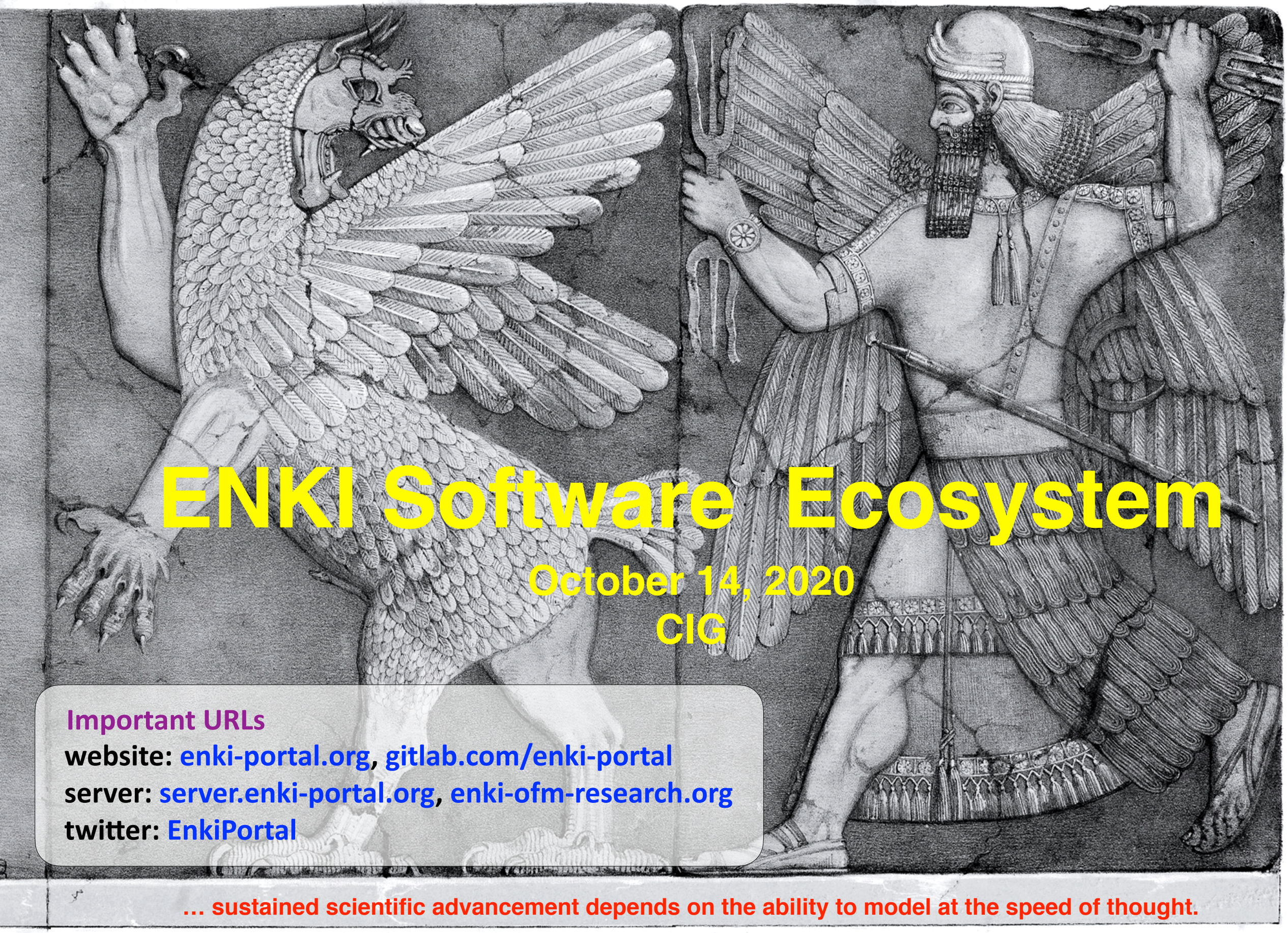


ENKI Software Ecosystem

October 14, 2020

CIG

Important URLs

website: enki-portal.org, gitlab.com/enki-portal

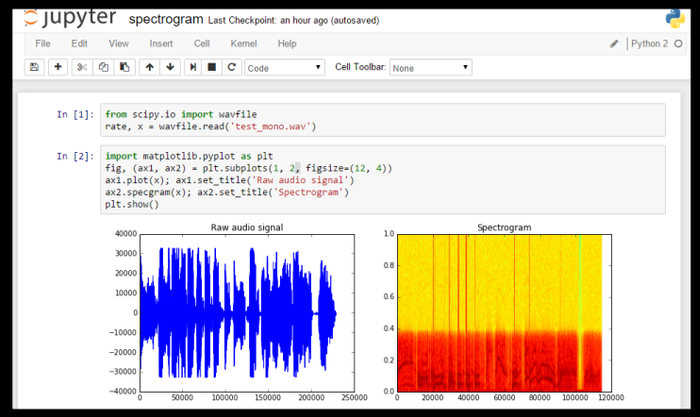
server: server.enki-portal.org, enki-ofm-research.org

