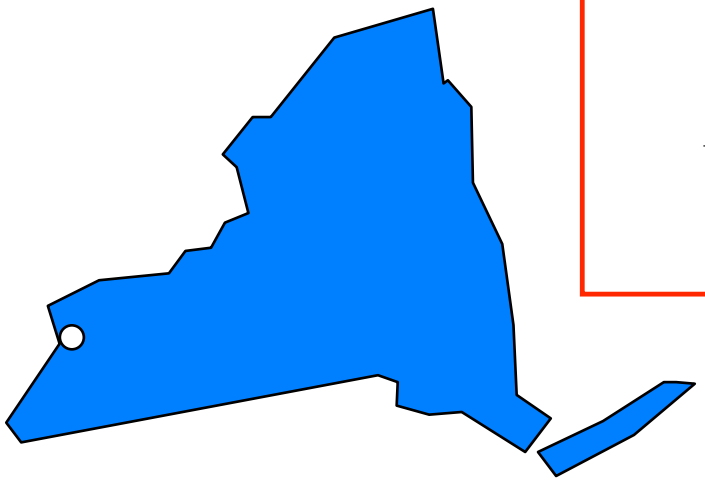
Etomica: An API for Molecular Simulation

David A. Kofke

Department of Chemical and Biological Engineering
University at Buffalo, the State University of New York



UNIVERSITY AT BUFFALO
State University of New York

Outline

- Background
- Design considerations
- Data structures
- Models
- Flow control, actions, and sampling
- Measurements and data processing
- I/O and graphics
- Utilities
- Supporting tools
- Performance
- Some applications

Background

- Fortran-based programmer for 15 years
  - Mostly MC applied to simple fluid and crystalline model systems
- Developed interest in object-oriented programming
  - Took up Java when it emerged in mid-1990's
- Sabbatical at UT-Knoxville/ORNL with PTC
  - 1996-97
- PTC initiated CACHE Molecular Modeling Task Force
  - Wrote successful proposal to develop molecular simulation modules for undergraduate instruction
  - Based in Java for portability, ability to construct browser-based applets
- Initiated molecular simulation API in support of activity

Educational Applications of Etomica

- Summer 1999, 2000
  - Two-week high-school workshop in Computational Chemistry at UB Center for Computational Research
- Fall 2001
  - Freshman/Sophomore honors seminar on molecular simulation
- Spring 2000, 2003
  - CE530 Molecular Simulation
- Summer 2000, 2003, 2004
  - REU students
- Fall 2000, Spring 2004
  - CE 526 Statistical Mechanics
- Spring 2004, 2005
  - Molecular Modeling (senior elective)
- Spring 2005
  - CE 304 Thermodynamics, homework problem

Research Applications of Etomica

- Development of free-energy methods
  - Lennard-Jones, water, simple ionic systems (no Ewald)
- Study of vacancies, defects, miscibility in solids
  - Hard spheres, Lennard-Jones, Valence-force-field models
- Mayer-sampling method for evaluation of virial coefficients
  - Hard spheres, Lennard-Jones, water, alkanes
- Study of vapor-liquid interfacial properties of associating fluids
  - Anisotropic Lennard-Jones, square-well based models

Design Considerations

- Goals
  - Extensible, broadly applicable
  - Computational efficiency
  - Suitable to run interactively or in batch
- Guidelines
  - Highly granular pieces with convenience classes that assemble them
  - Separate components as much as possible
    - Graphics separate from other parts
    - Used objects don't know about user
  - Try to re-use themes that guide design of data and other constructs
    - Parallel hierarchies
    - Event model

Simulation

- Simulation
  - Organizes other elements
  - Common point of reference
  - Independent entity—no simulation knows about or interacts with another Simulation instance
  - No graphical elements
  - Develop new simulations by extending Simulation
    - Assemble simulation in constructor
    - Most fields publicly accessible
    - Reusable in different contexts
  - SimulationContainer gives simulation an interface
    - Graphical elements
    - Remote access as a future consideration
  - Space is assigned to Simulation at construction

Space

- Factory for objects that depend on or define the physical space
  - Vector, Tensor, Orientation, Boundary
- All object methods are implemented in a spatially-independent manner
  - Vector methods defined for vector addition, scalar multiplication, dot product, simple compound operations, etc.
- Easy to convert from simulation in one dimension to another

Data Structures: Atom

- Atom
  - Represents physical atom being simulated
- Contains several fields assigned on construction
  - coord: Coordinate the defines the state of the atom
    - Position, velocity, orientation, spin, etc.
    - Constructed by Space
  - type: AtomType, with information common to multiple atoms
    - Many atoms will reference a single instance of a common AtomType
    - Holds common parameters such as atom mass, size, etc.
    - Holds other information used for operation of simulation
  - seq: AtomSequencer, which is used to place atom in linked list
  - node: AtomTreeNode, which places atom in a tree structure
  - agents[]: Array of objects for anything else
    - Any object can request an instance of its agent be placed in this array for all atoms

Data Structures: AtomFactory

- AtomFactory
  - Builds a molecule according to a specification
  - "Atom" is defined generally
    - "Leaf" atom corresponds to a physical atom
    - Group of atoms, even molecules, are represented by instances of Atom
    - Molecule is represented by a tree structure, using AtomTreeNode
- AtomFactoryMono, AtomFactoryHomo, AtomFactoryHetero
  - Hierarchical: Large molecules built from factories that comprise other factories that build the molecule subunits
- Each factory attaches a unique AtomType to all the Atoms it builds
- Factory has a Conformation that arranges atoms

