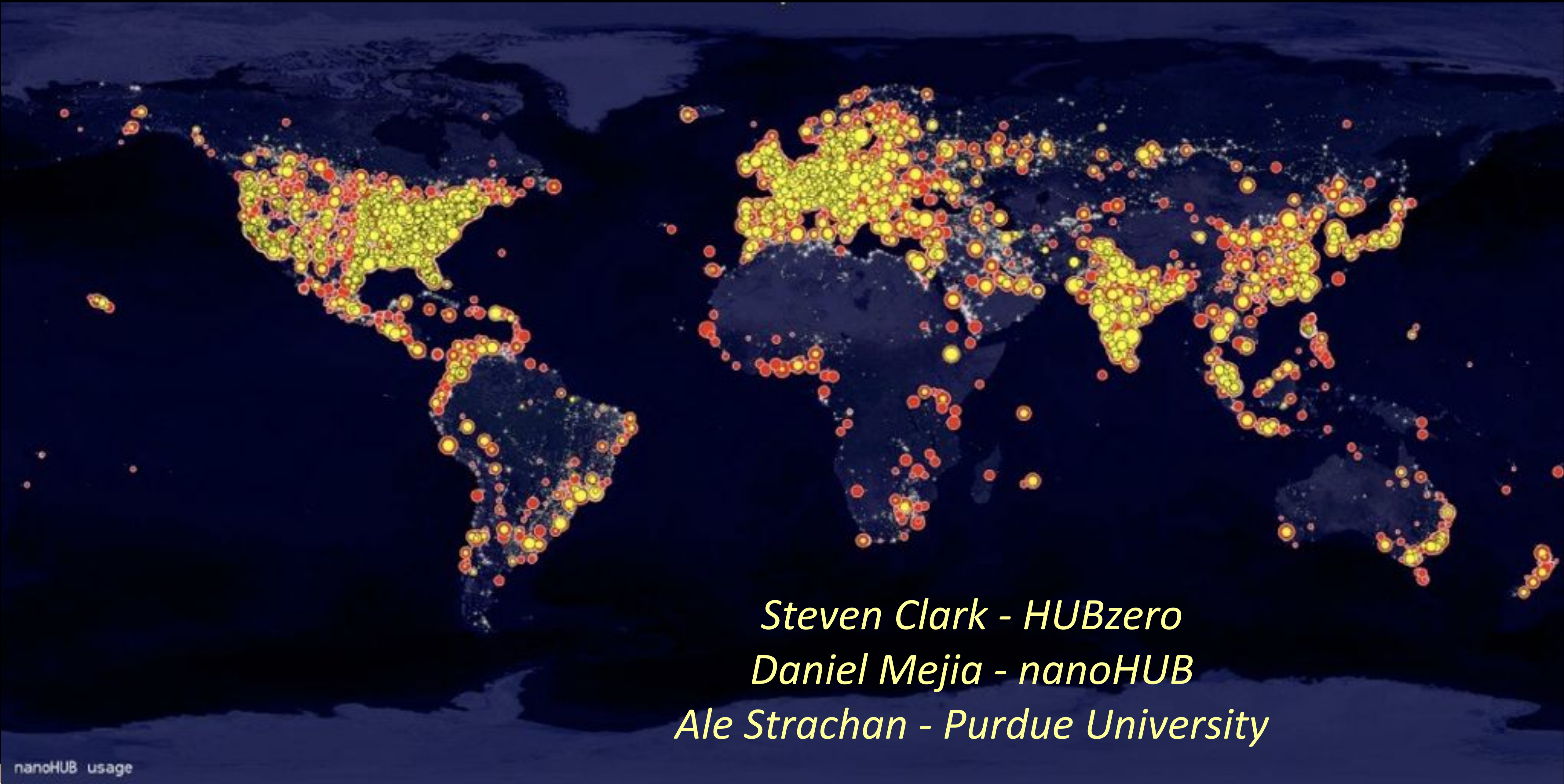
Sim2Ls: Software tools that are FAIR



Steven Clark - HUBzero
Daniel Mejia - nanoHUB
Ale Strachan - Purdue University

Gateways: accelerating innovation

via user-friendly, accessible apps, tools, and data

Science codes, data tools & repos

QUANTUM HOME PROJECT DOWNLOAD

Schred

LAMMPS

siesta

NEMO5

Polymer Modeler

Abinit

Padr

NAMD Scalable Mo

TensorFlow

Keras

OpenKIM

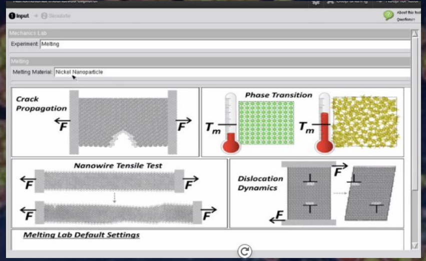
The Materials Project

CITRINE INFORMATICS

WolframAlpha computational intelligence

HUBzero services

Online Apps & Tools



Input

Mechanics Lab

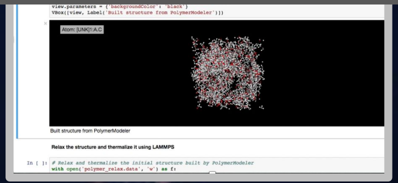
Experiment: Melting

Melting

Melting Material: Nickel Nanoparticle

Science workflows

Sim2Ls: Modern simulation & data delivery



```
view.parameters = {'backgroundColor': 'black'}
VBox([view, Label('Built structure from PolymerModeler')])
```

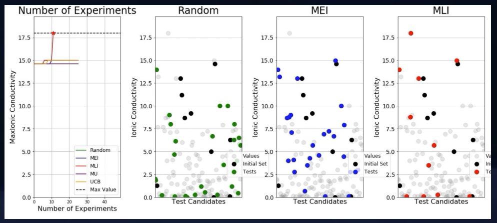
Atom: [UNK]1:A.C

Built structure from PolymerModeler

Relax the structure and thermalize it using LAMMPS

```
In [ ]: # Relax and thermalize the initial structure built by PolymerModeler
        with open('polymer_relax.data', 'w') as f:
```

Data science & machine learning



Research

NCN-affiliated documents
Non-NCN-affiliated documents
Three distinct dot sizes indicate the level of influence on h-index
Papers with relatively low secondary citations
Papers with potential to influence h-index
Papers affecting the h-index

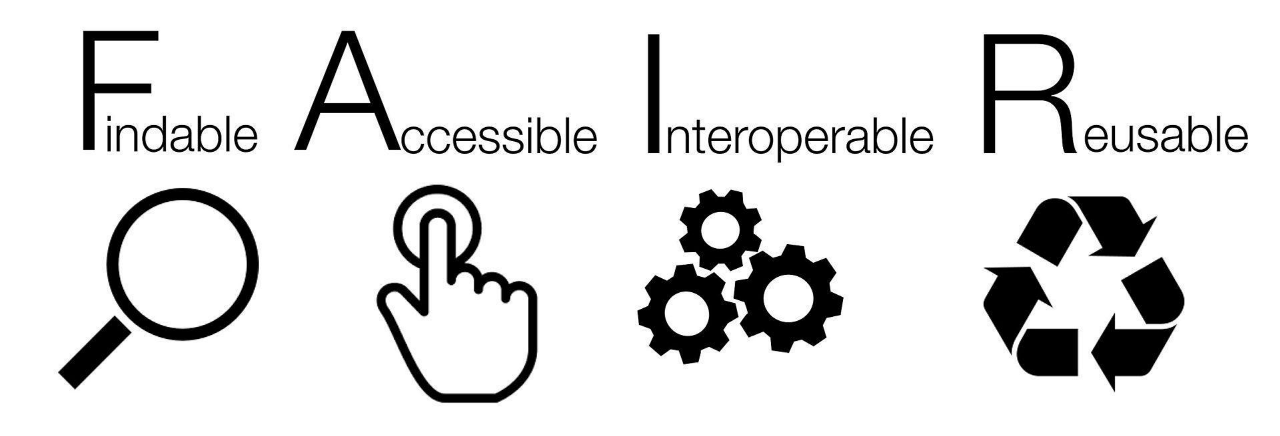
on & workforce development

Users

nanoHUB usage

Play FAIR

Simulation tools, associated workflows, and their results should adhere to FAIR principles



Pundir, S. (2016) https://commons.wikimedia.org/wiki/ File:FAIR_data_principles.jpg.

Findable

Data has unique identifier and is indexed in a searchable resource

Accessible

An access protocol that is open, free and universally implementable allows for retrieval of the data

Interoperable

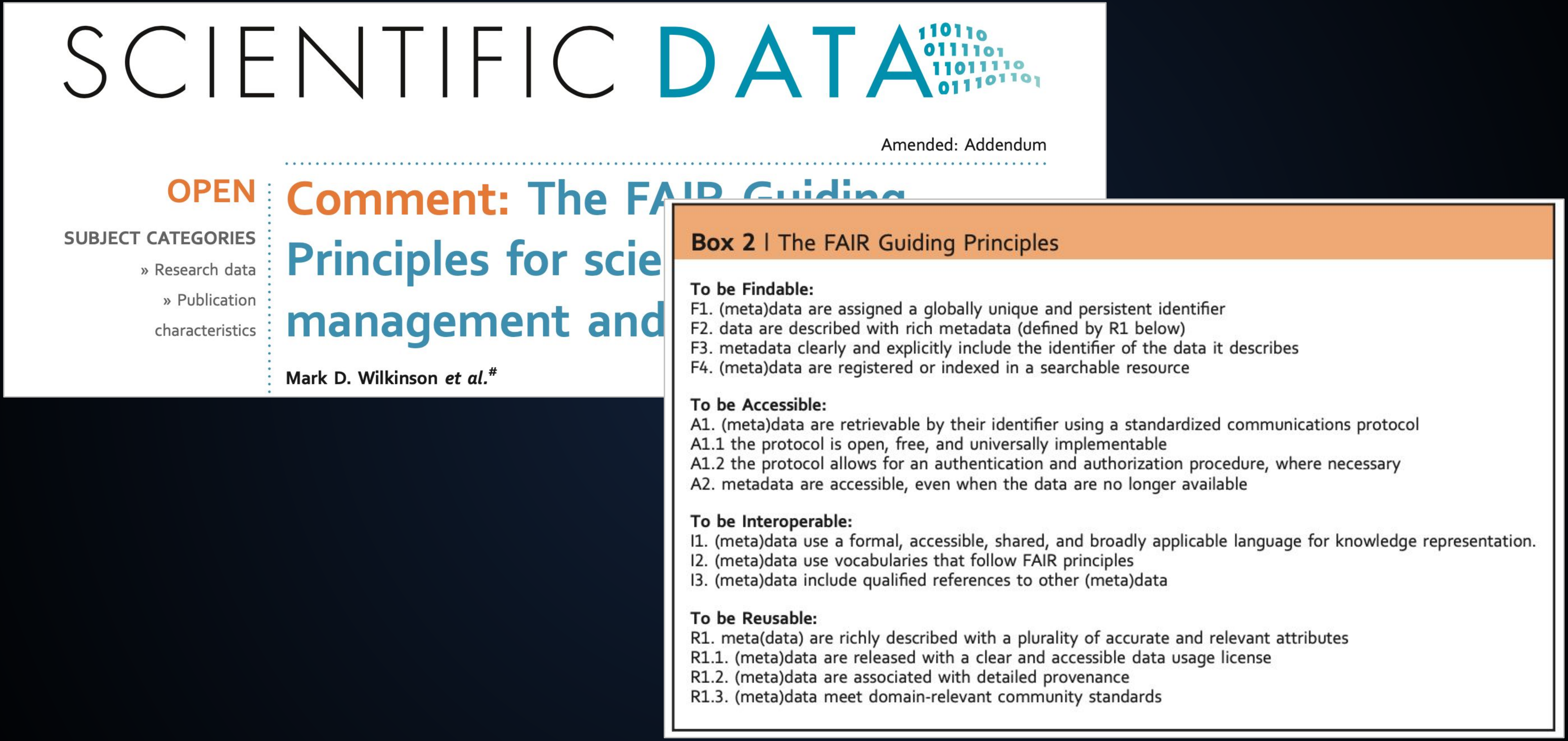
Data uses a formal, accessible and broadly applicable language for knowledge representation

