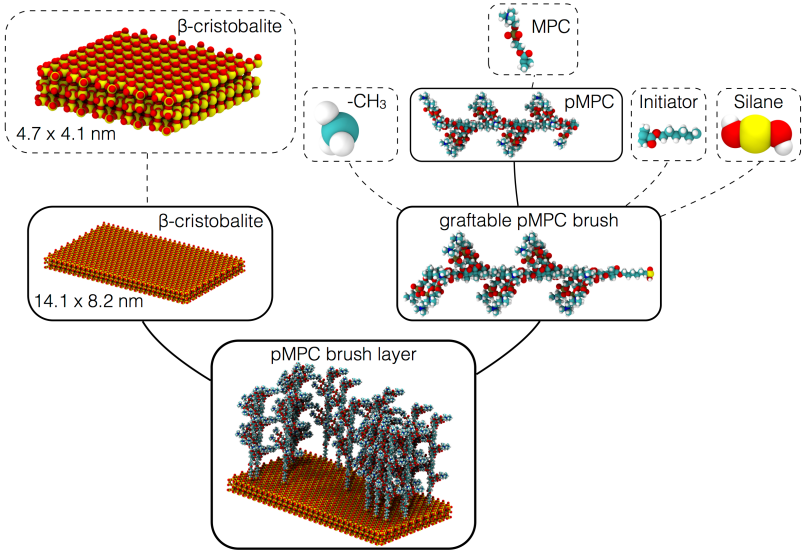
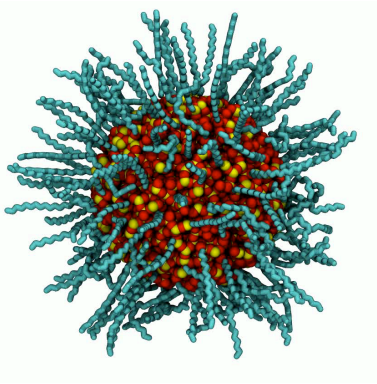
VANDERBILT

MoSDeF

A Molecular Simulation and Design Framework





ChBE 4830

September 5, 2017

VANDERBILT

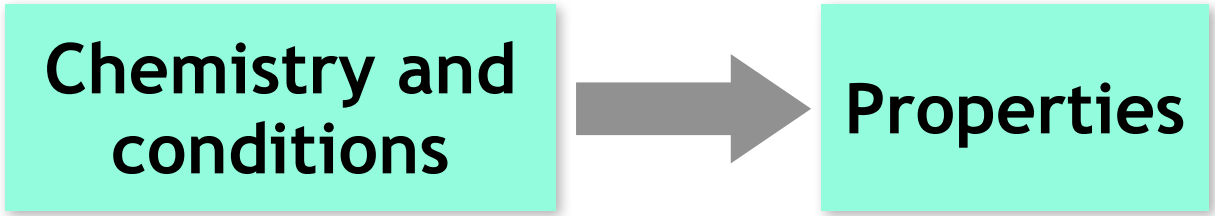
Plan for this week

- Today (9/5)
  - Introduction to MoSDeF
  - Installing mBuild, Foyer, and HOOMD
  - Homework
    - Lennard-Jones simulations in HOOMD
    - mBuild tutorials
- Thursday, 9/7
  - Interactive MoSDeF tutorial
  - Homework
    - Setting up a complex molecular system using mBuild and Foyer

VANDERBILT

MoSDeF – Motivation

- Develop a platform for performing Materials Genome Initiative (MGI) style screening of soft-matter systems



- The procedures grouped into the arrow are not always easy to reproduce
  - Procedures are not always well-documented/described
    - *Properties may depend on the exact details of the procedures*
- These procedures are often highly specific and not extensible
  - *ad hoc* tools/scripts are common

VANDERBILT

MoSDeF – Motivation

- Practical considerations/common problems we are attempting to address with MoSDeF
  - Incorrectly used force field parameters can dramatically change results
    - Such errors are not always easy to detect, especially once published
  - Considerable time and effort are often required to construct new molecular systems
    - Typical *ad hoc* codes may not be easy to extend or programmatic in nature (required for screening)
  - Existing software tools do not necessarily work together
    - Different languages, programming models, data formats, etc.
  - Simulations are not always easily reproduced due to lack of specific details of:
    - Exact simulation procedures/parameters
    - Codes/parameters/procedures used to implement and setup the models

VANDERBILT

MoSDeF – Goals

- Reduce the effort required for each of the steps for set-up/atom-typing/execution
- **TRUE** simulations
  - **T**ransparent
  - **R**eproducible
  - **U**sable by others
  - **E**xtensible
- Open-source workflow routines
  - All software hosted on Github
    - Others can view, download, and make modifications for their own research

