Using IPython Notebook with IPython Cluster for Reproducibility and Portability of Atomistic Simulations

Zachary T. Trautt[*], L.H. Friedman[*], C.A. Becker[†]
[*] Materials Measurement Science Division,
[†] Materials Science and Engineering Division,
[*†] National Institute of Standards and Technology

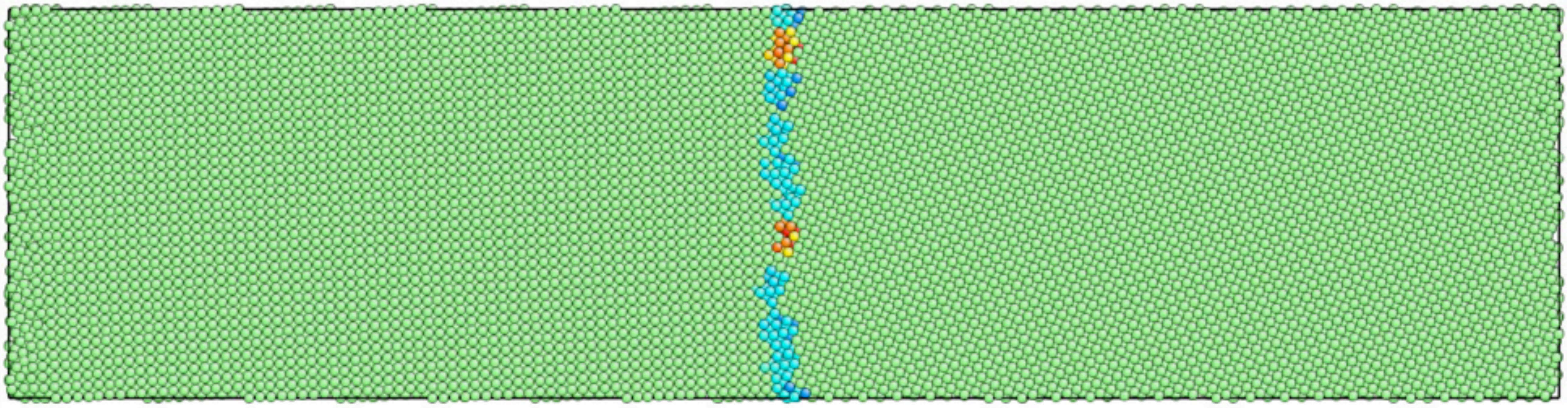
NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Outline

- Quick introduction to my work
- Why workflow is necessary
- How I use IPython Notebook and IPython Cluster

Example: Material Structure-Properties

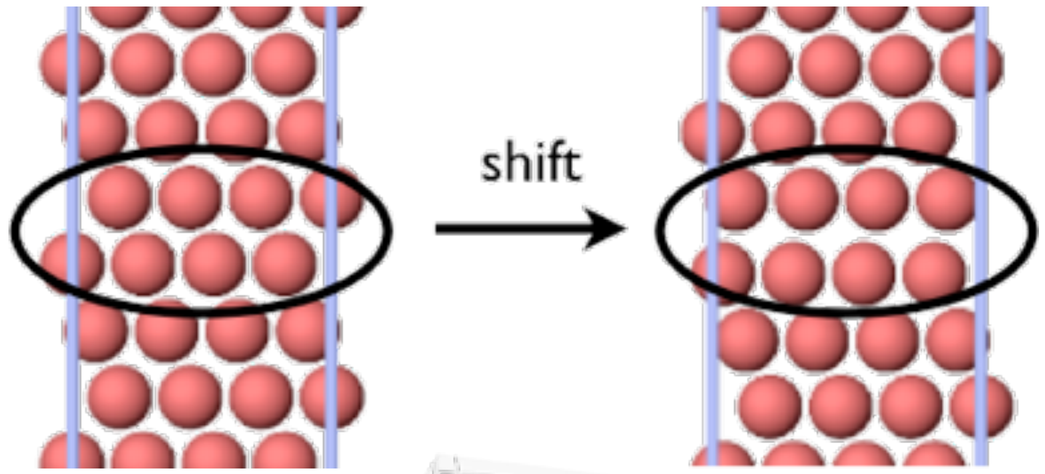
- Mechanical properties (strength of metal in your car)
  - Determined by microscopic structure
    - Study processing origin of microscopic structure with atomic-scale simulation

