CE 412 Molecular Modeling

Simulation Lecture 3

Object-Oriented Programming and Etomica

David A. Kofke
Department of Chemical Engineering
SUNY Buffalo
kofke@eng.buffalo.edu

Object-Oriented Programming

- Programming accomplished through the actions and interactions of objects
  - *everything is an object*
- Forces abstract thinking about the structure and activities of a program
- Promotes re-use of code and extension to new applications
- Good design is difficult to develop
  - *requires thorough understanding of application*
  - *conversely, its use facilitates a better understanding of application*

    presents a good vehicle for teaching
- It's fun!

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 can perform on it*
  - *every object is an "instance of a class"*

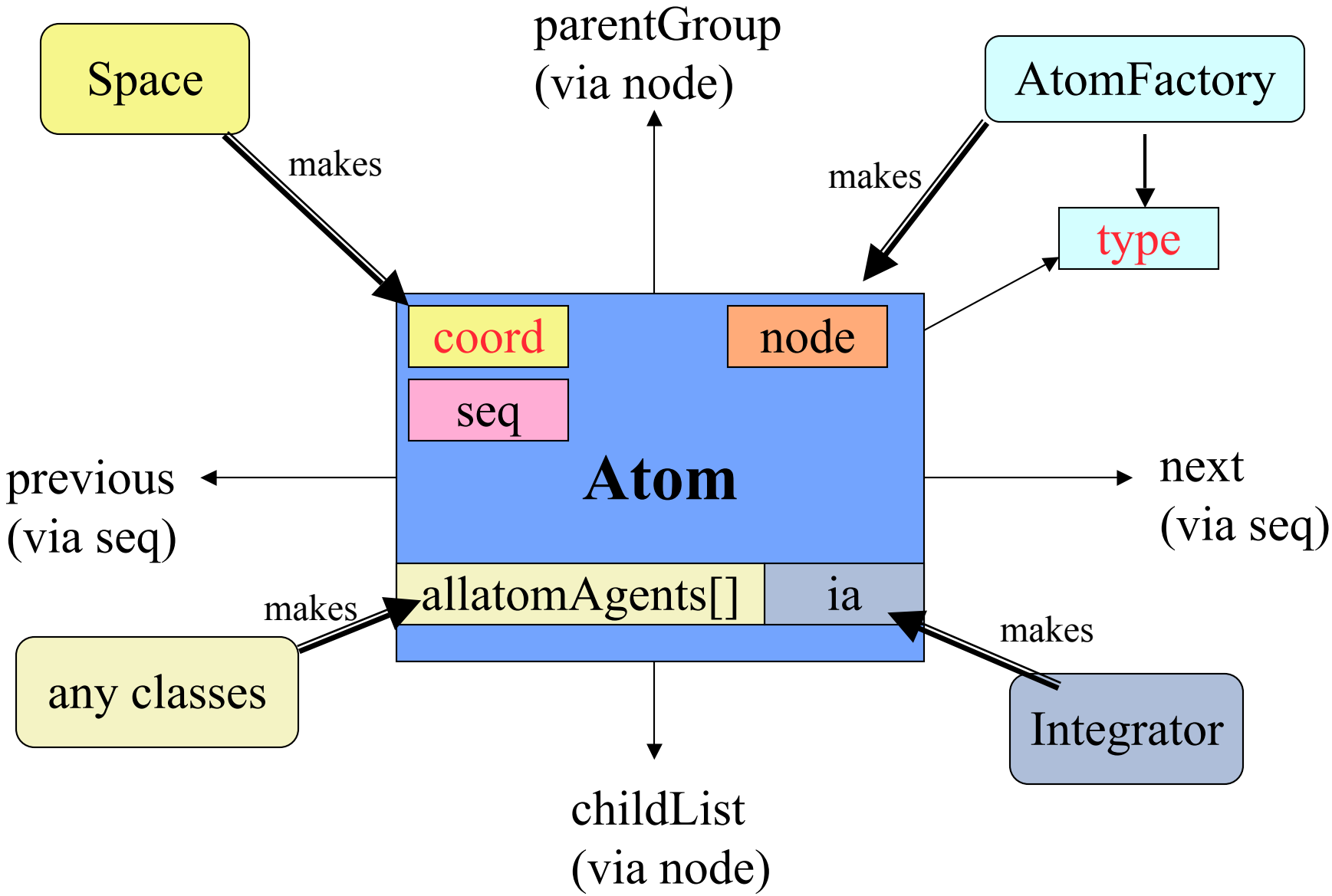
```
a1 = new Atom();
```

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*
- A class has an interface
  - *what the object presents to enable its manipulation*
  - *implementation (how it accomplishes its operations) can be hidden*
  - *object is viewed in terms of its "actions" and not its "thoughts"*
- 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*

