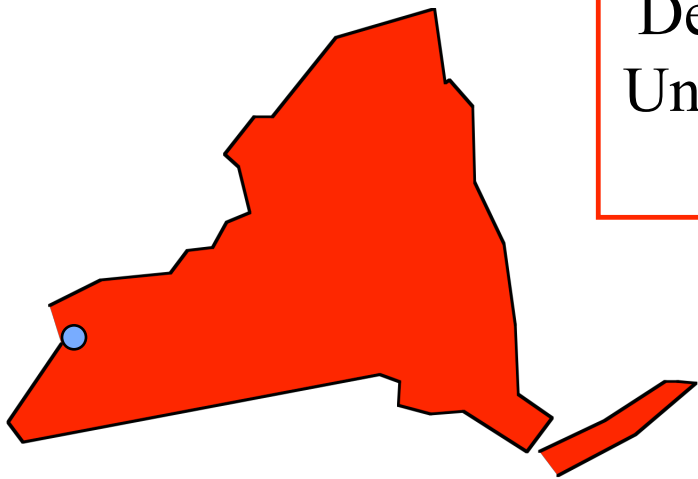


All About Meters and Data

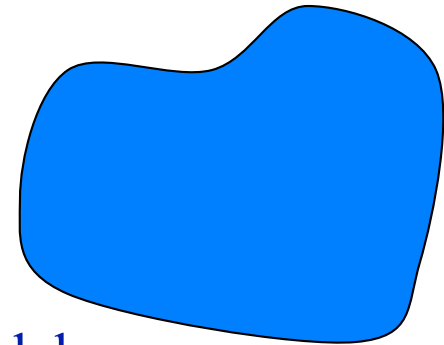
David A. Kofke

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



Statistical Mechanical Averages

- Mechanical properties
 - Value can be associated with each configuration
 - Examples
 - Energy
 - Pressure
 - Density
 - Structure
- Entropic properties
 - Value describes not one configuration, but the whole set
 - Examples
 - Entropy (multiplicity)
 - Free energy
 - Chemical potential
- Analogy
 - Average depth of a lake versus the area of the lake



Ensemble Averages

- Consider molecular configurations in proportion to their statistical mechanical weight
 - E.g., Boltzmann distribution

