Swarmfest ‘99 Tutorial

Session I: Introduction

Benedikt Stefansson

<benedikt at ucla.edu>

revised version 0.3

by YABUKI Taro for Java

First session

- Background
- History and overview of Swarm project
- Contributions of Swarm
- General structure of simulations in Swarm
- Quick and dirty introduction to Java
- Documentation, resources, user community

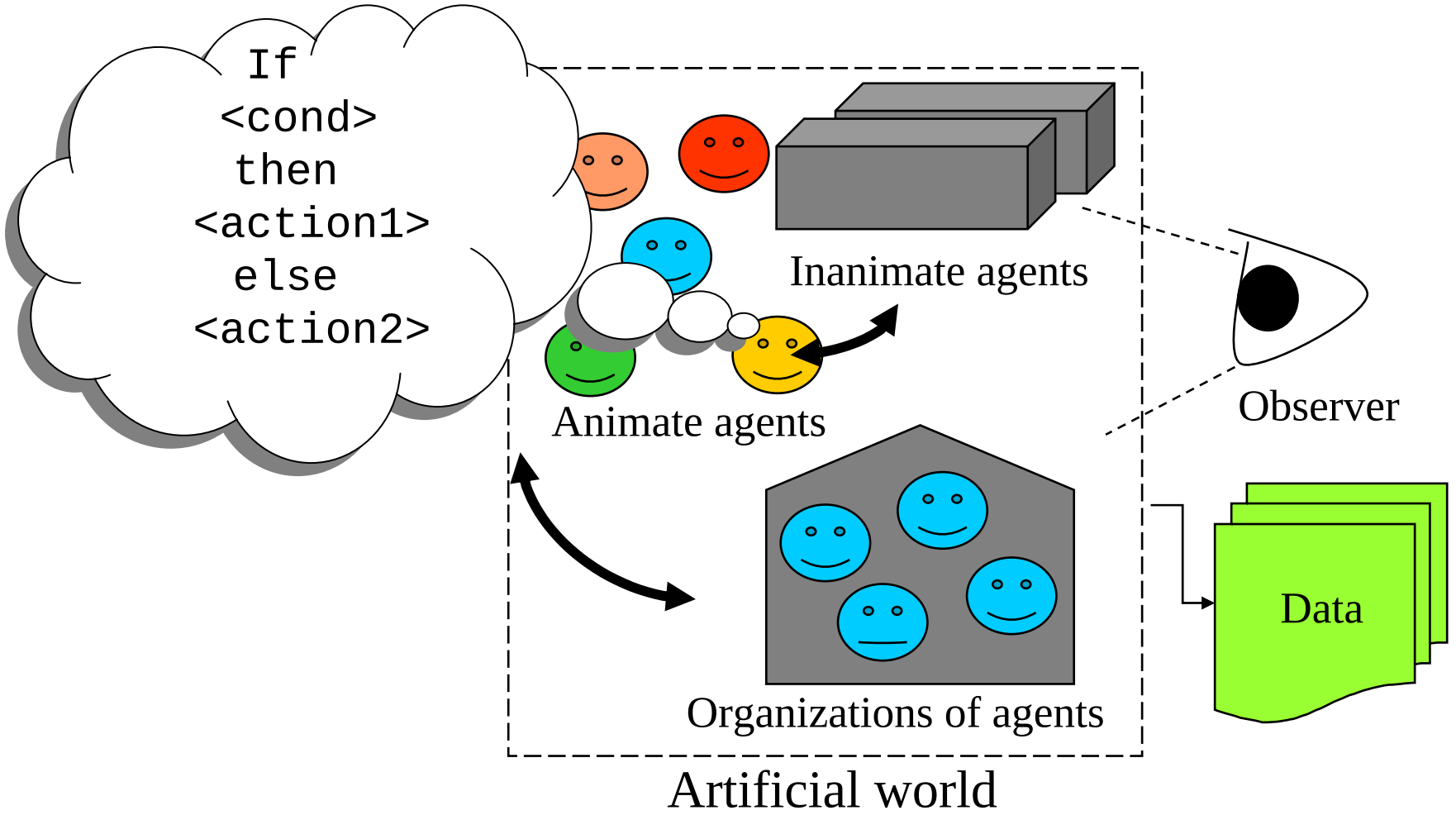
Focus for the day

- General structure of a Swarm simulation
- Main contributions of the Swarm project:
  - Event management: Swarms, Activity library
  - Information management: Probes
  - Graphical input and output: GUI objects
  - Memory management: Creation/destruction
  - Support for multiple languages

What is the point?

- To study complex nonlinear systems e.g.
  - An ecosystem
  - International conflict
  - Financial crisis
- Agent based models allow us to study
  - Spatial interaction
  - Adaptive, heterogeneous agents
  - Agents which face costs of information acquisition and processing
  - Nested subsystems - economy, markets, firms, plants, employees
  - etc.

Bottom up modeling



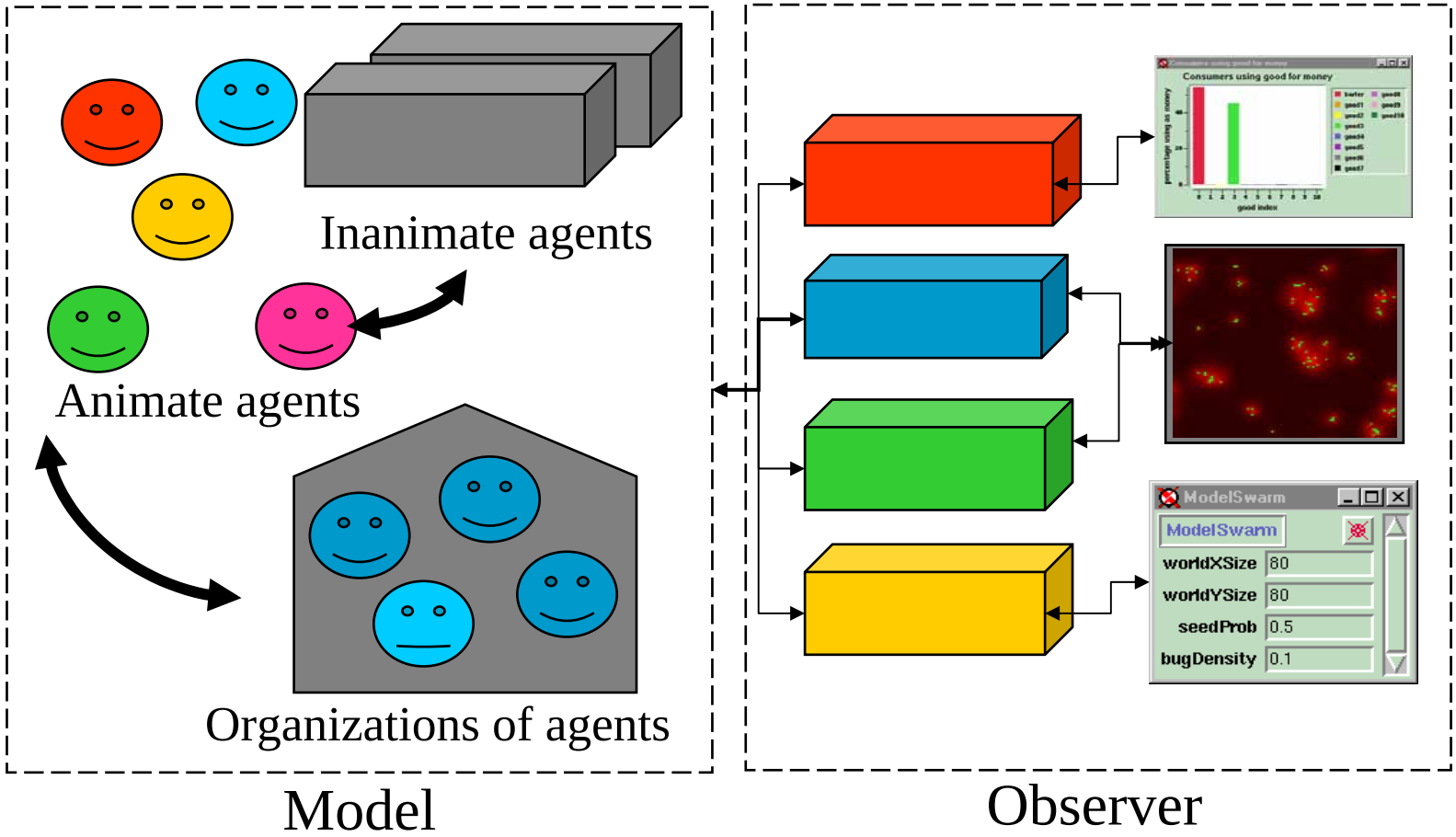
History of Swarm

- Launched in 1994 by Chris Langton at Santa Fe Institute in New Mexico (Contributors: Chris Langton, Roger Burkhart, Nelson Minar, Manor Askenazi, Glen Ropella, Sven Thommesen, Marcus Daniels, Alex Lancaster, Vladimir Jojic)
- Objectives
  - Create a shared simulation platform for ABM
  - Facilitate the development of models
- Released in beta in ‘95, 1.0 release in January ‘97, 1.1 with Win95/NT support April ‘98, 2.0 with Java April ‘99

