

Design Patterns

Plan ahead for change

Traditional programming:

- One large seamless code
- Single purpose
- Reusable subroutines

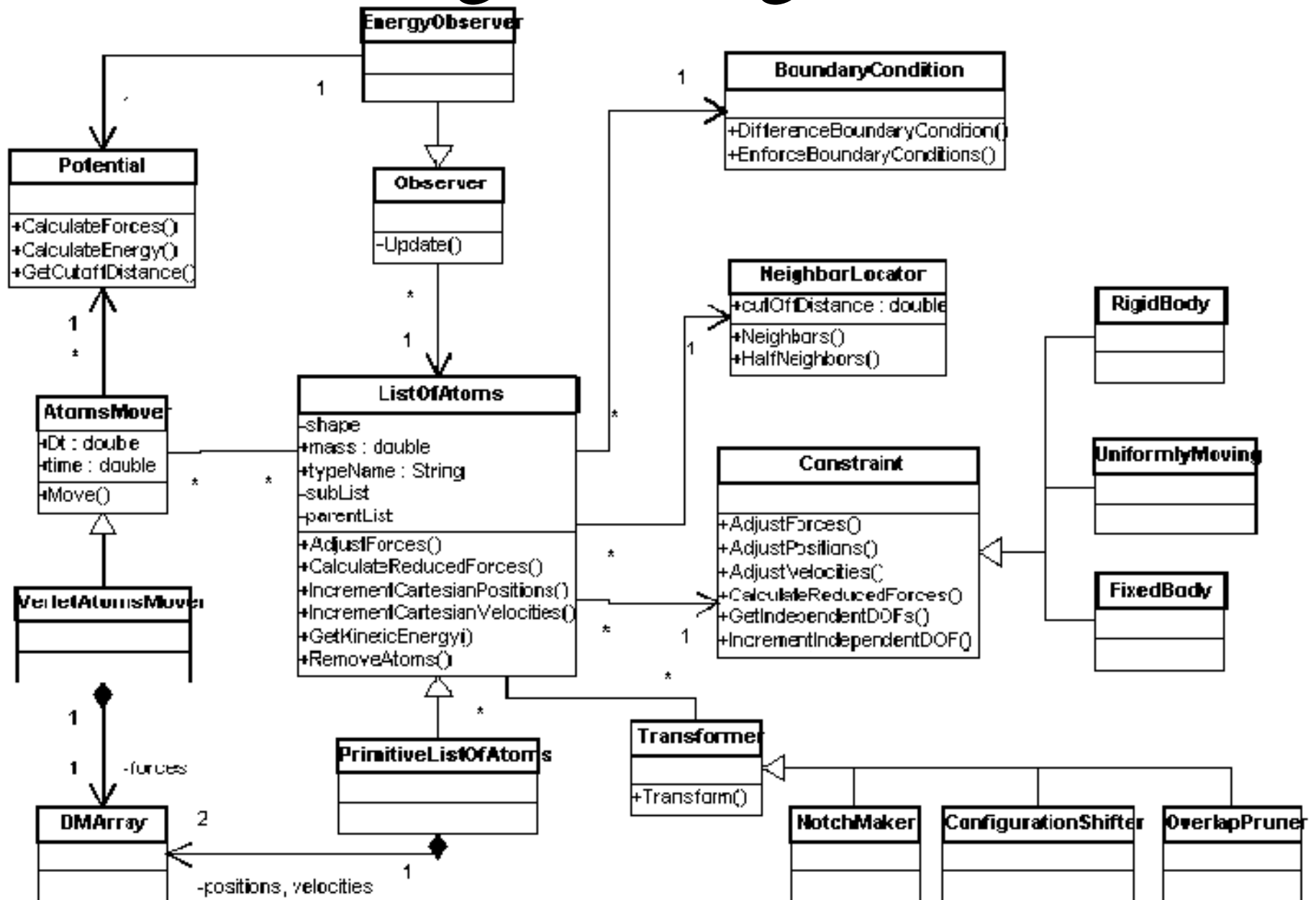
Object-oriented programming

- Define new kind of objects
 `g=Graph()`
- Attach data
 `g.nodes, g.neighbor_dict`
- Attach methods
 `g.AddEdge(), g.HasNode()`