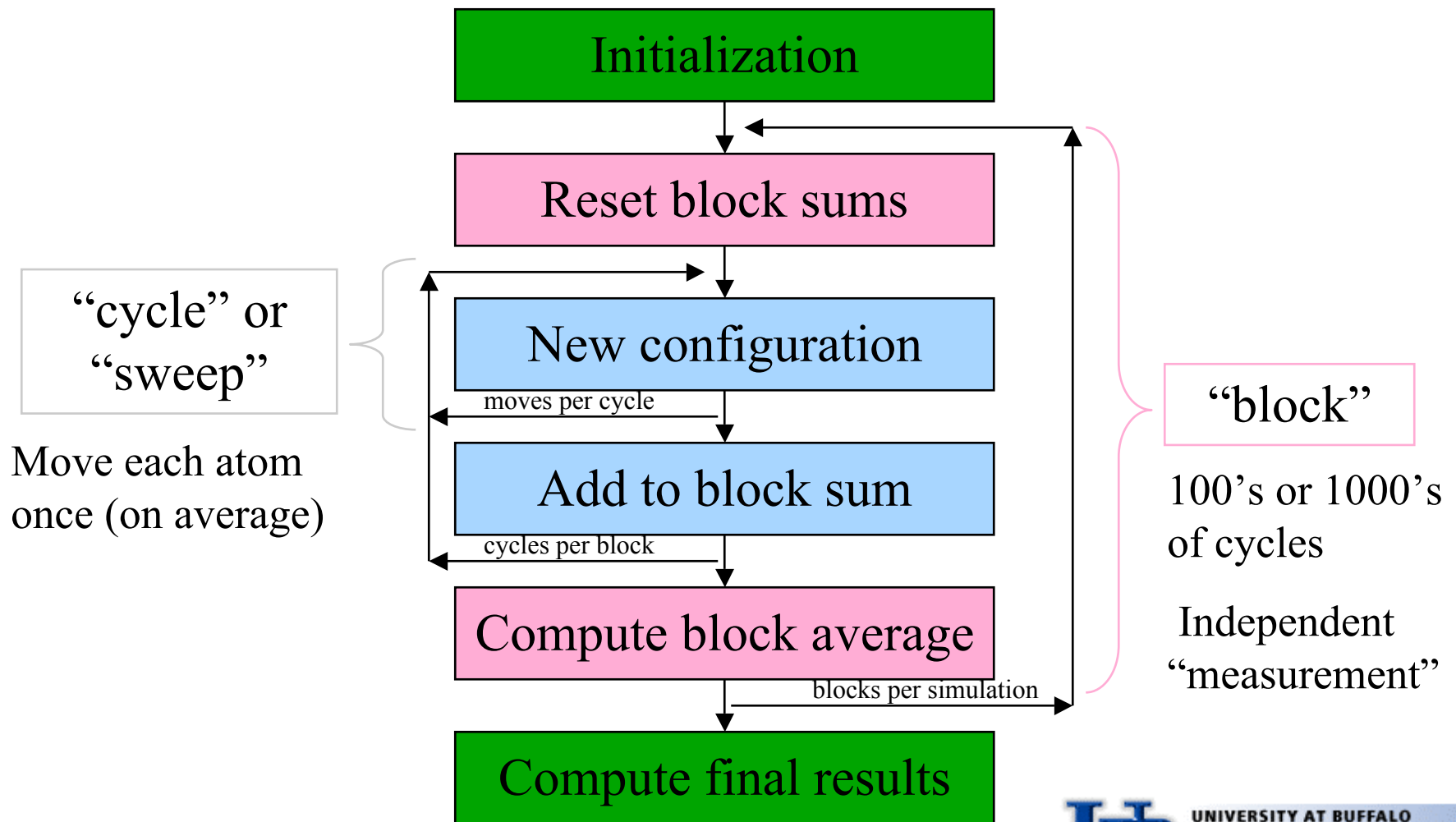
$$\pi_i = \frac{1}{Q} e^{-\beta U_i}$$

- Accumulate average of mechanical property over all configurations
 - E.g., Average internal energy

$$\begin{aligned} U &= \langle U \rangle_{\pi} \\ &= \int d\mathbf{r}^N U(\mathbf{r}^N) \pi(\mathbf{r}^N) \end{aligned}$$

