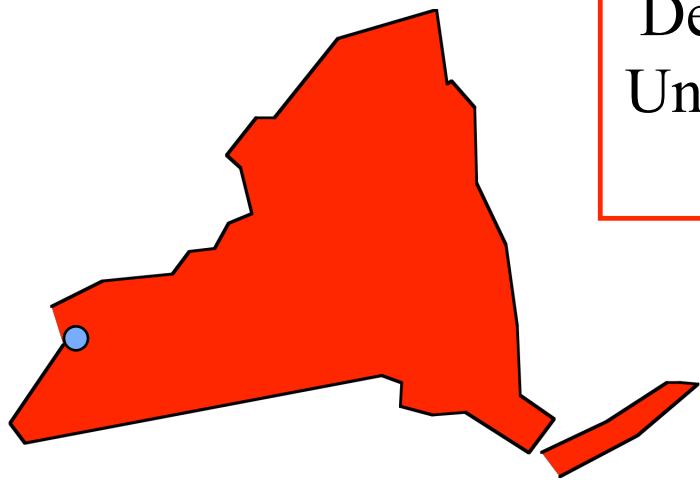


Workshop: Web-based Technologies for Using and Teaching Molecular Simulation

Ninth International Conference on
Properties and Phase Equilibria for Product and Process Design

David A. Kofke

Department of Chemical Engineering
University at Buffalo, State University
of New York



Audience and Aims

- Audience
 - Molecular Simulation Novice
 - To understand elements of molecular simulation
 - Educator
 - To learn about a tool for constructing educational applets, and an approach to teaching molecular simulation
 - Molecular Simulation Expert
 - To see another way to organize and implement a molecular simulation
- Present a conceptual basis for understanding simulation
 - Implementation details vary greatly from one person's simulation to the next
 - We give some implementation detail to fix ideas

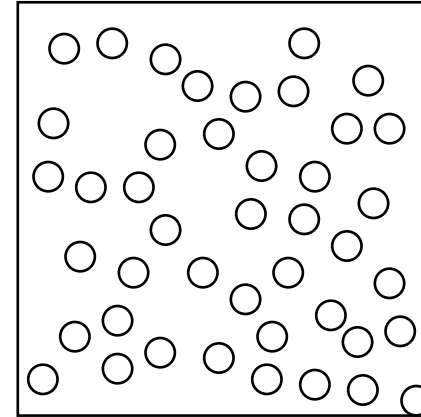
Presentation Will Have a Moving Focus

- Molecular simulation concepts
- Algorithmic details
- Programming implementation
- The *Etomica* development environment
- Prototype applets

What is Molecular Simulation?

- Molecular simulation is a computational “experiment” conducted on a molecular model.

*10 to 100,000 or more
atoms are simulated
(typically 500 - 1000)*



- Many configurations are generated, and averages taken to yield the “measurements.” One of two methods is used:

– Molecular dynamics

- Integration of equations of motion
- Deterministic
- Retains time element

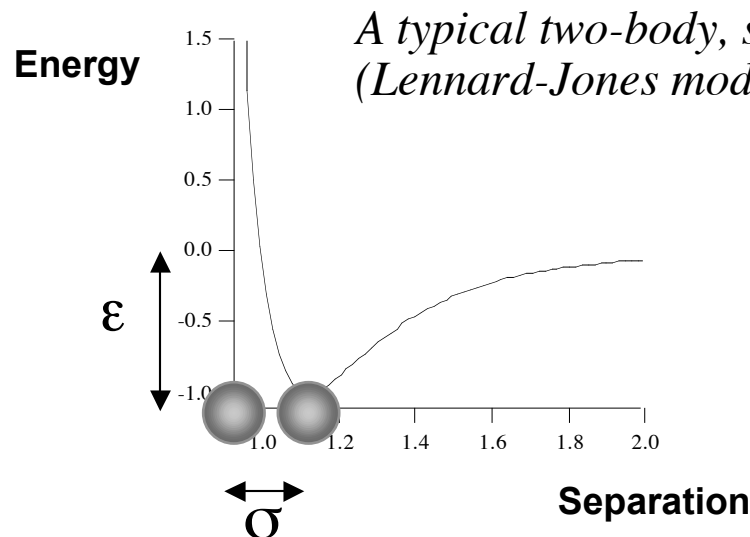
Monte Carlo

- Ensemble average
- Stochastic
- No element of time

- Molecular simulation has the character of both theory and experiment
- Applicable to molecules ranging in complexity from rare gases to polymers to electrolytes to metals

What is a Molecular Model?

- A molecular model postulates the interactions between molecules

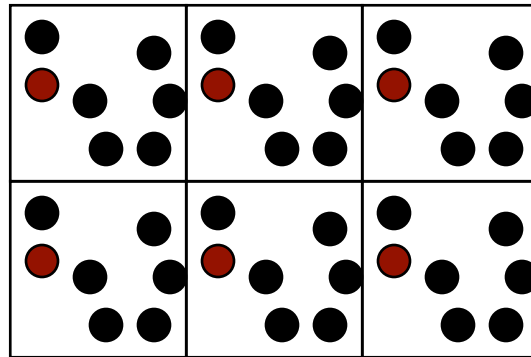


$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]$$

- More realistic models require other interatomic contributions
 - Intramolecular
 - stretch, bend, out-of-plane bend, torsion, +intermolecular terms
 - Intermolecular
 - van der Waals attraction and repulsion (Lennard-Jones form)
 - electrostatic
 - multibody

Boundary Conditions

- Impractical to contain system with a real boundary
 - Enhances finite-size effects
 - Artificial influence of boundary on system properties
- Instead surround with replicas of simulated system
 - “Periodic Boundary Conditions” (PBC)



- [Click here](#) to view an applet demonstrating PBC

Etomica

- GUI-based development environment
 - Simulation is constructed by piecing together elements
 - No programming required
 - Result can be exported to run stand-alone as applet or application
- Application Programming Interface (API)
 - Library of components used to assemble a simulation
 - Can be used independent of development environment
 - Invoked in code programmed using Emacs (for example)
- Written in Java
 - Widely used and platform independent
 - Features of a modern programming language
 - But is it Slow?
 - Object-oriented
- Vehicle for presentation of molecular simulation methods

What is an Object?

- A fancy variable
 - stores data
 - can perform operations using the data
- Every object has a type, or “class”
 - analogous to real, integer, etc.
 - Fortran: `real x, y, z`
 - Java: `Atom a1, a2;`
 - you define types (classes) as needed to solve your problems

