACCESS** — Done! Sep 2024
  - 1. **UPDATE** QCSk v2 models to Pydantic v2 — Done! Oct 2024
  - 2. rearrange model **LAYOUT** for composability, standardiation, and predictability — NYI, next!
- wait a year for v2 to percolate through ecosystem

ACCESS QCSk v1 (PYDANTIC v1 API)

```
py> from qcelemental.models import AtomicInput
py> from qcelemental.models.v1 import AtomicInput
```

ACCESS QCSk v2 (PYDANTIC v2 API)

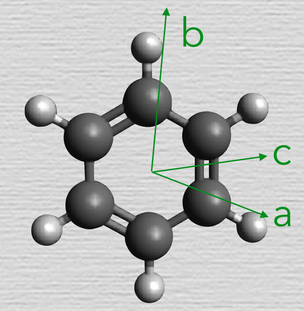
```
py> from qcelemental.models.v2 import AtomicInput
```

AtomicInput v2

minor changes

- Biggest change in **AtomicInput** is that ***HOW* FIELDS** move down a layer into a new **AtomicSpecification**, while *what* fields, mainly **Molecule**, remain at the top layer
- **AtomicResultProtocols** describes *how* to run SP so it **NESTS** within the specification and also receives a simpler name
- The single-point **CMS PROGRAM** can now be specified within the schema, as well as by function argument to `compute()`
- There's technical reasons schema v1 & v2 shouldn't be identical, so for Molecule, we're adding the periodic box vectors.

how best to specify?



RUN QCSk v2

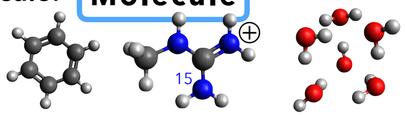
```
py> qcengine.compute(atin, )

py> qcengine.compute(atin)
```

RUN QCSk v1

```
py> qcengine.compute(atin, )
```

AtomicInput

schema_version: 1
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



protocols: customize return layout

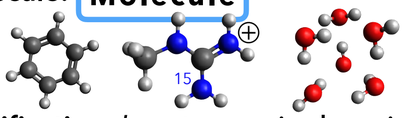
AtomicResultProtocols
stdout: save text output
wavefunction: save orbital info
native_files: save raw prog. files
error_correction: auto-edit&rerun

driver: single-point derivative
- energy
- gradient *E* *G* *H*
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

AtomicInput

schema_version: 2
provenance: creator, version, & RT info
`QCElemental, 0.28.0`
id: tracker for databases
extras: free-form storage
molecule: **Molecule**



specification: *how* to run single-point

AtomicSpecification
schema_version: 2
program: any SP in QCEngine
protocols: customize return layout

AtomicProtocols
stdout: save text output
wavefunction: save orbital info
native_files: save raw prog. files
error_correction: auto-edit&rerun

extras: free-form storage
driver: single-point derivative
- energy
- gradient *E* *G* *H*
- Hessian
- properties

model: model chemistry or FF
method: B3LYP-D3, CCSD(T)
basis: 6-31G*, cc-pVQZ
keywords: knobs in SP program
`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

AtomicResult v2

compose, not inherit

- Biggest change in **AtomicResult** is that input and results fields are now clearly distinguished
- Result and subschema get independent `schema_version`
- Both input (for restart) and final (for result properties frame) **Molecule** orientations are now accessible
- Fields `provenance` & `extras` are results-only
- **AtomicProperties** now versioned & name standardized
- **WAVEFUNCTION** changes discussed elsewhere

AtomicResult

AtomicInput (abr.)

schema_version: **1**

success: True

provenance: *(overwrites input)*

`xtb, 1.0.0, psinet, CPU E5-2630`

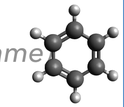
stdout: program's text output for humans

stderr: anything in error channel

native_files: requested raw program files

extras: *(merges with input)*

molecule: *(overwrites inp w/result frame)*



return_result: single-point derivative

properties: key results for single-point

AtomicResultProperties (abr.)

calcinfo_natom

calcinfo_nmo

nuclear_repulsion_energy

return_energy: E

return_gradient: G

scf_total_*egh*: ref. energy, gradient, etc.