Detailed Look: Atom

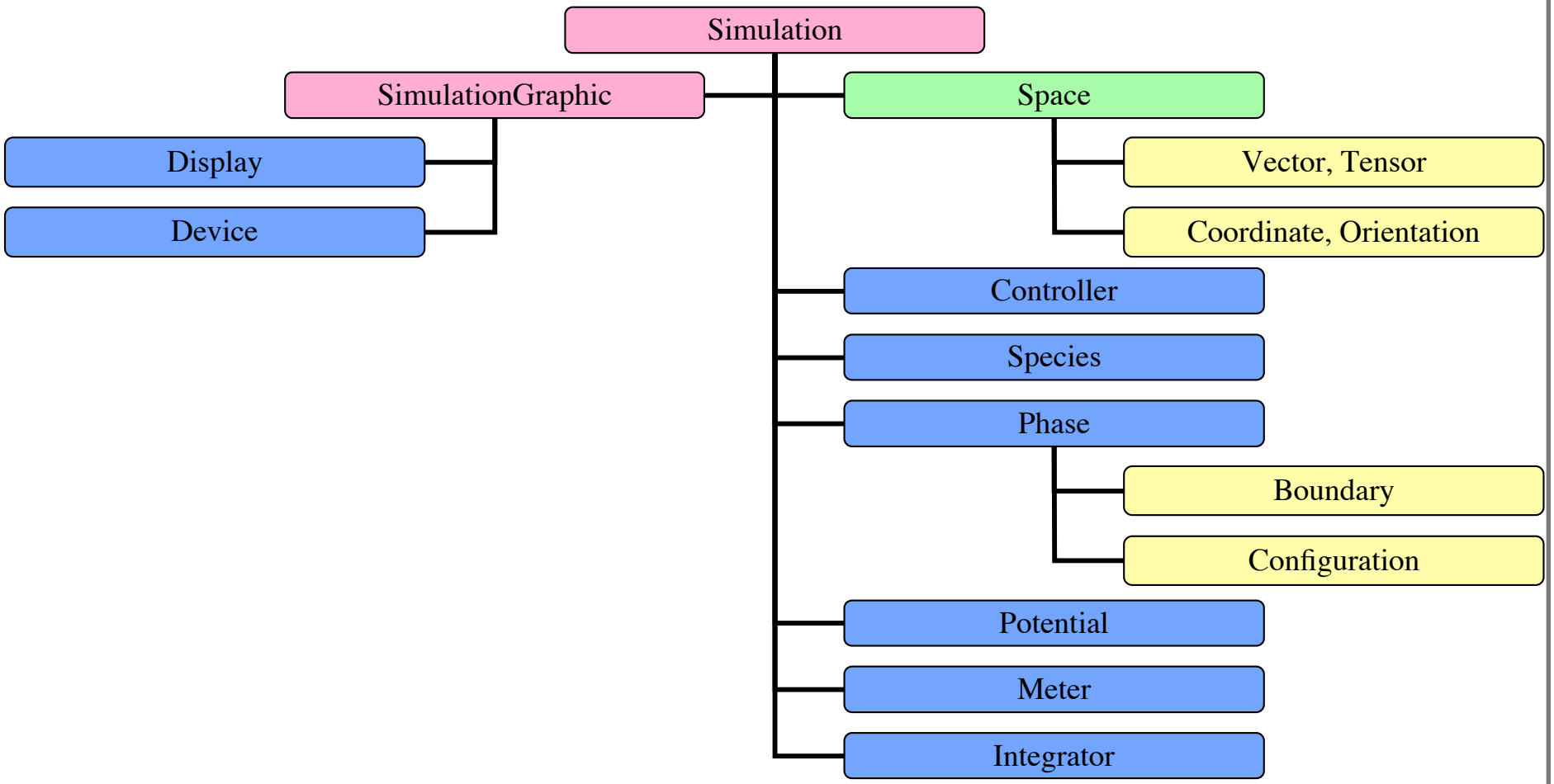


○ Click here for the complete API specification

Etomica

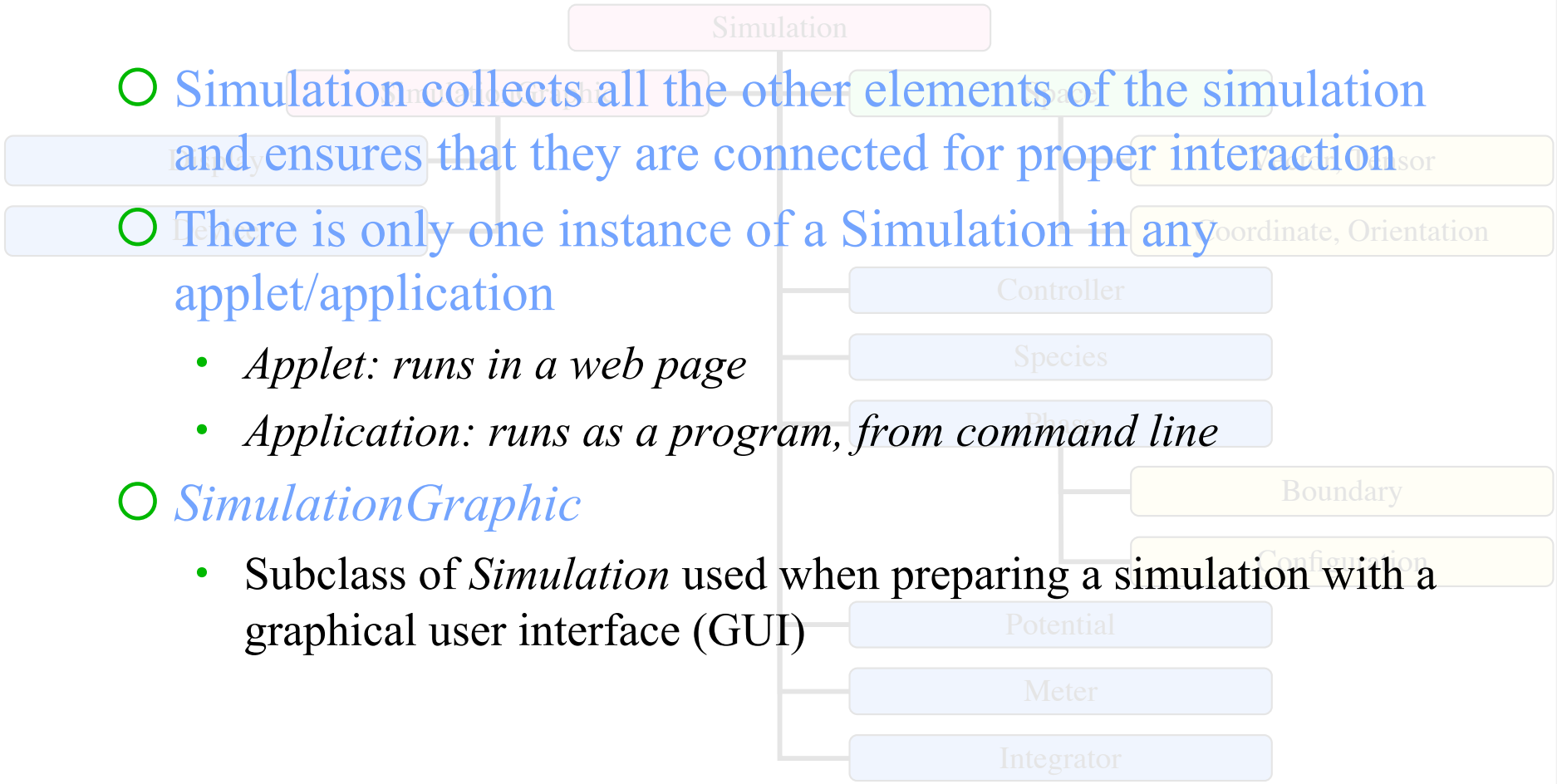
- Application Programming Interface (API)
  - *Library of components used to assemble a simulation*
  - *Can be used independent of development environment*
    - Invoked in code programmed using Emacs (for example)
- GUI-based development environment
  - *Simulation is constructed by piecing together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

Etomica API


Simulation
SimulationGraphic
Space
Display
Device
Vector, Tensor
Coordinate, Orientation
Controller
Species
Phase
Boundary
Configuration
Potential
Meter
Integrator