Reusable

Data is richly described with metadata for provenance and to guarantee domain-relevant community standards

Wilkinson, et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Scientific data, 3(1), 1-9.

FAIR principles



SCIENTIFIC DATA

Amended: Addendum

OPEN

SUBJECT CATEGORIES

» Research data

» Publication characteristics

Comment: The FAIR Guiding Principles for scie management and

Mark D. Wilkinson *et al.*#

Box 2 | The FAIR Guiding Principles

To be Findable:
F1. (meta)data are assigned a globally unique and persistent identifier
F2. data are described with rich metadata (defined by R1 below)
F3. metadata clearly and explicitly include the identifier of the data it describes
F4. (meta)data are registered or indexed in a searchable resource

To be Accessible:
A1. (meta)data are retrievable by their identifier using a standardized communications protocol
A1.1 the protocol is open, free, and universally implementable
A1.2 the protocol allows for an authentication and authorization procedure, where necessary
A2. metadata are accessible, even when the data are no longer available

To be Interoperable:
I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
I2. (meta)data use vocabularies that follow FAIR principles
I3. (meta)data include qualified references to other (meta)data

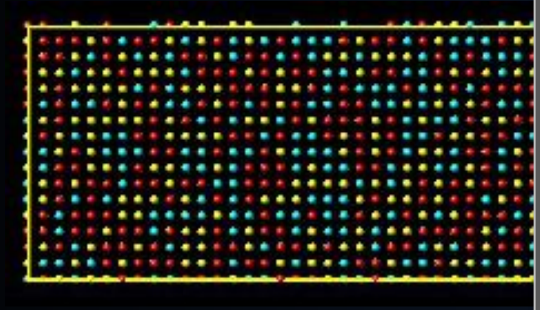
To be Reusable:
R1. meta(data) are richly described with a plurality of accurate and relevant attributes
R1.1. (meta)data are released with a clear and accessible data usage license
R1.2. (meta)data are associated with detailed provenance
R1.3. (meta)data meet domain-relevant community standards

Wilkinson, et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Scientific data, 3(1), 1-9.

The status quo: research workflows

MD to calculate the melting temperature of an alloy

1. Generate random alloy structure



Research workflows

- Are complex and involve multiple steps
- Not fully described in publications
- Often contain ad hoc scripts and manual steps

Consequently …

- Workflows are not findable, accessible, interoperable, or reusable
- The science generated is hard to reproduce

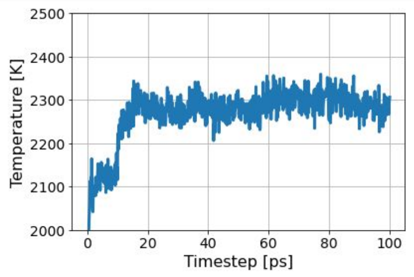
ze phases present

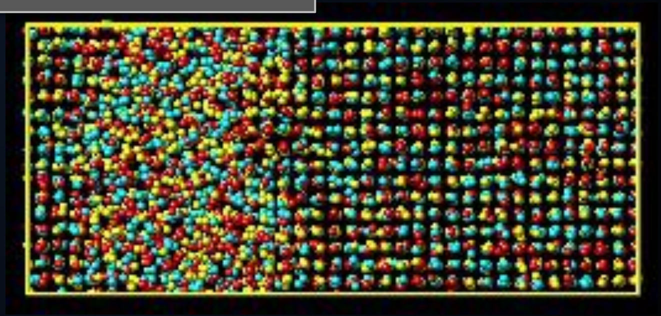
OVITO

2. Model to c
interactions

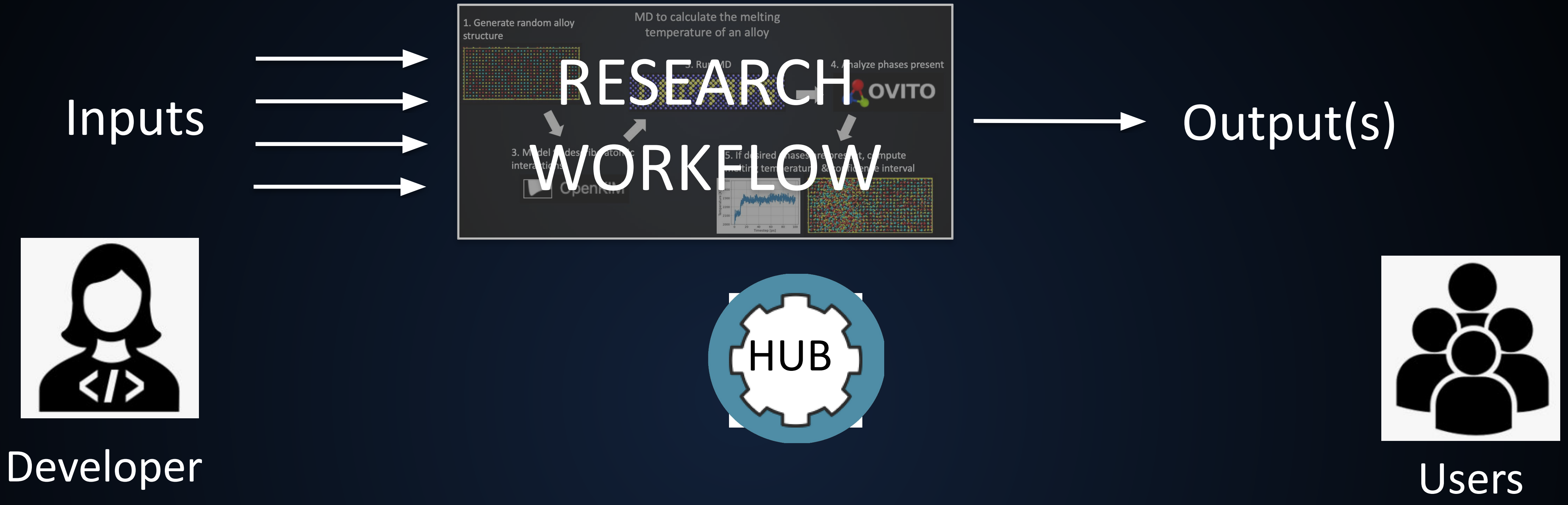
OpenKIM

compute
nce interval





Sim2Ls: making simulation workflows FAIR



Inputs

RESEARCH WORKFLOW

Output(s)

Developer

HUB

Users

1. Generate random alloy structure
MD to calculate the melting temperature of an alloy
2. Run MD
4. Analyze phases present
OVITO
3. Model to describe atomistic interactions
OpenKIM
5. If desired phases are present, compute melting temperature & confidence interval
Temperature [K]
Timestep [ps]

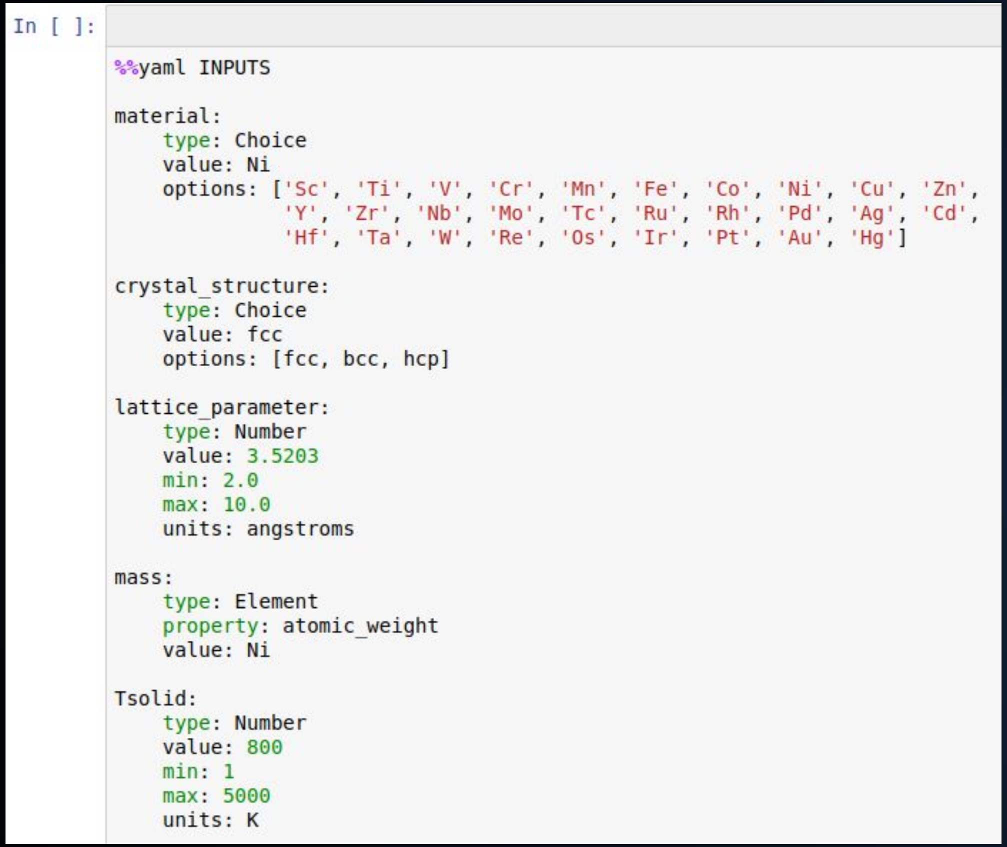
Sim2L:

- Using Jupyter notebooks
- Declare inputs & outputs (including metadata)
- Implement every step connecting INs to OUTs

Sim2L library

- Sim2Ls are registered & queryable
- Inputs and outputs (incl. metadata) are queryable
- Inputs are verified (incl. unit conversion) before run
- Outputs are checked after workflow execution
- Published Sim2Ls have DOIs & are indexed by WoS and google scholar
- Results are automatically cached & queryable (data & metadata)

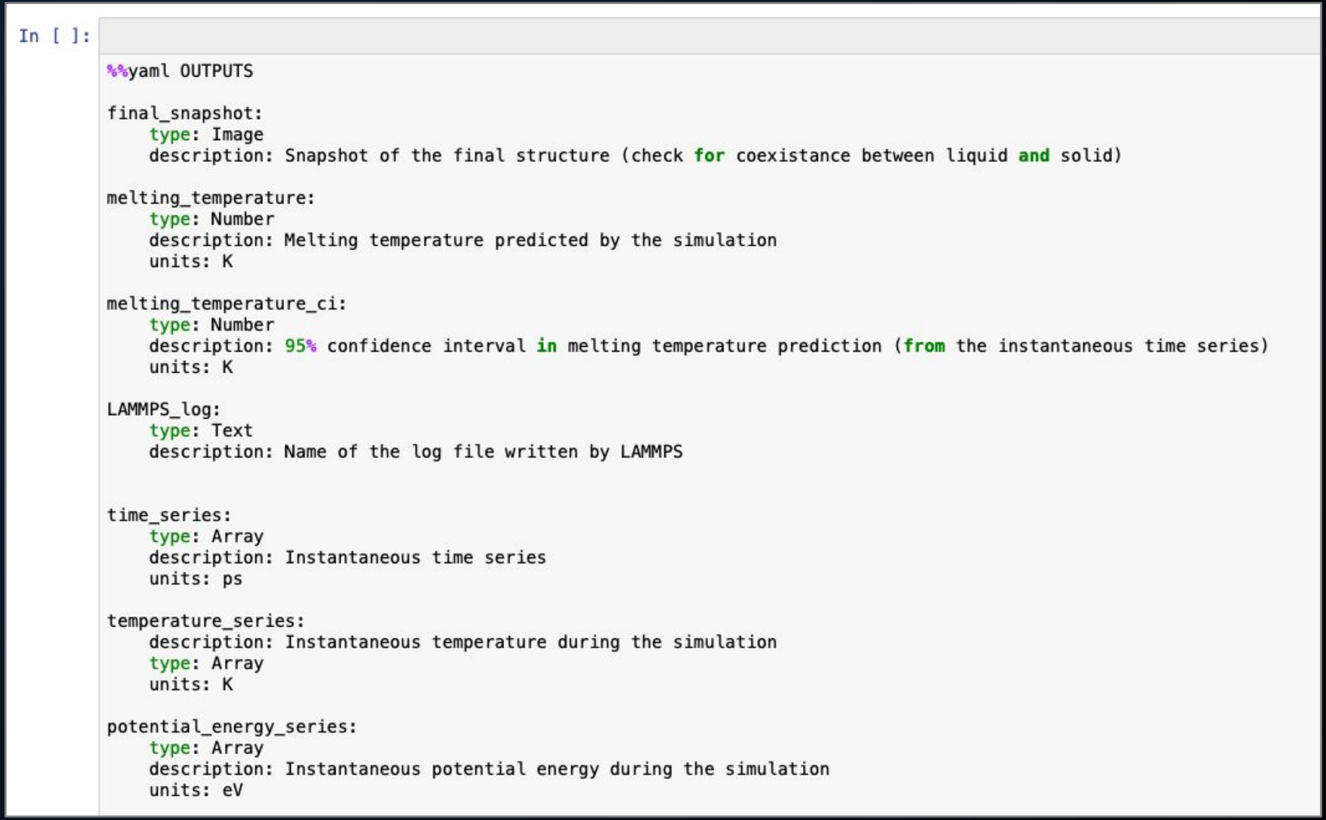
Developing a Sim2L



In [ ]:

```
%%yaml INPUTS

material:
    type: Choice
    value: Ni
    options: ['Sc', 'Ti', 'V', 'Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn',
              'Y', 'Zr', 'Nb', 'Mo', 'Tc', 'Ru', 'Rh', 'Pd', 'Ag', 'Cd',
              'Hf', 'Ta', 'W', 'Re', 'Os', 'Ir', 'Pt', 'Au', 'Hg']

crystal_structure:
    type: Choice
    value: fcc
    options: [fcc, bcc, hcp]

lattice_parameter:
    type: Number
    value: 3.5203
    min: 2.0
    max: 10.0
    units: angstroms

mass:
    type: Element
    property: atomic_weight
    value: Ni

Tsolid:
    type: Number
    value: 800
    min: 1
    max: 5000
    units: K
```

1. Declare inputs and outputs using YAML
    - Note units, descriptions, ranges

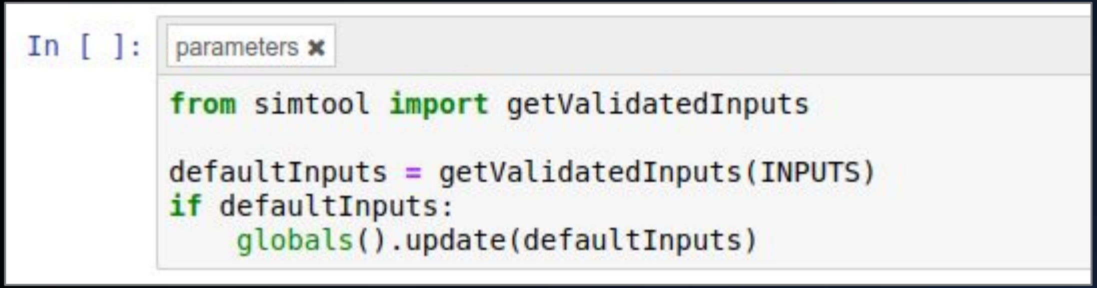


In [ ]:

```
%%yaml OUTPUTS

final_snapshot:
    type: Image
    description: Snapshot of the final structure (check for coexistance between liquid and solid)

melting_temperature:
    type: Number
    description: Melting temperature predicted by the simulation
    units: K

melting_temperature_ci:
    type: Number
    description: 95% confidence interval in melting temperature prediction (from the instantaneous time series)
    units: K

LAMMPS_log:
    type: Text
    description: Name of the log file written by LAMMPS


time_series:
    type: Array
    description: Instantaneous time series
    units: ps

temperature_series:
    description: Instantaneous temperature during the simulation
    type: Array
    units: K

potential_energy_series:
    type: Array
    description: Instantaneous potential energy during the simulation
    units: eV
```

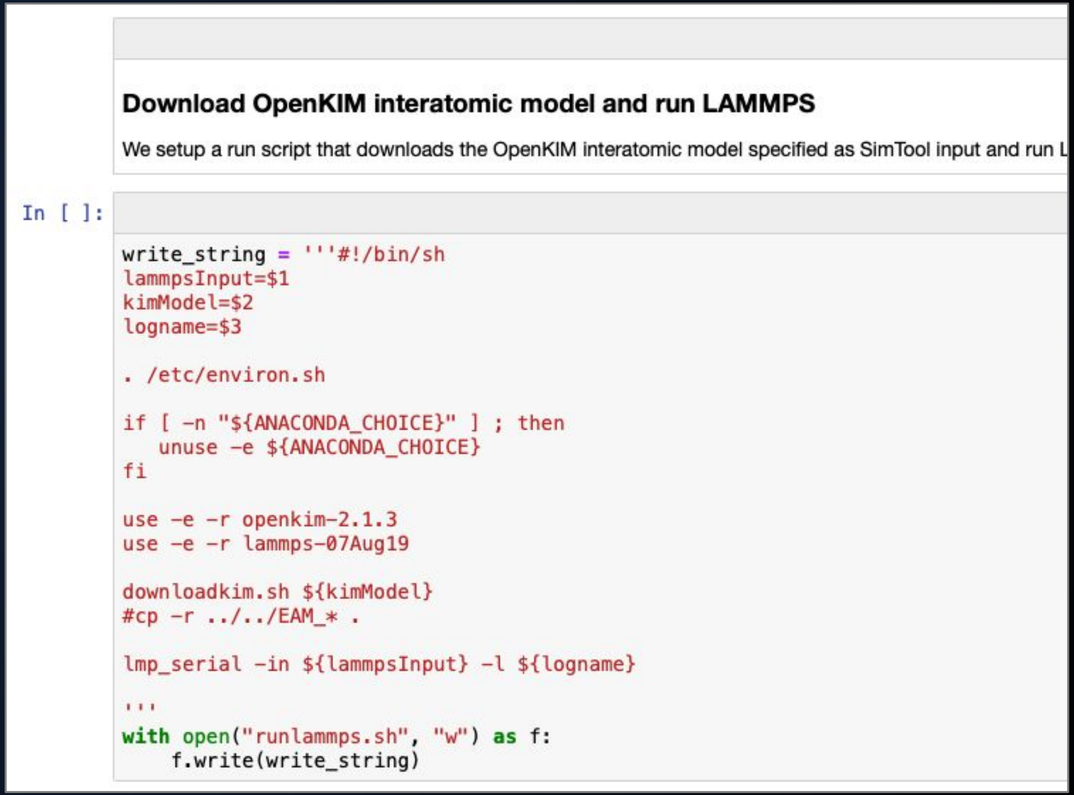
Developing a Sim2L

2. Parameterization (using papermill)



```
from simtool import getValidatedInputs

defaultInputs = getValidatedInputs(INPUTS)
if defaultInputs:
    globals().update(defaultInputs)
```