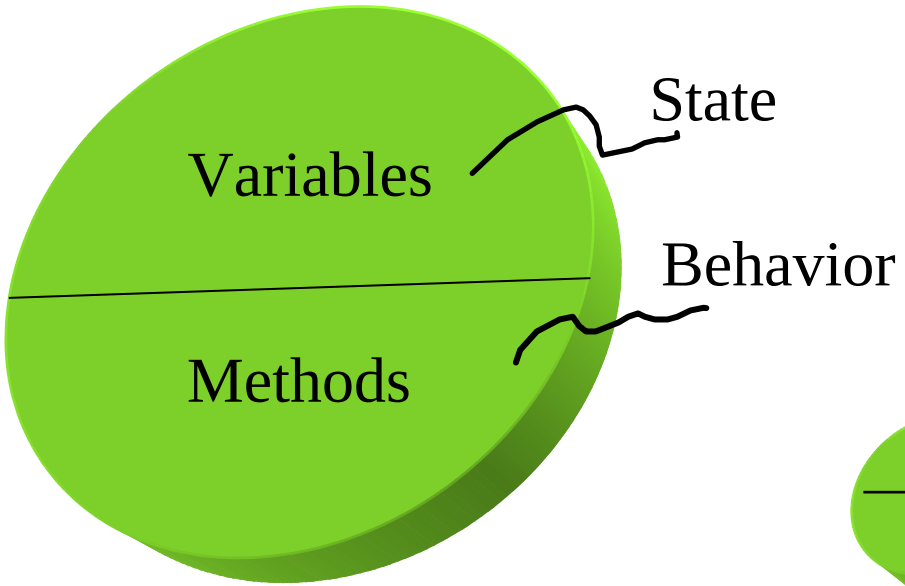
Basic facts

- Swarm is a collection of object oriented software libraries which provide support for simulation programming
- Users build simulations by incorporating Swarm library objects in their own programs
- Libraries are written in
  - Java : The main island of Indonesia
  - Tcl/Tk: Scripting language and GUI widgets (transparent to user)
- Available for Unix and Windows 95/98/NT.

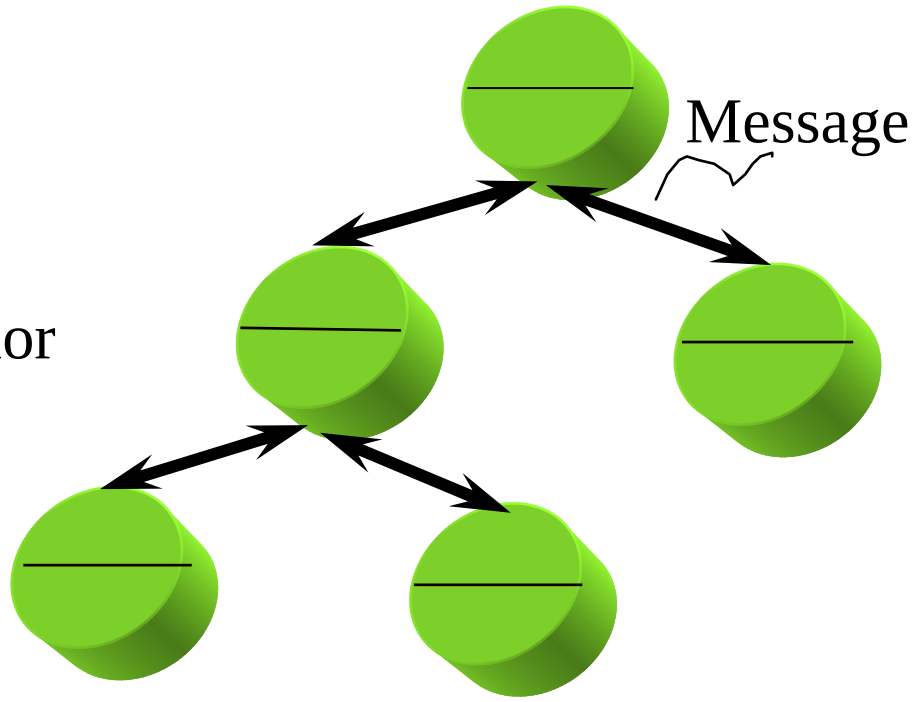
Swarm modeling



Object Oriented Programming

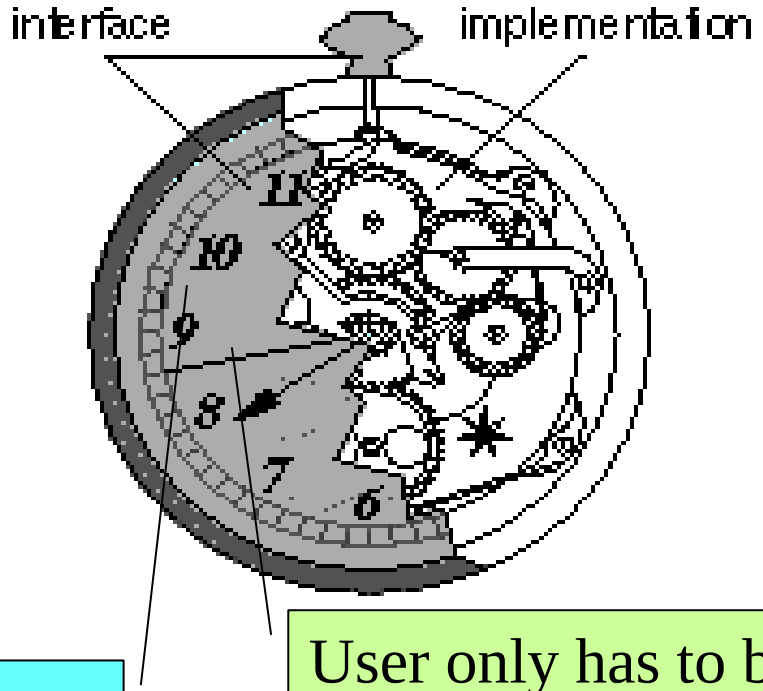


An object



A program

Interface vs. implementation



Objects hide their functions and data

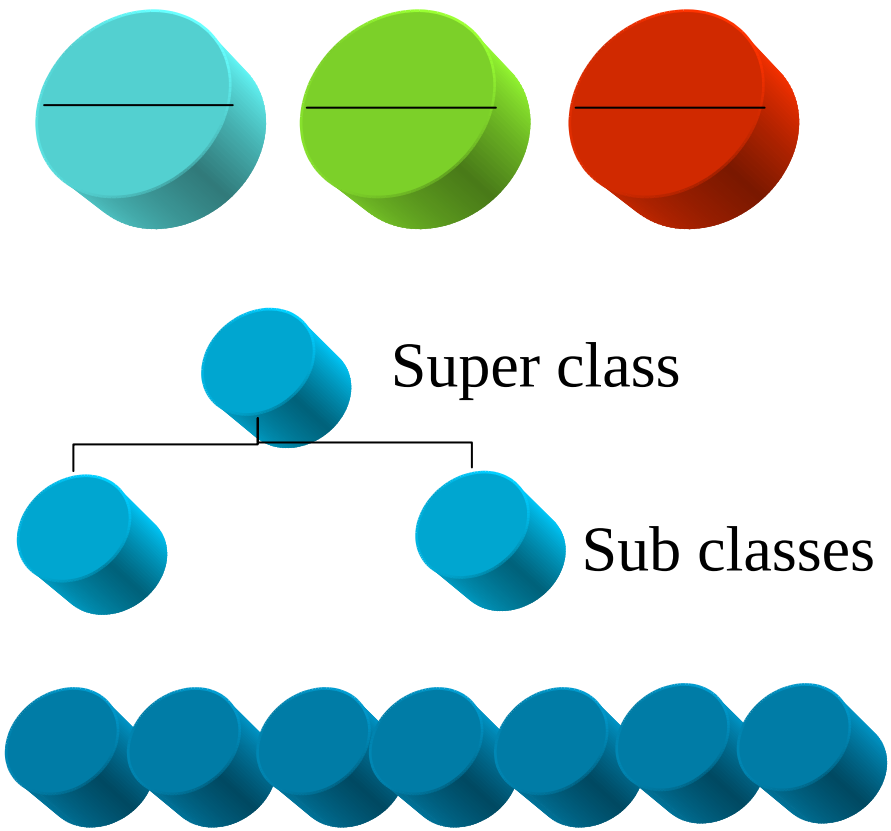
User only has to be familiar with the interface of an object, not it’s implementation

