UMBC

Chemical Biochemical and Environmental Engineering

LEAN THEOREM PROVER

Lean for Scientists and Engineers

Tyler R. Josephson

AI & Theory-Oriented Molecular Science (ATOMS) Lab

University of Maryland, Baltimore County

Look at the Sky
Porter Robinson

Twitter: @trjosephson
Email: tjo@umbc.edu

Lean for Scientists and Engineers 2024

1. Logic and proofs for scientists and engineers
    1. Introduction to theorem proving
    2. Writing proofs in Lean
    3. Formalizing derivations in science and engineering
2. Functional programming in Lean 4
    1. Functional vs. imperative programming
    2. Numerical vs. symbolic mathematics
    3. Writing executable programs in Lean
3. Provably-correct programs for scientific computing

Schedule (tentative)

Logic and proofs for scientists and engineers
Functional programming in Lean 4
Provably-correct programs for scientific computing

July 9, 2024	Introduction to Lean and proofs
July 10, 2024	Equalities and inequalities
July 16, 2024	Proofs with structure
July 17, 2024	Proofs with structure II
July 23, 2024	Proofs about functions; types
July 24, 2024	Calculus-based-proofs
July 30-31, 2024	Prof. Josephson traveling
August 6, 2024	Functions, recursion, structures
August 7, 2024	Polymorphic functions for floats and reals; lists, arrays
August 13, 2024	Lists, indexing, Input / output, compiling Lean to C
August 14, 2024	Break
August 20, 2024	LeanMD & BET Analysis in Lean
August 21, 2024	SciLean tutorial, by Tomáš Skřivan

Content inspired by:
Mechanics of Proof, by Heather Macbeth
Functional Programming in Lean, by David Christiansen

Guest instructor: Tomáš Skřivan

Schedule for today

- Recap Lectures 1-9
- LeanBET
- LeanMD
- What do you want to build?

Errors in scientific computing software

Category of error	Example	Intervention
Syntax	Not closing parentheses	Editor

Errors in scientific computing software

Category of error	Example	Intervention
Syntax	Not closing parentheses	Editor
Runtime	Accessing element in list that doesn't exist	Run the program, program gives error message
Semantic	Missing a minus sign, transposing tensor indices	Human inspection of the code; test-driven development; observing anomalous behavior

Slide from Lecture 1

Errors in scientific computing software

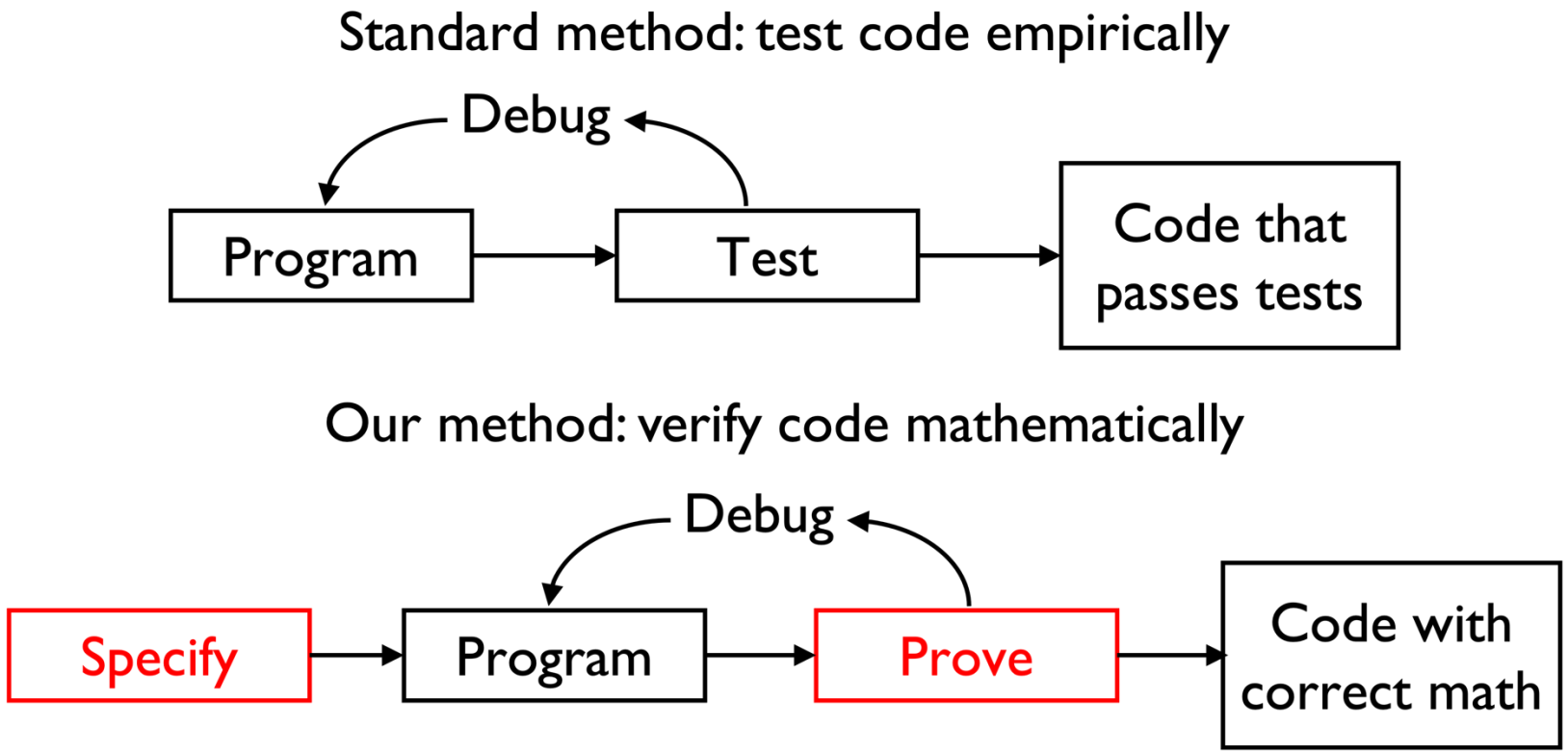
Category of error	Example	Intervention	Lean
Syntax	Not closing parentheses	Editor	Editor
Runtime	Accessing element in list that doesn't exist	Run the program, program gives error message	Editor
Semantic	Missing a minus sign, transposing tensor indices	Human inspection of the code; test-driven development; observing anomalous behavior	Editor

Errors in scientific computing software

Category of error	Example	Intervention	Lean
Syntax	Not closing parentheses	Editor	Editor
Runtime	Accessing element in list that doesn't exist	Run the program, program gives error message	Editor
Semantic	Missing a minus sign, transposing tensor indices	Human inspection of the code; test-driven development; observing anomalous behavior	Editor
Floating point / Round off	Subtracting small values from large values	Checking energy conservation	

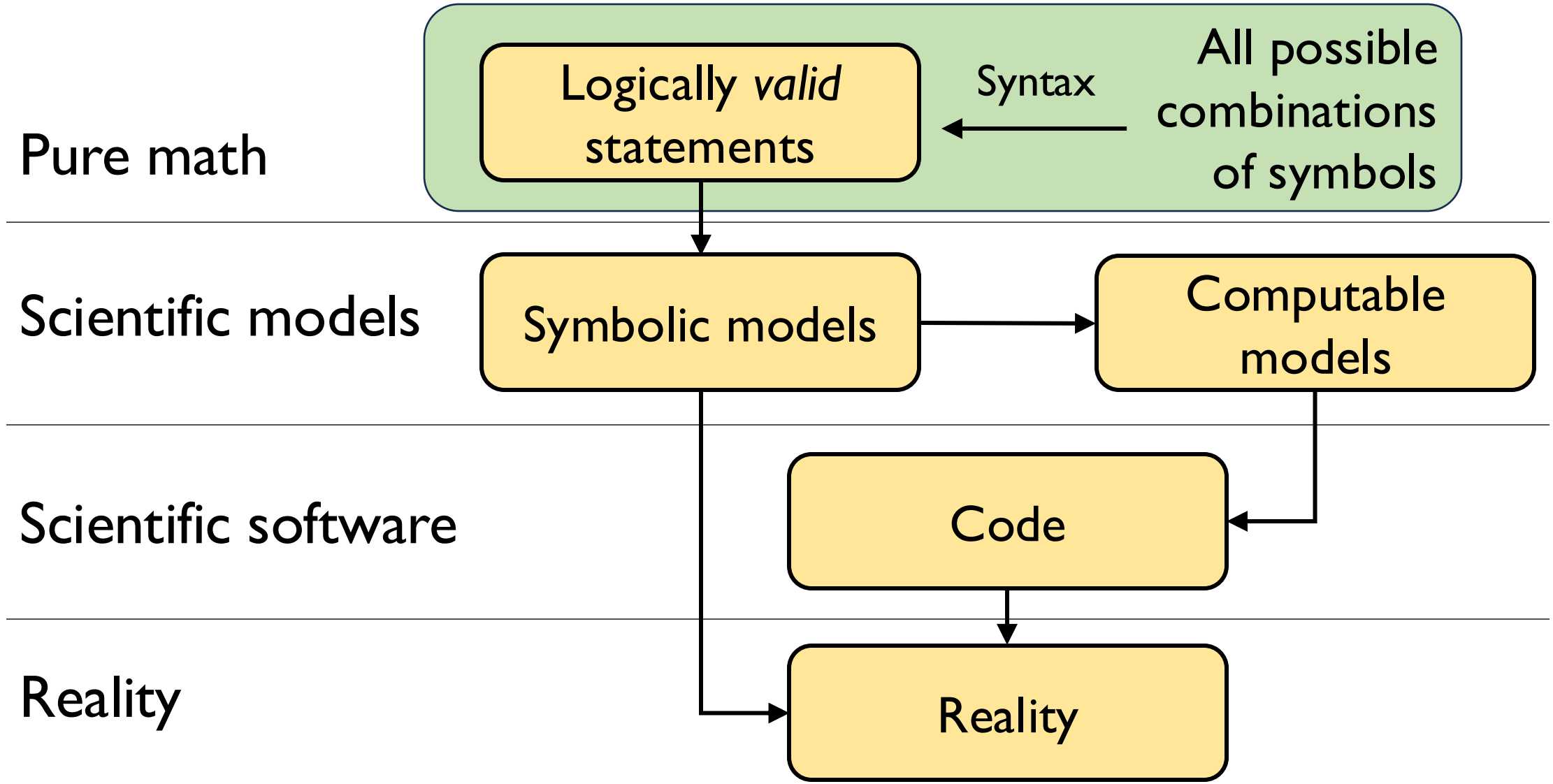
A vision for bug-free scientific computing

Selsam, Liang, Dill, "Developing Bug-Free Machine Learning Systems with Formal Mathematics," ICML 2017.



