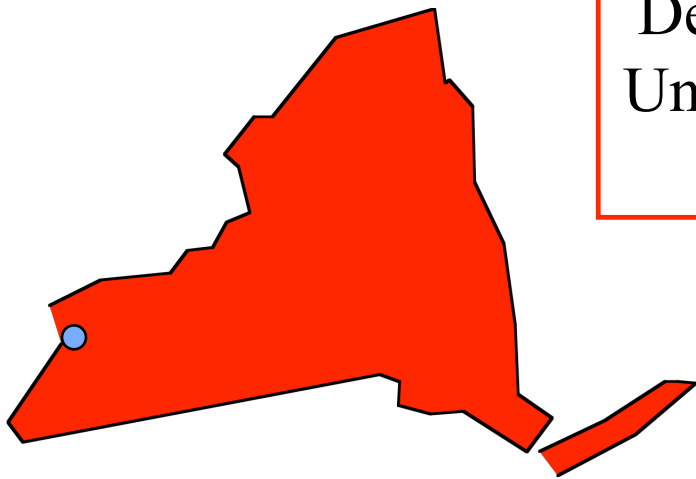


All About Meters

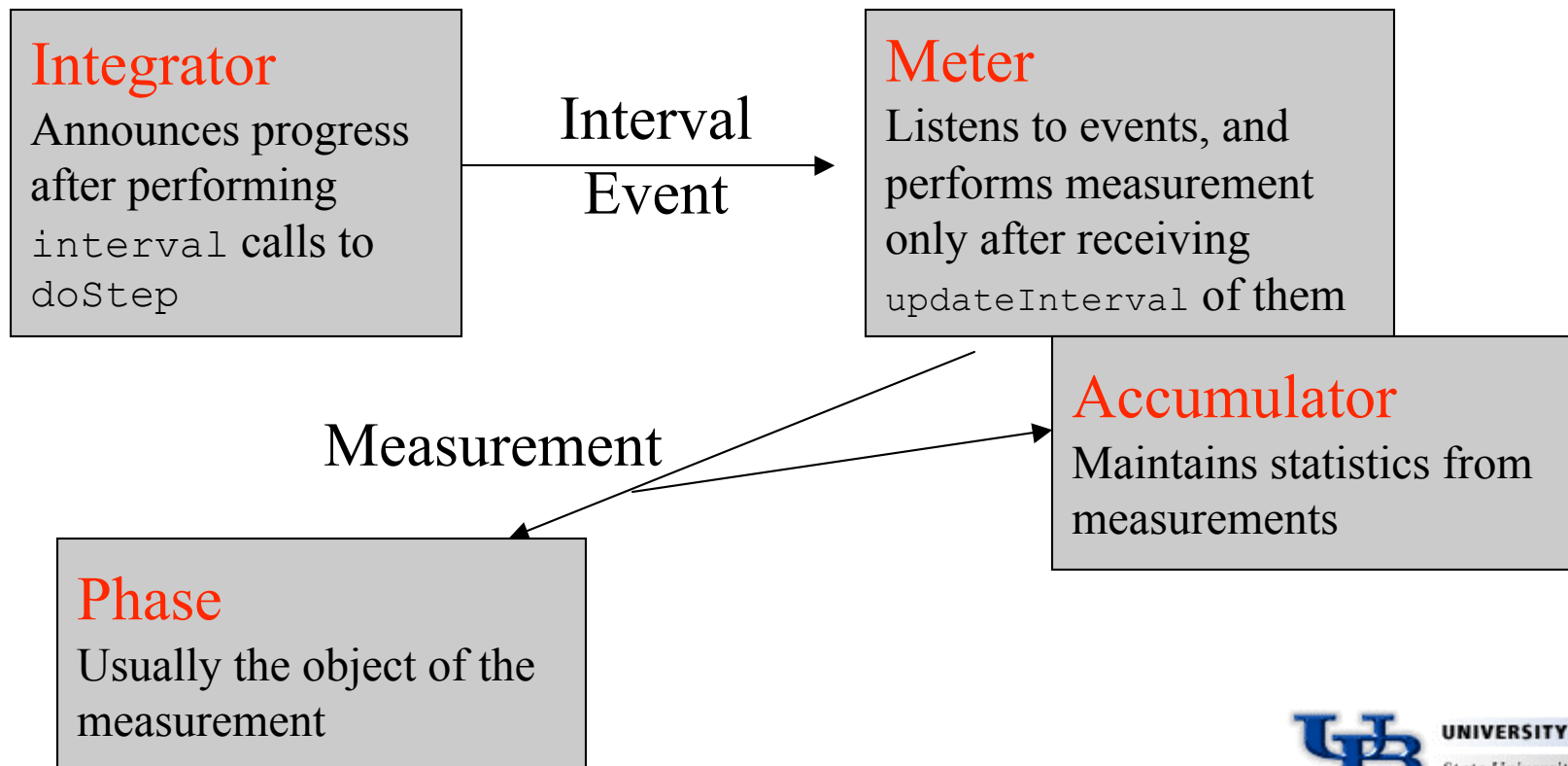
David A. Kofke

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



Measurement of Data

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



Relevant Code. Event Firing

- Inside Integrator

```
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
```

Relevant Code. Event Handling

- Inside MeterAbstract

```
public void intervalAction(Integrator.IntervalEvent evt) {  
    //meter can be turned off  
    if(!active) return;  
  
    //don't act on start, done, initialize events  
    if(evt.type() != Integrator.IntervalEvent.INTERVAL) return;  
  
    //go ahead  
    if(--iieCount == 0) {  
        iieCount = updateInterval;  
        updateSums();  
    }  
}  
  
public abstract void updateSums();
```

- Meters vary in data types they measure
 - `updateSums` defined differently for each kind

Subclasses of MeterAbstract

- Meter
 - Single value is outcome of measurement
 - Measurement is defined in `currentValue` method

```
public void updateSums() {accumulator.add(currentValue());}  
  
public abstract double currentValue();
```

- MeterFunction
 - 1-D array of values (points on a function) is outcome of measurement

```
public void updateSums() {  
    double[] values = currentValue();  
    for(int i=0; i<nPoints; i++) accumulator[i].add(values[i]); //accumulator for each value  
}  
public abstract double[] currentValue();
```

- MeterTensor
 - Defined similarly

Accumulator

- MeterAbstract.Accumulator (should make as top-level class)
- Evaluates statistics on data
 - Averages, confidence limits
 - Histograms, history

```
public void add(double value) {
    mostRecent = value; //hold to access most-recent without recalculation
    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);
}
```

Accessing Statistics

- Available from the Meter

```
public double average() {
    return (function==null) ? accumulator.average() : function.f(accumulator.average());
}

public double variance() {return accumulator.variance();}

public double error() {
    if(function == null) return accumulator.error();
    else {//have not carefully considered if this is correct
        return Math.abs(function.dfdx(accumulator.average()))*accumulator.error();
    }