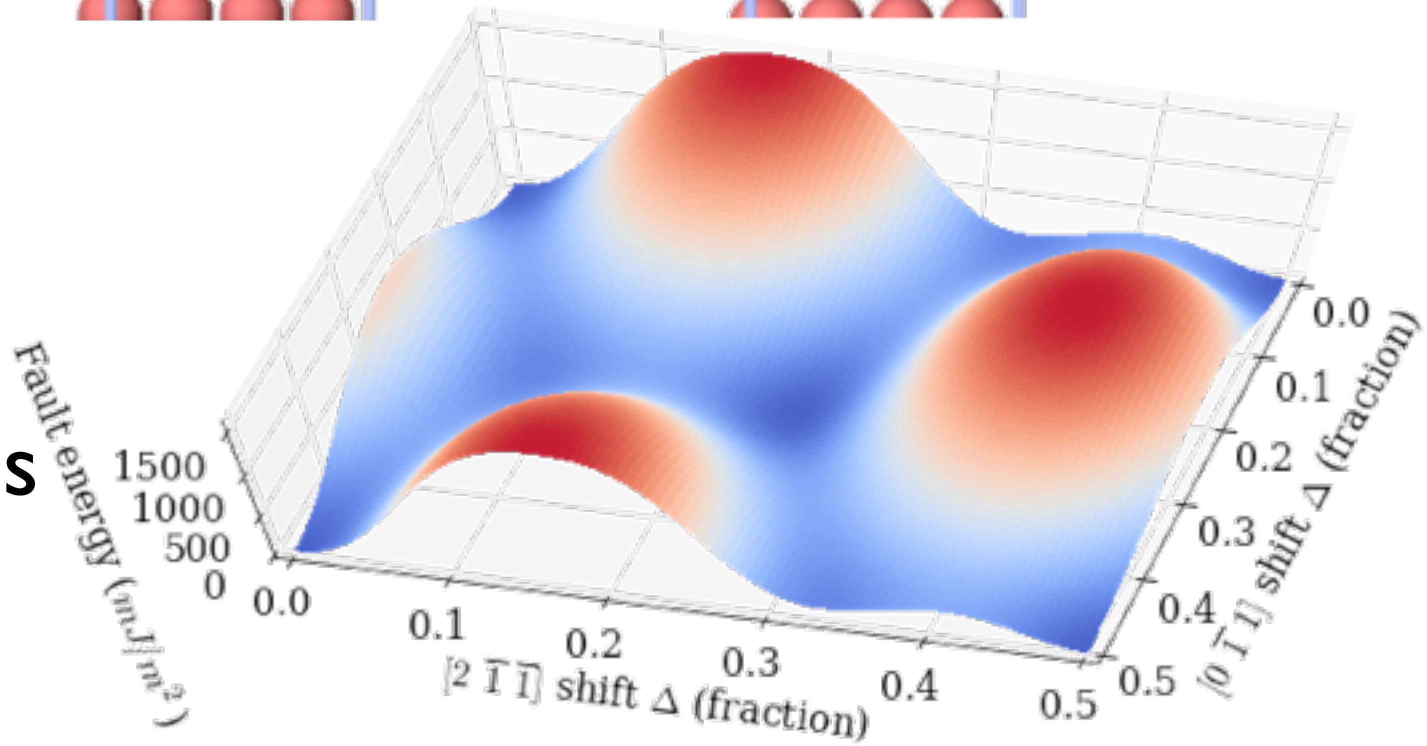


Trautt ZT, Adland A, Karma A, Mishin Y. Acta Mater 2012;60:6528.

Problem Complexity

- Many parameters: **many simulations**
  - Composition
  - Geometry
  - Temperature
  - Random seed
  - Methodology
  - Implementation details
  - Etc.





- Large simulation domain: **parallelism mandatory**
- Solution: must address distributable parallel tasks

What I have done in the past

- A **lot** of bash scripting...
  - for f in $params; do mkdir ...
  - for f in $folders; do cd $f; qsub preprossing ...
  - for f in $folders; do cd $f; qsub production ...
  - for f in $folders; do cd $f; qsub postprocessing ...
  - for f in $folders; do gather data ...
- Reproducibility with logging tool?
  - for f in $params; do mkdir ...
  - for f in $folders; do cd $f; qsub TheLoggerTool preprossing ...
  - for f in $folders; do cd $f; qsub TheLoggerTool production ...
  - for f in $folders; do cd $f; qsub TheLoggerTool postprocessing ...
  - for f in $folders; do gather data ...
- Reproducible, but not scalable to full problem complexity.
- Reproducible tasks, but have I recorded the recipe?
- Workflow tool?

Workflow, because logging is not enough

	Workflow Tool	Log Tool
Task automation, assembly, distribution	✓ Yes	?
Reproducibility	✓ Run workflow again (same input)	✓ Rerun a task
Reuse (explore parameter space)	✓ Run workflow again (different input)	?
Benchmark on local cluster, exec. in cloud	✓ Run workflow again (more/larger/longer simulations)	?
Efficiencies gained in automation	✓ Yes	?
Easy to build upon past work	✓ Change applicable portions of workflow and run again	?