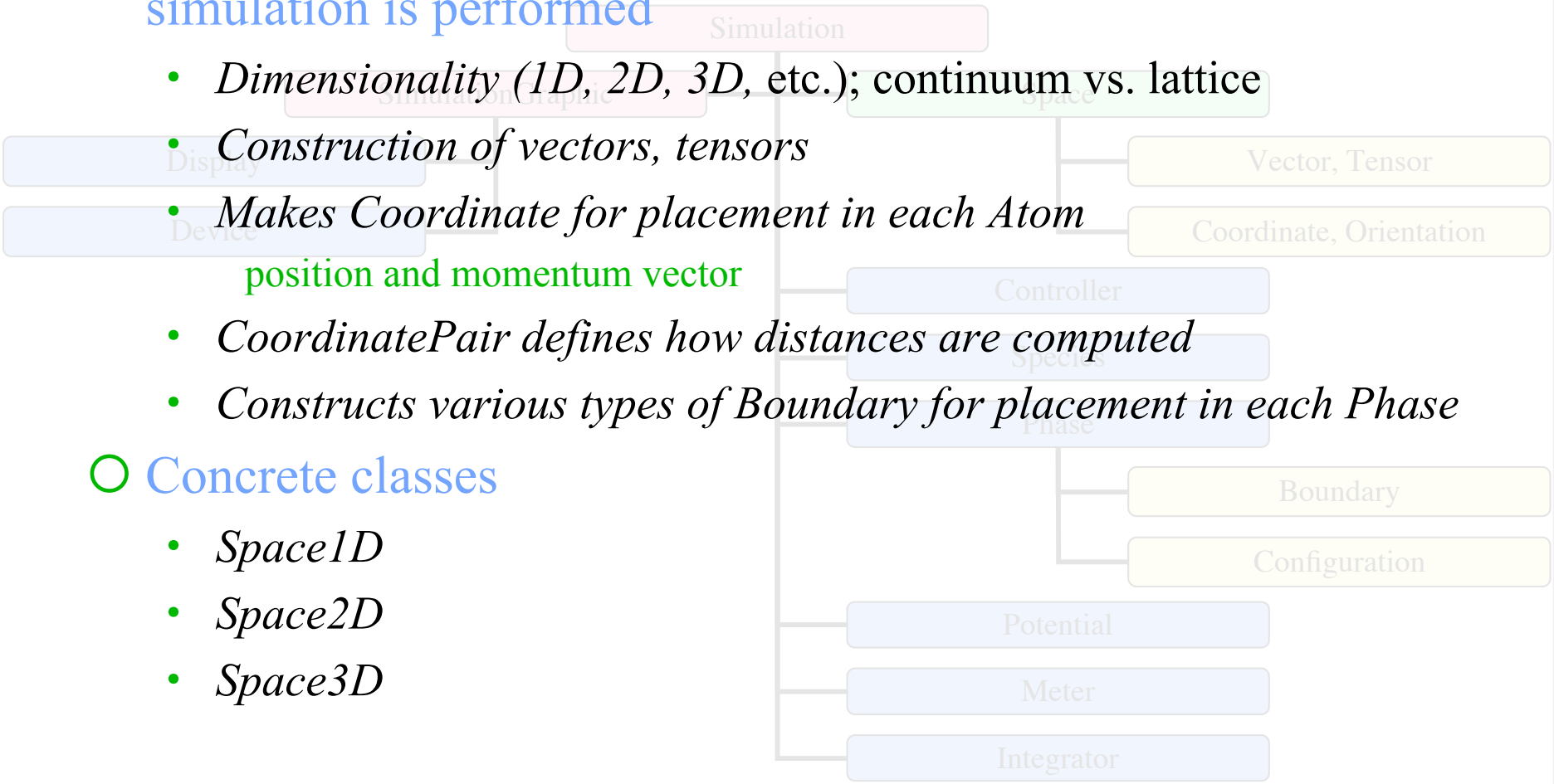
Simulation Elements: Simulation

- Simulation collects all the other elements of the simulation and ensures that they are connected for proper interaction
- There is only one instance of a Simulation in any applet/application
  - *Applet: runs in a web page*
  - *Application: runs as a program, from command line*
- *SimulationGraphic*
  - Subclass of *Simulation* used when preparing a simulation with a graphical user interface (GUI)



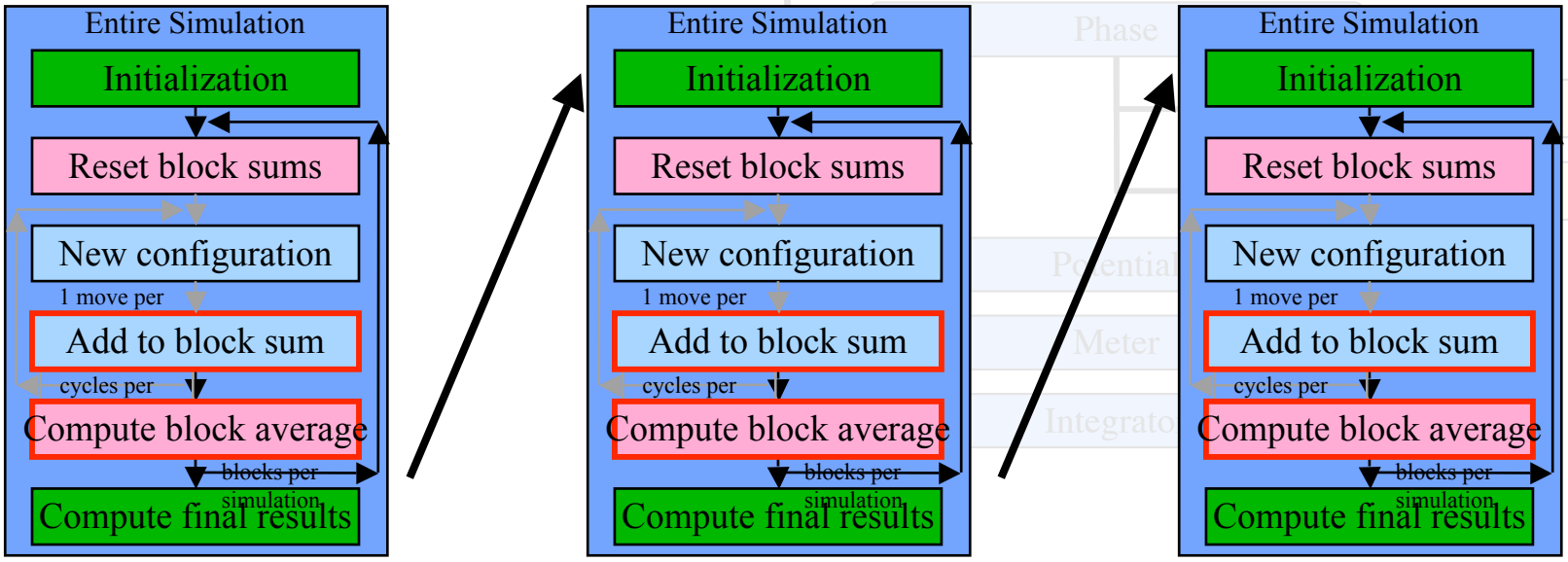
Simulation Elements: Space

- Space defines properties of the physical space in which the simulation is performed
  - *Dimensionality (1D, 2D, 3D,* etc.); continuum vs. lattice
  - *Construction of vectors, tensors*
  - *Makes Coordinate for placement in each Atom*
    position and momentum vector
  - *CoordinatePair defines how distances are computed*
  - *Constructs various types of Boundary for placement in each Phase*
- Concrete classes
  - *Space1D*
  - *Space2D*
  - *Space3D*