VANDERBILT

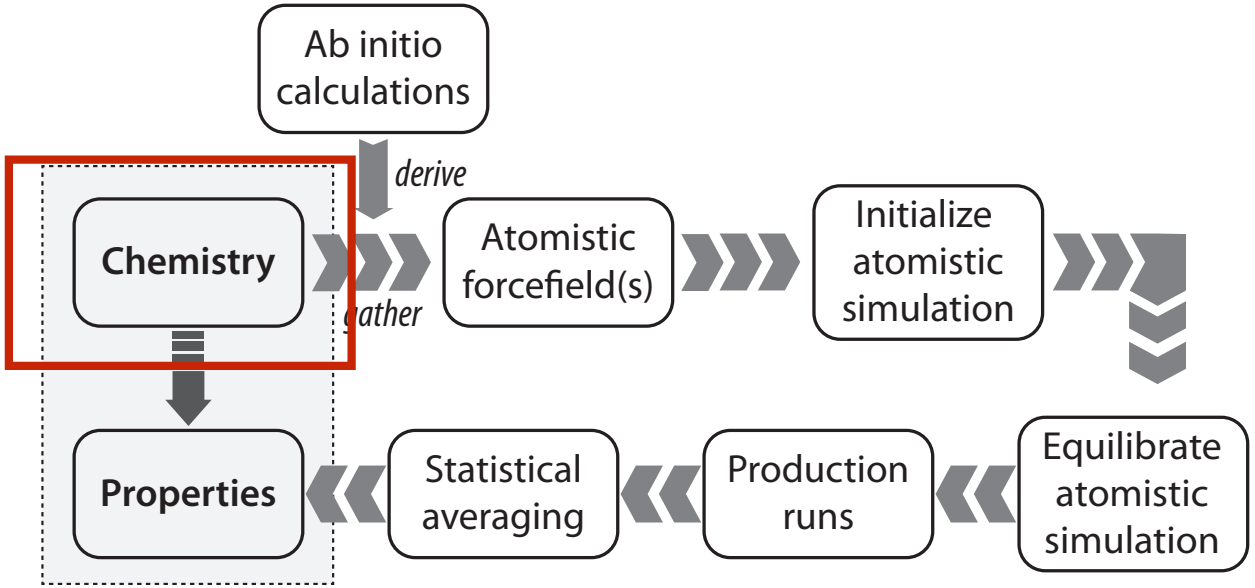
MoSDeF – Tools

- mBuild
  - A hierarchical, component-based toolkit for construction of complex molecular systems
- Foyer
  - A general atom-typer, for determining the proper force field parameters
- Signac*
  - Handles simulation configuration, execution, and archiving

*Developed by the Glotzer group at the University of Michigan

VANDERBILT

Idealized simulation workflow



- Chemistry goes in
  - Properties come out
- Use whatever force field/simulation engine/analysis tools you like

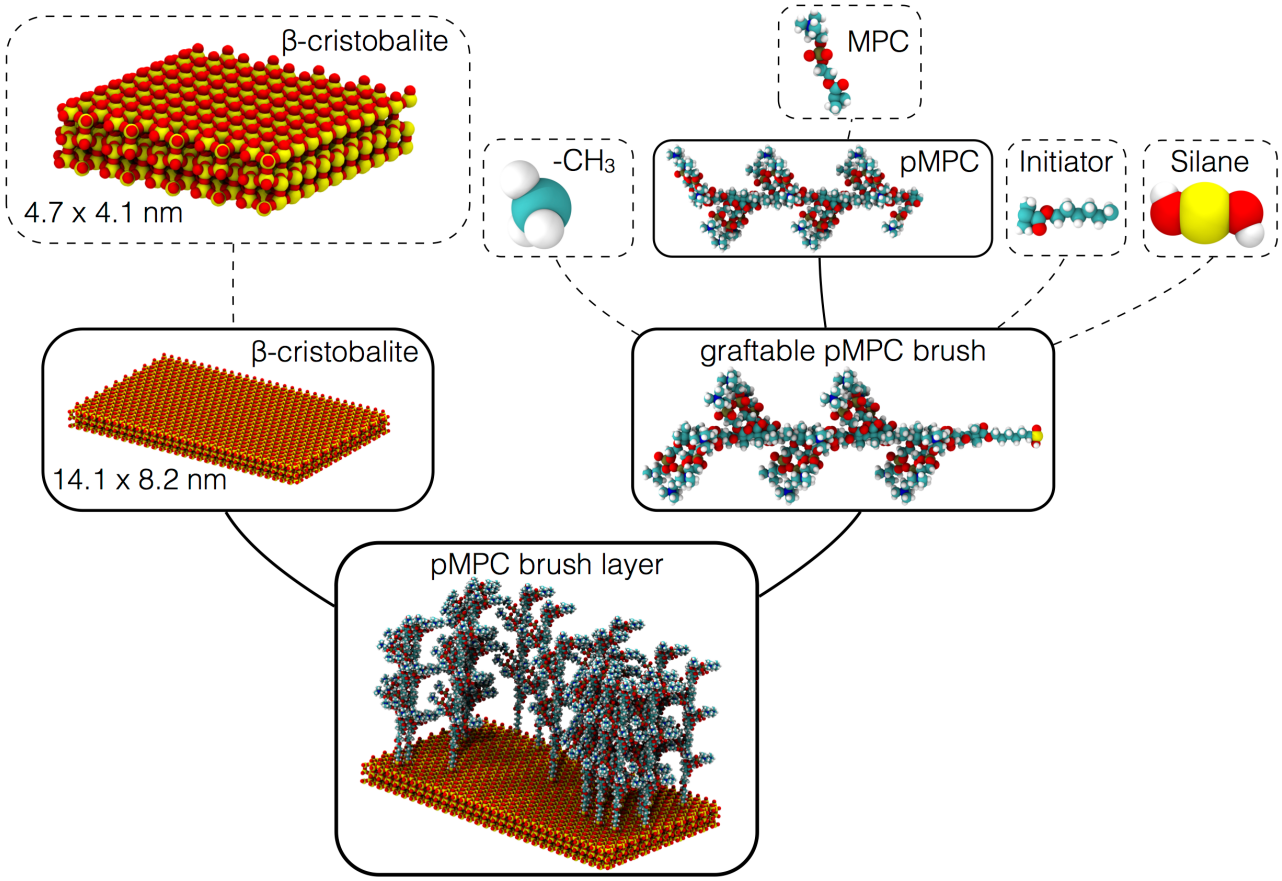
VANDERBILT

Desired functionality for defining chemistry

- Seamlessly construct systems with substrates, polymers, arbitrary solvents, etc. in a similar and programmatic fashion
- Naturally express features such as
  - Surface functionalization
  - Replicating molecules in a box
  - Defining attachments between atoms (i.e. bonds)
- Expose chemical parameters as top-level variables
  - e.g. bulk polymer
    - Polymer chemistry, number of monomers per polymer, number of polymer molecules
  - Need to be able to easily tune these knobs
    - Screening/optimization