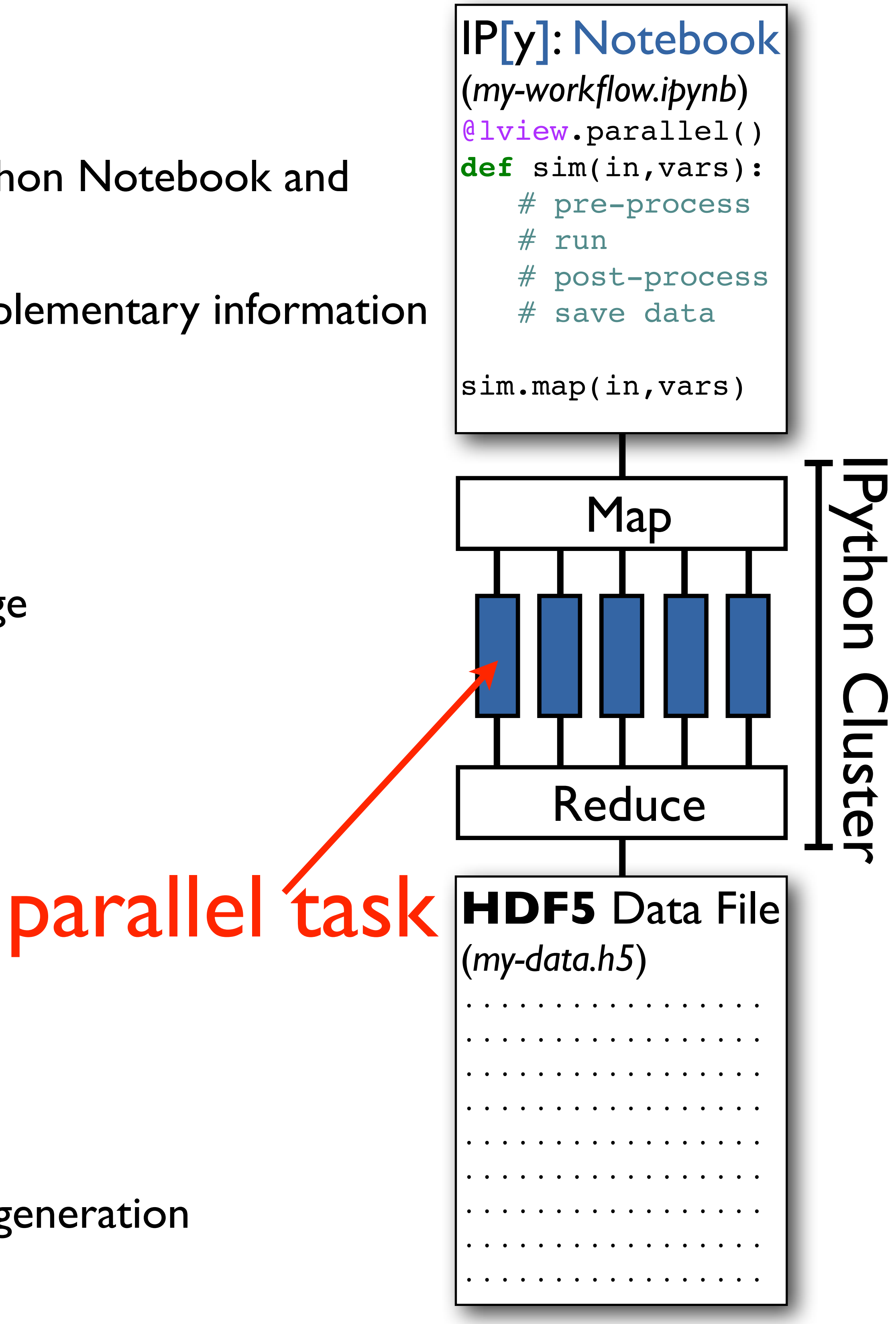
A look at traditional workflow tools, alternatives, etc.

- Is it possible to distribute parallel tasks to remote resources?
  - Some: (✓) yes
  - Some: (✗) no
  - Many: (?) unknown
- Does it appear to provide more benefit (automation) than cost (time invested in installation/setup/use)?
  - Some: (✓) yes
  - Some: (✗) no
  - Many: (?) insufficient applicable examples
- Is there strong package development to internalize scientific work and avoid reinventing the wheel?
  - Some: (✓) yes
  - Most: (✗) no

A "short" list:
DAGMan, Pegasus, Triana, ICENI, Taverna, GridAnt, GrADS, GridFlow, Unicore, Gridbus workflow, Askalon, Karajan, Kepler, Grid Mathematica, Galaxy, VisTrails, Workflow4ever, WorkWays, LabArchives, iRODS, Accelrys Pipeline Pilot, RapidMiner, LONI Pipeline, GWorkflowDL, Bioclipse Scripting Language, BioExtract Server, Tesla, BioVeL, SHIWA, HTCondor, Trident, Amazon Simple Workflow Service, Nipype, Sumatra, Wakari, Anaconda, joblib, Trac, gridfields

IPython benefits

- Easy to use!
- Plethora of packages and active development
- My entire workflow is defined and executed from IPython Notebook and results stored in a pytable
  - Both can be easily posted, shared, or added as supplementary information with a paper
- Improve reproducibility:
  - How to reproduce results
    - Download/compile molecular dynamics package
    - Install/configure IPython Cluster/Notebook
    - Run Notebook
      - Only trivial changes in notebook: /users/remote/project/directory/name/
- End-to-end automation
  - Improve efficiency
  - Reduce random mistakes
  - Enable moving work to the "cloud" for rapid data generation



```
@lview.parallel()
def sim(in,vars):
    # pre-process
    # run
    # post-process
    # save data

sim.map(in,vars)
```

Workflow - IPython Notebook

```
def simStep1(...):
    import numpy
    # do stuff
dview.push(dict(simStep1=simStep1));
```

Define Tasks

```
@lview.parallel()
def simulation(Temp,seed,Theta,Phi):
    simStep1(...)
    simStep2(...)
    simStep3(...)
```

Define Workflow

```
simulation.map(TEMP,SEED,THETA,PHI)
```

Execute

```
TEMP=[0,100,200,300,...] # (K)
```

- You can do a workflow in serial
- Or distribute to cluster

Workflow - IPython Notebook

```
def simStep1(...):
    import numpy
    # do stuff
dview.push(dict(simStep1=simStep1));
```

```
@lview.parallel()
def simulation(Temp,seed,Theta,Phi):
    simStep1(...)
    simStep2(...)
    simStep3(...)
```

```
simulation.map(TEMP,SEED,THETA,PHI)
```