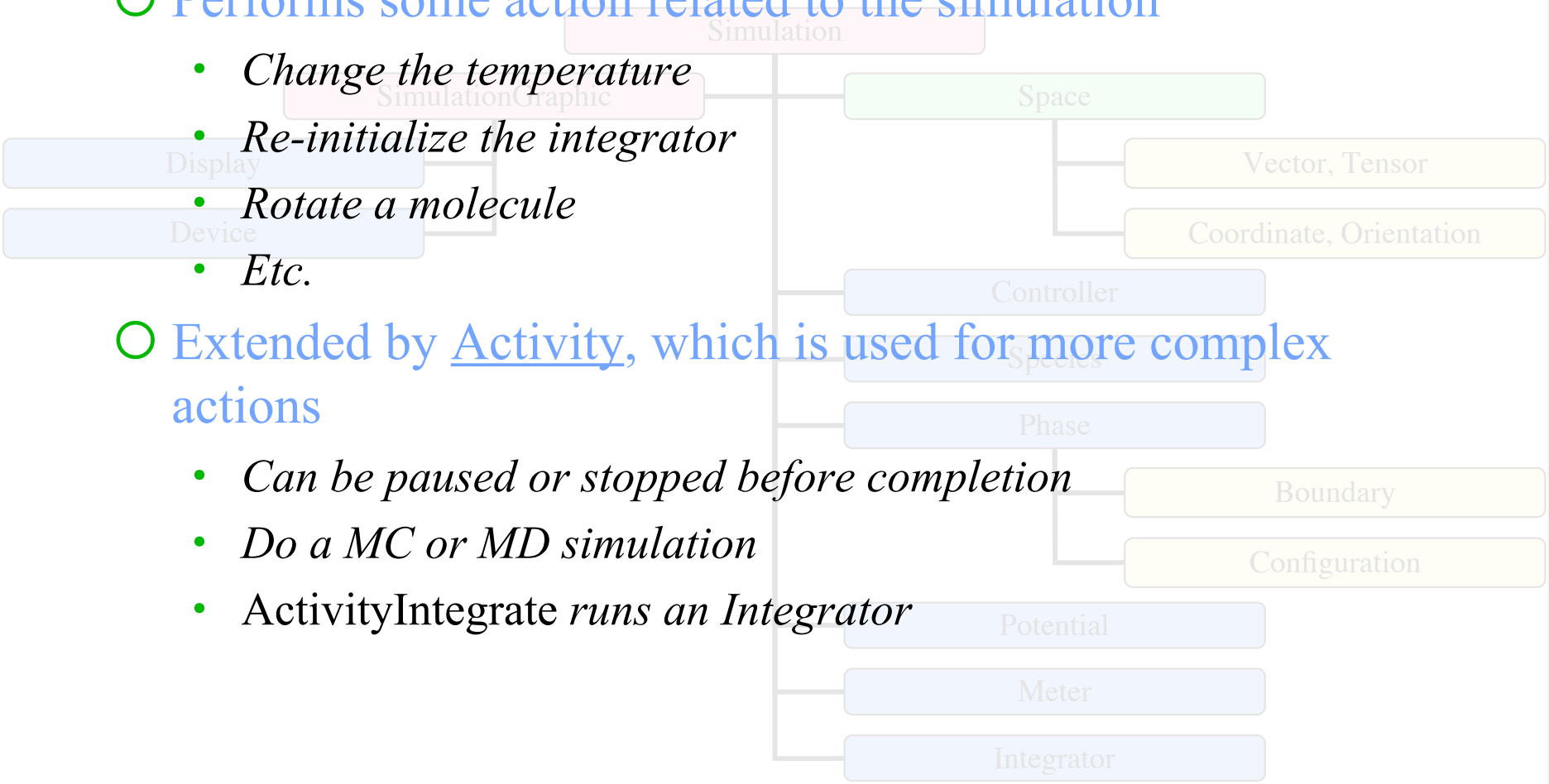
Simulation Elements: Controller

- Governs general plan of action for simulation
  - *For example:*
    - run forever, as controlled by an interactive Stop/Resume button
    - run a fixed number of relaxation/production cycles, then quit
    - run over a series of state conditions for a fixed duration at each
- Oversees activities of Integrator
  - *Makes connections between Integrator and other elements to ensure proper functioning*
  - *Turns Integrator on and off according to plan of action*