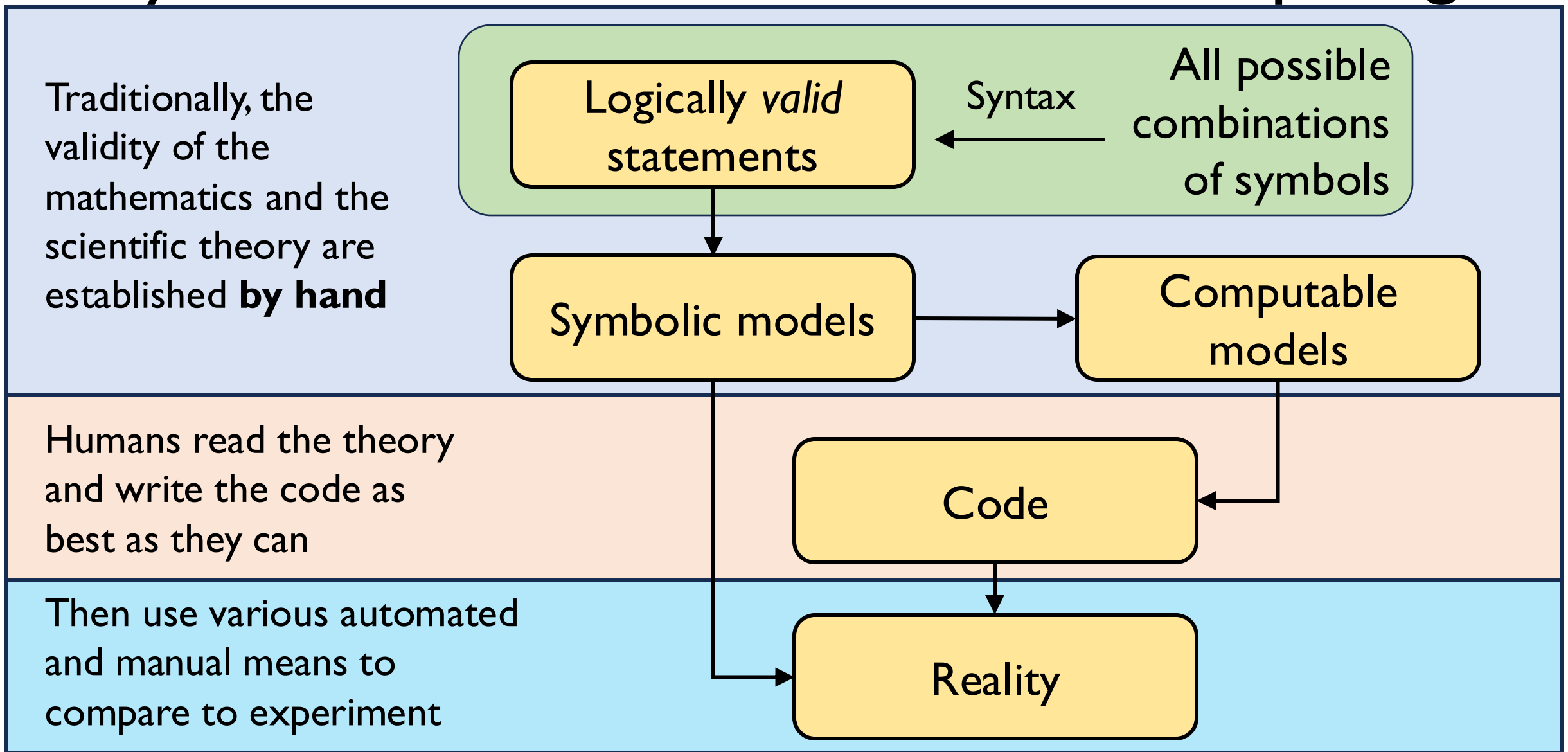
Slide from Lecture 3

Syntax and semantics in scientific computing



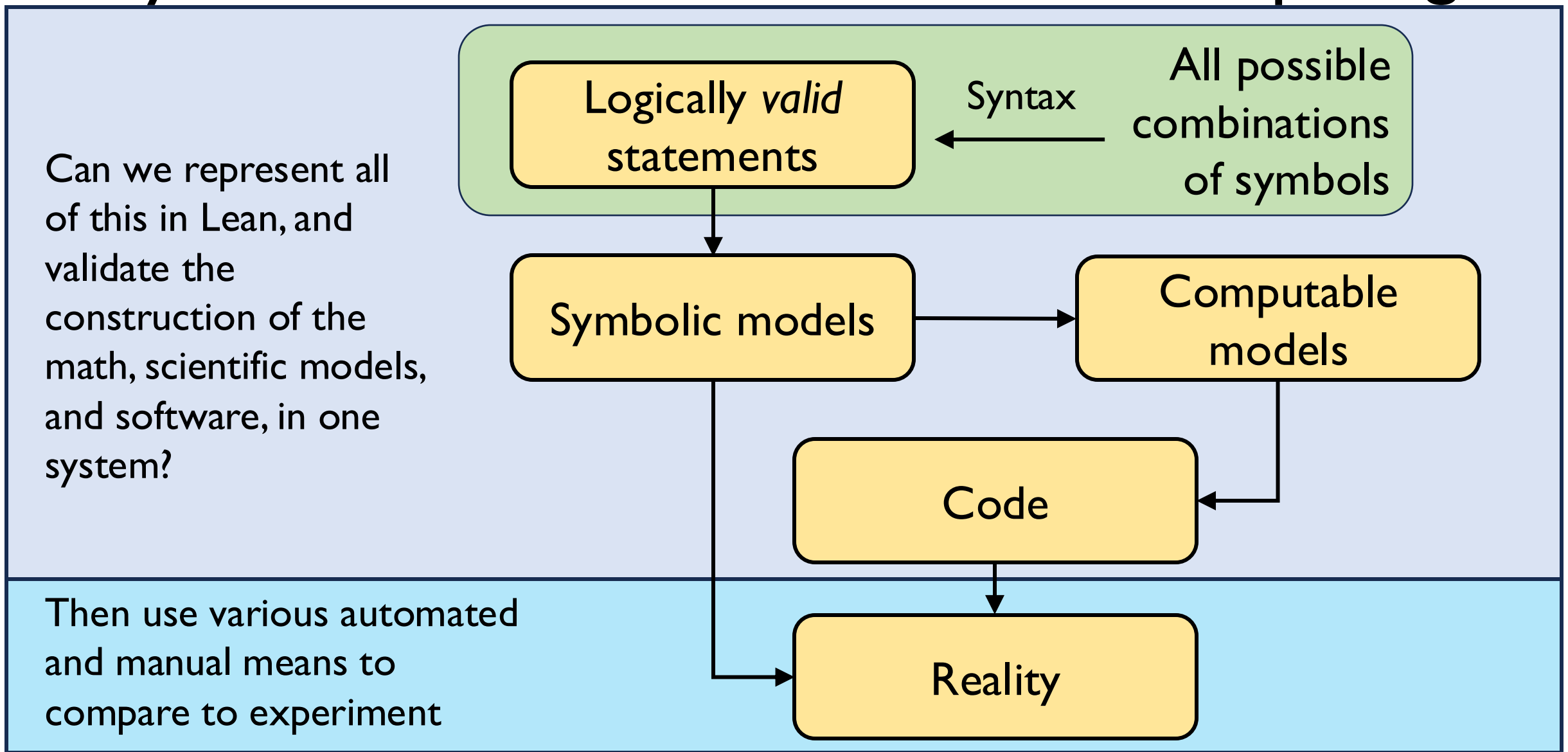
Slide from Lecture 3

Syntax and semantics in scientific computing



Slide from Lecture 3

Syntax and semantics in scientific computing

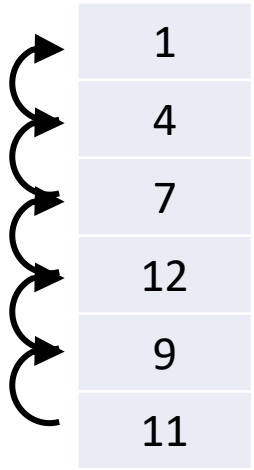


Slide from Lecture 8

Lists vs Arrays

A "list" in Lean is a linked list

A "list" in Python is an array!

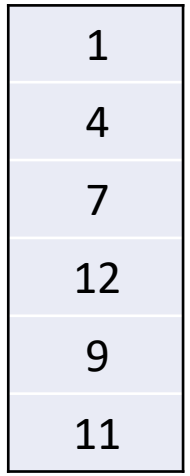


Linked Lists

- Each node is connected to the next node.
- Dynamic in size.
- Accessing an element requires traversal of whole list.
- Insertion and deletion is fast.
- Uses more memory than an array because it stores the next value as well.

Arrays

- Each element has an index which acts like an address in the array
- Fixed in size.
- Elements can be accessed easily.
- Insertion and deletion takes a lot of time.
- Uses less memory compared to a linked list.



https://medium.com/@bilal_k/wtf-is-linked-list-5d58b8a3bfe7

Slide from Lecture 8

Polymorphic functions

- *Polymorphism* is when a single symbol represents different types
- A *polymorphic function* takes variables that can be more than one type
- Python uses polymorphism (most languages do), so a relatively short list of familiar symbols can address diverse tasks

```
def plus(a,b):
    return a + b
```

```
plus(1,2)
3
```

```
plus(1.0,2.0)
3.0
```

```
plus('1','2')
'12'
```

```
plus([1],[2])
[1, 2]
```

```
plus(1.0,2)
3.0
```

Polymorphism in Python is ad hoc – under the hood, these are compiled as distinct functions

Slide from Lecture 8

Polymorphism in Lean

- In functional programming languages, polymorphism is made possible using generic types, which get inhabited by specific types based on context

- For example, let's revisit the structure Point from last time
- We can define a similar structure PPoint that's polymorphic (from FPIL 1.6)

```
structure Point where
  x : Float
  y : Float
deriving Repr
```

```
structure PPoint (α : Type) where
  x : α
  y : α
deriving Repr
```

Polymorphism to combine theory and computation

```
-- Langmuir Adsorption Equation
def Langmuir [Mul α] [Add α] [Div α] [One α] (K_eq : α ) (P : α ) : α :=
  K_eq * P / (1 + K_eq*P)

-- We can't do calculations!
#eval Langmuir 5.0 9.0

-- We can do proofs

theorem LangmuirAdsorption {θ K P r_ad r_d k_ad k_d A S_tot S : ℝ}
  (hrad : r_ad = k_ad * P * S) -- Adsorption rate expression
  (hrd : r_d = k_d * A)        -- Desorption rate expression
  (heq : r_ad = r_d)           -- Equilibrium assumption
  (hK : K = k_ad / k_d)        -- Definition of adsorption constant
  (hS_tot : S_tot = S + A)     -- Site balance
  (hθ : θ = A / S_tot)         -- Definition of fractional coverage
  -- Physical constraints
  (hc1 : S + A ≠ 0)
  (hc2 : k_d + k_ad * P ≠ 0)
  (hc3 : k_d ≠ 0) :
  θ = Langmuir K P := by
  dsimp [Langmuir]
  rw [hrad, hrd] at heq
  rw [hθ, hS_tot, hK]
  field_simp
  calc
    A * (k_d + k_ad * P) = k_d * A + k_ad * P * A := by ring
    _ = k_ad * P * S + k_ad * P * A := by rw[heq]
    _ = k_ad * P * (S + A) := by ring
```

Slide from Lecture 9

Input / Output: An Analogy

Kitchen (back of house)	Waiter	Dining room (front of house)

Pure functions Mathlib Verified logical syntax	IO Monad	Messy, unpredictable real world