Direct and load-leveled view of our IPython Cluster

```
from IPython.parallel import Client
rc = Client()
lview = rc.load_balanced_view()
lview.block = True
dview = rc[:]
dview.block = True
```

Workflow - IPython Notebook

```python
def simStep1(...):
    import numpy
    # do stuff
dview.push(dict(simStep1=simStep1));
```

```python
@lview.parallel()
def simulation(Temp,seed,Theta,Phi):
    simStep1(...)
    simStep2(...)
    simStep3(...)
```

```python
simulation.map(TEMP,SEED,THETA,PHI)
```

Make a unique working directory

```python
def mkWorkDir(basedir):
    import os, uuid
    uniqueID=str(uuid.uuid4())
    workDir=basedir+'ipengine-'+uniqueID
    os.mkdir(workDir)
    return workDir
dview.push(dict(mkWorkDir=mkWorkDir));
```

Workflow - IPython Notebook

```
def simStep1(...):
    import numpy
    # do stuff
dview.push(dict(simStep1=simStep1));
```

```
@lview.parallel()
def simulation(Temp,seed,Theta,Phi):
    simStep1(...)
    simStep2(...)
    simStep3(...)
```

```
simulation.map(TEMP,SEED,THETA,PHI)
```

Large-scale Atomic/Molecular Massively Parallel Simulator

http://lammps.sandia.gov/index.html

S Plimpton J Comp Phys, 1995;117:1-19.

Write a LAMMPS input script

```
def lmpscript(... timestep, ...):
    ...
    f.write('timestep %f\n' % timestep)
    ...
    f.close()
dview.push(dict(lmpscript=lmpscript));
```

Workflow - IPython Notebook

```python
def simStep1(...):
    import numpy
    # do stuff
dview.push(dict(simStep1=simStep1));
```

```python
@lview.parallel()
def simulation(Temp,seed,Theta,Phi):
    simStep1(...)
    simStep2(...)
    simStep3(...)
```

```python
simulation.map(TEMP,SEED,THETA,PHI)
```

Large-scale Atomic/Molecular Massively Parallel Simulator

http://lammps.sandia.gov/index.html

S Plimpton J Comp Phys, 1995;117:1-19.

Run LAMMPS

```python
def runlmp(workDir,lmpExe,lmpName,cpus):
    import os, subprocess
    os.chdir(workDir)
    string='mpirun -np '+str(cpus)+' '+lmpExe+' < '+lmpName
    subprocess.call([string],shell=True)
dview.push(dict(runlmp=runlmp));
```

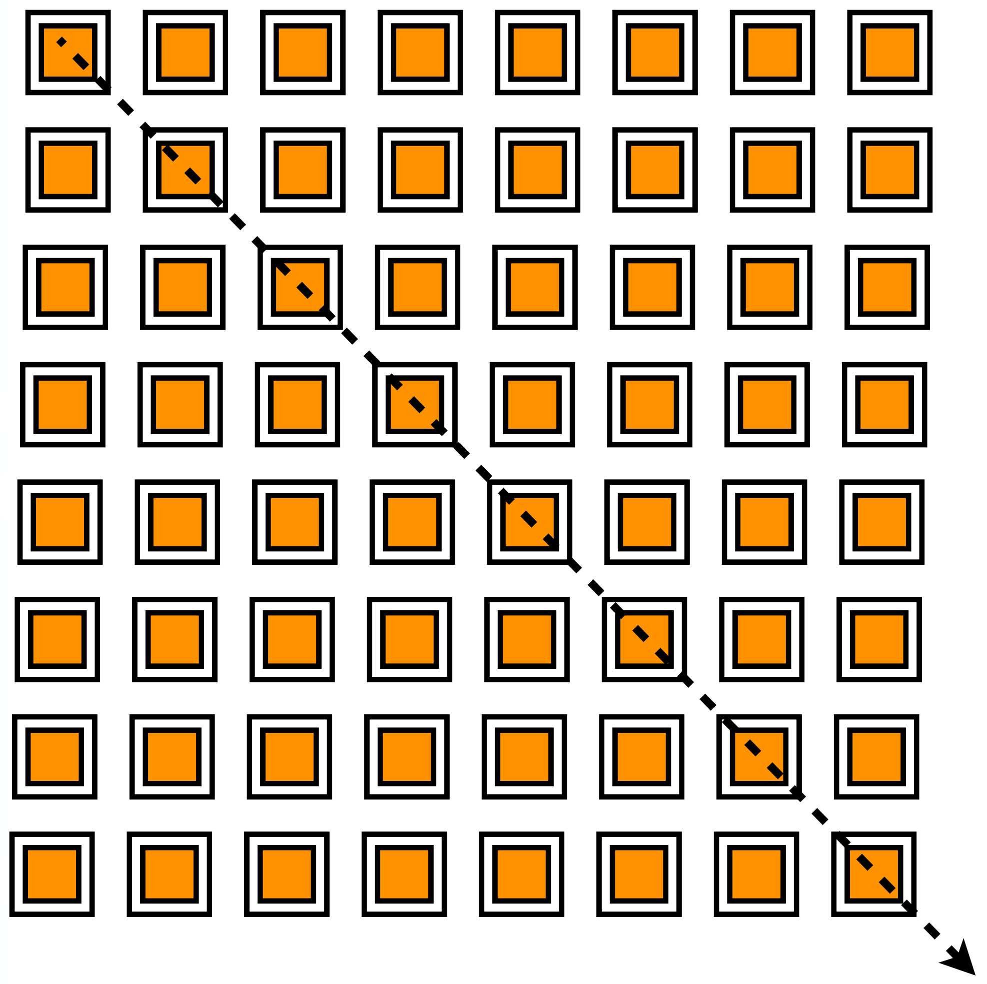
Parallel External Tasks - IPython Cluster

```
simulation.map(TEMP,SEED,THETA,PHI)
```