twitter: EnkiPortal

… sustained scientific advancement depends on the ability to model at the speed of thought.

The ENKI software ecosystem aims to provide:

- a central platform for access to models and data - interactive Jupyter notebooks; scripting
- a consistent and standardized APIs for coded models
- consistent and standardized interfaces to underlying databases
- a mechanism for building model connections and complex model scenarios - model workflows
- a mechanism for documenting model development and model usage to address the goal of both replicable and reproducible science
- a mechanism for publication and sharing of models and calculations that are based on models or combinations of models



Model Calibration, Updating, Documentation Infrastructure

Data for model building

What modeling deficiencies does ENKI address?

- Lack of **interoperability** of existing modeling software
- Lack of infrastructure and data resources to **update** models
- Lack of frameworks for **accessing and distributing** models
- Lack of a framework for **reproducible** workflows

What science workflow deficiencies does ENKI address?

- **Excessive use of Apps** - consequences?
- **Excessive use of Excel** - consequences?

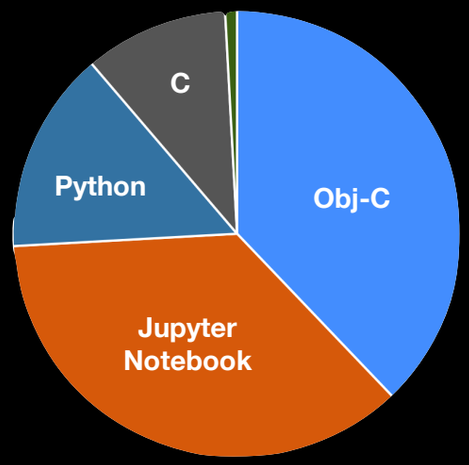
What science workflow paradigms does ENKI encourage?

- **Standardization of APIs** for software libraries
- Generation of tutorial and example interfaces (Jupyter notebooks) to those libraries that **encourage coding** to suit problem definition

ENKI provides an infrastructure to **encourage synthesis science** - *How do we know what new experiments need to be done?*

ThermoEngine package (GitLab.com)