Slide from Lecture 1

Adsorption

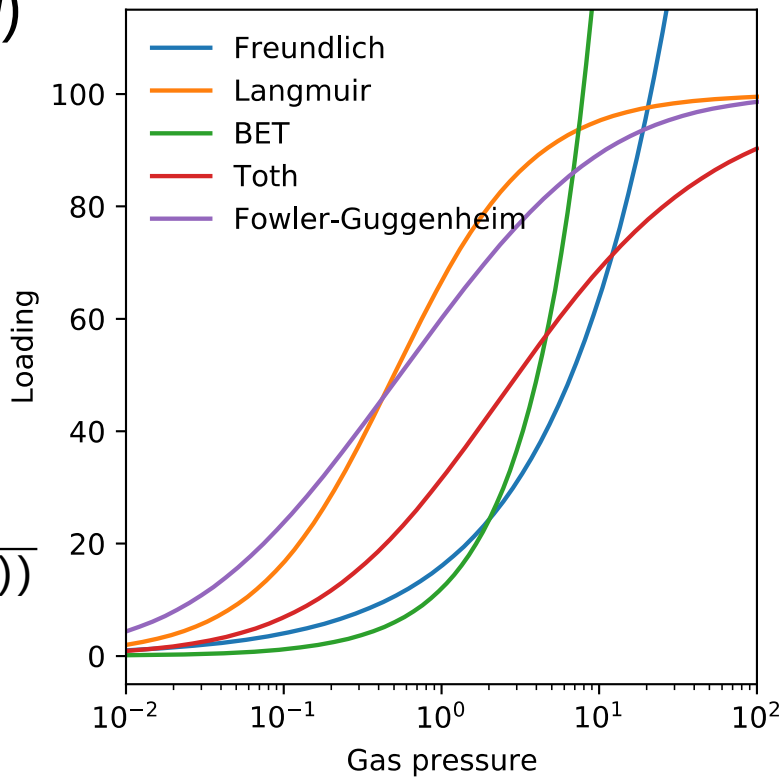
When molecules from a gas or liquid "stick" onto a solid material

a)

b)

c)

d)



e) Freundlich $q = K_F p^n$

Langmuir $q = \frac{q_{max} K_L p}{1 + K_L p}$

BET $q = \frac{q_m c_{BET} p}{(p_0 - p)(1 + (c_{BET} - 1)(p/p_0))}$

Toth $q = \frac{q_{max} p}{(b + p^t)^{1/t}}$

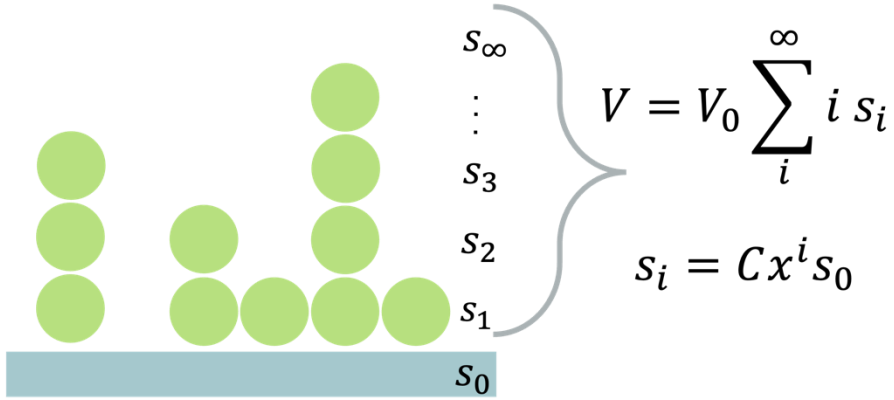
Fowler-Guggenheim $K_{FG} p = \frac{\checkmark}{1 - \checkmark} \exp\left(\frac{2\checkmark w}{RT}\right)$

Adsorption of Gases in Multimolecular Layers

Brunauer, Emmett, and Teller (yes, the one from the Manhattan Project)

1938

36000+ citations (Google Scholar)



BET Adsorption

Loading = f(p)

$$q = \frac{v_m c p}{(p_0 - p)(1 + (c-1)(p/p_0))}$$

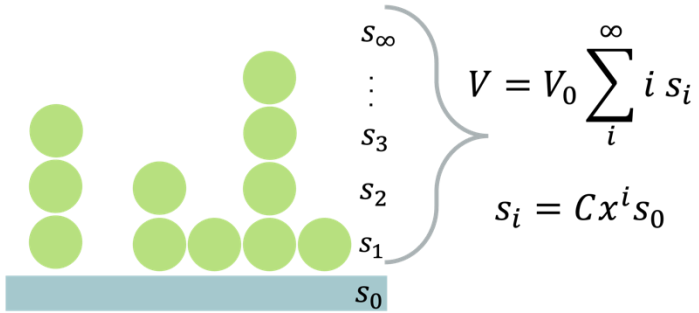
Linearized form

$$\frac{p}{q(p_0 - p)} = \frac{1}{v_m} + \frac{c-1}{v_m c}\frac{p}{p_0}$$

Formalizing Chemical Physics in the Lean Theorem Prover

- It was hard to sort out "what are assumptions" from "what are intermediate steps"
- You also have some bit of freedom around where you start the formalization process
- We eventually sorted this into 6 foundational premises
  - https://github.com/ATOMSLab/LeanBET/blob/main/BET/BETInfinite.lean
- Proved that Eq. 26 and Eq. 28 from Brunauer, et al. follow from these premises

Adsorption Analysis using BET Theory



BET Adsorption

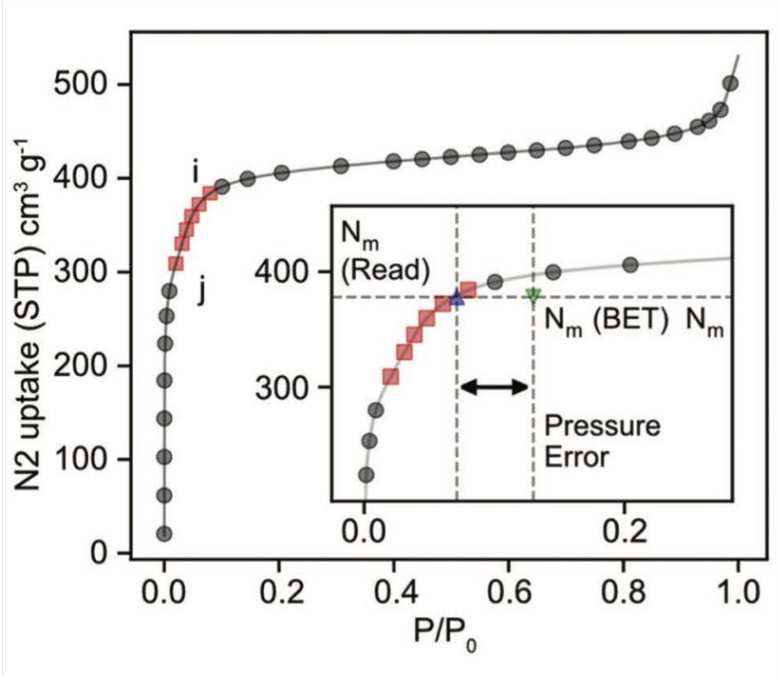
Osterrieth, et al. Adv. Mat. 2022

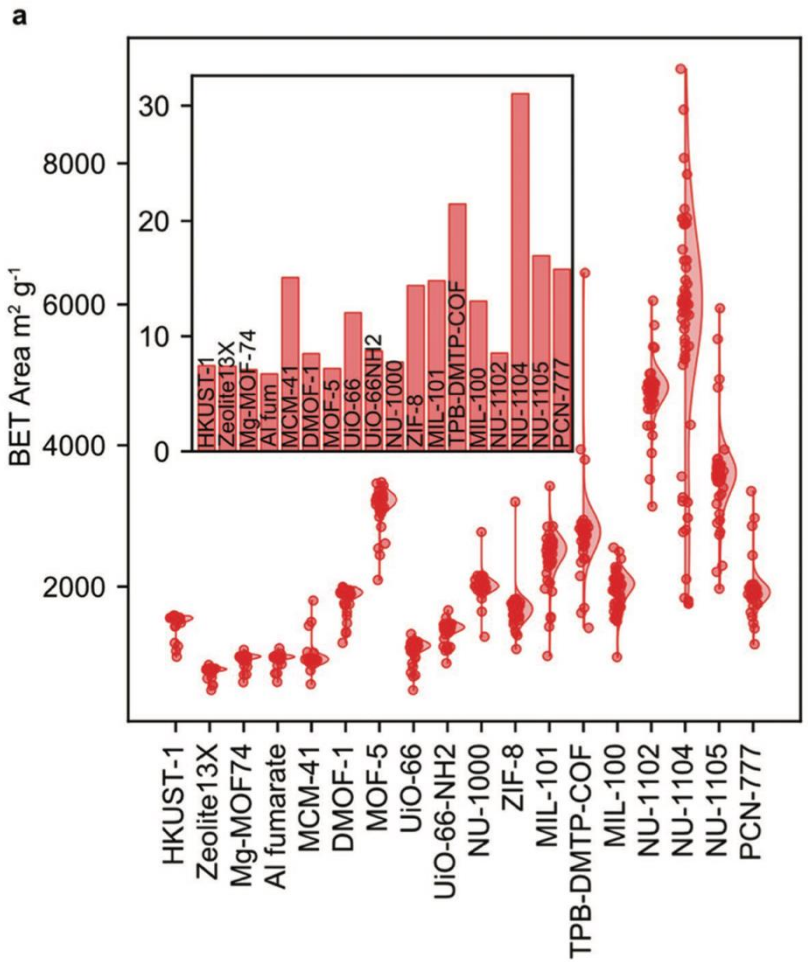
Loading = f(p)

$$q = \frac{v_m c p}{(p_0 - p)(1 + (c-1)(p/p_0))}$$

Linearized form

$$\frac{p}{q(p_0 - p)} = \frac{1}{v_m} + \frac{c-1}{v_m c}\frac{p}{p_0}$$





Roquerol Criteria

Osterrieth, et al. Adv. Mat. 2022

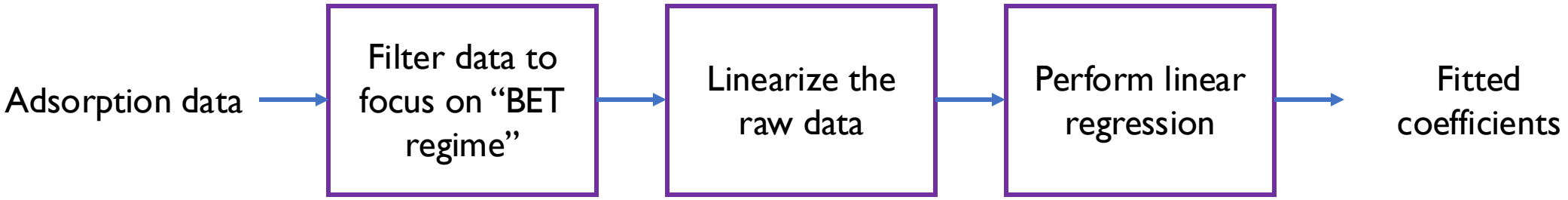
Open paper

Python code available here:

https://github.com/nakulrampal/betsi-gui

Slide from Lecture 1

Bug-Free BET Analysis



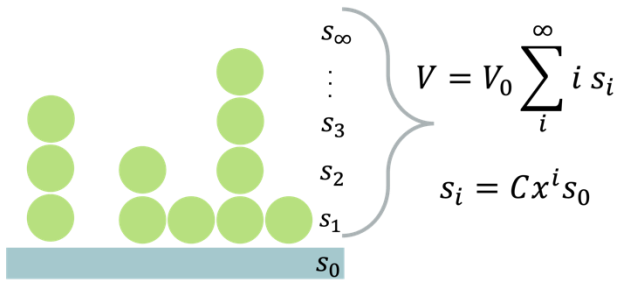
Bug-Free BET Analysis

Adsorption data → Filter data to focus on "BET regime" → Linearize the raw data → Perform linear regression → Fitted coefficients

Formal proof of BET Theory

$$q = \frac{v_m c p}{(p_0 - p)(1 + (c-1)(p/p_0))}$$

follows from a body of assumptions about



$$V = V_0 \sum_i^\infty i\, s_i$$

$$s_i = C x^i s_0$$

BET Adsorption

Proof that algebra for linearization is correct

Proof that linear regression minimizes least squares error

Proof that output corresponds to meaningful parameters

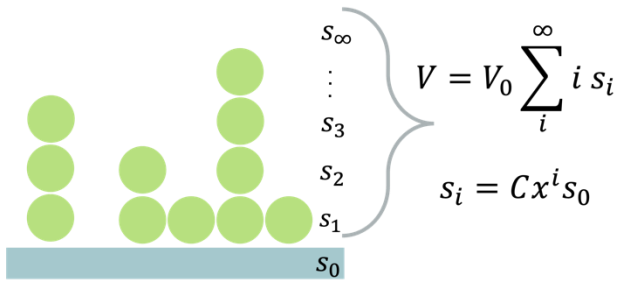
Bug-Free BET Analysis

Adsorption data → Filter data to focus on "BET regime" → Linearize the raw data → Perform linear regression → Fitted coefficients

Formal proof of BET Theory

$$q = \frac{v_m c p}{(p_0 - p)(1 + (c-1)(p/p_0))}$$

follows from a body of assumptions about



$$V = V_0 \sum_i^\infty i\, s_i$$

$$s_i = C x^i s_0$$

BET Adsorption

Proof that algebra for linearization is correct

Proof that linear regression minimizes least squares error

Proof that output corresponds to meaningful parameters

Polymorphic functions to bridge floats and reals

Adsorption data → Filter data to focus on "BET regime" → Linearize the raw data → Perform linear regression → Fitted coefficients