***mtd*_correlation_energy:** post-scf energy

***mtd*_iterations:** count to convergence

***mtd*_mpole_moment:** dipole etc. array

wavefunction: orbital-level results

WavefunctionProperties

AtomicResult

schema_version: **2**

input_data: record of starting conditions

AtomicInput (abr.)

schema_version: **2**

success: True

provenance: creator, version, & RT info

`xtb, 1.0.0, psinet, CPU E5-2630`

id: tracker for databases

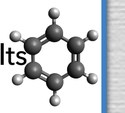
stdout: program's text output for humans

stderr: anything in error channel

native_files: requested raw program files

extras: free-form storage

molecule: orientation & frame for results



return_result: single-point derivative

properties: key results for single-point

AtomicProperties (abr.)

schema_version: **2**

calcinfo_natom

calcinfo_nmo

nuclear_repulsion_energy

return_energy: E

return_gradient: G

scf_total_*egh*: ref. energy, gradient, etc.

***mtd*_correlation_energy:** post-scf energy

***mtd*_iterations:** count to convergence

***mtd*_mpole_moment:** dipole etc. array

wavefunction: orbital-level results

WavefunctionProperties

OptimizationInput v2

bigger changes

- Biggest change in **OptimizationInput** is that ***HOW OPT*** fields move down a layer into a new **OptimizationSpecification**, while the *what opt* fields (**Molecule**) remains at top layer
- Also, *how opt* fields cleanly sep. from *how gradient* fields
- Optimization can reuse **AtomicSpecification** from **AtomicInput**
- Runner programs specified in fields (or opt. on the cmdine) rather than mixed cmdline, named field, and dictionary key
- Able to allow **OptimizationInput** to direct more complicated manybody jobs, not just SP

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

RUN QCSk v2

```
py> qcengine.compute(optin, )

py> qcengine.compute(optin)
```

OptimizationInput

schema_version: 1

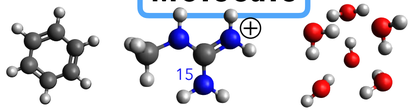
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



protocols: customize opt return layout

keywords: knobs in the geo opt program

`{coordsys: tric, program: }`

input_specification: how to run *gradients* for opt

OptimizationInput

schema_version: 2

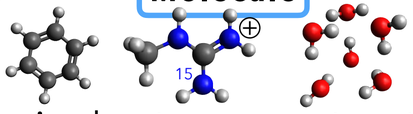
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



specification: how to run *geometry opts*

OptimizationInput v2

bigger changes

- Biggest change in **OptimizationInput** is that ***HOW OPT*** fields move down a layer into a new **OptimizationSpecification**, while the *what opt* fields (**Molecule**) remains at top layer
- Also, *how opt* fields cleanly sep. from *how gradient* fields
- Optimization can reuse **AtomicSpecification** from **AtomicInput**
- Runner programs specified in fields (or opt. on the cmdine) rather than mixed cmdline, named field, and dictionary key
- Able to allow **OptimizationInput** to direct more complicated manybody jobs, not just SP