Data Structures: Phase

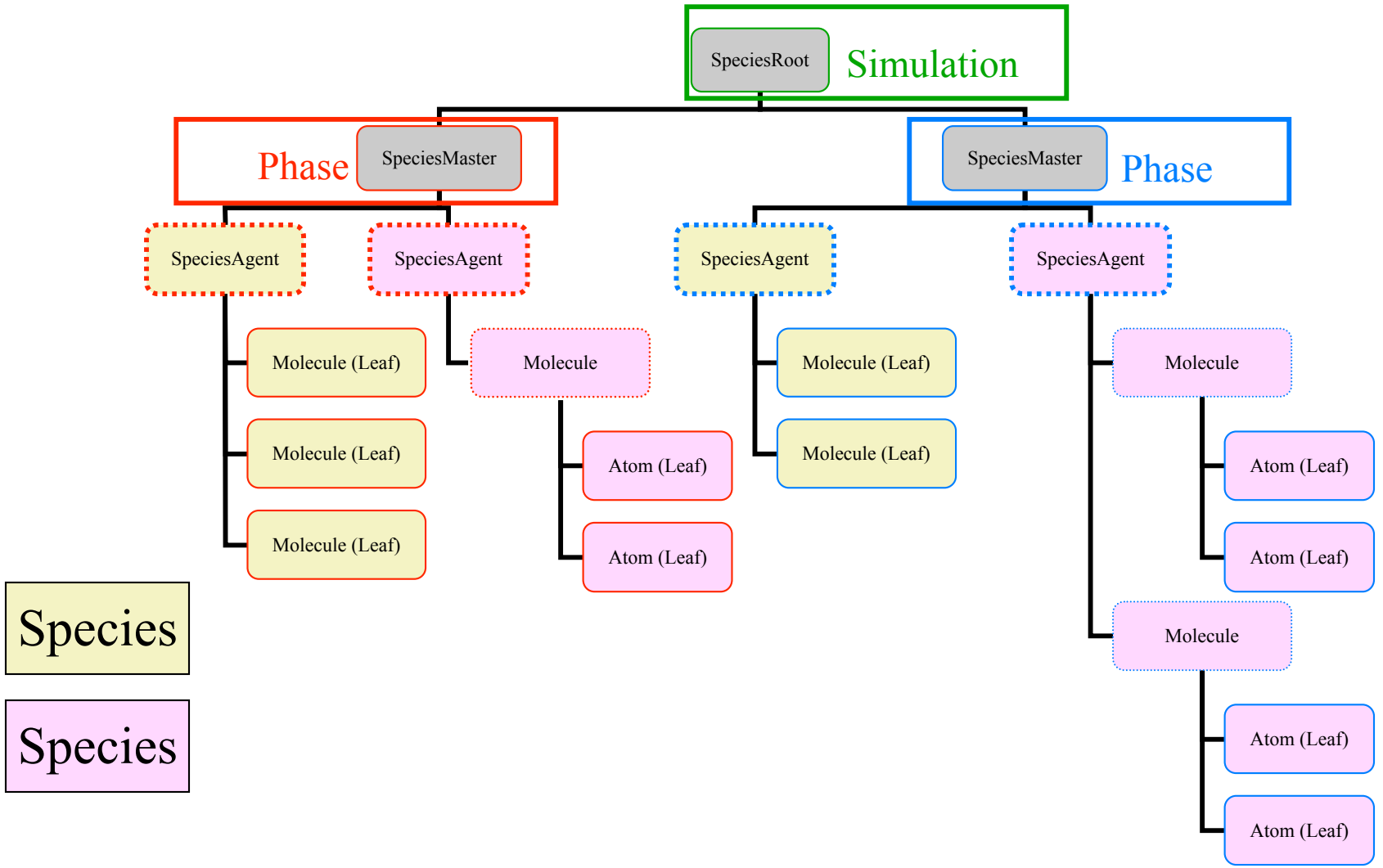
- Phase
  - Collects all atoms that interact with each other
- A single Simulation may employ multiple Phase instances
  - Parallel tempering, Gibbs ensemble
  - No atoms in one Phase interact with atoms in another Phase
  - Phases may interact in more abstract ways (e.g., exchange volume)
- Phase holds a Boundary instance
  - Constructed by Space
  - Implements (or not) periodic boundary conditions
- Phase holds a Lattice, dividing space into cells
  - Useful for neighbor listing, perhaps other things
- Phase holds a Configuration, used to arrange the molecules in a standard way

Data Structures: Species

- Species classes collect information needed to construct and manage molecules
  - AtomFactory
- Extended to make convenience classes
- SpeciesAgent represents Species in each Phase instance
  - extends Atom

