Simplifying Multilevel Quantum Chemistry Procedures through Psi4 and QCArchive

LORI A. BURNS

MOLSSI WORKSHOP ON HPC IN COMPUTATIONAL CHEMISTRY & MATERIALS SCIENCE, VIRTUAL

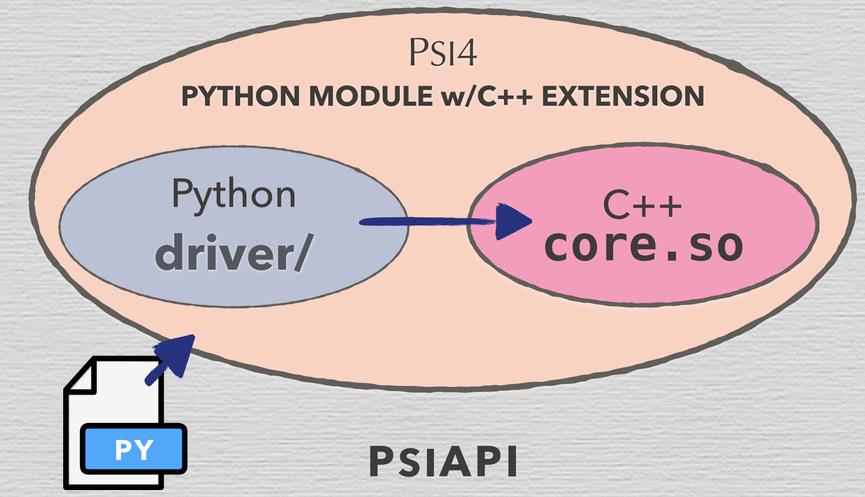
14 DECEMBER 2021

MOLSSI

NSF

PSI4 **ORIENTATION**

v1.5 released 2021



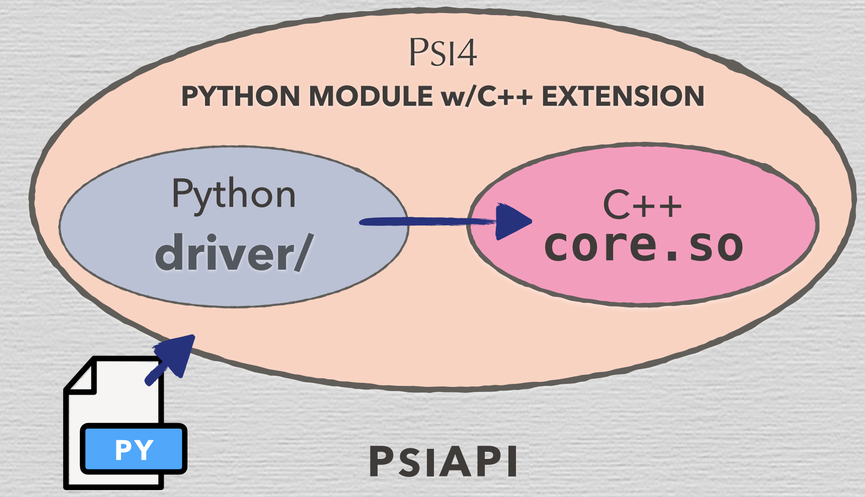

```
import psi4

psi4.geometry('''
Ne
Ne 1 3.0
''')

psi4.set_options({
 'freeze_core': 'True'})

psi4.energy('ccsd(t)/cc-pvtz')
```

> python in.py

PSI4 ORIENTATION

v1.5 released 2021

- **FULL-FEATURED** (lots of initialisms)
- **SMP** parallelism
- **HYBRID** C++/Python
  - ~70% C++
  - ~30% Python
- **INPUT** by DSL, API, or JSON
- **STRIVE** for easy production-quality user input: `energy()`, `gradient()`, `optimize()`, `hessian()`, `frequency()` are the five driver "user-facing" functions through which 99+% of QC is run in Psi4, so easy to guess command. Minimal entry points
- **SPECIALTIES** include density-fitting & intermolecular interactions



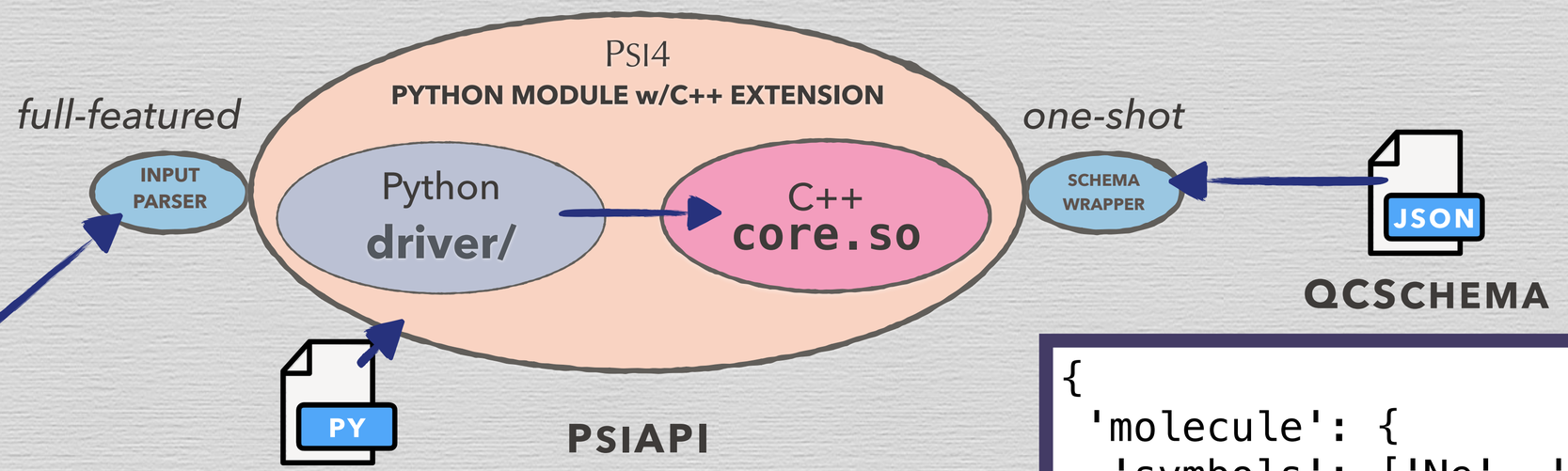
PSIAPI

```
import psi4

psi4.geometry('''
Ne
Ne 1 3.0
''')

psi4.set_options({
 'freeze_core': 'True'})

psi4.energy('ccsd(t)/cc-pvtz')
```

> python in.py

LANGUAGE INTERACTION IN PSI4

feeding and gross anatomy




```
molecule {
Ne
Ne 1 3.0
}

set freeze_core True

energy('ccsd(t)/cc-pvtz')
```

```
import psi4

psi4.geometry('''
Ne
Ne 1 3.0
''')

psi4.set_options({
 'freeze_core': 'True'})

psi4.energy('ccsd(t)/cc-pvtz')
```

```
{
 'molecule': {
  'symbols': ['Ne', 'Ne'],
  'geometry':
    [0, 0, 0, 5.67, 0, 0]
 },
 'driver': 'energy',
 'model': {
  'method': 'ccsd(t)',
  'basis': 'cc-pvtz'
 },
 'keywords':
  {'freeze_core': 'True'}
}
```

> psi4 in.txt

> python in.py

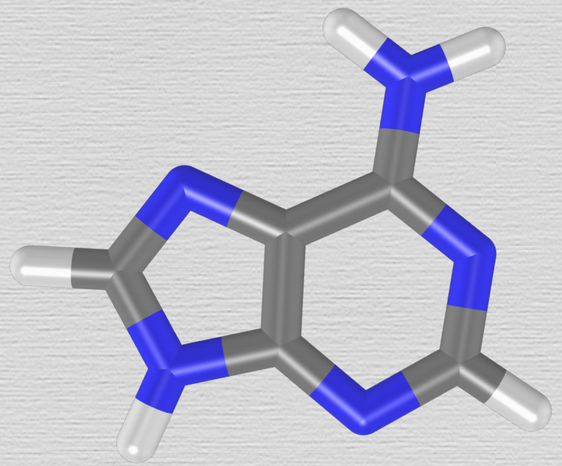
> psi4 --qcschema in.json

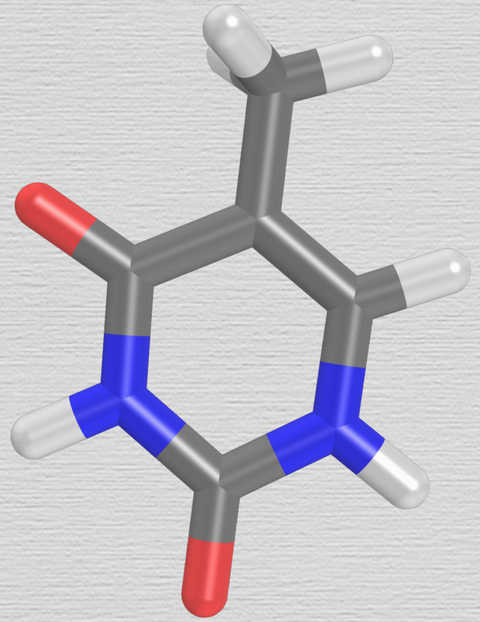
PROBING NON-COVALENT INTERACTIONS

a specialty of Psi4

PROBING NON-COVALENT INTERACTIONS

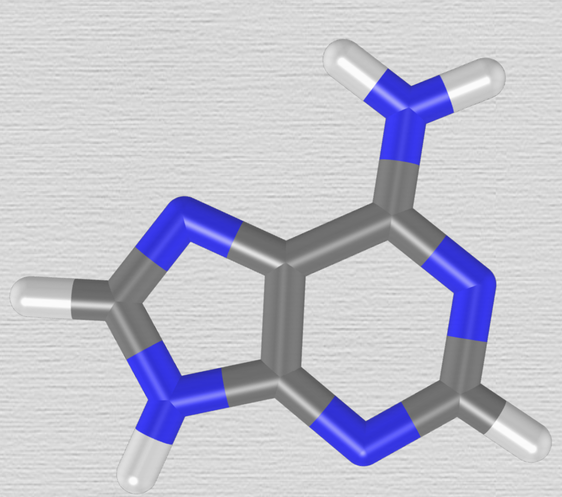
a specialty of Psi4

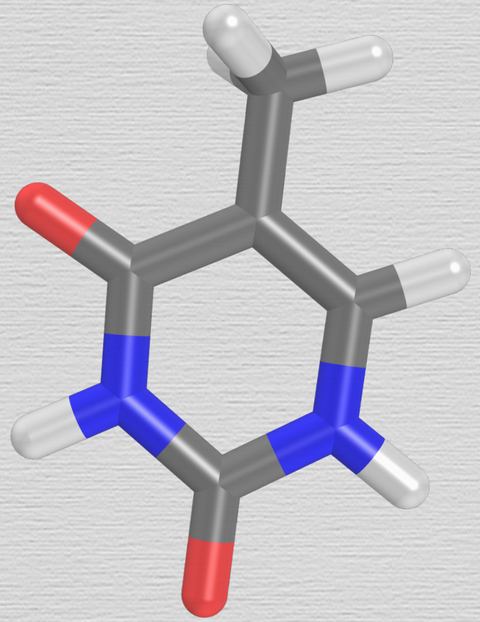




PROBING NON-COVALENT INTERACTIONS

a specialty of Psi4





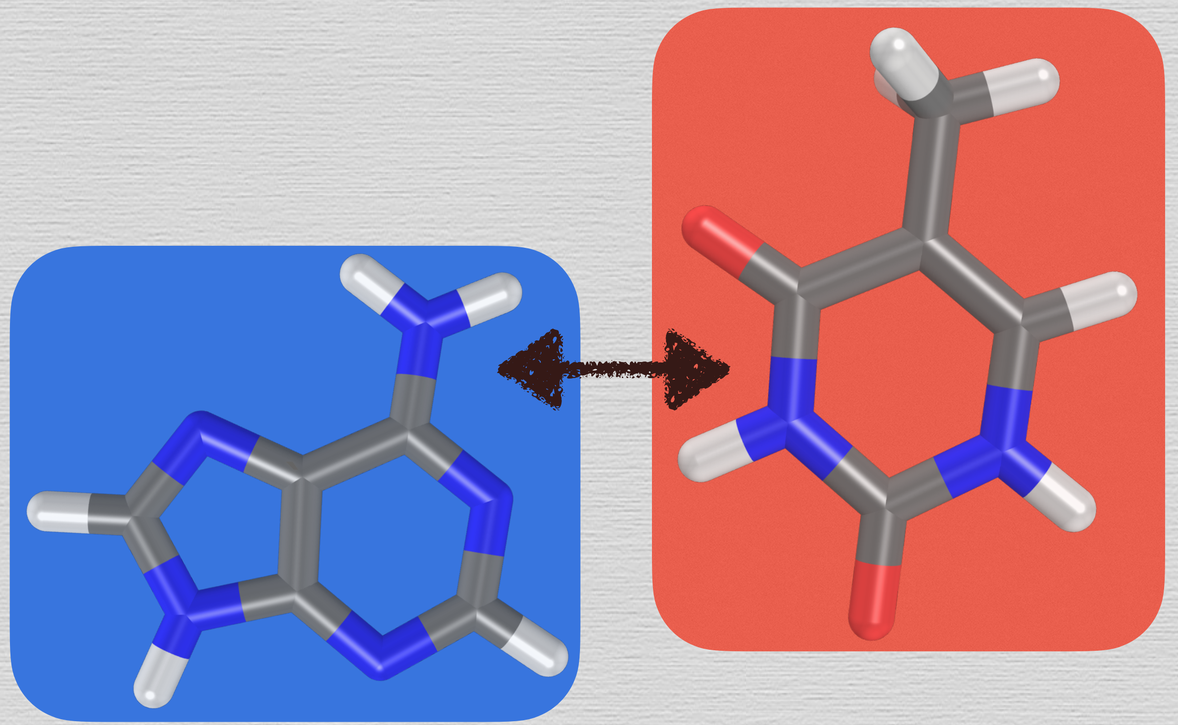
E_A

E_B

$$E_{AB} = E_A + E_B$$

PROBING NON-COVALENT INTERACTIONS

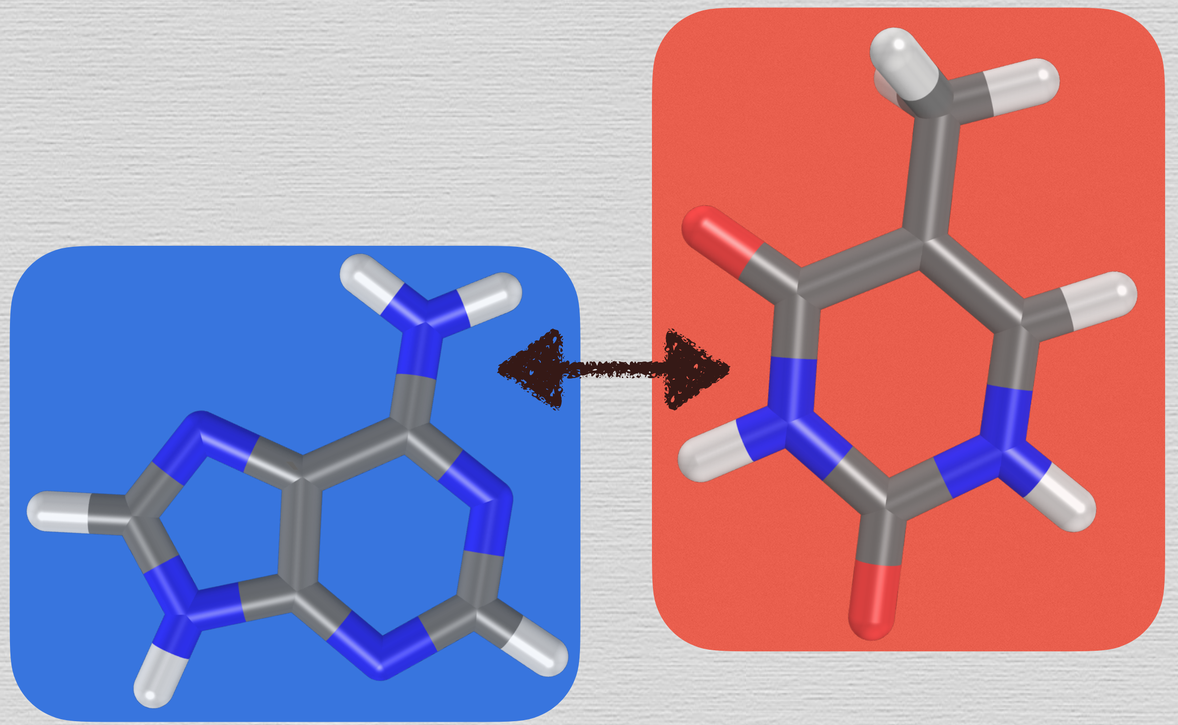
supermolecular approach



$$E_{AB} = E_A + E_B$$

PROBING NON-COVALENT INTERACTIONS

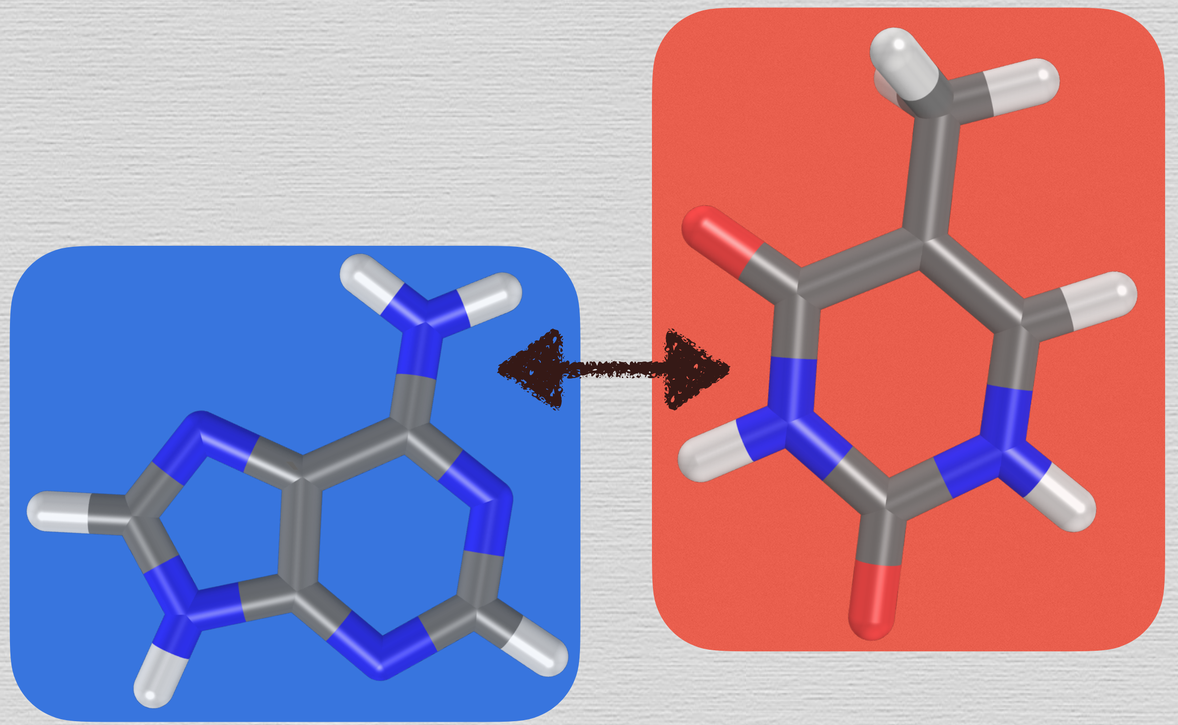
supermolecular approach



$$E_{AB} = E_A + E_B + \Delta E_{AB}$$

PROBING NON-COVALENT INTERACTIONS

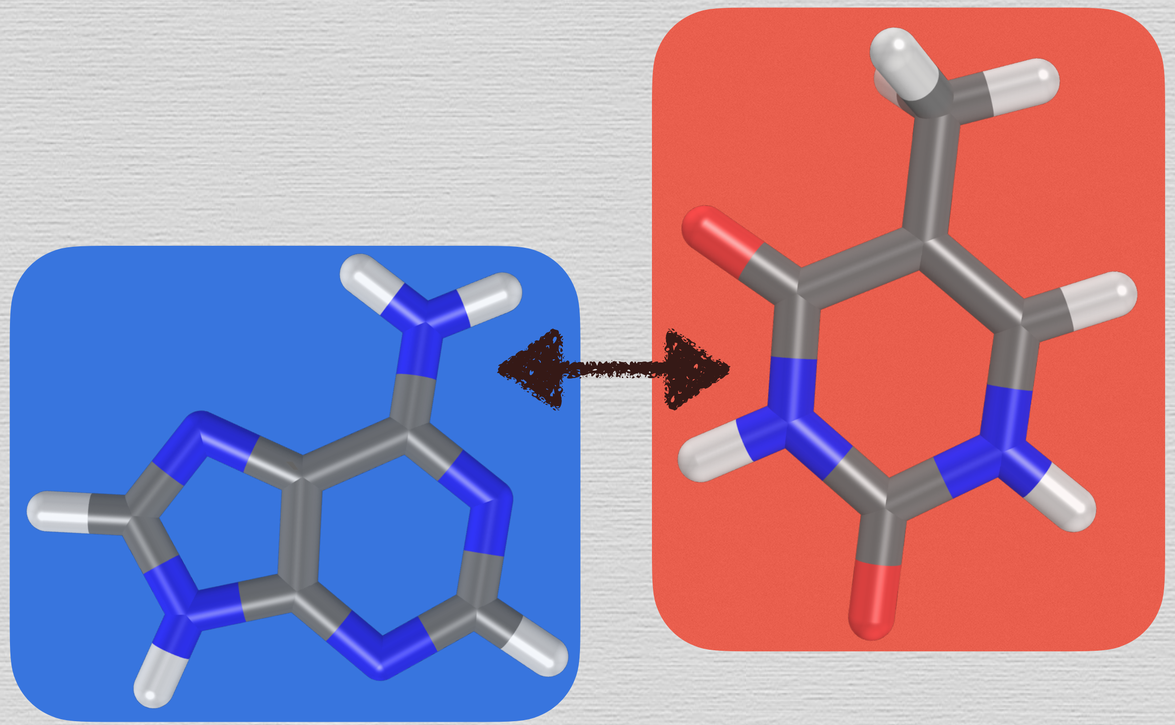
supermolecular approach



$$E_{AB} = E_A + E_B + IE_{AB}$$

PROBING NON-COVALENT INTERACTIONS

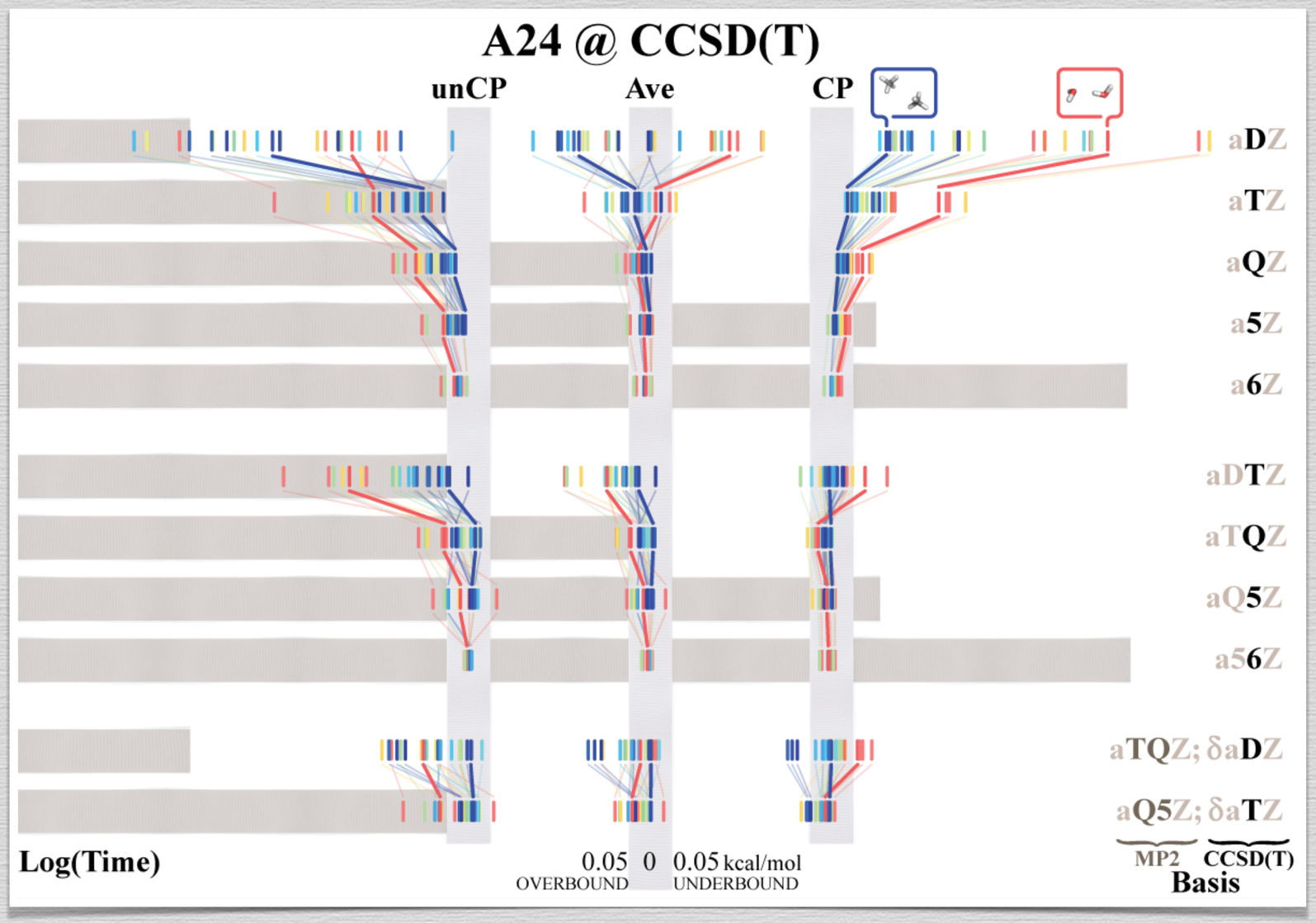
supermolecular approach



$$IE_{AB} = E_{AB} - E_A - E_B$$

COMPOSITE METHODS APPROXIMATIONS

complete basis set (CBS) approximations, delta corrections, focal point methods

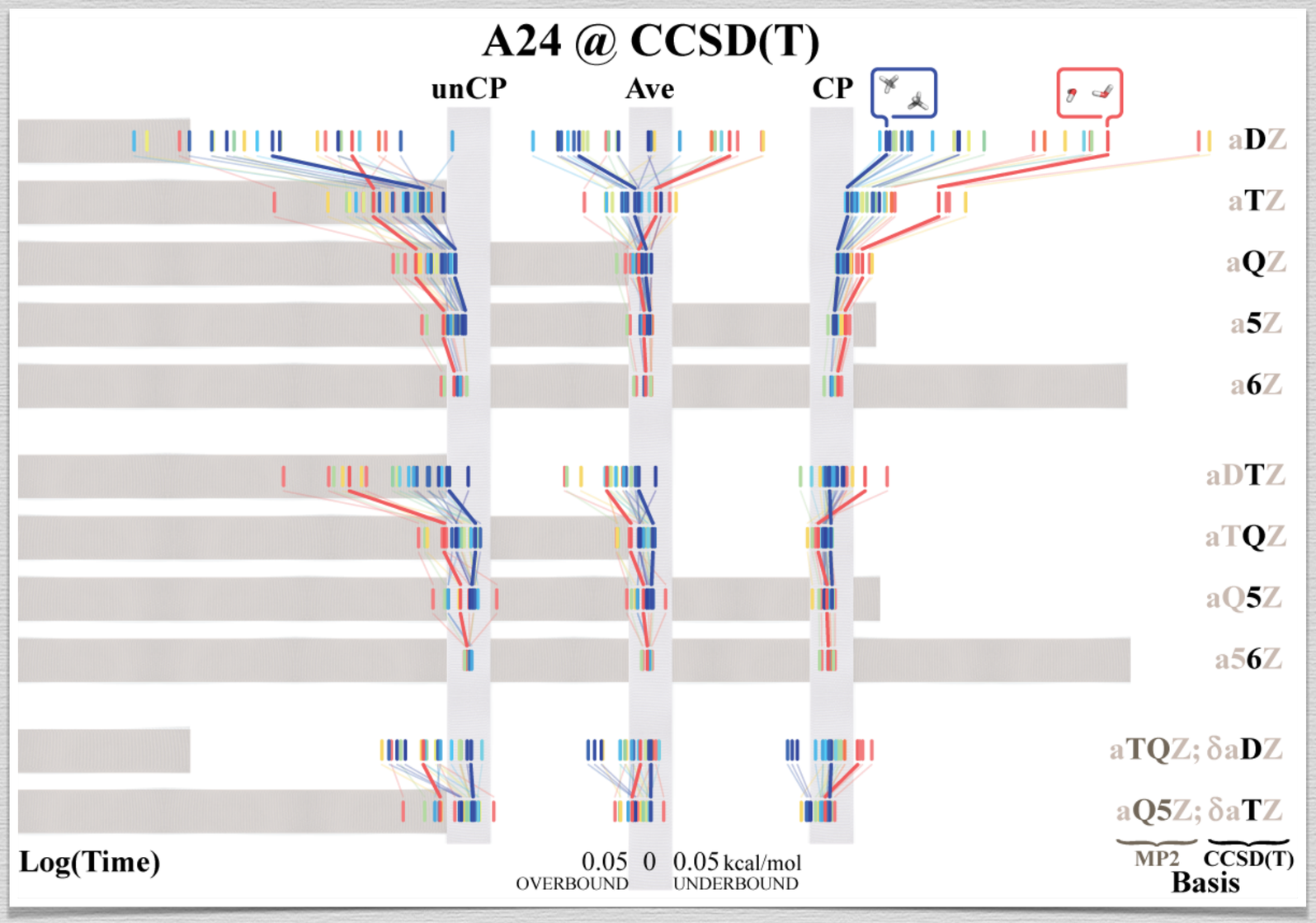



```
psi4.energy("ccsd(t)/cc-pvdz")
```

```
psi4.energy("ccsd(t)/cc-pvtz")
```