mBuild: a hierarchical molecular builder

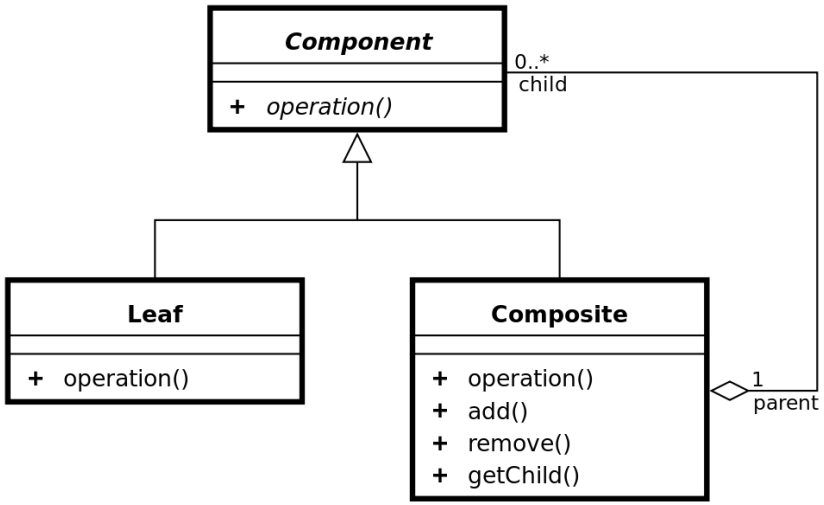
- Build molecular systems using a "ground-up" approach
  - Sketch or otherwise create simple components (dashed boxes)
  - Use built-in routines to manipulate these components into complex structures



VANDERBILT

The hierarchy: composite design pattern

- Every mbuild.Compound can contain other mbuild.Compound's
  - Compound's know about their children and their parent
- Leaves in the tree are referred to as Particle's

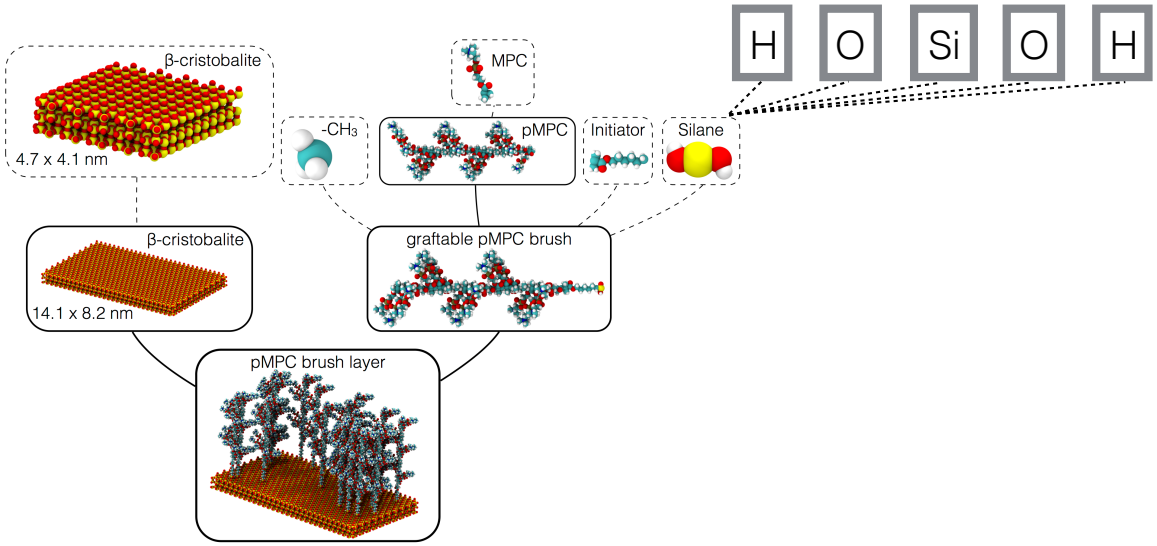


https://www.wikiwand.com/en/Composite_pattern

- Particle positions and bonds maintained only at leaf level
  - Higher level components can compute properties based on contained leaves

The hierarchy: composite design pattern

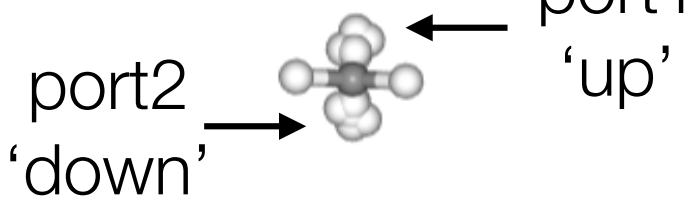
- Every mbuild.Compound can contain other mbuild.Compound's
  - Compound's know about their children and their parent
- Leaves in the tree are referred to as Particle's