}

public double mostRecent() {
    return (function==null) ?
        accumulator.mostRecent() : function.f(accumulator.mostRecent());}

public double mostRecentBlock() {
    return (function==null) ?
        accumulator.mostRecentBlock() : function.f(accumulator.mostRecentBlock());}

public Histogram getHistogram() {return accumulator.histogram();}
```

- Function can be used to modify value...

Function (An aside)

- Interface for a function (transforming a double to a double)

```
package etomica.utility;

public interface Function {
    public double f(double x);
    public double inverse(double f);
    public double dfdx(double x);

    // The function f(x) = 1/x
    public static class Reciprocal implements Function {

        public double f(double x){return 1.0/x;}
        public double dfdx(double x){return -1.0/(x*x);}
        public double inverse(double x){return 1.0/x;}
    }

    // The function f(x) = a*x + b
    public static class Linear implements Function {
        private final double a, b, ra;
        public Linear(double slope, double intercept) {
            this.a = slope;
            this.b = intercept;
            ra = 1.0/a;
        }
        public double f(double x) {return a*x + b;}
        public double inverse(double f) {return ra*(f-b);}
        public double dfdx(double x) {return a;}
    }

    //etc.
}
```



Accessing Any Statistic

- Sometimes need to specify desired statistic at run time
- `value(MeterAbstract.DataType type)` method permits this

- Inside
Meter..

```
public double value(MeterAbstract.ValueType type) {
    if(type==MeterAbstract.AVERAGE || type == null) return average();
    else if(type==MeterAbstract.MOST_RECENT) return mostRecent();
    else if(type==MeterAbstract.CURRENT) return currentValue();
    else if(type==MeterAbstract.MOST_RECENT_BLOCK) return mostRecentBlock();
    else if(type==MeterAbstract.ERROR) return error();
    else if(type==MeterAbstract.VARIANCE) return variance();
    else return Double.NaN;
}
```

- Method often applied by Display objects
 - Inside DisplayBox...

```
public void doUpdate() {
    if(source == null) return;
    value.setText(format(unit.fromSim(source.value(whichValue)),precision));
}
```

- Used particularly by Etomica GUI...(demo)



Typed (Enumerated) Constants 1.