Some terminology

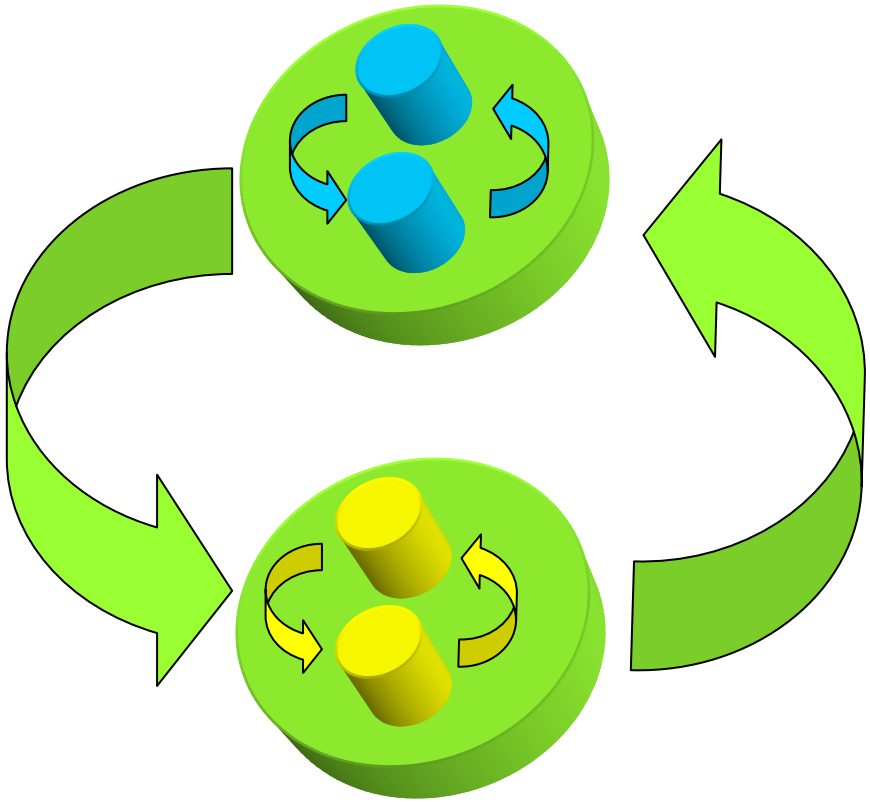
- Class
  - The definition of an object and the object factory
- Superclass
  - A class that an object inherits behavior and variables from (recursive)
- Subclass
  - A class that inherits behavior and variables from a superclass
- Instance
  - An object (instance of a class) that has been created and exists in memory
- Instance variable
  - A variable available to all functions in an object
- Method
  - A function. Can be called by sending message to an object of this class

The three principles of OOP

- Encapsulation
  - Objects hide their functions (**methods**) and data (**instance variables** and **method variables**)
- Inheritance
  - Each **subclass** inherits all variables of its **superclass**
- Polymorphism
  - Multiple instances of same class, sharing behavior but not state or memory



Discrete event simulation



- Simulation proceeds in discrete time steps
- Interaction between agents or procedures within simulation may have own event schedule

Simulation in procedural language

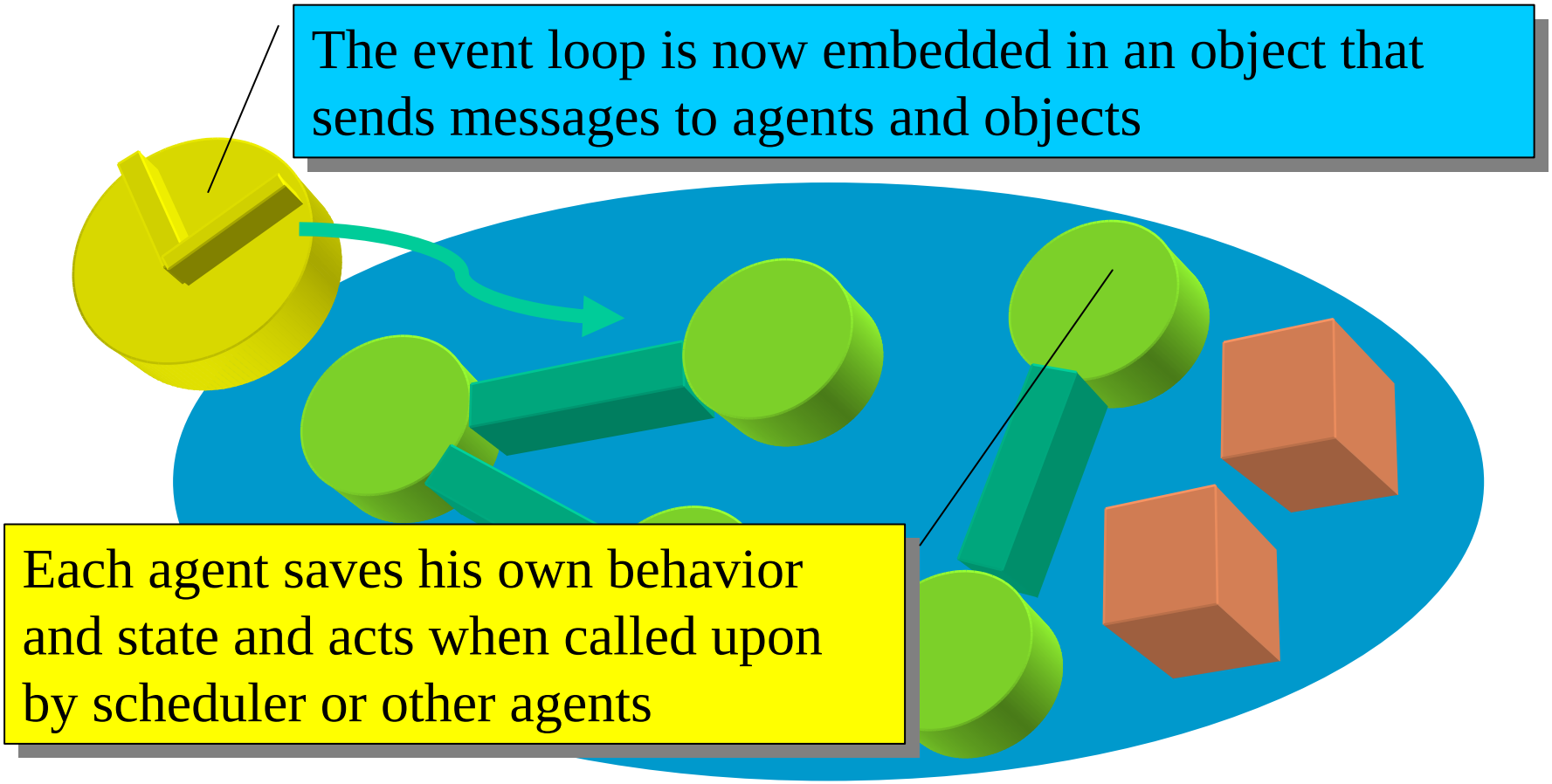
```
get parameters
    initialize
    for 1 to timesteps do:
        for 1 to num_agents do:
            agent-i-do-something
        end for
        show state
    end for
quit
```