3. Setup simulation(s) & execute

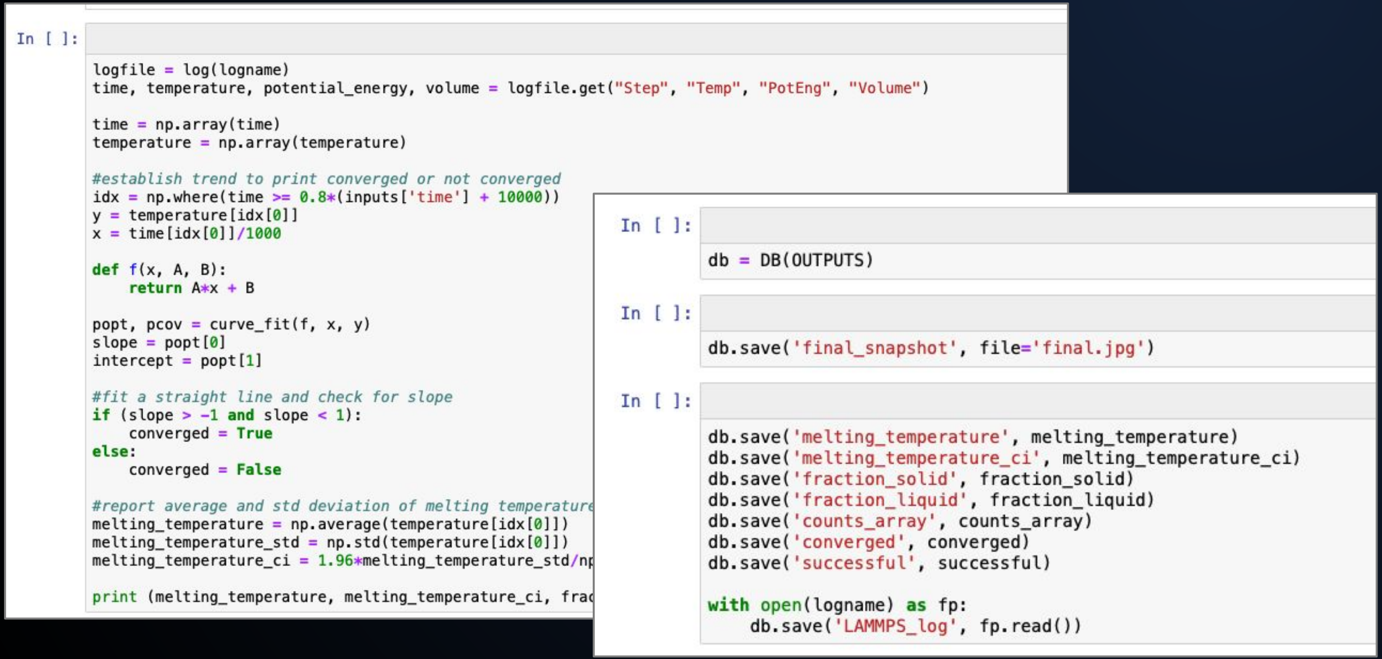


Download OpenKIM interatomic model and run LAMMPS

We setup a run script that downloads the OpenKIM interatomic model specified as SimTool input and run L

```
write_string = '''#!/bin/sh
lammpsInput=$1
kimModel=$2
logname=$3

. /etc/environ.sh

if [ -n "${ANACONDA_CHOICE}" ] ; then
   unuse -e ${ANACONDA_CHOICE}
fi

use -e -r openkim-2.1.3
use -e -r lammps-07Aug19

downloadkim.sh ${kimModel}
#cp -r ../../EAM_* .

lmp_serial -in ${lammpsInput} -l ${logname}

'''
with open("runlammps.sh", "w") as f:
    f.write(write_string)
```

4. Postprocessing and save results



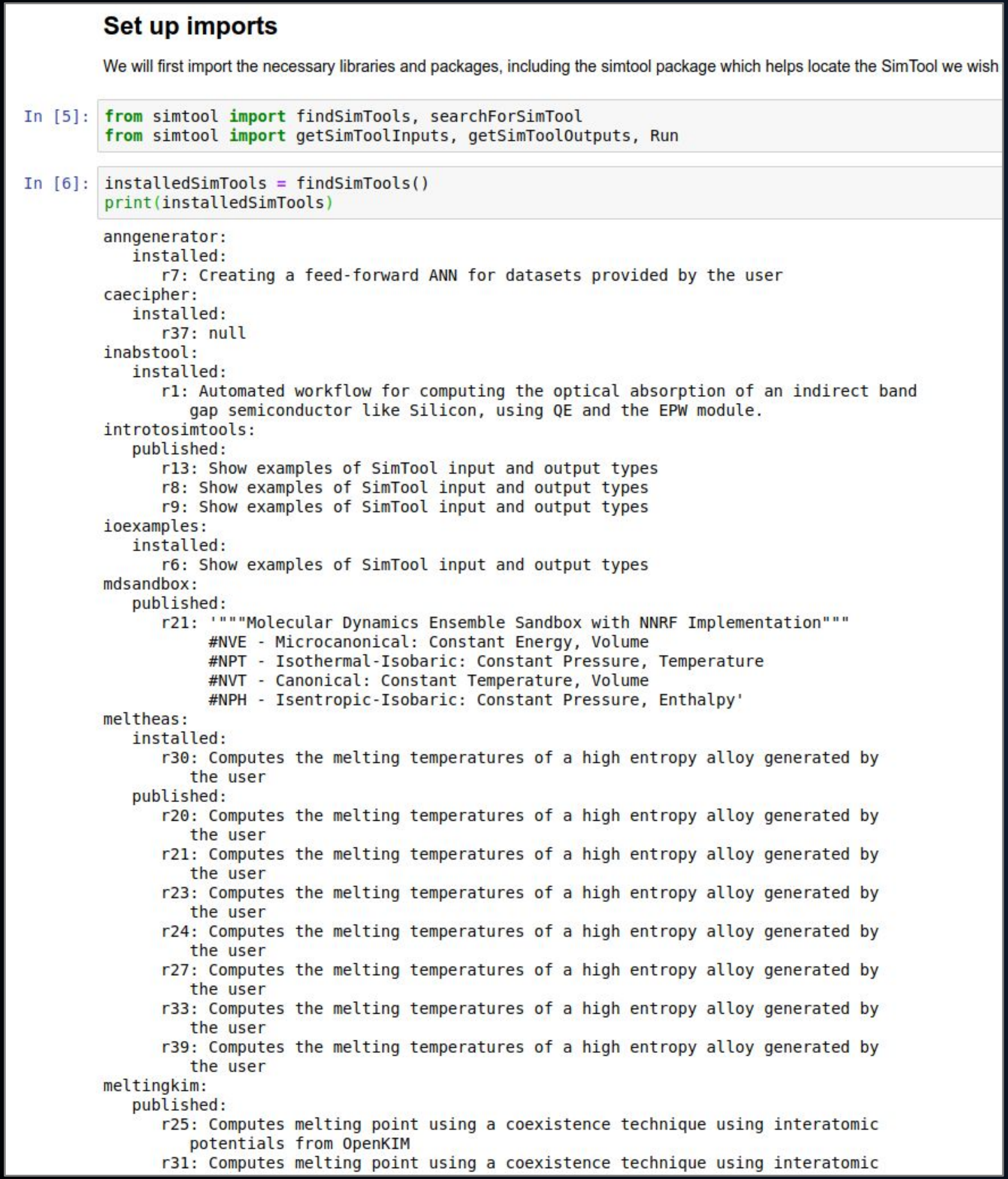
```
logfile = log(logname)
time, temperature, potential_energy, volume = logfile.get("Step", "Temp", "PotEng", "Volume")

time = np.array(time)
temperature = np.array(temperature)

#establish trend to print converged or not converged
idx = np.where(time >= 0.8*(inputs['time'] + 10000))
y = temperature[idx[0]]
x = time[idx[0]]/1000

def f(x, A, B):
    return A*x + B

popt, pcov = curve_fit(f, x, y)
slope = popt[0]
intercept = popt[1]

#fit a straight line and check for slope
if (slope > -1 and slope < 1):
    converged = True
else:
    converged = False

#report average and std deviation of melting temperature
melting_temperature = np.average(temperature[idx[0]])
melting_temperature_std = np.std(temperature[idx[0]])
melting_temperature_ci = 1.96*melting_temperature_std/np

print (melting_temperature, melting_temperature_ci, frac
```

```
db = DB(OUTPUTS)
```

```
db.save('final_snapshot', file='final.jpg')
```

```
db.save('melting_temperature', melting_temperature)
db.save('melting_temperature_ci', melting_temperature_ci)
db.save('fraction_solid', fraction_solid)
db.save('fraction_liquid', fraction_liquid)
db.save('counts_array', counts_array)
db.save('converged', converged)
db.save('successful', successful)

with open(logname) as fp:
    db.save('LAMMPS_log', fp.read())
```

Find a Sim2L for your problem



Set up imports

We will first import the necessary libraries and packages, including the simtool package which helps locate the SimTool we wish

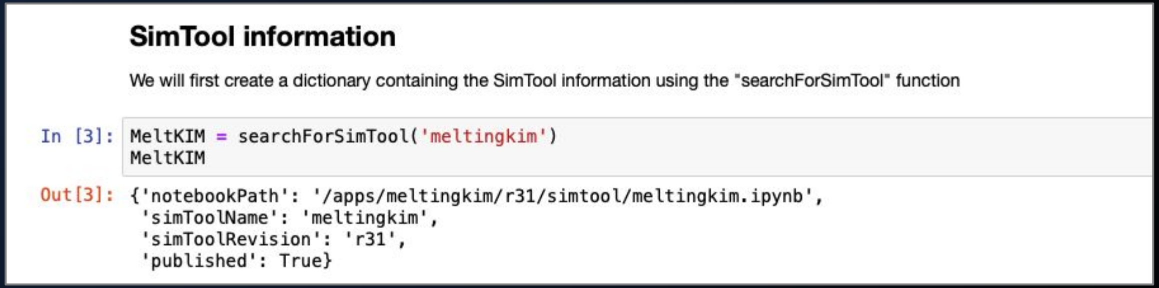
```
In [5]: from simtool import findSimTools, searchForSimTool
        from simtool import getSimToolInputs, getSimToolOutputs, Run
```

```
In [6]: installedSimTools = findSimTools()
        print(installedSimTools)
```

```
anngenerator:
   installed:
      r7: Creating a feed-forward ANN for datasets provided by the user
caecipher:
   installed:
      r37: null
inabstool:
   installed:
      r1: Automated workflow for computing the optical absorption of an indirect band
         gap semiconductor like Silicon, using QE and the EPW module.
introtosimtools:
   published:
      r13: Show examples of SimTool input and output types
      r8: Show examples of SimTool input and output types
      r9: Show examples of SimTool input and output types
ioexamples:
   installed:
      r6: Show examples of SimTool input and output types
mdsandbox:
   published:
      r21: '"""Molecular Dynamics Ensemble Sandbox with NNRF Implementation"""
           #NVE - Microcanonical: Constant Energy, Volume
           #NPT - Isothermal-Isobaric: Constant Pressure, Temperature
           #NVT - Canonical: Constant Temperature, Volume
           #NPH - Isentropic-Isobaric: Constant Pressure, Enthalpy'
meltheas:
   installed:
      r30: Computes the melting temperatures of a high entropy alloy generated by
         the user
   published:
      r20: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r21: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r23: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r24: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r27: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r33: Computes the melting temperatures of a high entropy alloy generated by
         the user
      r39: Computes the melting temperatures of a high entropy alloy generated by
         the user
meltingkim:
   published:
      r25: Computes melting point using a coexistence technique using interatomic
         potentials from OpenKIM
      r31: Computes melting point using a coexistence technique using interatomic
```