Design pattern philosophy

- Many small objects work together
 ListOfAtoms
 Potential
 BoundaryConditions
 NeighborLocator
 Observer
 Mover
- Build many types of simulations from pieces
- Factor problem into components
- Refactor several times to get right

Class Diagram: Digital Material



Gravity Cluster built from pieces

```
gravityPotential = GravityPotential(g=g)
LennardJonesPotential = LennardJonesCutPotential()
potential = CompositePotential([gravityPotential, LennardJonesPotential])
boundaryConditions = ReflectiveBoundaryConditions(L)
neighborLocator = SimpleNeighborLocator(LennardJonesPotential.cutoff,
                                         boundaryConditions)
atoms = TriangularSphericalClusterListOfAtoms(
    R=R, center=[L/2., L/2.], temperature=T,
    radius=LennardJonesPotential.latticeSpacing/2.0)
displayObserver = VisualDisplayAtomsObserver(atoms,L)
energyObserver = DM.EnergyObserver(potential, neighborLocator,
                                   boundaryConditions)
observers = [displayObserver, energyObserver]
mover = RunVelocityVerlet;
sys = MDSystem(L, atoms, observers, neighborLocator, boundaryConditions,
              potential, mover)
```

DigitalMaterial

ListOfAtoms

NOT a list of atom objects: use fast array operations!

positions = numpy.array((nAtoms, dimension))

velocities = numpy.array((nAtoms, dimension))

mass, radius, color

KineticEnergy: return $0.5 * \text{sum}(\text{mass} * \text{velocity} * \text{velocity})$

Initializers (RandomListOfAtoms, TriangularSphericalClusterListOfAtoms)

Potentials (GravityPotential, LennardJonesPotential)

Provides Forces and PotentialEnergy

Movers (ThermalizingTransformer, RunVelocityVerlet, QuickMin)

Observers (EnergyObserver, VisualDisplayObserver,
VelocityTrajectoryObserver)

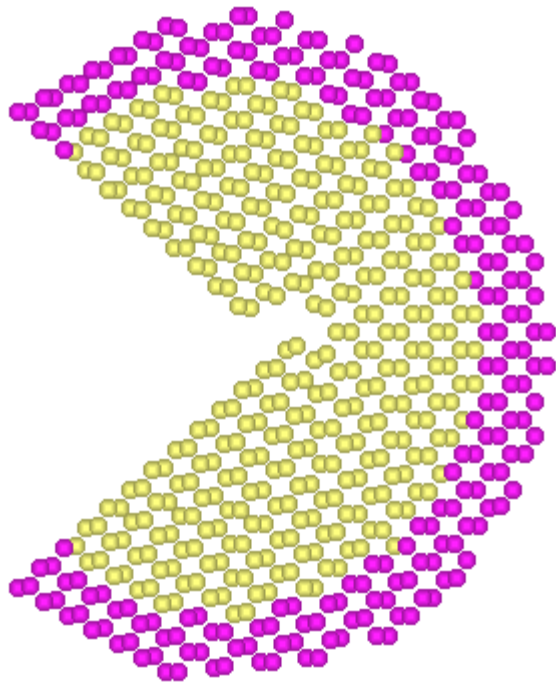
Subject-Observer pattern

Have Update; list of observers all called by Mover once per step;

Flexible! Removes complex code from inner loop of program!

BoundaryConditions (Periodic, Reflective, Free)

NeighborLocators and Constraints



NeighborLocator (CellNeighborLocator,
NeighborList, SimpleNeighborLocator,
NoNeighborLocator!)
used, e.g., by Potential
look only within cutoff
Gives Neighbors(), displacements,
HalfNeighbors()

Constraints (RigidBody, Fixed, ...)
Called by Mover
AdjustForces(), AdjustPositions(),
AdjustVelocities()