Generally set's up data structures and support for output

Here must provide data structure to save agent's state and implement behavior

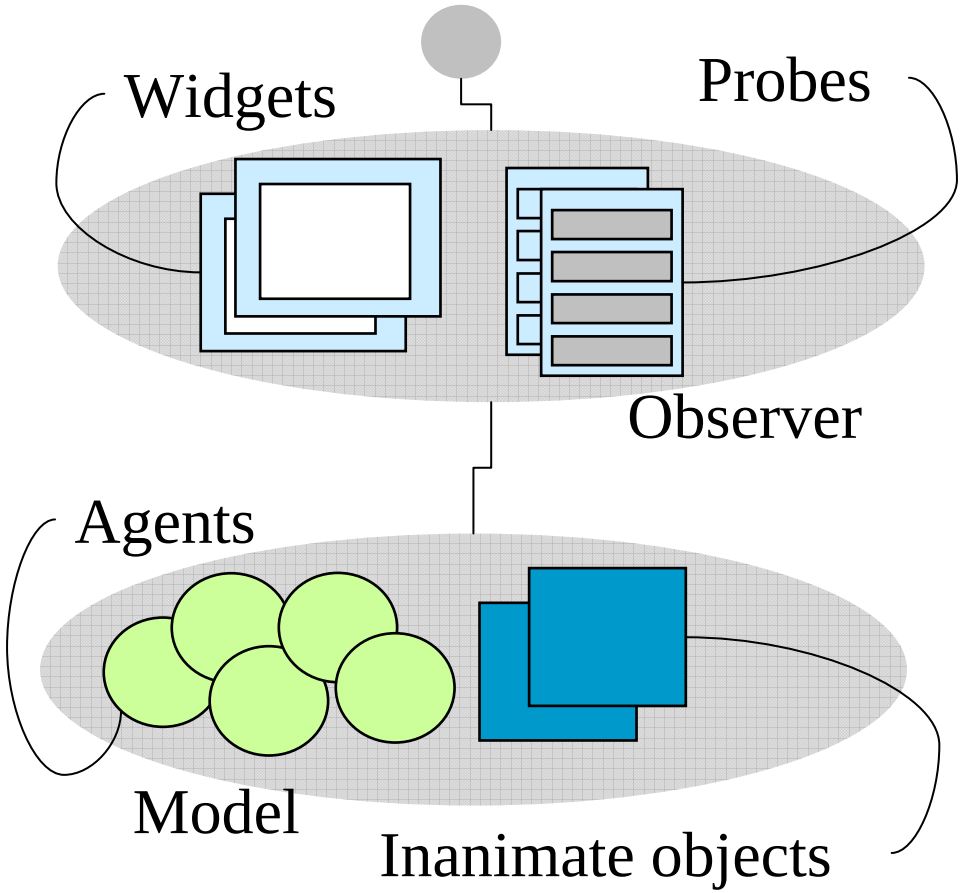
Implementation of output often left to the programmer

The (Swarm) OOP way



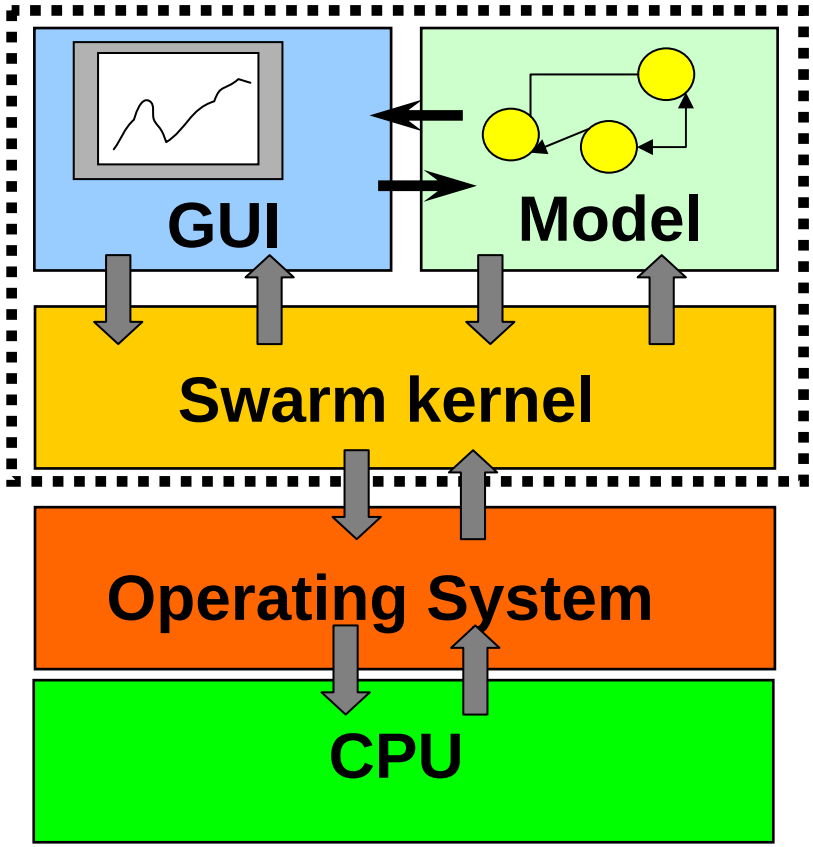
The Swarms

- A Swarm is an object that implements
  - memory allocation
  - event scheduling
- A basic Swarm simulation consists of
  - Observer Swarm
  - Model Swarm



A Swarm as a virtual computer

- A computer's CPU executes program instructions
- Swarm kernel is virtual CPU running model and GUI events
- Nested Swarms merge activity schedules into one

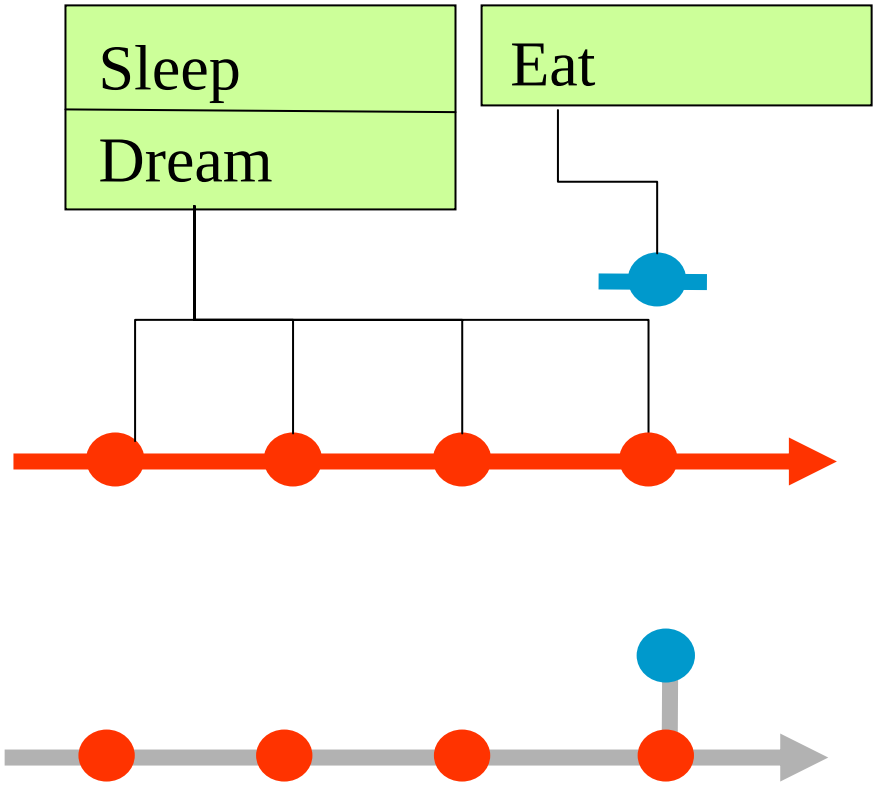


The Activity framework

- Provides abstractions that allow programmer to treat event scheduling in the artificial world as objects, separate from elements of model
- Provide some useful shorthand to group actions and sort according to the desired order of events
- Since schedules are objects, all agents in simulation can communicate with them, to schedule future events or cause actions to be dropped based on specified contingencies