Floating point numbers

Polymorphic functions

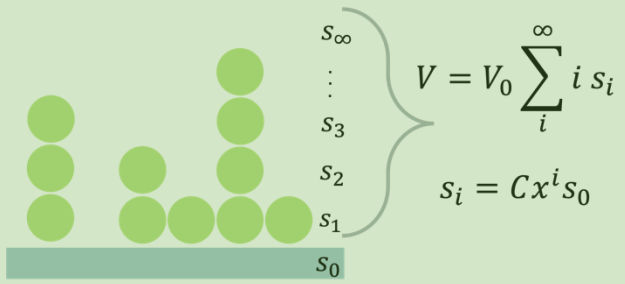
Real numbers

$\mathbb{R}$

Formal proof of BET Theory

$$q = \frac{v_m c p}{(p_0 - p)(1 + (c-1)(p/p_0))}$$

follows from a body of assumptions about



BET Adsorption

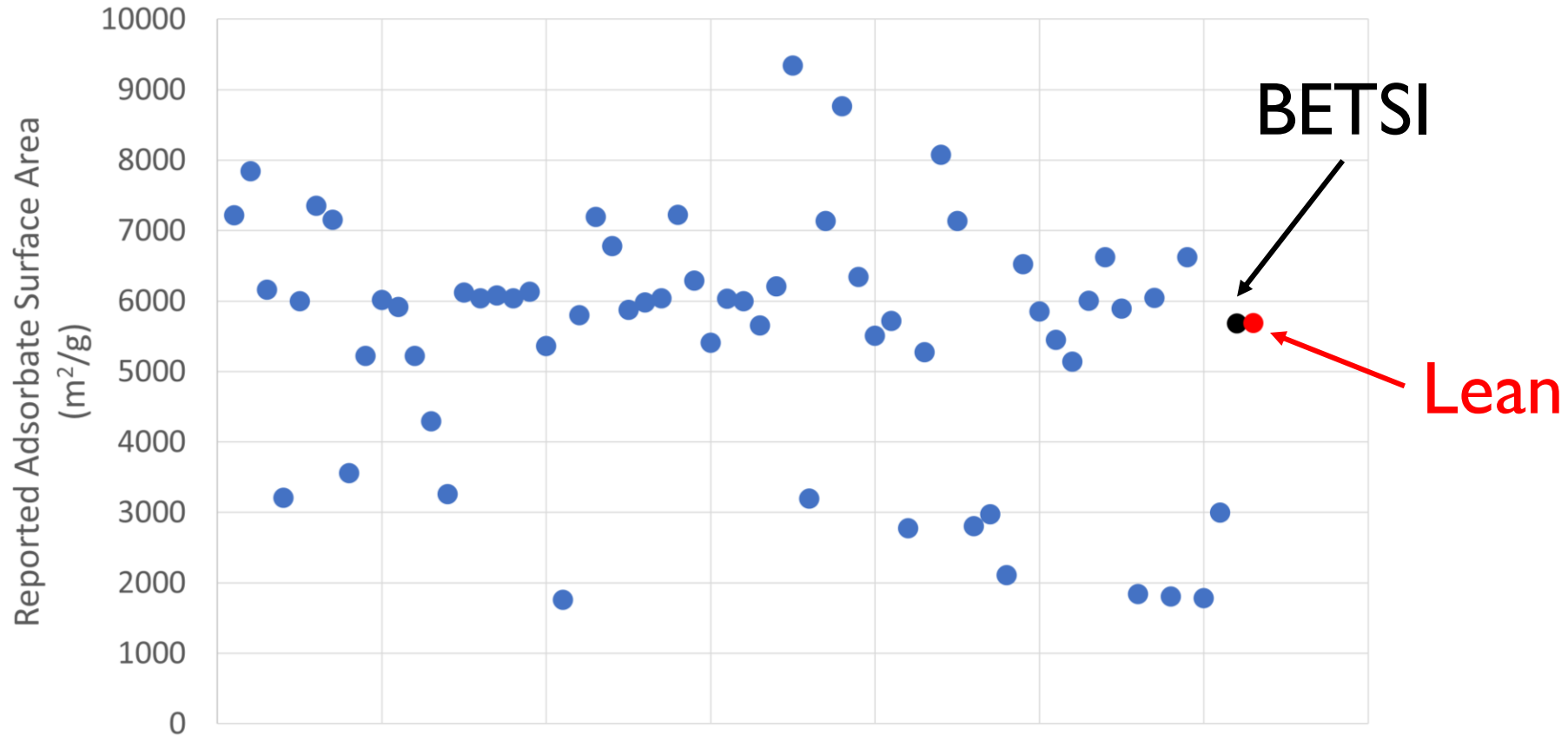
Proof that algebra for linearization is correct

Proof that linear regression minimizes least squares error

Proof that output corresponds to meaningful parameters

Regression with Lean matches BETSI standard

Osterrieth, et al. Adv. Mat. 2022



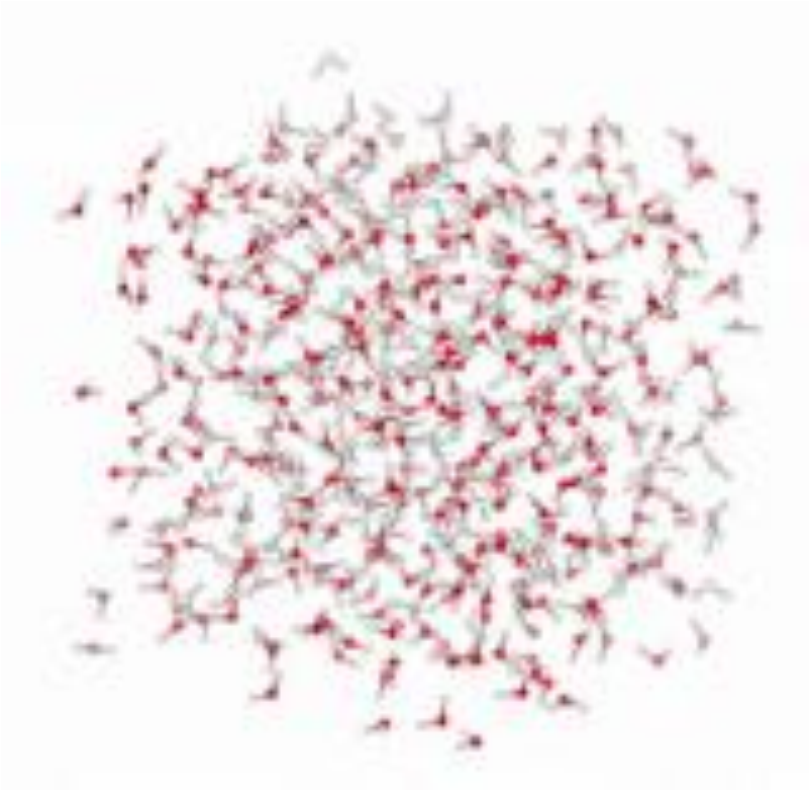
Caveat: Filtering applied in BETSI

Go to code

- Right now (no guarantees about future code versions)
  1. Run BET-CSV-examples.lean
  2. Importing StreamRead – triggers the main function in StreamRead
  3. Calls CSV_LRM_Model
  4. In StreamRead, main calls process
  5. process calls dump in StreamRead
  6. dump calls parse from CSVCat
     1. This is applied recursively and shrinks the buffer
     2. Strings get converted into UTF8, then floats
  7. Once the buffer is empty, call LRprocess to start doing the math
  8. linReg does the math and handles exceptions
  9. linReg returns LRM_Out

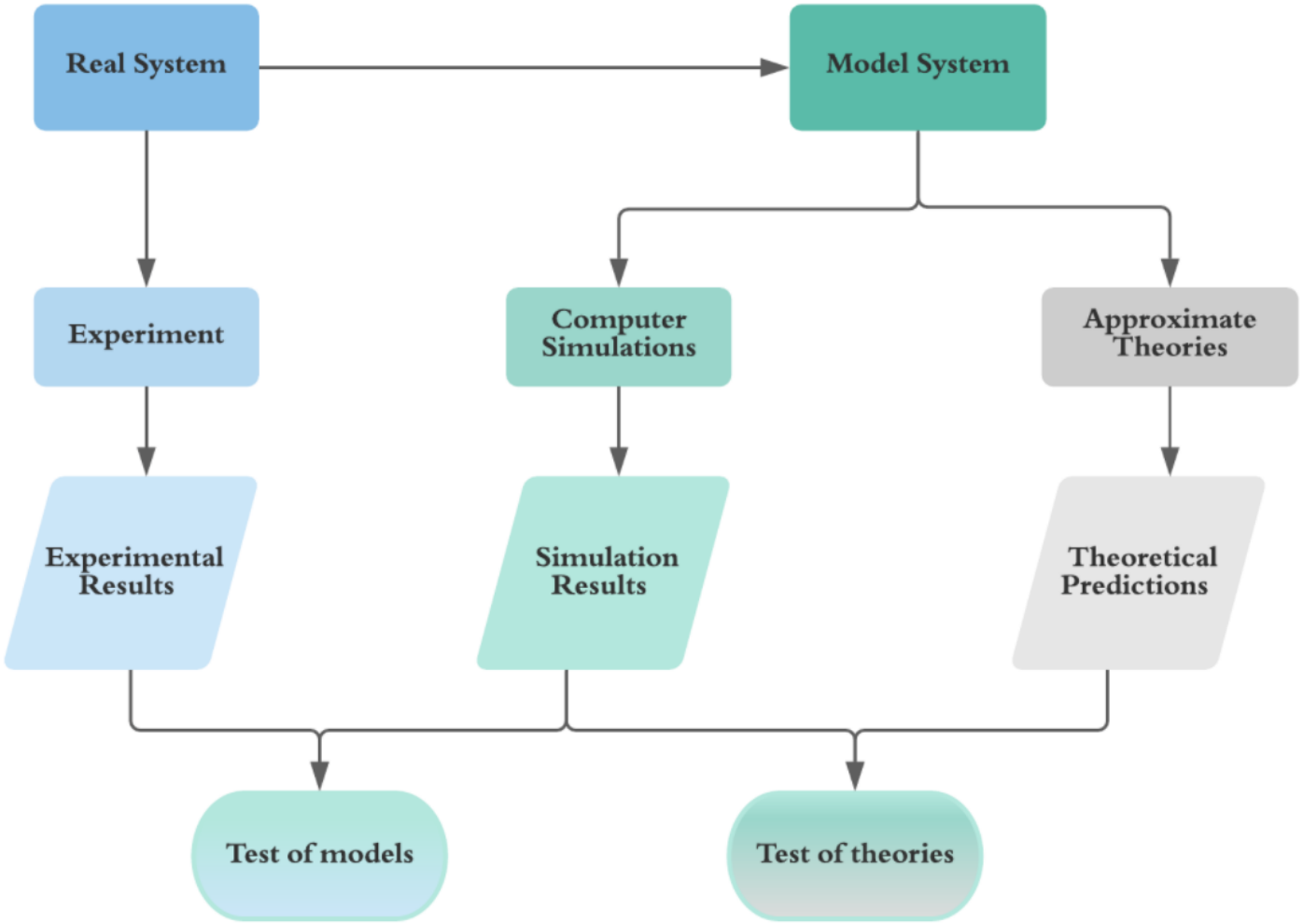
Slides from Samiha Sharlin

Molecular simulations



Try in your browser!
https://physics.weber.edu/schroeder/md/InteractiveMD.html

Simulations are "*computer experiments*"



Adapted from *Computer Simulation of Liquids* by M. P. Allen and D. J. Tildesley

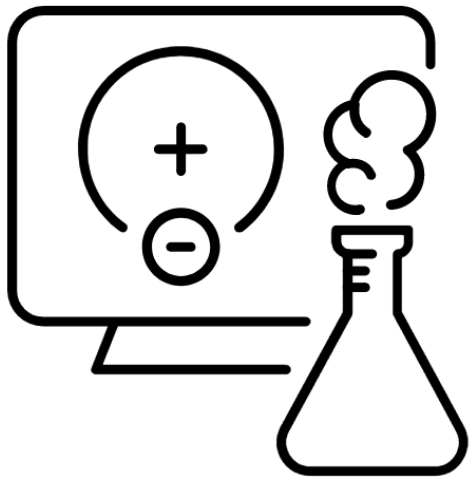
Molecular simulation is a computational "*experiment*" conducted on a *molecular model*

Simulation results are compared to experimental results to test the effectiveness of models

It has the characteristics of both theory and experiment

Simulations can be compared with predictions of analytical theory to test their validity

Why do molecular simulations?