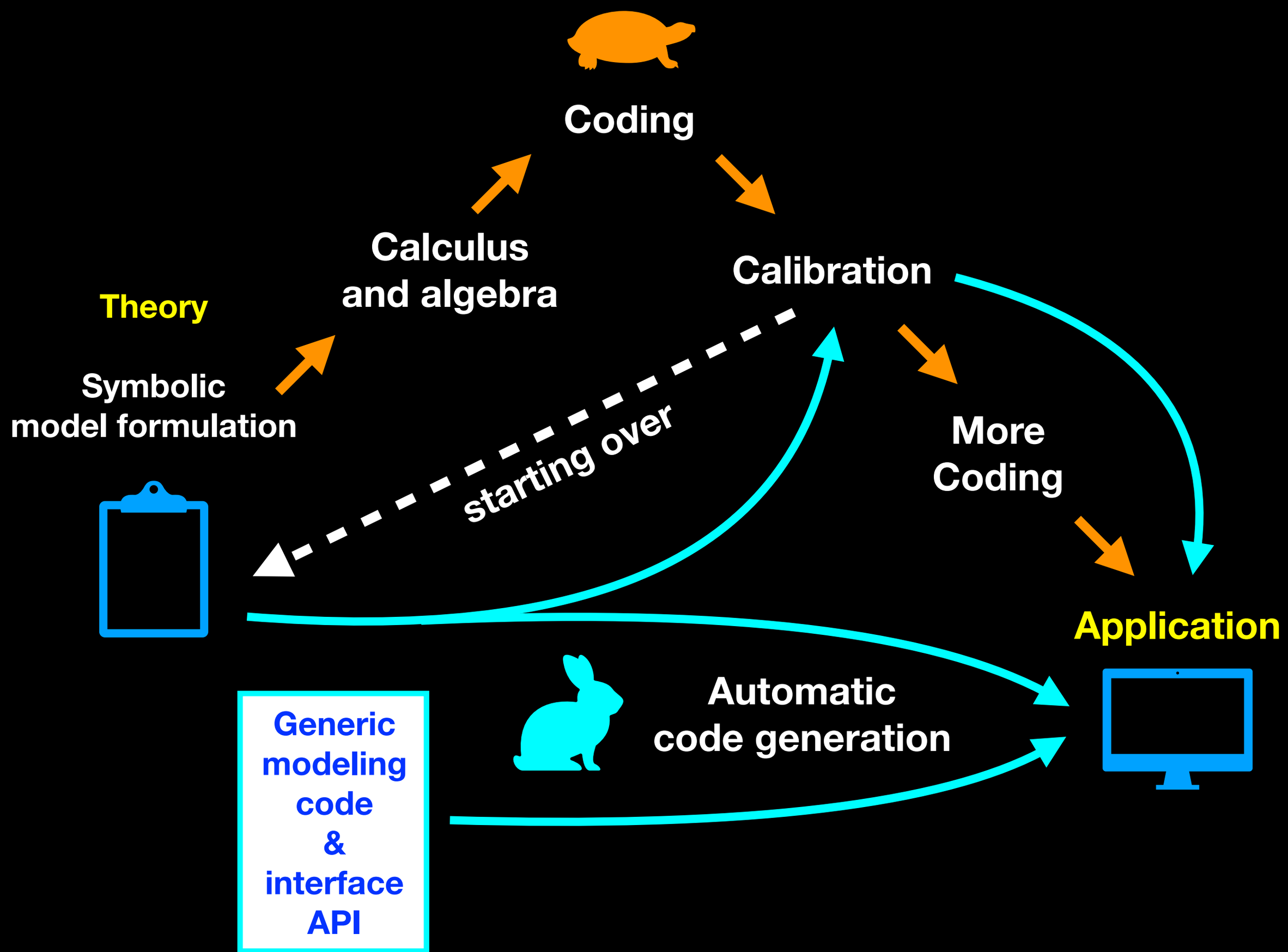
The principal ENKI API is the ThermoEngine Python package, which provides an interface to compiled libraries that implement thermodynamic property retrieval and computational thermodynamics algorithms. Modules include:



- calibrate - provides methods for supporting thermodynamic model calibration using modern Bayesian statistical methods; provides tools to interface with properties and experimental phase equilibrium databases; implements an architecture for replicable calibration and provides visualization and statistical tools for calibration assessment
- coder - provides methods for construction of thermodynamic models using symbolic mathematical expressions and for automated generation of expressions for derivative thermodynamic properties and for source code implementation; generates "C" code for inclusion in the phases module and C++ code for generation of computational libraries that support fluid dynamical modeling software
- core - provides an interface to our legacy objective C code base and implements generic compositional transformation routines
- equilibrate - provides methods that implement equilibrium calculations, including generic equilibrium calculators for Gibbs free energy, Helmholtz free energy, enthalpy and entropy minimization as well as open system calculations; MELTS; speciation models (DEW); equilibrium calculators for systems missing an omnicomponent phase; phase diagram and pseudo section generators
- graphics - provides methods for graphical display of properties and phase diagrams
- model - methods for loading legacy databases and coder generated model implementations; convenience methods for accessing reaction-based functions; integrated with the phases module
- phases - methods implementing a uniform standardized API for accessing thermodynamic properties of pure phases and solution phases; tightly integrated with the model and coder modules

Models/Databases

- Berman
- Holland & Powell
- Stixrude & Lithgow Bertelloni
- DEW - Deep Earth Water model
- HKF and extensions
- SWIM - Standard Water Integrated Model
- MELTS (1.0.2, 1.1, 1.2, pMELTS)
- MELTS+DEW


Coding
Calculus
and algebra
Calibration
Theory
Symbolic
model formulation
starting over
More
Coding
Application
Automatic
code generation
Generic
modeling
code
&
interface
API

Example: Thermodynamic Model Development

Olivine Phase Loop

1

This notebook demonstrates calculation of the olivine liquid-solid phase loop under the assumption that both phases behave as ideal solutions.
The workflow is:

- Use the **coder** module to generate endmember properties of both solutions; the only thermodynamic properties that are specified are the enthalpy and entropy of fusion
- Use the **coder** module to generate solid and liquid solution properties
- Import the generated code using the **model** module
- Use the **equilibrate** module to compute the liquid-solid phase loop
- Plot results

```
import numpy as np
import scipy as sp
import sympy as sym
import matplotlib.pyplot as plt
from thermoengine import model, equilibrate, coder
%matplotlib inline
```

Endmember properties

2

Write code into a working subdirectory

```
working_dir = "working"
!mkdir -p {working_dir}
%cd {working_dir}
```

```
/Users/ghiorso/Documents/ARCHIVE_XCODE/ThermoEngineMSG/Notebooks/Equilibrate/working
```

Model generation function

```
def make_endmembers(module='none', name='none', formula='none', Hrefvalue=0.0, Srefvalue=0.0):
    mdl = coder.StdStateModel()
    T = mdl.get_symbol_for_t()
    GPr,Href,Sref = sym.symbols('GPr Href Sref')
    GPr = Href - T*Sref
    params = [('Href', 'J', Href), ('Sref', 'J/K', Sref)]
    mdl.add_expression_to_model(GPr, params)
    mdl.set_module_name(module)
    paramValues = {'Href':Hrefvalue, 'Sref':Srefvalue, 'T_r':298.15, 'P_r':1.0}
    mdl.create_code_module(phase=name, formula=formula, params=paramValues,
                           module_type='calib', silent=True)
```

Forsterite Solid

```
make_endmembers(module='OlvSolid', name='Fo', formula='Mg(2)Si(1)O(4)', Hrefvalue=-100000.0, Srefvalue=0.0)
%cp OlvSolid.pyx endmembersolids.pyx
```

Fayalite Solid

```
make_endmembers(module='OlvSolid', name='Fa', formula='Fe(2)Si(1)O(4)', Hrefvalue=-100000.0, Srefvalue=0.0)
%cat OlvSolid.pyx >> endmembersolids.pyx
```

Forsterite Liquid

Fusion temperature is 2163 K, entropy is 57.2 J/K

```
make_endmembers(module='OlvLiquid', name='Fo', formula='Mg(2)Si(1)O(4)', Hrefvalue=-100000.0+57.2*2163.0, Srefvalue=57.2)
%cp OlvLiquid.pyx endmemberliquids.pyx
```

Fayalite Liquid

Fusion temperature is 1490 K, entropy is 59.9 J/K

```
make_endmembers(module='OlvLiquid', name='Fa', formula='Fe(2)Si(1)O(4)', Hrefvalue=-100000.0+59.9*1490.0, Srefvalue=59.9)
%cat OlvLiquid.pyx >> endmemberliquids.pyx
```

Solution Properties

3

Model generation function

```
def make_solution(module='none', name='none', endmembers=[]):
    c = 2
    mdl = coder.SimpleSolnModel(nc=c)
    n = mdl.n
    nT = mdl.nT
    X = n/nT
    T = mdl.get_symbol_for_t()
    mu = mdl.mu
    G_ss = (n.transpose()*mu)[0]
    S_config,R = sym.symbols('S_config R')
    S_config = 0
    for i in range(0,c):
        S_config += X[i]*sym.log(X[i])
    S_config *= -R*nT
    G_config = sym.simplify(-T*S_config)
    G = G_ss + G_config
    mdl.add_expression_to_model(G, [('dummy', 'none', sym.symbols('dummy'))])
    mdl.module = module
    mdl.formula_string = 'Mg[Mg]Fe[Fe]Si[Si]O4'
    mdl.conversion_string = ['[0]=[Mg]', '[1]=[Fe]']
    mdl.test_string = ['[0] > 0.0', '[1] > 0.0']
    mdl.create_code_module(phase=name, params={'dummy':0.0, 'T_r':298.15, 'P_r':1.0},
                           endmembers=endmembers,
                           prefix="cy", module_type='calib', silent=True)
```

Solid solution

```
make_solution(module='OlvSolid', name='Olivine', endmembers=['Fo_OlvSolid', 'Fa_OlvSolid'])
%cat endmembersolids.pyx >> OlvSolid.pyx
```

Liquid solution

```
make_solution(module='OlvLiquid', name='Liquid', endmembers=['Fo_OlvLiquid', 'Fa_OlvLiquid'])
%cat endmemberliquids.pyx >> OlvLiquid.pyx
```

```
import OlvSolid
import OlvLiquid
%cd ..
```