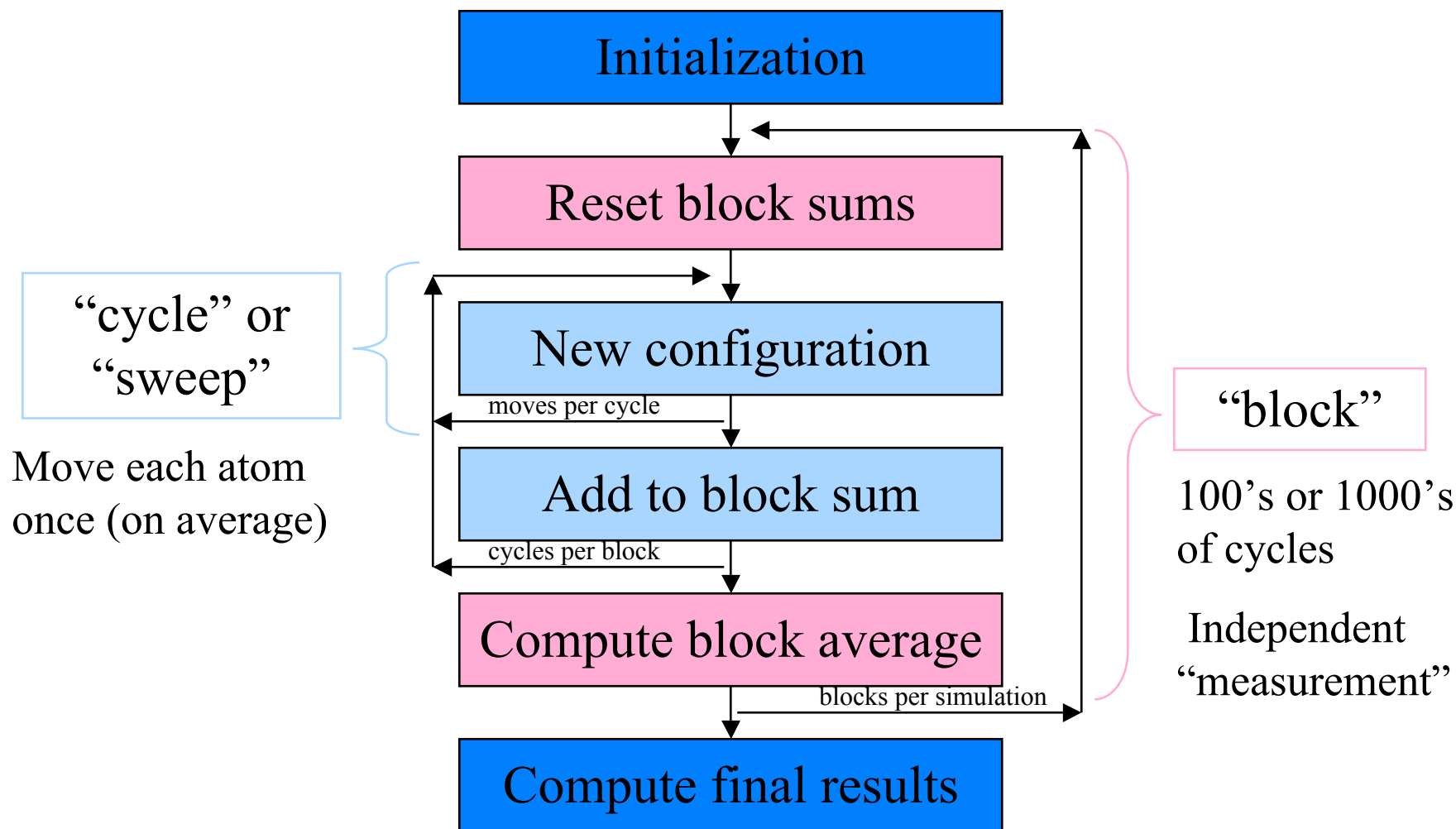
```
public class Atom {
    double mass;
    Vector r, p;
}
```
 - types differ in the data they hold and the actions they perform
 - every object is an “instance of a class”

```
a1 = new Atom();
```

Makeup of an Object

- Fields (data)
 - primitive types (integer, float, double, boolean, etc.)
 - handles to other objects
 - complex objects are composed from simpler objects (composition)
- Methods (actions)
 - “subroutines and functions”
 - may take arguments and return values
 - have complete access to all fields of object
- Inheritance
 - can define subclasses which inherit features of parent class
 - same interface, but different implementations
 - subclasses can be used anywhere parent class is expected
 - mechanism to change behavior of simulation

Structure of a Molecular Simulation



Simulation Element: Integrator

- Advances molecule positions according to some rule
- Here's the core method in parent *Integrator* class

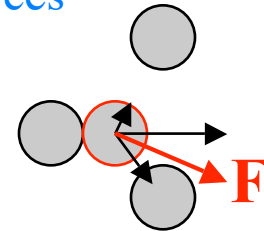
```
public void run() {
    stepCount = 0;
    int iieCount = interval+1;
    while(stepCount < maxSteps) {
        while(pauseRequested) doWait();
        if(resetRequested) {doReset(); resetRequested = false;}
        if(haltRequested) break;
        doStep(); //abstract method in Integrator. subclasses implement algorithms (MD/MC)
        if(--iieCount == 0) { //count down to determine when a cycle is completed
            fireIntervalEvent(intervalEvent); //notify listeners of completion of cycle
            iieCount = interval;
        }
        if(doSleep) { //slow down simulation so display can keep up
            try { Thread.sleep(sleepPeriod); }
            catch (InterruptedException e) { }
        }
        stepCount++;
    } //end of while loop
    fireIntervalEvent(new IntervalEvent(this, IntervalEvent.DONE));
} //end of run method
```

Integration Algorithms

- Equations of motion in cartesian coordinates

$$\begin{aligned} \frac{d\mathbf{r}_j}{dt} &= \frac{\mathbf{p}_j}{m} \\ \frac{d\mathbf{p}_j}{dt} &= \mathbf{F}_j \end{aligned}$$

$$\left. \begin{aligned} \mathbf{r} &= (r_x, r_y) \\ \mathbf{p} &= (p_x, p_y) \end{aligned} \right\} \text{2-dimensional space (for example)}$$
$$\mathbf{F}_j = \sum_{\substack{i=1 \\ i \neq j}}^N \mathbf{F}_{ij} \quad \text{pairwise additive forces}$$



- Desirable features of an integrator
 - minimal need to compute forces (a very expensive calculation)
 - good stability for large time steps
 - good accuracy
 - conserves energy and momentum
- Soft- and hard-potential integrators work differently

Verlet Algorithm 1. Equations

- Very simple, very good, very popular algorithm
- Consider expansion of coordinate forward and backward in time

$$\mathbf{r}(t + \delta t) = \mathbf{r}(t) + \frac{1}{m}\mathbf{p}(t)\delta t + \frac{1}{2m}\mathbf{F}(t)\delta t^2 + \frac{1}{3!}\mathbf{a}(t)\delta t^3 + O(\delta t^4)$$

$$\mathbf{r}(t - \delta t) = \mathbf{r}(t) - \frac{1}{m}\mathbf{p}(t)\delta t + \frac{1}{2m}\mathbf{F}(t)\delta t^2 - \frac{1}{3!}\mathbf{a}(t)\delta t^3 + O(\delta t^4)$$

- Add these together

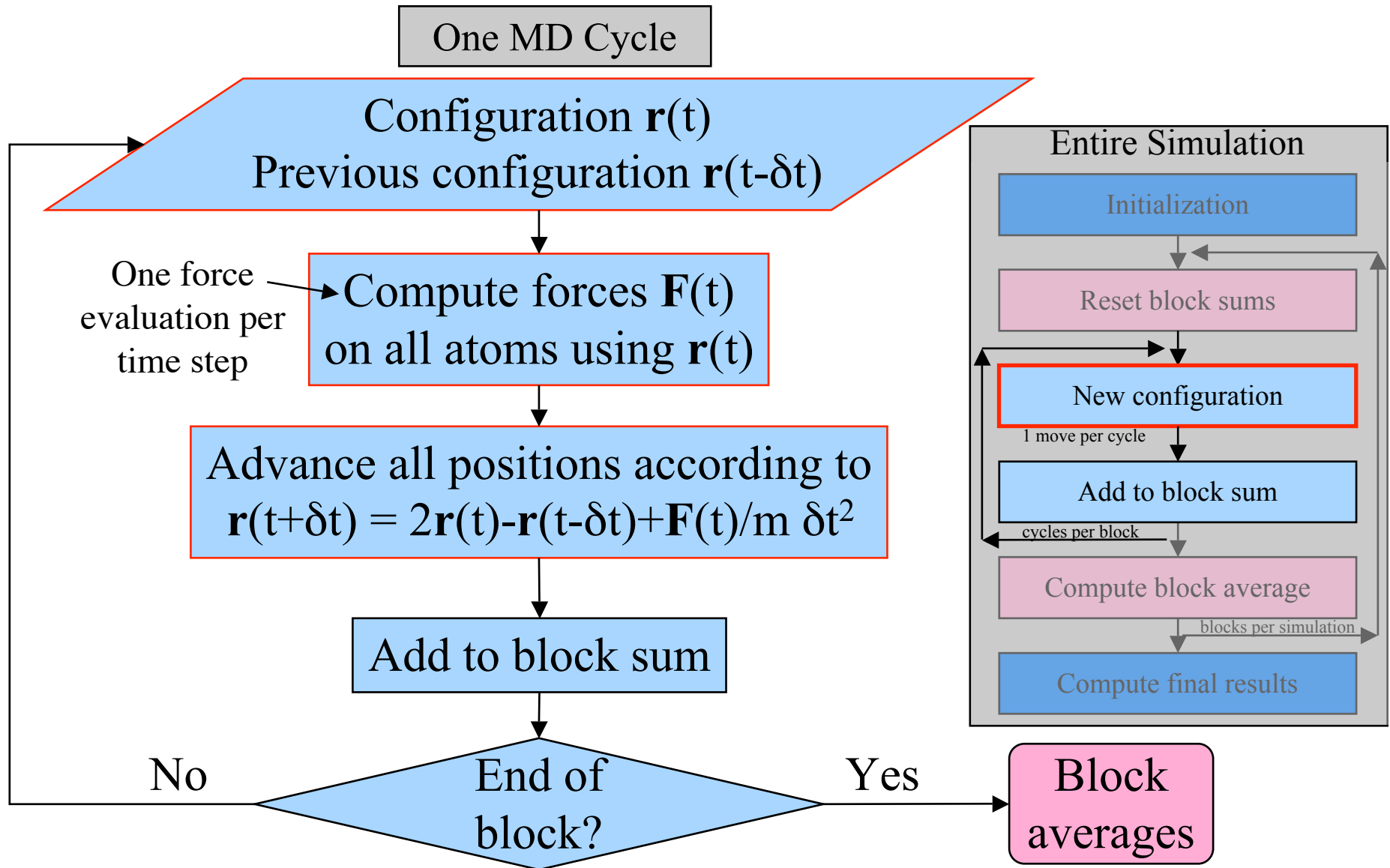
$$\mathbf{r}(t + \delta t) + \mathbf{r}(t - \delta t) = 2\mathbf{r}(t) + \frac{1}{m}\mathbf{F}(t)\delta t^2 + O(\delta t^4)$$