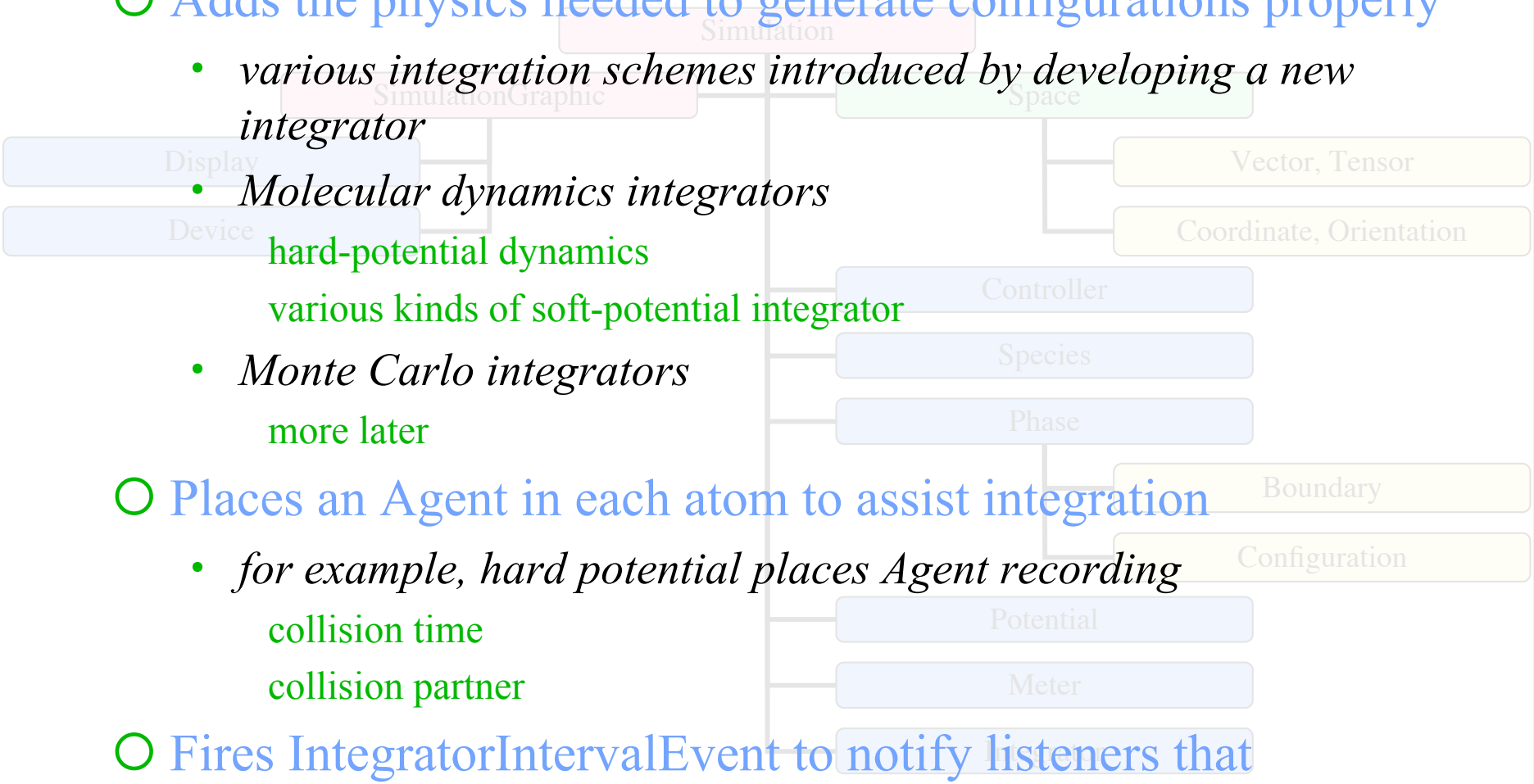
Simulation Elements: Action

- Performs some action related to the simulation
  - *Change the temperature*
  - *Re-initialize the integrator*
  - *Rotate a molecule*
  - *Etc.*
- Extended by Activity, which is used for more complex actions
  - *Can be paused or stopped before completion*
  - *Do a MC or MD simulation*
  - ActivityIntegrate *runs an Integrator*



Simulation Elements: Integrator

- Adds the physics needed to generate configurations properly
  - *various integration schemes introduced by developing a new integrator*
    - *Molecular dynamics integrators*
      - hard-potential dynamics
      - various kinds of soft-potential integrator
    - *Monte Carlo integrators*
      - more later
- Places an Agent in each atom to assist integration
  - *for example, hard potential places Agent recording*
    - collision time
    - collision partner
- Fires IntegratorIntervalEvent to notify listeners that simulation has proceeded one step



Simulation Elements: Integrator

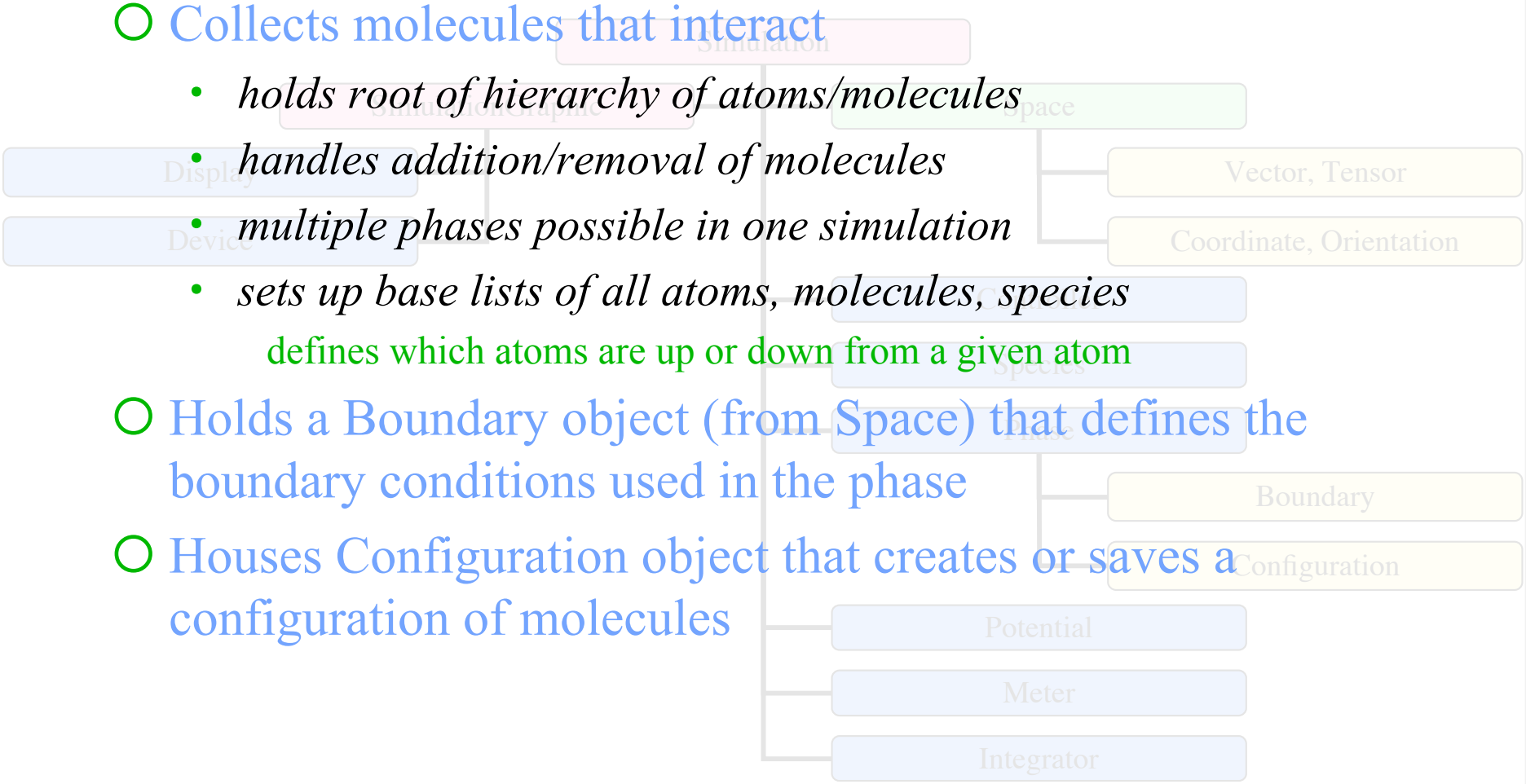
- *run* method for top-level Integrator class

```
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
```