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

RUN QCSk v2

```
py> qcengine.compute(optin, )

py> qcengine.compute(optin)
```

OptimizationInput

schema_version: 1

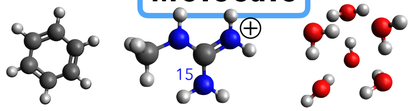
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program: }`

input_specification: how to run *gradients* for opt

OptimizationInput

schema_version: 2

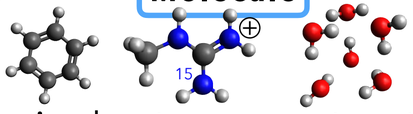
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



specification: how to run *geometry opts*

OptimizationSpecification

schema_version: 2

program: any optimizer in QCEngine

pyberny

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

extras: free-form storage

keywords: knobs in the geo opt program

`{coordsys: tric, maxiter: 100}`

specification: how to run *gradients* for opt

- single-point

OptimizationInput v2

bigger changes

- Biggest change in **OptimizationInput** is that ***HOW OPT*** fields move down a layer into a new **OptimizationSpecification**, while the *what opt* fields (**Molecule**) remains at top layer
- Also, *how opt* fields cleanly sep. from *how gradient* fields
- Optimization can reuse **AtomicSpecification** from **AtomicInput**
- Runner programs specified in fields (or opt. on the cmdine) rather than mixed cmdline, named field, and dictionary key
- Able to allow **OptimizationInput** to direct more complicated manybody jobs, not just SP

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

RUN QCSk v2

```
py> qcengine.compute(optin, )

py> qcengine.compute(optin)
```

OptimizationInput

schema_version: 1

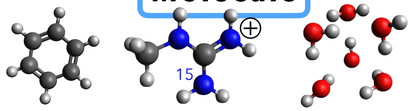
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program: }`

input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

OptimizationInput

schema_version: 2

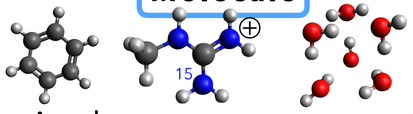
provenance: creator name and version info

`QCElemental, 0.28.0`

id: tracker for databases

extras: free-form storage

initial_molecule: **Molecule**



specification: how to run *geometry opts*

OptimizationSpecification

schema_version: 2

program: any optimizer in QCEngine

pyberny

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

extras: free-form storage

keywords: knobs in the geo opt program

`{coordsys: tric, maxiter: 100}`

specification: how to run *gradients* for opt

- single-point

AtomicSpecification

schema_version: 2

program: any SP in QCEngine

protocols: customize return layout

AtomicProtocols

stdout: save text output

wavefunction: save orbital info

native_files: save raw prog. files

error_correction: auto-edit&rerun

OptimizationInput v2

bigger changes

- Biggest change in **OptimizationInput** is that ***HOW OPT*** fields move down a layer into a new **OptimizationSpecification**, while the *what opt* fields (**Molecule**) remains at top layer
- Also, *how opt* fields cleanly sep. from *how gradient* fields
- Optimization can reuse **AtomicSpecification** from **AtomicInput**
- Runner programs specified in fields (or opt. on the cmdine) rather than mixed cmdline, named field, and dictionary key
- Able to allow **OptimizationInput** to direct more complicated manybody jobs, not just SP

RUN QCSk v1

```
py> qcengine.compute_procedure(optin, )
```

RUN QCSk v2

```
py> qcengine.compute(optin, )

py> qcengine.compute(optin)
```

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

keywords: knobs in the geo opt program

`{coordsys: tric, program: }`

input_specification: how to run *gradients* for opt

QCInputSpecification

schema_version: 1

extras: free-form storage

driver: single-point derivative

- **energy**
- **gradient** E G H
- **Hessian**
- **properties**

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

`{scftyp: uhf, cc_conv: 7, docc: [4,1]}`

pyberny

protocols: customize opt return layout

OptimizationProtocols

trajectory: save fewer opt steps

extras: free-form storage

keywords: knobs in the geo opt program

`{coordsys: tric, maxiter: 100}`

specification: how to run *gradients* for opt

- single-point

AtomicSpecification

schema_version: 2

program: any SP in QCEngine

protocols: customize return layout

AtomicProtocols

stdout: save text output

wavefunction: save orbital info

native_files: save raw prog. files

error_correction: auto-edit&rerun

extras: free-form storage

driver: single-point derivative

- **energy**
- **gradient** E G H
- **Hessian**
- **properties**

model: model chemistry or FF

method: B3LYP-D3, CCSD(T)