- Rearrange

$$\mathbf{r}(t + \delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \delta t) + \frac{1}{m}\mathbf{F}(t)\delta t^2 + O(\delta t^4)$$

- update without ever consulting velocities!

Verlet Algorithm 2. Flow diagram



Verlet Algorithm 3. Java Code

`public class IntegratorVerlet extends Integrator`

```
//Performs one timestep increment in the Verlet algorithm
public void doStep() {

    atomIterator.reset();
    while(atomIterator.hasNext()) { //zero forces on all atoms
        ((Agent)atomIterator.next().ia).force.E(0.0);
    }
    pairIterator.allPairs(forceSum); //sum forces on all pairs

    double t2 = tStep*tStep;
    atomIterator.reset();
    while(atomIterator.hasNext()) { //loop over all atoms, moving according to Verlet
        Atom a = atomIterator.next();
        Agent agent = (Agent)a.ia;
        Space.Vector r = a.position(); //current position of the atom
        temp.E(r); //save it
        r.TE(2.0); //2*r
        r.ME(agent.rLast); //2*r-rLast
        agent.force.TE(a.rm()*t2); // f/m dt^2
        r.PE(agent.force); //2*r - rLast + f/m dt^2
        agent.rLast.E(temp); //rLast gets present r
    }
    return;
}
```

```
run() method in Integrator
public void run() {
    stepCount = 0;
    int iieCount = interval+1;
    while(stepCount < maxSteps) {
        while(pauseRequested)
            doWait();
        if(resetRequested)
            doReset();
        if(haltRequested) break;
        doStep();
        if(--iieCount == 0) {
            fireIntervalEvent();
            iieCount = interval;
        }
        if(doSleep) {
        }
        stepCount++;
    } //end of while loop
    fireIntervalEvent();
} //end of run method
```

- Time for a demonstration...

Verlet Algorithm 3. Java Code

`public class IntegratorVerlet extends Integrator`

```
//Performs one timestep increment in the Verlet algorithm
public void doStep() {

    atomIterator.reset();
    while(atomIterator.hasNext()) { //zero forces on all atoms
        ((Agent)atomIterator.next().ia).force.E(0.0);
    }
    pairIterator.allPairs(forceSum); //sum forces on all pairs

    double t2 = tStep*tStep;
    atomIterator.reset();
    while(atomIterator.hasNext()) { //loop over all atoms, moving according to Verlet
        Atom a = atomIterator.next();
        Agent agent = (Agent)a.ia;
        Space.Vector r = a.position(); //current position of the atom
        temp.E(r); //save it
        r.TE(2.0); //2*r
        r.ME(agent.rLast); //2*r-rLast
        agent.force.TE(a.rm()*t2); // f/m dt^2
        r.PE(agent.force); //2*r - rLast + f/m dt^2
        agent.rLast.E(temp); //rLast gets present r
    }
    return;
}
```

- Time for a demonstration...

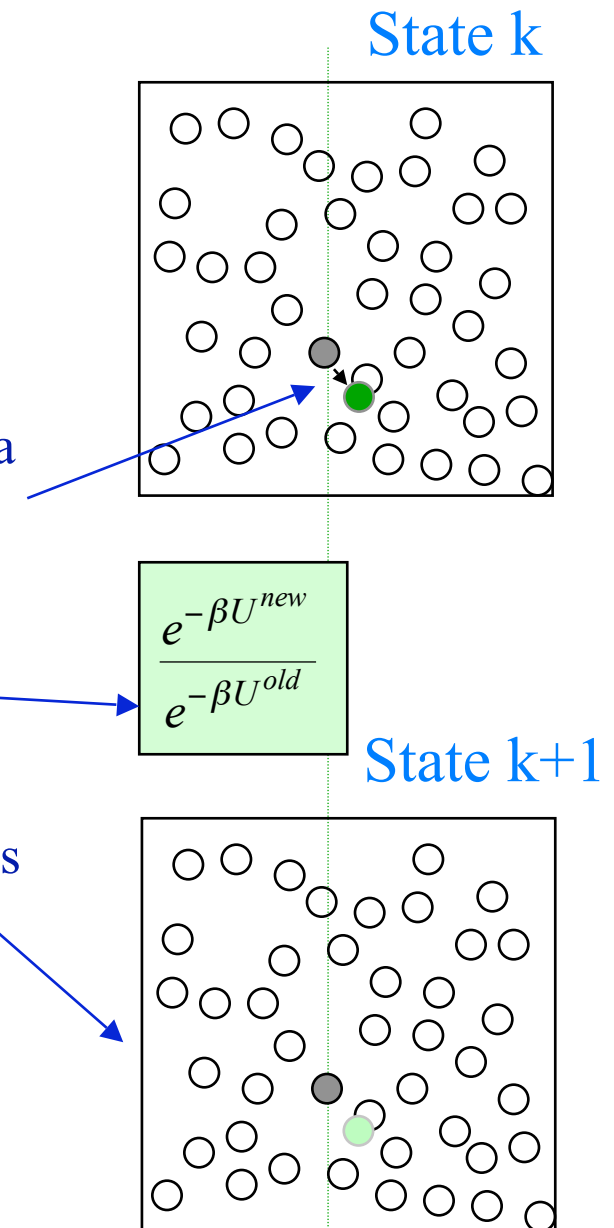
Monte Carlo Simulation 1.