Simulation Elements: Phase

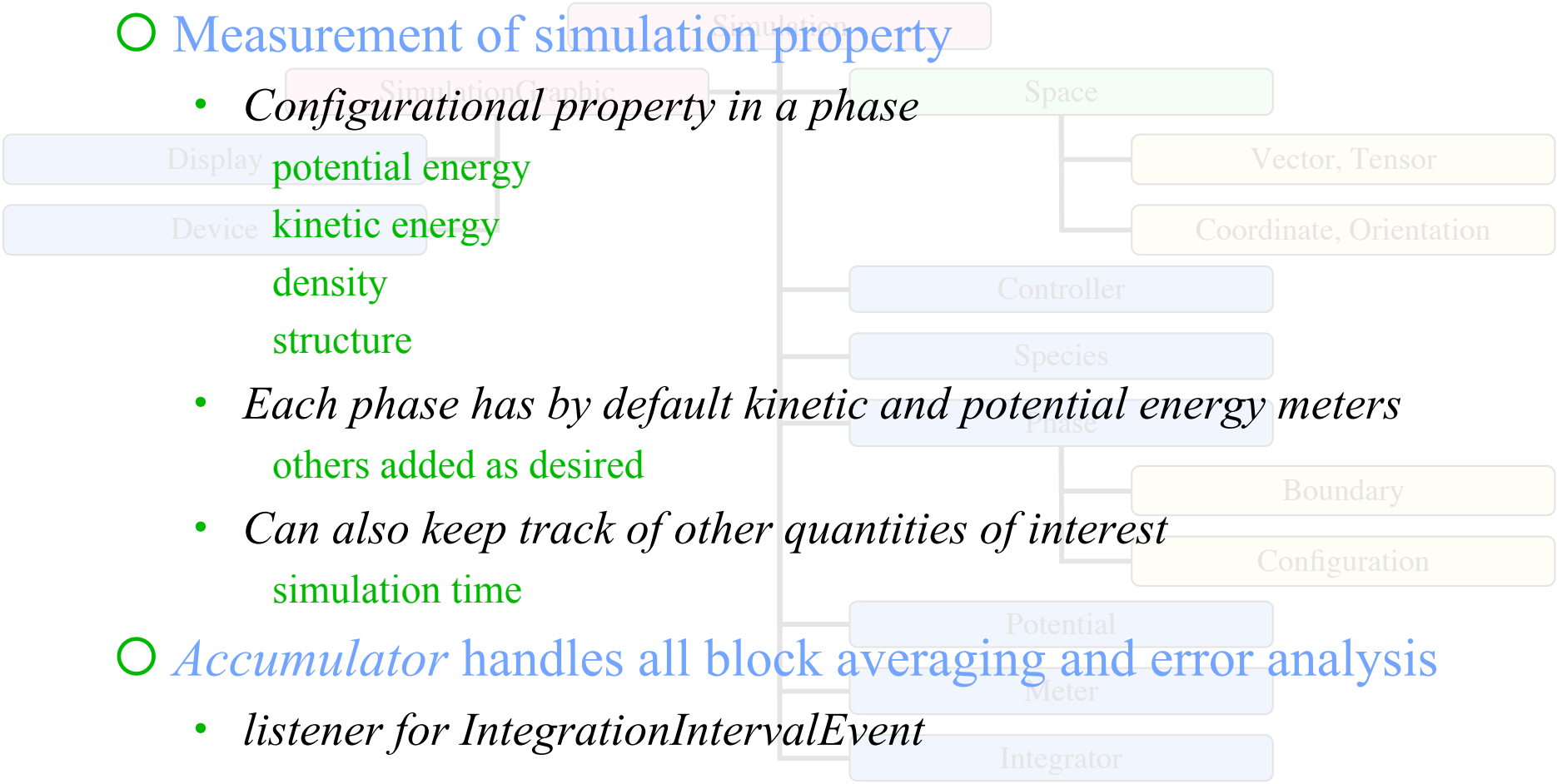
- Collects molecules that interact
  - *holds root of hierarchy of atoms/molecules*
  - *handles addition/removal of molecules*
  - *multiple phases possible in one simulation*
  - *sets up base lists of all atoms, molecules, species*

    defines which atoms are up or down from a given atom
- Holds a Boundary object (from Space) that defines the boundary conditions used in the phase
- Houses Configuration object that creates or saves a configuration of molecules



Simulation Elements: Meter

- Measurement of simulation property
  - *Configurational property in a phase*
    - potential energy
    - kinetic energy
    - density
    - structure
  - *Each phase has by default kinetic and potential energy meters*
    - others added as desired
  - *Can also keep track of other quantities of interest*
    - simulation time
- *Accumulator* handles all block averaging and error analysis
  - *listener for IntegrationIntervalEvent*