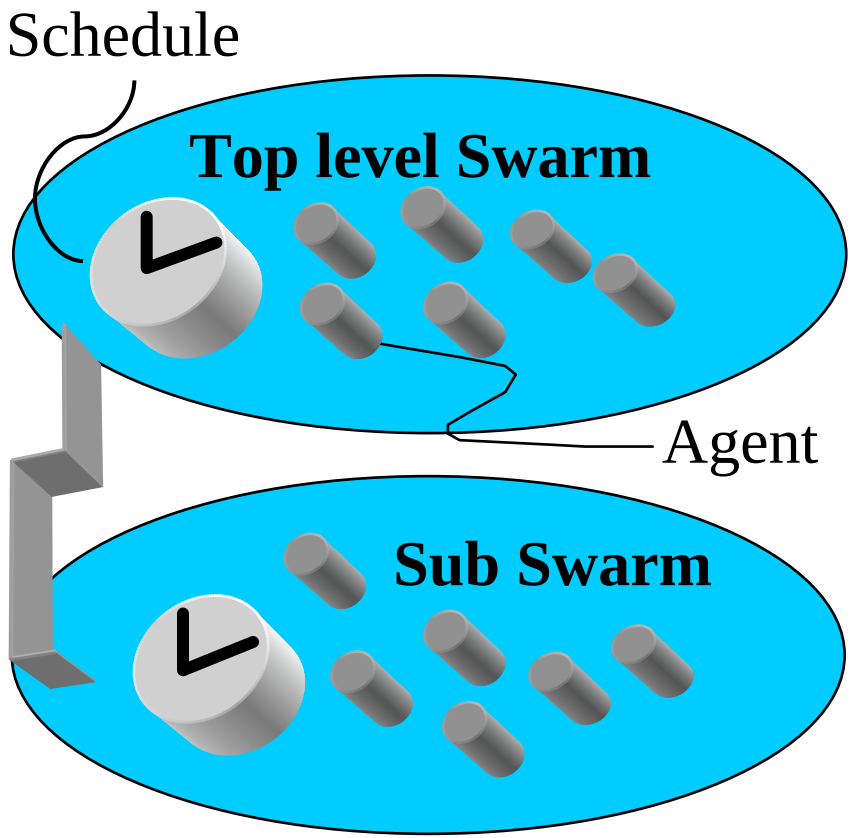
Elements of Activity library

- ActivityGroup
  - Unordered set of "simultaneous" events
- Schedules
  - Ordered set of events or groups of events
- SwarmActivity
  - Underlying VM executing schedules



Merging schedules in Swarms

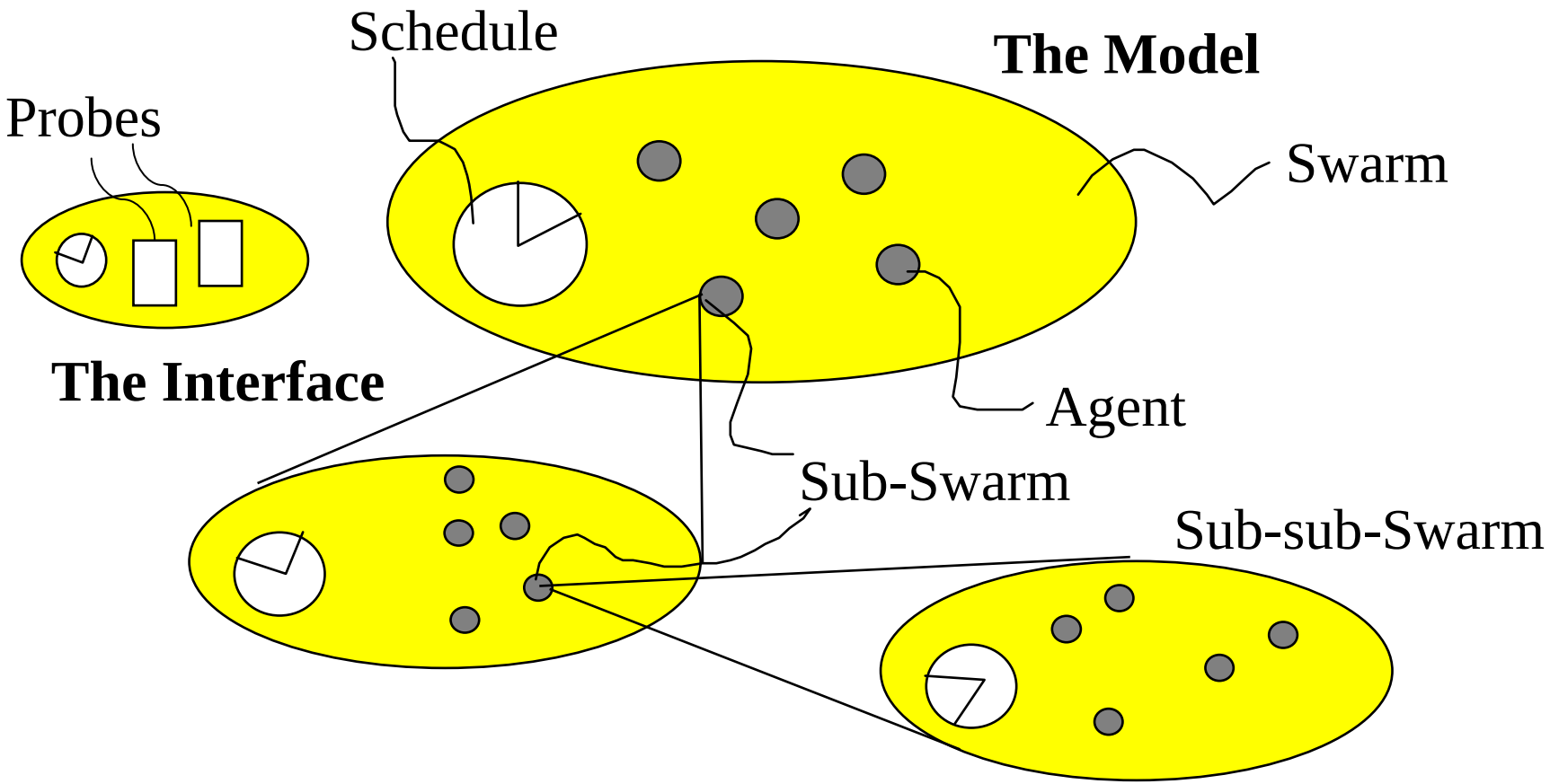
- The schedules of each sub Swarm are merged into the schedule of next level Swarm
- Finally all schedules are merged in the top level Swarm
- Allows you to treat the model as a nested hierarchy of models



Managing memory in Swarms

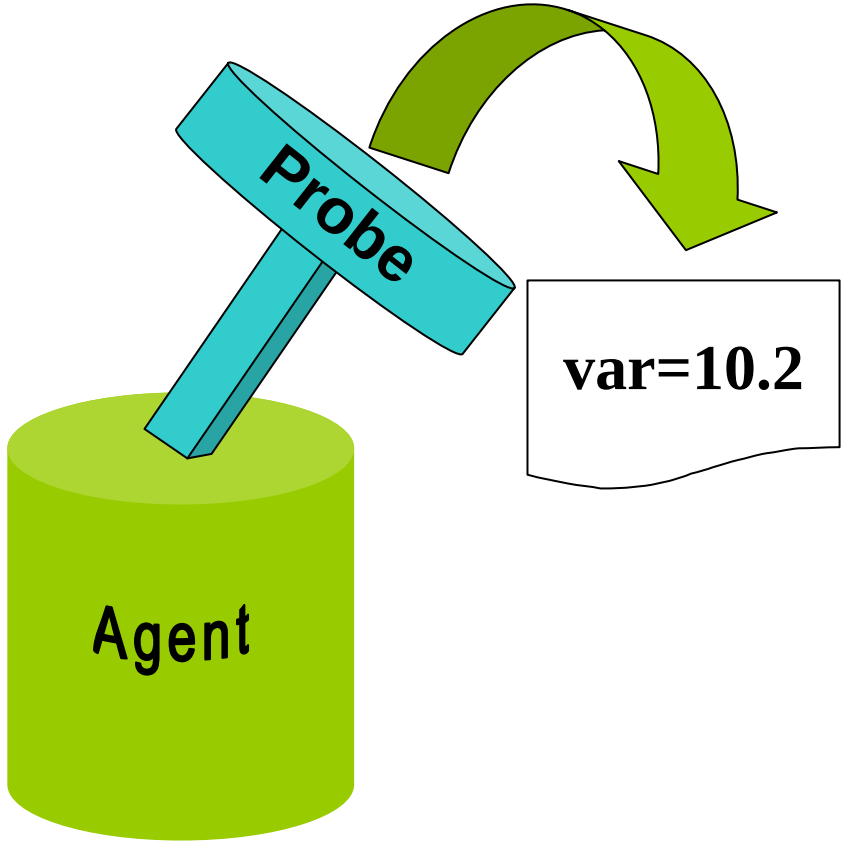
- The allocation and deallocation of memory is a necessary feature in an OO system
- Swarm provides libraries for this purpose which make the process transparent to user
- Basic feature: Objects are created and destroyed using a notion of **memory zones**
- By dropping a memory zone can drop entire collections or sub Swarms