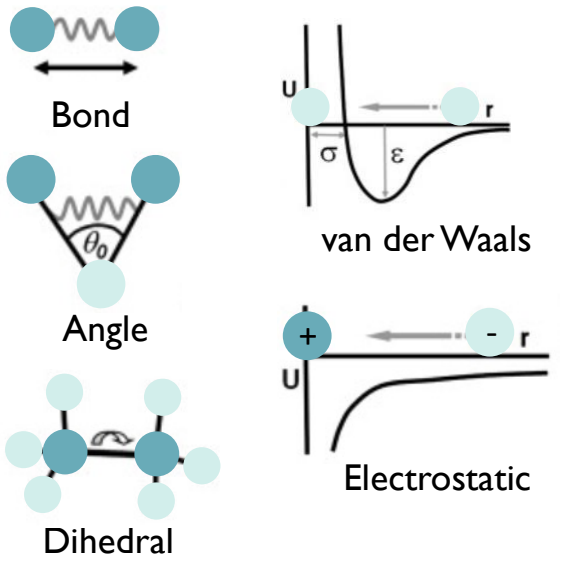
Molecular simulation is the only means for accurately determining the thermophysical properties of a molecular model system

Computer simulations help us to visualize events that have not taken place in real or are impossible to observe experimentally

High throughput simulations require less effort per analysis when compared to traditional means of experimentation

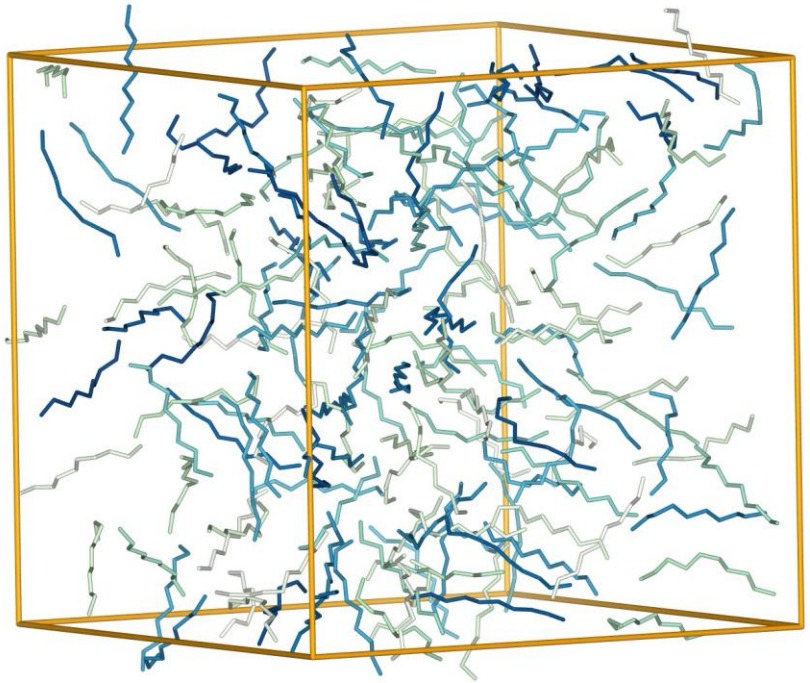
How do molecular simulations work?

Molecular Modeling



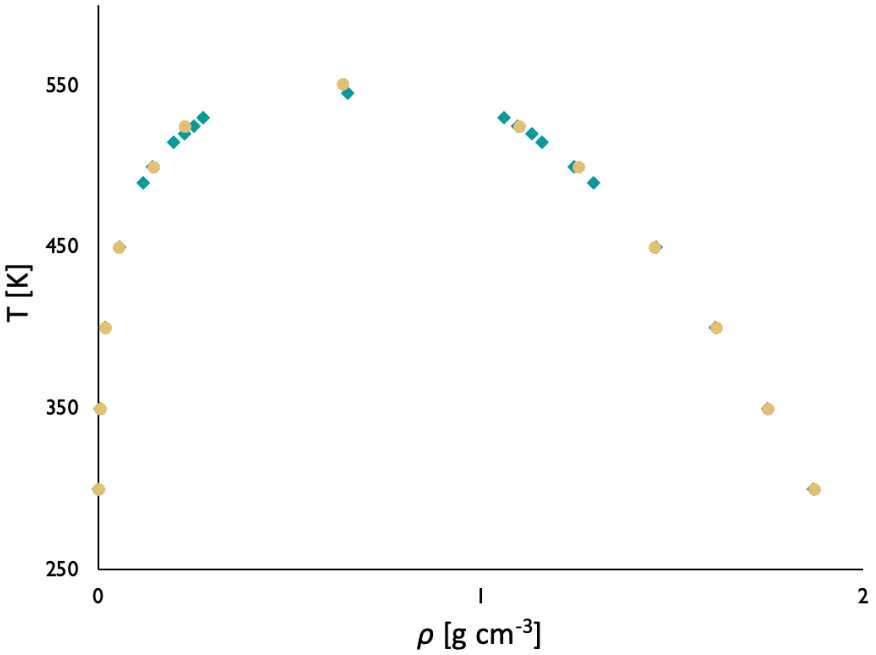
Modeling individual particles using force fields or quantum mechanics

Configuration Sampling



Computing movement of model particles

Data Analyzing

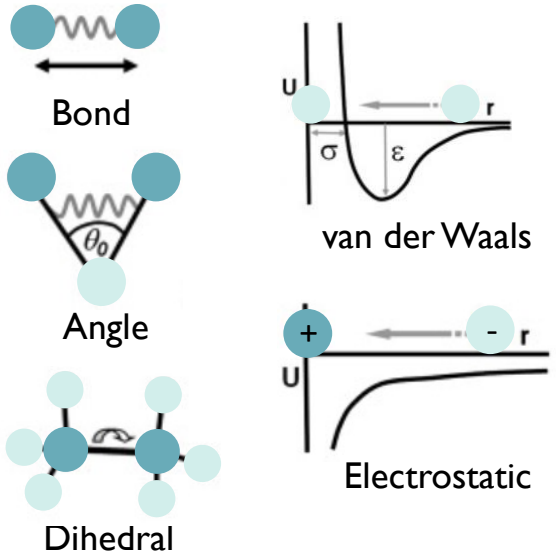


Analyzing simulation data to estimate thermodynamic properties

Molecular Modeling

" ..all things are made of atoms, and that everything that living things do can be understood in terms of the jigglings and wigglings of atoms."[1]

-- Richard Feynman



Quantum Mechanics

Uses electronic structure to compute forces by solving the Schrödinger equation

Force Fields

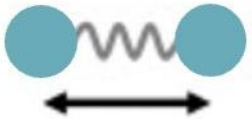
Uses simple empirical functions to model, most commonly as a sum of bonded and nonbonded interactions

1. The Feynman Lectures On Physics; Addison-Wesley, 1963; Vol. 1.

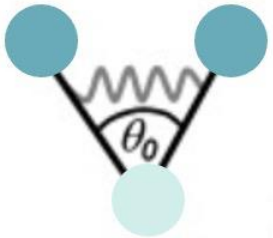
Force Fields

$U \rightarrow$ energy
$b \rightarrow$ bond length
$b_{eq} \rightarrow$ equilibrium bond length
$\theta \rightarrow$ bond angle
$\theta_{eq} \rightarrow$ equilibrium bond angle
$k \rightarrow$ force constant
φ = dihedral angle

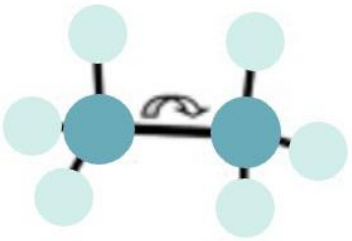
$$E_{bonded} = E_{bond} + E_{angle} + E_{dihedral}$$



Bond



Angle



Dihedral

Harmonic Potentials

$$U_{bonds}(b) = \frac{k_b}{2}(b - b_{eq})^2$$

Harmonic Potentials

$$U_{bends}(\theta) = \frac{k_\theta}{2}(\theta - \theta_{eq})^2$$

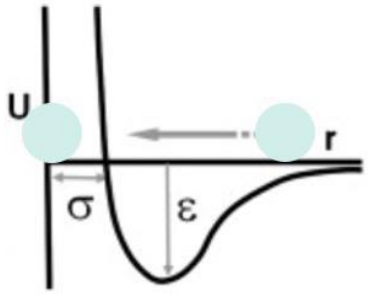
Cosine Functions

$$U_{torsions}(cosine\ \varphi)$$

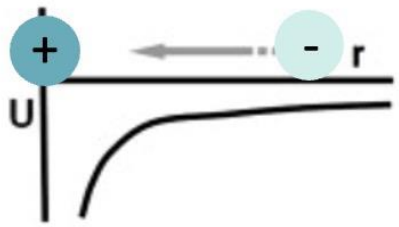
Force Fields

U → energy
r_{ij} → site-site separation
ε_{ij} → LJ well depth
σ_{ij} → LJ distance
q →partial charge
ε_0 → absolute permittivity

$$E_{\text{non-bonded}} = E_{\text{van der Waals}} + E_{\text{electrostatic}}$$



van der Waals



Electrostatic

Lennard-Jones (LJ)

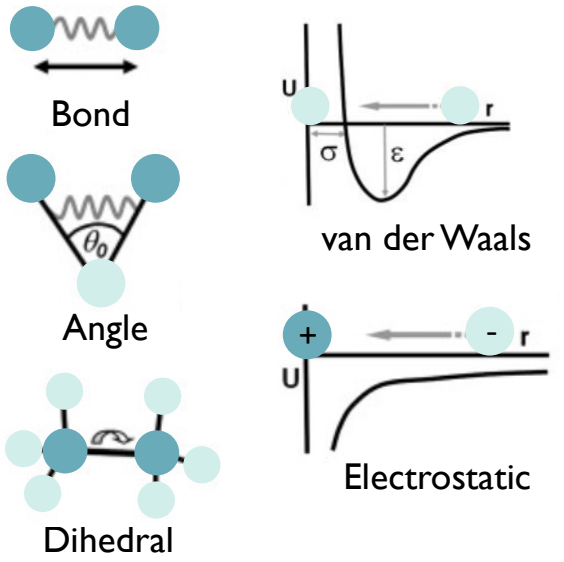
$$U(r_{ij}) = 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

Coulomb Potentials

$$U(r_{ij}) = \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

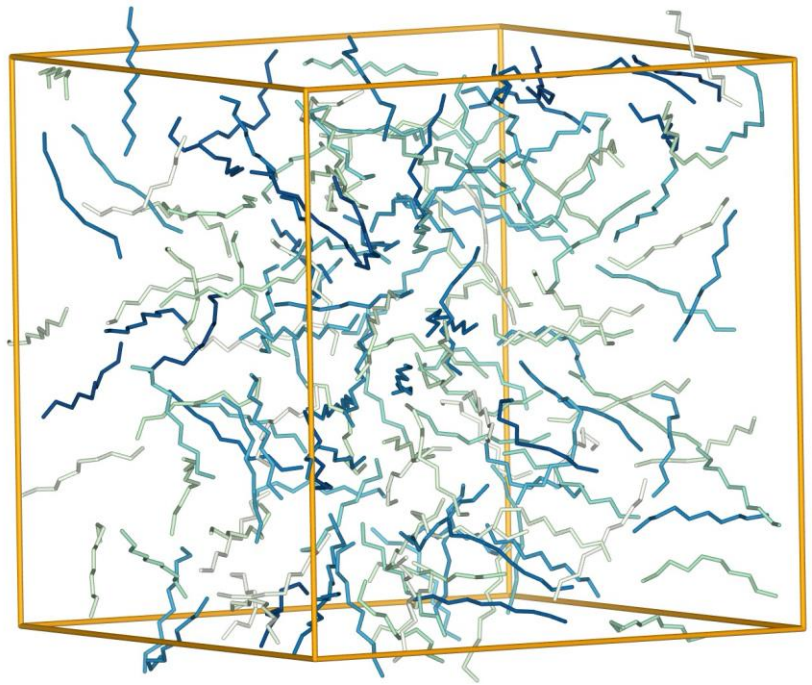
How do molecular simulations work?