IPython Cluster for Serial Tasks:

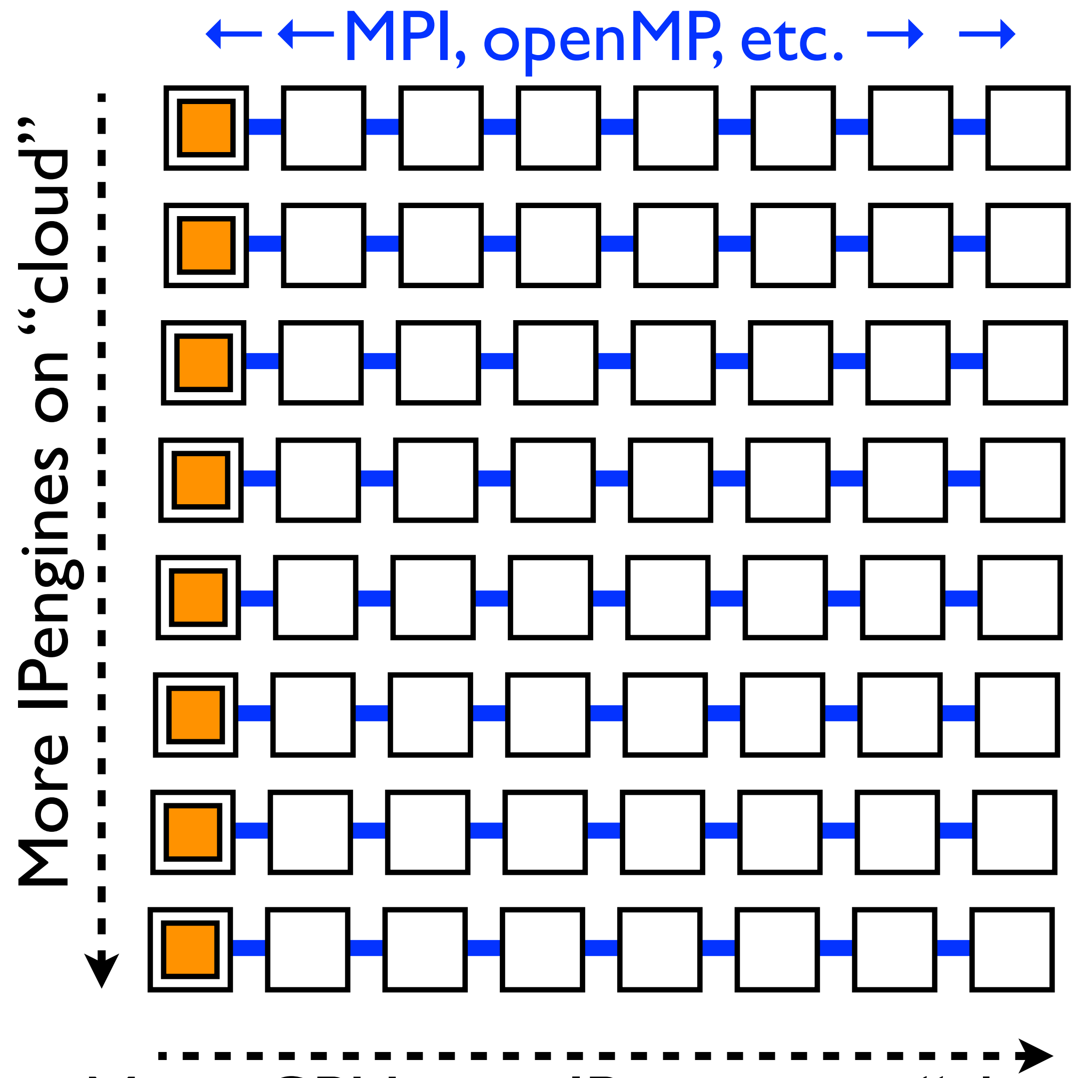
□ : CPU core

🟧: IPython Engine



More CPUs and IPengines on "cloud"

IPython Cluster for Parallel External Tasks:

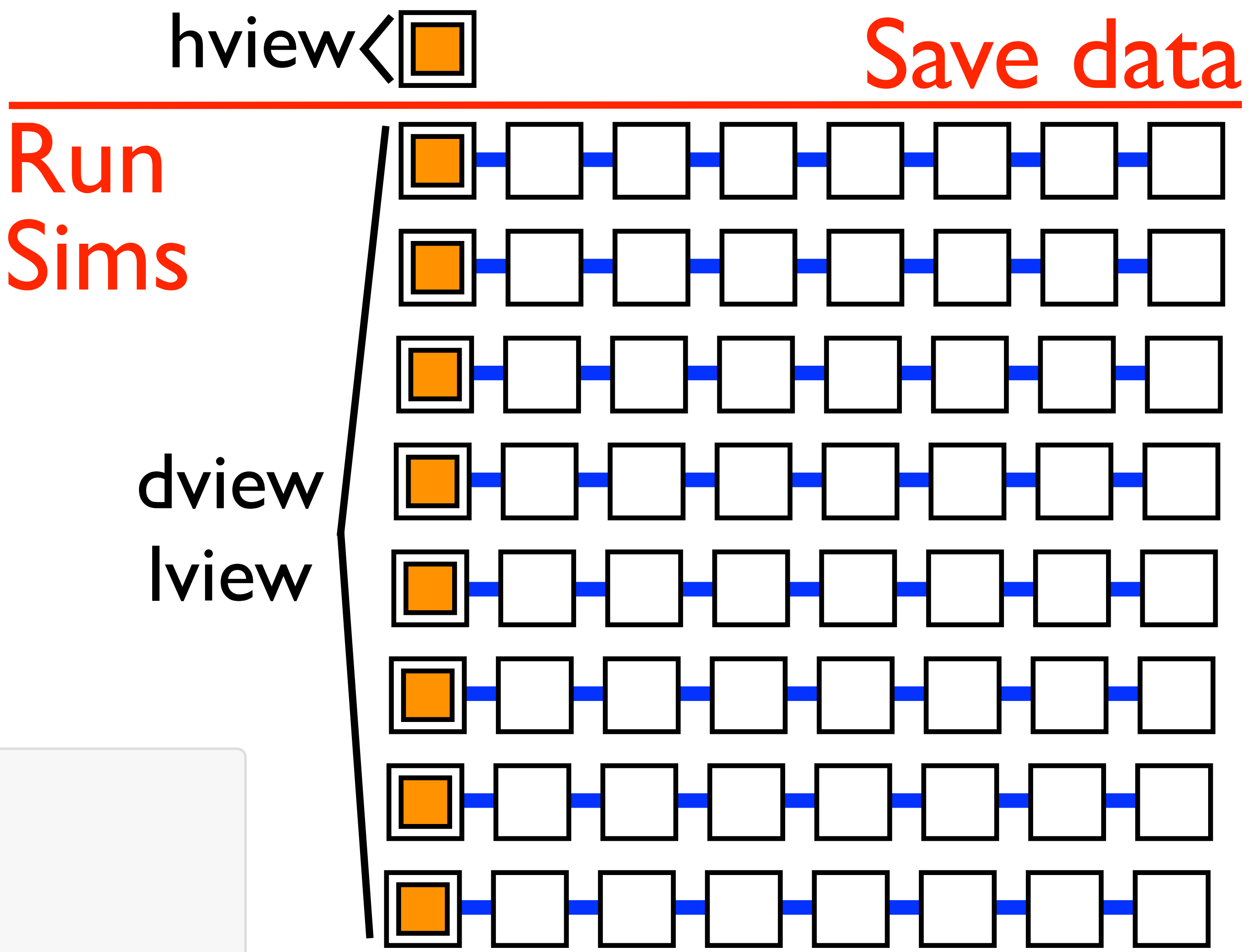


More CPUs per IPengine on "cloud"

Saving Data - HDF5

```
from IPython.parallel import Client
rc = Client()
myEngines=[ i for i in xrange(1,len(rc.ids))]

lview = rc.load_balanced_view(myEngines)
lview.block = True

dview = rc[1:]
dview.block = True

hview = rc[0]
hview.block = True

@dview.remote()
def sethview():
    from IPython.parallel import Client
    rc = Client()
    global hview
    hview = rc[0]
    hview.block = True
sethview();
```

```
def recordResult(C11,...):
    ...
    simulation['C11']=C11
    simulation.append()
    ...
dview.push(dict(recordResult=recordResult));
hview.push(dict(recordResult=recordResult));
```