basis: 6-31G*, cc-pVQZ

keywords: knobs in SP program

`{scftyp: uhf, cc_conv: 7, docc:[4,1]}`

- MBE

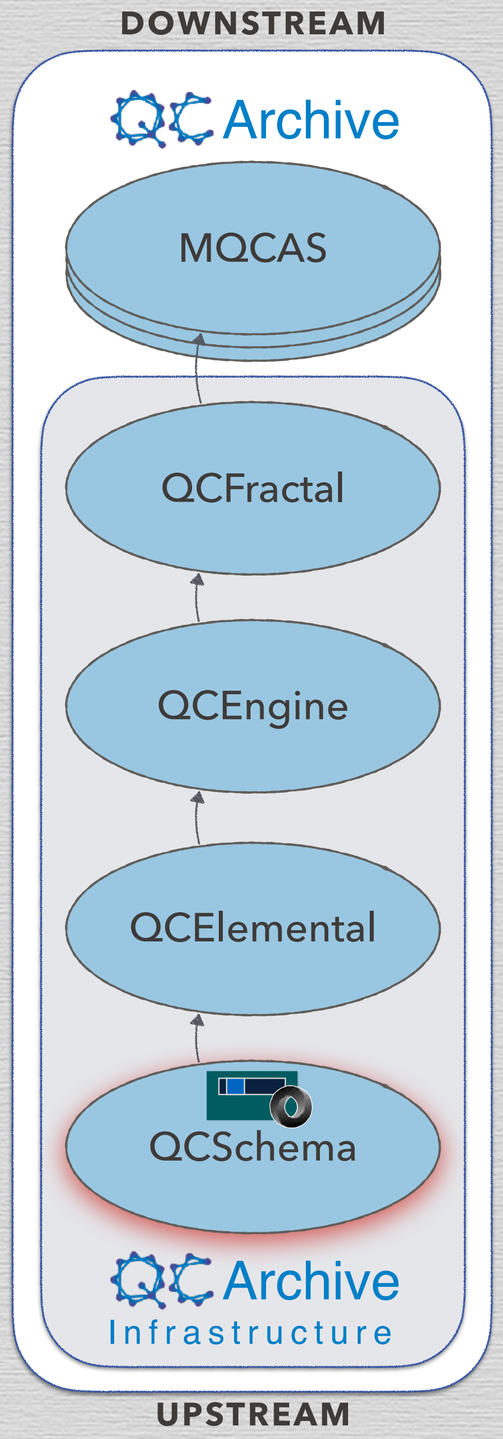
ManyBodySpecification

BEYOND QCSCHEMA V2

next phase for QCA stack

- revive QCSchema repository for pure **JSON** schema exported from (simpler than QCELEMENTAL) Pydantic models
- coordinate closer with **CCLIB** for QCENGINE
- explore connections with TREXIO, NOMAD, etc.
- "manage" certain capabilities by cross-implementing and registering labels in QCENGINE and tracking in QCSCHEMA
  - input schema **BasisSet**, not just string name
  - point charges for quantum chemistry
  - solvation
- restart talks with community on **V3 FEATURES**

  medium-term (1–2y) requests?
- set up community governance so QCSchema respository become the source of truth and a steering committee reviews changes fast enough to get research done and slow enough to allow research to get done :-)



github.com/MolSSI/QCSchema

- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

Psi4

Jerome Gonthier
Berkeley → QC Ware

Rob Parrish
Stanford → QC Ware

Holger Kruse
Czech Acad. Sci.