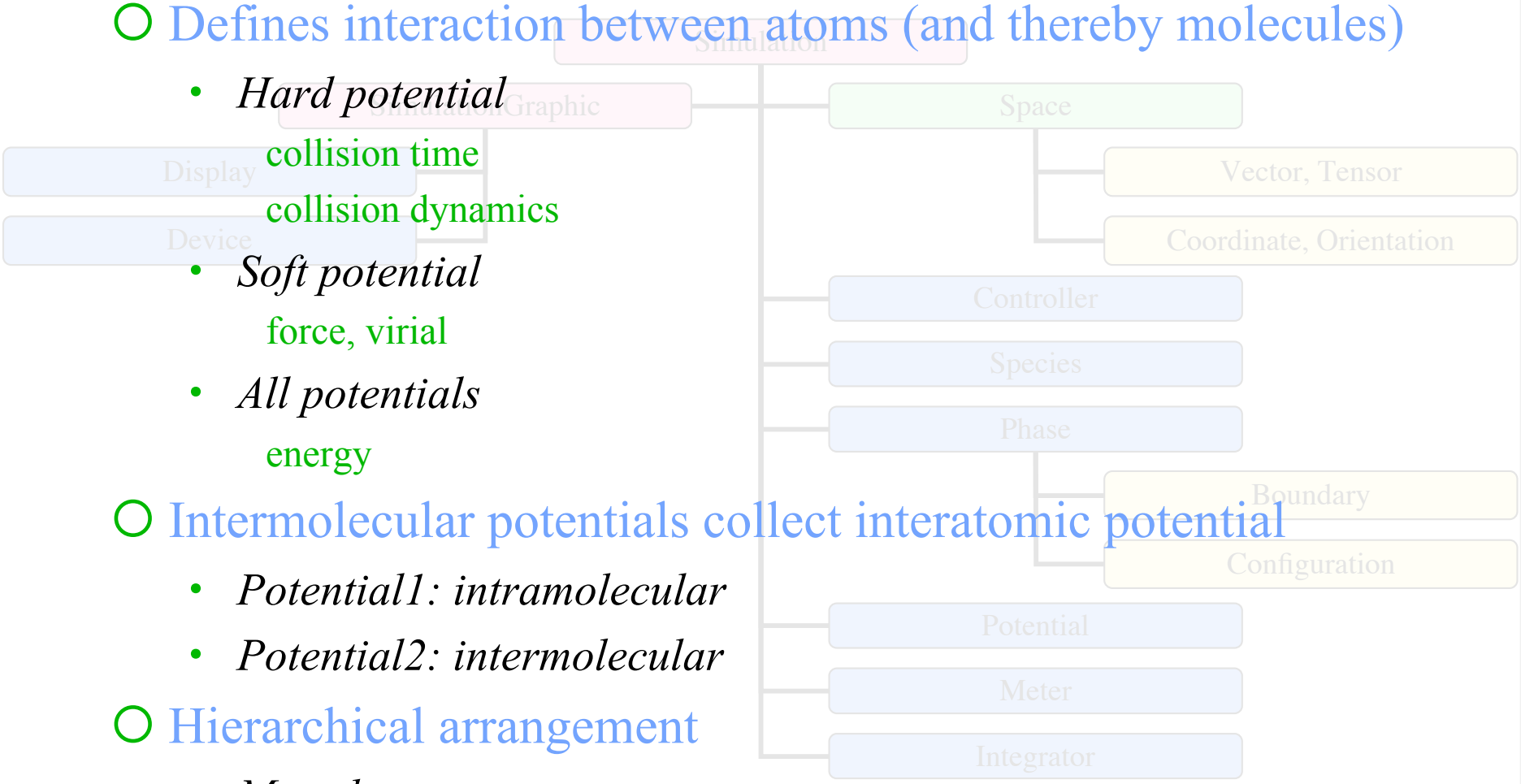


Simulation Elements: Species

- Holds AtomFactory that defines how a particular type of molecule is constructed
  - *how many atoms of which type*
  - *nominal arrangement of atoms in molecule*
- Places SpeciesAgent in each phase to manage molecules there
  - *root of atom hierarchy for species in phase*
  - *looping over molecules*
  - *addition and removal of atoms*
- Basis for defining interaction potential between molecules
  - *but does not by itself describe the pair potential*

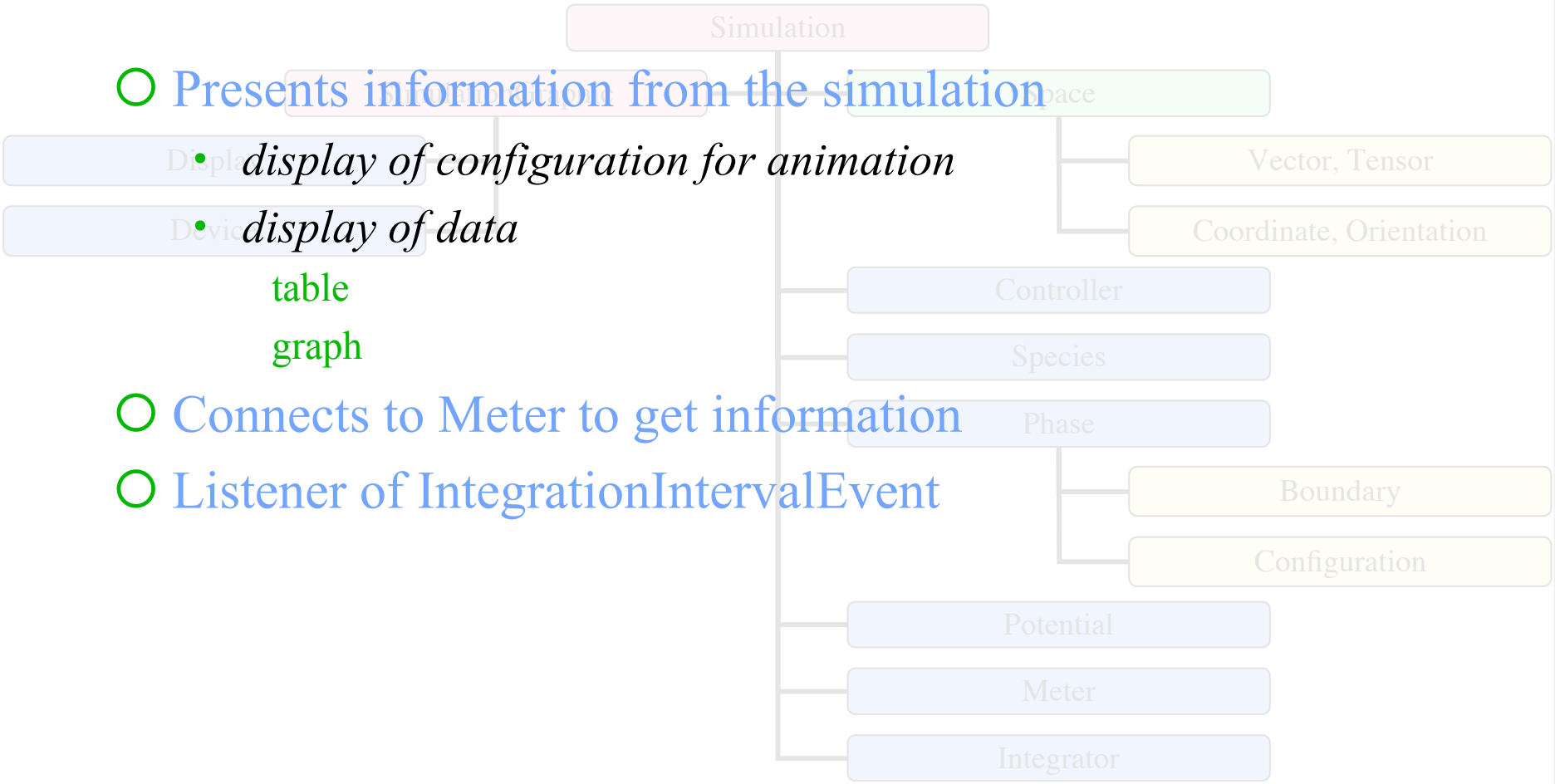
Simulation Elements: Potential

- Defines interaction between atoms (and thereby molecules)
  - *Hard potential*
    - collision time
    - collision dynamics
  - *Soft potential*
    - force, virial
  - *All potentials*
    - energy
- Intermolecular potentials collect interatomic potential
  - *Potential1: intramolecular*
  - *Potential2: intermolecular*
- Hierarchical arrangement
  - *More later*