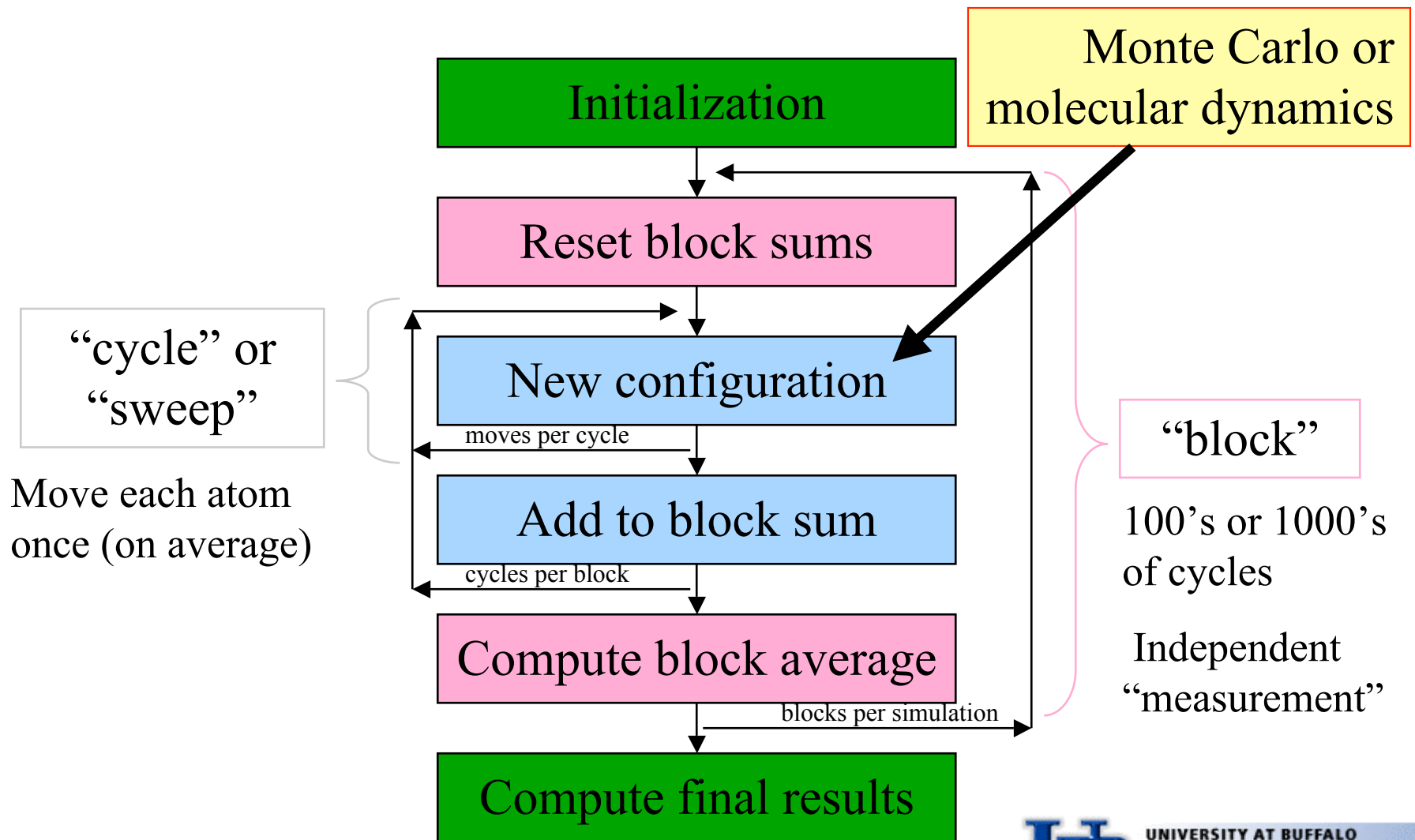
Measurements in Molecular Simulation

- General structure of a simulation



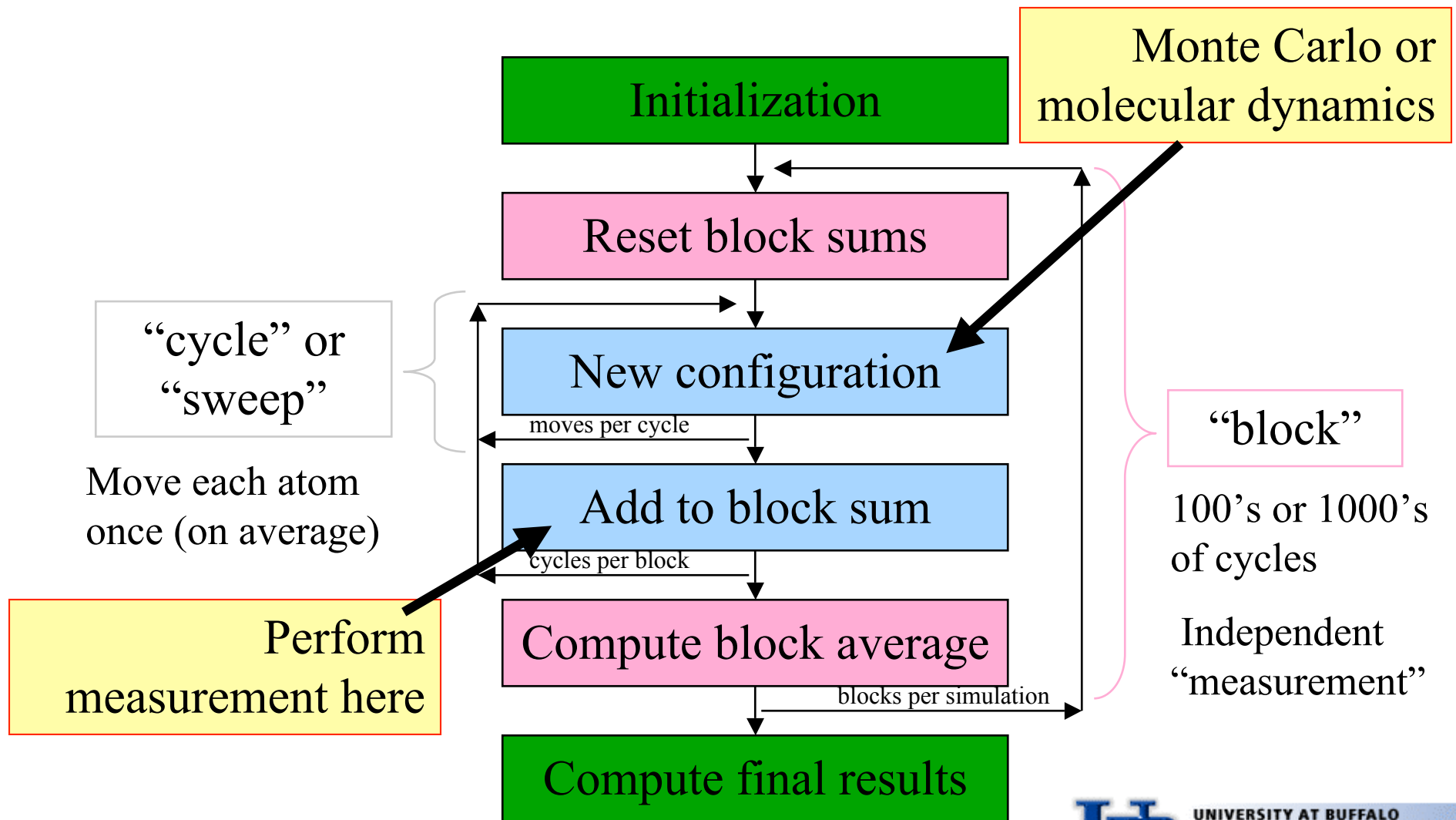
Measurements in Molecular Simulation

- General structure of a simulation



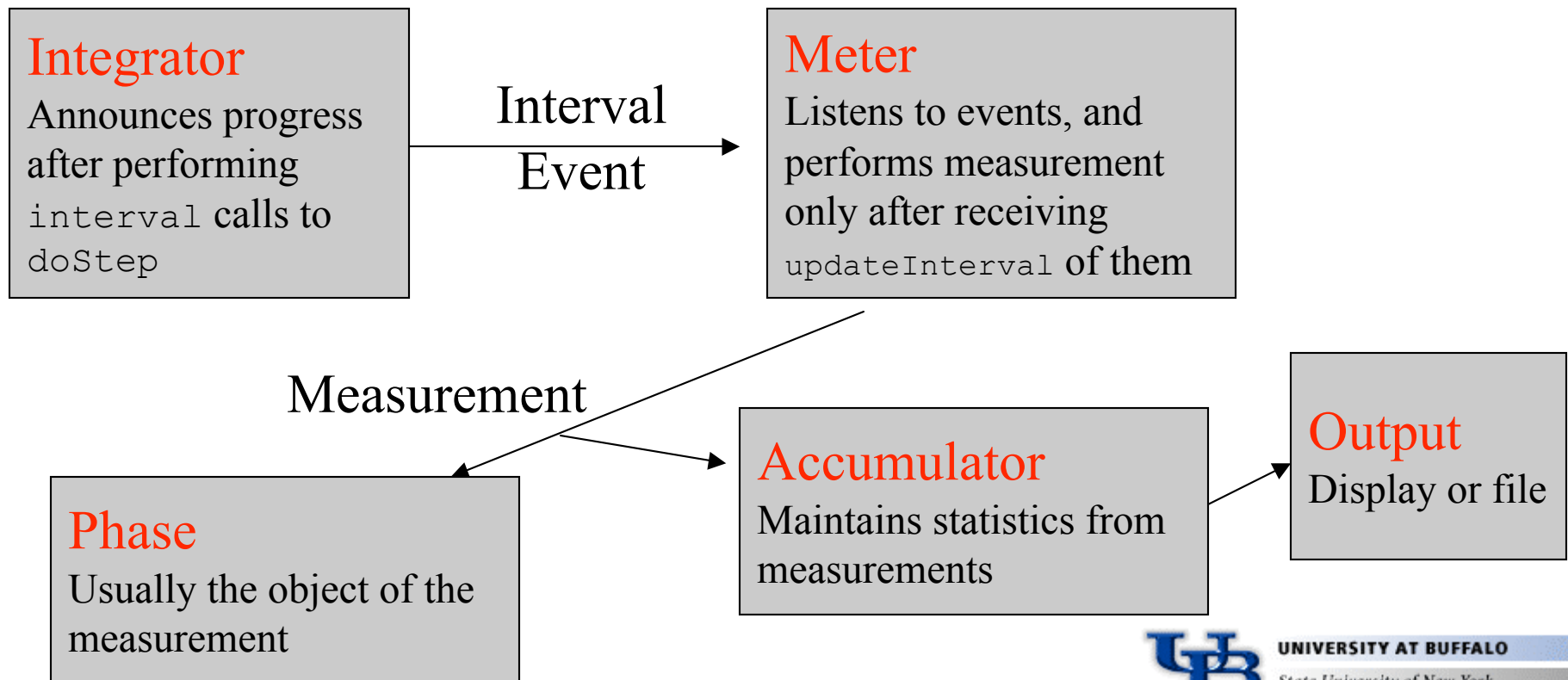
Measurements in Molecular Simulation

- General structure of a simulation

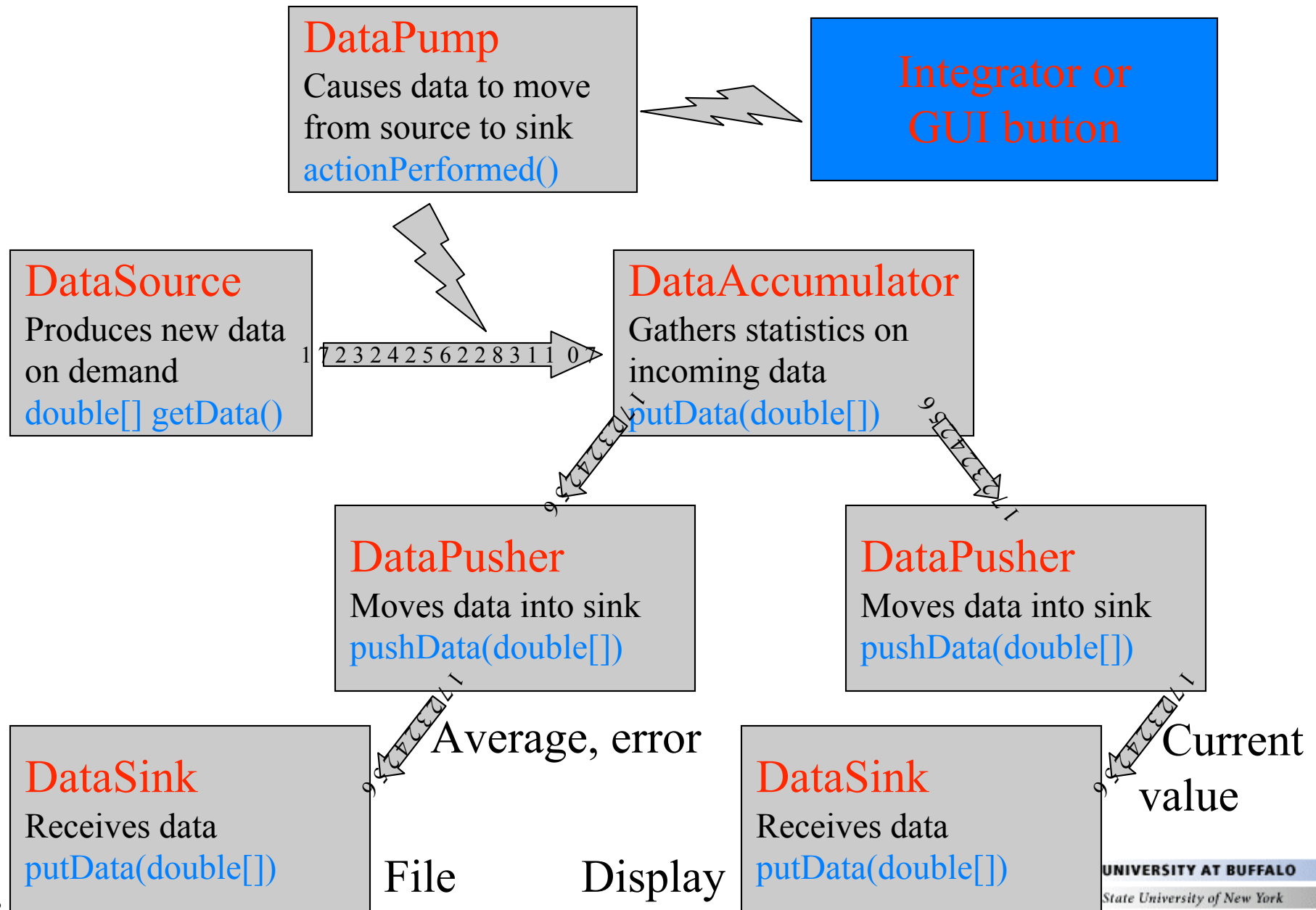


Property Measurement in Etomica

- Meters perform measurements on phases
 - Conducted on Integrator's thread, so system is static while measurement is performed
- Chain of events leading to a measurement



Data Flows in Etomica



DataSource, DataSink, Meter

- DataSink – Interface for classes that receive data

```