Data Structures: Species Hierarchy

- All Atom instances in a Simulation arranged in a hierarchy



Data Structures: AtomsetIterator

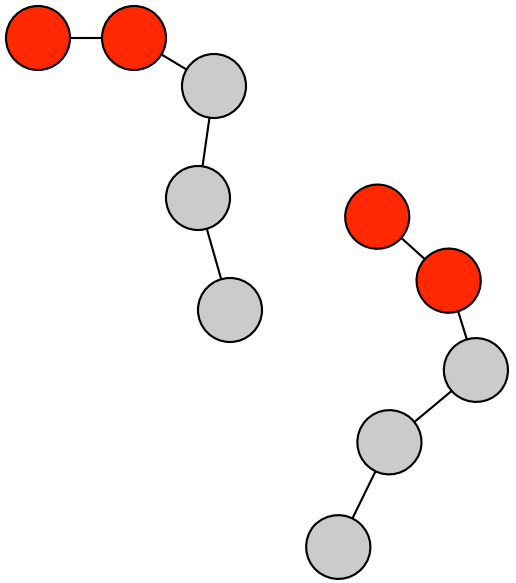
- AtomSet
  - Interface for a set of atoms
    - Atom, AtomPair most often used
- Many types of atom-set iterators
  - Iterate atoms or atom pairs at a particular level in hierarchy
  - Iterate pairs formed with a particular atom
  - Iterate in one or both directions from a given atom
  - Many interfaces defined
    - AtomsetIteratorPhaseDependent
    - AtomsetIteratorBasisDependent
    - AtomsetIteratorDirectable
    - AtomsetIteratorTargetable
    - AtomsetIteratorListDependent
    - etc.

Models: Potential

- Potential
  - Defines manner of interaction of atoms
  - public void energy(AtomSet atoms)
- Subclasses specific to 1-body, 2-body, *etc.* forms
- Interfaces for hard and soft potentials
  - PotentialSoft
    - energy, virial, hypervirial, gradient
  - PotentialHard
    - energy, collisionTime, bump

Models: PotentialGroup

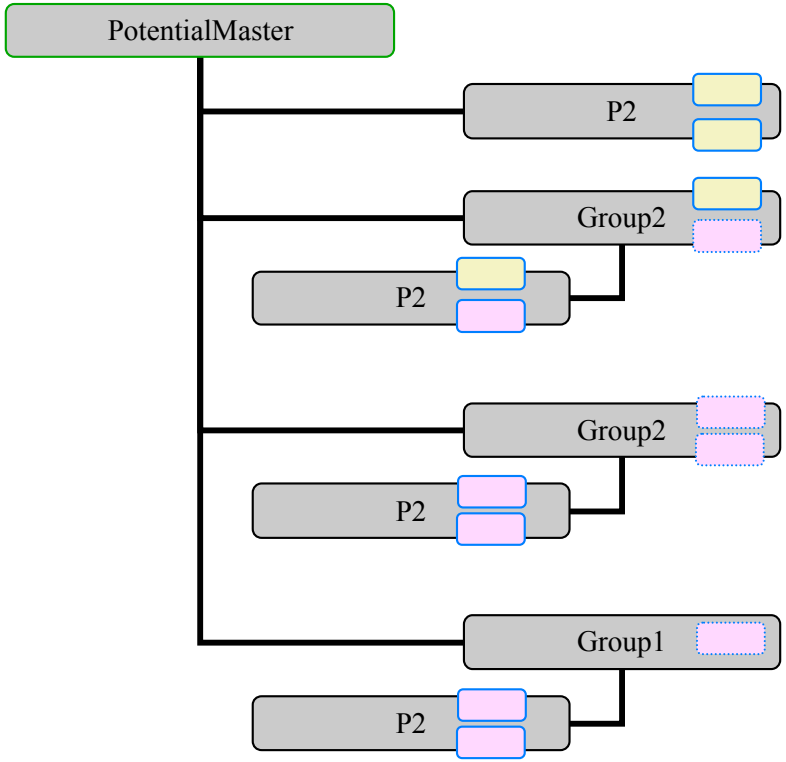
- PotentialGroup
  - Collects several potentials that all interact on a single AtomSet
- 1-body PotentialGroup
  - acts on a single Atom (which typically is a group of atoms)
  - collects intramolecular interactions
- 2-body PotentialGroup
  - acts between two Atom instance
  - collects intermolecular interactions