- Gives properties via ensemble averaging
 - No time integration
 - Cannot measure dynamical properties
- Employs stochastic methods to generate a (large) sample of members of an ensemble
 - “random numbers” guide the selection of new samples
- Permits great flexibility
 - members of ensemble can be generated according to any convenient probability distribution...
 - ...and probability distribution can be sampled many ways
 - strategies developed to optimize quality of results
 - ergodicity — better sampling of all relevant regions of configuration space
 - variance minimization — better precision of results
- MC “simulation” is the evaluation of statistical-mechanics integrals

$$\langle U \rangle = \frac{1}{Z_N} \frac{1}{N!} \int dr^N U(r^N) e^{-\beta U(r^N)}$$

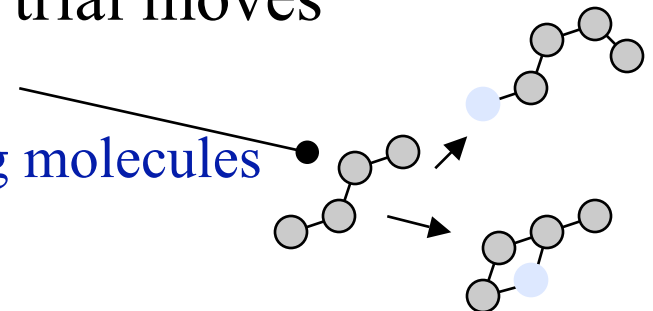
Monte Carlo Simulation 2.

- Almost always involves a Markov process
 - move to a new configuration from an existing one according to a well-defined transition probability
- Simulation procedure
 - generate a new “trial” configuration by making a perturbation to the present configuration
 - accept the new configuration based on the ratio of the probabilities for the new and old configurations, according to the Metropolis algorithm
 - if the trial is rejected, the present configuration is taken as the next one in the Markov chain
 - repeat this many times, accumulating sums for averages

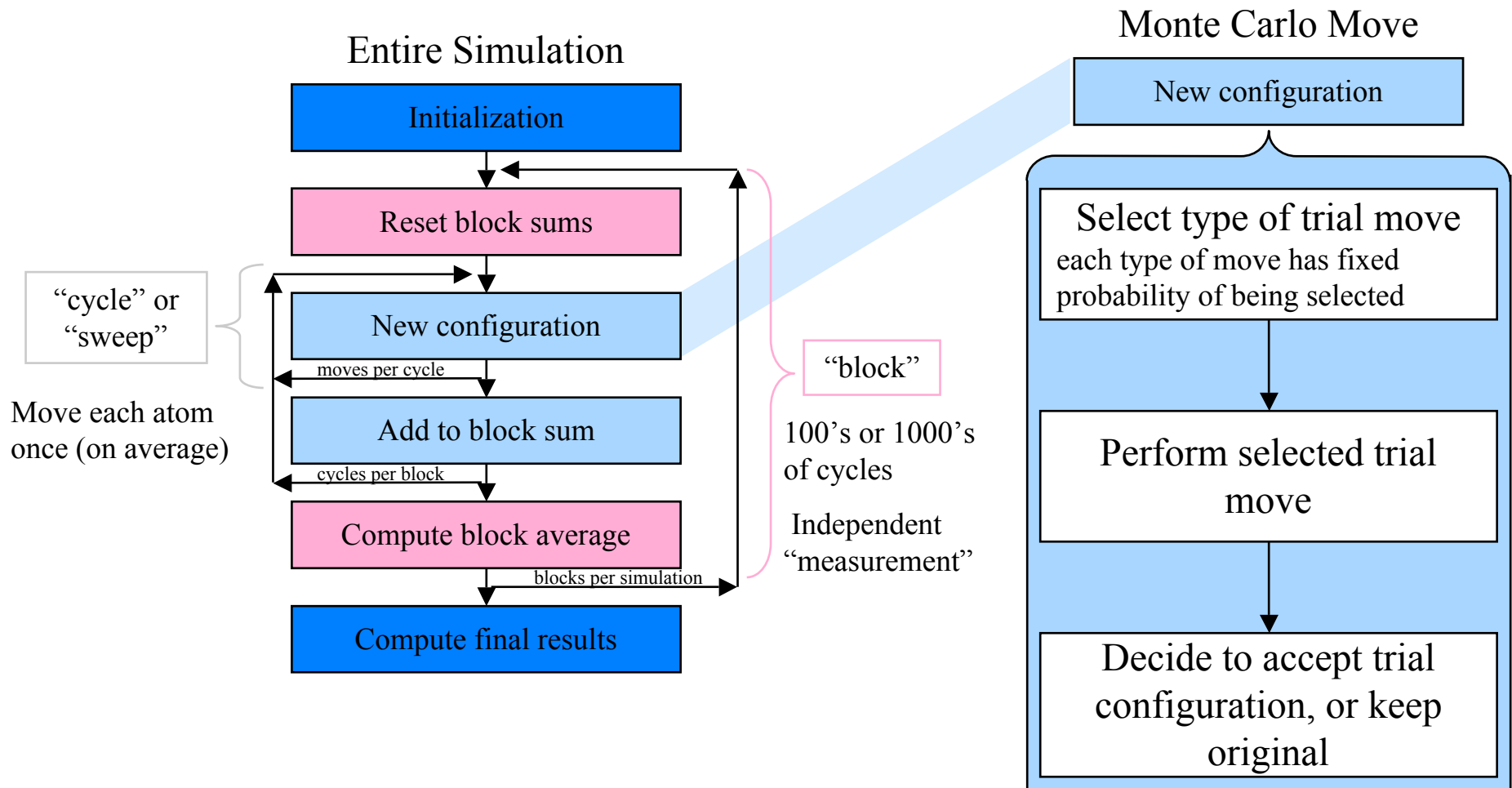


Trial Moves

- A great variety of trial moves can be made
- Basic selection of trial moves is dictated by choice of ensemble
 - almost all MC is performed at constant T
 - no need to ensure trial holds energy fixed
 - must ensure relevant elements of ensemble are sampled
 - all ensembles have molecule displacement, rotation; atom displacement
 - isobaric ensembles have trials that change the volume
 - grand-canonical ensembles have trials that insert/delete a molecule
- Significant increase in efficiency of algorithm can be achieved by the introduction of clever trial moves
 - reptation, crankshaft moves for polymers
 - multi-molecule movements of associating molecules
 - many more



General Form of MC Algorithm



Monte Carlo Integrator

- Here's the core method in *IntegratorMC* class

public class IntegratorMC extends Integrator

```
public void doStep() {
    int i = (int)(rand.nextDouble()*frequencyTotal); //select trial
    MCMove trialMove = firstMove;
    while((i-=trialMove.getFrequency()) >= 0) {
        trialMove = trialMove.nextMove();
    }
    trialMove.doTrial(); //perform trial move and decide acceptance
}
```

```
run() method in Integrator
public void run() {
    stepCount = 0;
    int iieCount = interval+1;
    while(stepCount < maxSteps) {
        while(pauseRequested)
            doWait();
        if(resetRequested)
            {doReset();}
        if(haltRequested) break;
        doStep();
        if(--iieCount == 0) {
            fireIntervalEvent();
            iieCount = interval;
        }
        if(doSleep) {
        }
        stepCount++;
    } //end of while loop
    fireIntervalEvent();
} //end of run method
```

- All the work is done by the MCMove class

public class MCMove

```
public void doTrial() {
    nTrials++;
    thisTrial();
    if(relaxation && nTrials > adjustInterval*frequency) {adjustStepSize();}
}
```

Monte Carlo Integrator

- Here's the core method in *IntegratorMC* class

```
public class IntegratorMC extends Integrator
```

```
public void doStep() {  
    int i = (int)(rand.nextDouble()*frequencyTotal); //select trial  
    MCMove trialMove = firstMove;  
    while((i-=trialMove.getFrequency()) >= 0) {  
        trialMove = trialMove.nextMove();  
    }  
    trialMove.doTrial(); //perform trial move and decide acceptance  
}
```

- All the work is done by the MCMove class

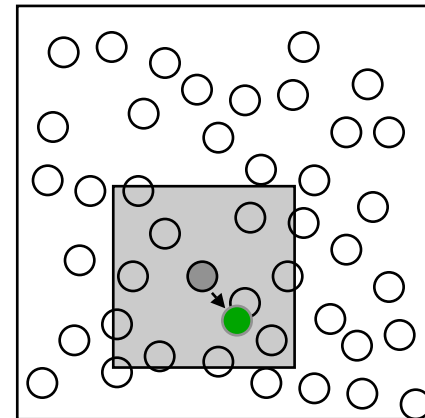
```
public class MCMove
```

```
public void doTrial() {  
    nTrials++;  
    thisTrial();  
    if(relaxation && nTrials > adjustInterval*frequency) {adjustStepSize();}  
}
```