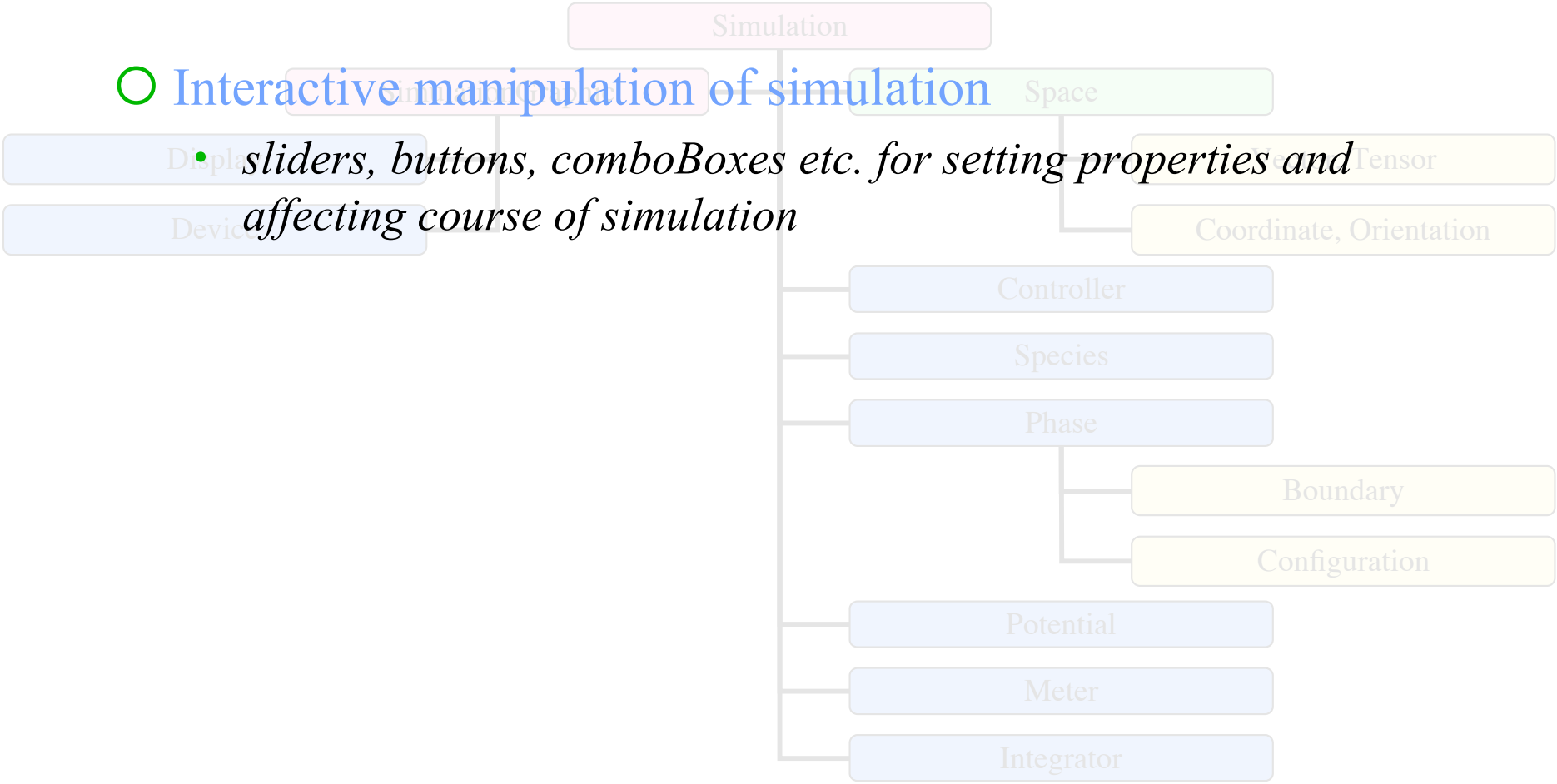
Simulation Elements: Display

- Presents information from the simulation
  - *display of configuration for animation*
  - *display of data*
    - table
    - graph
- Connects to Meter to get information
- Listener of IntegrationIntervalEvent



Simulation Elements: Device

- Interactive manipulation of simulation
  - *sliders, buttons, comboBoxes etc. for setting properties and affecting course of simulation*



Putting It Together

```
//import classes from library
import etomica.*;
import etomica.graphics.*;
import etomica.action.PhaseImposePbc;
import etomica.action.activity.ActivityIntegrate;
import etomica.graphics.SimulationGraphic;
import etomica.integrator.IntegratorHard;
import etomica.potential.P2HardSphere;
import etomica.space2d.Space2D;

public class MySimulation extends Simulation {
```

Continued…

Save this code in a file named *MySimulation..java*

Putting It Together

```
public class MySimulation extends Simulation {

    public ActivityIntegrate activityIntegrate;

//Constructor
    public MySimulation(Space2D space) {
        super(space);
         //Instantiate classes
        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});
//Tie elements together
        integrator.addIntervalListener(new PhaseImposePbc(phase));
        phase.speciesMaster.addSpecies(species);
        integrator.addPhase(phase);
    }//end of constructor
```

Save this code in a file named *MySimulation..java*

Continued…

Putting It Together

```
}//end of constructor

    /**
     * Demonstrates how this class is implemented.
     */
    public static void main(String[] args) {
        MySimulation sim = new MySimulation(new Space2D());
        SimulationGraphic graphic = new SimulationGraphic(sim);
        sim.activityIntegrate.setDoSleep(true);
        graphic.makeAndDisplayFrame();
    }//end of main
} //End of MySimulation class
```

Save this code in a file named *MySimulation..java*

Running It: Application

- If needed, obtain the java runtime for your platform from Sun
- Get the etomica class library archive, etomica.jar. Place it in the same directory as your source file
- Compile your source
  - javac –classpath etomica.jar MySimulation.java

    creates *MySimulation.class*
- Run as an application
  - java -cp etomica.jar;. MySimulation

Running It: Applet

- Add an Applet inner class to your source

```
 } //End of main
   public static class Applet extends javax.swing.JApplet {
     public void init() {
         MySimulation sim = new MySimulation(new Space2D());
         SimulationGraphic graphic = new SimulationGraphic(sim);
         getContentPane().add(graphic.panel());
     }
   }//end of Applet
} //End of MySimulation class
```

- Compile to create MySimulation.class, as for application
- Create an html file, named (for example) Applet.html

```
<HTML>
<HEAD> <TITLE>Applet HTML</TITLE> </HEAD>
<BODY bgcolor="#ffffff">
<APPLET CODE= "MySimulation$Applet.class" archive= "etomica.jar"
   WIDTH=640 HEIGHT=350></APPLET>
</BODY>
</HTML>
```

- Run applet, either
  - *Using appletviewer at command line:* appletviewer Applet.html
  - *Or using your web browser, pointed at* Applet.html *on your machine*
  - *Or deploy to a web server for the rest of the world to enjoy!*

Extend Your Application

- Examine the specification of the classes in the API.
- Consider some modifications
  - *Change the number of atoms*
  - *Add a meter for the temperature*
  - *Add a meter for the radial distribution function*
  - *Add a device to change the temperature*
  - *Add a device to change the density*
- Try them out!