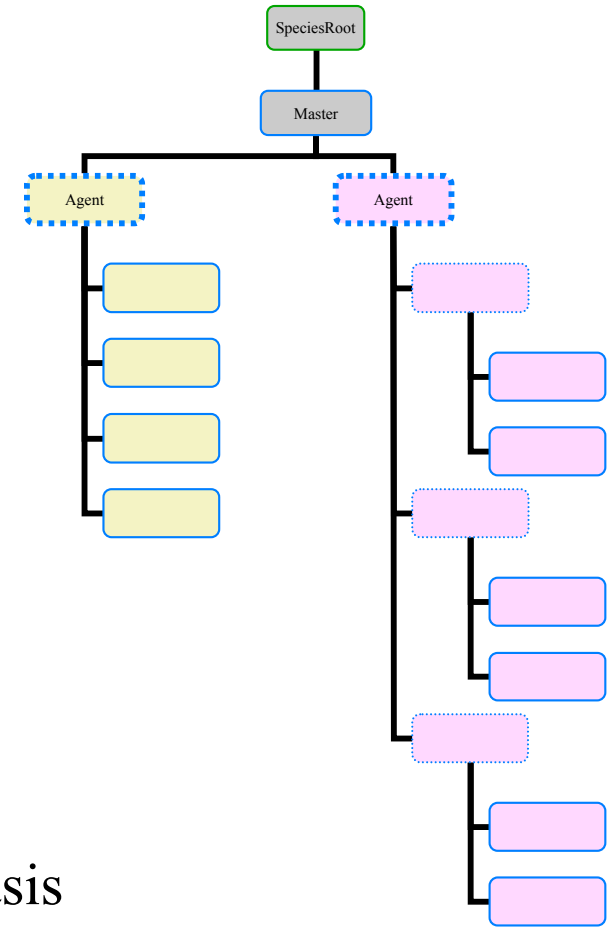


UNIVERSITY AT BUFFALO
State University of New York

Models: Potential Hierarchy

- Parallels Species hierarchy
  - but not segregated by Phase

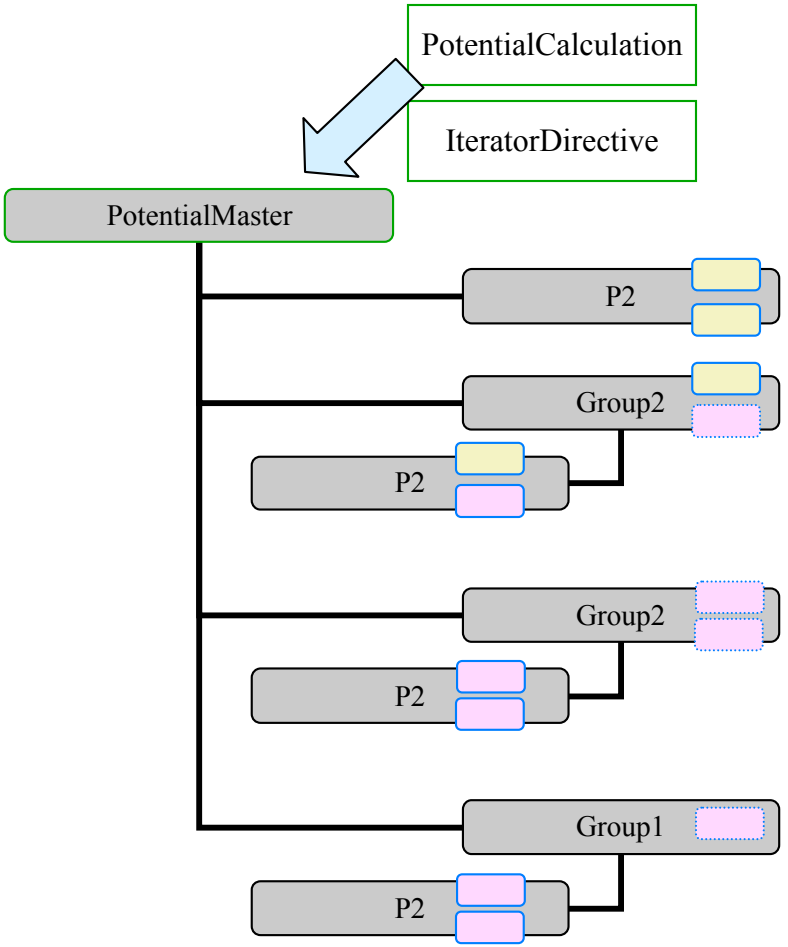




- Each potential iterates over atoms in a basis
  - Iterates form basis for subpotential iteration

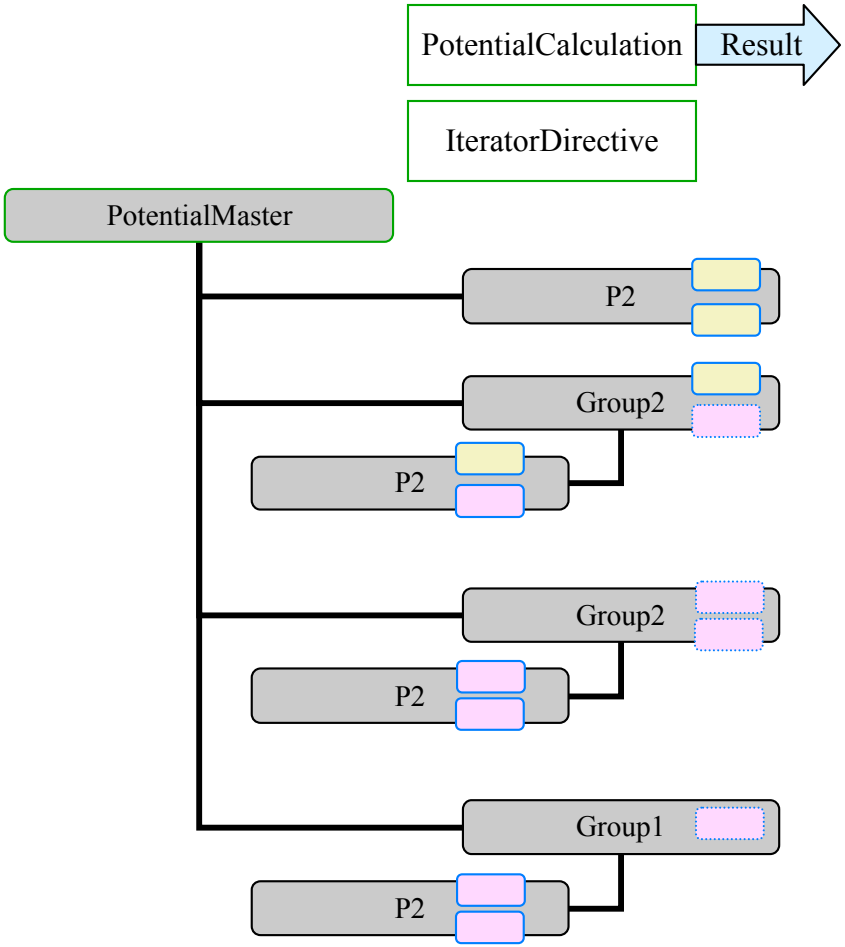
Models: PotentialCalculation

- PotentialCalculation
  - Encapsulates the calculation of a property that depends on the potential
  - For example
    - Energy sum
    - Virial sum
    - Collision time
- IteratorDirective
  - Encapsulates specification of atoms to which calculation is applied
  - For example
    - All leaf atom pairs
    - All pairs formed with a specific Atom
    - All child Atoms of a specifc Atom
    - All leaf atoms of a specific Species