Displacement Trial

- Gives new configuration of same volume and number of molecules
- Basic trial:
 - displace a randomly selected atom to a point chosen with uniform probability inside a cubic volume of edge $2d$ centered on the current position of the atom

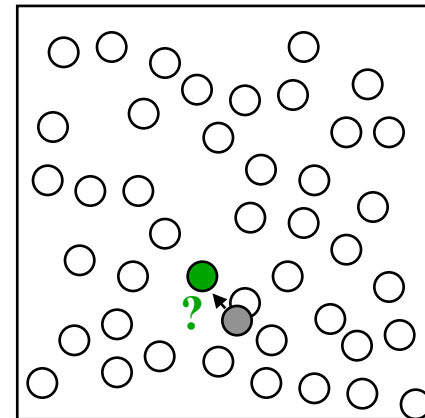
*Move atom to
point chosen
uniformly in
region*



Displacement Trial

- Gives new configuration of same volume and number of molecules
- Basic trial:
 - displace a randomly selected atom to a point chosen with uniform probability inside a cubic volume of edge $2d$ centered on the current position of the atom

*Consider
acceptance of new
configuration*

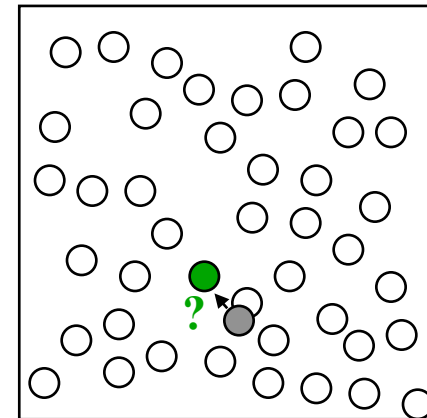


Displacement Trial

- Gives new configuration of same volume and number of molecules
- Basic trial:
 - displace a randomly selected atom to a point chosen with uniform probability inside a cubic volume of edge $2d$ centered on the current position of the atom
- Limiting probability distribution
 - canonical ensemble

$$\pi(\mathbf{r}^N)d\mathbf{r}^N = \frac{1}{Z_N} e^{-\beta U(\mathbf{r}^N)} d\mathbf{r}^N$$

Examine underlying transition probabilities to formulate acceptance criterion



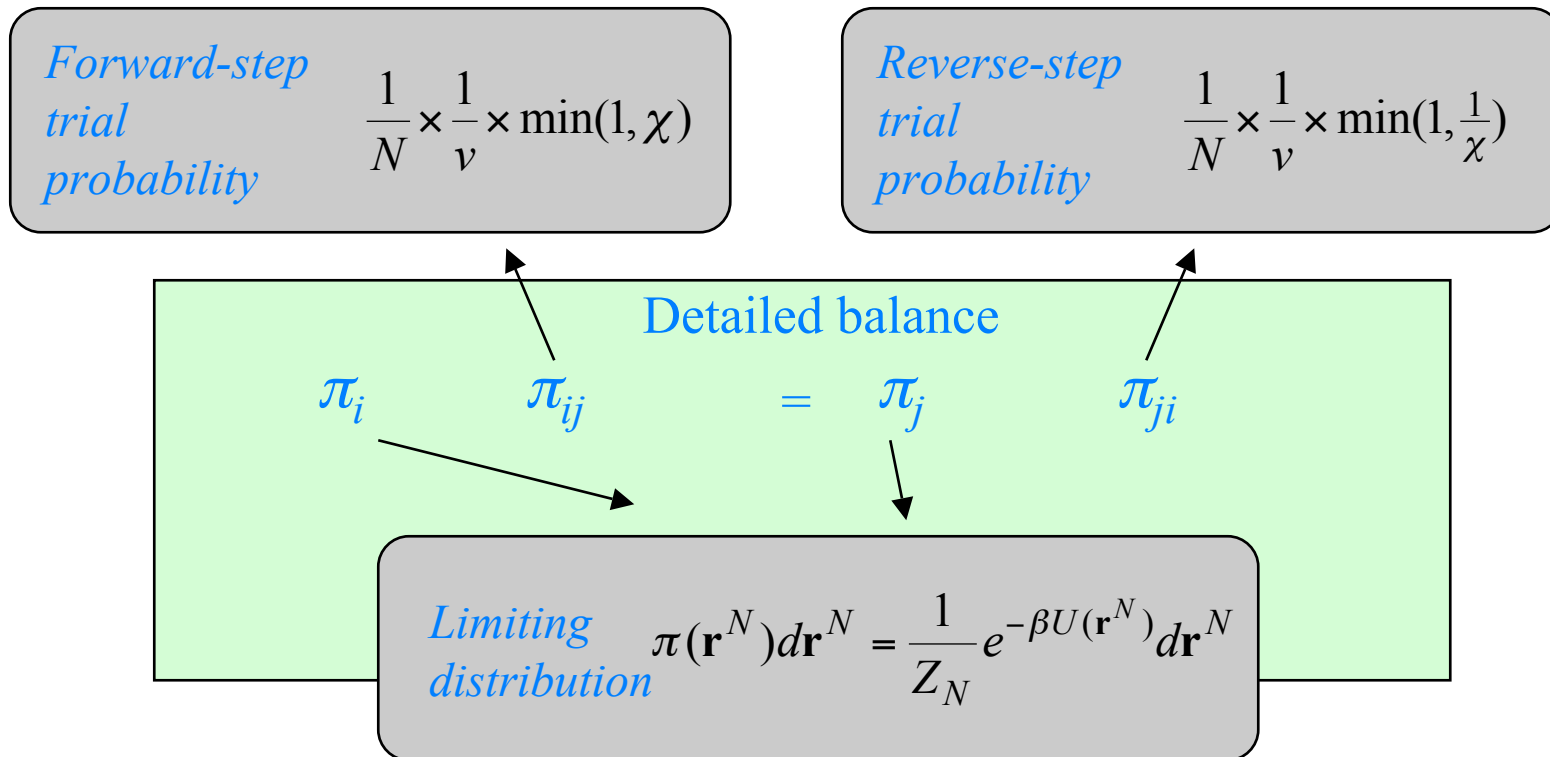
Analysis of Trial Probabilities

- Detailed specification of trial moves and probabilities

Event [reverse event]	Probability [reverse probability]	
Select molecule k [select molecule k]	1/N [1/N]	<i>Forward-step trial probability</i> $\frac{1}{N} \times \frac{1}{v} \times \min(1, \chi)$
Move to r^{new} [move back to r^{old}]	$v = (2\delta)^d \rightarrow 1/v$ [1/v]	
Accept move [accept move]	$\min(1, \chi)$ [min(1, 1/χ)]	<i>Reverse-step trial probability</i> $\frac{1}{N} \times \frac{1}{v} \times \min(1, \frac{1}{\chi})$

χ is formulated to satisfy detailed balance

Analysis of Detailed Balance



Analysis of Detailed Balance

*Forward-step
trial
probability*

$$\frac{1}{N} \times \frac{1}{v} \times \min(1, \chi)$$

*Reverse-step
trial
probability*

$$\frac{1}{N} \times \frac{1}{v} \times \min(1, \frac{1}{\chi})$$

Detailed balance

$$\pi_i \quad \pi_{ij} \quad = \quad \pi_j \quad \pi_{ji}$$
$$\frac{e^{-\beta U^{old}} d\mathbf{r}^N}{Z_N} \left[\frac{1}{N} \times \frac{1}{v} \times \min(1, \chi) \right] = \frac{e^{-\beta U^{new}} d\mathbf{r}^N}{Z_N} \left[\frac{1}{N} \times \frac{1}{v} \times \min(1, \frac{1}{\chi}) \right]$$