1. Explore available Sim2Ls

2. Load Sim2L of interest

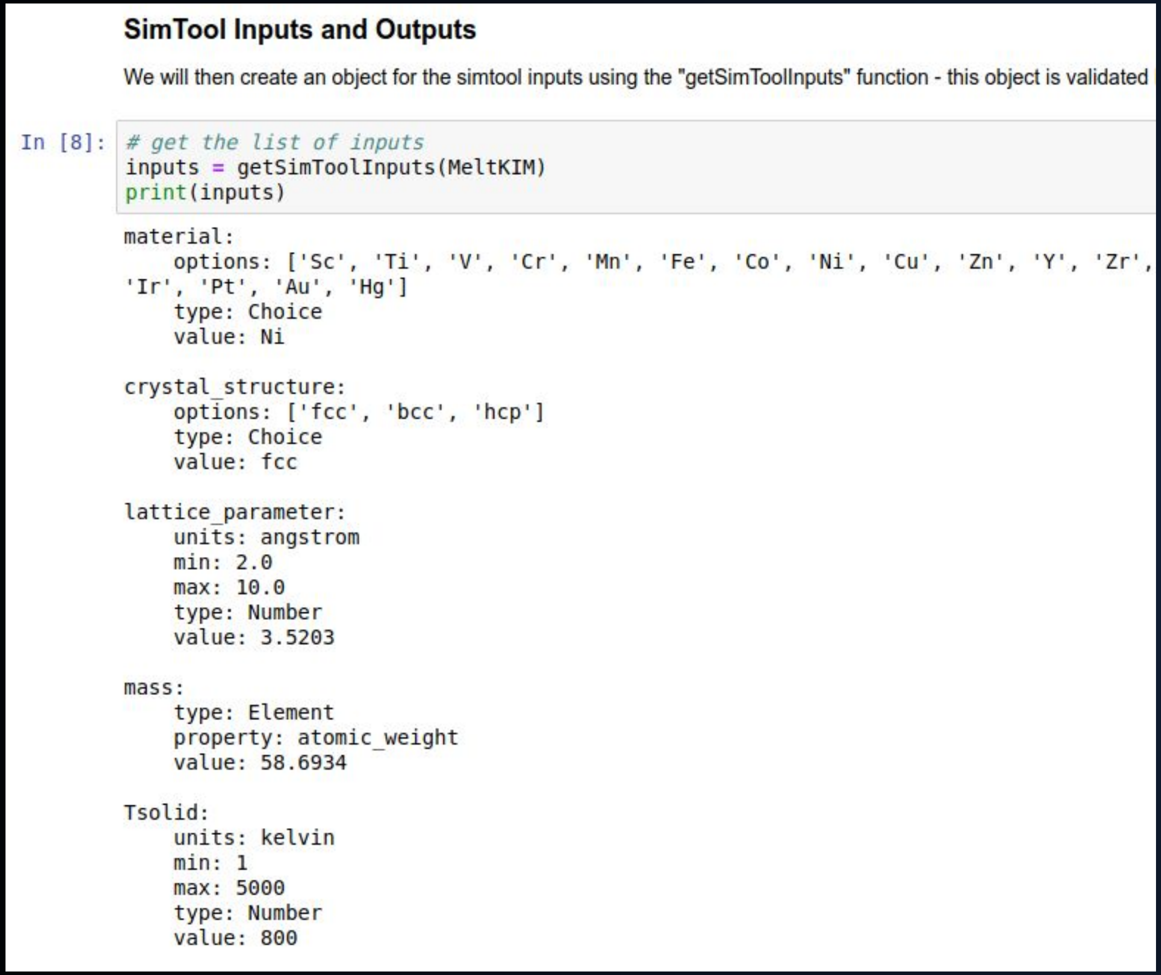


SimTool information

We will first create a dictionary containing the SimTool information using the "searchForSimTool" function

```
In [3]: MeltKIM = searchForSimTool('meltingkim')
        MeltKIM

Out[3]: {'notebookPath': '/apps/meltingkim/r31/simtool/meltingkim.ipynb',
         'simToolName': 'meltingkim',
         'simToolRevision': 'r31',
         'published': True}
```

Learn Sim2L requirements & services



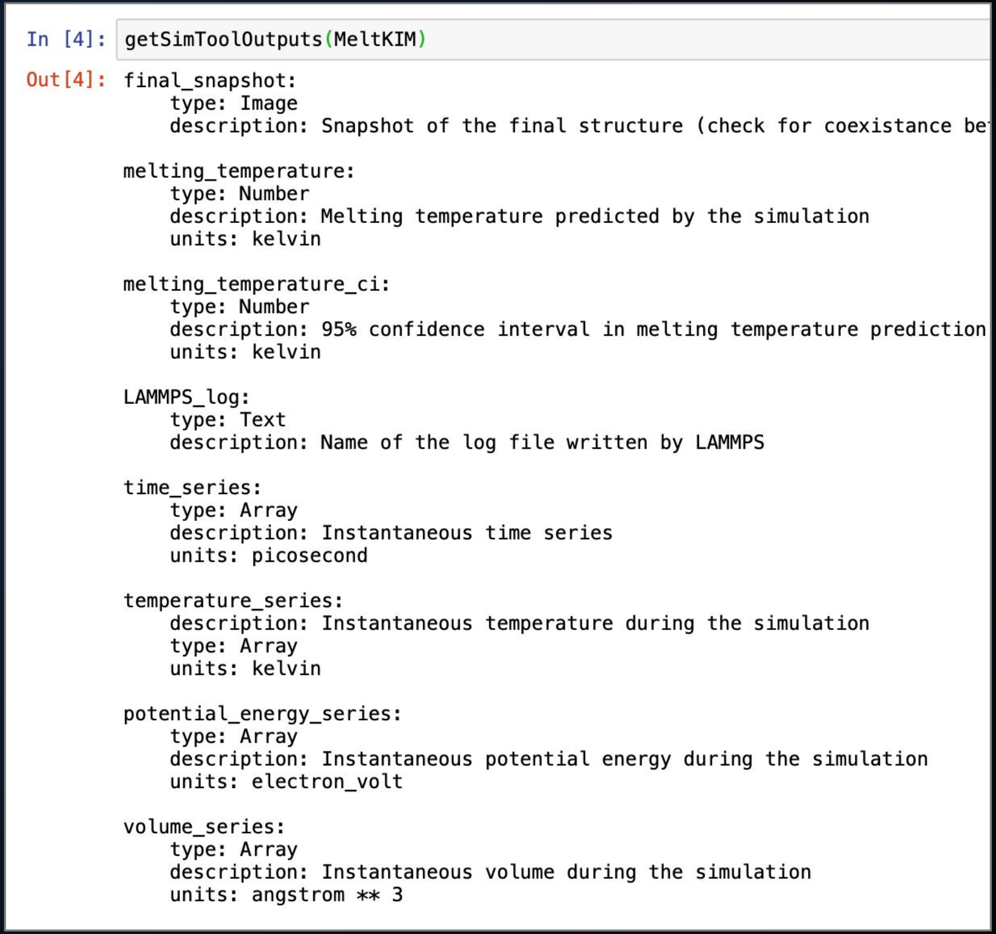
SimTool Inputs and Outputs

We will then create an object for the simtool inputs using the "getSimToolInputs" function - this object is validated

```
In [8]: # get the list of inputs
        inputs = getSimToolInputs(MeltKIM)
        print(inputs)

        material:
            options: ['Sc', 'Ti', 'V', 'Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn', 'Y', 'Zr',
        'Ir', 'Pt', 'Au', 'Hg']
            type: Choice
            value: Ni

        crystal_structure:
            options: ['fcc', 'bcc', 'hcp']
            type: Choice
            value: fcc

        lattice_parameter:
            units: angstrom
            min: 2.0
            max: 10.0
            type: Number
            value: 3.5203

        mass:
            type: Element
            property: atomic_weight
            value: 58.6934

        Tsolid:
            units: kelvin
            min: 1
            max: 5000
            type: Number
            value: 800
```