Models: PotentialCalculation

- PotentialCalculation
  - Encapsulates the calculation of a property that depends on the potential
  - For example
    - Energy sum
    - Virial sum
    - Collision time
- IteratorDirective
  - Encapsulates specification of atoms to which calculation is applied
  - For example
    - All leaf atom pairs
    - All pairs formed with a specific Atom
    - All child Atoms of a specifc Atom
    - All leaf atoms of a specific Species



Models: PotentialMaster

- Traversal of hierarchy inefficient for routine application
  - Main role is to define the model
- PotentialMaster subclasses
  - Same interface
  - Permits extension to other iteration schemes
  - PotentialMasterCell
    - Cell-based neighbor iteration
  - PotentialMasterNbr
    - List-based neighbor iteration
- Implement parallel/high-performance schemes by subclassing PotentialMaster

Models: Model

- Model
  - class to simplify specification of model
  - coordinates construction of species and potentials
  - still in development

Flow Control: Action and Activity

- Action
  - interface for abstract, elementary action that does something
  - public void actionPerformed()
  - can be grouped for series implementation
  - for example
    - AtomActionRandomizeVelocity
    - AtomActionTranslateBy
    - IntegratorReset
    - PhaseInflate
- Activity
  - more complex, time-consuming extension of Action
  - can be started, stopped, paused, resumed
  - can be grouped for series or parallel implementation
  - for example
    - ActivityIntegrate
    - EquilibrationProduction

Flow Control: Controller

- Two ways to conduct simulation
  - interactively
  - batch
  - (or hybrid of both)
- Specification of actions must be mutable
  - even while simulation proceeds
- Controller
  - schedules actions to be performed
  - single instance constructed for each Simulation
  - actions/activities can be added to queue
  - urgentAction can be requested for immediate implementation
    - all GUI-driven changes follow this path
  - carefully synchronized

Flow Control: Integrator

- Integrator
  - repeatedly changes configuration to follow a sampling algorithm
  - public void doStep()
  - deploys subclass-specific agent to each atom
  - only one integrator acts on a given phase
  - some integrators act on multiple phases
    - IntegratorGEMC (Gibbs ensemble Monte Carlo)
    - IntegratorPT (Parallel tempering)
- IntegratorMD
  - IntegratorVelocityVerlet
  - IntegratorHard
    - discontinuous molecular dynamics
- IntegratorMC

Flow Control: IntegratorMC

- IntegratorMC
  - Monte Carlo sampling
  - Selects trial move, performs trial, decides acceptance, notifies move and other listeners
- MCMove
  - Performs Monte Carlo trial
  - Reports information needed to determine acceptance
    - $\ln(p_{new}/p_{old})$, $\ln(t_{ij}/t_{ji})$
    - Holds fields needed for evaluation
  - Does appropriate update for acceptance or rejection
  - For example
    - MCMoveAtom
    - MCMoveInsertDelete
    - MCMoveRotateMolecule
    - MCMoveVolume
  - Sampled ensemble is determined by set of MCMoves added to integrator

Flow Control: IntegratorEvent

- IntegratorEvent
  - integrator fires event to registered listeners to notify of progress with simulation
- IntegratorListener
  - IntegratorIntervalListener
    - receives repeated events reporting progress
  - IntegratorNonintervalListener
    - receives only events indicating initialization, start, end, etc.
  - For example
    - objects pushing data measurement and processing
    - cell- and neighborlist-updating

Data Processing: DataSource, DataSink

- DataSource
  - interface for class that can provide data
  - data is generally represented by array of double
  - `public double[] getData();`
  - Meter is a DataSource that acts on a Phase
  - for example
    - MeterDensity, MeterEnergy, MeterRDF, MeterTemperature
    - DataSourceCountCollisions, DataSourceCountTime
- DataSink
  - interface for class that can receive data
  - `public void putData(double[] data);`
  - for example
    - DisplayBox, DataSinkConsole, DataBin
    - DataPipe