*Limiting
distribution*

$$\pi(\mathbf{r}^N) d\mathbf{r}^N = \frac{1}{Z_N} e^{-\beta U(\mathbf{r}^N)} d\mathbf{r}^N$$

Analysis of Detailed Balance

*Forward-step
trial
probability*

$$\frac{1}{N} \times \frac{1}{v} \times \min(1, \chi)$$

*Reverse-step
trial
probability*

$$\frac{1}{N} \times \frac{1}{v} \times \min(1, \frac{1}{\chi})$$

Detailed balance

$$\pi_i \pi_{ij} = \pi_j \pi_{ji}$$

$$\frac{\cancel{e^{-\beta U^{old}}} \cancel{d\mathbf{r}^N}}{\cancel{Z_N}} \left[\frac{1}{\cancel{N}} \times \frac{1}{\cancel{v}} \times \min(1, \chi) \right] = \frac{\cancel{e^{-\beta U^{new}}} \cancel{d\mathbf{r}^N}}{\cancel{Z_N}} \left[\frac{1}{\cancel{N}} \times \frac{1}{\cancel{v}} \times \min(1, \frac{1}{\chi}) \right]$$

$$e^{-\beta U^{old}} \chi = e^{-\beta U^{new}}$$

$$\chi = e^{-\beta(U^{new} - U^{old})}$$

Acceptance probability

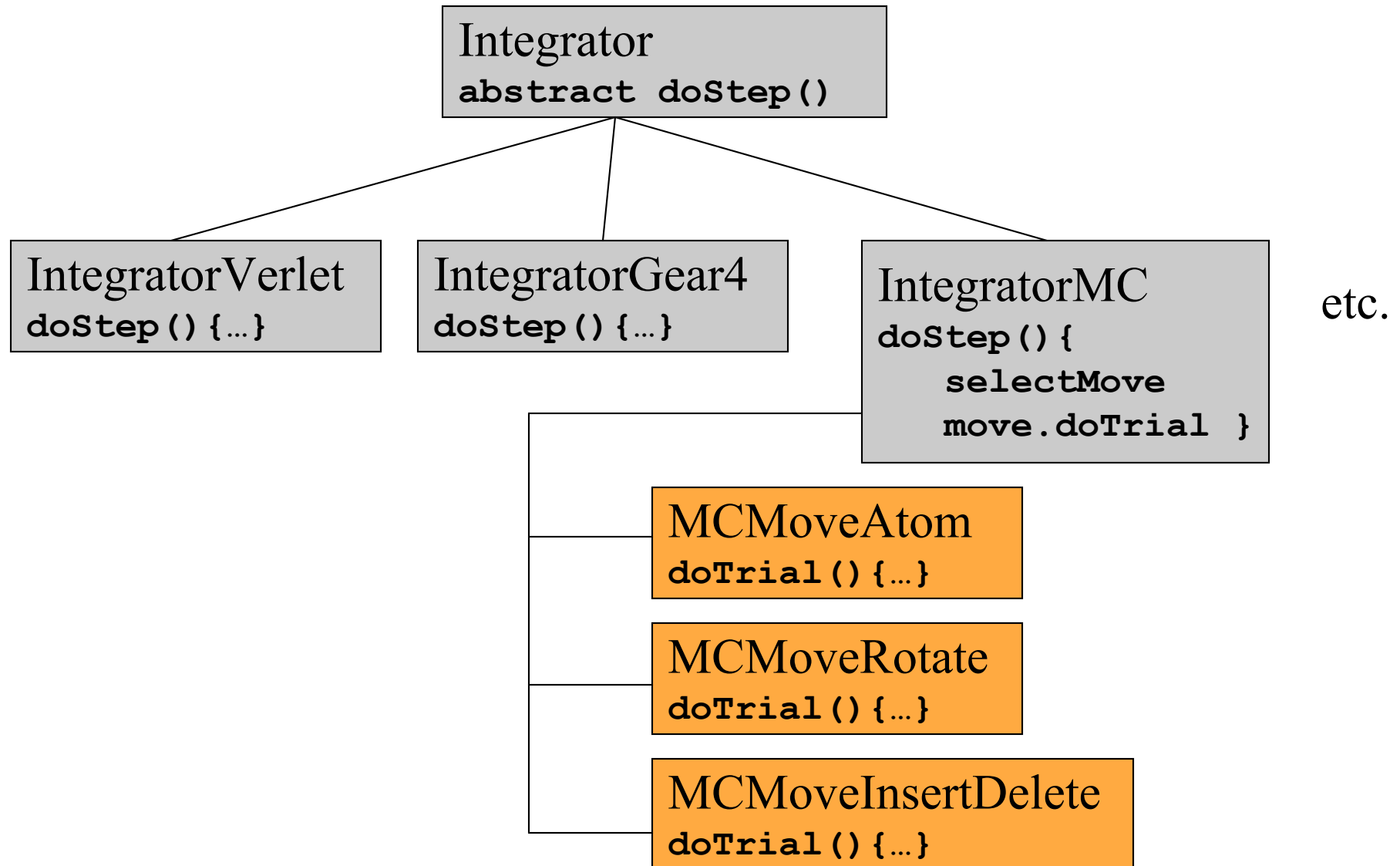
Examination of Java Code

```
public class MCMoveAtom extends MCMove
```

```
public void thisTrial(Phase phase) {  
  
    double uOld, uNew;  
    if(phase.atomCount==0) {return;}           //no atoms to move  
  
    Atom a = phase.randomAtom();  
  
    uOld = phase.potentialEnergy.currentValue(a); //calculate its contribution to the energy  
    a.displaceWithin(stepSize);                 //move it within a local volume  
    uNew = phase.potentialEnergy.currentValue(a); //calculate its new contribution to energy  
  
    if(uNew < uOld) {                           //accept if energy decreased  
        nAccept++;  
        return;  
    }  
    if(uNew >= Double.MAX_VALUE || //reject if energy is huge or doesn't pass test  
        Math.exp(-(uNew-uOld)/parentIntegrator.temperature) < rand.nextDouble()) {  
        a.replace();                          //...put it back in its original position  
        return;  
    }  
    nAccept++;                                 //if reached here, move is accepted  
}
```

- Time for a demonstration...

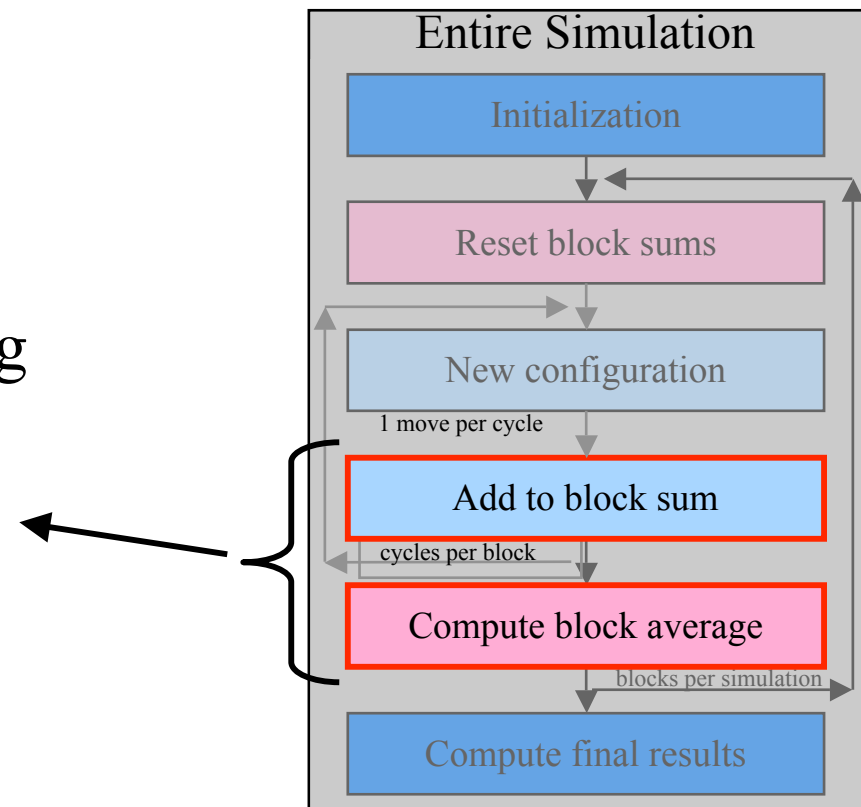
Summary: Integrator Class Structure



etc.