Molecular Modeling



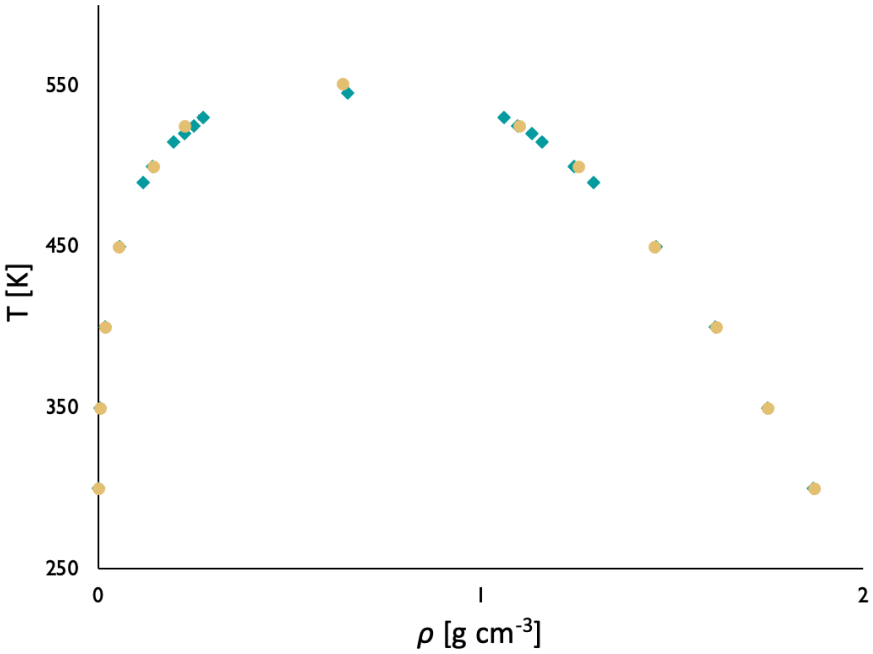
Modeling individual particles using force fields or quantum mechanics

Configuration Sampling



Computing movement of model particles

Data Analyzing



Analyzing simulation data to estimate thermodynamic properties

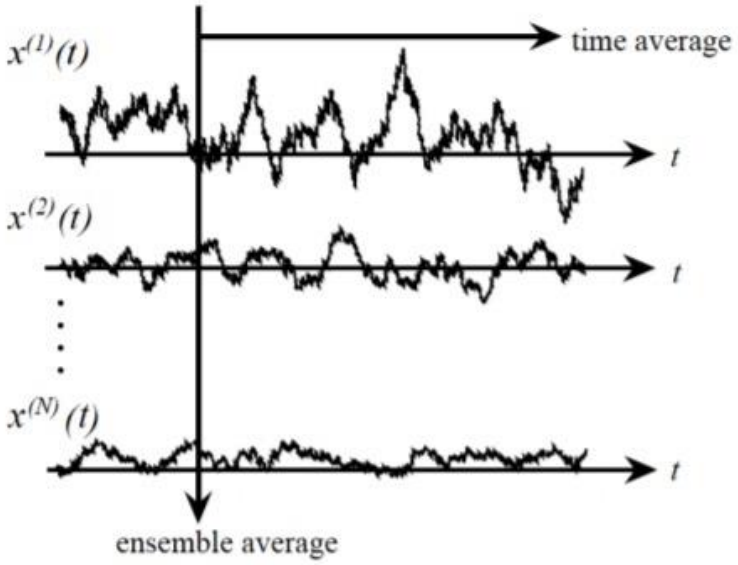
Simulation types

$A \rightarrow obervable\ variable$
$p \rightarrow momentum$
$r \rightarrow position$
$t \rightarrow time\ step$
$N \rightarrow dimension$
$\wp \rightarrow probability\ distibution$

Molecular Dynamics (MD)

Time average is the averaged quantity of a single system over a time interval

$$\langle A \rangle_t = \frac{1}{t} \int_0^t \mathbf{dt}\, A\,[r^N(t), p^N(t)]$$



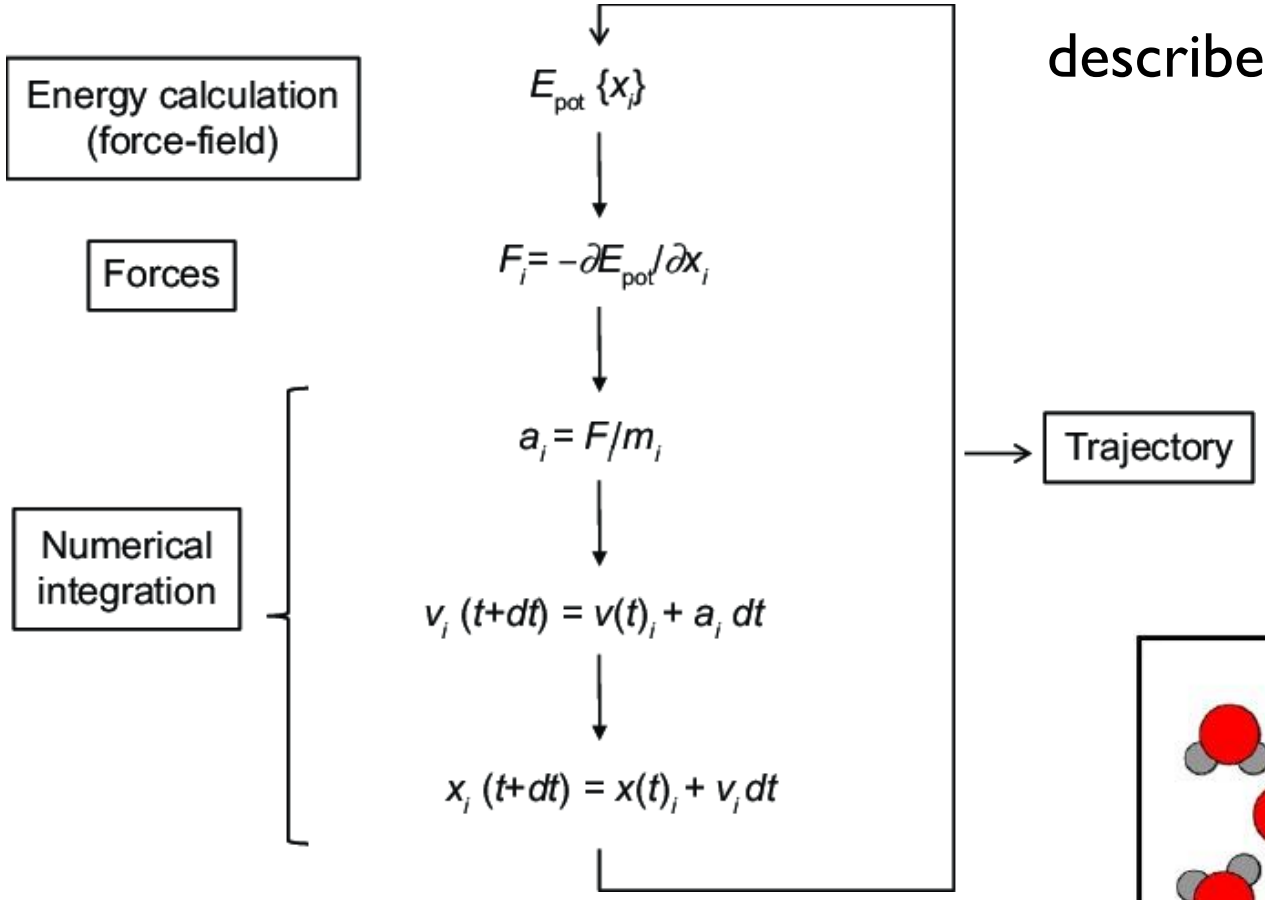
Fundamentals of Noise Processes by Yoshihisa Yamamoto

Monte Carlo (MC)

Ensemble average is the averaged quantity of many identical systems at a certain time using the probability distribution function of the systems as measure

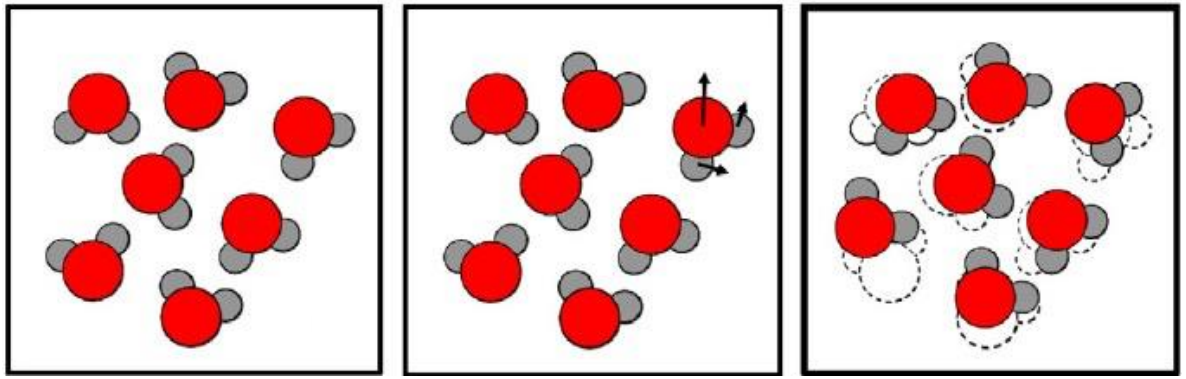
$$\langle A \rangle = \int \mathbf{dp^N\, dr^N}\, \wp(p^N, r^N)\, A(p^N, r^N)$$

MD trajectories



MD is the time evolution of atomic systems described by Newton's laws of motions

$$\langle A \rangle_t = \frac{1}{t}\int_0^t \mathbf{dt}\, A\,[r^N(t), p^N(t)]$$



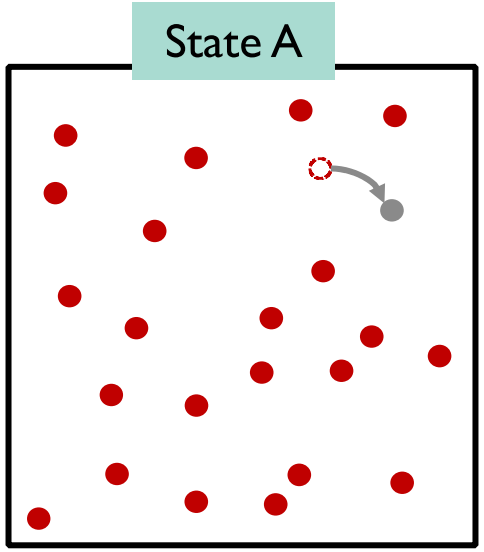
MC moves

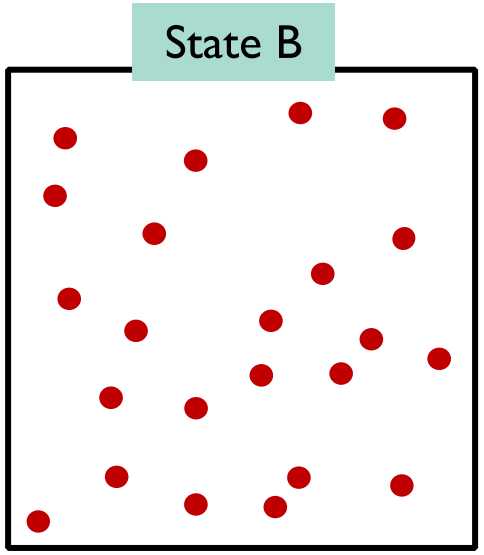
Generate new configuration by making perturbations to present configuration