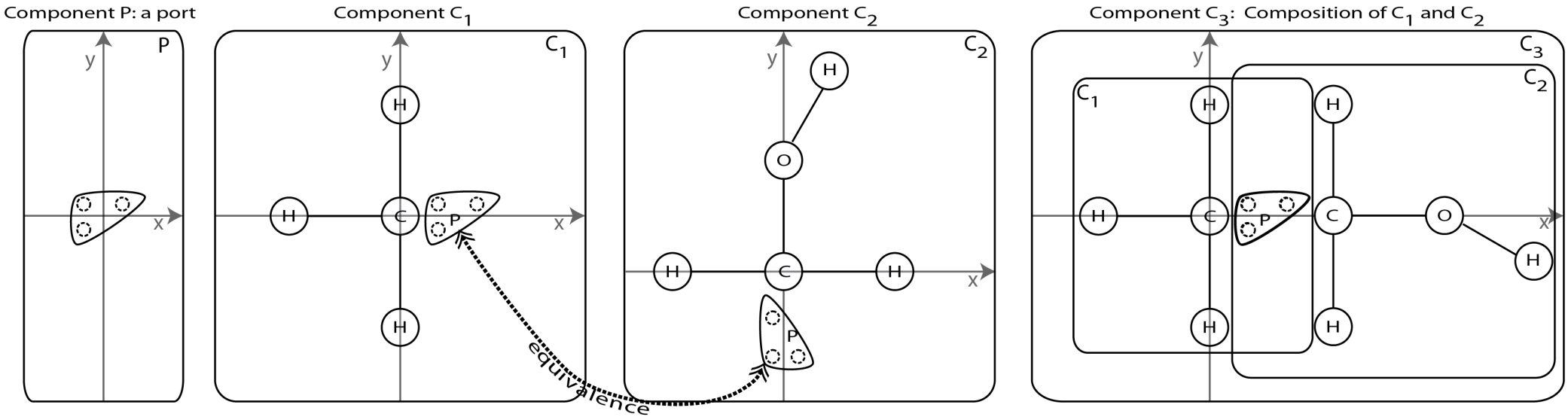


- Particle positions and bonds maintained only at leaf level
  - Higher level components can compute properties based on contained leaves

Connecting components

- Port particles placed where connections can exist
  - Have orientation and direction
- Force overlap of port particles
  - Solve for rotation matrix and translation vector that map ports onto one another
  - Rotate and translate attached component by same amount

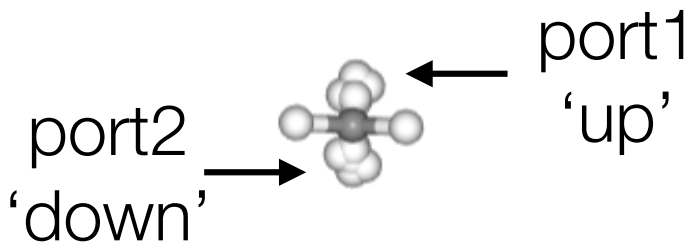




VANDERBILT

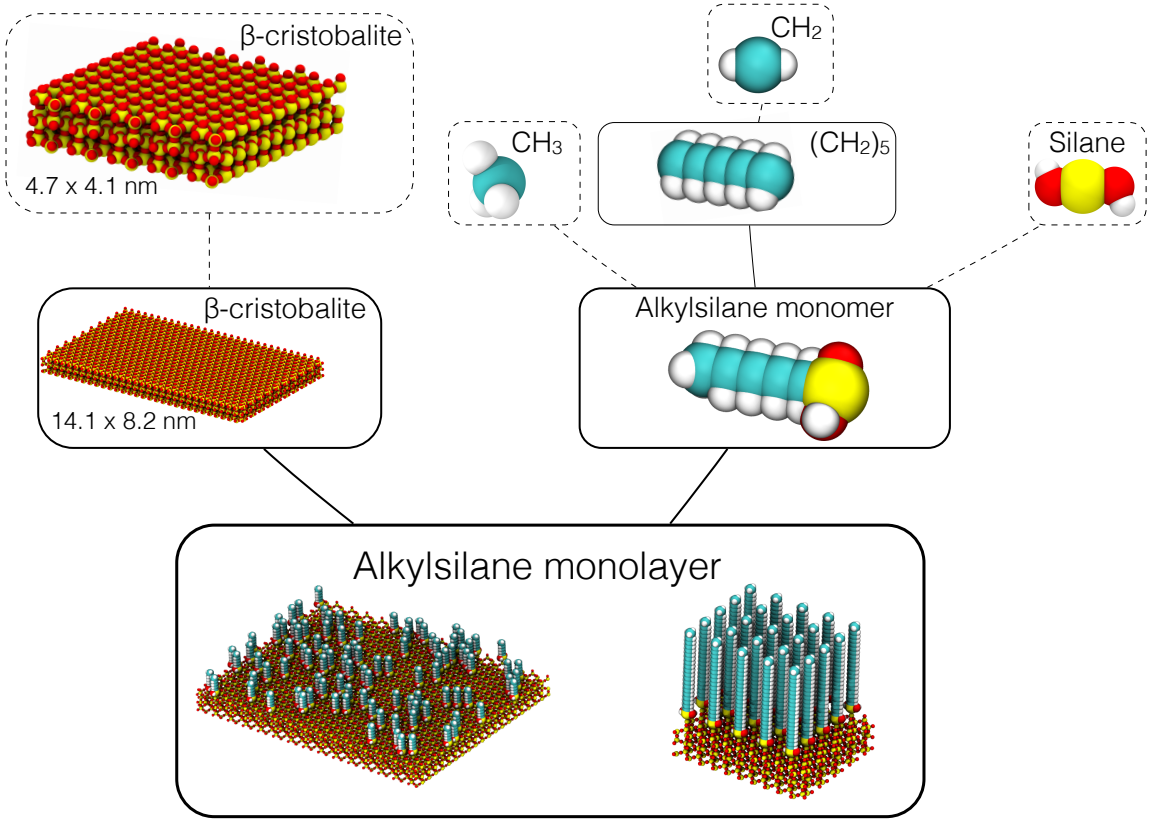
Example: polymerization

```
import mbuild as mb
from mbuild.lib.moieties import CH2

polymer = mb.Compound()

last_monomer = CH2()
polymer.add(last_monomer)

for _ in range(10):
    current_monomer = mb.clone(last_monomer)

    mb.force_overlap(move_this=current_monomer,
                     from_port=current_monomer['up'],
                     to_port=last_monomer['down'])

    polymer.add(current_monomer)
    last_monomer = current_monomer
```