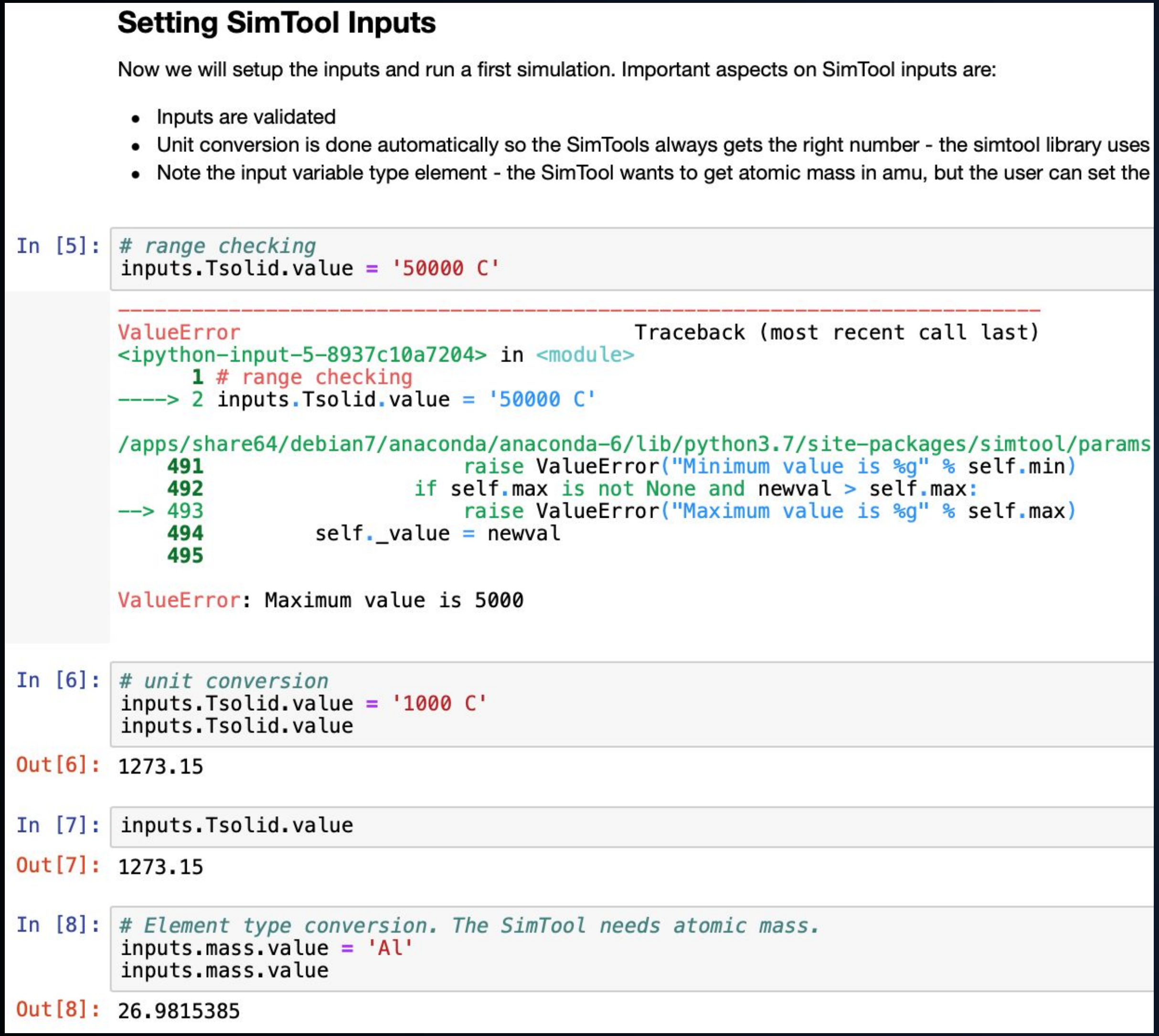
3. Explore inputs & outputs

- Note units and types



```
In [4]: getSimToolOutputs(MeltKIM)

Out[4]: final_snapshot:
            type: Image
            description: Snapshot of the final structure (check for coexistance be

        melting_temperature:
            type: Number
            description: Melting temperature predicted by the simulation
            units: kelvin

        melting_temperature_ci:
            type: Number
            description: 95% confidence interval in melting temperature prediction
            units: kelvin

        LAMMPS_log:
            type: Text
            description: Name of the log file written by LAMMPS

        time_series:
            type: Array
            description: Instantaneous time series
            units: picosecond

        temperature_series:
            description: Instantaneous temperature during the simulation
            type: Array
            units: kelvin

        potential_energy_series:
            type: Array
            description: Instantaneous potential energy during the simulation
            units: electron_volt

        volume_series:
            type: Array
            description: Instantaneous volume during the simulation
            units: angstrom ** 3
```

Parameterize a Sim2L



Setting SimTool Inputs

Now we will setup the inputs and run a first simulation. Important aspects on SimTool inputs are:

- Inputs are validated
- Unit conversion is done automatically so the SimTools always gets the right number - the simtool library uses
- Note the input variable type element - the SimTool wants to get atomic mass in amu, but the user can set the

```
In [5]: # range checking
        inputs.Tsolid.value = '50000 C'
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-8937c10a7204> in <module>
      1 # range checking
----> 2 inputs.Tsolid.value = '50000 C'

/apps/share64/debian7/anaconda/anaconda-6/lib/python3.7/site-packages/simtool/params
    491                     raise ValueError("Minimum value is %g" % self.min)
    492                 if self.max is not None and newval > self.max:
--> 493                     raise ValueError("Maximum value is %g" % self.max)
    494         self._value = newval
    495 

ValueError: Maximum value is 5000
```

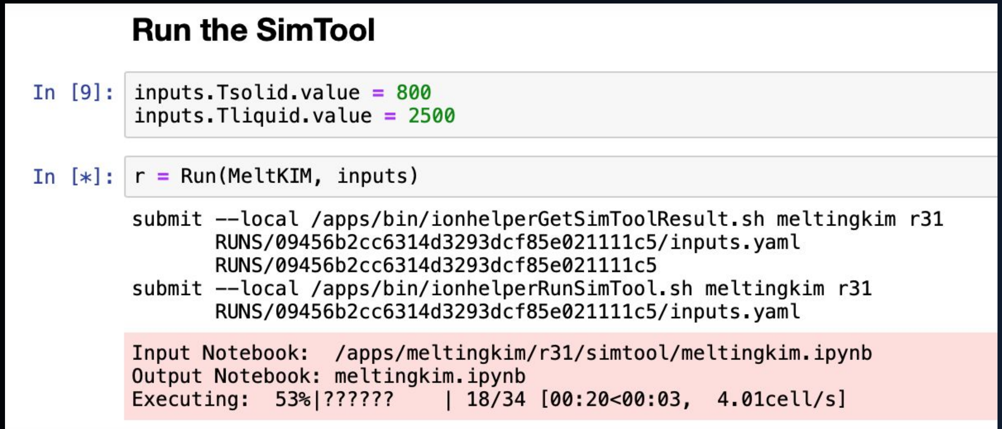
```
In [6]: # unit conversion
        inputs.Tsolid.value = '1000 C'
        inputs.Tsolid.value
Out[6]: 1273.15
```

```
In [7]: inputs.Tsolid.value
Out[7]: 1273.15
```

```
In [8]: # Element type conversion. The SimTool needs atomic mass.
        inputs.mass.value = 'Al'
        inputs.mass.value
Out[8]: 26.9815385
```

4. Parameterize input object
- Unit conversion with pint
- Element type with mendeleev

Run a Sim2L & view results

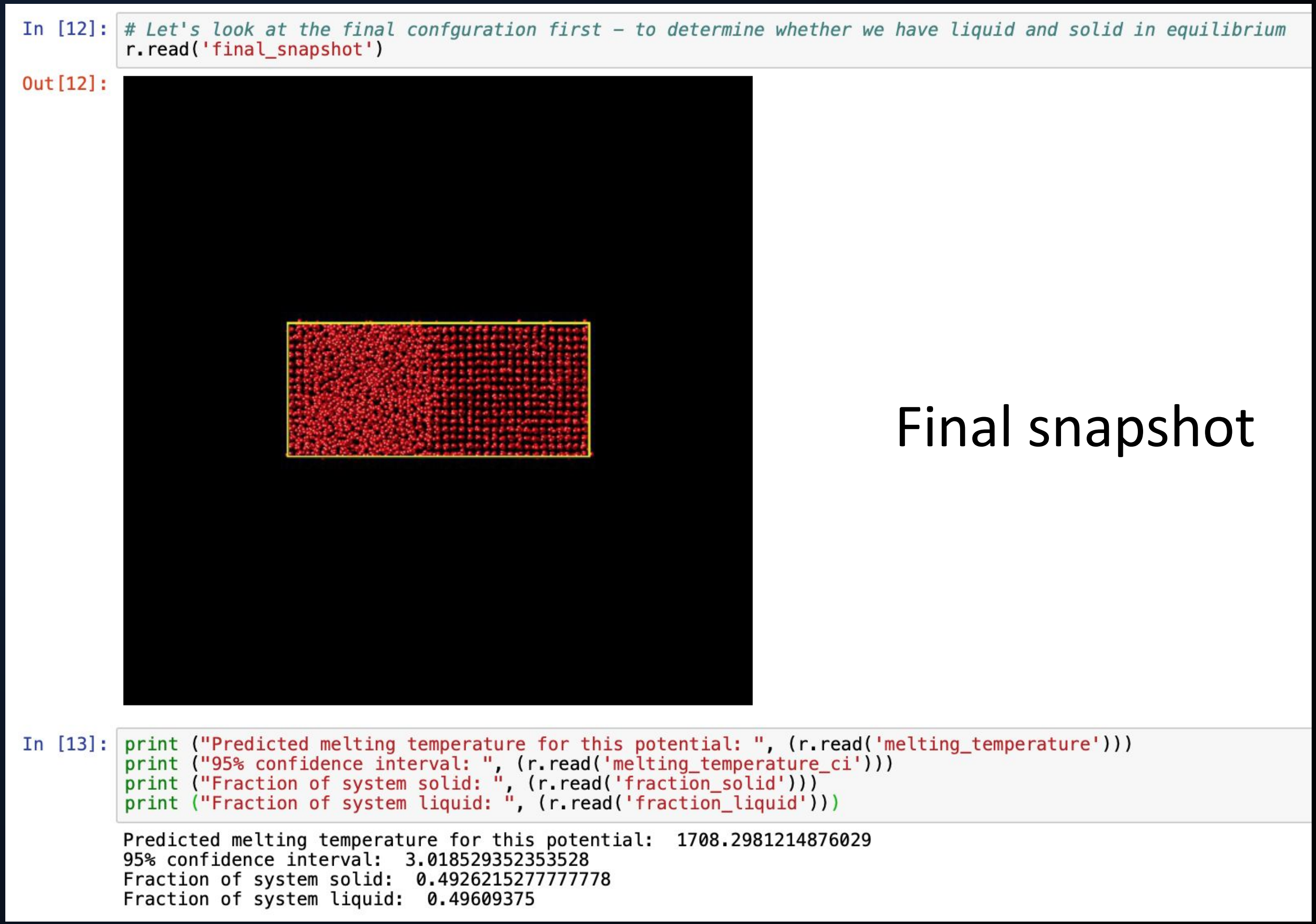


Run the SimTool

```
In [9]:  inputs.Tsolid.value = 800
         inputs.Tliquid.value = 2500

In [*]:  r = Run(MeltKIM, inputs)

         submit --local /apps/bin/ionhelperGetSimToolResult.sh meltingkim r31
                RUNS/09456b2cc6314d3293dcf85e021111c5/inputs.yaml
                RUNS/09456b2cc6314d3293dcf85e021111c5
         submit --local /apps/bin/ionhelperRunSimTool.sh meltingkim r31
                RUNS/09456b2cc6314d3293dcf85e021111c5/inputs.yaml

         Input Notebook:  /apps/meltingkim/r31/simtool/meltingkim.ipynb
         Output Notebook: meltingkim.ipynb
         Executing:  53%|??????    | 18/34 [00:20<00:03,  4.01cell/s]
```

5. Run the Sim2L

6. Explore results



```
In [12]:  # Let's look at the final confguration first - to determine whether we have liquid and solid in equilibrium
          r.read('final_snapshot')
Out[12]:
```