COMPOSITE METHODS APPROXIMATIONS

complete basis set (CBS) approximations, delta corrections, focal point methods




```
psi4.energy("ccsd(t)/cc-pvdz")
```

```
psi4.energy("ccsd(t)/cc-pvtz")
```

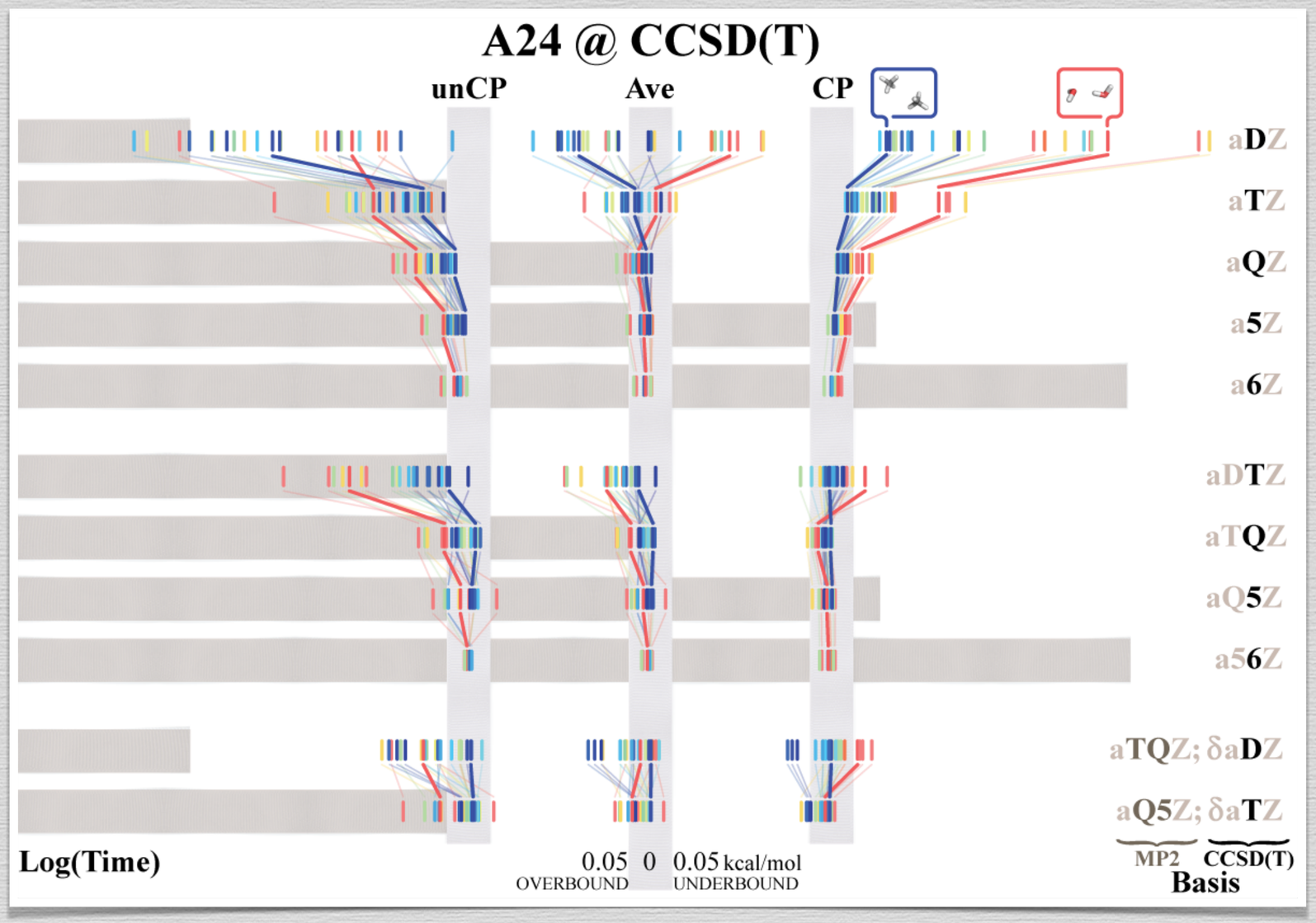
$$\mathrm{AB}\zeta: \quad E_{\mathrm{total}}^{\text{post-HF/TQ}\zeta} = E_{\mathrm{total}}^{\mathrm{HF/Q}\zeta} + \frac{\ell_{\max}^{\mathrm{pow}}}{\ell_{\max}^{\mathrm{pow}} - (\ell_{\max}-1)^{\mathrm{pow}}} E_{\mathrm{corr}}^{\text{post-HF/Q}\zeta} - \frac{(\ell_{\max}-1)^{\mathrm{pow}}}{\ell_{\max}^{\mathrm{pow}} - (\ell_{\max}-1)^{\mathrm{pow}}} E_{\mathrm{corr}}^{\text{post-HF/T}\zeta}. \qquad (7)$$

```
psi4.energy("ccsd(t)/cc-pv[dt]z")
```

```
psi4.energy("ccsd(t)/cc-pv[tq]z")
```

COMPOSITE METHODS APPROXIMATIONS

complete basis set (CBS) approximations, delta corrections, focal point methods




```
psi4.energy("ccsd(t)/cc-pvdz")
```

```
psi4.energy("ccsd(t)/cc-pvtz")
```

$$\mathrm{AB}\zeta: \quad E_{\mathrm{total}}^{\text{post-HF/TQ}\zeta} = E_{\mathrm{total}}^{\mathrm{HF/Q}\zeta} + \frac{\ell_{\max}^{\mathrm{pow}}}{\ell_{\max}^{\mathrm{pow}} - (\ell_{\max}-1)^{\mathrm{pow}}} E_{\mathrm{corr}}^{\text{post-HF/Q}\zeta} - \frac{(\ell_{\max}-1)^{\mathrm{pow}}}{\ell_{\max}^{\mathrm{pow}} - (\ell_{\max}-1)^{\mathrm{pow}}} E_{\mathrm{corr}}^{\text{post-HF/T}\zeta}. \qquad (7)$$

```
psi4.energy("ccsd(t)/cc-pv[dt]z")
```

```
psi4.energy("ccsd(t)/cc-pv[tq]z")
```

```
psi4.energy("mp2/cc-pv[tq]z +
             D:ccsd(t)/cc-pvdz")
```

$$[\mathrm{A}\zeta;\delta{:}\mathrm{B}\zeta]: \quad E_{\mathrm{total}}^{\text{post-MP2/[Q5}\zeta;\delta{:}\mathrm{T}\zeta]} = E_{\mathrm{total}}^{\mathrm{MP2/Q5}\zeta} + \delta_{\mathrm{MP2/T}\zeta}^{\text{post-MP2/T}\zeta}, \qquad (8)$$

$$= E_{\mathrm{total}}^{\mathrm{HF/5}\zeta} + E_{\mathrm{corr}}^{\mathrm{MP2/Q5}\zeta} + E_{\mathrm{corr}}^{\text{post-MP2/T}\zeta} - E_{\mathrm{corr}}^{\mathrm{MP2/T}\zeta}, \qquad (9)$$

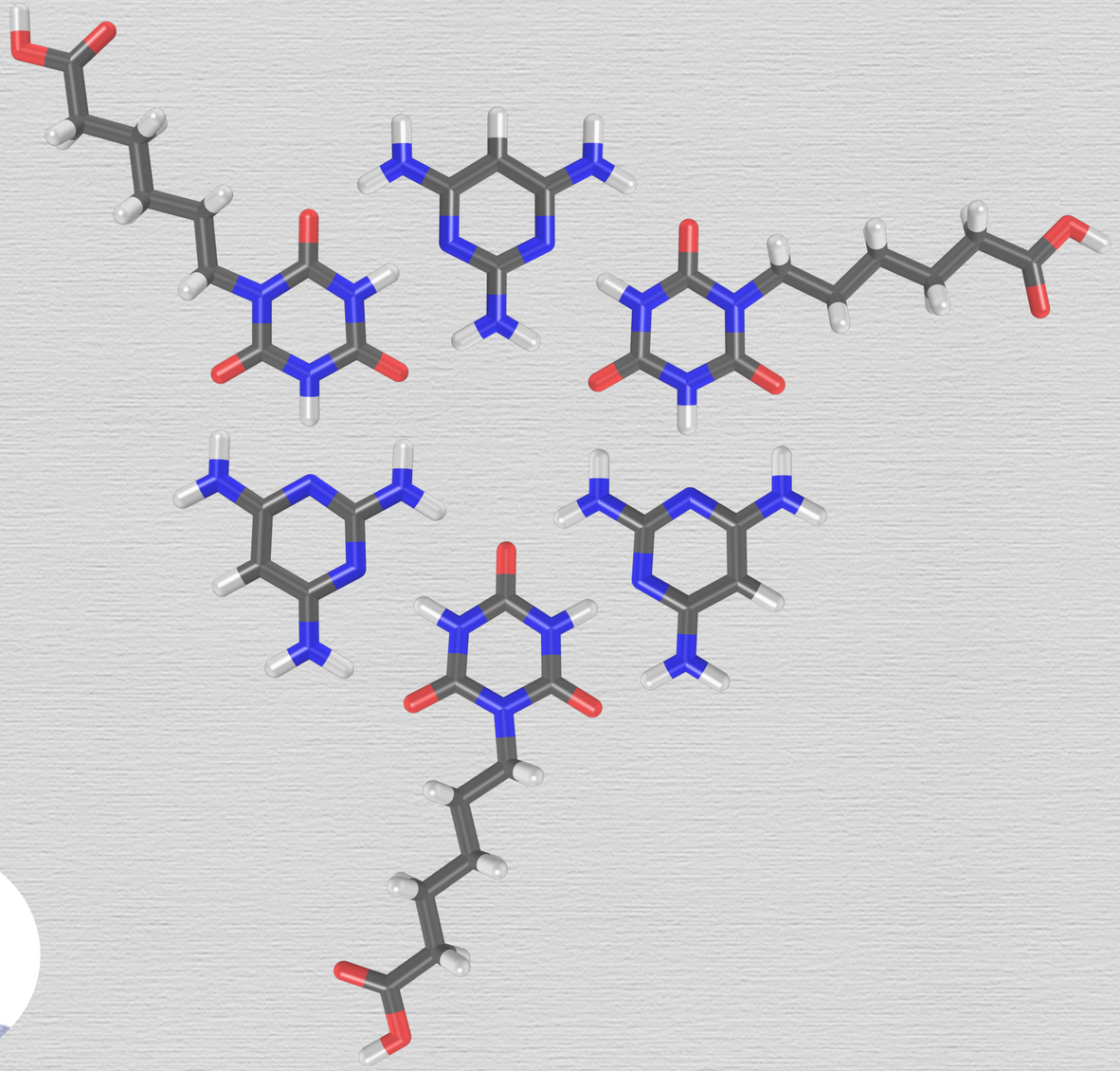
MANY-BODY EXPANSION (MBE)

and basis-set superposition error (BSSE) corrections

slide by Asem Alenaizan

MANY-BODY EXPANSION (MBE)

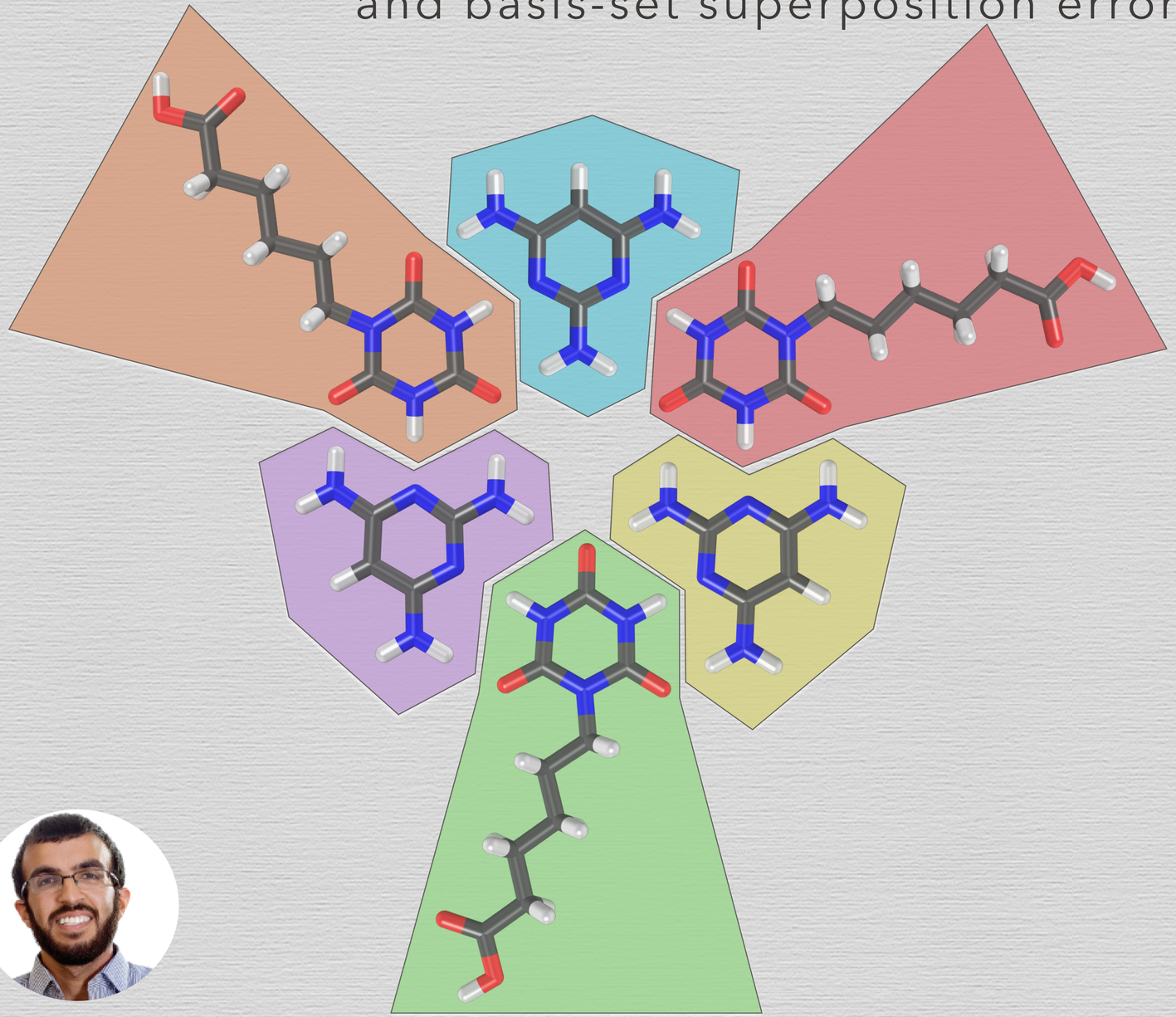
and basis-set superposition error (BSSE) corrections



slide by Asem Alenaizan

MANY-BODY EXPANSION (MBE)

and basis-set superposition error (BSSE) corrections

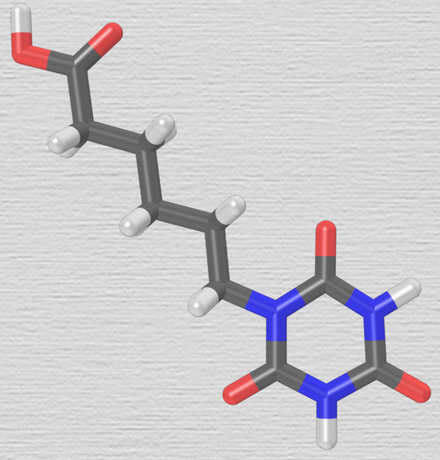


$$E_{\text{hex}} \approx \sum_{I} E_I$$

slide by Asem Alenaizan

MANY-BODY EXPANSION (MBE)

and basis-set superposition error (BSSE) corrections



$$E_{\text{hex}} \approx \sum_{I} E_I + \sum_{I>J} \Delta E_{IJ}^{(2)}$$

```
psi4.energy("mp2/cc-pvdz", bsse_type="cp", max_nbody=2)
```

slide by Asem Alenaizan

MANY-BODY EXPANSION (MBE)

and basis-set superposition error (BSSE) corrections

$$E_{\text{hex}} \approx \sum_I E_I + \sum_{I>J} \Delta E_{IJ}^{(2)} + \sum_{I>J>K} \Delta E_{IJK}^{(3)}$$

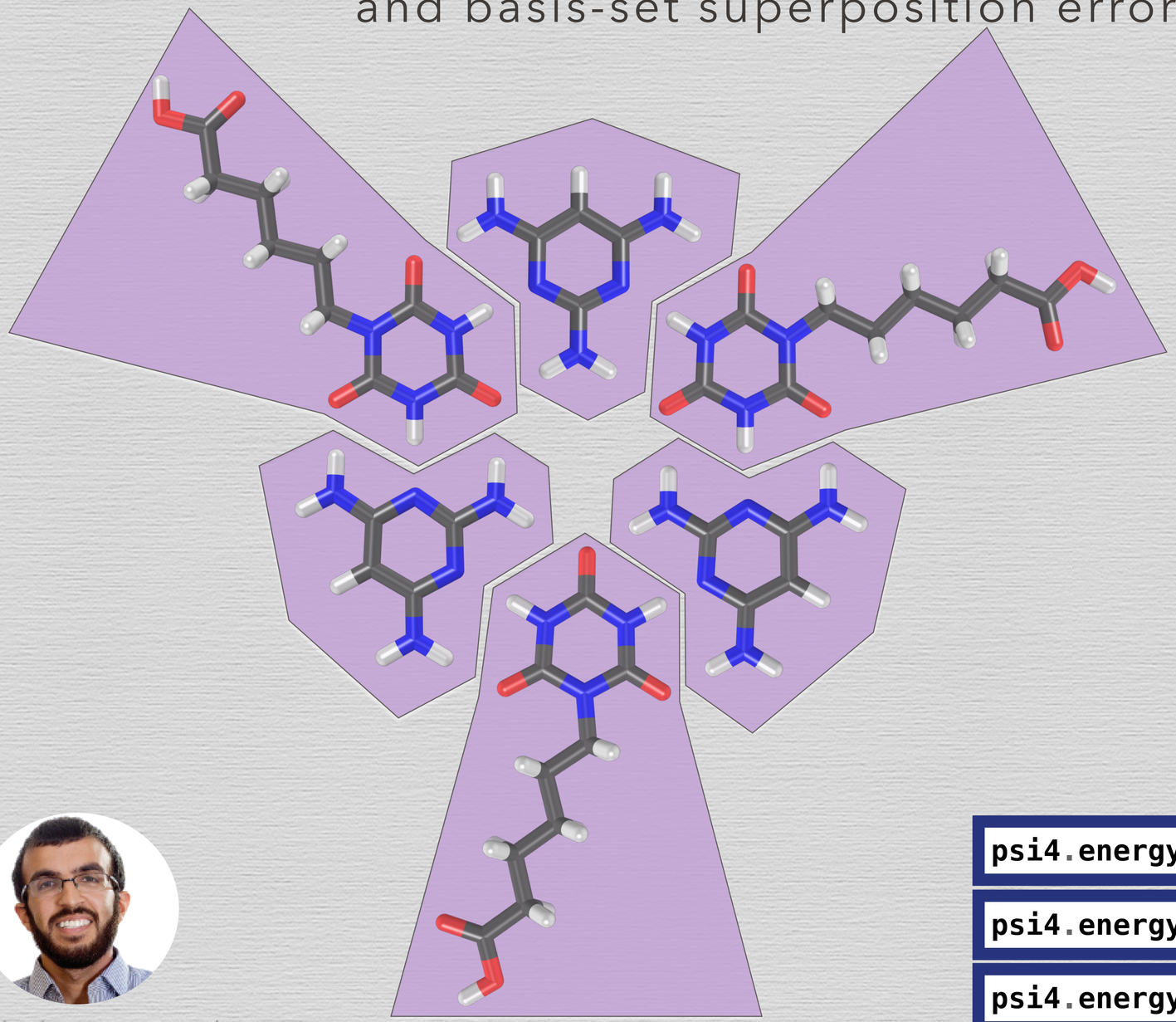
```
psi4.energy("mp2/cc-pvdz", bsse_type="cp", max_nbody=2)
```

```
psi4.energy("mp2/cc-pvdz", bsse_type="cp", max_nbody=3)
```

slide by Asem Alenaizan

MANY-BODY EXPANSION (MBE)

and basis-set superposition error (BSSE) corrections



$$E_{\text{hex}} \approx \sum_{I} E_I + \sum_{I>J} \Delta E_{IJ}^{(2)} + \sum_{I>J>K} \Delta E_{IJK}^{(3)} + \ldots$$

```
psi4.energy("mp2/cc-pvdz", bsse_type="cp", max_nbody=2)
psi4.energy("mp2/cc-pvdz", bsse_type="cp", max_nbody=3)
psi4.energy("mp2/cc-pvdz", bsse_type="cp")
```


slide by Asem Alenaizan

(1) Psi4 COMPOSITE & BSSE BY HAND

flexible but error-prone

```
psi4.energy("mp2/cc-pvtz", molecule= )
psi4.energy("mp2/cc-pvtz", molecule= )
psi4.energy("mp2/cc-pvtz", molecule= )
```

```
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
```

Local Compute

- **AVAILABLE** with Psi4 since 2010.
- **SPECIFICATION** through multiple files of slight variation. Procedure manual, so high risk of user error.
- **RATE-LIMITED** by single longest calc since can run parallel in queue.
- **RETRIEVAL** from filesystem if named systematically.
- **FLEXIBILITY** to mix QC programs to access more or more efficient methods since separate input files.

(1) Psi4 COMPOSITE & BSSE BY HAND

flexible but error-prone

```
psi4.energy("mp2/cc-pvtz", molecule= )
psi4.energy("mp2/cc-pvtz", molecule= )
psi4.energy("mp2/cc-pvtz", molecule= )
```

```
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
psi4.energy("ccsd(t)/cc-pvdz", molecule= )
```

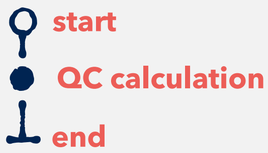
Local Compute

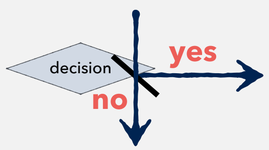
- **AVAILABLE** with Psi4 since 2010.
- **SPECIFICATION** through multiple files of slight variation. Procedure manual, so high risk of user error.
- **RATE-LIMITED** by single longest calc since can run parallel in queue.
- **RETRIEVAL** from filesystem if named systematically.
- **FLEXIBILITY** to mix QC programs to access more or more efficient methods since separate input files.

`def energy(method):` `def gradient(method):` `def hessian(method):`

PSI4 RECURSIVE DRIVER

capable but complicated





E · *G* *H*

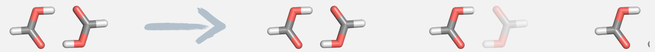
PSI4 RECURSIVE DRIVER

capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

`def manybody ():`

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



RUN `for frag in fragments:`

```
energy
gradient(method,frag)
hessian
```

Assemble n-body & interaction results from fragments.

E · G ⁞ H

PSI4 RECURSIVE DRIVER

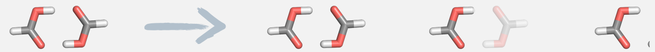
capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

E · G ⋮ H

`def manybody ():`

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



RUN `for frag in fragments:`

```
energy
gradient(method,frag)
hessian
```

Assemble n-body & interaction results from fragments.

`def findif ():`

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

RUN `for disp in displacements:`

```
energy
gradient(method,disp)
hessian
```

Assemble derivative results from displacements.

PSI4 RECURSIVE DRIVER

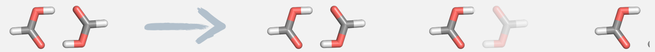
capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

E · G ⁞ H

`def manybody ():`

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



RUN `for frag in fragments:`

```
energy
gradient(method,frag)
hessian
```

Assemble n-body & interaction results from fragments.

`def findif ():`

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

RUN `for disp in displacements:`

```
energy
gradient(method,disp)
hessian
```

Assemble derivative results from displacements.

`def composite ():`

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

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

RUN `for mc in modelchems:`

```
energy
gradient(mc,molecule)
hessian
```

Assemble extrapolations & total results from modelchems.

PSI4 RECURSIVE DRIVER

capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

start
QC calculation
end

decision
yes
no

BSSE in **options**?

composite **method**?

analytic Hessian for **method**?

analytic gradient for **method**?

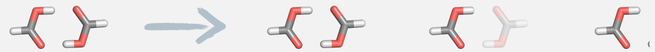
analytic energy for **method**?

`procedures[func][method](molecule)`

E · G ⋮ H

`def manybody ():`

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



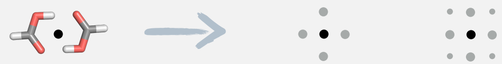
RUN `for frag in fragments:`

`energy gradient hessian (method,frag)`

Assemble n-body & interaction results from fragments.

`def findif ():`

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



RUN `for disp in displacements:`

`energy gradient hessian (method,disp)`

Assemble derivative results from displacements.

`def composite ():`

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

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

RUN `for mc in modelchems:`

`energy gradient hessian (mc,molecule)`

Assemble extrapolations & total results from modelchems.

PSI4 RECURSIVE DRIVER

capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

start
QC calculation
end

decision
yes
no

BSSE in **options**?

composite **method**?

analytic Hessian for **method**?

analytic gradient for **method**?

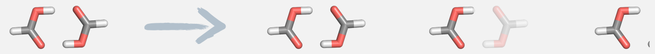
analytic energy for **method**?