VANDERBILT

Exposing tunable chemistry into simple variables

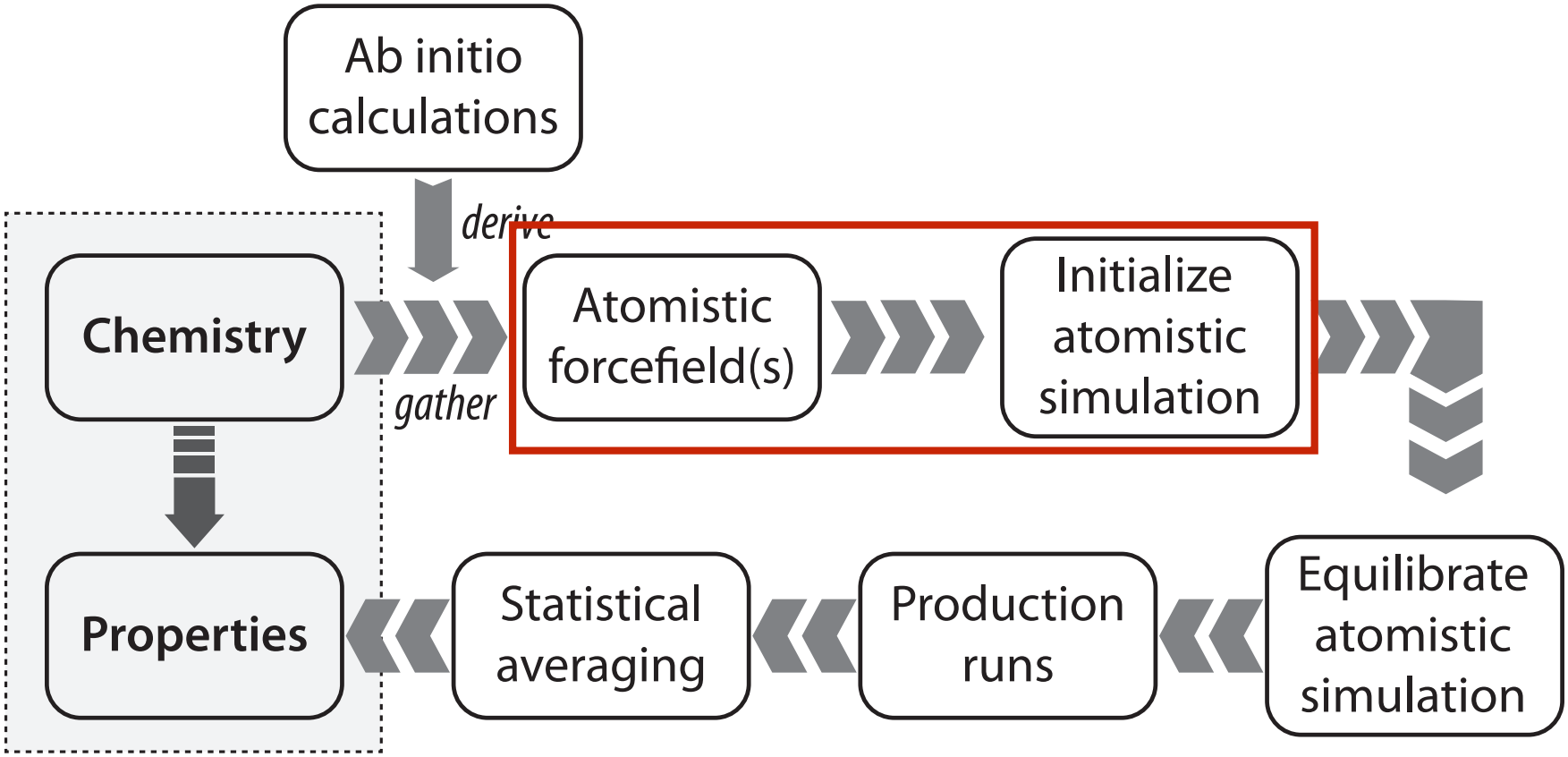



```
pattern = Random2DPattern(100)
chain_length = 5
n_tiles = [2, 3, 1]
```

```
pattern = Grid2DPattern(5, 5)
chain_length = 20
n_tiles = [1, 1, 1]
```

```
monolayer = AlkaneMonolayer(chain_length, pattern, n_tiles)
```

VANDERBILT


Ab initio calculations
derive
Chemistry
gather
Atomistic forcefield(s)
Initialize atomistic simulation
Properties
Statistical averaging
Production runs
Equilibrate atomistic simulation

VANDERBILT

Atom typing and applying force fields

- Atom typing refers to assigning an "atom type" to each atom/particle in a molecular system
  - Force fields define different interaction parameters for different atom types
- Atom types are defined by the local chemistry
  - For example, a carbon atom in an alkane, an aromatic ring, and a carbonyl will likely have different interaction parameters
    - Although all three are the same element, they will have different atom types

VANDERBILT

Atom typing and applying force fields

- Need a general purpose automated atom typer
  - Arbitrary chemistry means we cannot rely on templates
  - Needs to be easy to develop new atom types and associated logic
  - Don't want to rely on rigid rule hierarchies for atom type definitions
    - Ordered least to most general
- Removing gap between logic and parameters
  - Parameters exist in various files
  - Logic often encoded as series of if/else statements deep in source code
  - Why not combine the two?