- Some methods are meant to accept only a limited set of values for their arguments
 - VERTICAL/HORIZONTAL
 - NORTH/EAST/SOUTH/WEST
 - CURRENT, AVERAGE, ERROR, VARIANCE, MOST_RECENT
- One strategy is to key each value to a static integer constant
 - `public static final HORIZONTAL = 0; etc.`
 - `public void setOrientation(int k) {...}`
- Disadvantages
 - Method will accept any integer
 - No way to access full set of acceptable values at runtime
- Alternative approach is provided by Typed Constants
 - Define a type for each set of values
 - Create unique instances of only acceptable values
 - Key actions to equality with unique instances
 - Disadvantage: won't work in `case` statement

Typed (Enumerated) Constants 2.

- Inside `etomica.Constants...`

```
public static abstract class TypedConstant implements java.io.Serializable {
    private final String label;
    protected TypedConstant(String s) {label = s;} //constructor accessible only to subclasses
    public String toString() {return label;}
    public abstract TypedConstant[] choices();
}
/**
 * Typed constant for specifying HORIZONTAL/VERTICAL alignment.
 */
public static class Alignment extends TypedConstant {
    private Alignment(String label) {super(label);} //cannot instantiate externally
    public static final Alignment[] CHOICES = new Alignment[] {
        new Alignment("Horizontal"), //these are the only instances that will ever be made
        new Alignment("Vertical")};
    public final TypedConstant[] choices() {return CHOICES;}
}
public static final Alignment HORIZONTAL = Alignment.CHOICES[0];
public static final Alignment VERTICAL = Alignment.CHOICES[1];
```

- Access
 - `Constants.HORIZONTAL`
 - `Constants.VERTICAL`
 - `Constants.Alignment.CHOICES` or `instance.choices()`

DatumSource/DataSource Interfaces

- Data might be displayed from sources other than Meter
 - E.g., Controller that integrates over a range of conditions
 - Interface lets displays operate with other sources

```
public interface DataSource {
    public double[] values(ValueType type);
    //Returns a label used to describe the data when presented
    public String getLabel();
    //Returns the physical dimensions (e.g., length) of the data
    public etomica.units.Dimension getDimension();
    //Type class used to indicate to the data source which data is requested
    public static abstract class ValueType extends Constants.TypedConstant {
        protected ValueType(String label) {super(label);}
    }
    // Interface for a data source that has associated "x" values
    public interface X extends DataSource {
        public double[] xValues();
        public String getXLabel();
        public etomica.units.Dimension getXDimension();
    }
    // Indicates an object that uses a DataSource. Useful mainly to the Mediator
    public interface User {
        public void setDataSource(DataSource source);
        public DataSource getDataSource();
    }
    public interface MultiUser {...
    public interface Wrapper {...
} //end of DataSource
```

- 12
- DatumSource defined similarly to yield a single value

History and Histogram

- Accumulator can provide other information
 - Histogram of values passed to it via the `add` method
 - History of those values
 - Development needed here to make History treat long-period data in different ways
 - Cycle back to beginning (current functionality)
 - Coarse-grain
 - Expand window with addition of new data
- Functionality initiated only if directed via
 - `setHistogramming(true)`; or
 - `setHistorying(true)`
- History/Histogram obtained from meter (get methods)
 - Objects obtained this way implement `DataSource` interface

MeterGroup

- Some properties are best calculated all together, but a function isn't appropriate
 - E.g., species mole fractions
- MeterGroup acts like a set of independent meters
 - `public Meter[] allMeters()` method gives array of pseudo-meters that each act as stand-alone meters would
 - Internally, calculations for each pseudo-meter are performed together
- For example, see `MeterDimerFraction` class defined as part of the `KineticsModule` simulation
- `MeterMultiFunction` (sort of) acts similarly for a group of `MeterFunctions`

Miscellany

- **MeterCollisional**
 - Interface that defines method to be called every time IntegratorHard processes a collision
 - Meter acts on data generated by these calls
- **MeterProfile**
 - Wraps a meter that implements Meter.Atomic, which guarantees that property can be measured separately for each atom
 - Profile keeps track of values as a function of linear position in the simulation volume
- **MeterDatumSourceWrapper**
 - Wraps a non-Meter DatumSource to add Meter-like functionality, such as historying or histogramming