`procedures[func][method](molecule)`

E · G ⋮⋮ H

`def manybody ():`

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



RUN `for frag in fragments:`

`energy gradient hessian (method, frag)`

Assemble n-body & interaction results from fragments.

`def findif ():`

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

RUN `for disp in displacements:`

`energy gradient hessian (method, disp)`

Assemble derivative results from displacements.

`def composite ():`

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

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

RUN `for mc in modelchems:`

`energy gradient hessian (mc, molecule)`

Assemble extrapolations & total results from modelchems.

PSI4 RECURSIVE DRIVER

capable but complicated

`def energy(method):` `def gradient(method):` `def hessian(method):`

start
QC calculation
end

decision
yes
no

BSSE in **options**?

composite **method**?

analytic Hessian for **method**?

analytic gradient for **method**?

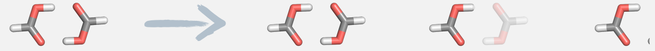
analytic energy for **method**?

```
procedures[func][method](molecule)
```

E · G ⋮ H

`def manybody ():`

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



RUN `for frag in fragments:`

```
energy
gradient(method,frag)
hessian
```

Assemble n-body & interaction results from fragments.

`def findif ():`

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

RUN `for disp in displacements:`

```
energy
gradient(method,disp)
hessian
```

Assemble derivative results from displacements.

`def composite ():`

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

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

RUN `for mc in modelchems:`

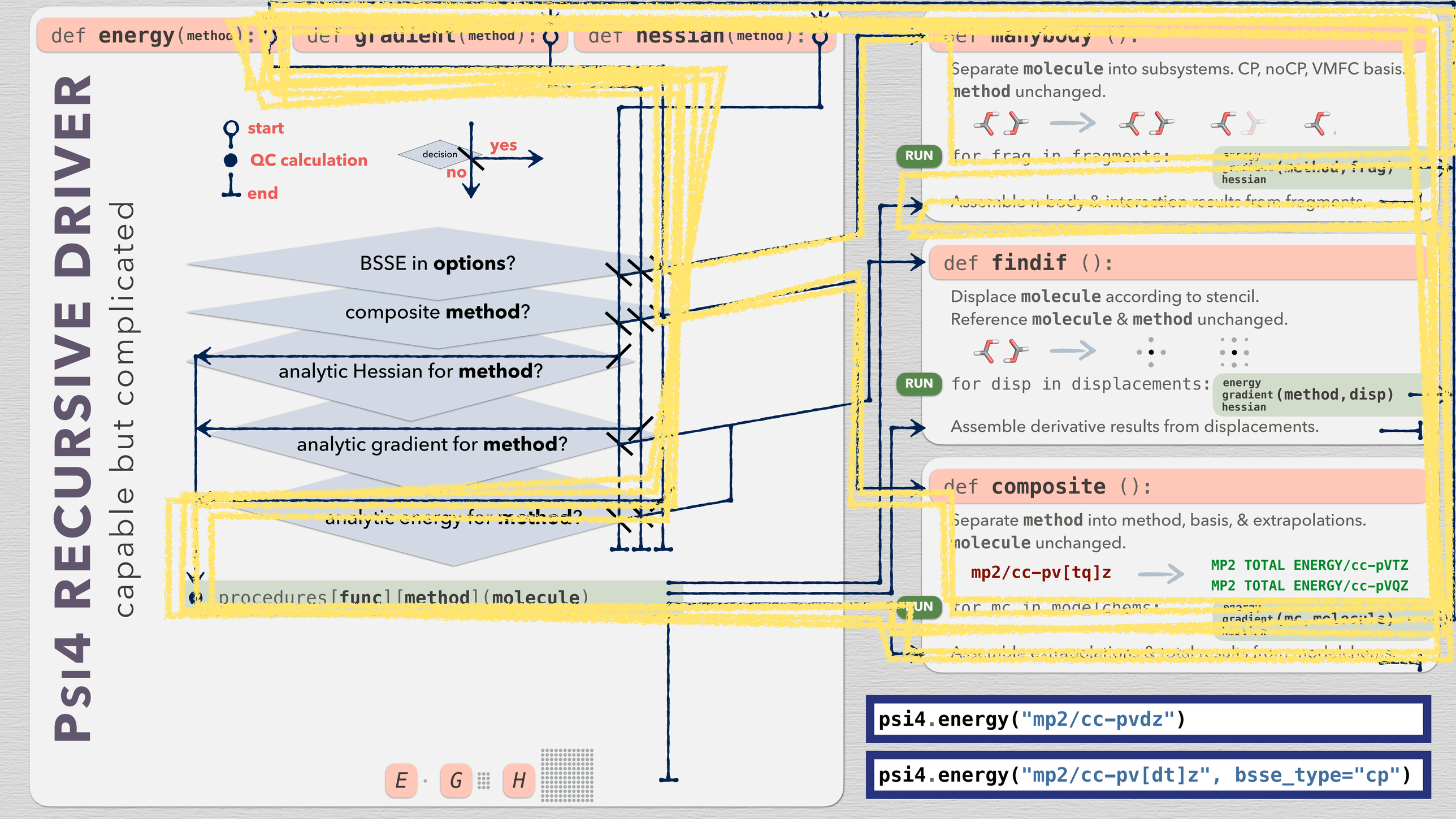
```
energy
gradient(mc,molecule)
hessian
```

Assemble extrapolations & total results from modelchems.

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

PSI4 RECURSIVE DRIVER

capable but complicated




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

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

E · G ⋮⋮ H

(2) Psi4 RECURSIVE DRIVER, INTERNAL

capable but complicated

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

Local Compute

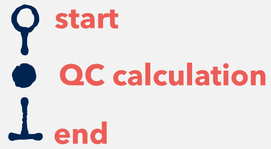
- **AVAILABLE** with Psi4 in current form since 2016.
- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by sum of all calcs since run sequentially.
- **RETRIEVAL** from filesystem difficult since many calcs in aggregated outfile.
- **FLEXIBILITY** limited to Psi4 only.

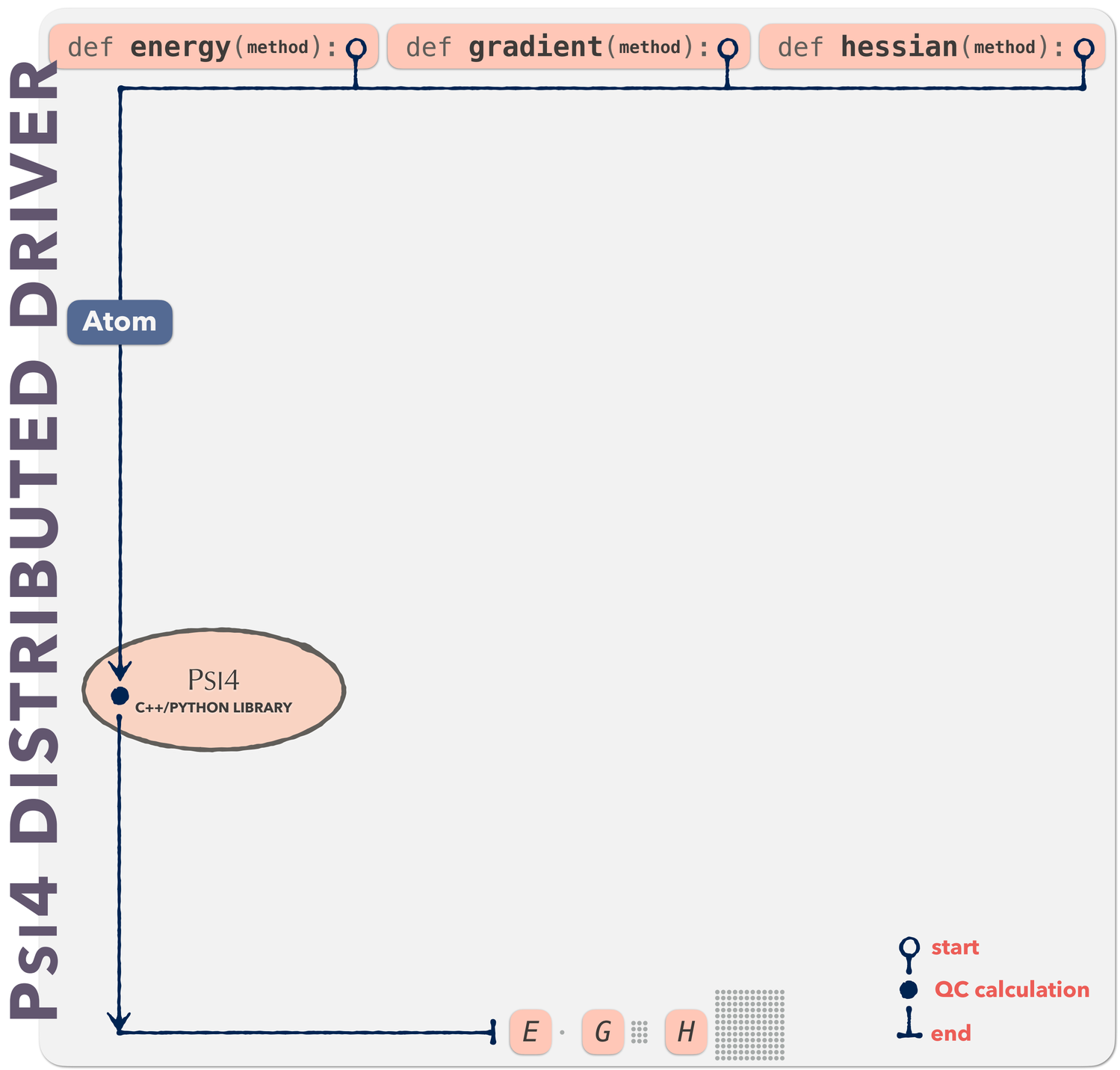
PSI4 DISTRIBUTED DRIVER

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

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

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




PSI4 DISTRIBUTED DRIVER
def energy(method):
def gradient(method):
def hessian(method):
Atom
PSI4
C++/PYTHON LIBRARY
E
G
H
start
QC calculation
end

PSI4 DISTRIBUTED DRIVER

`def energy(method):` `def gradient(method):` `def hessian(method):`

Atom

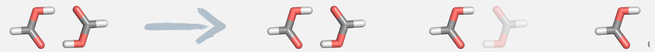
PSI4
C++/PYTHON LIBRARY

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

`def energy(method):` `def gradient(method):` `def hessian(method):`

Atom

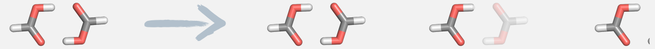
PSI4
C++/PYTHON LIBRARY

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.

`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

`def energy(method):` `def gradient(method):` `def hessian(method):`

Atom

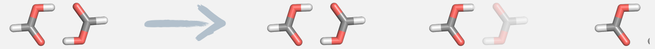
PSI4
C++/PYTHON LIBRARY

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.

`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

`def energy(method):` `def gradient(method):` `def hessian(method):`

Atom

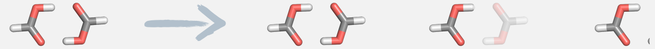
PSI4
C++/PYTHON LIBRARY

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.

`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

`def energy(method):` `def gradient(method):` `def hessian(method):`

MBE FD CBS Atom

POOL

I
QCSchema:
AtomicInput

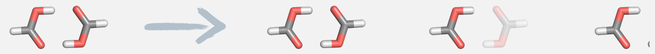
PSI4
C++/PYTHON LIBRARY

E · G ⋮ H

start
QC calculation
end

`class ManyBodyComputer ():`

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

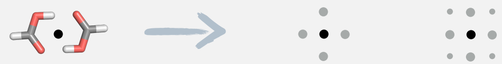


`for frag in fragments: return qcschema` QCSchema

ASM Assemble n-body & interaction results from fragments.

`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`

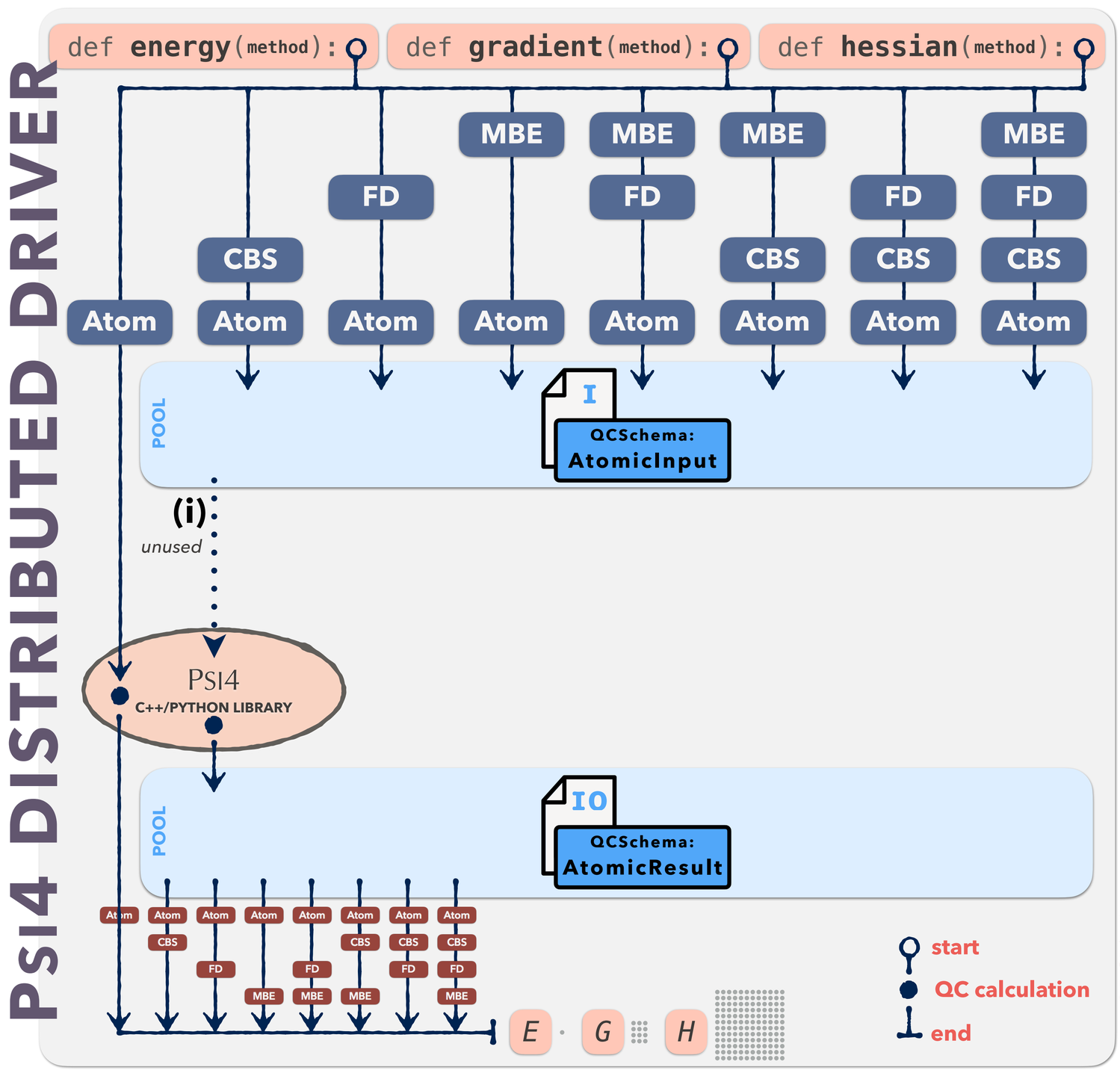
`for mc in modelchems: return qcschema` QCSchema

ASM Assemble extrapolations & total results from modelchems.

`class AtomicComputer ():`

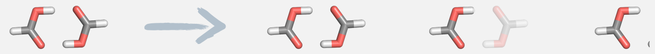
PLAN **`molecule`** & **`method`** unchanged. `return qcschema` QCSchema

ASM Return analytic energy, gradient, or Hessian.



class ManyBodyComputer ():

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

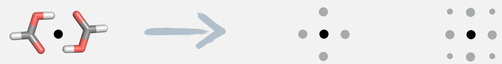


`for frag in fragments: return qcschema` QCSchema

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

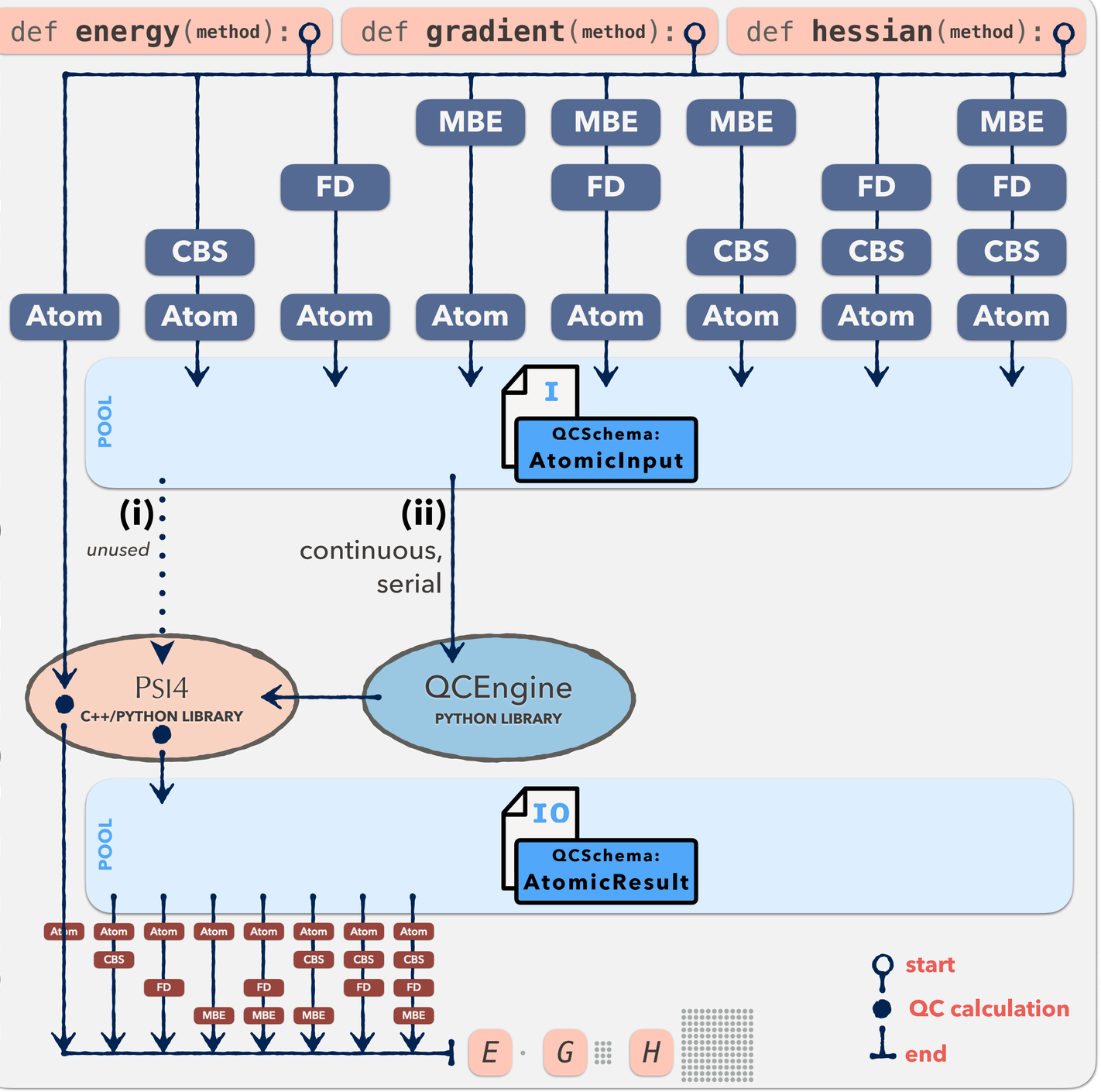
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

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

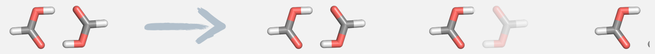
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

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



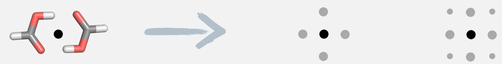
```
for frag in fragments: return qcschema
```

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

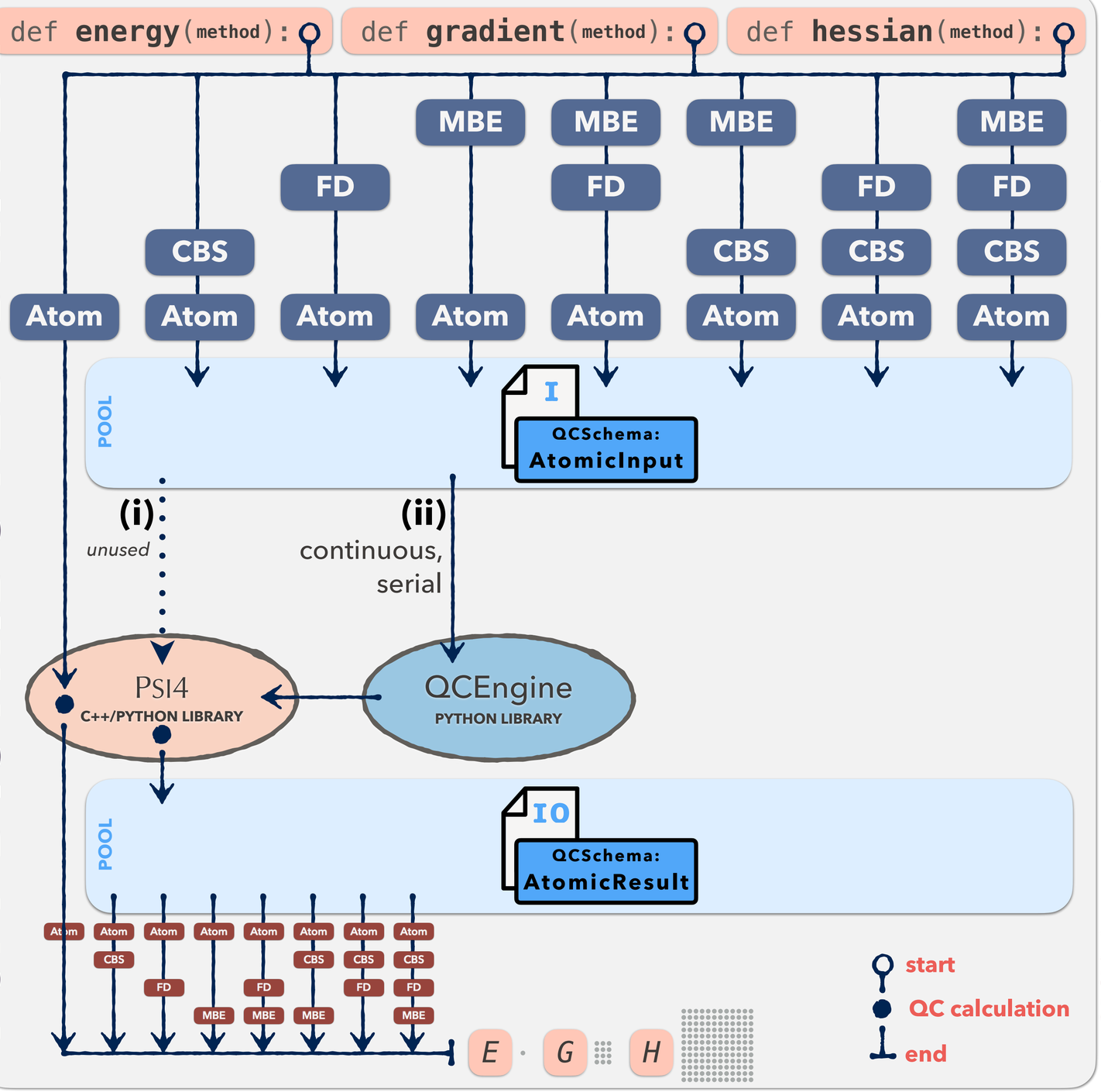
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

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

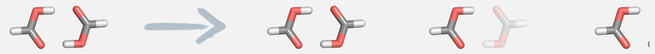
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

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



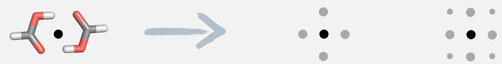
```
for frag in fragments: return qcschema
```

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

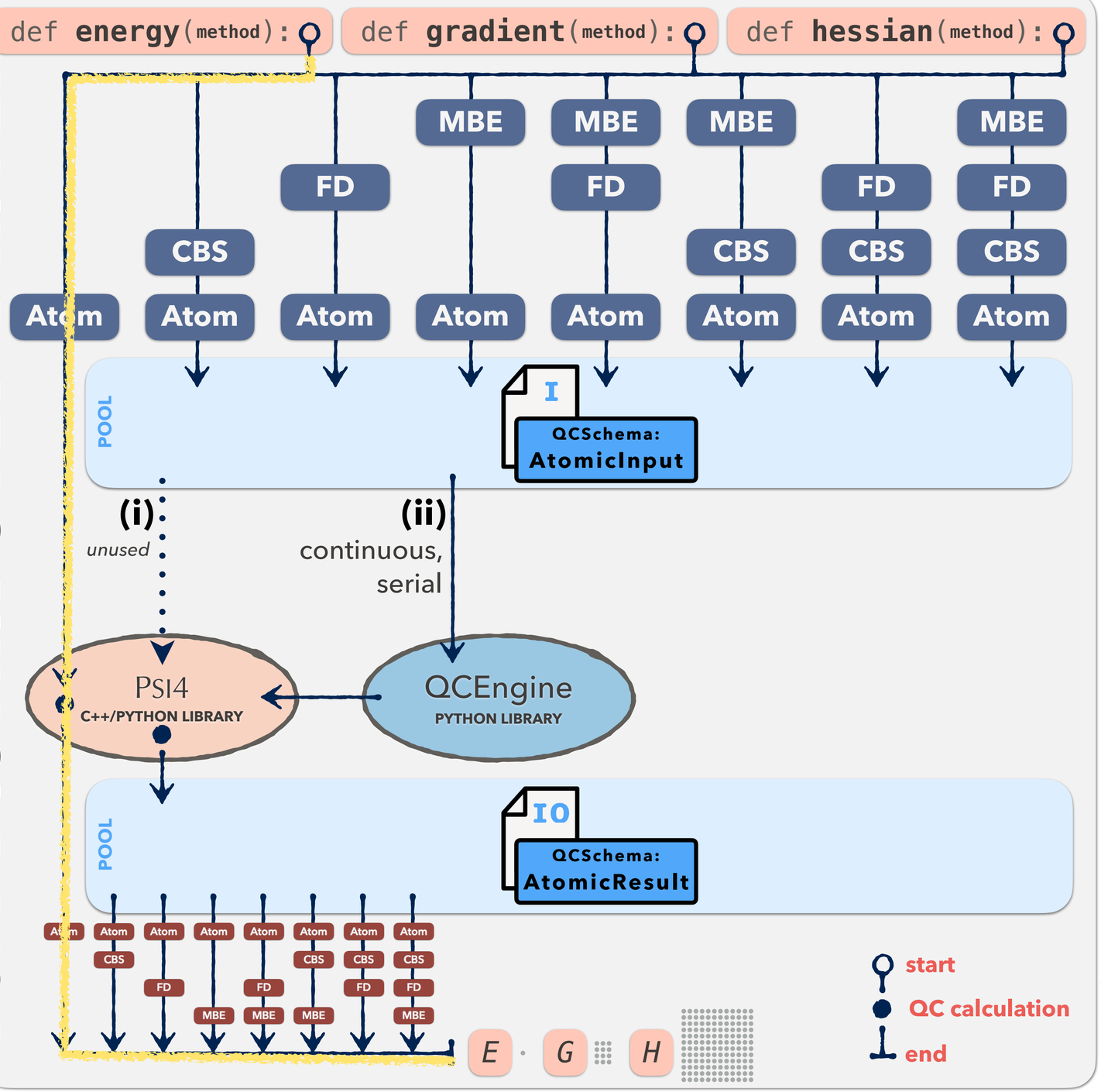
ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