Simulation Element: Meter

- Measures some property of interest
- Keeps average
- Coordinates block averaging and error analysis
- Can keep history and histograms



Integrator Events

- Integrator notifies listeners when it has made some progress
- Remember core method in *Integrator* class

```
public void run() {
    stepCount = 0;
    int iieCount = interval+1;
    while(stepCount < maxSteps) {
        while(pauseRequested) doWait();
        if(resetRequested) {doReset(); resetRequested = false;}
        if(haltRequested) break;
        doStep(); //abstract method in Integrator. subclasses implement algorithms (MD/MC)
        if(--iieCount == 0) { //count down to determine when a cycle is completed
            fireIntervalEvent(intervalEvent); //notify listeners of completion of cycle
            iieCount = interval;
        }
        if(doSleep) { //slow down simulation so display can keep up
            try { Thread.sleep(sleepPeriod); }
            catch (InterruptedException e) { }
        }
        stepCount++;
    } //end of while loop
    fireIntervalEvent(new IntervalEvent(this, IntervalEvent.DONE));
} //end of run method
```

All Meters are Integrator Listeners

- On receiving event, meter takes action to measure its property

```
public abstract class MeterAbstract
```

```
public void intervalAction(Integrator.IntervalEvent evt) {  
    if(!active) return;  
    if(evt.type() != Integrator.IntervalEvent.INTERVAL) return; //don't act on start, done, etc  
    if(--iieCount == 0) {  
        iieCount = updateInterval;  
        accumulator.add(currentValue());  
    }  
}  
  
public abstract double currentValue();
```

- Each meter implements a different **currentValue** method

Accumulator Does the Bookkeeping

```
public class Accumulator
```

```
public void add(double value) {
    mostRecent = value;
    if(Double.isNaN(value)) return;

    blockSum += value;
    if(--blockCountDown == 0) { //count down to zero to determine completion of block
        blockSum /= blockSize; //compute block average
        sum += blockSum;
        sumSquare += blockSum*blockSum;
        count++;
        if(count > 1) {
            double avg = sum/(double)count;
            error = Math.sqrt((sumSquare/(double)count - avg*avg)/(double)(count-1));
        }
        //reset blocks
        mostRecentBlock = blockSum;
        blockCountDown = blockSize;
        blockSum = 0.0;
    }
    if(histogramming) histogram.addValue(value);
    if(historying) history.addValue(value);
}
```

Some Meter Implementations

public class MeterDensity

```
public double currentValue() {  
    return phase.moleculeCount / phase.volume();  
}
```

public class MeterKineticEnergy

```
public double currentValue() {  
    double ke = 0.0;  
    atomIterator.reset();  
    while(atomIterator.hasNext()) {  
        ke += atomIterator.next().kineticEnergy();  
    }  
    return ke;  
}
```

public class MeterPressureHard

```
public double currentValue() {  
    double t = integrator.elapsedTime();  
    if(t > t0) {  
        double flux = -virialSum/((t-t0)*D*phase.atomCount());  
        value = integrator.temperature() + flux;  
        t0 = t;  
        virialSum = 0.0;  
    }  
    return value;  
}
```

- Time for a demonstration...

Simulation Elements: Displays and Devices

- Displays: Output of information
 - Integrator listener...update when informed of integrator's progress
 - Plot: graphical display
 - Table: tabular display
 - Box: single value in a box
 - Log: write results to a log file
 - Phase: animation of atom positions
- Devices: Input of data and manipulation of simulation
 - Slider: connects to a property
 - Table: property-sheet like adjustment of values
 - Temperature chooser: comboBox selection of temperatures
- Most Displays/Devices oriented toward interactive use
- Let's demonstrate...

Simulation Elements: Species and Potentials

- Species: Data structure
 - Control of number of molecules, atoms per molecule
 - Atom positions, velocities, masses, shapes
 - Examples
 - Spheres, walls
 - Specific elements and compounds (H_2O) appear as special cases
 - There is no “Lennard-Jones species”
- Potentials: How atoms interact
 - Soft potentials
 - Energy, force, virial
 - Hard potentials
 - Energy, collision time, collision dynamics
 - Examples
 - Lennard-Jones, hard spheres, square well, etc.
- Let’s demonstrate...

Simulation Element: Phase

- Phase collects all the molecules that are interacting
- Holds boundary conditions
 - Let's see...
- Normally have one phase per simulation...
- ...but some simulation techniques work with several phases simultaneously
 - Methods for evaluation of phase equilibria
 - Gibbs ensemble
 - Gibbs-Duhem integration
 - Sampling-enhancement methods
 - Parallel tempering
- Let's build a Gibbs-ensemble simulation...

Simulation Element: Space

- “Behind the scenes” simulation element
- Provides data structures for positions, velocities, rotations, tensors and phase boundaries

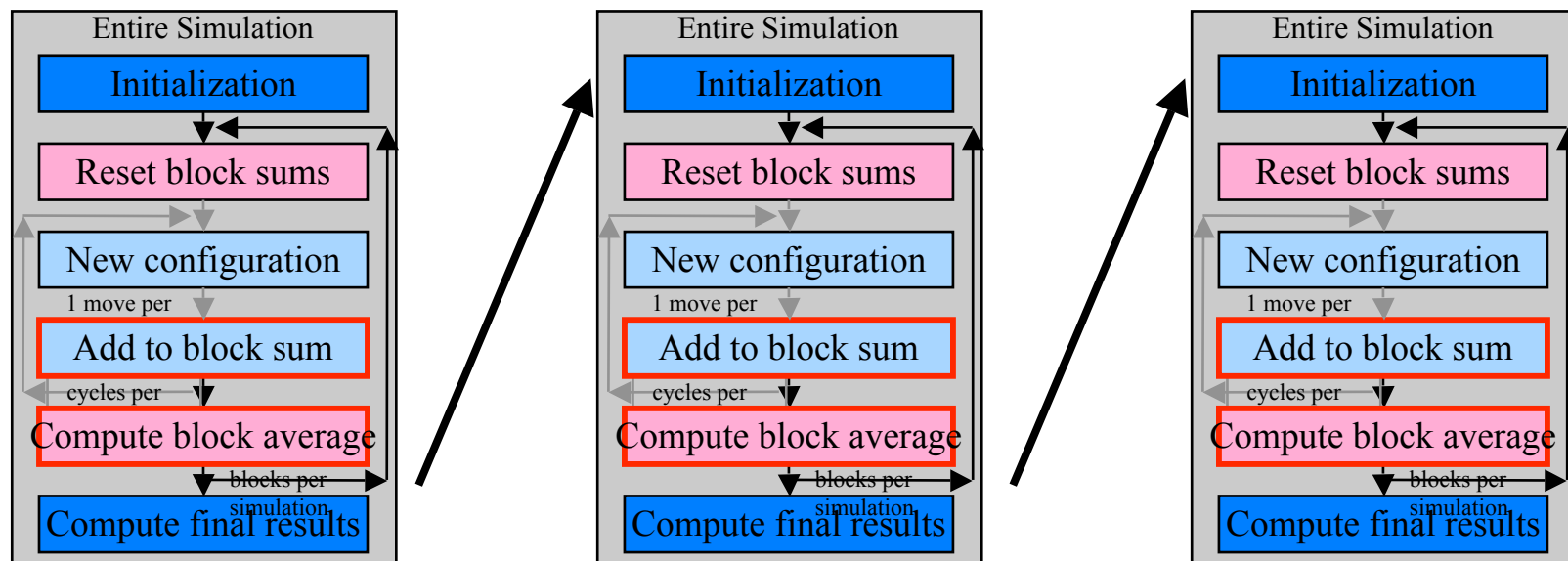
```
public class Space2D.Vector
```

```
public double x, y;  
public void E(Vector u) {x = u.x; y = u.y;} //set one vector equal to another  
public void PE(Vector u) {x += u.x; y += u.y;} //add one vector to another  
public void TE(double a) {x *= a; y *= a;} //multiply a vector by a scalar  
public void TE(Vector u) {x *= u.x; y *= u.y;} //multiply vectors term by term  
public double squared() {return x*x + y*y;} //square of vector magnitude  
public double dot(Vector u) {return x*u.x + y*u.y;} //dot product of vector with another  
etc.
```

- Permits arbitrary specification of spatial dimension
- Must select when beginning to construct simulation
- For example...