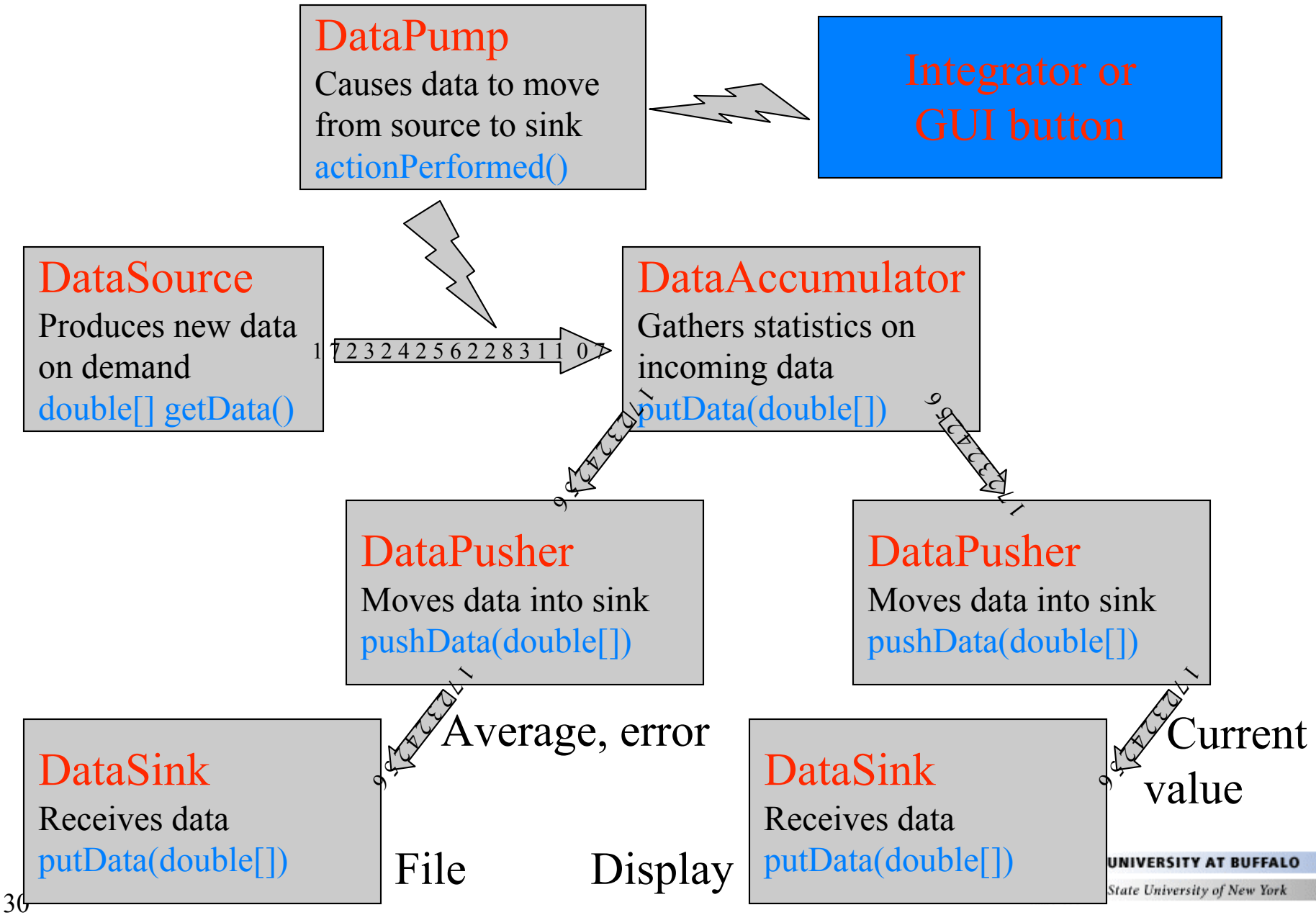
Data Processing: Pipelines

- Data is pushed from a source to a sink
  - It may pass through other elements along the way
  - Each pushes data on to the next element
- DataPusher
  - Holds one or more sinks and pushes data to them on request
  - Has methods to manage data sinks
- DataPipe
  - Abstract, extends DataPusher, implements DataSink
  - Takes data given to it, does something to it, and pushes new data
  - DataAccumulator
    - Collects statistics on data it receives, and pushes it on at intervals
    - *e.g.* AccumulatorAverage, AccumulatorHistory, AccumulatorHistogram
  - DataTranformer
    - Modifies data and immediately pushes it downstream

Data Processing: DataPump

- DataPump
  - Extends DataPusher
  - Holds a DataSource, and moves data from it to the sinks
  - Provides the impetus for moving the data from a source into a pipe
  - Implements Action
    - Typically activated via Integrator IntervalEvent, or GUI action

Data Flows in Etomica



I/O and Graphics: Display

- Display
  - Object to present data in graphical interface
- Boxes, plots, tables, etc.
- All are treated as implementing DataSink
- Logging capabilities still not well developed
- Units
  - Internally, all data are represented in a common unit system
    - picosecond, Angstrom, Dalton
  - Unit classes are defined to handle conversions
  - All I/O and graphics classes hold a Unit instance
  - Classes can declare Dimension for fields so that appropriate units are offered

I/O and Graphics: Device

- Device
  - Widget that allows user to interact with simulation
- Examples
  - DeviceButton
    - Connects to an action, performs action when button is pressed
  - DeviceSlider
    - Changes value of some quantity with movement of a slider
  - DeviceThermoController
    - ComboBox that permits selection from several temperatures
  - DeviceCheckBox
    - Toggles a boolean value using a checkbox
  - DeviceControllerButton
    - Start/stop/pause/resume simulation
- Acts via Controller
  - Invokes urgentAction
  - Controller handles Action request ASAP
    - Pauses current Activity, or finishes current Action
    - then attends to requested Action
  - Prevents collision between user and integrator threads