public interface DataSink {  
    public abstract void putData(double[] values);  
}
```

- DataSource – Interface for classes that provide data

```
public interface DataSource {  
    public abstract double[] getData();  
}
```

- MeterAbstract – DataSource that acts on a Phase

```
public interface DataSource {  
    public abstract double[] getData();  
}
```

DataPusher

- Abstract class
- Holds one or more data sinks, and pushes data into them on request
- Has methods to manage data sinks

```
public abstract class DataPusher {  
  
    protected void pushData(double[] data) {  
        for(int i=dataSinkList.length-1; i>=0; i--) {  
            dataSinkList[i].putData(data);  
        }  
    }  
  
    public void addDataSink(DataSink dataSink) {  
        ...  
    }  
  
    public void removeDataSink(DataSink dataSink) {  
        ...  
    }  
  
}
```

DataPump

- Extends DataPusher
- Holds a DataSource, and moves data from it to the sinks
- Implements action
 - Typically activated via Integrator IntervalEvent, or GUI action

```
public class DataPump {  
  
    public void actionPerformed() {  
        pushData(dataSource.getData());  
    }  
}
```

- IntervalActionAdapter
 - Transmits an Integrator IntervalEvent into an Action

```
public class IntervalActionAdapter {  
    public void intervalAction(Integrator.IntervalEvent evt)  
    {  
        if (--iieCount == 0) {  
            iieCount = actionInterval;  
            action.actionPerformed();  
        }  
    }  
}
```

DataPipe

- Abstract, extends DataPusher
- Implements DataSink
- Takes data given to it, does something to it, and passes on new data

DataAccumulator

- Abstract, extends DataPipe implements DataSource
- Collects statistics on data given to it
- Passes data down stream only after some interval of data collecting

```
public abstract class DataAccumulator {  
  
    public void putData(double[] newData) {  
        if(!active) return;  
        addData(newData);  
        if (--putCount == 0) {  
            putCount = pushInterval;  
            pushData(getData());  
        }  
    }  
  
    public abstract void addData(double[] data);  
}
```