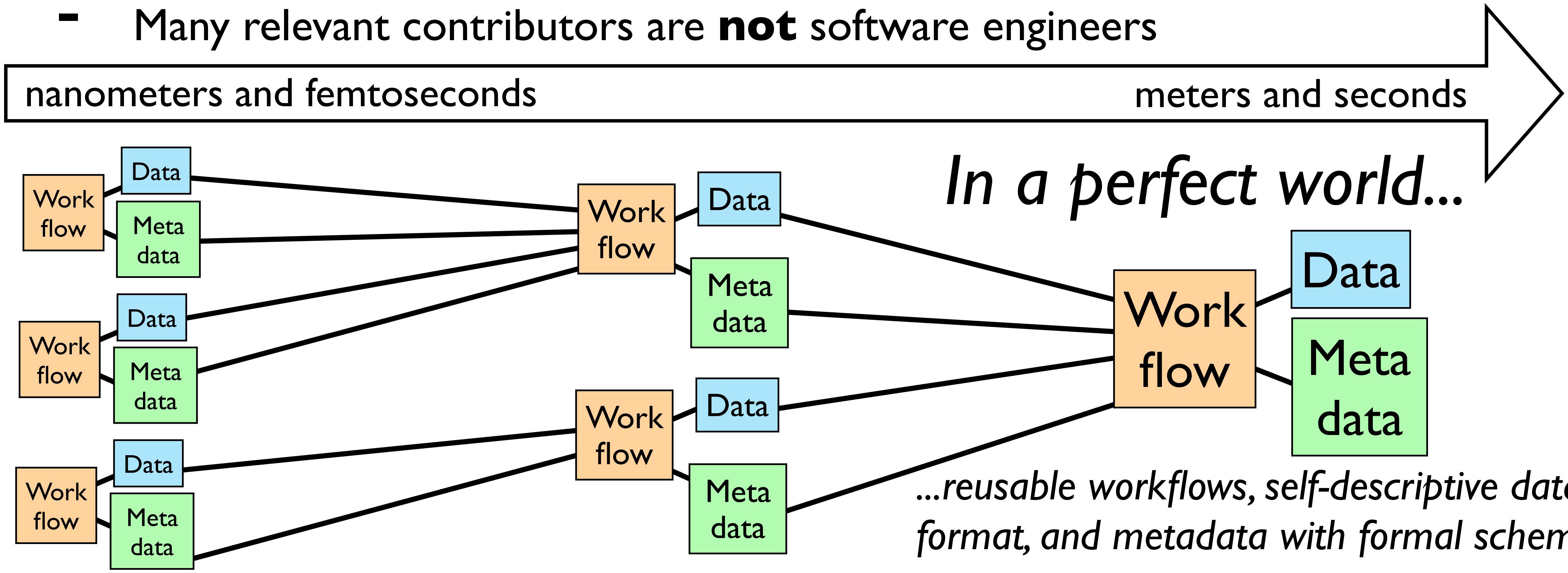


```
@lview.parallel()
def simulation(...):
    ...
    hview.apply_sync(recordResult,...)

simulation.map(...)
```

Example IPython notebook

Summary

- IPython Notebook + IPython Cluster
  - Easy: use, reproducibility, reuse, etc.
  - Do more work with less effort and very little code
- Grand challenges in materials science
  - Complex problem: big, disparate data
  - Relevant behavior at different length/time scales: multi-scale modeling
  - Many relevant contributors are **not** software engineers

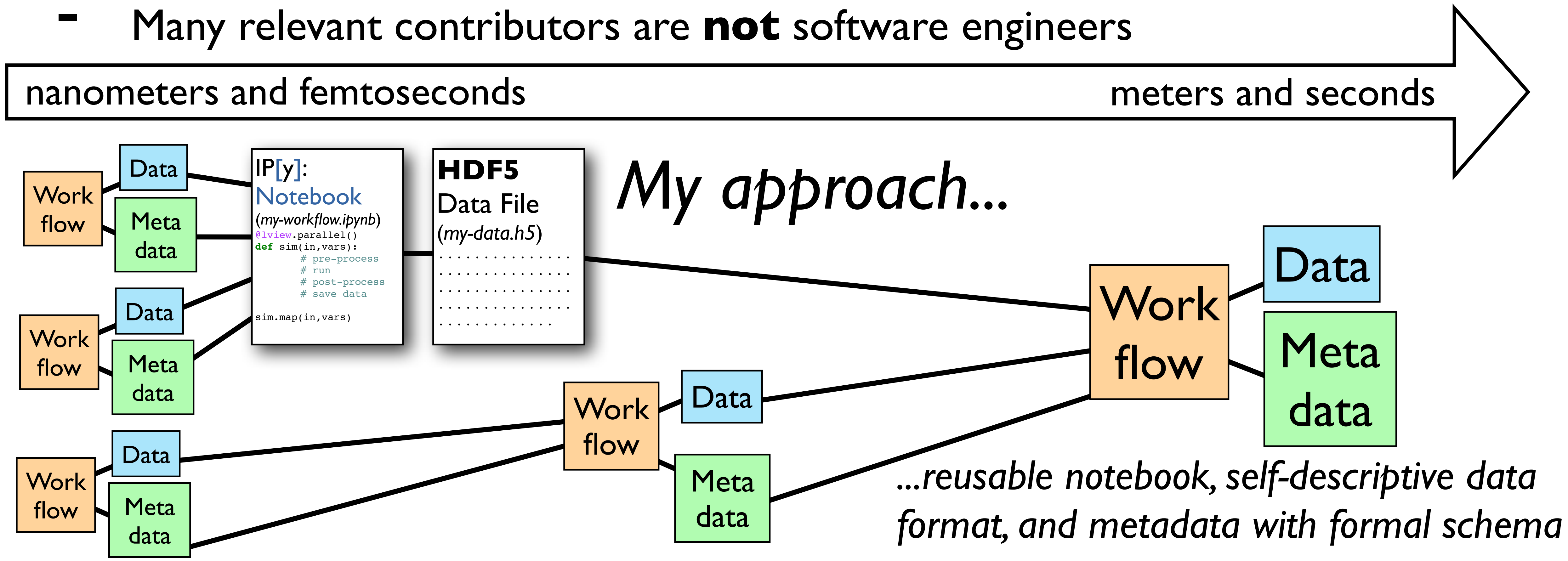


In a perfect world...