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

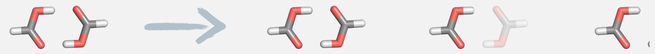
ASM Return analytic energy, gradient, or Hessian.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

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



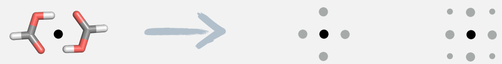
```
for frag in fragments: return qcschema
```

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
```

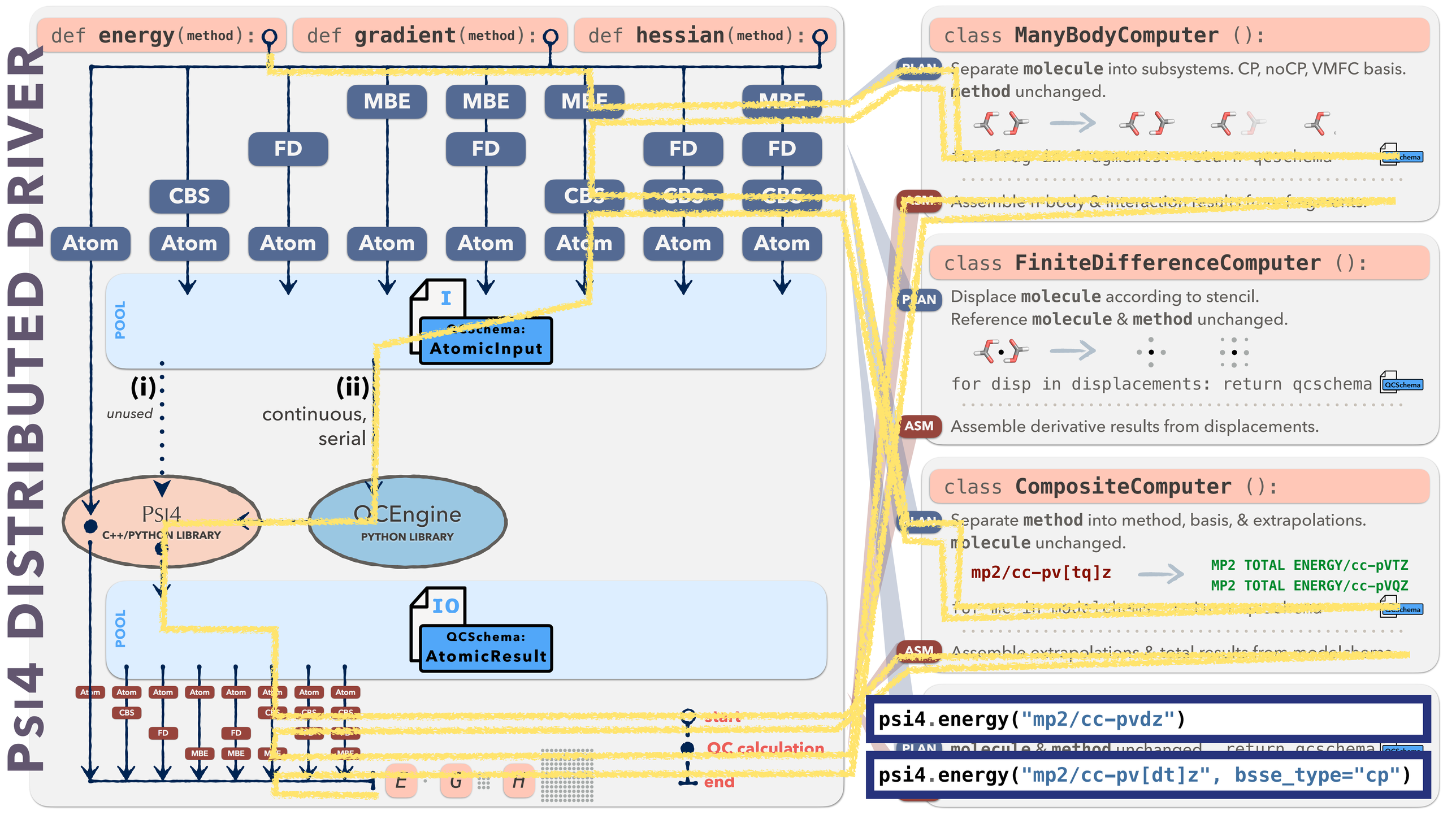
QCSchema

ASM Assemble extrapolations & total results from modelchems.

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

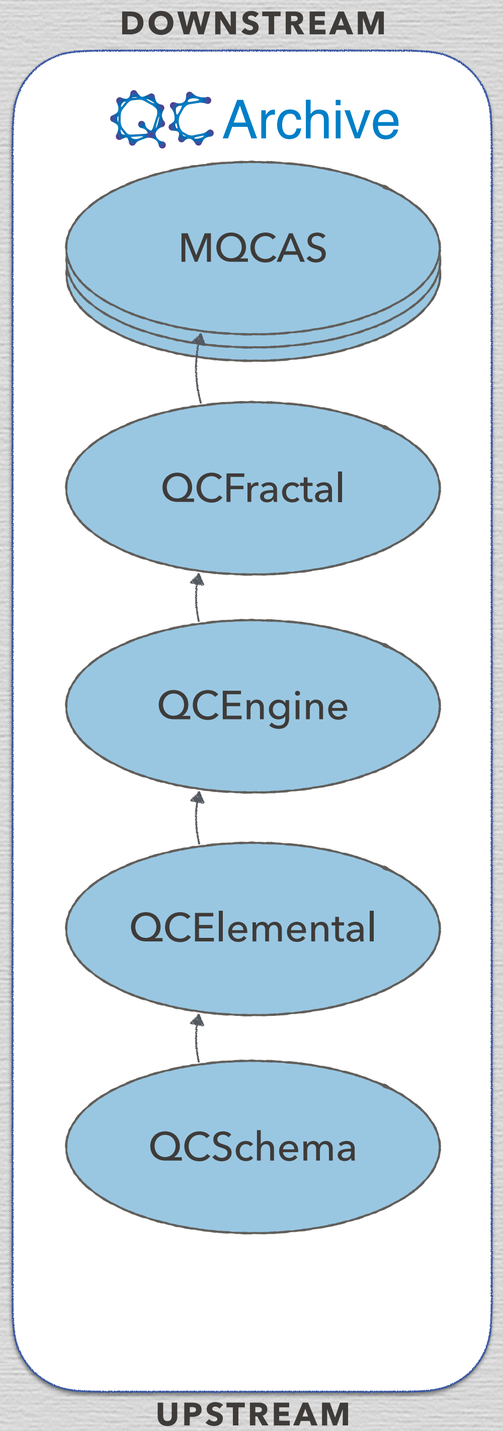
PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

ASM Return analytic energy, gradient, or Hessian.


PSI4 DISTRIBUTED DRIVER
def energy(method):
def gradient(method):
def hessian(method):
MBE
FD
CBS
Atom
POOL
I
QCSchema:
AtomicInput
(i)
unused
(ii)
continuous,
serial
PSI4
C++/PYTHON LIBRARY
QCEngine
PYTHON LIBRARY
IO
QCSchema:
AtomicResult
E
G
H
start
QC calculation
end
class ManyBodyComputer ():
PLAN
Separate molecule into subsystems. CP, noCP, VMFC basis.
method unchanged.
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
ASM
psi4.energy("mp2/cc-pvdz")
psi4.energy("mp2/cc-pv[dt]z", bsse_type="cp")

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

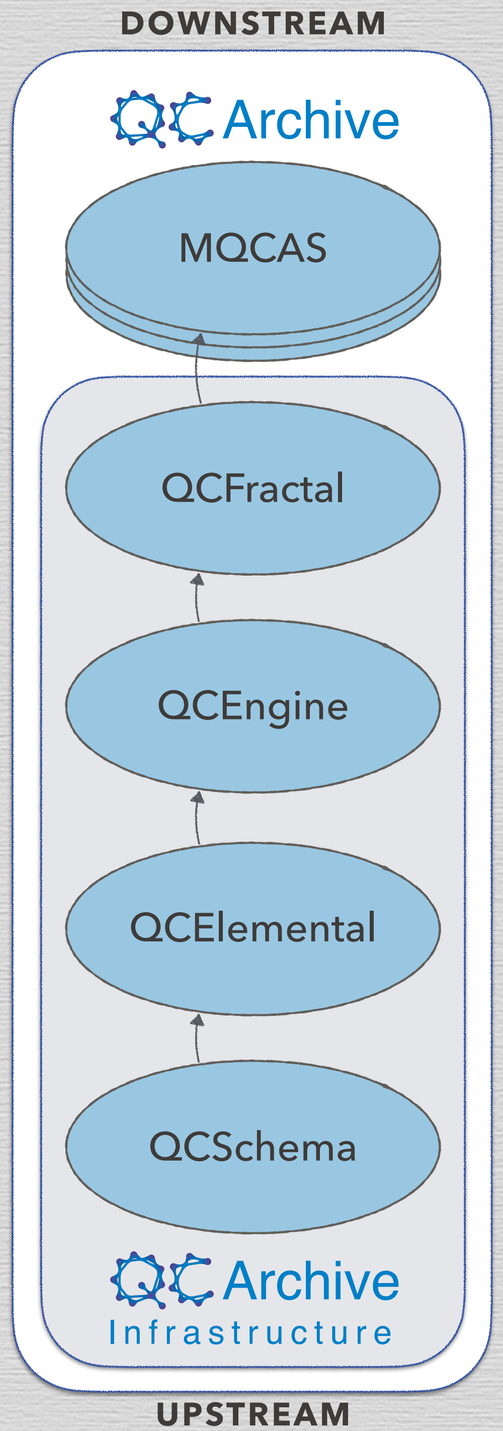
- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

- 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/



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

github.com/MolSSI/QCFractal

pip install qcfractal

conda install qcfractal -c conda-forge

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

github.com/MolSSI/QCEngine

pip install qcengine

conda install qcengine -c conda-forge

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

github.com/MolSSI/QCElemental

pip install elemental

conda install qcelemental -c conda-forge

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

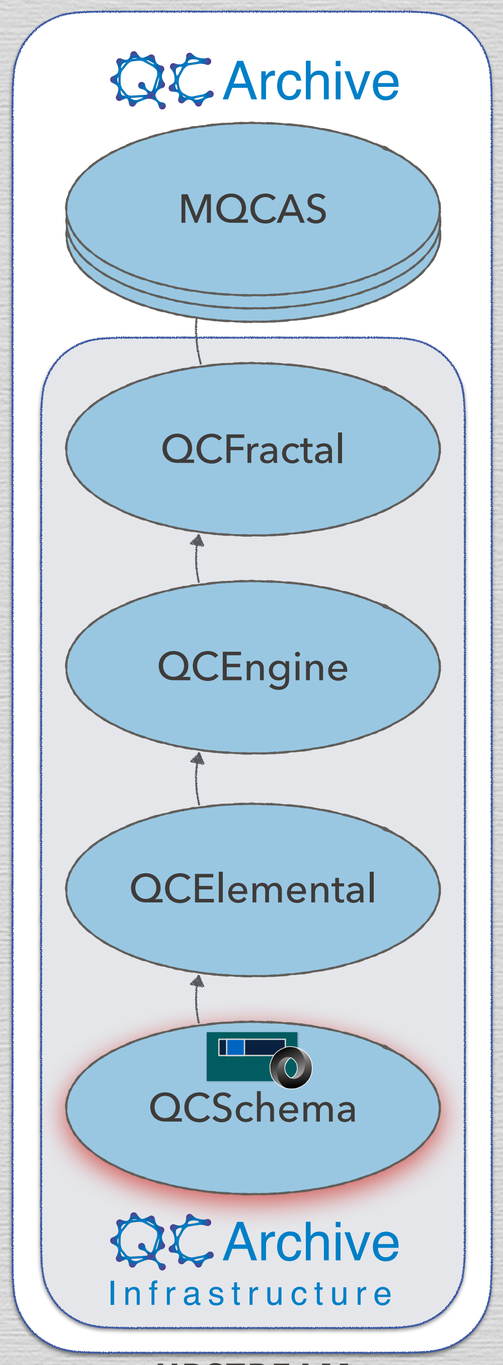
github.com/MolSSI/QCSchema

- 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")

github.com/MolSSI/QCFractal

pip install qcfractal

conda install qcfractal -c conda-forge

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

github.com/MolSSI/QCEngine

pip install qcengine

conda install qcengine -c conda-forge

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

github.com/MolSSI/QCElemental

pip install elemental

conda install qcelemental -c conda-forge

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

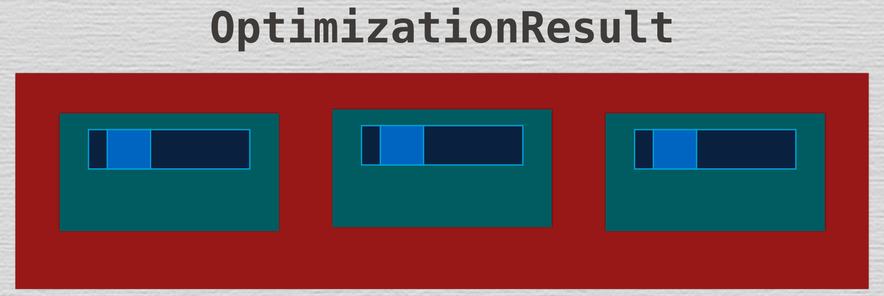
github.com/MolSSI/QCSchema

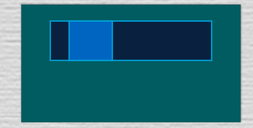
- QC data layout & descriptions
- e.g., molecule, DFT properties, grad. input, opt. output
- language agnostic

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** development stage since 2017. Most changes for QCA. More community participation desired but little momentum.
- **PRIOR ART** present but varied. Inviting all of the groups.
- **SCHEMA** defined in Python module but not used directly in QCA.






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

QUANTUM CHEMISTRY SCHEMA

primary schemas defined thus far

- **MOLECULE** Cartesian, atomic units (AU) based specification.
- **ATOMICINPUT** job directive for analytic single-point – energy, gradient, H
  or property – as defined by **DRIVER**.
- **ATOMICRESULT** append to **ATOMICINPUT** with output, simple scalar and ar
  results, and wavefunction results.
- **PROVENANCE** who wrote the data – program, version, module, compute
  information.
- **BASISSET** similar to BSE. allows **WAVEFUNCTION** data in CCA ordering.
- **OPTIMIZATIONINPUT & OPTIMIZATIONRESULT** job directive for abstract gen
  optimizer call generic QC program for gradient computation.

Molecule

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

```
O 0 0 0
H 2 0 0
--
@22Ne 5 0 0
units bohr
```

Snippet 1:

```
{"atom_labels": ["", "", ""],
 "atomic_numbers": [ 8,  1, 10],
 "fix_com": False,
 "fix_orientation": False,
 "fragment_charges": [0.0, 0.0],
 "fragment_multiplicities": [2, 1],
 "fragments": [[0, 1], [2]],
 "geometry": [[0., 0., 0.],
       [2., 0., 0.],
       [5., 0., 0.]],
 "mass_numbers": [16,  1, 22],
 "masses": [15.99491462,  1.00782503, 21.99138511],
 "molecular_charge": 0.0,
 "molecular_multiplicity": 2,
 "name": "HNeO",
 "provenance": {"creator": "QCElemental",
                "routine": "qcelemental.molparse.from_schema",
                "version": "v0.8.0"},
 "real": [ True,  True, False],
 "schema_name": "qcschema_molecule",
 "schema_version": 2,
 "symbols": ["O", "H", "Ne"],
 "validated": True}
```

Snippet 2: QCSCHEMA `Molecule` from Snippet 1.

QUANTUM CHEMISTRY SCHEMA

primary schemas defined thus far

- **MOLECULE** Cartesian, atomic units (AU) based specification.
- **ATOMICINPUT** job directive for analytic single-point – energy, gradient, Hessian, or property – as defined by **DRIVER**.
- **ATOMICRESULT** append to **ATOMICINPUT** with output, simple scalar and array results, and wavefunction results.
- **PROVENANCE** who wrote the data – program, version, module, compute information.
- **BASISSET** similar to BSE. allows **WAVEFUNCTION** data in CCA ordering.
- **OPTIMIZATIONINPUT & OPTIMIZATIONRESULT** job directive for abstract generic optimizer call generic QC program for gradient computation.

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": {},
}
```

QUANTUM CHEMISTRY SCHEMA

primary schemas defined thus far

- **MOLECULE** Cartesian, atomic units (AU) based specification.
- **ATOMICINPUT** job directive for analytic single-point – energy, gradient, Hessian, or property – as defined by **DRIVER**.
- **ATOMICRESULT** append to **ATOMICINPUT** with output, simple scalar and array results, and wavefunction results.
- **PROVENANCE** who wrote the data – program, version, module, compute information.
- **BASISSET** similar to BSE. allows **WAVEFUNCTION** data in CCA ordering.
- **OPTIMIZATIONINPUT & OPTIMIZATIONRESULT** job directive for abstract generic optimizer call generic QC program for gradient computation.

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": "..."
}
```

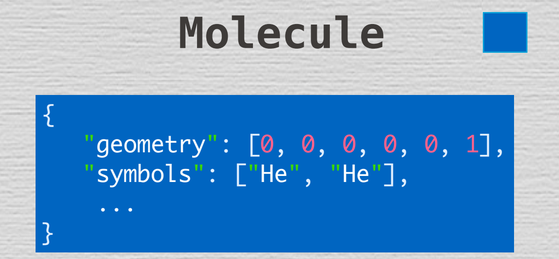
QUANTUM CHEMISTRY SCHEMA

primary schemas defined thus far

- **MOLECULE** Cartesian, atomic units (AU) based specification.
- **ATOMICINPUT** job directive for analytic single-point – energy, gradient, Hessian, or property – as defined by **DRIVER**.
- **ATOMICRESULT** append to **ATOMICINPUT** with output, simple scalar and array results, and wavefunction results.
- **PROVENANCE** who wrote the data – program, version, module, compute information.
- **BASISSET** similar to BSE. allows **WAVEFUNCTION** data in CCA ordering.
- **OPTIMIZATIONINPUT & OPTIMIZATIONRESULT** job directive for abstract generic optimizer call generic QC program for gradient computation.

- orbitals_a
- orbitals_b
- density_a
- density_b
- fock_a
- fock_b
- eigenvalues_a
- eigenvalues_b
- occupations_a
- occupations_b

Molecule

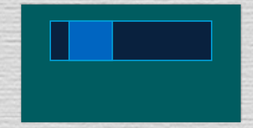


```
{
   "geometry": [0, 0, 0, 0, 0, 1],
   "symbols": ["He", "He"],
    ...
}
```

AtomicInput

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

AtomicResult



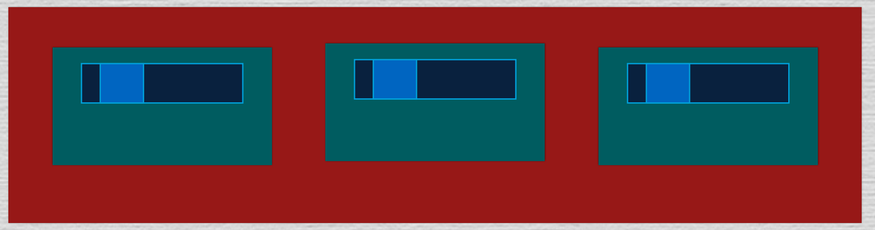
```
{
    ... AtomicInput ...
    "provenance": {
        "creator": "My QM Program",
        "version": "1.1rc1",
    },
    "properties": {
        "calcinfo_nalpha": 5,
        "scf_total_energy": -5.433191881443323,
        "nuclear_repulsion_energy": 2.11670883436,
        "scf_iterations": 8.0,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

QUANTUM CHEMISTRY SCHEMA

primary schemas defined thus far

- **MOLECULE** Cartesian, atomic units (AU) based specification.
- **ATOMICINPUT** job directive for analytic single-point – energy, gradient, Hessian, or property – as defined by **DRIVER**.
- **ATOMICRESULT** append to **ATOMICINPUT** with output, simple scalar and array results, and wavefunction results.
- **PROVENANCE** who wrote the data – program, version, module, compute information.
- **BASISSET** similar to BSE. allows **WAVEFUNCTION** data in CCA ordering.
- **OPTIMIZATIONINPUT & OPTIMIZATIONRESULT** job directive for abstract generic optimizer call generic QC program for gradient computation.

OptimizationResult



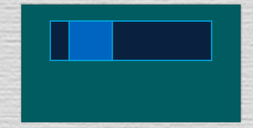
Molecule

```
{
   "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 DEFINES DATA LAYOUTS

and text advice, nothing more

- **LAYOUT** JSON Schema defines a data layout and some minimal type and count checking.
- **CONVENTIONS** JSON Schema can't check conventions you rely upon like AU units, CCA ordering, center-of-mass positioning

DOMAIN-SPECIFIC ASCII

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

GAMESS Input

DATA LAYOUT TRANSLATION

QCSchema: AtomicInput

```
{
  'molecule': {
    'symbols': [              ],
    'geometry': [                 ],
    'molecular_multiplicity':  },
  'driver': '       ',
  'model': {
    'method': '     ',
    'basis': '     '},
  'keywords': {


                      },
}
```

QCSchema DEFINES DATA LAYOUTS

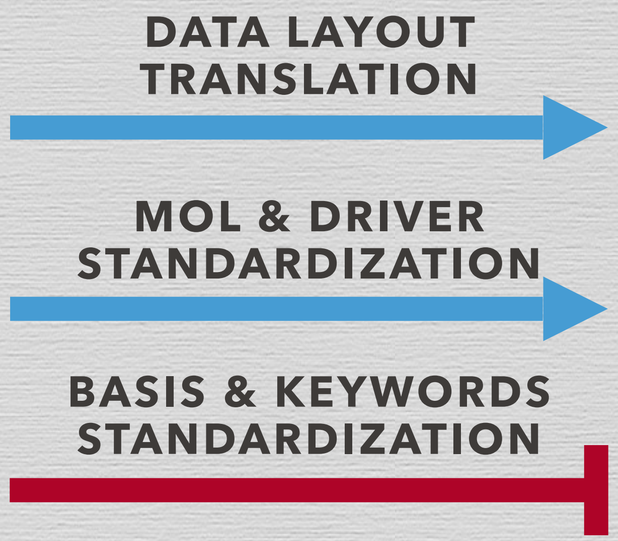
and text advice, nothing more

- **LAYOUT** JSON Schema defines a data layout and some minimal type and count checking.
- **CONVENTIONS** JSON Schema can't check conventions you rely upon like AU units, CCA ordering, center-of-mass positioning

DOMAIN-SPECIFIC ASCII

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

GAMESS Input



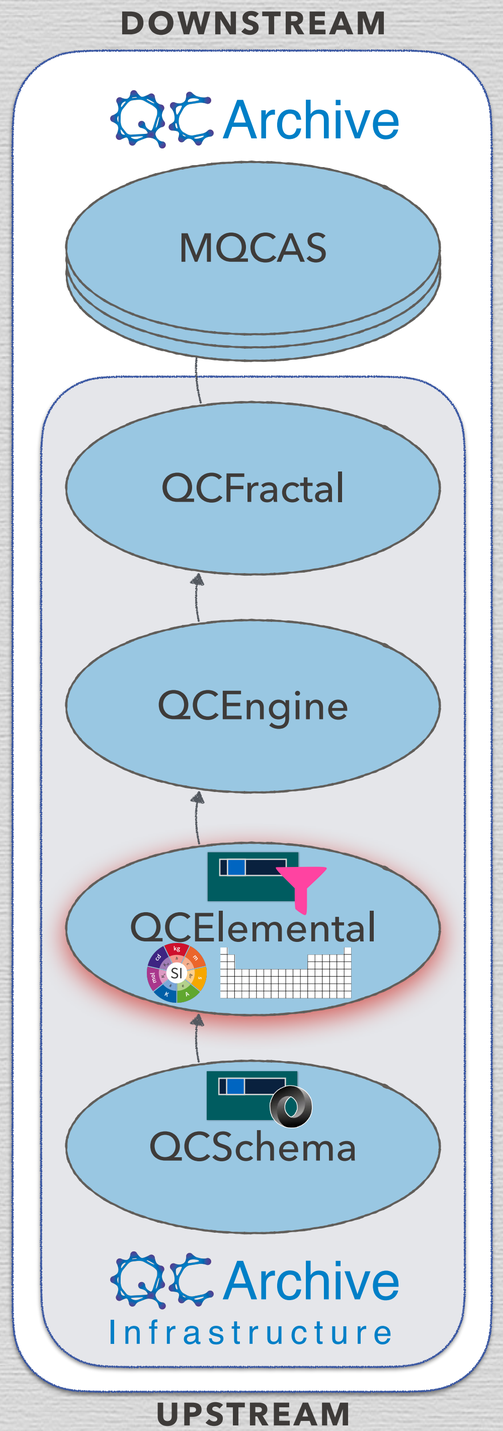
QCSchema: AtomicInput

```
{
  'molecule': {
    'symbols': ['N', 'H', 'H'],
    'geometry': [0.000, ..., 1.014],
    'molecular_multiplicity': 2},
  'driver': 'energy',
  'model': {
    'method': 'ccsd',
    'basis': 'accd'},
  'keywords': {
    'contrl__ispher': 1,
    'contrl__scftyp': 'rohf',
    'ccinp__ncore': 0},
}
```

GAMESS QCEngine Input

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 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

QCElemental provides datasets

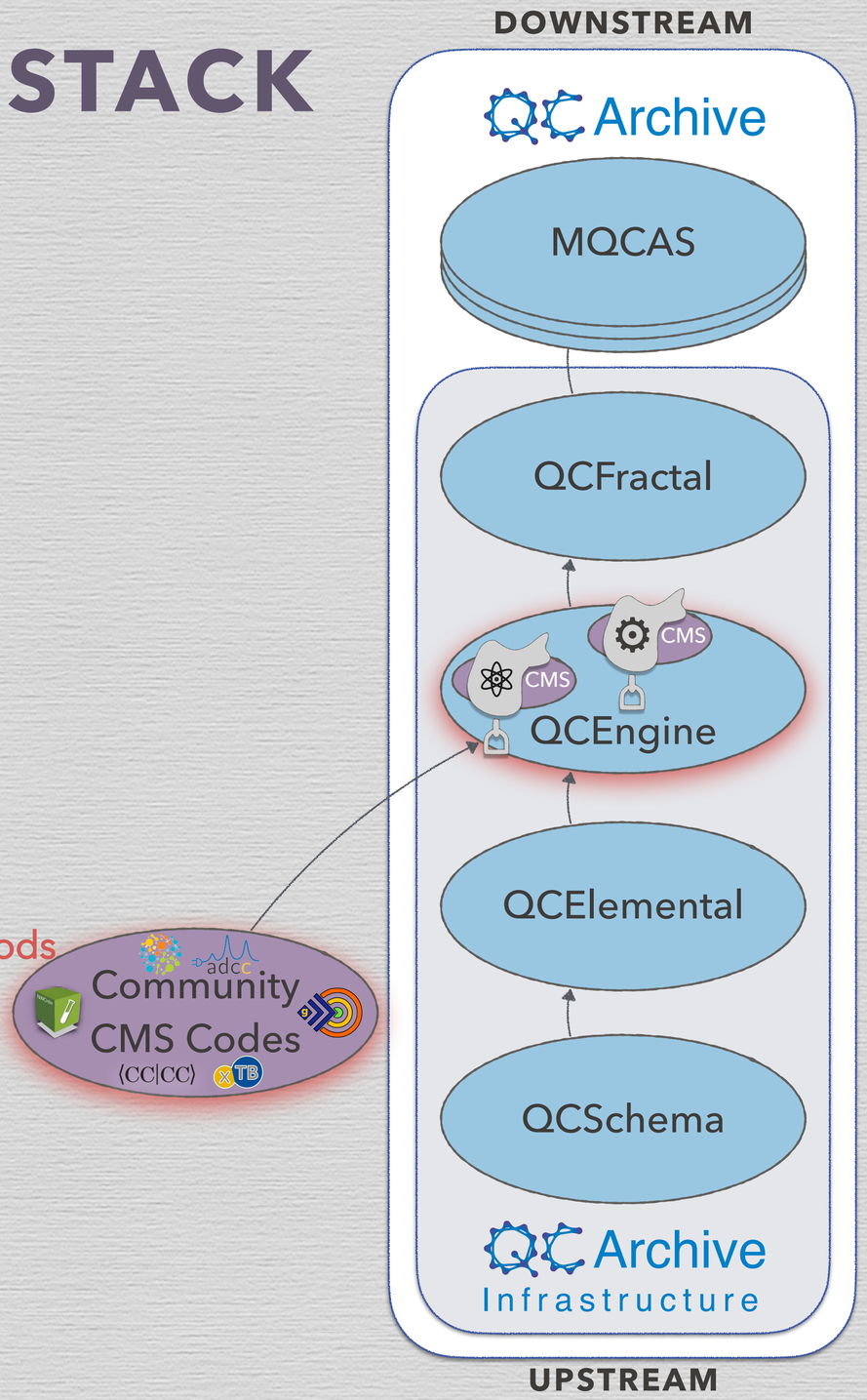
NIST CODATA, covalent and vdW radii, Periodic Table

- **STRUCTURED DATA** processed into light Python API directly from NIST structured data, if available.
- **PRECISION** original significant figures value always available
- **VERSIONED** store& upgrade: when reproducibility demands old versions, accessing both helps transitions.
- **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
```