VANDERBILT

Foyer: applying and disseminating force fields

- End user should only require:
  - Force field file
  - Chemical topology

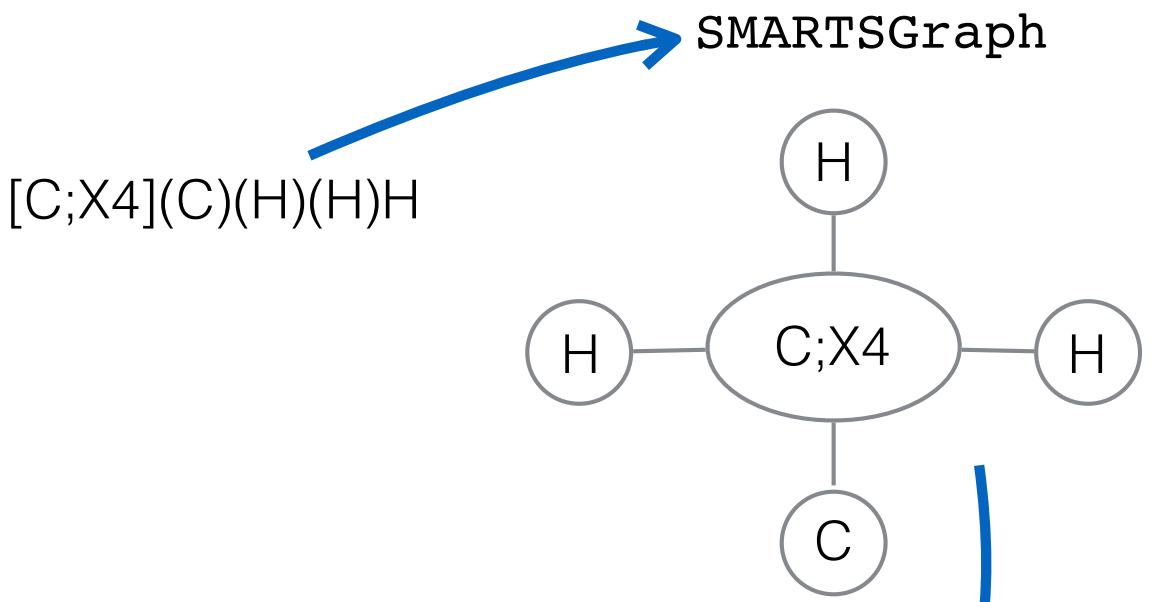
```
from foyer import Forcefield
import parmed as pmd

untyped_ethane = pmd.load_file('ethane.mol2', structure=True)
oplsaa = Forcefield(forcefield_files='oplsaa.xml')
ethane = oplsaa.apply(untyped_ethane)

# Save to any format supported by ParmEd
ethane.save('ethane.top')
ethane.save('ethane.gro')
```

VANDERBILT

Defining the logic: SMARTS strings

- SMARTS strings are an extension of SMILES (Simplified Molecular-Input Line-Entry System)
  - Provides a language to describe chemistry
  - Various symbols/operators for defining, e.g.
    - The number of bonds
    - Whether an atom is part of a ring
  - More information: http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html
- Examples
  - [C;X4](C)(H)(H)H
    - Alkane CH3
  - [C;X3]([O;X1])H
    - C in aldehyde

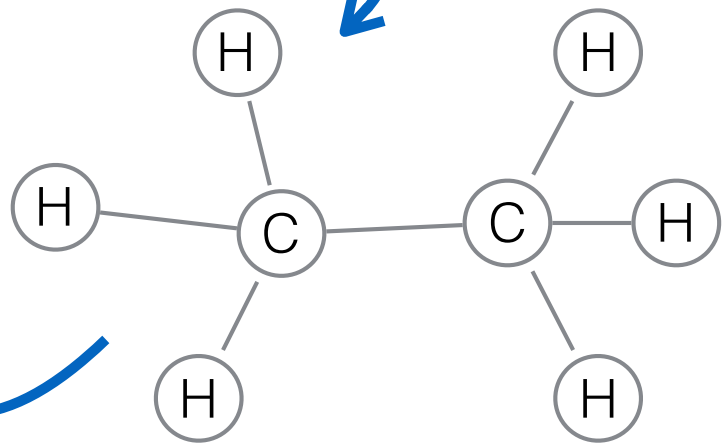
Evaluating the SMARTS based definitions



- Parse SMARTS into a graph
- Perform an isomorphism on graph of entire system

`SMARTSGraph.find_matches(topology)`

yield matching atom indices



[0, 1]

the two carbons in this example

VANDERBILT

Iteratively evaluating rules

- Each atom maintains a white- and a blacklist
  - Iteratively evaluate SMARTSGraph's on atoms in topology
    - Add matches to white list and overrides to blacklist
    - Once converged, set difference between white and blacklist yields atom type

alkene CH, opls_142: [C;X3](C)(C)H
benzene C, opls_145: [C;X3;r6]1[C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6]1
overrides="opls_141, opls_142"

- Pros:
  - Easy to add new atom types
    - Blacklisting gives user extra debug info
      - *Blacklisting network can show e.g. disjoint graphs (types that can never be reached)*
- Cons:
  - Slightly decreased performance (although not currently bottleneck)