DataAccumulator subclasses

- AccumulatorAverage
 - Generates statistics for average, error, standard deviation, and more

```
public class AccumulatorAverage extends DataAccumulator {  
  
    public void addData(double[] value) {  
        for(int i=nDataMinus1; i>=0; i--) {  
            double v = value[i];  
            mostRecent[i] = v;  
            blockSum[i] += v;  
            blockSumSq[i] += v*v;  
        }  
    }  
}
```

- AccumulatorHistogram
 - Generates histogram of data given to it
- AccumulatorHistory
 - Records history of data given to it

DataAccumulator subclasses

- AccumulatorAverage
 - Generates statistics for average, error, standard deviation, and more

```
public class AccumulatorAverage extends DataAccumulator {  
  
    public void addData(double[] value) {  
        for(int i=nDataMinus1; i>=0; i--) {  
            double v = value[i];  
            mostRecent[i] = v;  
            blockSum[i] += v;  
            blockSumSq[i] += v*v;  
        }  
    }  
    public double[] getData() {  
        ...  
    }  
}
```

- AccumulatorHistogram
 - Generates histogram of data given to it
- AccumulatorHistory
 - Records history of data given to it

Display

- Abstract class that puts information to screen
- Sometimes implements DataSink
- Example: DisplayBoxesCAE
 - Designed to take data from AccumulatorAverage

```
public class DisplayBoxesCAE extends Display implements DataSink {  
  
    public void setAccumulator(AccumulatorAverage accumulatorAverage) {  
        this.accumulatorAverage = accumulatorAverage;  
        accumulatorAverage.makeDataPusher(new AccumulatorAverage.Type[] {  
            AccumulatorAverage.MOST_RECENT,  
            AccumulatorAverage.AVERAGE,  
            AccumulatorAverage.ERROR}).addDataSink(this);  
    }  
  
    public void putData(double[] data) {  
        datumC[0] = data[0];  
        currentBox.putData(datumC);  
        datumA[0] = data[1];  
        averageBox.putData(datumA);  
        datumE[0] = data[2];  
        errorBox.putData(datumE);  
    }  
}
```


Putting it together

```
import etomica.*;
import etomica.action.*;
import etomica.data.*;
import etomica.graphics.*;
import etomica.integrator.*;
import etomica.potential.P2HardSphere;
import etomica.space2d.Space2D;

public class HSMD2D extends Simulation {

    public ActivityIntegrate activityIntegrate;
    public AccumulatorAverage pressureAverage;

    public HSMD2D() {
        super(new Space2D());

        IntegratorHard integrator = new IntegratorHard(potentialMaster);
        integrator.setIsothermal(false);
        activityIntegrate = new ActivityIntegrate(integrator);
        getController().addAction(activityIntegrate);
        SpeciesSpheresMono species = new SpeciesSpheresMono(this);
        species.setNMolecules(64);
        Phase phase = new Phase(space);
        P2HardSphere potential = new P2HardSphere(space);
        potentialMaster.setSpecies(potential,new Species[]{species,species});

        // (CONT'D)
```

Putting it together

```
integrator.addIntervalListener(new PhaseImposePbc(phase));
phase.speciesMaster.addSpecies(species);
integrator.addPhase(phase);
integrator.setIsothermal(true);

MeterPressureHard meterPressure = new MeterPressureHard(integrator);
meterPressure.setPhase(phase);
pressureAverage = new AccumulatorAverage();
DataPump pressurePump = new DataPump(meterPressure, pressureAverage);
IntervalActionAdapter pressureAction =
    new IntervalActionAdapter(pressurePump, integrator);
}

/**
 * Demonstrates how this class is implemented.
 */
public static void main(String[] args) {
    HSMD2D sim = new HSMD2D();
    SimulationGraphic graphic = new SimulationGraphic(sim);
    sim.activityIntegrate.setDoSleep(true);
    DisplayBoxesCAE pressureDisplay = new DisplayBoxesCAE();
    pressureDisplay.setAccumulator(sim.pressureAverage);
    graphic.add(pressureDisplay);
    graphic.makeAndDisplayFrame();
} //end of main
}
```