Rollin King
Bethel

Alexander Sokolov
Ohio State

David Sherrill
GaTech

Lori Burns
GaTech

Asim Alenaizan
KFUPM

Maximilian Scheurer
Heidelberg → Covestro

Zach Glick
GaTech → Lavo

Jeff Schriber
Iona

Francesco Evangelista
Emory

Eugene DePrince
FSU

Fritz Schaefer
UGA

Justin Turney
UGA

Daniel Crawford
VaTech

Daniel Smith
MolSSI → Abiologics

Konrad Patkowski
Auburn

Ben Pritchard
MolSSI

Jonathon Misiewicz
VaTech

Ed Valeev
VaTech

Andy Simmonett
NIH → QC Ware

Ed Hohenstein
Stanford → QC Ware

Roberto Di Remigio
ENCCS

Ugur Bozkaya
Hacettepe

Peter Kraus
TUB

Susi Lehtola
Helsinki

QCENGINE

ADCC

Maximilian Scheurer
Heidelberg

Michael Herbst
EPFL

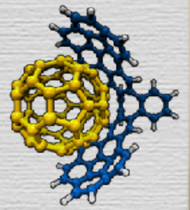
Andreas Dreuw
Heidelberg

CFOUR

John Stanton
UFL

Devin Matthews
SMU

SE / DISP.



Sebastian Ehlert
Bonn

Holger Kruse
Czech Acad. Sci.

Jiri Šponer
Czech Acad. Sci.