The recursive structure of Swarms



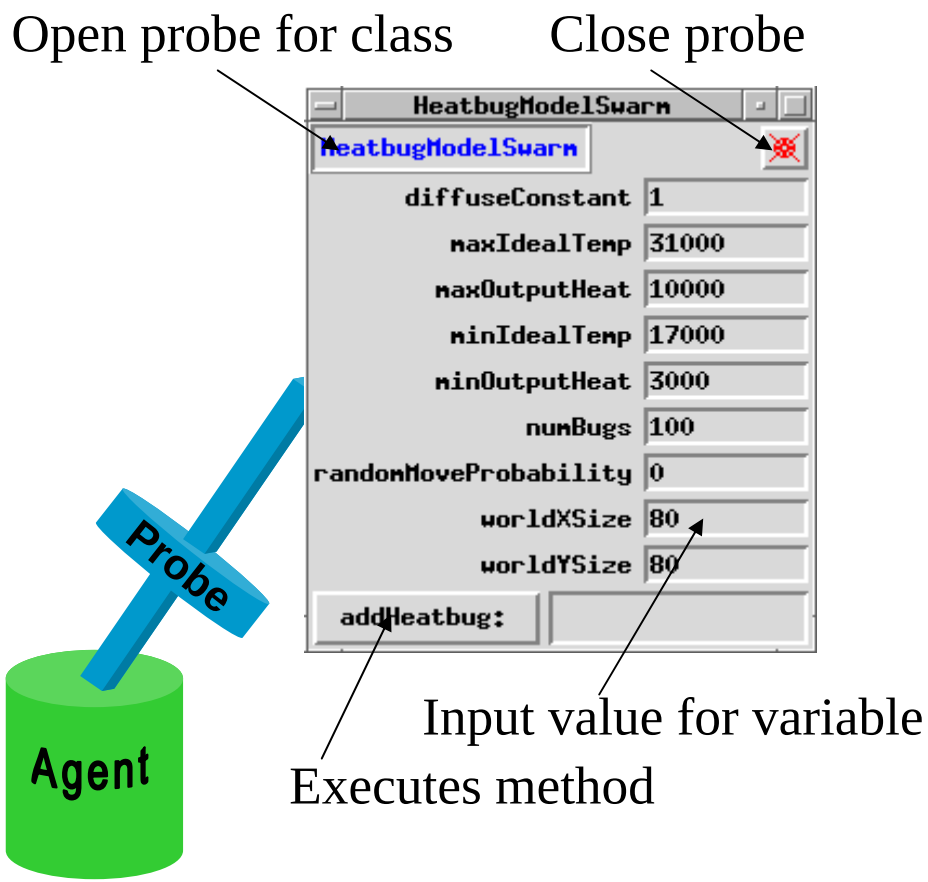
The Probes

- All agents and objects in Swarm can be probed
- Probe attaches itself to an agent, can send a message, change a variable or retrieve values by calling agent or reading variable directly



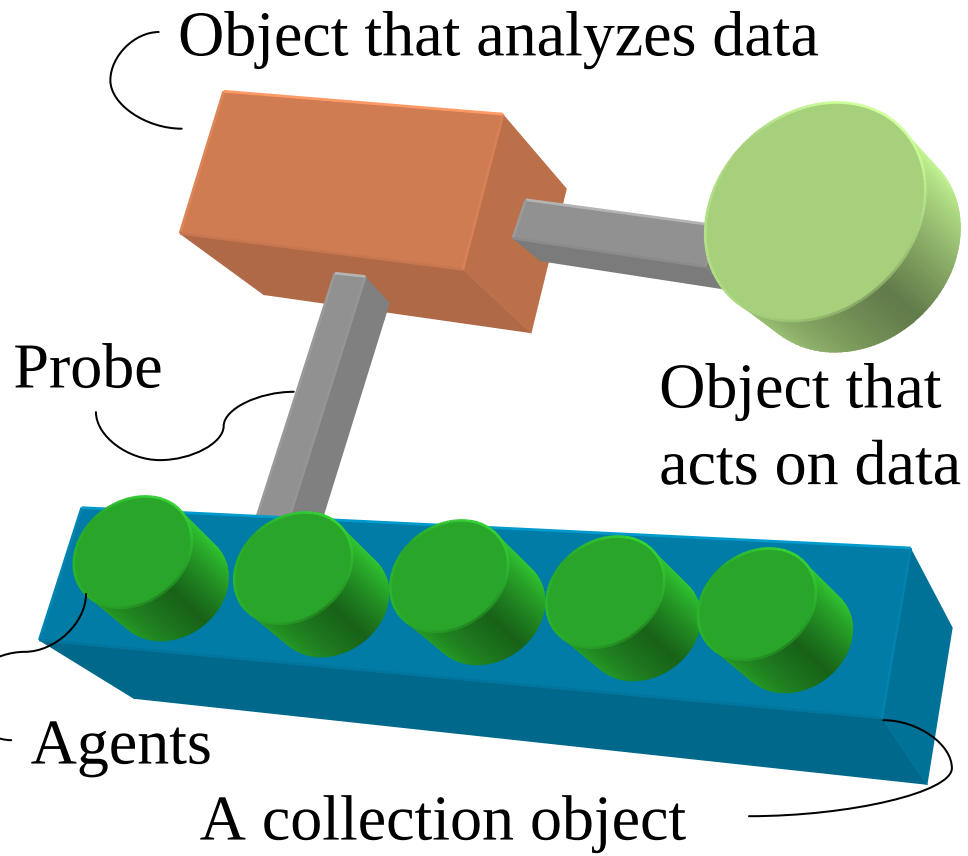
Probes and the GUI

- A probe can be used to communicate with objects in real time through a graphical widget
- A default **probe map** of agent shows all variables and functions - can also customize display



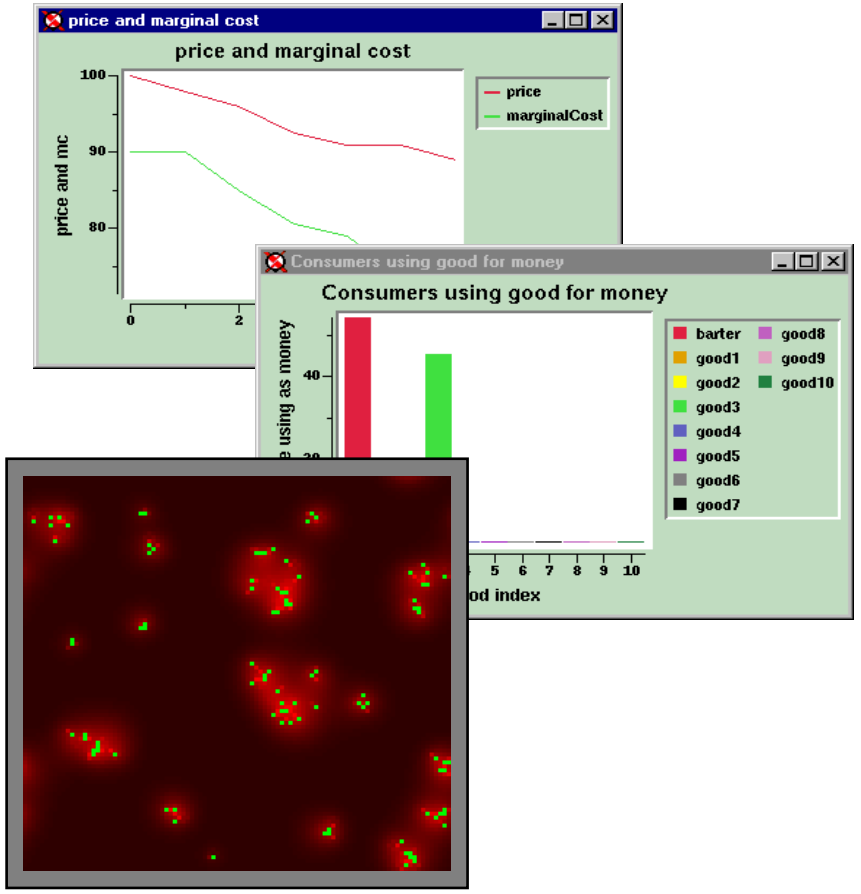
Probes without a GUI

- Probes are also used to gather information dynamically from single agents or collection of agents
- Among other features is the ability to define and execute method calls on the fly at runtime