Simulation Element: Controller

- Controls when simulations start and what happens when they finish. For example...
 - On and off with a button
 - Equilibration, production, finish
- Perform series of simulations
 - Thermodynamic integration, Gibbs-Duhem integration
 - Set state conditions and turn integrator(s) on and off



Saving/Exporting Simulations with Etomica

- **Serialization (Java technology for saving objects)**
 - Saves simulation in its present state (even after running)
 - Can read back into Etomica and continue
 - Can import as an applet to a web page (almost)
- **Write out as Java source**
 - Saves changes made by Etomica editor, but not changes from running simulation
 - Save as application or (to be implemented) applet
 - Need to compile and (if applet) write html page separately
 - Some further development needed
 - Let's see...

Etomica is Extensible

- New components easily added
 - Selection of simulation elements is constructed on-the-fly, by examining files in class path
- Introspective
 - Editable properties of components discovered automatically
 - Classes should conform to (extended) Javabeans specification

```
public class SpeciesDisks
```

```
public double getMass() {return mass;}  
public void setMass(double mass) {mass = m;}  
public Dimension getMassDimension() {return Dimension.MASS;}  
  
public double getDiameter() {return diameter;}  
public void setDiameter(double d) {diameter = d;}  
public Dimension getDiameterDimension() {return Dimension.LENGTH;}
```

Summary: Etomica Class Structure

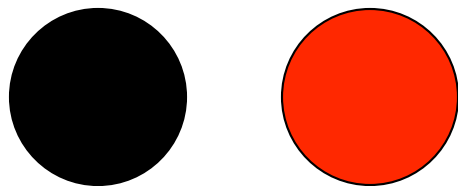
- Simulation
 - Organization and reference
- Space
 - 1-D, 2-D, 3-D, lattice, etc.
 - Vector, Tensor, Boundary
- Phase
 - Container of molecules
- Species
 - Molecule data structure
- Potential
 - Molecule interactions
- Controller
 - Protocol for simulation
- Integrator
 - Generation of configurations
- Meter
 - Property measurement
- Modulator
 - Changing of parameters
- Display
 - Presentation of data
- Device
 - Interaction with simulation
- Support
 - Units, Iterators, Atom types, Actions, Color schemes, Constants, Defaults, Configurations

Application: Piston-cylinder Apparatus

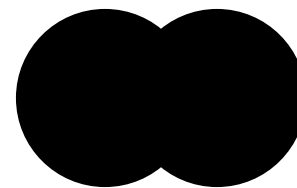
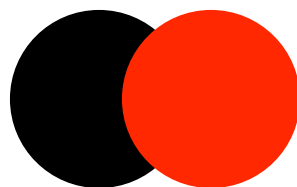
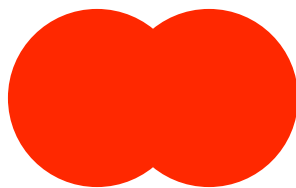
- Simulation as a teaching laboratory for thermodynamics
- “Take these data and compute...”
- Piston-cylinder applet
- Example of a homework problem
- And its solution

Application: Reaction Equilibrium (under construction)

- Simple model permits analysis of reaction kinetics and equilibria
- Two 1-atom “radicals”...



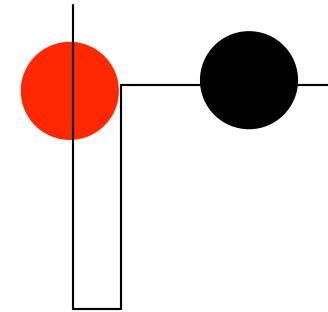
- ...combine to form 3 molecular species



Application: Reaction Equilibrium (under construction)

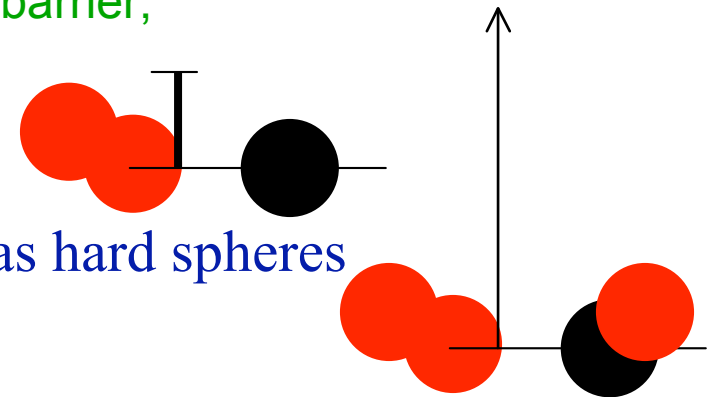
- Simple potentials govern atom interactions, but depend on binding state

- Two radicals bind with square well potential



- Radical/bound-atom interaction has a barrier

- If collide with enough energy to surmount barrier, bound atom dissociates



- Two non-mutually bound atoms interact as hard spheres

- Aim is to measure equilibrium concentrations and reaction rates as functions of potential parameters

Ways to Use Etomica

- Educator or casual user
 - Write simple new classes as needed, using existing classes for guidance
 - Construct simulation with development environment, export source
 - Compile and use as desired
- Advanced user or researcher
 - Construct draft simulation with development environment, export source
 - Extend by editing source directly
 - More efficient, more options
 - Develop new classes as needed
 - Compile and run standalone, or import back into Etomica to run interactively
- Student
 - Run applets/applications developed with Etomica
 - Do “design” problems that employ development environment

Educational Applications of Etomica

- Summer 1999
 - High-school workshop in Computational Chemistry at UB Center for Computational Research
- Spring 2000
 - CE530 Molecular Simulation course at UB
- Summer 2000
 - Repeat of high-school workshop
 - REU student
- Fall 2000
 - CE 526 Statistical Mechanics course at UB
- Fall 2001
 - Freshman/Sophomore honors seminar on molecular simulation
- Future growth
 - Multi-investigator project to develop educational molecular simulation modules
 - Proposal pending to interact with Buffalo area science teachers to examine implementation in high-school curricula

Ongoing Research Applications

- Jhumpa Adhikari
 - Free-energy calculations in solids
 - New Controller, Integrator, Meter
 - Phase diagrams for mixtures
 - New Controller, MCMove, Meter
- Jayant Singh
 - Surface tension in associating fluids
 - New Potential, Meter, MCMove
- Nandou Lu
 - New approach to isobaric and grand-canonical simulations
 - New MCMove classes
 - Free-energy method for polymers
 - New MCMove, Meter
- Sang-kyu Kwak
 - Benchmarking
 - Study of solid defects planned

Some More Applets

- Early prototypes
- Molecular simulation course

Acknowledgments

- National Science Foundation
- UB Educational Technology Fund
- Students
 - Bryan Mihalick
 - Dan Barnes
- Java plotting classes
 - *ptplot*, from the UC Berkeley
 - `ptolemy.eecs.berkeley.edu`
- Etomica web site
 - `www.ccr.buffalo.edu/etomica`
 - Installer for Etomica development environment
 - All source files for Etomica API
 - Documentation
 - Additional links