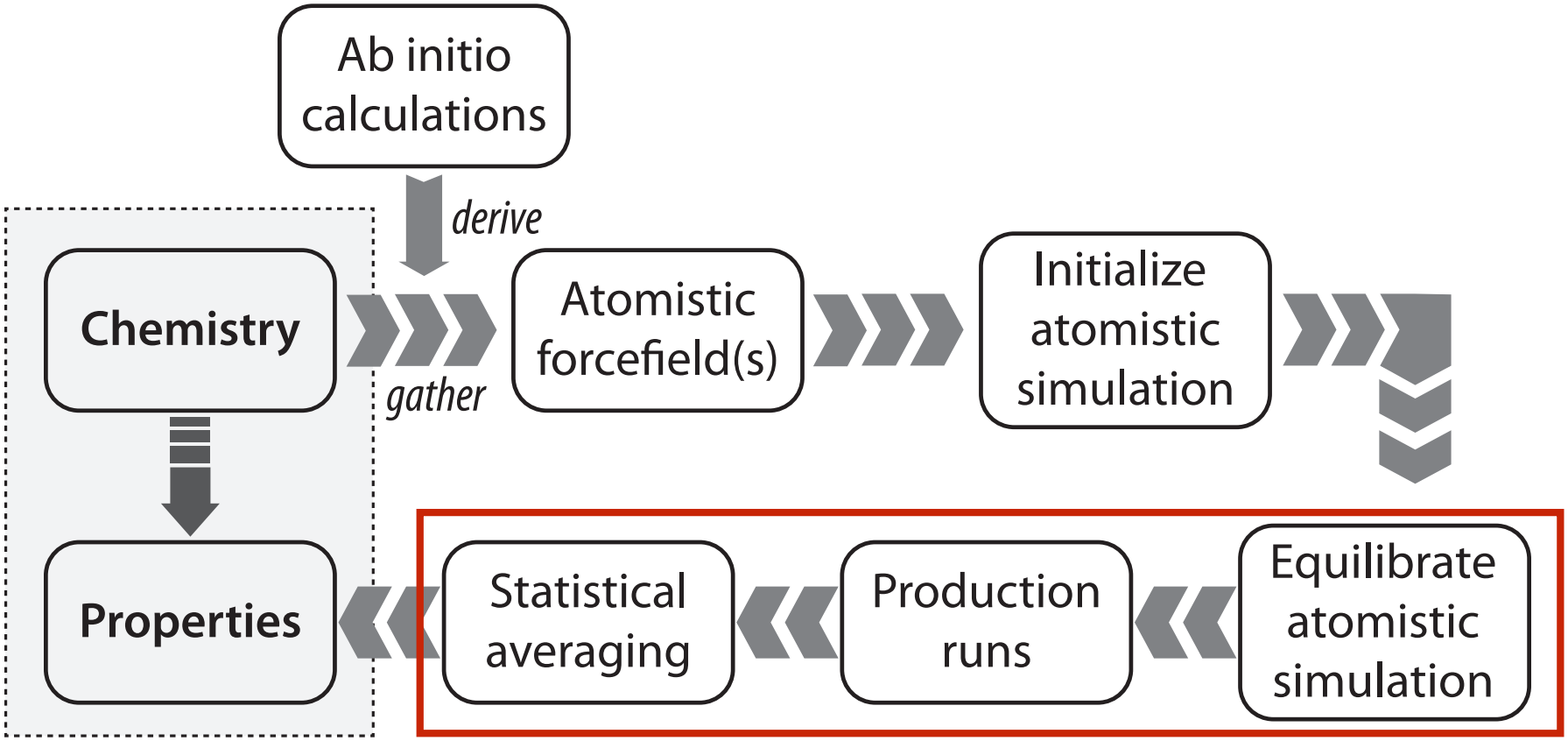
VANDERBILT

Promoting dissemination of force fields

- Template repo for building foyer compatible force field files
  - https://github.com/mosdef-hub/forcefield_template
  - Skeleton force field file
  - Skeleton testing setup
    - User needs to only add correctly typed .mol2 files
  - Tutorial for adding SMARTS definitions
    - Walkthrough for adding more and more complex definitions
    - Testing with small example set of hydro carbons from alkanes through alkenes and benzene
    - How to run tests with py.test and enable TravisCI for your repo
- Force field development within our group

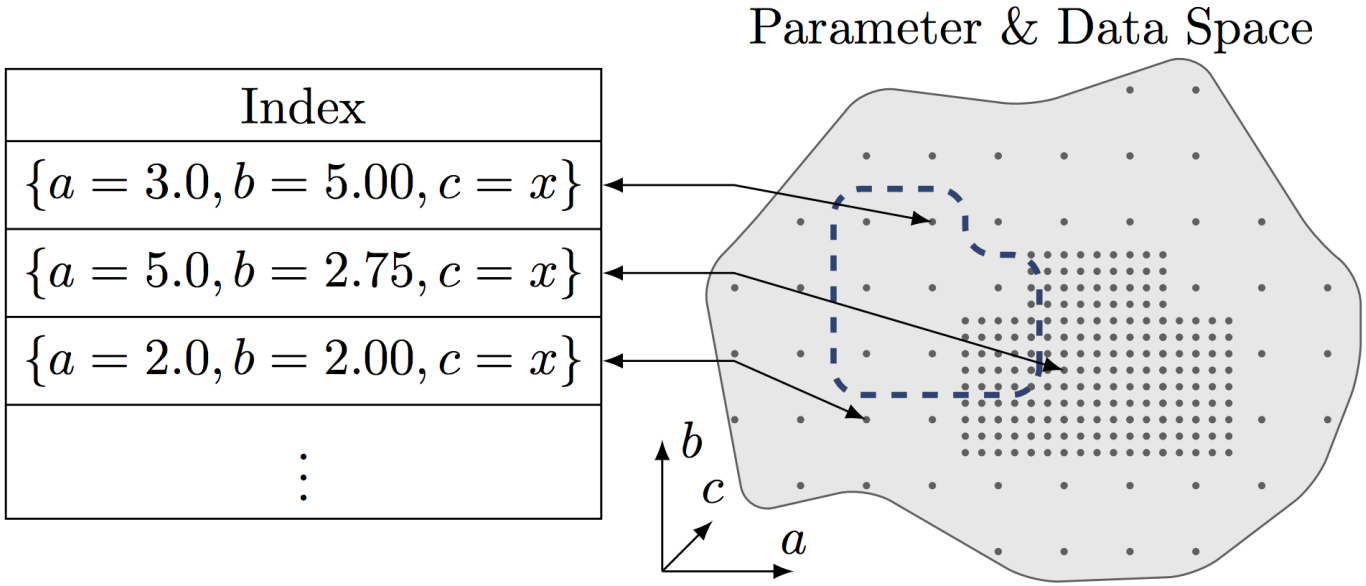
  OPLS-AA compatible parameters for perfluoroethers