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

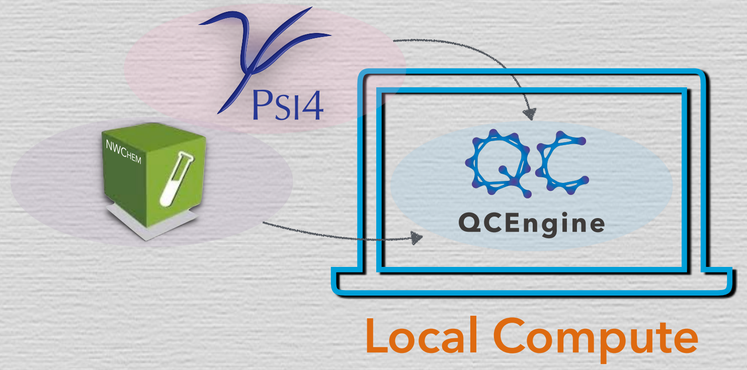
- Python QCSchema implementations & validation
- NIST periodic table & physical constants
- molecule parsing, validation, export

- 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
```

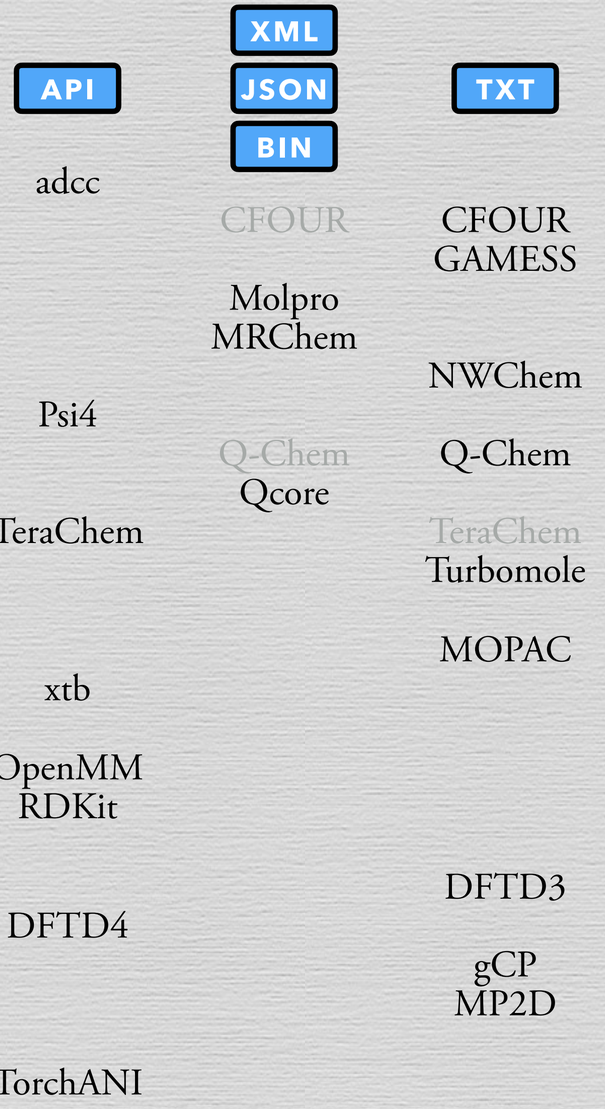
PYAPI

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

QCENGINE: AVAILABLE PROGRAM HARNESSES

github.com/MolSSI/QCEngine

CMS Program	QCEngine E	G	H	Prop.	Wfn
Quantum Chemistry					
adcc	✓	✗	✗	✓	✗
CFOUR	✓	✓	✓	✓	✗
GAMESS	✓	✓	✗	✓	✗
Molpro	✓	✓	✗	✓	✗
MRChem	✓	✗	✗	✓	✗
NWChem	✓	✓	✓	✓	✗
Psi4	✓	✓	✓	✓	✓
Q-Chem	✓	✓	✓	✗	✗
Qcore	✓	✓	✓	✗	✓
TeraChem	✓	✓	✗	✓	✗
Turbomole	✓	✓	✓	✗	✗
Semi-Empirical					
MOPAC	✓	✓	✗	✓	✗
xtb	✓	✓	✗	✓	✗
Molecular Mechanics					
OpenMM	✓	✓	✗	✓	
RDKit	✓	✓	✗	✓	
Analytical Corrections					
DFTD3	✓	✓	✗	✗	
DFTD4	✓	✓	✗	✗	
gCP	✓	✓	✗	✗	
MP2D	✓	✓	✗	✗	
Machine Learning Inference					
TorchANI	✓	✓	✓	✓	



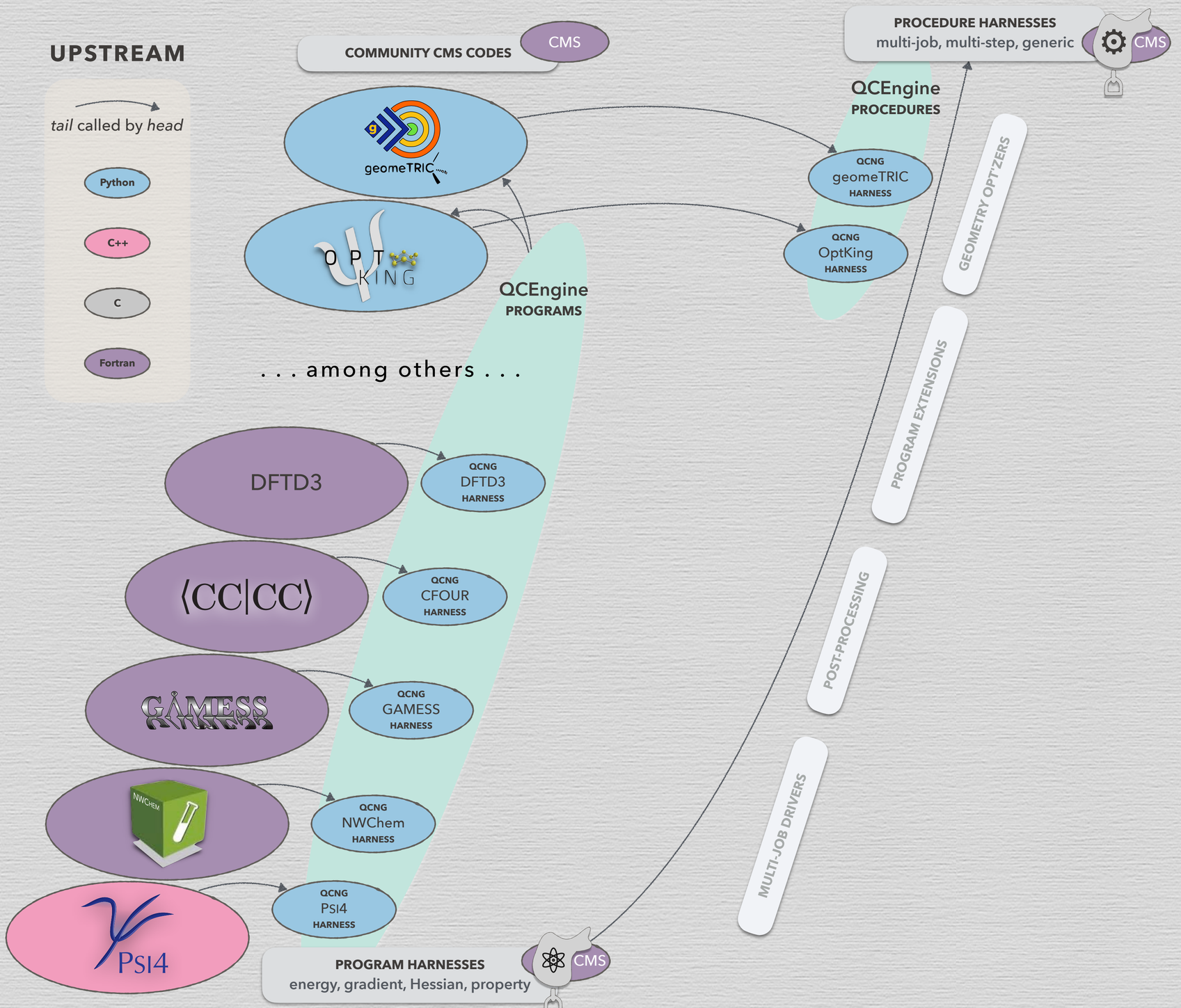
- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. Only optimizers so far.
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety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

QCENGINE: AVAILABLE PROGRAM HARNESSES

github.com/MolSSI/QCEngine

CMS Program	QCENGINE E	G	H	Prop.	Wfn
Quantum Chemistry					
ADCC	✓	✗	✗	✓	✗
CFOUR	✓	✓	✓	✓	✗
GAMESS	✓	✓	✗	✓	✗
MOLPRO	✓	✓	✗	✓	✗
MRCHEM	✓	✗	✗	✓	✗
NWCHEM	✓	✓	✓	✓	✗
PSI4	✓	✓	✓	✓	✓
Q-CHEM	✓	✓	✓	✗	✗
QCORE	✓	✓	✓	✗	✓
TERACHEM	✓	✓	✗	✓	✗
TURBOMOLE	✓	✓	✓	✗	✗
Semi-Empirical					
MOPAC	✓	✓	✗	✓	✗
XTB	✓	✓	✗	✓	✗
Molecular Mechanics					
OPENMM	✓	✓	✗	✓	
RDKIT	✓	✓	✗	✓	
Analytical Corrections					
DFTD3	✓	✓	✗	✗	
DFTD4	✓	✓	✗	✗	
GCP	✓	✓	✗	✗	
MP2D	✓	✓	✗	✗	
Machine Learning Inference					
TORCHANI	✓	✓	✓	✓	

API	XML / JSON / BIN	TXT	QCSk
adcc			
	CFOUR	CFOUR	
		GAMESS	
	Molpro		
	MRChem		
		NWChem	
Psi4			Psi4
	Q-Chem	Q-Chem	
	Qcore		
TeraChem		TeraChem	TeraChem
		Turbomole	
		MOPAC	
xtb			xtb
OpenMM			
RDKit			
		DFTD3	
DFTD4			DFTD4
		gCP	
		MP2D	
TorchANI			



QCENGINE: INTERNAL MAP

github.com/MolSSI/QCEngine

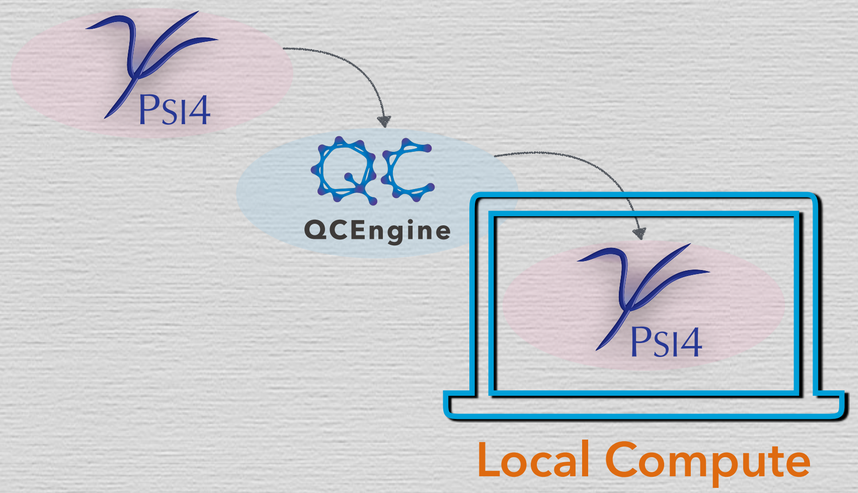
- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. Only optimizers so far.
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety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

DOWNSTREAM

(3) Psi4 DISTRIBUTED DRIVER, INTERNAL

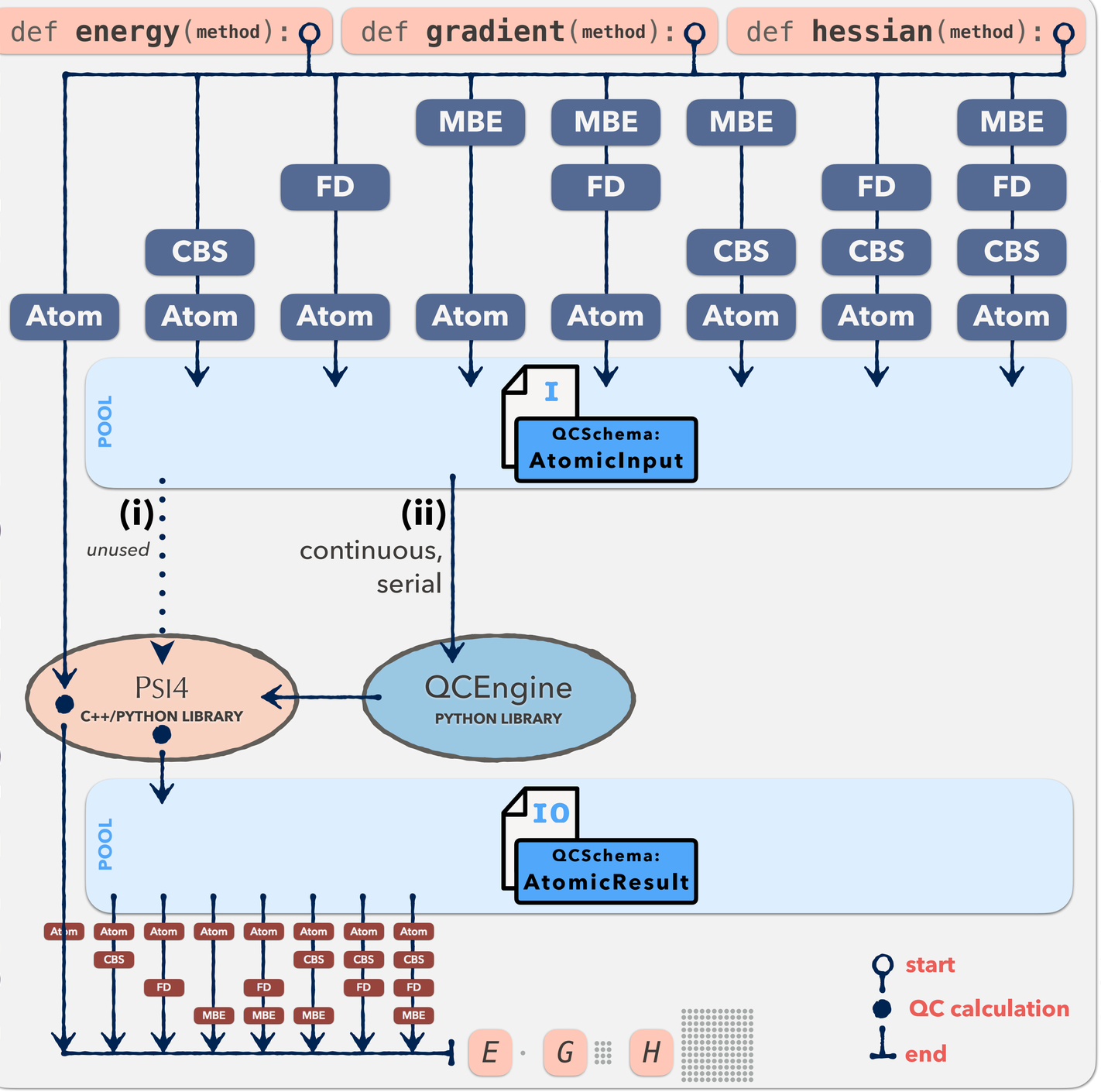
still capable, now orderly and extensible

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



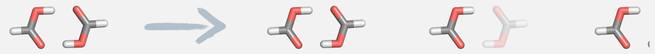
- **AVAILABLE** now with Psi4 dev branch & QCEngine (already a Psi4 dep).
- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by sum of all calcs since run sequentially.
- **RETRIEVAL** from filesystem difficult since many calcs in aggregated outfile.
- **FLEXIBILITY** limited to Psi4 only.

PSI4 DISTRIBUTED DRIVER



class ManyBodyComputer ():

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



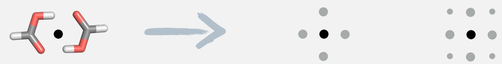
```
for frag in fragments: return qcschema
```

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
```

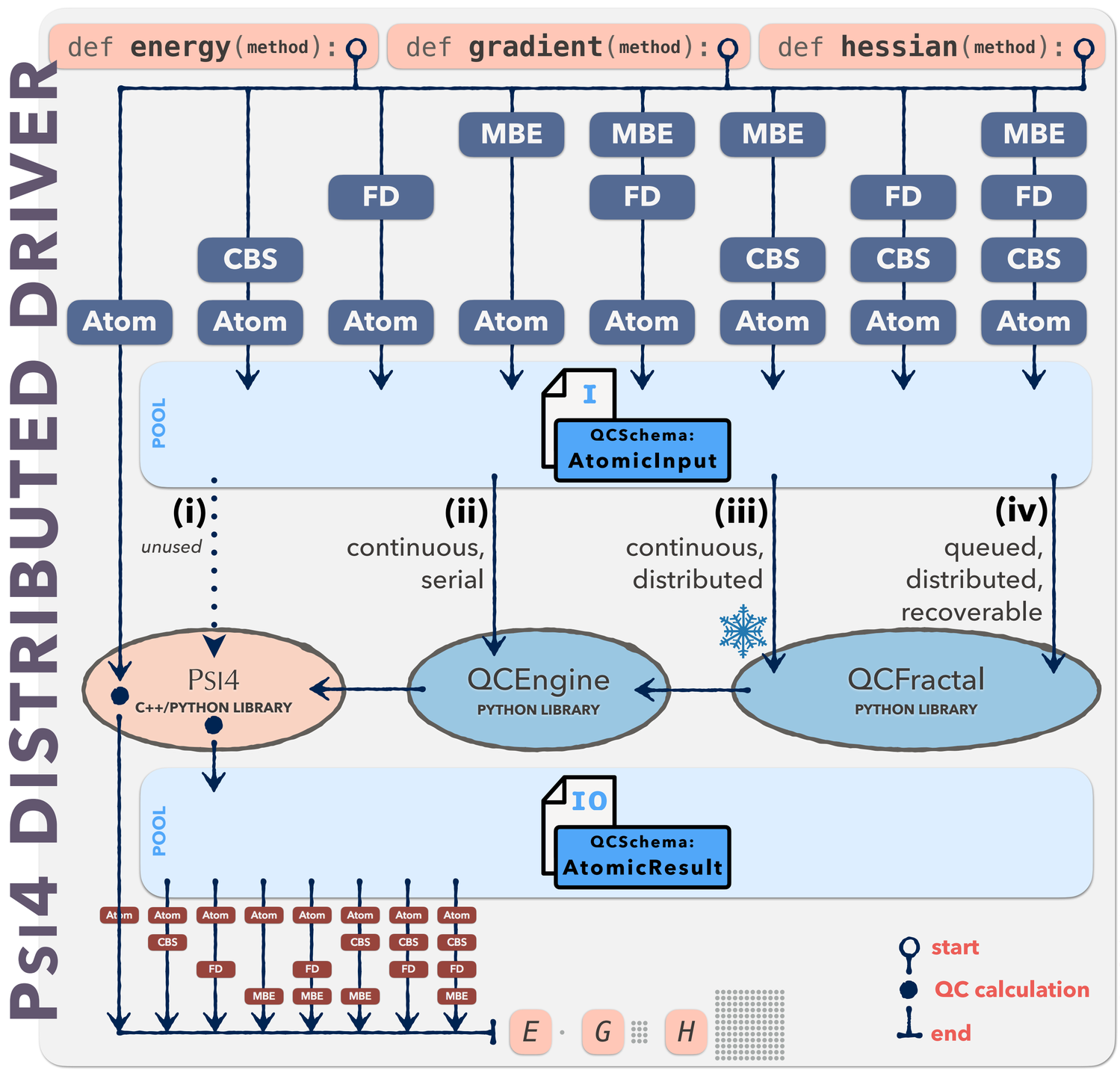
QCSchema

ASM Assemble extrapolations & total results from modelchems.

class AtomicComputer ():

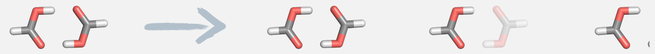
PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

ASM Return analytic energy, gradient, or Hessian.



class ManyBodyComputer ():

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

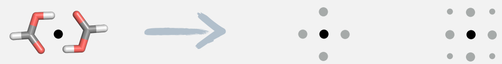


`for frag in fragments: return qcschema` QCSchema

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 ():

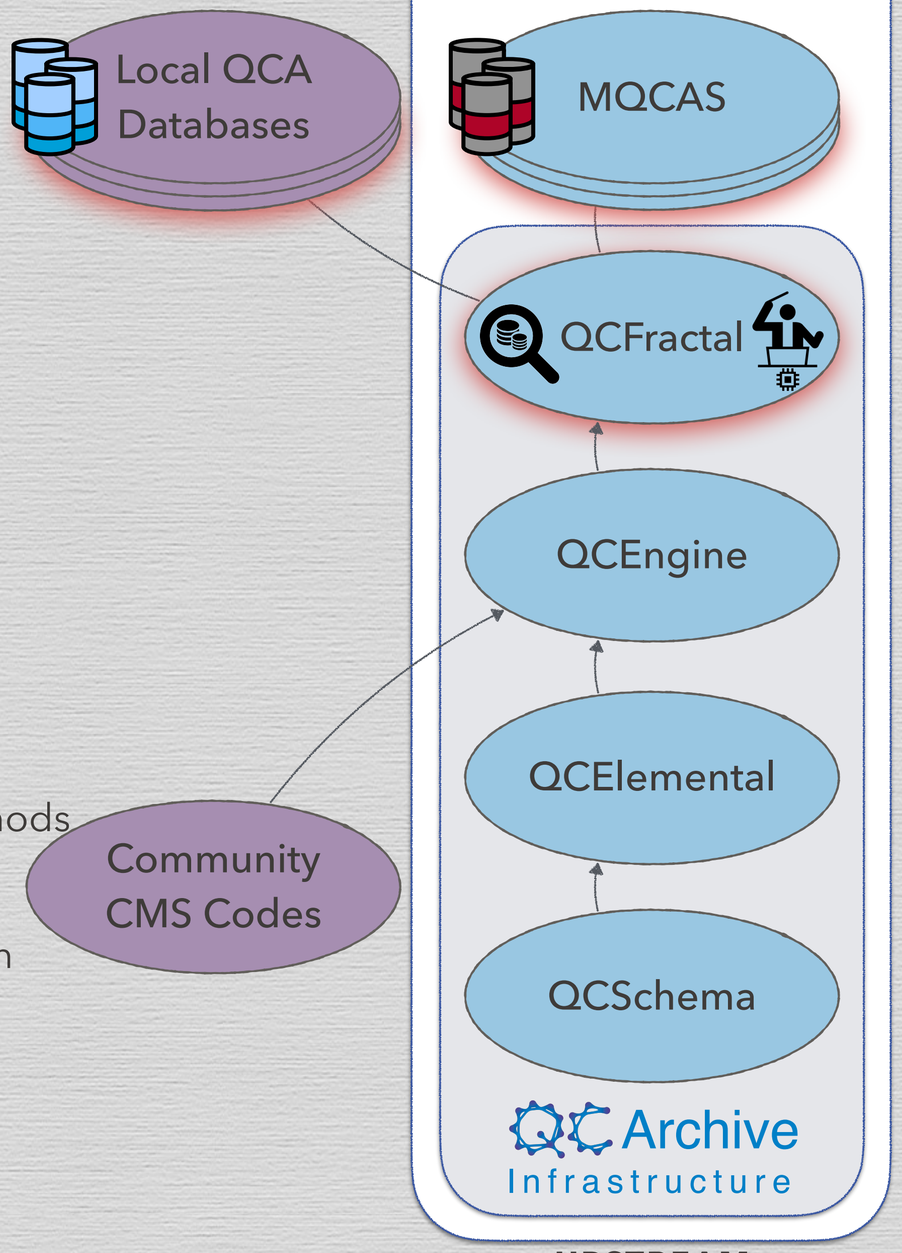
PLAN **molecule** & **method** unchanged. `return qcschema` QCSchema

ASM Return analytic energy, gradient, or Hessian.