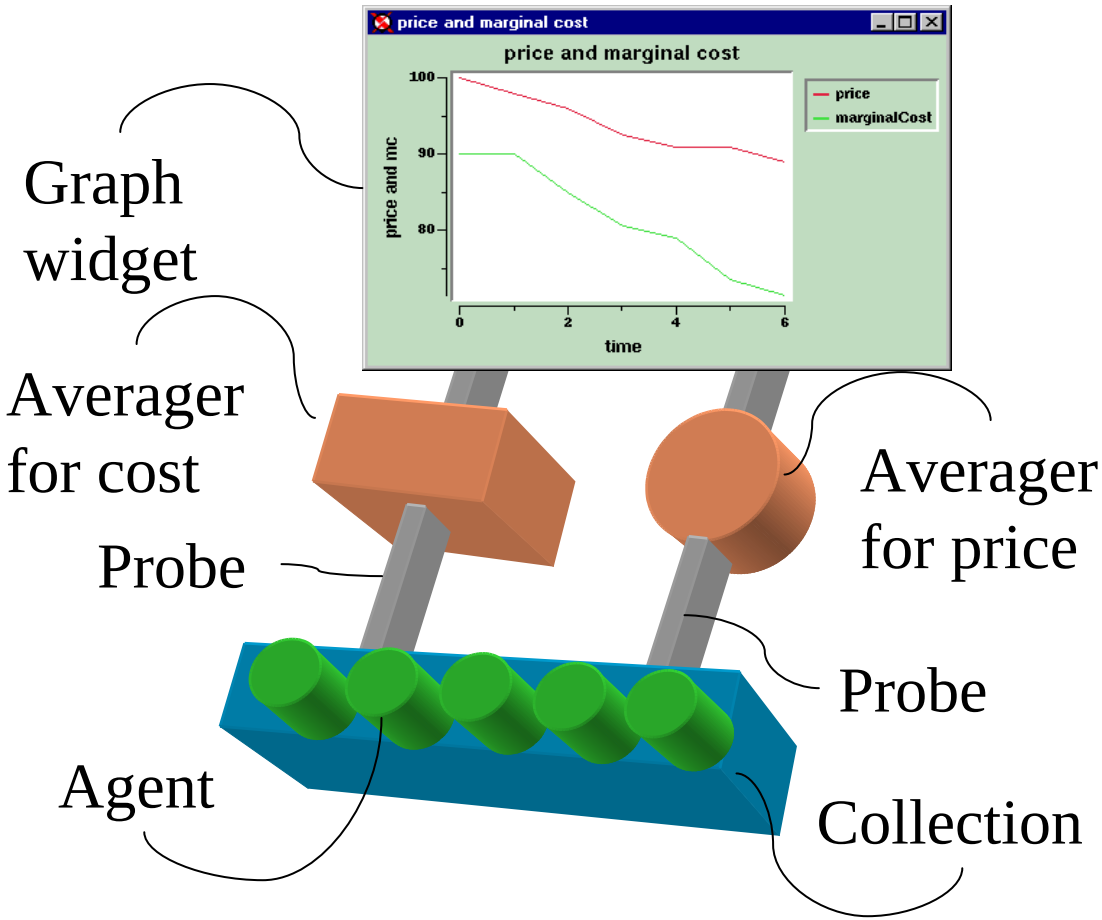
The Graphical User Interface

- Objects provided to create and manage
  - Line graphs
  - Histograms
  - Raster images
  - Digraphs
- Data collection, calculation and updating is provided through support objects to the GUI widgets



Feeding data to the GUI widgets

- Output on graphs or other widgets relies heavily on the probes and dynamic access to agent's data
- Swarm collection objects also facilitate process

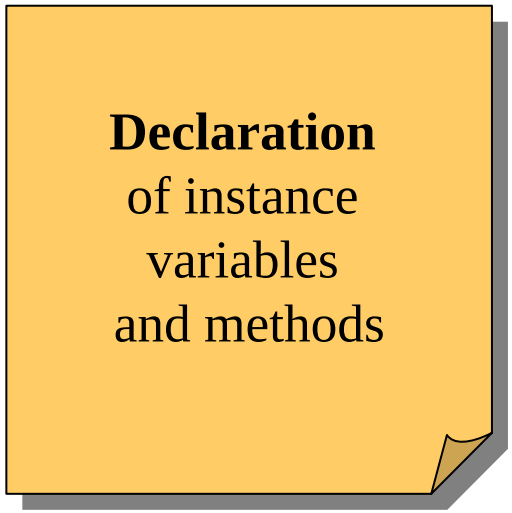


Why Java?

- History: Refer to JAVA inside story (http://www.hotwired.co.jp/wiredmagazine/2.03/java.html)
- Main difference between C++ and Java:
  - Java doesn't have operator overloading
  - Java doesn't support a template class
  - Java has garbage collection

Writing objects in Java

- A program or project consists of a collection of object definitions (*.java)



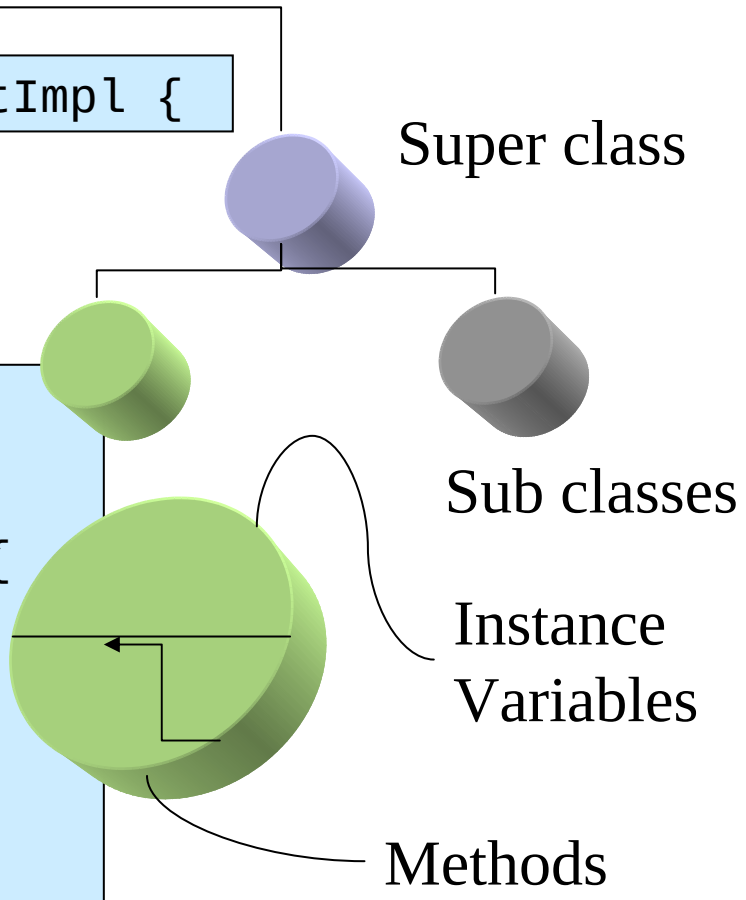
*.java

A few Java basics

```
public class Bug extends SwarmObjectImpl {

   int xPos, yPos;
   int worldXSize, worldYSize;
   id foodSpace;

   public Bug(Zone aZone){
     super(aZone);
   }//constructor
   public Object setX$Y(int x,int y){
     ...
   }
   public Object step(){
     ...
   }
}
```



Some basic syntax

```
public class Bug extends
 SwarmObjectImpl {
  int xPos, yPos;
  int worldXSize,worldYSize;
  FoodSpace foodSpace;
  public Bug(Zone aZone){
    super(aZone);
  }//constructor
  public Object step(){
    ...
  }
  public Object setX$Y
   (int x,int y){
    ...
  }
}
```

- `type message()`
  - declares a method called `message'. E.g. ‘Object step(){}’
- `type message((type) v)`
  - declares method called ‘message(v)’ that takes one argument
- `type setX$Y(int x,int y)`
  - declares method called ‘setX$Y(x,y)’ that takes two arguments

More Java syntax

- Defining methods

```
type aMessage((type) v){...}
type aMessage$with$and((type)v1,(type)v2, (type)
  v3){...}
```

- Calling methods

```
object.aMessage(val)
object.aMessage(val1,val2,val3)
```

The Object variable type etc.

- Default variable type for object in Java is Object
- Think of this as a special variable type (which is actually a pointer to a special data structure - namely the object)
- All objects can refer to themselves by

  this.
- All objects can refer to superclass by

  super.