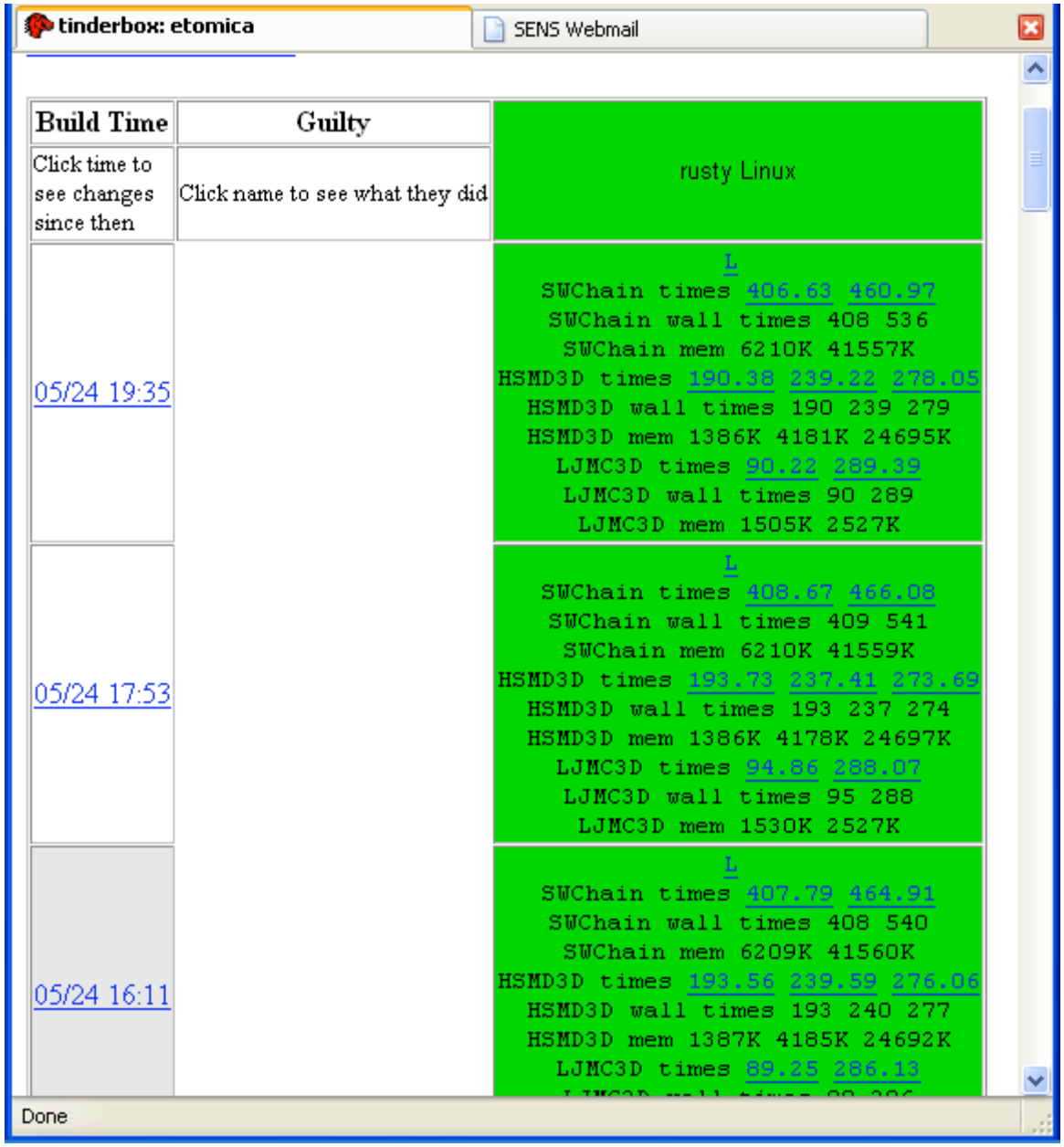
Utilities

- Utility classes developed as needed
  - versatile lattice capabilities
  - Polytope for defining shapes
  - very small set of math classes
    - linear algebra
    - special functions
    - permutations/combinations