Set up phase loop calculation

4

```
modelDBsol = model.Database(database="CoderModule", calib='calib',
                            phase_tuple=('OlvSolid', {
                                'Ol':['Olivine','solution'],
                                'Fo':['Fo','pure'],
                                'Fa':['Fa','pure']
                            }))
modelDBliq = model.Database(database="CoderModule", calib='calib',
                            phase_tuple=('OlvLiquid', {
                                'Liq':['Liquid','solution'],
                                'Fo':['Fo','pure'],
                                'Fa':['Fa','pure']
                            }))
```

5

```
olivine = modelDBsol.get_phase("Ol")
liquid = modelDBliq.get_phase("Liq")
```

```
elm_sys = ['O','Mg','Si','Fe']
phs_sys = [liquid, olivine]
```

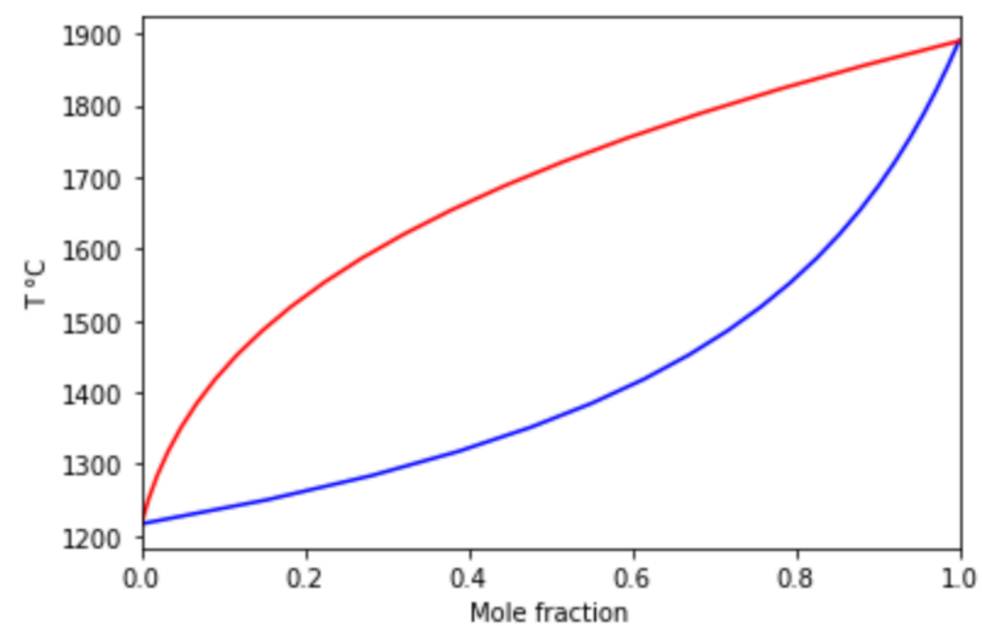
Compute the loop

```
xFoSol = [1.0]
xFoLiq = [1.0]
tC = [2163.0-273.15]
p = 1.0
for i in range(1,20):
    XFo = 1.0 - i*0.05
    XFa = 1.0 - XFo
    blk_cmp = np.array([4.0*(XFo+XFa), 2.0*XFo, XFo+XFa, 2.0*XFa])
    equil = equilibrate.Equilibrate(elm_sys, phs_sys)
    t = 2163.0*XFo + 1490.0*XFa
    state = equil.execute(t, p, bulk_comp=blk_cmp, debug=0, stats=True)
    state.print_state()
    tC.append(t-273.15)
    xFoSol.append(state.compositions(phase_name='Olivine', units='mole_frac')[0])
    xFoLiq.append(state.compositions(phase_name='Liquid', units='mole_frac')[0])
xFoSol.append(0.0)
xFoLiq.append(0.0)
tC.append(1490.0-273.15)
```

6

```
plt.plot(xFoSol, tC, 'b-')
plt.plot(xFoLiq, tC, 'r-')
plt.ylabel('T °C')
plt.xlabel('Mole fraction')
plt.xlim(0.0, 1.0)
```

```
(0.0, 1.0)
```



Cloud implementation

The ENKI software framework is open source and available under the ENKI-portal group at GitLab.com. Pre-configured Docker images that deploy a Jupyter Lab portal to the ENKI computational environment are also available at GitLab.com from the repository

ThermoEngine. Users can access a current version of ENKI supported by Jupyter Hub running within a Kubernetes cluster on Google Cloud at server.enki-portal.org. The cloud resource features persistent storage. Access to group shared cloud resources is under development.