QCArchive STACK

DOWNSTREAM

- 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

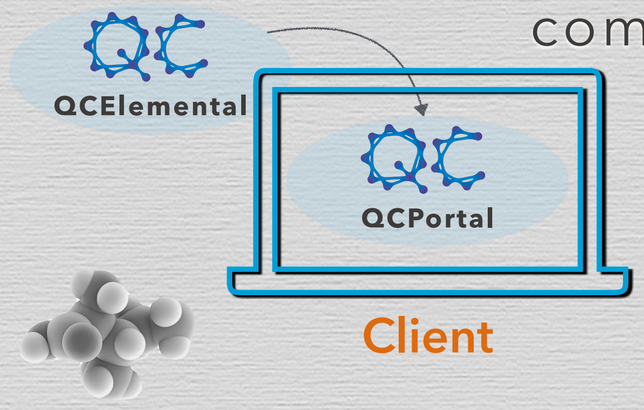
- the difficult part – coded QC methods
- structured output uncommon
- DSL input; API/schema uncommon

UPSTREAM

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org

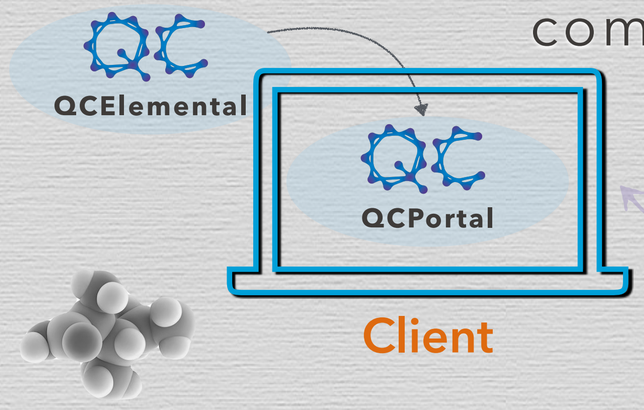


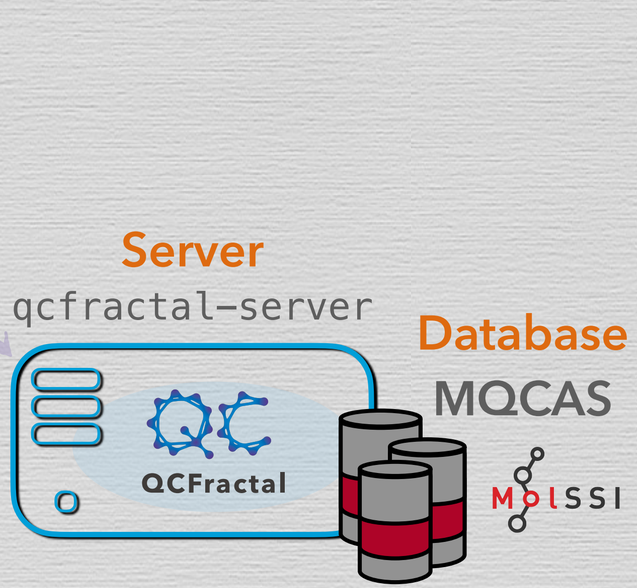
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- Completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE -** Up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC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

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



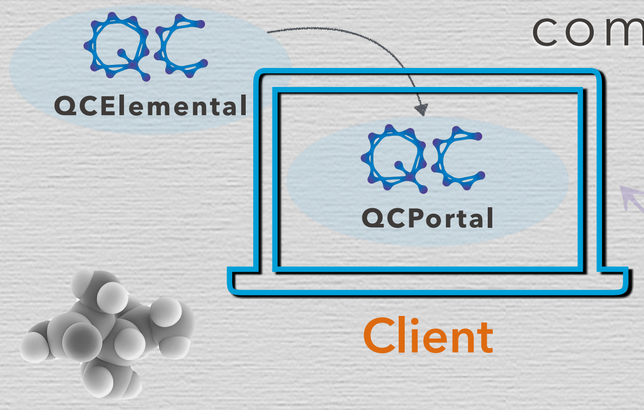


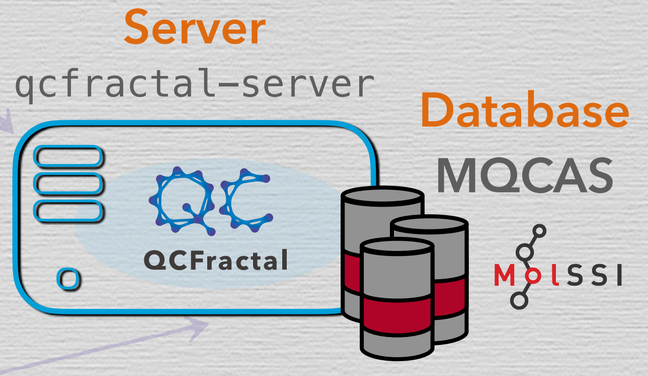
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- Completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE -** Up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC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

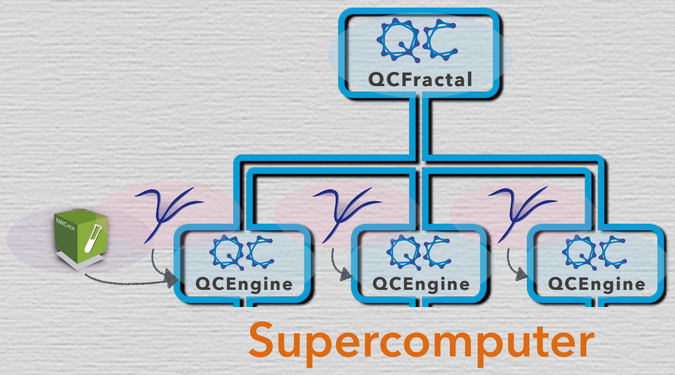
QCArchive OVERVIEW

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

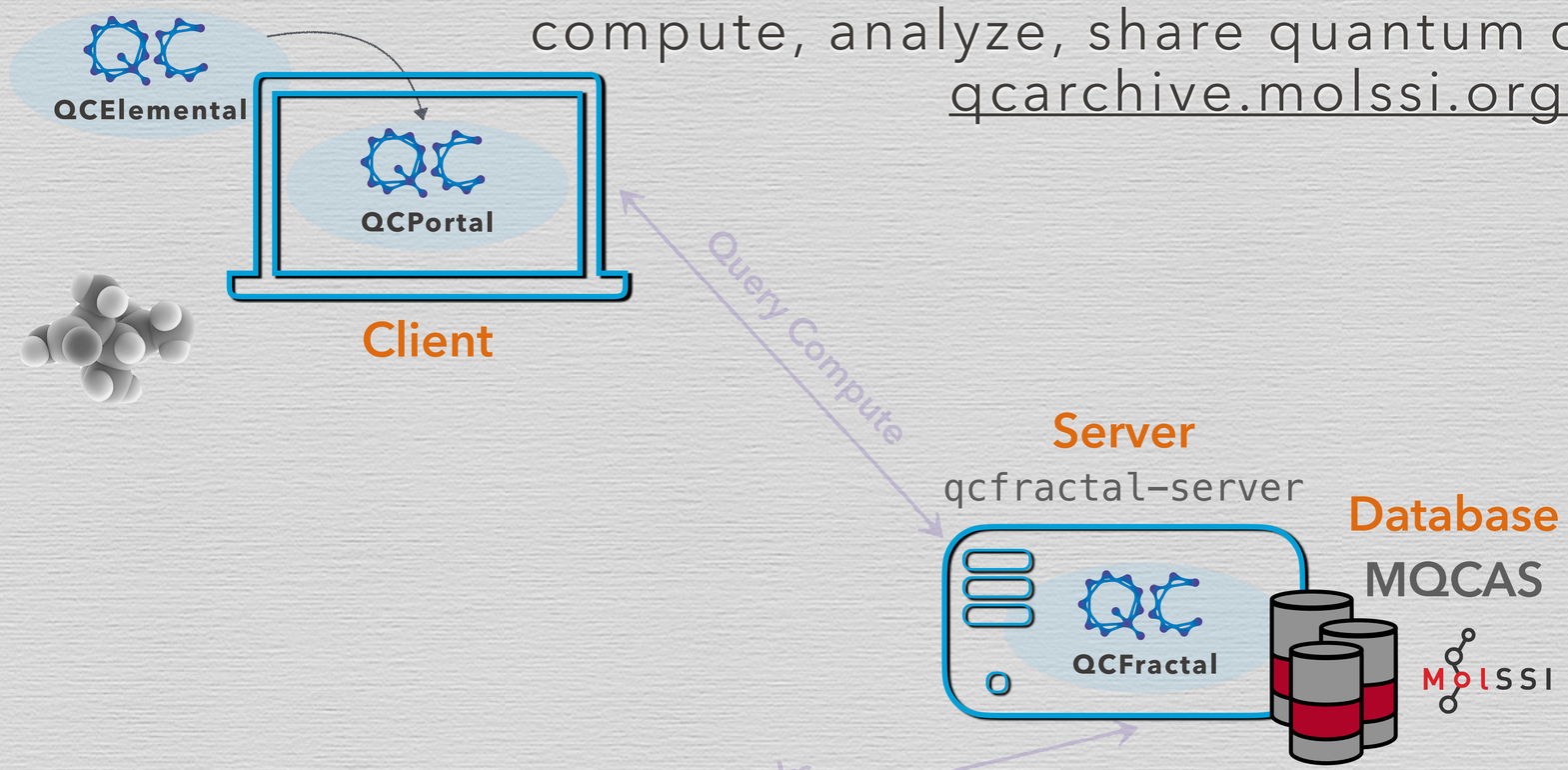
`qcfractal-manager` **Distributed Compute**

worker **Compute Nodes**

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org



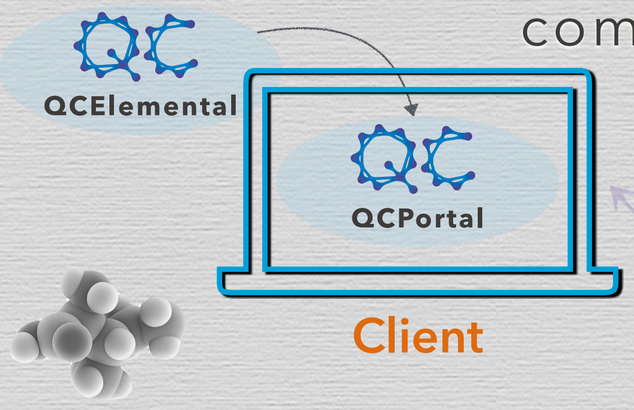
- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- Completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE -** Up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC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

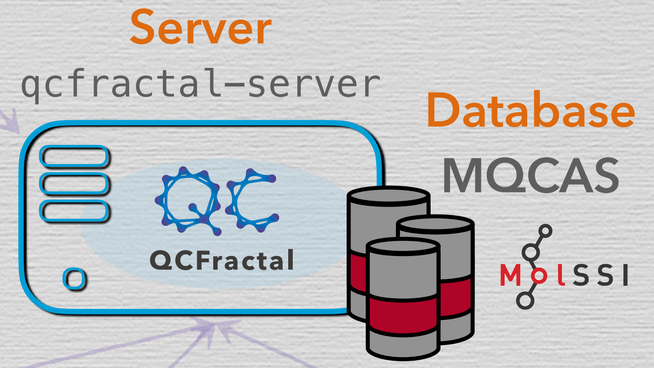
`qcfractal-manager`	**Distributed Compute**
adapter	**Task Management**
scheduler	**Task Scheduling**
worker	**Compute Nodes**

QCArchive OVERVIEW

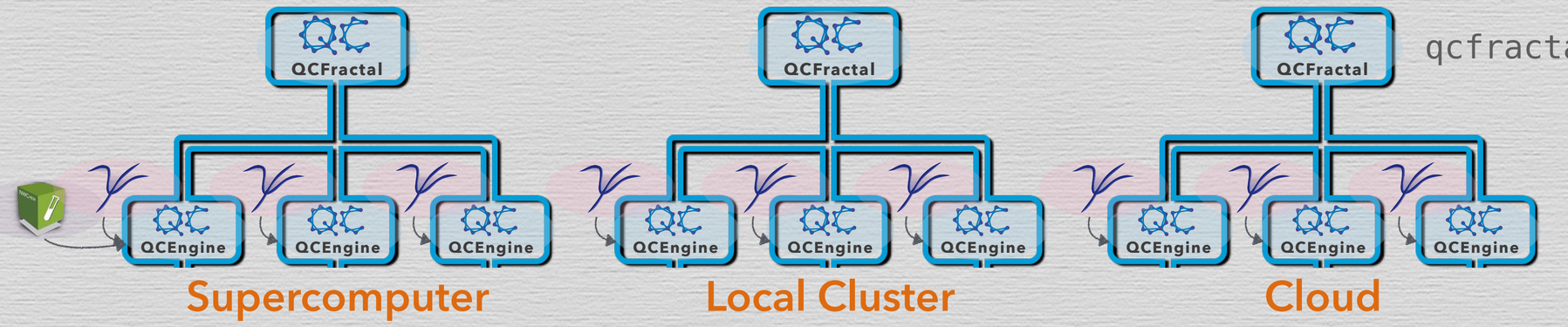
compute, analyze, share quantum chemistry data

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

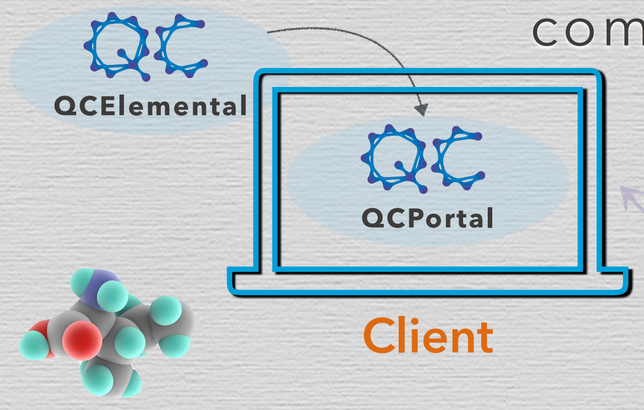


`qcfractal-manager`	Distributed Compute
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

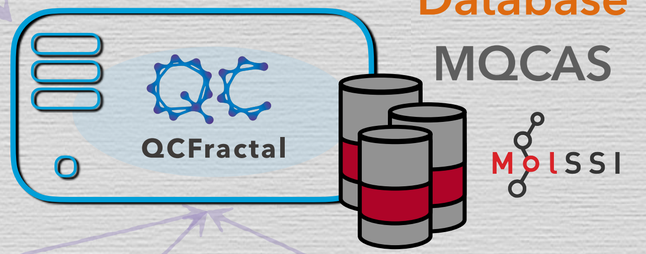
qcarchive.molssi.org



Query Compute

Server

qcfractal-server



Database

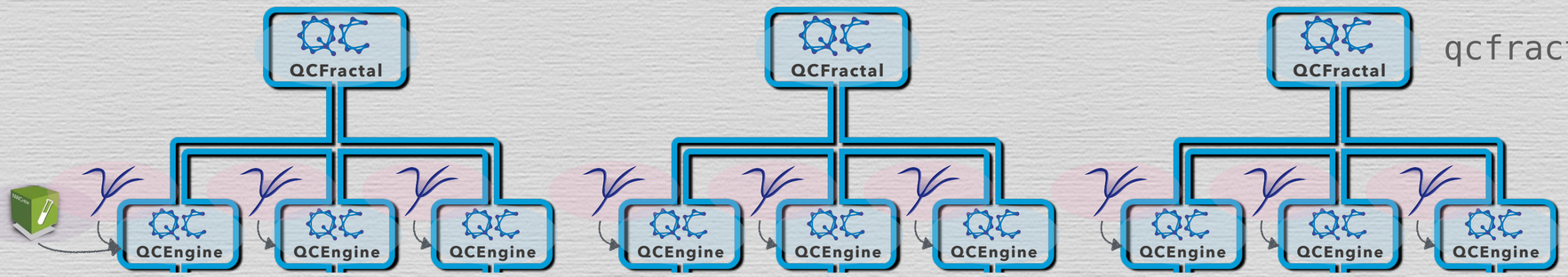
MQCAS

MolSSI

- 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

New Tasks

Complete Tasks

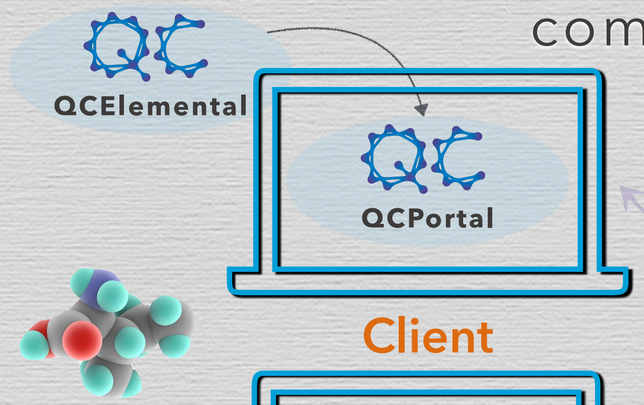


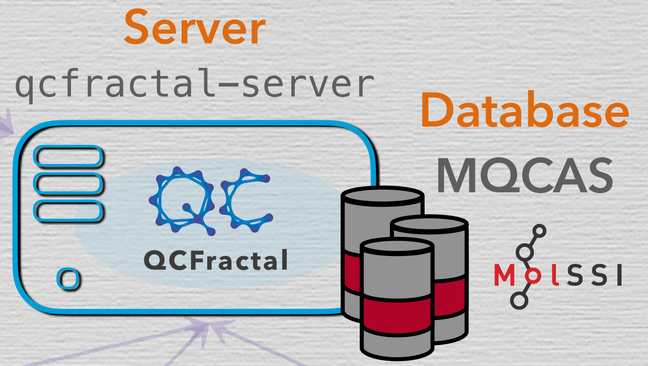
`qcfractal-manager`	Distributed Compute
adapter	Task Management
scheduler	Task Scheduling
worker	Compute Nodes

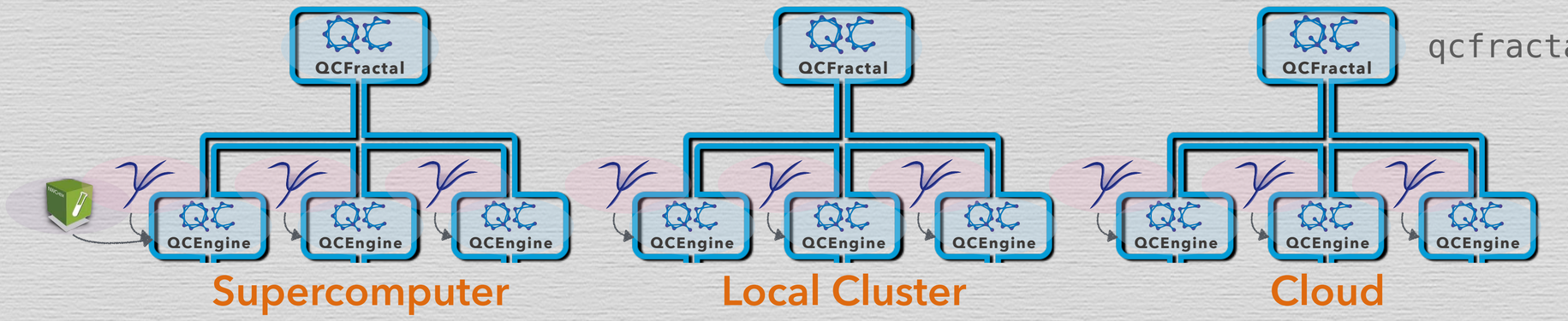
QCArchive OVERVIEW

compute, analyze, share quantum chemistry data

qcarchive.molssi.org





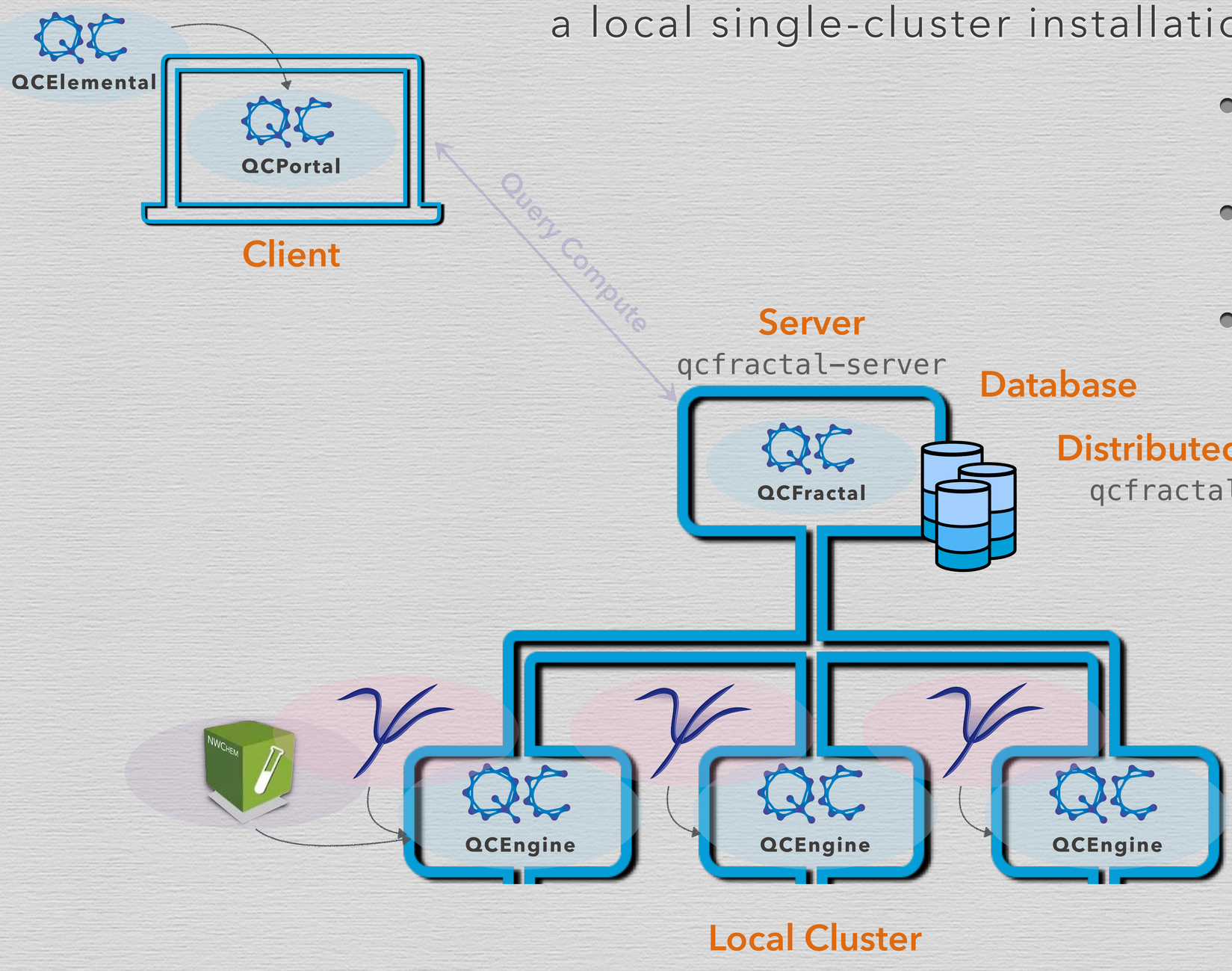


- hybrid **COMPUTE** and **DATA** management tool for quantum chemistry
- **SHARE** and **COLLABORATE** structured data
- **EASE OF USE** for non-specialist
- Completely installable from **CONDA**
- **MULTI-PHYSICAL-SITE** compute
- **SCALE -** Up to 500 tasks/second, 300,000 concurrent tasks
- **ELASTIC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

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

QCARCHIVE, SMALLER

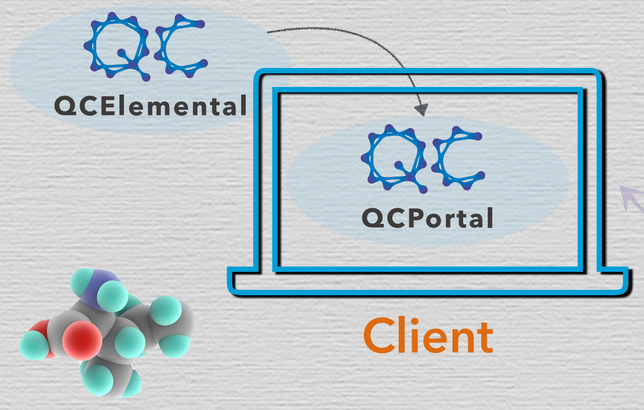
a local single-cluster installation



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

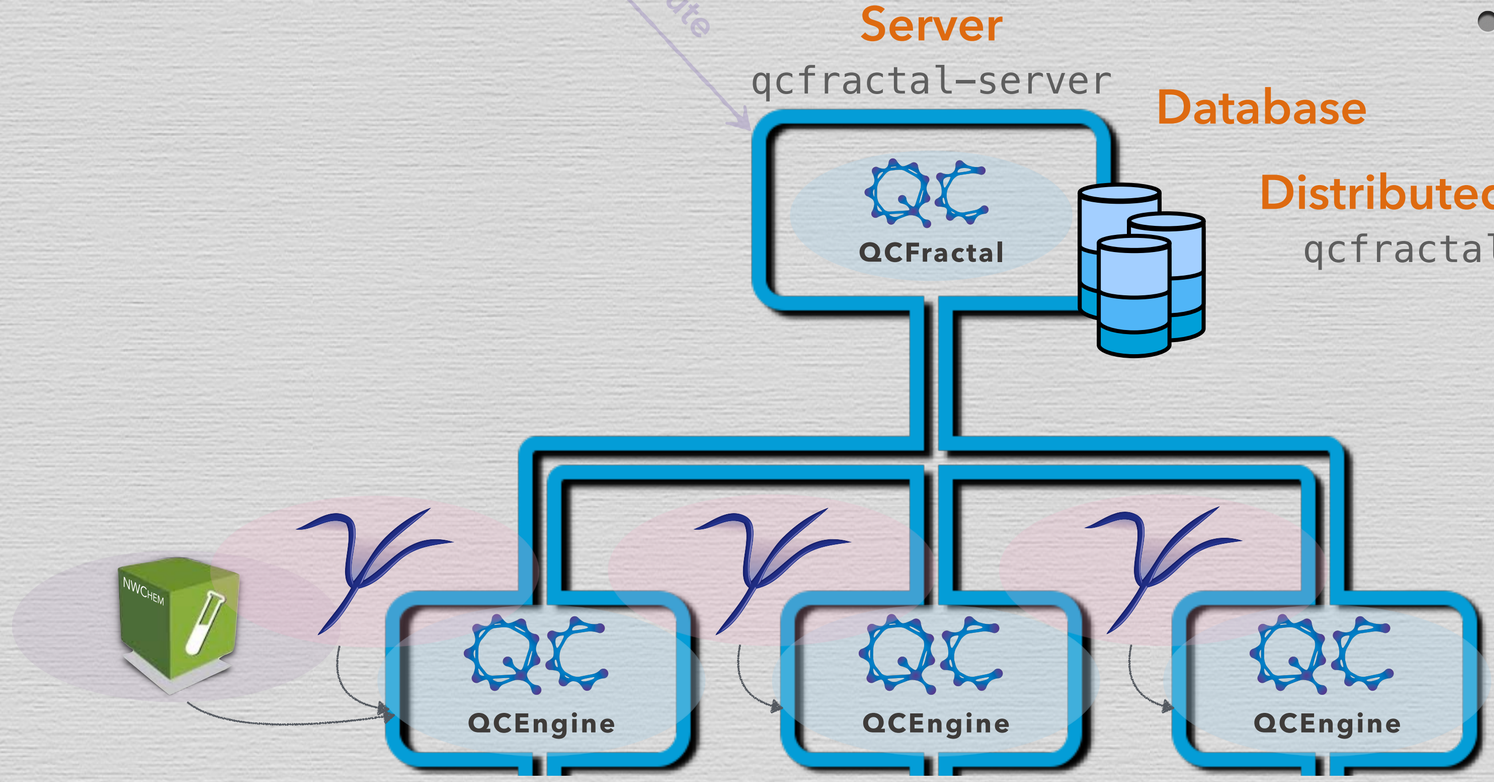
QCARCHIVE, SMALLER

a local single-cluster installation



Query Compute

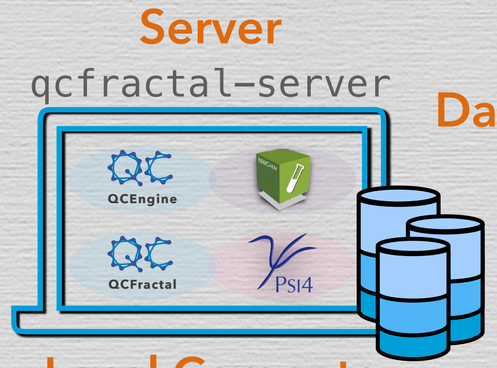
- **QCARCHIVE INFRASTRUCTURE** software stack unchanged
- **COLOCATE** QCFractal and database for task submission and orchestration
- **LOCAL** database fully powerful as MQCAS but locally controlled