  DOI 10.5281/zenodo.56807
  - Coarse grained models for skin lipids
  - Automating force field derivation for supercapacitor electrodes


Ab initio calculations
derive
Chemistry
gather
Atomistic forcefield(s)
Initialize atomistic simulation
Properties
Statistical averaging
Production runs
Equilibrate atomistic simulation

Putting it all together: Signac workflow manager

- Define heterogenous parameter spaces
- Provides link between data and metadata
  - Every job and its data is tied to a point in parameter space
  - Provides glue for seamless building, execution, and analysis



Adorf et al., arXiv:1611.03543 [cs.DB]

VANDERBILT

Putting it all together: Signac workflow manager

- Define arbitrary operations as Python functions
- Our tribology operations
  - mBuild + save with Foyer
  - Minimize
  - Equilibrate
  - Compress to contact
  - Shear at a range of normal loads

```
def initialize(job):
    "Initialize the simulation configuration."
    import hoomd
    if hoomd.context.exec_conf is None:
        hoomd.context.initialize('')
    with job:
        with hoomd.context.SimulationContext():
            n = ceil(pow(job.sp.N, 1/3))
            assert n**3 == job.sp.N
            hoomd.init.create_lattice(unitcell=hoomd.lattice.sc(a=1.0), n=n)
            hoomd.dump.gsd('init.gsd', period=None, group=hoomd.group.all())
```

```
def estimate(job):
    "Ideal-gas estimate operation."
    sp = job.statepoint()
    # Calculate volume using ideal gas law
    V = sp['N'] * sp['kT'] / sp['p']
    job.document['volume_estimate'] = V
```

```
def sample(job):
    "Sample operation."
    import hoomd
    from hoomd import md
```

VANDERBILT

Putting it all together: Signac workflow manager

- Customize compute environment
  - Defaults provided for common environments
  - Collection of environments for major computing centers
    - OLCF, NERSC, NICS
- Execute and monitor

```
class MyTorqueEnvironment(flow.environment.TorqueEnvironment):
    hostname_pattern = 'mymoabcluster.university.edu'
    cores_per_node = 16

    @classmethod
    def mpi_cmd(cls, cmd, np):
        return 'mpirun -np {np} {cmd}'.format(n=np, cmd=cmd)

    @classmethod
    def script(cls, _id, nn, walltime, ppn=None, **kwargs):
        if ppn is None:
            ppn = cls.core_per_node
        js = super(MyTorqueEnvironment, cls).script()
        js.writeline('#!/bin/sh -l')
        js.writeline('#PBS -j oe')
        js.writeline('#PBS -l nodes={}:ppn={}'.format(nn, ppn))
        js.writeline('#PBS -l walltime={}'.format(format_timedelta(walltime)))
        js.writeline('#PBS -q low')
        js.writeline('#PBS -N {}'.format(_id))
        js.writeline('#PBS -V')
```

```
>>> project = MyProject()
>>> project.print_status(detailed=True, params=('a',))
Status project 'test-project':
Total # of jobs: 10
label         progress
------------  ------------------------------------------------
initialized   |########################################| 100.00%
processed     |########################----------------| 66.67%
Detailed view:
job_id                           S  next_job  a  labels
-------------------------------- -  --------  -  ------------------------
108ef78ec381244447a108f931fe80db U            1  initialized, processed
be01a9fd6b3044cf12c4a83ee9612f84 U            2  initialized, processed
32764c28ef130baefebeba76a158ac4e U  process   3  initialized
# ...
>>>
```

VANDERBILT

Examples

- Basic example - bulk alkanes
  - https://github.com/mattwthompson/gromacs_signac_template
- More complex - screening functionalized monolayers for frictional properties
  - https://github.com/summeraz/terminal_group_screening

VANDERBILT

Homework assignment

- Lennard-Jones simulations using HOOMD
  - Directions and necessary files located on the ChBE 4830 Github
    - https://github.com/summeraz/chbe4830
    - Navigate to the `Assignment3` directory
  - Determine optimal timestep for an NVE simulation
  - Determine optimal neighborlist parameters
- Also:
  - Install mBuild and Foyer using Anaconda
  - Work through the tutorials provided at the mBuild website
    - http://mosdef-hub.github.io/mbuild/
  - We will be working through interactive tutorials on Thursday (9/7) so it will be important to have everything installed and to be familiar with the basic concepts of how mBuild operates