Compute the change in potential energy, $\Delta U = U_B - U_A$

Accept or reject the new configuration based on *Metropolis* criterion

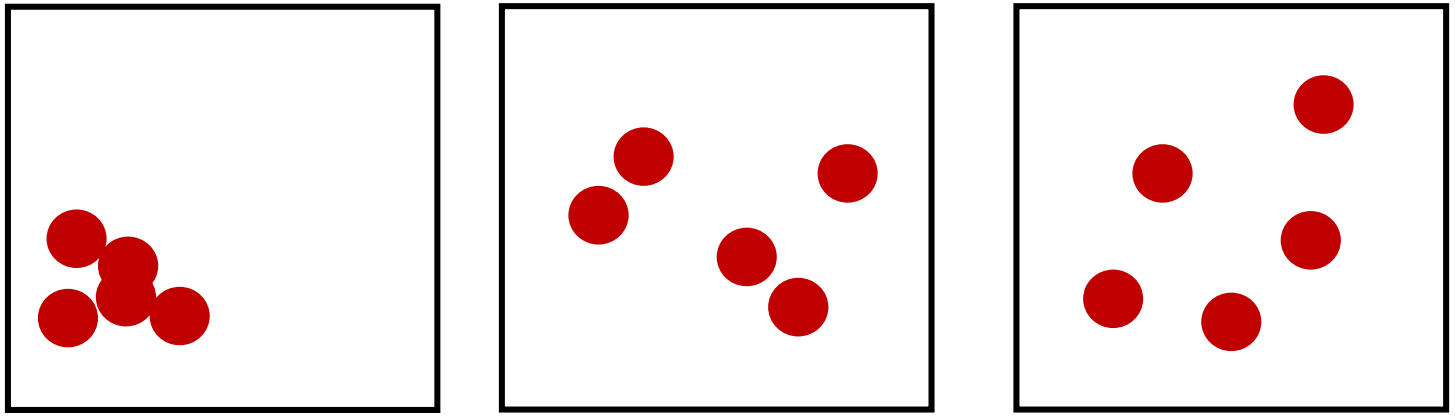
If the trial is accepted, update running averages with it; if rejected, the original configuration is updated with the running average





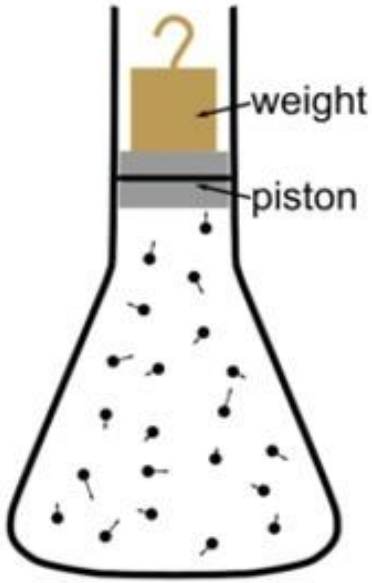
Ensemble

It is a collection of systems or microstates in the phase space at a certain time and is distributed according to a *probability density function* consistent with the macroscopic constraints defining the system (NPT, NVT, NVE, etc.)



Canonical (NVT)

Grand canonical (μVT)



Isothermal – Isobaric (NPT)

Images from Wikipedia:
https://en.wikipedia.org/wiki/Statistical_ensemble_(mathematical_physics)

Canonical partition function is the normalizing factor for these probabilities:

$$Q_{NVT} = \frac{1}{N!}\frac{1}{h^{3N}}\int\int d\mathbf{p}^N d\mathbf{r}^N \exp\left[-\frac{H(\mathbf{p}^N,\mathbf{r}^N)}{k_B T}\right]$$

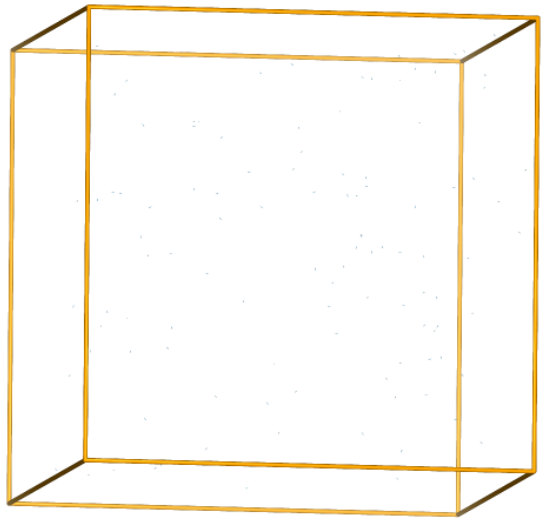
4 (+#,)#)is the Hamiltonian corresponding to the system's *total energy* which is a function of configurational space (*3N* positions and *3N* momenta). It can be written as the sum of *kinetic* and *potential energies* of the system:

$$H(\mathbf{p}^N,\mathbf{r}^N) = \sum_{i=1}^{N}\frac{|\mathbf{p}_i|^2}{2m} + \mathcal{U}(\mathbf{r}^N)$$

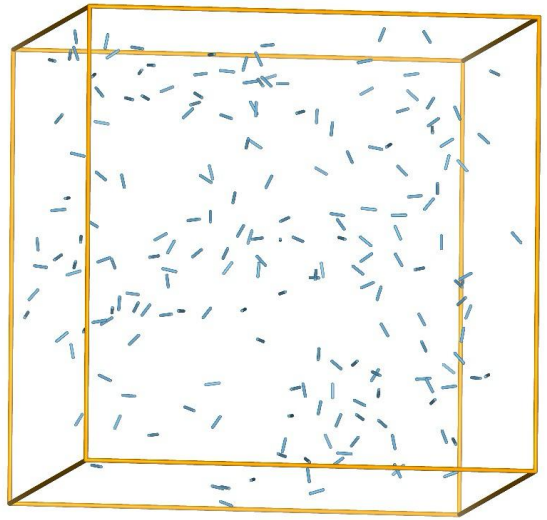
The *momentum integral* can be solved analytically: $\int d\mathbf{p}^N \exp\left[-\frac{|\mathbf{p}|^2}{2mk_BT}\right] = (2\pi m k_B T)^{3N/2}$

Thus, we have: $Q_{NVT} = \frac{1}{N!}\frac{1}{h^{3N}}(2\pi m k_B T)^{3N/2}\int d\mathbf{r}^N \exp\left[-\frac{\mathcal{U}(\mathbf{r}^N)}{k_B T}\right]$

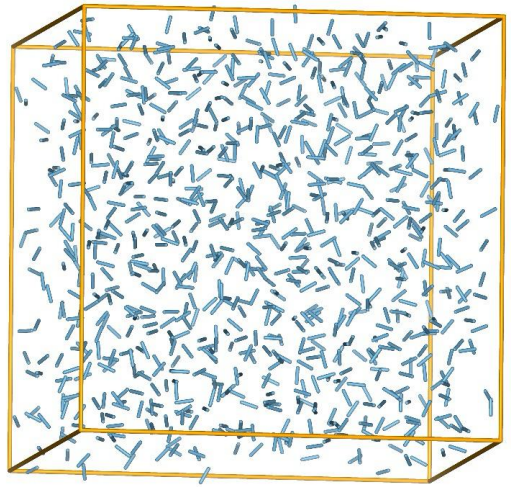
Simulation snapshots



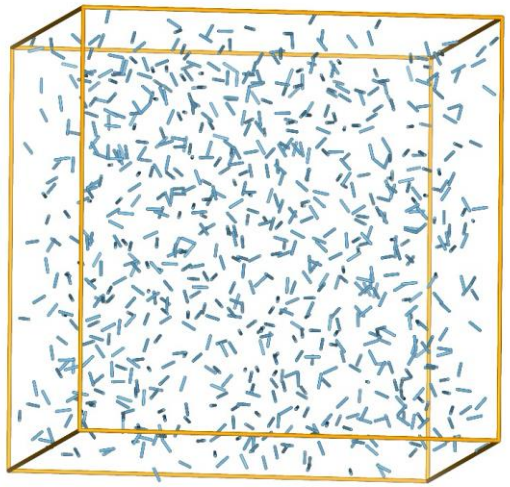
Gas phase at T = 160 K
Box length = 262 Å



Gas phase at T = 300K
Box length = 39 Å



Liquid phase at T = 160 K
Box length = 35 Å



Liquid phase at T = 300 K
Box length = 43 Å

Eight independent simulations
with *40,000 MC cycles* for *equilibration* and
50,000 MC cycles for *production*
for a system size of *1000* molecules*

Molar volume of *vapor box drops*
as temperature rises

Drastic change observed
near critical region

Vapor box length at 160 K
is ~ *7 times larger* than that at 300 K

*Simulation runs above 300 K required 2000 molecules

Applications of molecular simulation

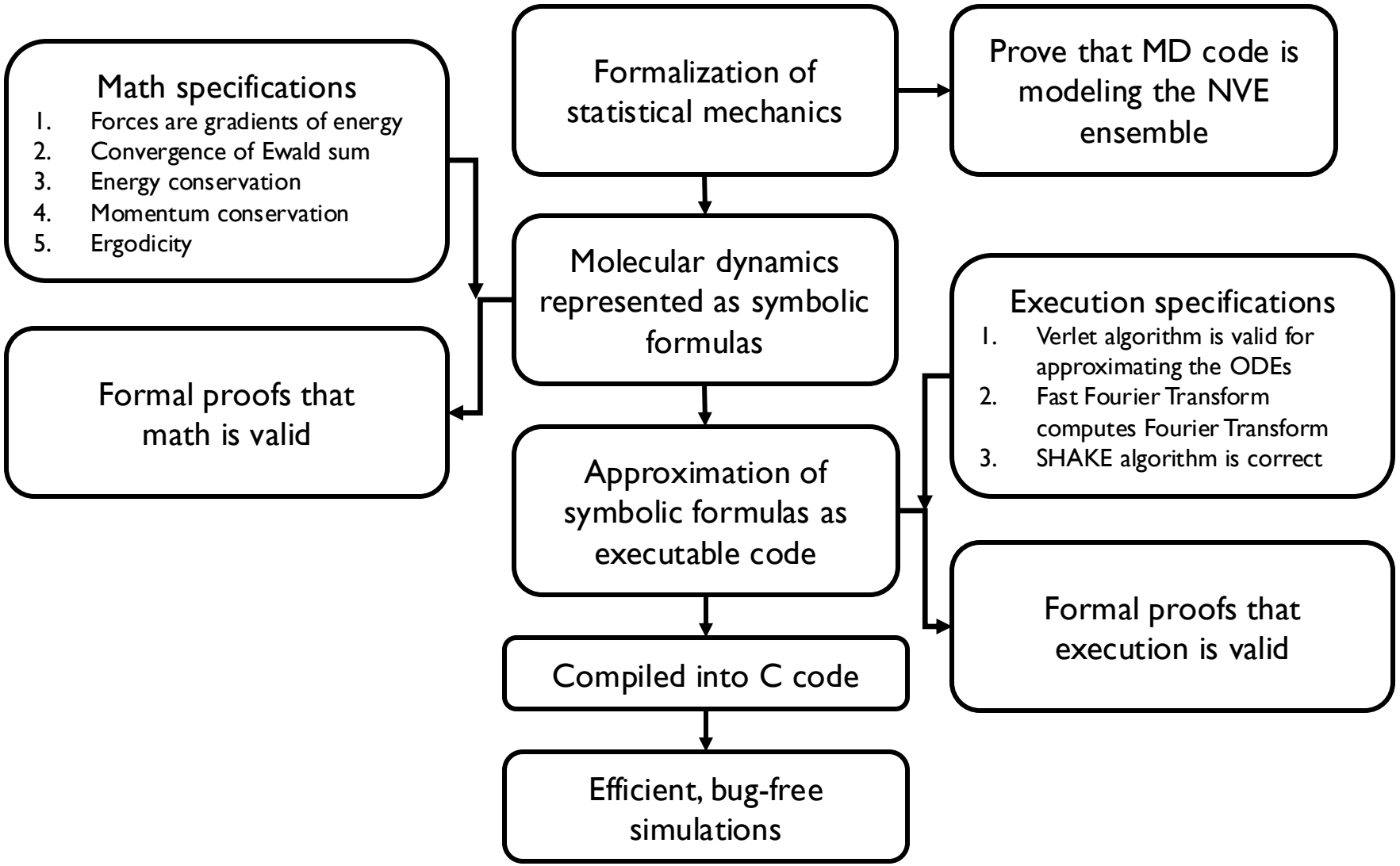
- Design materials for gas separation
  - Carbon dioxide capture
  - Water harvesting from air
  - Purification of medical-grade oxygen
- Modeling biomolecular systems
  - Protein structure and dynamics
  - Drug design
  - Simulating lipid bilayers
- Physical chemistry
  - Phase transitions and critical phenomena
  - Thermodynamics of fluids and mixtures