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



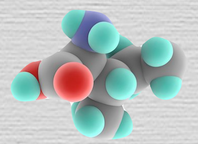
Distributed Compute

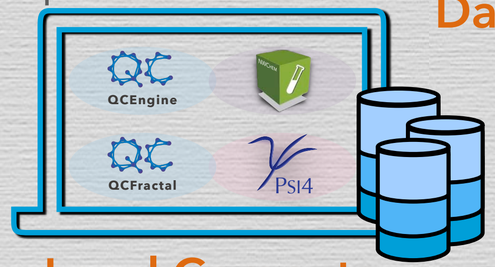
`qcfractal-manager`

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





Server

`qcfractal-server`

Database

Distributed Compute

`qcfractal-manager`

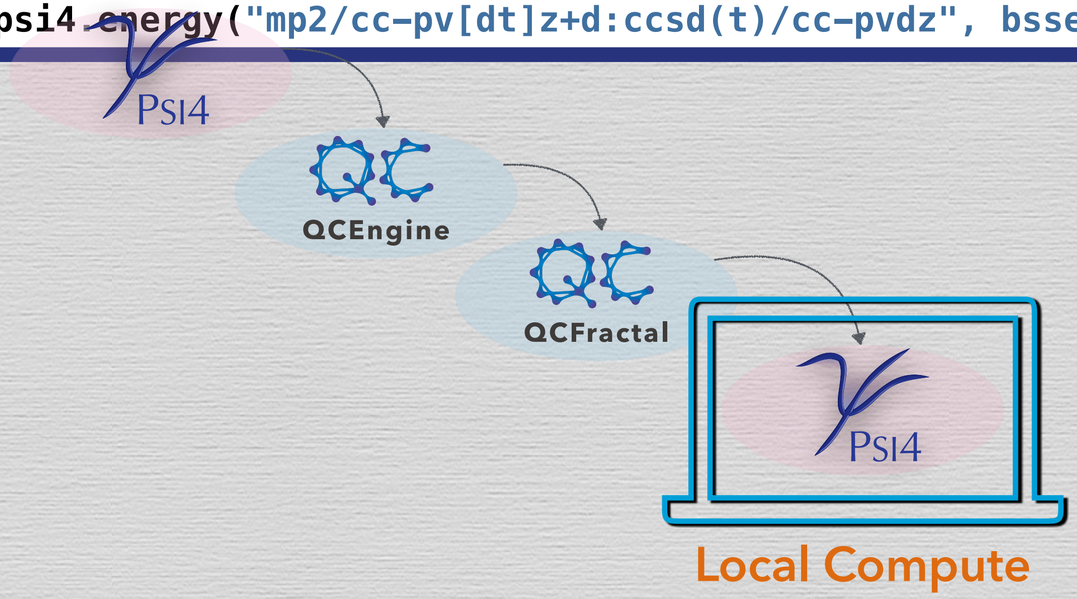
Local Compute

worker

(4) 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= )
```



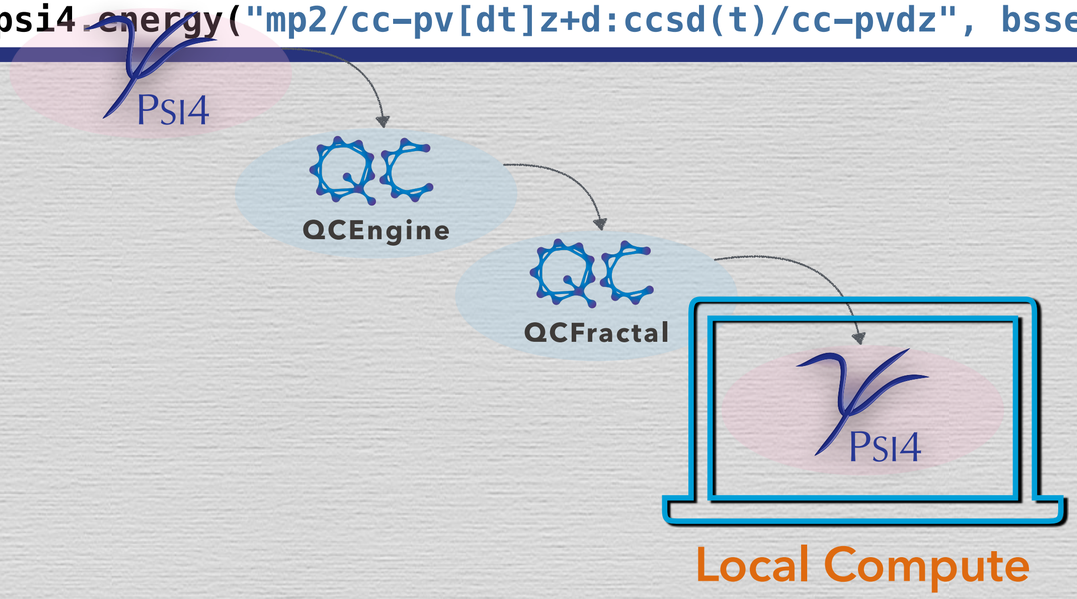

```
import psi4
from qcfractal import FractalServer

server = FractalServer()
client = server.client()

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

qcsk = plan.get_results(client)
print(qcsk.return_result)  # CP IE
```

- **AVAILABLE** now with Psi4 dev branch & QCEngine & QCFractal
- **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.

(4) 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 qcfractal import FractalServer

server = FractalServer()
client = server.client()

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

qcsk = plan.get_results(client)
print(qcsk.return_result)  # CP IE
```

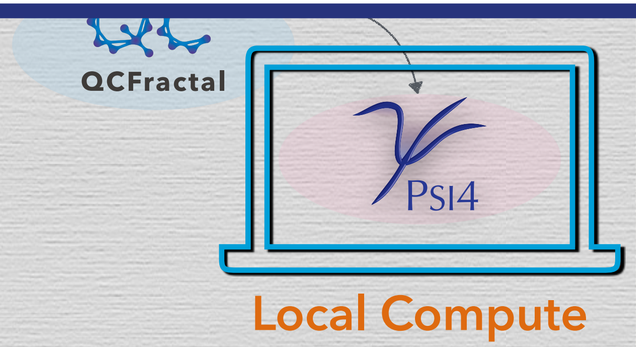
```
plan = psi4.energy("CCSD(T)/cc-pVDZ",
                   return_plan=True,
                   molecule= )
plan.compute(client)  # free! calcs in database

qcsk = plan.get_results(client)
print(qcsk.return_result)  # CCSD(T) energy
```

- **AVAILABLE** now with Psi4 dev branch & QCEngine & QCFractal
- **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.

(4) 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=  )
psi4.gradient("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=  )
psi4.hessian("mp2/cc-pv[dt]z+d:ccsd(t)/cc-pvdz", bsse_type="cp", molecule=  )
```

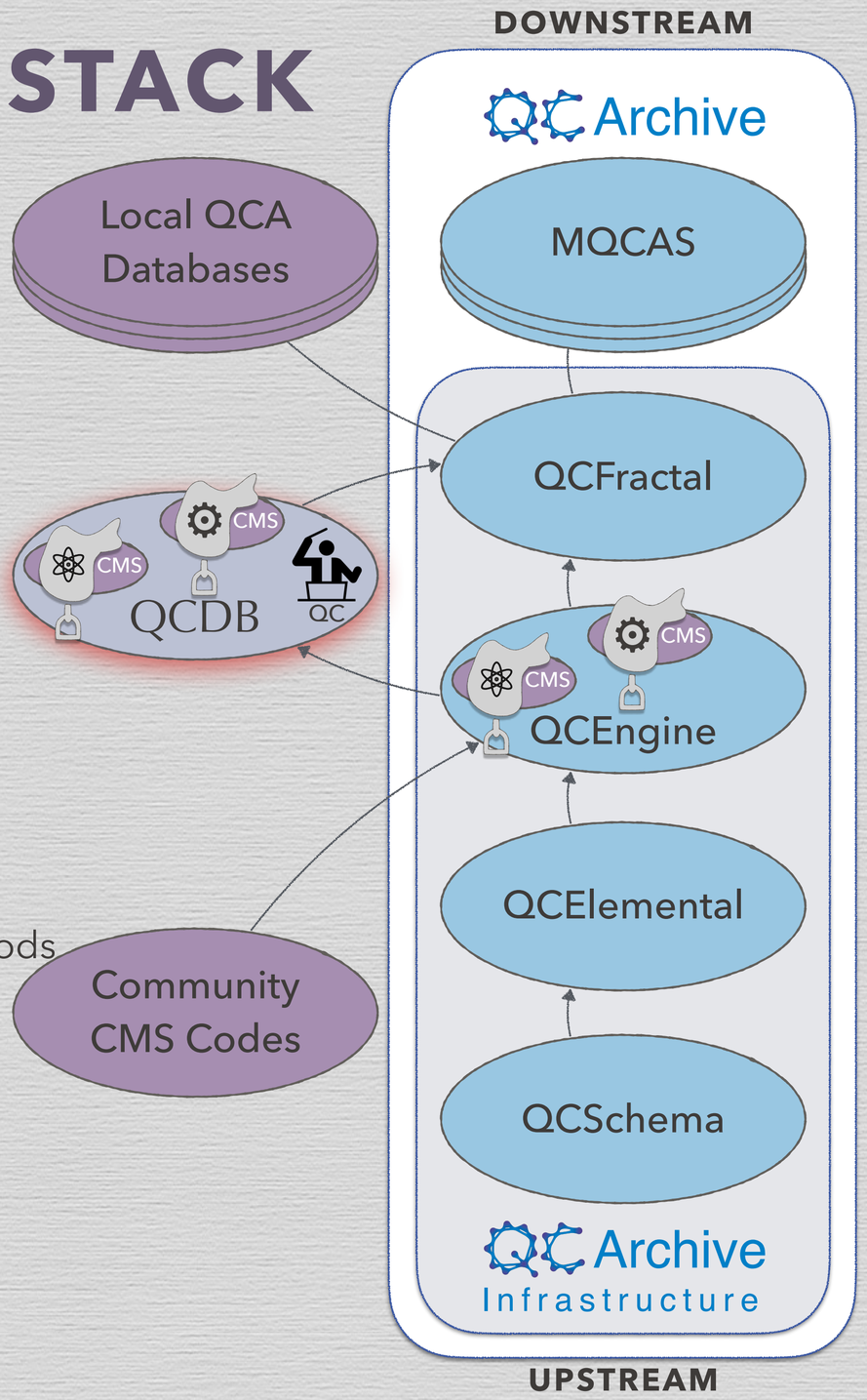



```
import psi4
server = FractalServer()
client = server.client()

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

qcsk = plan.get_results(client)
print(qcsk.return_result)  # CP IE
```

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

qcsk = plan.get_results(client)
print(qcsk.return_result)  # CCSD(T) energy
```

- **AVAILABLE** now with Psi4 dev branch & QCEngine & QCFractal
- **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 STACK



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

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

- DSL or unified input syntax
- multi-QCprogram workflows
- standardization layers

- 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

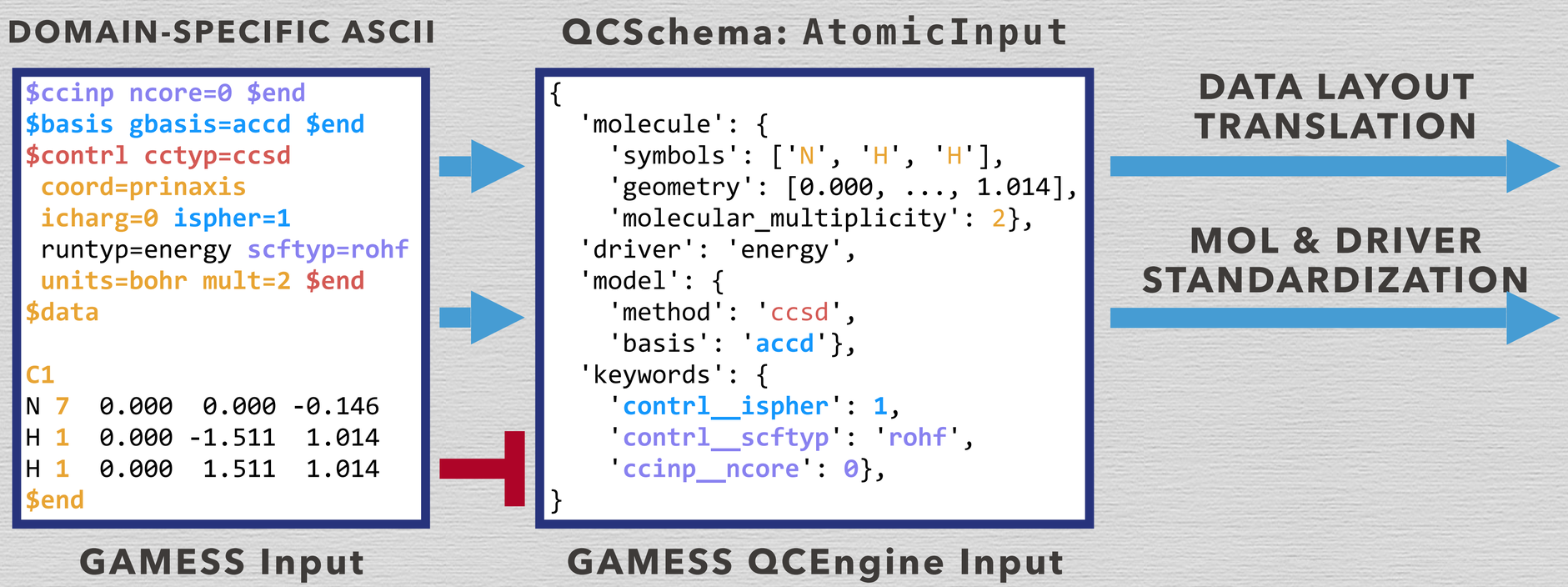
- the difficult part – coded QC methods
- structured output uncommon
- DSL input; API/schema uncommon

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

QCSchema DEFINES DATA LAYOUTS

QCDB unifies data contents

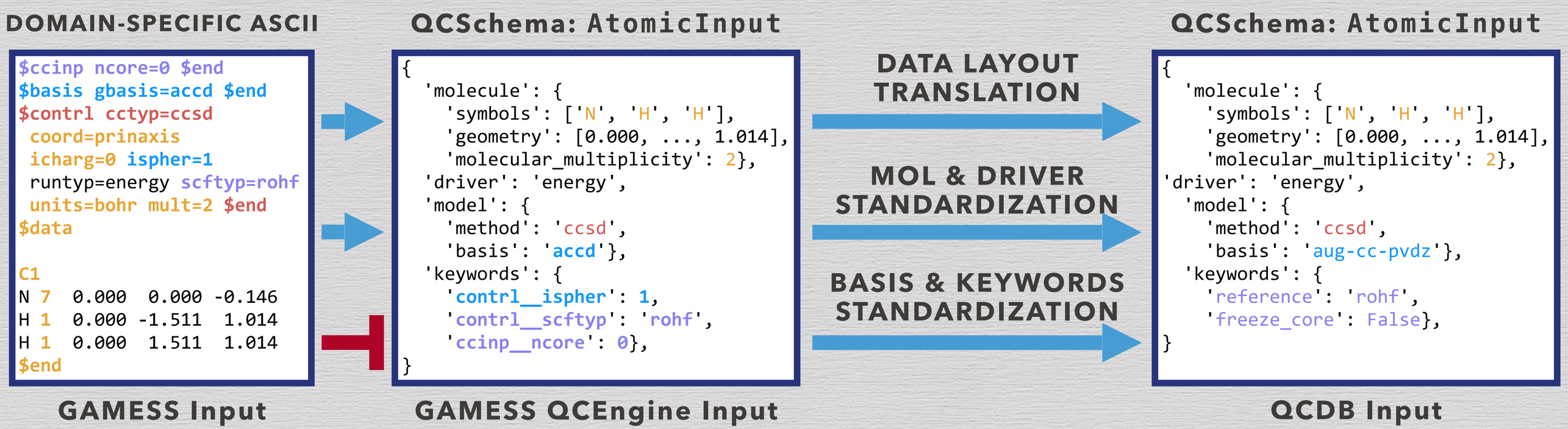
- **LAYOUT** JSON Schema defines a data layout and some minimal type and count checking.
- **CONVENTIONS** JSON Schema can't check conventions you rely upon like AU units, CCA ordering, center-of-mass positioning

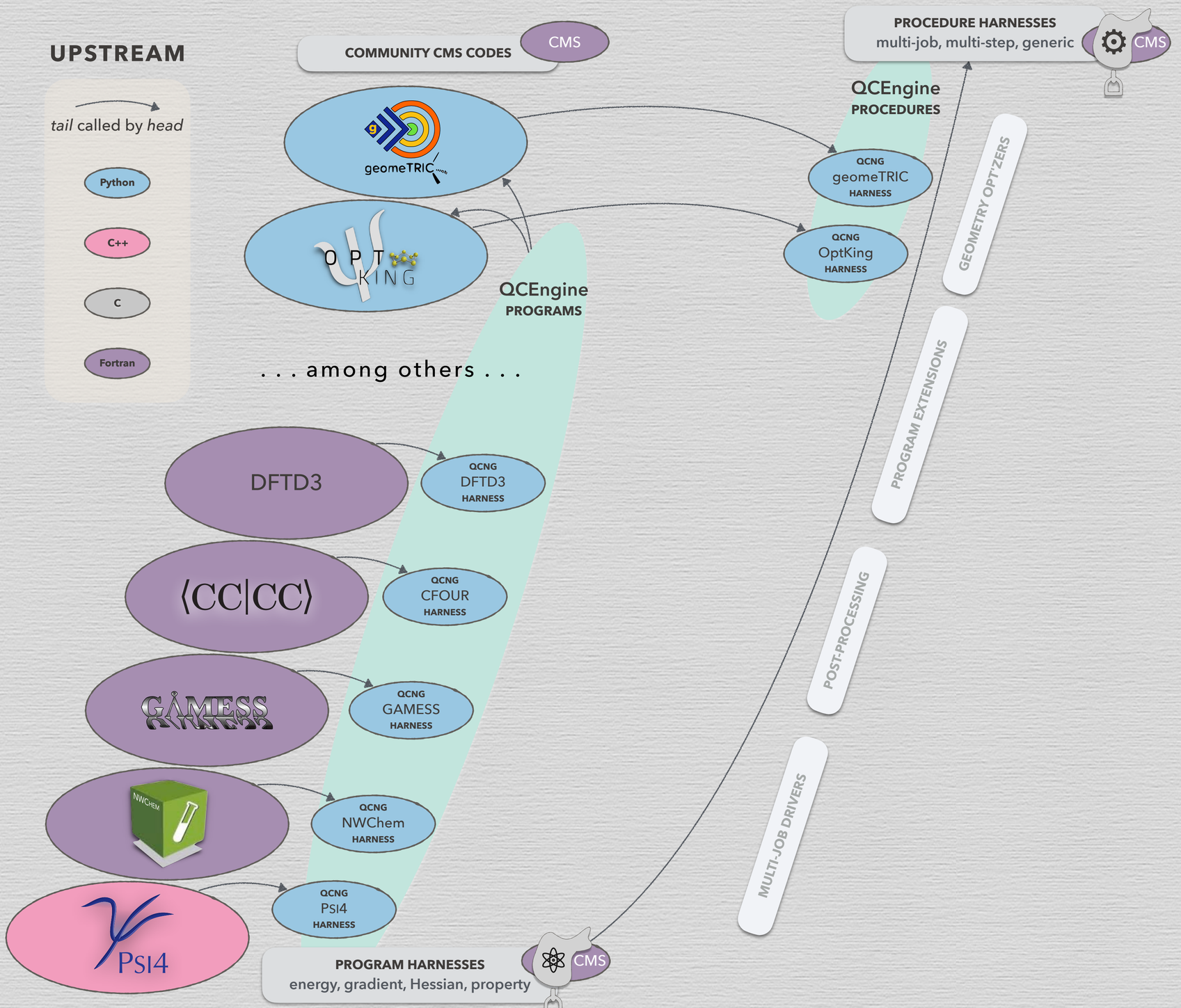


QCSchema DEFINES DATA LAYOUTS

QCDB unifies data contents

- **LAYOUT** JSON Schema defines a data layout and some minimal type and count checking.
- **CONVENTIONS** JSON Schema can't check conventions you rely upon like AU units, CCA ordering, center-of-mass positioning
- **QCDB UNIFIED CONTROL & KEYWORD TRANSLATION** allows users to focus on scientific choices, not DSL.
- **LOWER BARRIERS** to using a variety of codes, many with unique features.





QCENGINE: INTERNAL MAP

github.com/MolSSI/QCEngine

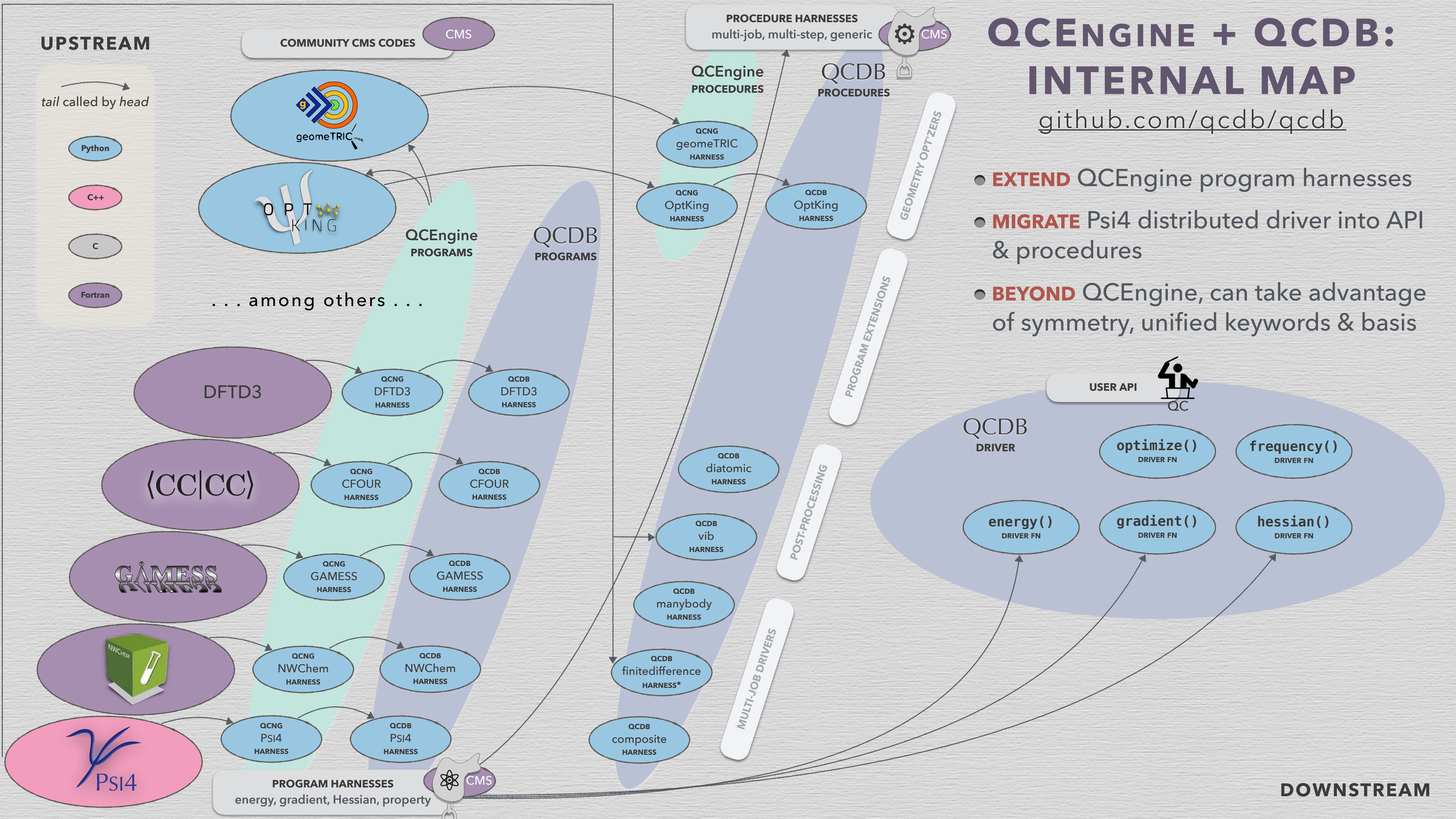
- **SCHEMA** runner
- **PROGRAMS** any analytic single point from CMS code
- **PROCEDURES** anything except analytic single point. Only optimizers so far.
- **PROGRAM CHECKS** indicate some methods accessible through QCSchema
- **CMS** primarily QM but also SE, MM, partial, or ML
- **INTERFACE** in variety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

DOWNSTREAM

QCENGINE + QCDB: INTERNAL MAP

github.com/qcdb/qcdb

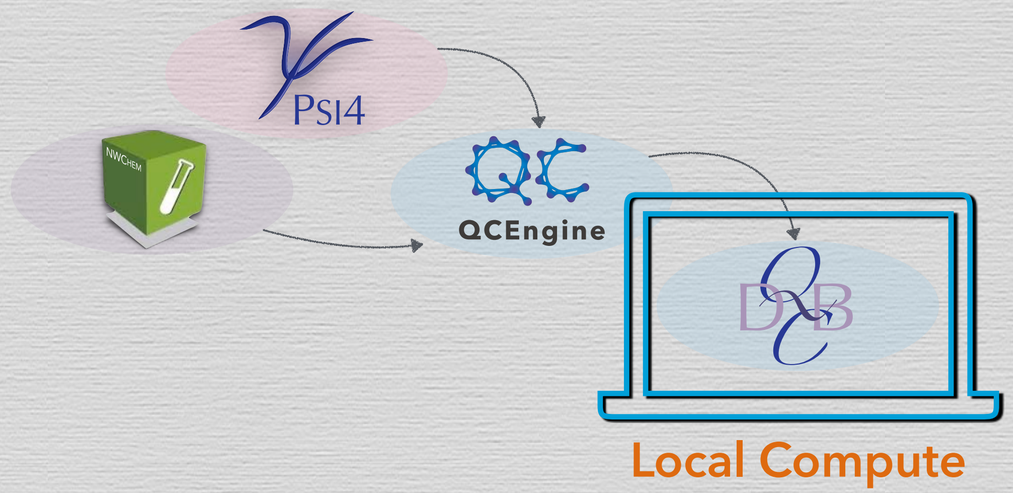
- **EXTEND** QCEngine program harnesses
- **MIGRATE** Psi4 distributed driver into API & procedures
- **BEYOND** QCEngine, can take advantage of symmetry, unified keywords & basis



(5) QCDB DISTRIBUTED DRIVER, INTERNAL

Psi4 procedures generalized to other QC programs

```
qcdb.energy("p4-mp2/cc-pv[dt]z+d:nwc-ccsd(t)/cc-pvdz", bsse_type="cp", molecule= )
```

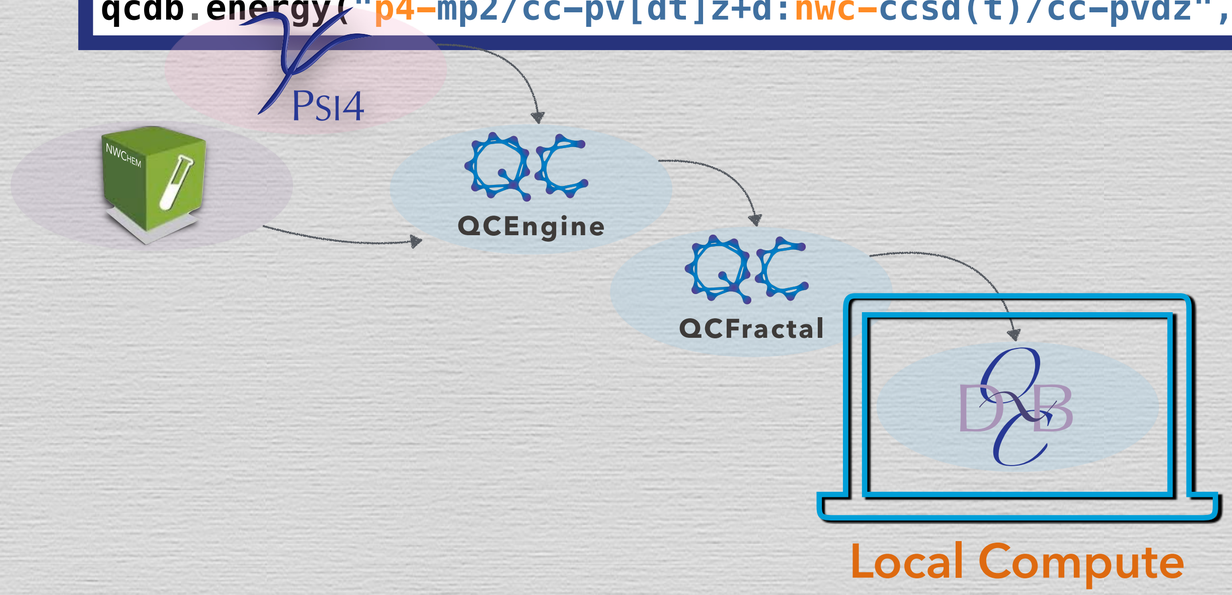


- **AVAILABLE** now with QCDB dev branch & Psi4 & QCEngine & NWChem (or CFOUR or GAMESS).
- **SPECIFICATION** through single file. Procedure automated, so low risk of user error.
- **RATE-LIMITED** by sum of all calcs since run sequentially.
- **RETRIEVAL** from filesystem difficult since many calcs in aggregated outfile.
- **FLEXIBILITY** to mix QC programs to access more or more efficient methods.