GAMESS

Mark Gordon
Iowa State

Nuwan de Silva
W. New England U

ML

Farhad Ramezanghorbani
UFL → Schrödinger

MOLPRO

Sebastian Lee
CalTech

MRCHEM

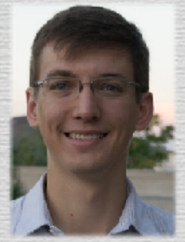
Roberto Di Remigio
Uppsala

NWCHEM

Theresa Windus
Iowa State

Jiyoung Lee
U Texas

Annabelle Lolinco
Iowa State


Logan Ward
Argonne

Adrian Hurtado
Stony Brook

OPENFF / MM

John Chodera
MSKCC

Jeffrey Wagner
UC Irvine

David Dotson
UC Boulder

Joshua Horton
Newcastle

Jamshed Anwar
Lancaster

TERACHEM

Fang Liu
Emory

Heather Kulik
MIT

Colton Hicks
Stanford

Todd Martínez
Stanford

TURBOMOLE

Johannes Steinmetzer
Friedrich Schiller U

HIST./DB/TEST

Nick Petosa
GaTech → Two Sigma

Daniel Nascimento
U Memphis

Dom Sirianni
GaTech

Chaya Stern
MSKCC

ENGINES

Taylor Barnes
MolSSI

OPTIMIZERS

Lee-Ping Wang
UC Davis

Jan Hermann
Free U Berlin

Alexander Heide
UGA

Rollin King
Bethel

PROCEDURES

Asim Alenaizan
GaTech → KFUPM

Peter Kraus
Curtin

Jonathon Misiewicz
Emory

Jeff Schriber
GaTech → Iona

Carlos Borca
Princeton

Philip Nelson
GaTech

QCArchive Acknowledgements

QCArchive

Daniel Smith
MolSSI → Abiologics

Levi Naden
MolSSI

Doaa Altarawy
MolSSI → Alexandria

Matthew Welborn
MolSSI → Entos

Ben Pritchard
MolSSI

MolSSI

Theresa Windus
Iowa State

Daniel Crawford
VaTech

Jessica Nash
MolSSI

Sam Ellis
MolSSI

Susi Lehtola
Helsinki

Open Force Field Consortium

openforcefield.org

psicode.org

molssi.github.io/QCFractal

QCSchema

Daniel Smith
MolSSI → Abiologics

Aaron Virshup
Arzeda

Bert de Jong
LBNL

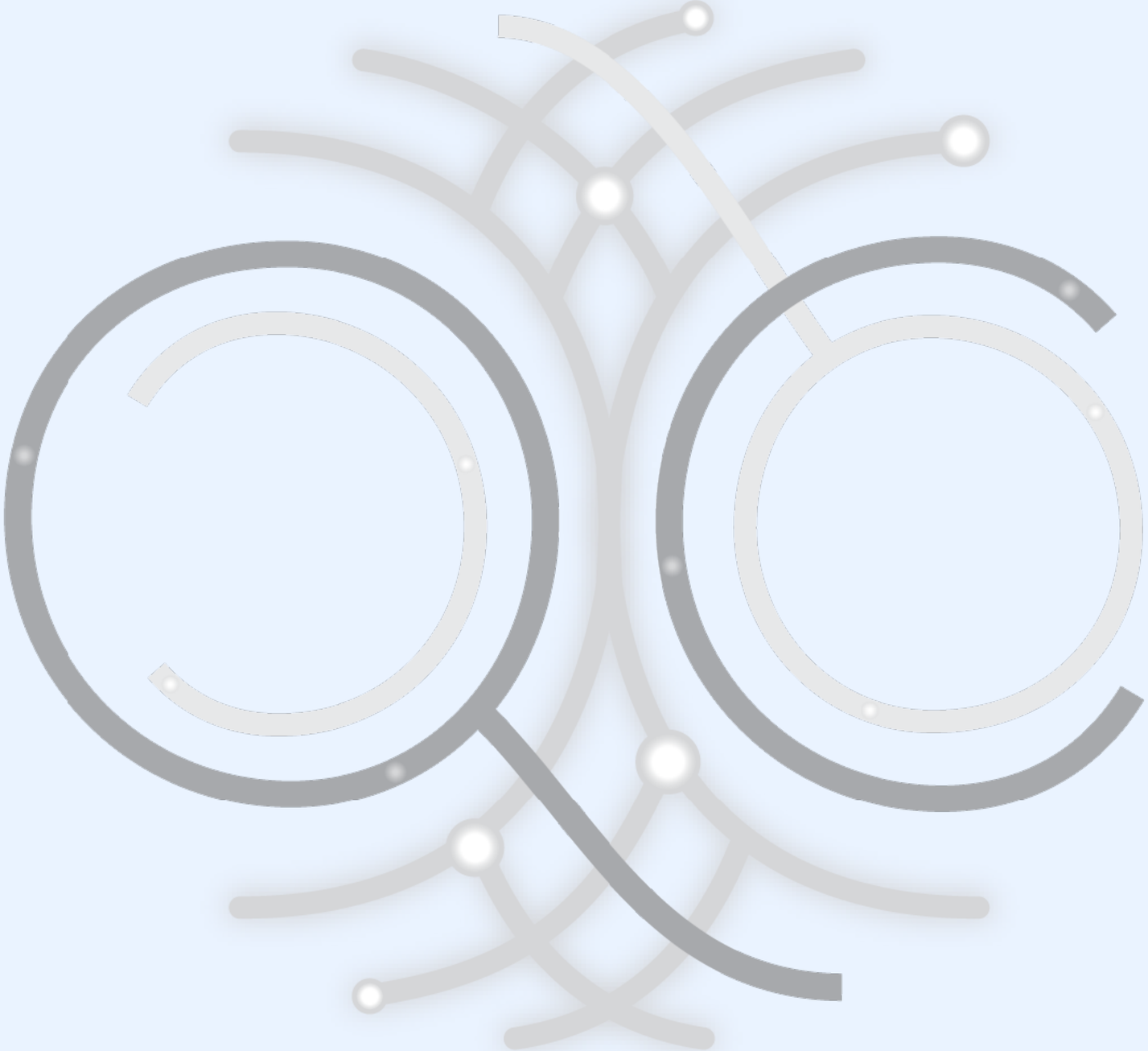
Geoff Hutchison
U Pittsburgh

Marcus Hanwell
Kitware → Voltron

Eric Berquist
Sandia

Ben Pritchard
MolSSI

Lori Burns
GaTech

Matthew Welborn
MolSSI → Entos

Colton Hicks
Stanford