Between 2000 and 2009,
over 100k papers published
(Allen & Tildesley, 2017)

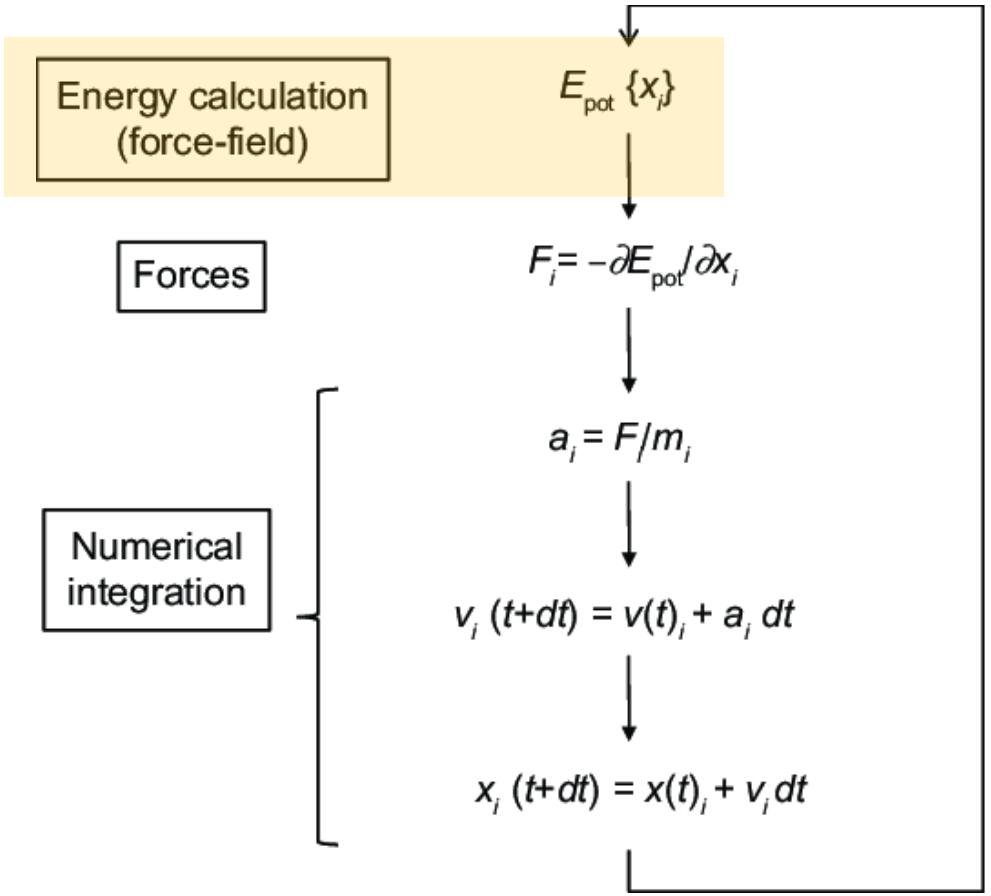
Ingredients of molecular dynamics simulations

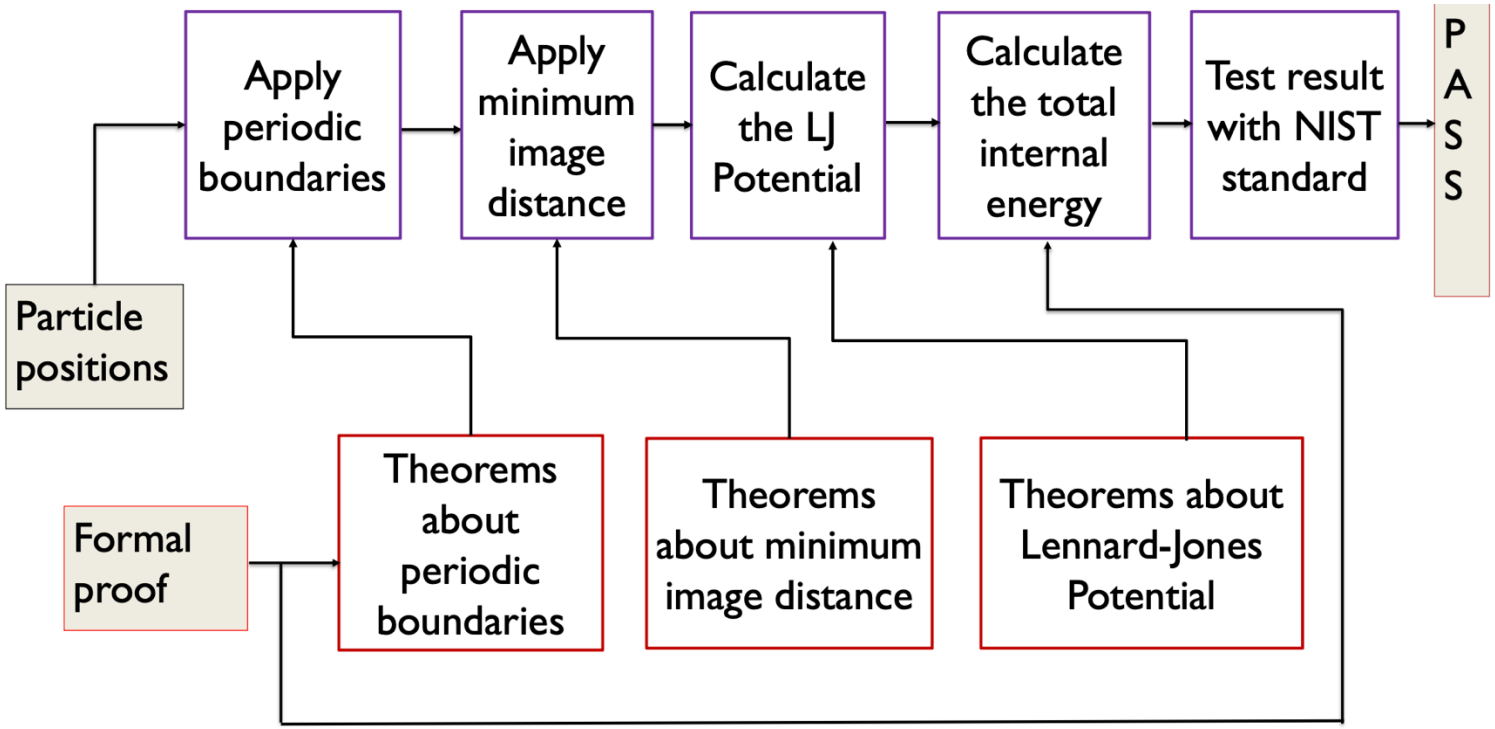
1. Simulation box
2. Particles
3. Force field (interactions among particles)
    1. Energies
    2. Forces
4. Move particles via Newton's equations of motion

LeanMD: Formally-verified molecular dynamics



LeanMD (so far)



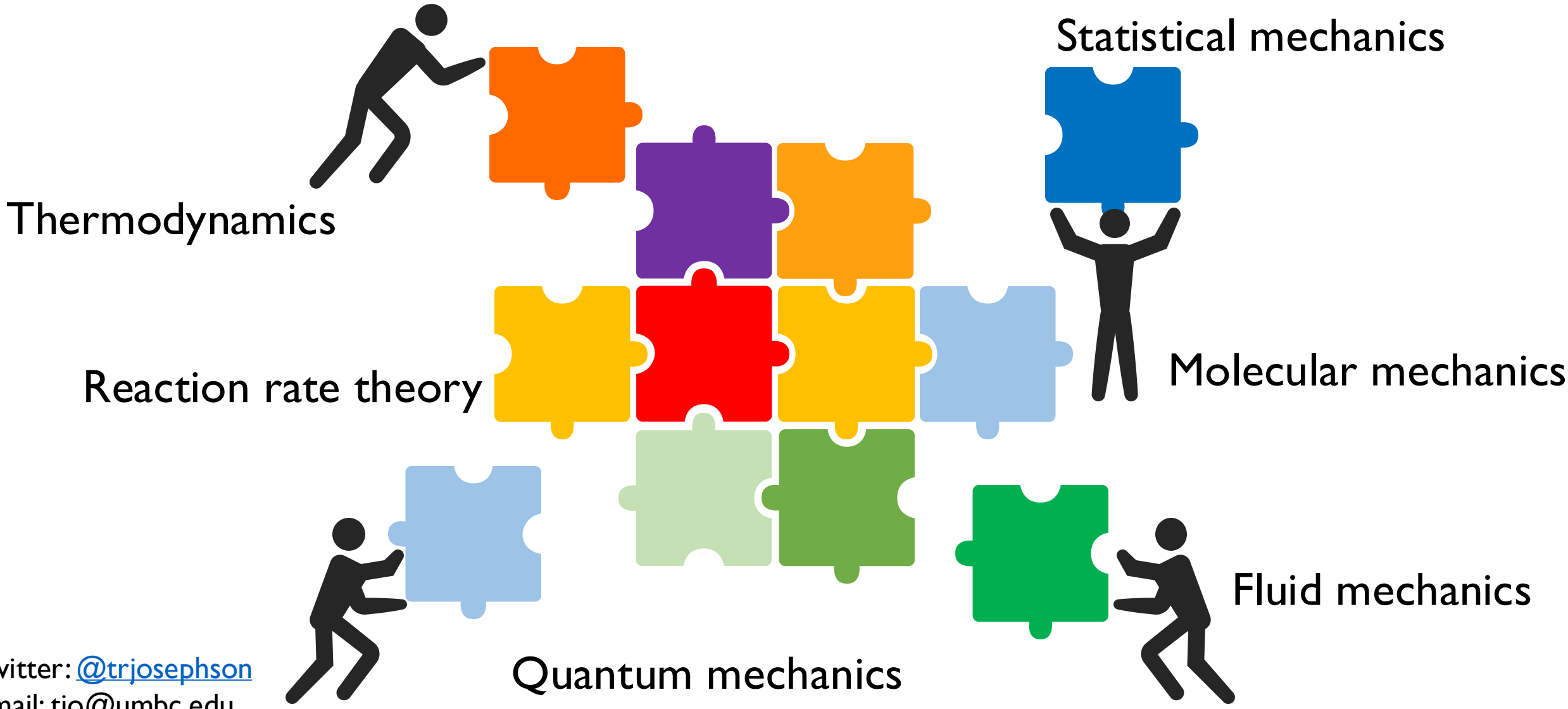


NIST Standard Reference Simulation Website

- https://www.nist.gov/programs-projects/nist-standard-reference-simulation-website

Go to code

SciLib, database of formally verified science



Twitter: @trjosephson
Email: tjo@umbc.edu

What do you want to build?

- Project ideas discussion on Zulip: https://leanprover.zulipchat.com/#narrow/stream/445230-Lean-for-Scientists-and-Engineers-2024/topic/Project.20ideas
- Probability theory (formalizing math)
- Formalizing definition of AIXI (reinforcement learning)
- Markov chain Monte Carlo (Metropolis-Hastings)
- Translating textbook on statistics into textbook with Lean examples
- Data science topics
  - pandas DataFrames in Lean?
  - Linear regression
  - Connecting Lean to data visualization tools in external languages (e.g. Python)