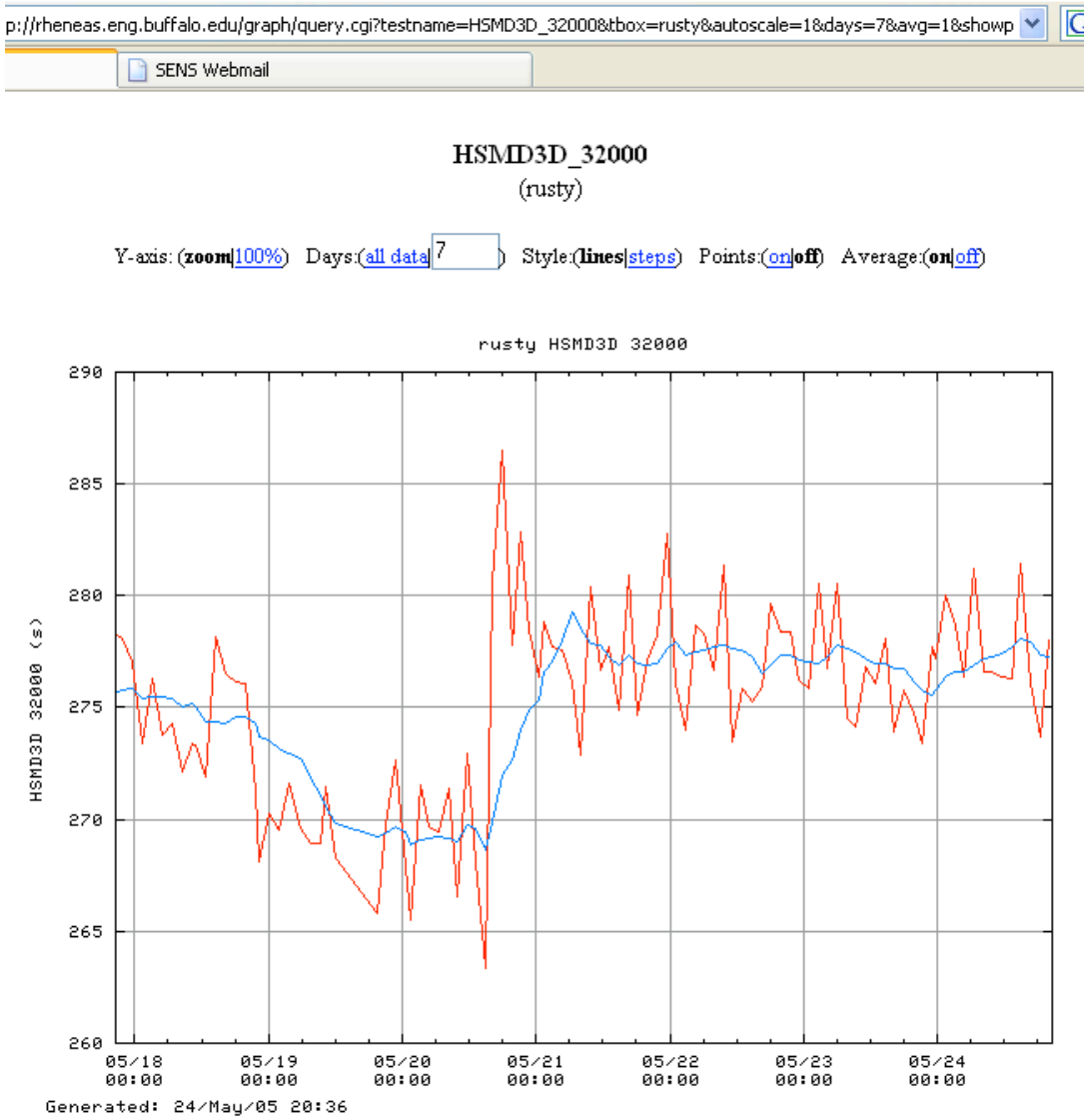
Supporting Tools

- CVS
- JUnit
  - facility for developing unit tests
- javadoc
  - facility to generate hyperlinked documentation from comments
- bugzilla
  - bug tracking
- tinderbox
  - performance tracking

Supporting Tools: Tinderbox



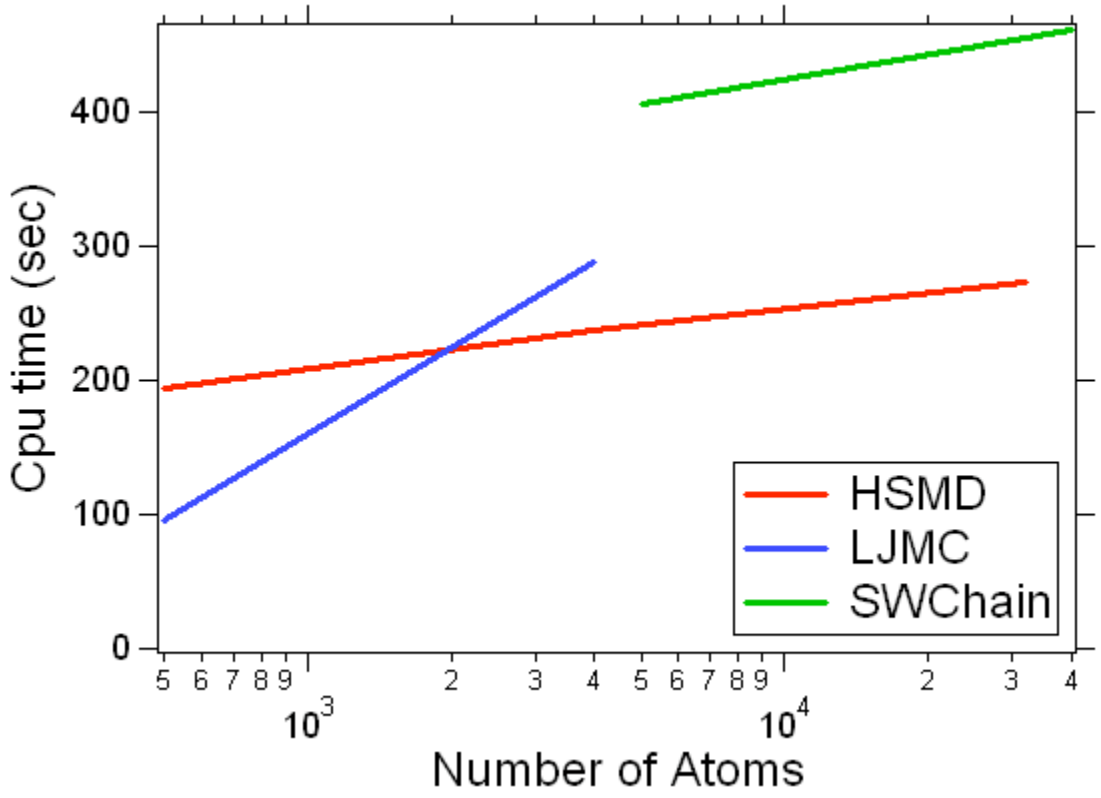
Supporting Tools: Tinderbox



Performance

- Benchmark tests
  - Hard-sphere molecular dynamics
  - Square-well chain molecular dynamics
  - Lennard-Jones Monte Carlo
- Comparison to specialized fortran codes
  - 2- to 4-times slower
- No problem scaling to large systems

Performance: Scaling



Some Applications

- Piston-cylinder MD simulation
- Diffusion through a nanotube: DCVGCMD
- Ising model

Acknowledgements

- Support from the National Science Foundation
- Peter Cummings
- C. Daniel Barnes
- Andrew Schultz

Outline

- Background
- Design considerations
- Data structures
- Models
- Flow control, actions, and sampling
- Measurements and data processing
- I/O and graphics
- Utilities
- Supporting tools
- Performance
- Some applications