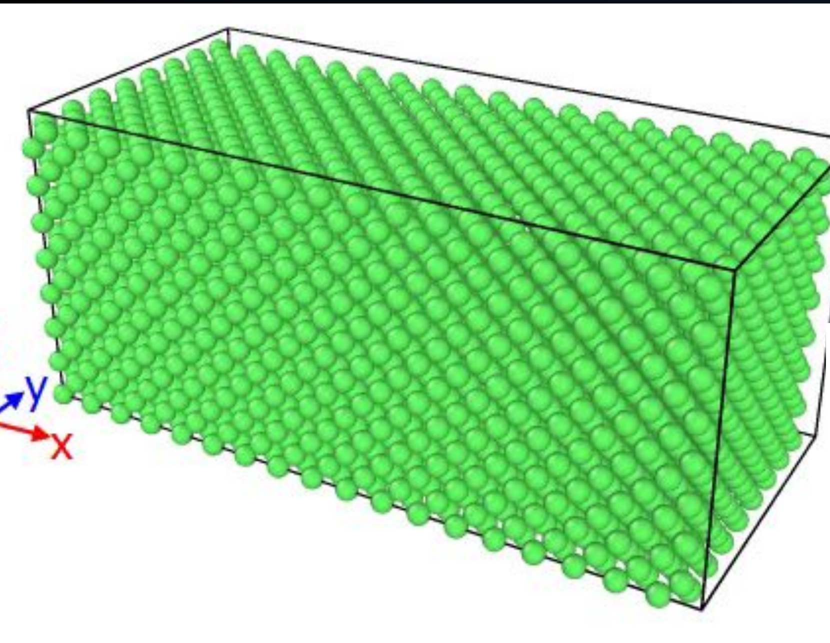
```
In [13]:  print ("Predicted melting temperature for this potential: ", (r.read('melting_temperature')))
          print ("95% confidence interval: ", (r.read('melting_temperature_ci')))
          print ("Fraction of system solid: ", (r.read('fraction_solid')))
          print ("Fraction of system liquid: ", (r.read('fraction_liquid')))

          Predicted melting temperature for this potential:  1708.2981214876029
          95% confidence interval:  3.0185293523535 28
          Fraction of system solid:  0.49262152777777778
          Fraction of system liquid:  0.49609375
```

Final snapshot

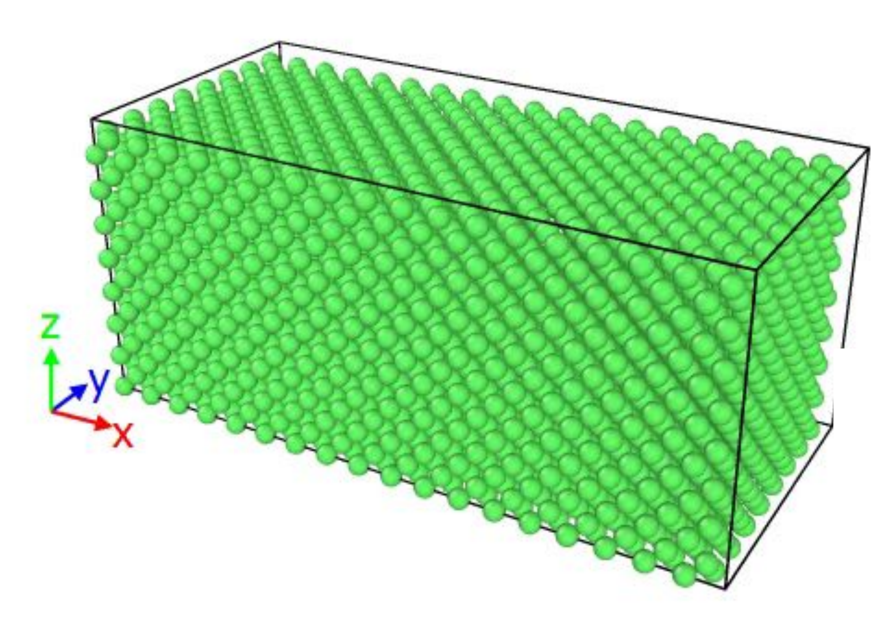
Predicted values

https://nanohub.org/tools/meltingkim

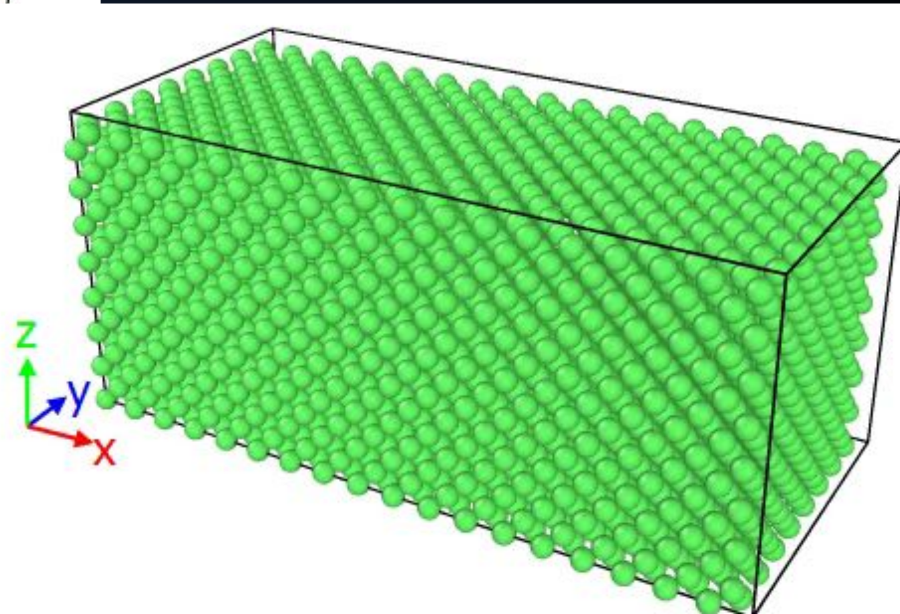
Sim2L in action: melting temperature



Too cold



Goldilocks region
(melting temperature)

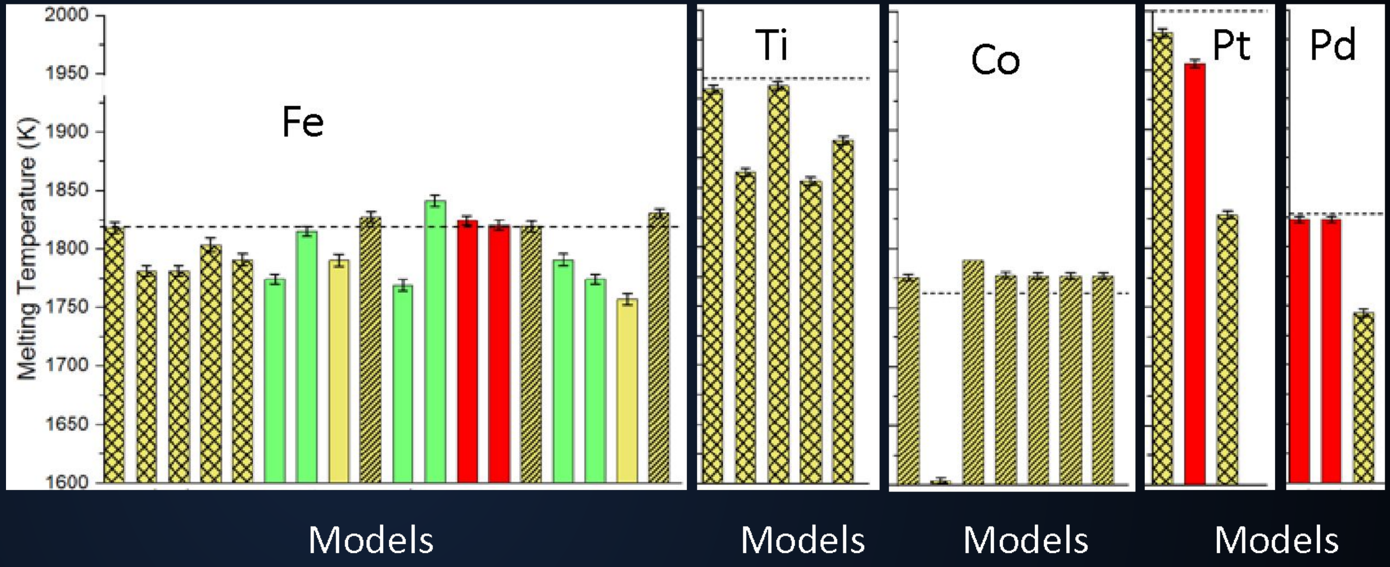


Too hot

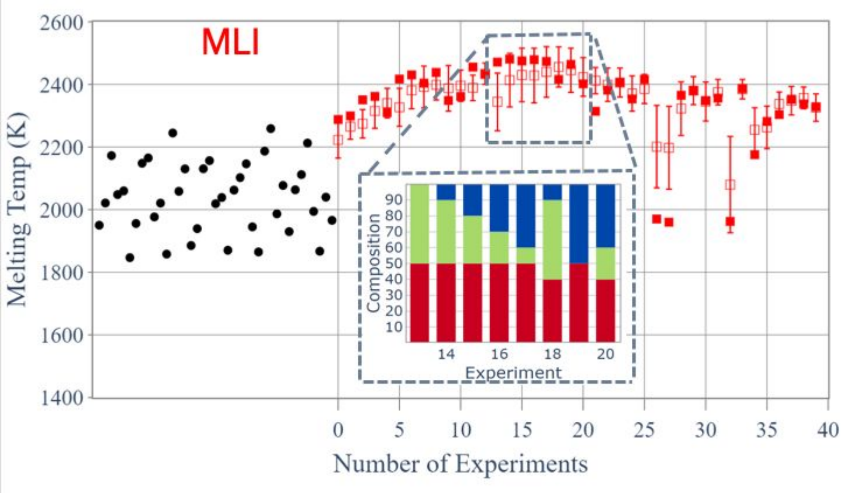
https://nanohub.org/tools/meltingkim

Cool things you can do with a Sim2L

High throughput simulations:



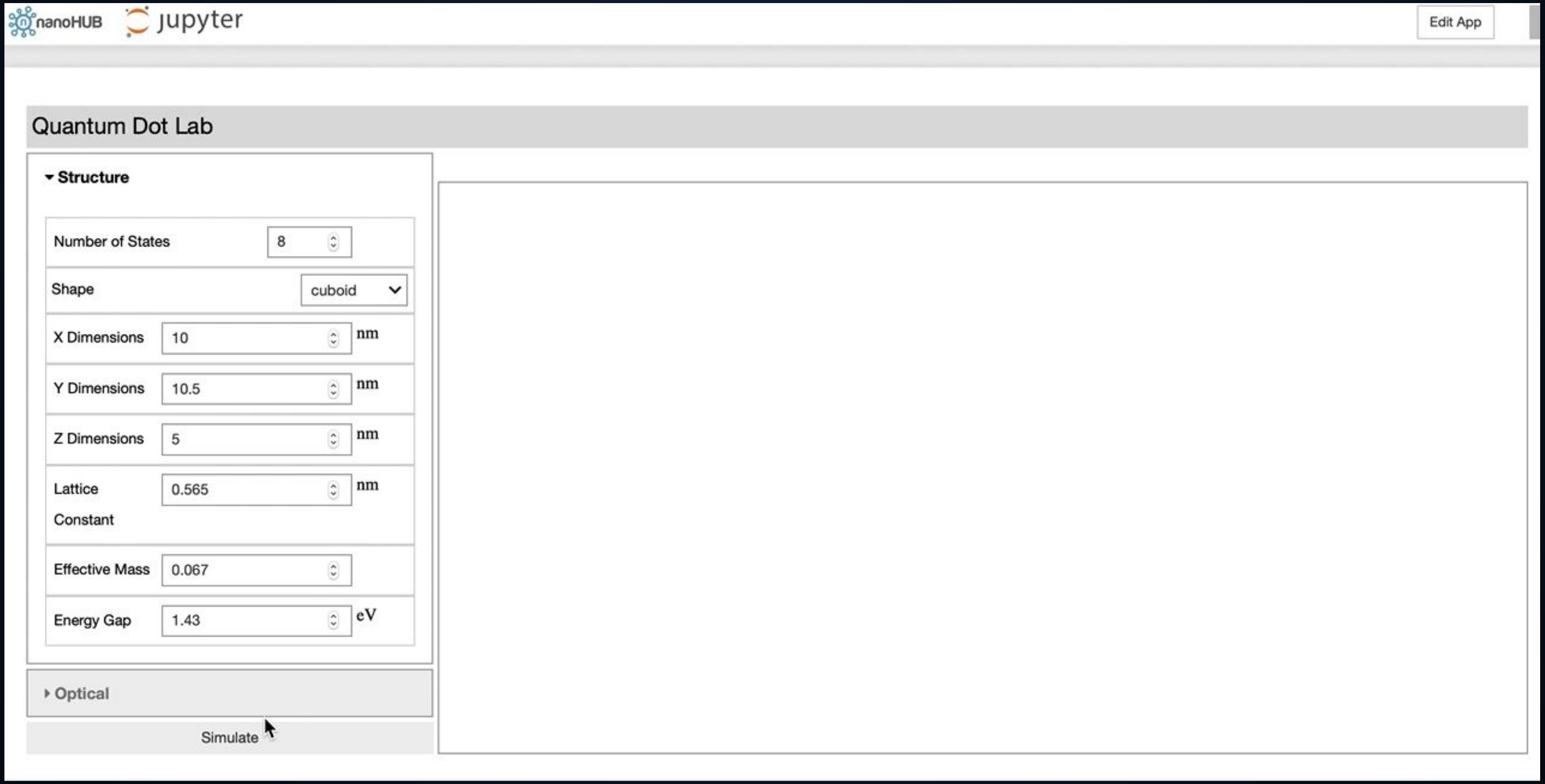
Machine learning driven workflows:



Cool things you can do with a Sim2L: II

Apps designed for end users

- Researchers and instructors
- Focus on the physics and not on the computational details



Quantum Dot Lab

▾ Structure

Number of States: 8

Shape: cuboid

X Dimensions: 10 nm

Y Dimensions: 10.5 nm

Z Dimensions: 5 nm

Lattice Constant: 0.565 nm

Effective Mass: 0.067

Energy Gap: 1.43 eV

▸ Optical

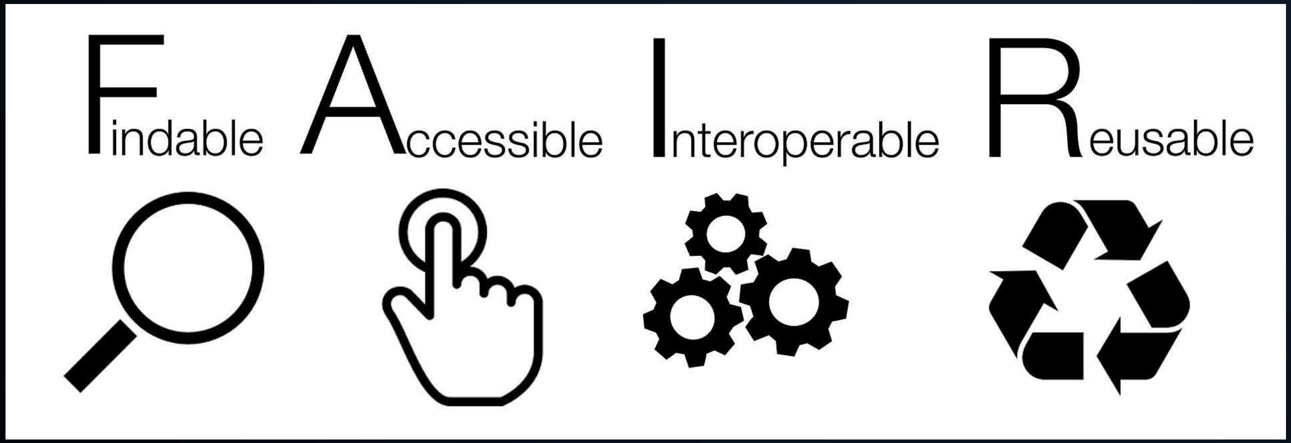
Simulate

https://nanohub.org/tools/qdotjuptest

https://nanohub.org/tools/st4pnjunction/

Summary – Sim2L features

- End to end computational workflow (Repro)
- Published Sim2Ls:
  - Are containerized (Repro)
  - Have DOIs and are indexed by Web of Science & google scholar (F,A)
- Declared and validated inputs and outputs (R, I)
- Services, including metadata, are queryable (F, A, I)
- Automatic result caching (A, R, I)

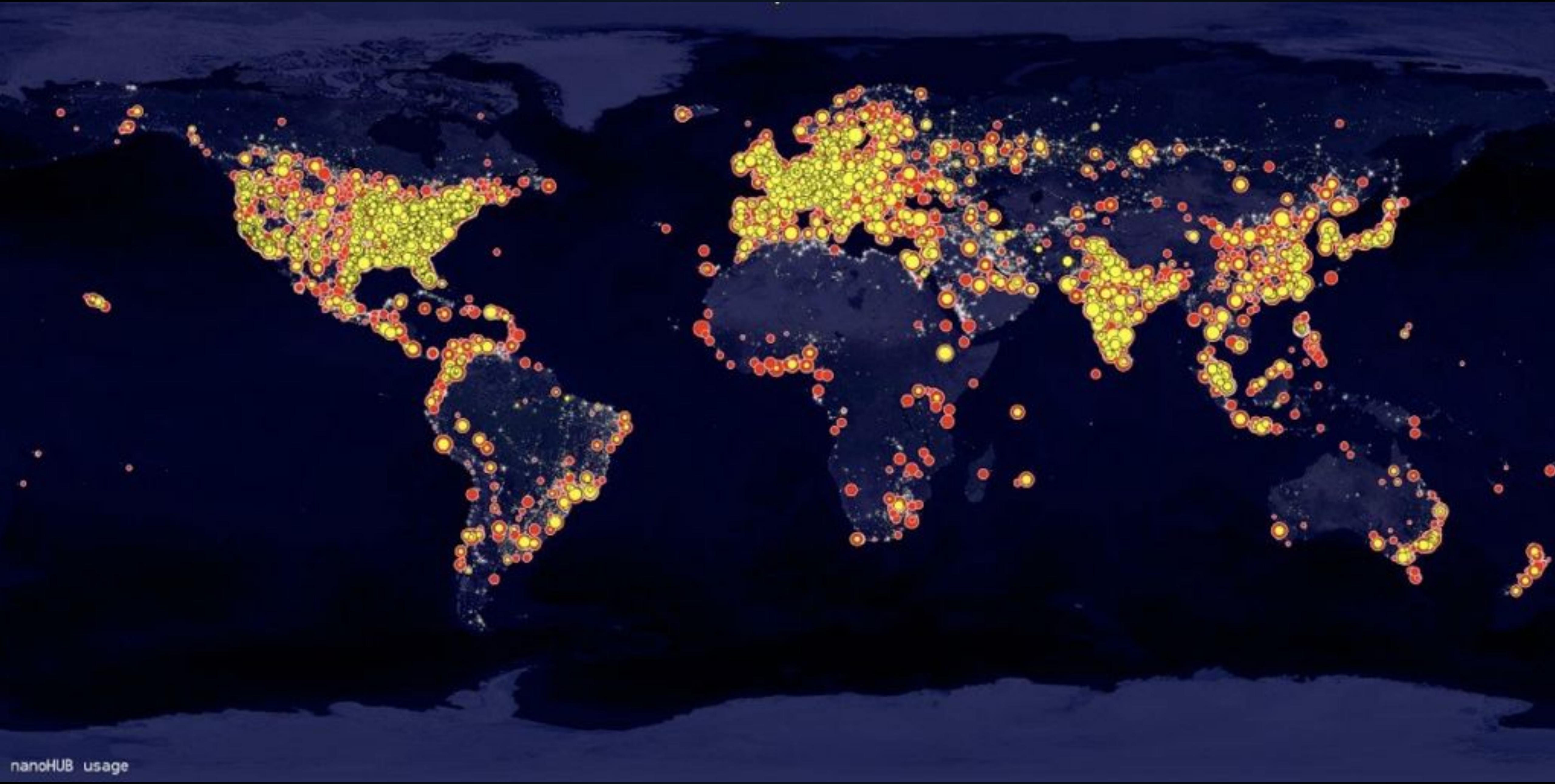


+ Reproducible

Additional resources

- Sim2L Documentation
  - https://simtool.readthedocs.io/en/latest/
- Explore a Sim2L example including all possible IN/OUT types
  - https://nanohub.org/tools/introtosimtools/
- Learn about nanoHUB software development environment
  - Overview of possible app/tool types & publication process:
    - https://nanohub.org/whypublish
  - Working with Jupyter in nanoHUB:
    - https://nanohub.org/resources/34611

Thanks


nanoHUB usage