...reusable workflows, self-descriptive data format, and metadata with formal schema

Summary

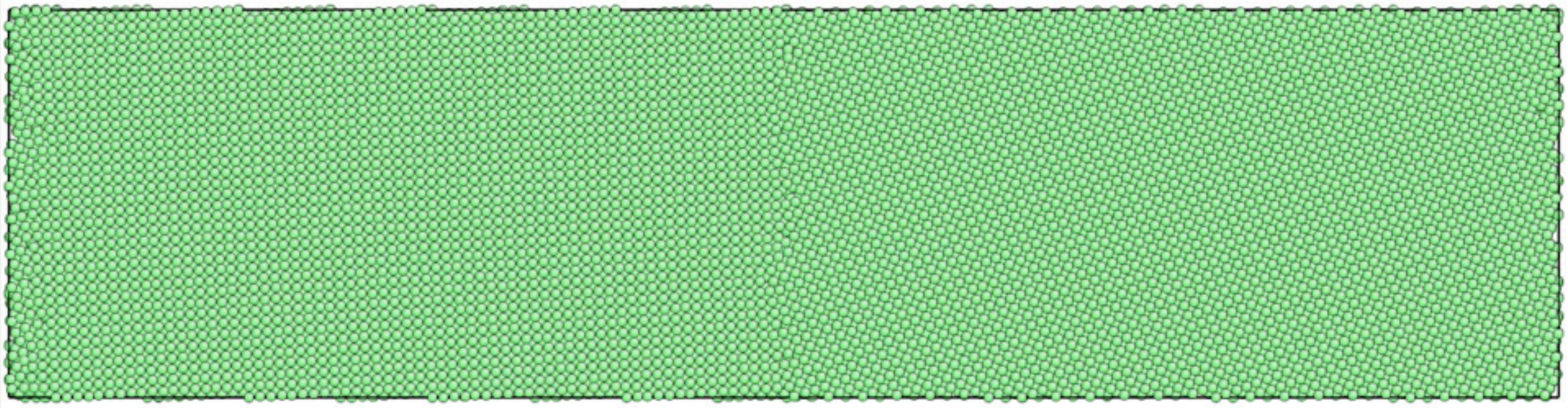
- IPython Notebook + IPython Cluster
  - Easy: use, reproducibility, reuse, etc.
  - Do more work with less effort and very little code
- Grand challenges in materials science
  - Complex problem: big, disparate data
  - Relevant behavior at different length/time scales: multi-scale modeling
  - Many relevant contributors are **not** software engineers



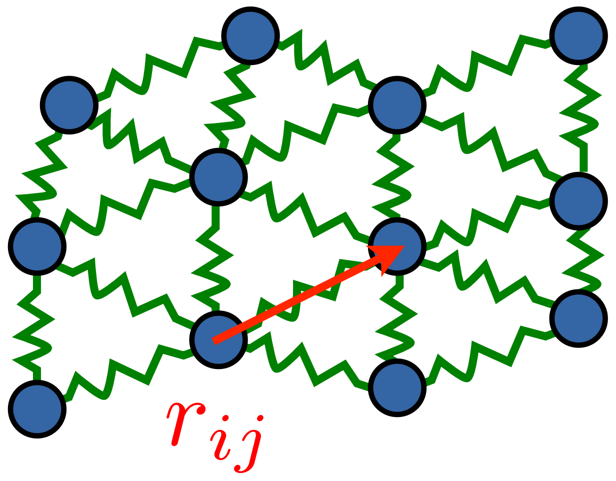
```
@lview.parallel()
def sim(in,vars):
        # pre-process
        # run
        # post-process
        # save data

sim.map(in,vars)
```

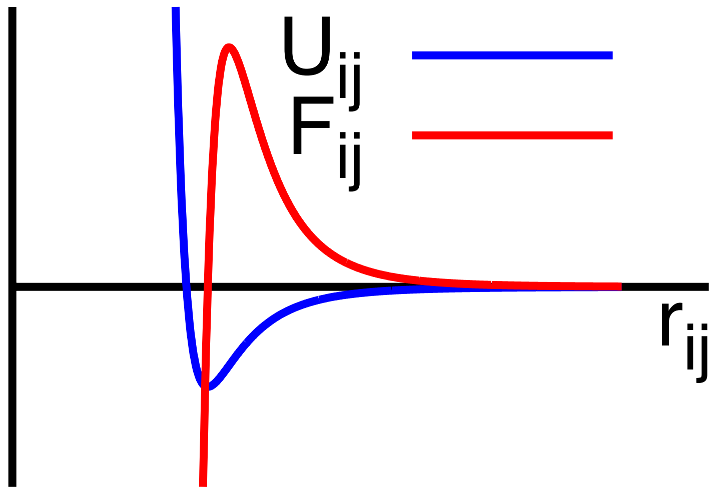
...reusable notebook, self-descriptive data format, and metadata with formal schema

Thank you!



Atomistic Simulation





$$\vec{F} = m\vec{a}$$

- Define an atomic configuration
- Define interaction forces and energy between atoms
- Numerically solve Newton's equation of motion

IPython Engine - MPI task

Portable Batch System (PBS) Script:

```
#PBS -l nodes=1:ppn=8:amd
#PBS -o status.log
#PBS -e status.err
#PBS -N ipython
#PBS -q default
ipengine
```

8 CPUs per IPython Engine