(6) QCDB DISTRIBUTED DRIVER, QCARCHIVE

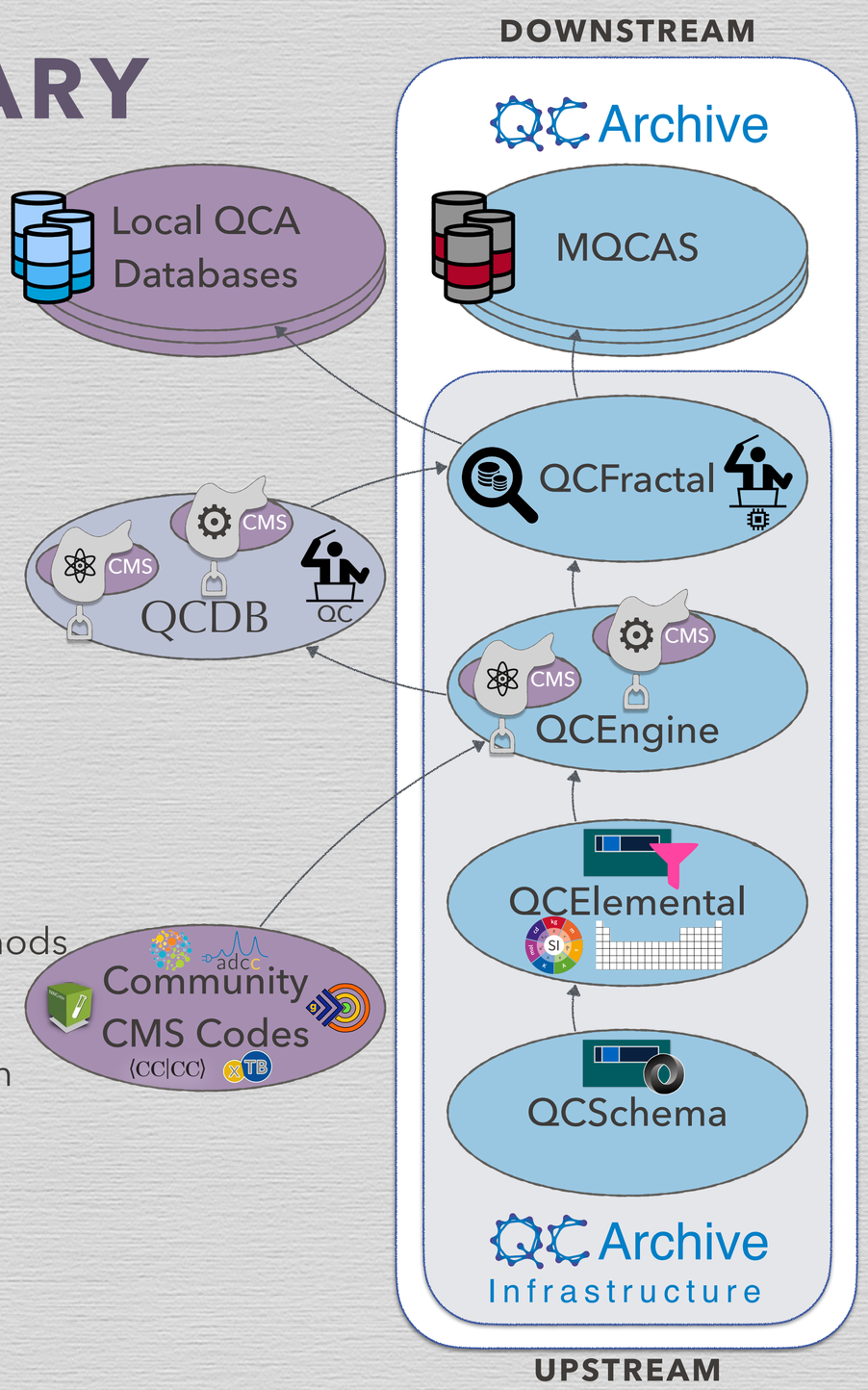
distributed, searchable, error-resistant, *and* generalized

```
qcdb.energy("p4-mp2/cc-pv[dt]z+d:nwc-ccsd(t)/cc-pvdz", bsse_type="cp", molecule=      )
```



- **AVAILABLE** untested with QCDB dev branch & Psi4 & QCEngine & QCFractal & NWChem (or CFOUR or GAMESS).
- **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** to mix QC programs to access more or more efficient methods.

SUMMARY



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

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

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

- DSL or unified input syntax
- multi-QCprogram workflows
- standardization layers

- 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

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

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

QCArchive Acknowledgements

QCArchive


Daniel Smith
MolSSI → Entos


Levi Naden
MolSSI


Doaa Altarawy
MolSSI


Matthew Welborn
MolSSI → Entos


Ben Pritchard
MolSSI

MolSSI


Theresa Windus
Iowa State


Daniel Crawford
VaTech


Jessica Nash
MolSSI


Sam Ellis
MolSSI


Susi Lehtola
MolSSI

Open Force Field Consortium

openforcefield.org/

MolSSI

qcarchive.molssi.org

QCSchema


Daniel Smith
MolSSI → Entos


Aaron Virshup
Arzeda


Bert de Jong
LBNL


Geoff Hutchison
U Pittsburgh


Marcus Hanwell
Kitware → Brookhaven


Eric Berquist
Q-Chem


Ben Pritchard
MolSSI


Lori Burns
GaTech


Matthew Welborn
MolSSI → Entos

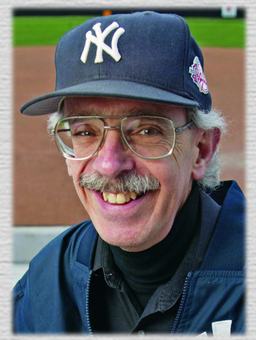
ADCC

Maximilian Scheurer
Heidelberg

Michael Herbst
Aachen

Andreas Dreuw
Heidelberg

GAMESS



Mark Gordon
Iowa State

Nuwan de Silva
W. New England U

TeraChem

Fang Liu
Emory

Heather Kulik
MIT

Colton Hicks
Stanford

Todd Martínez
Stanford

MRChem

Roberto Di Remigio
Uppsala

PSI4

Daniel Smith
MolSSI → Entos

Lori Burns
GaTech

David Sherrill
GaTech

Justin Turney
UGA

Andy Simmonett
NIH

Fritz Schaefer
UGA

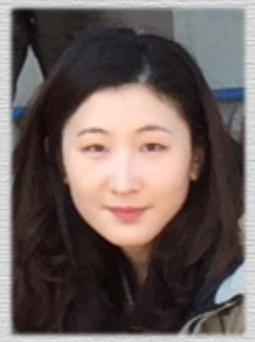
Zach Glick
GaTech

TURBOMOLE

Johannes Steinmetzer
Friedrich Schiller U

NWChem

Theresa Windus
Iowa State



Jiyoung Lee
U Texas

Annabelle Lolinco
Iowa State

Logan Ward
Argonne

Adrian Hurtado
Stony Brook

MOLPRO

Sebastian Lee
CalTech

CFOUR

John Stanton
UFL

Devin Matthews
SMU

OpenFF / MM

John Chodera
MSKCC

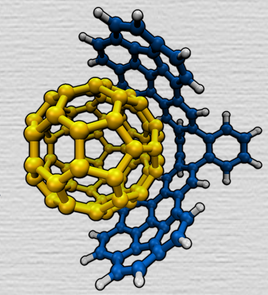
Jeffrey Wagner
UC Irvine

David Dotson
UC Boulder

Joshua Horton
Newcastle

Jamshed Anwar
Lancaster

SE / DISP.

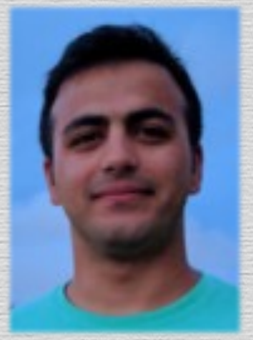


Sebastian Ehlert
Bonn

Holger Kruse
Czech Acad. Sci.

Jirí Šponer
Czech Acad. Sci.

ML



Farhad Ramezanghorbani
UFL → Schrödinger

OPTIMIZERS

Lee-Ping Wang
UC Davis

Jan Hermann
Free U Berlin

Alexander Heide
UGA

Rollin King
Bethel

PROCEDURES

Asim Alenaizan
GaTech

Peter Kraus
Curtin

Jonathon Misiewicz
Emory

Jeff Schriber
GaTech

Carlos Borca
Princeton

HIST./DB/TEST

Nick Petosa
GaTech → Two Sigma

Daniel Nascimento
U Memphis

Dom Sirianni
GaTech

Chaya Stern
MSKCC

ENGINES

Taylor Barnes
MolSSI

Psi4 Developers

Jerome Gonthier
Berkeley

Rob Parrish
Stanford

Holger Kruse
Czech Acad. Sci.

Rollin King
Bethel

Alexander Sokolov
Ohio State

David Sherrill
GaTech

Lori Burns
GaTech

Asim Alenaizan
GaTech

Zach Glick
GaTech

Jeff Schriber
GaTech

Francesco Evangelista
Emory

Eugene DePrince
FSU

Fritz Schaefer
UGA

Justin Turney
UGA

Daniel Crawford
VaTech

Daniel Smith
MolSSI

Ben Pritchard
MolSSI

Jonathon Misiewicz
Emory

Ed Valeev
VaTech

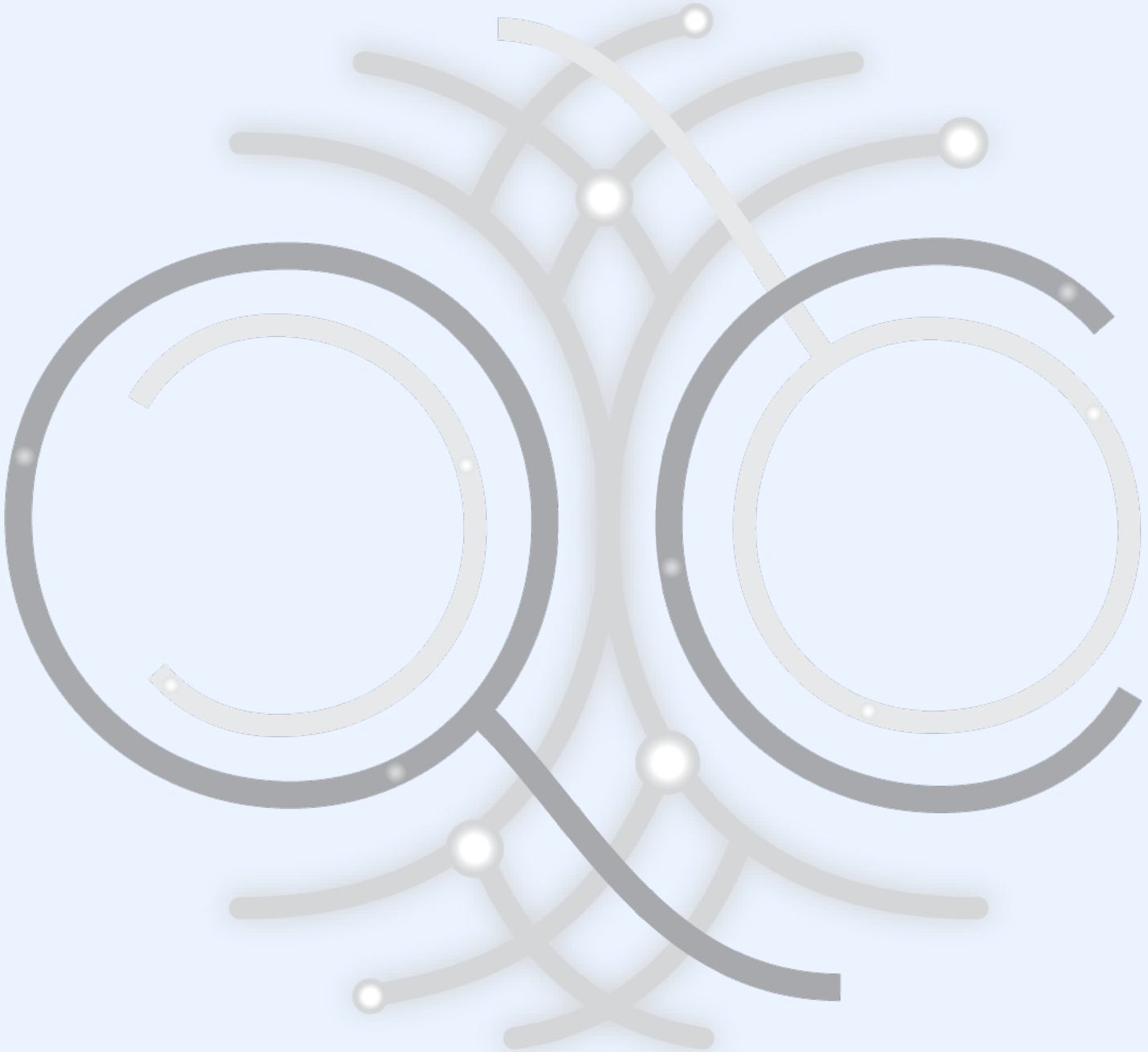
Andy Simmonett
NIH

Ed Hohenstein
CCNY

Roberto Di Remigio
Tromso

Ugur Bozkaya
Hacettepe

Peter Kraus
Curtin



PUBLICATIONS

Home > The Journal of Chemical Physics > Volume 152, Issue 18 > 10.1063/5.0006002

No Access • Submitted: 26 February 2020 • Accepted: 12 April 2020 • Published Online: 13 May 2020

PSI4 1.4: Open-source software for high-throughput quantum chemistry

J. Chem. Phys. **152**, 184108 (2020); https://doi.org/10.1063/5.0006002

Daniel G. A. Smith[1], Lori A. Burns[2], Andrew C. Simmonett[3], Robert M. Parrish[2], Matthew C. Schieber[2], Raimondas Galvelis[4], Peter Kraus[5], Holger Kruse[6], Roberto Di Remigio[7], Asem Alenaizan[2], Andrew M. James[8], Susi Lehtola[9], Jonathon P. Misiewicz[10], Maximilian Scheurer[11], Robert A. Shaw[12], Jeffrey B. Schriber[2], Yi Xie[2], Zachary L. Glick[2], Dominic A. Sirianni[2], Joseph Senan O'Brien[2], Jonathan M. Waldrop[13], Ashutosh Kumar[8], Edward G. Hohenstein[14], Benjamin P. Pritchard[1], Bernard R. Brooks[3], Henry F. Schaefer III[10], Alexander Yu. Sokolov[15], Konrad Patkowski[13], A. Eugene DePrince III[16], Uğur Bozkaya[17], Rollin A. King[18], Francesco A. Evangelista[19], Justin M. Turney[10], T. Daniel Crawford[1,8], *and* C. David Sherrill[2,a)]

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 | Cita

Home > The Journal of Chemical Physics > Volume 155, Issue 20 > 10.1063/5.0059356

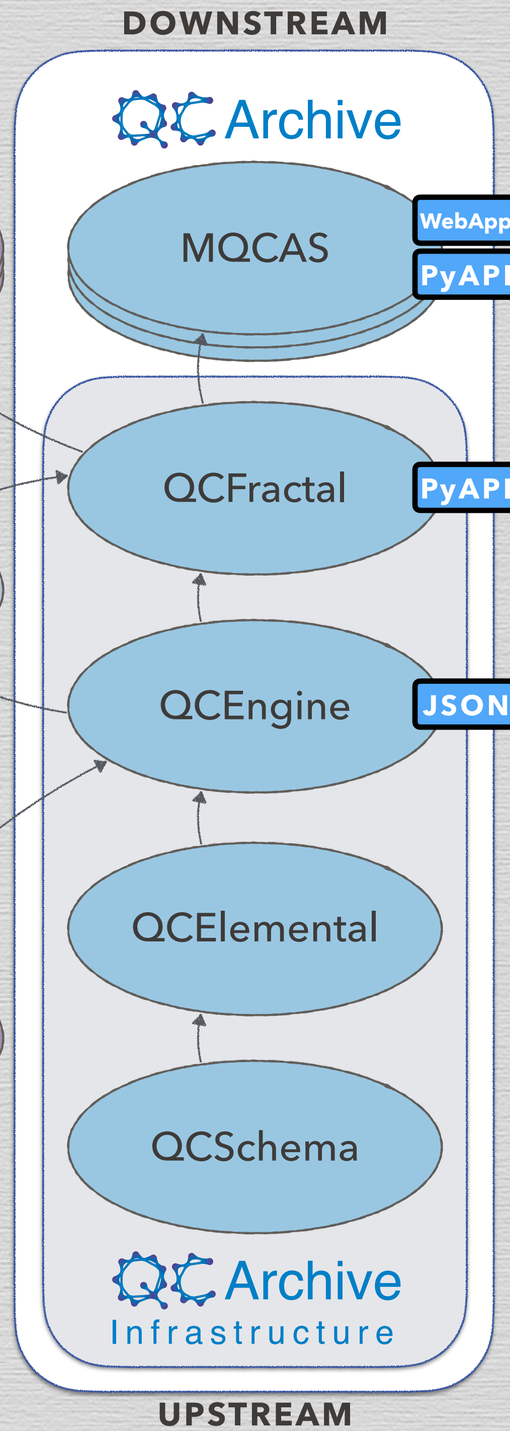
No Access • Submitted: 08 June 2021 • Accepted: 01 October 2021 • Published Online: 22 November 2021

Quantum Chemistry Common Driver and Databases (QCDB) and Quantum Chemistry Engine (QCEngine): Automation and interoperability among computational chemistry programs

J. Chem. Phys. **155**, 204801 (2021); https://doi.org/10.1063/5.0059356

Daniel G. A. Smith[1,a)], Annabelle T. Lolinco[2], Zachary L. Glick[3], Jiyoung Lee[2,b)], Asem Alenaizan[3,c)], Taylor A. Barnes[1], Carlos H. Borca[3], Roberto Di Remigio[4,d)], David L. Dotson[5], Sebastian Ehlert[6], Alexander G. Heide[7], Michael F. Herbst[8], Jan Hermann[9], Colton B. Hicks[10,11], Joshua T. Horton[12], Adrian G. Hurtado[13], Peter Kraus[14], Holger Kruse[15], Sebastian J. R. Lee[16], Jonathon P. Misiewicz[7,e)], Levi N. Naden[1], Farhad Ramezanghorbani[17], Maximilian Scheurer[18], Jeffrey B. Schriber[3], Andrew C. Simmonett[19], Johannes Steinmetzer[20], Jeffrey R. Wagner[5,21], Logan Ward[22], Matthew Welborn[1,a)], Doaa Altarawy[1,23], Jamshed Anwar[12], John D. Chodera[24], Andreas Dreuw[18], Heather J. Kulik[25], Fang Liu[25,e)], Todd J. Martínez[10,11], Devin A. Matthews[26,f)], Henry F. Schaefer III[7], Jiří Šponer[15], Justin M. Turney[7], Lee-Ping Wang[27], Nuwan De Silva[2,g)], Rollin A. King[28], John F. Stanton[29], Mark S. Gordon[30], Theresa L. Windus[30], C. David Sherrill[3], *and* Lori A. Burns[3,h)]

QCArchive STACK



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

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

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

- DSL or unified input syntax
- multi-QCprogram workflows
- standardization layers

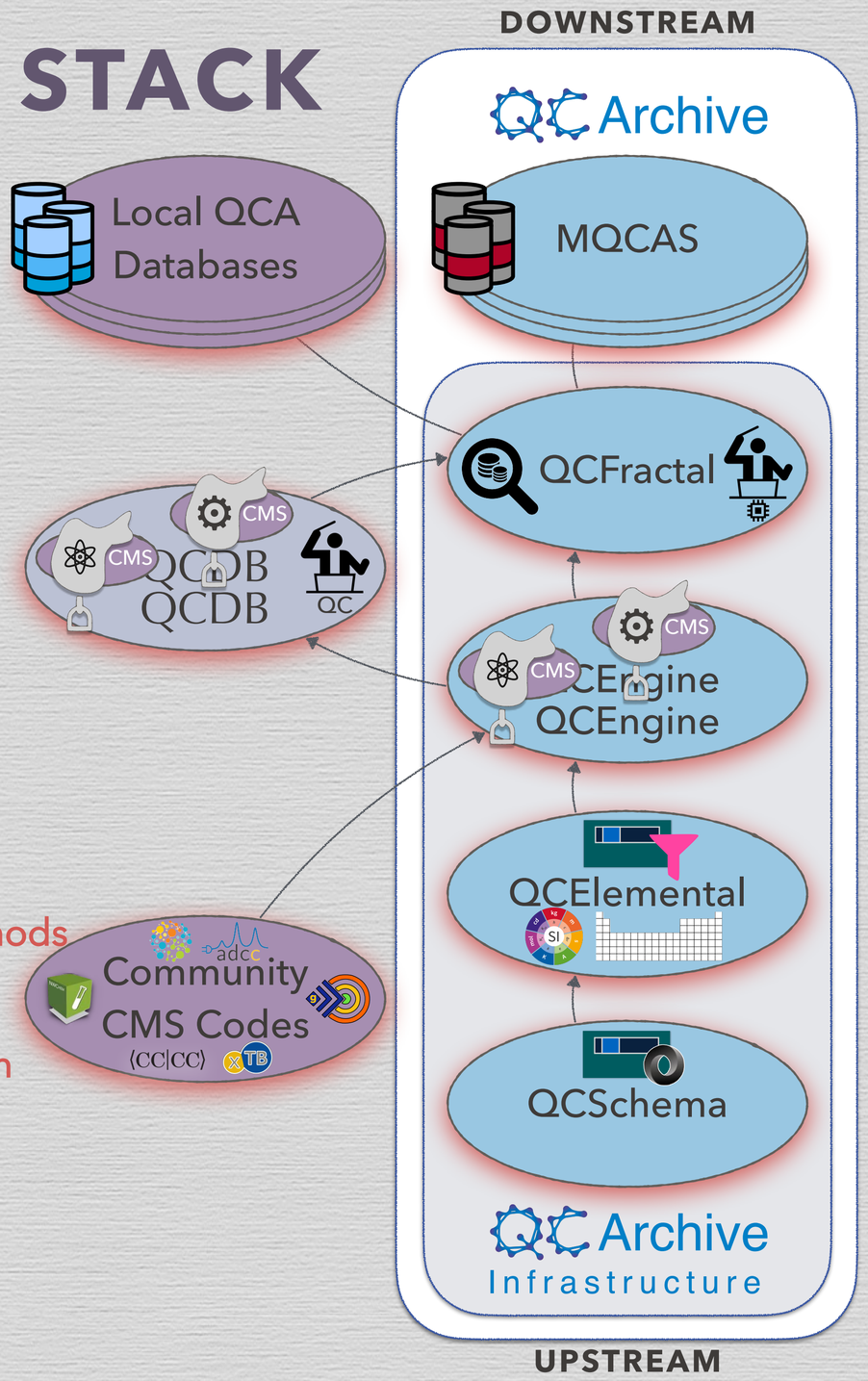
- Python QCSchema runner
- hardware compute configuration (e.g., memory, nodes)
- DSL input syntax for QC programs

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

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

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

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

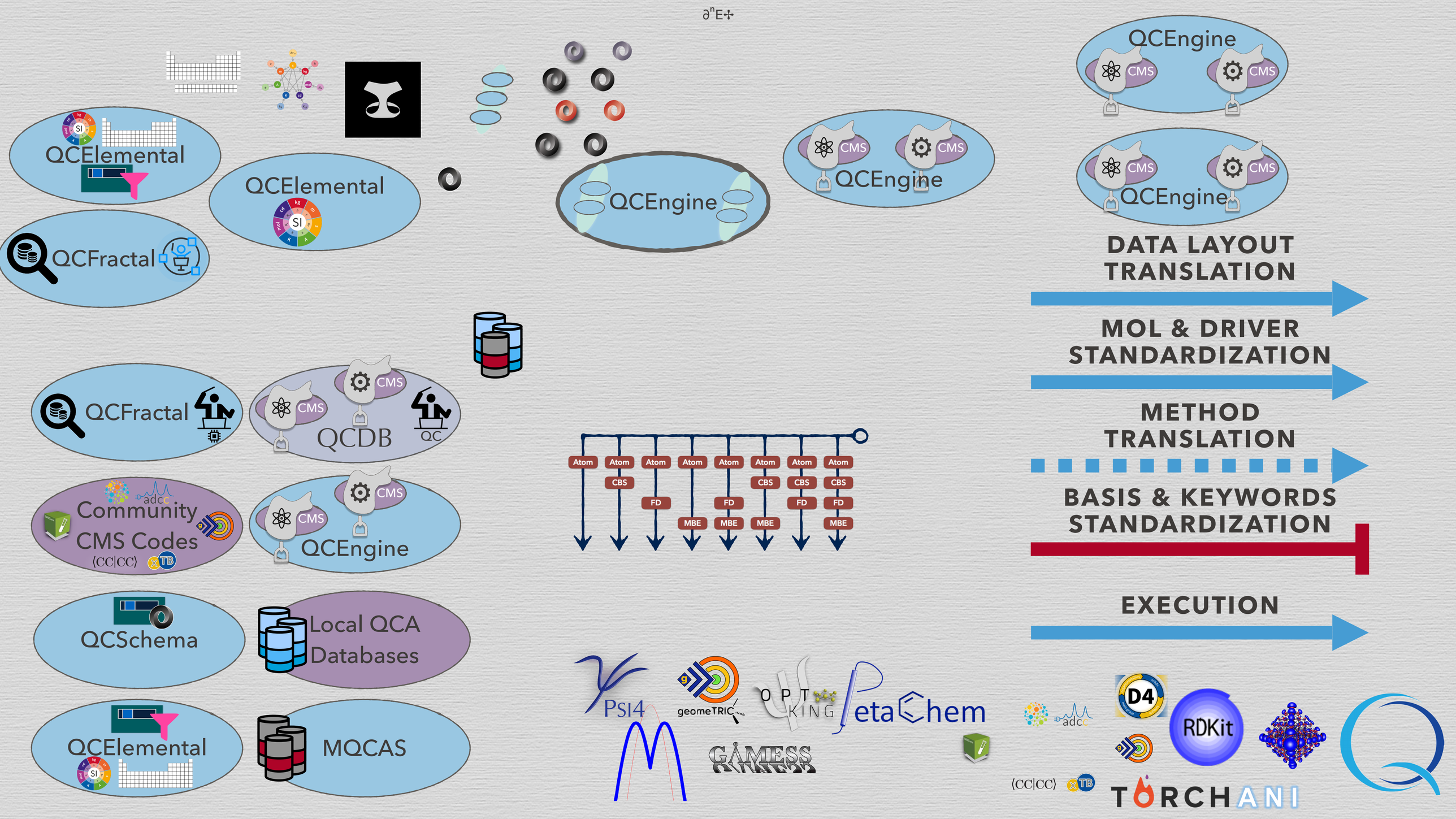
- DSL or unified input syntax
- multi-QCprogram workflows
- standardization layers

- 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

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

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


QCElemental
QCElemental
QCFractal
QCEngine
QCEngine
CMS
CMS
QCEngine
CMS
CMS
QCEngine
DATA LAYOUT TRANSLATION
MOL & DRIVER STANDARDIZATION
METHOD TRANSLATION
BASIS & KEYWORDS STANDARDIZATION
EXECUTION
QCFractal
QCDB
CMS
CMS
QC
Community CMS Codes
adcc
CMS
CMS
QCEngine
Atom
CBS
FD
MBE
QCSchema
Local QCA Databases
QCElemental
MQCAS
PSI4
geomeTRIC
OPTKING
PetaChem
GAMESS
adcc
D4
RDKit
TORCHANI