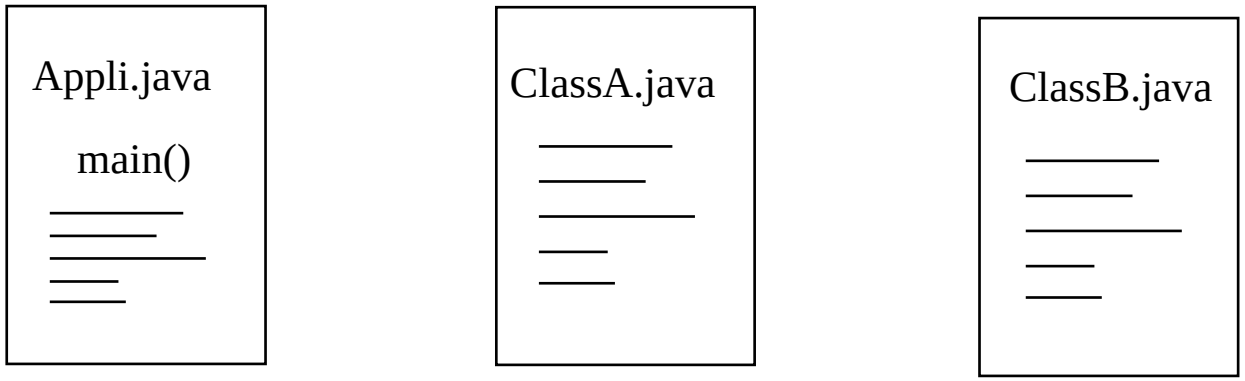
Declaring and Defining a class

- The header file or interface declares
  1. – Class name
  2. – It's superclass
  3. – Instance variables
  4. – Methods

```
1 public class Name
    extends SuperClass{ 2
      type Iv1
      type Iv2                    3
      ...
      type IvN

      public Name(Zone aZone){    4
        super(aZone);
      }//constructor
      public (type) bMethod((type)
        v1){
        ...
      }
}
```

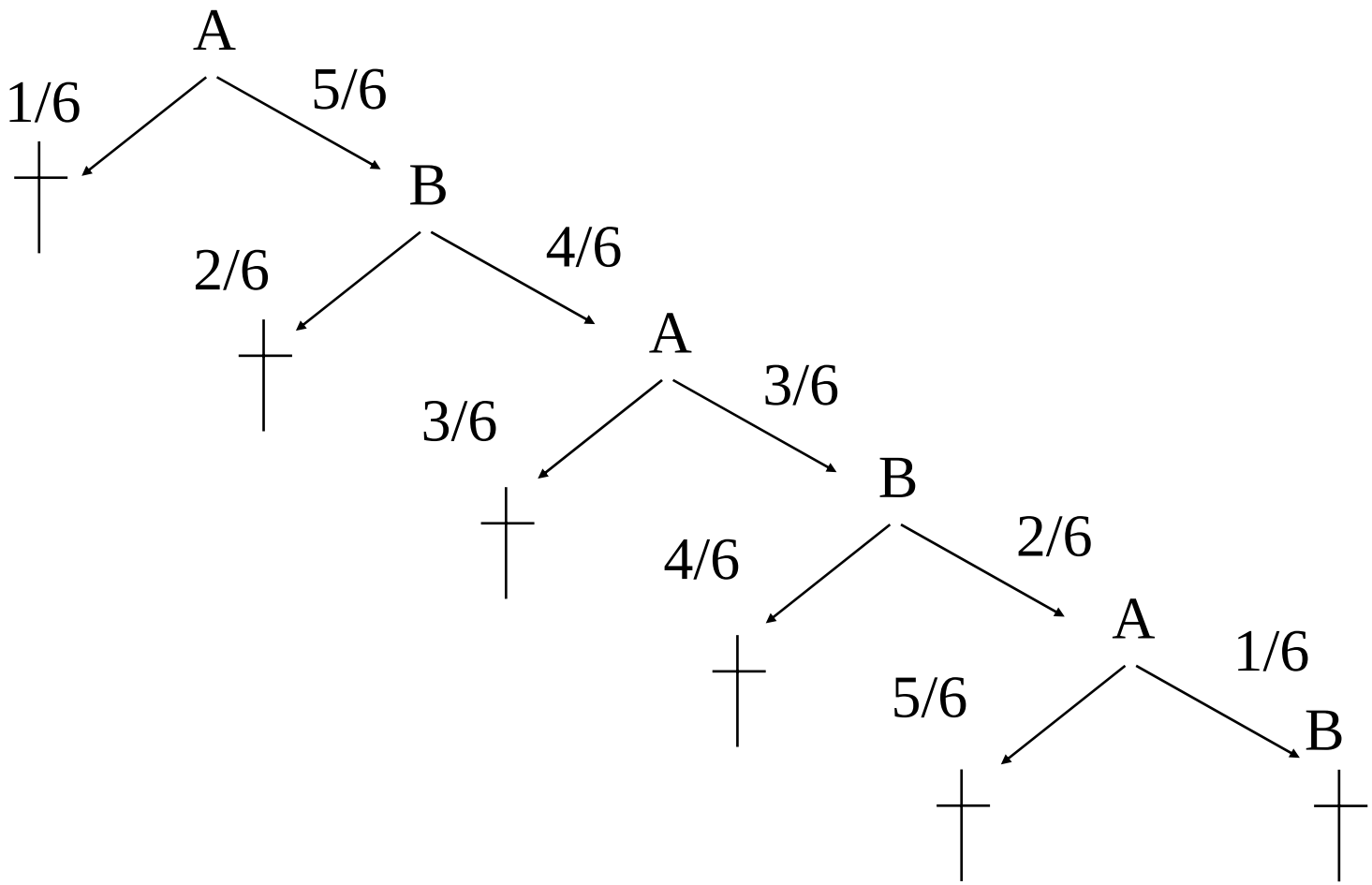
Typical small Java project



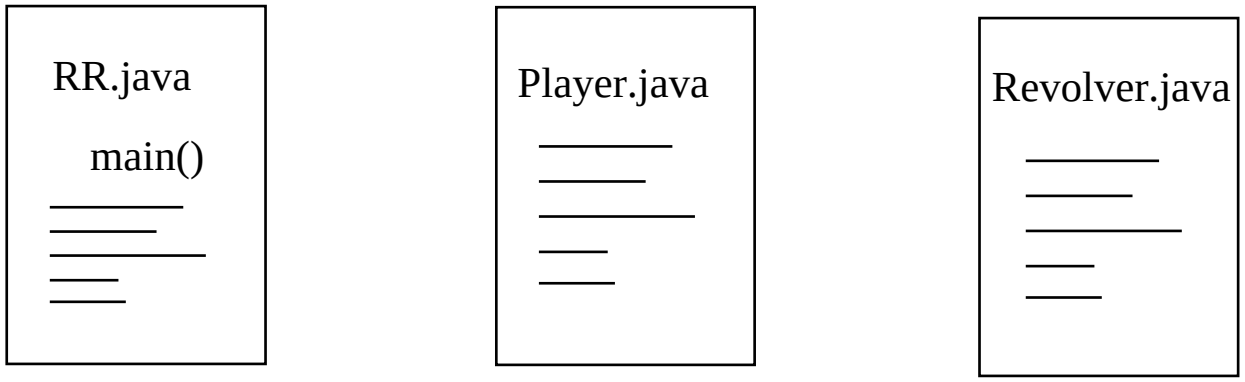
Russian roulette

- Two players A and B alternate aiming revolver at themselves
- Gun starts empty
- Before pulling trigger each player must load new bullet in revolver
- If revolver doesn’t go off player hands gun over to opponent

Survival probabilities



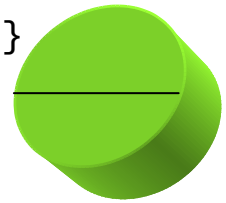
Example project: Russian roulette



The objects: Player

- Variables: Player knows
  - His own name
  - Whether he is alive
  - Idenity of other player
- Behavior: Player can
  - Set variables
  - Respond to “alive?”
  - Use revolver

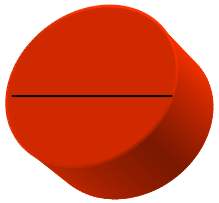
```
public class Player extends
   SwarmObjectImpl{
  int name;
  int alive;
  Player other;
  public Player(Zone aZone){...}
  public void init(int n){
    ...
  }
  public void setOther(Player o){
    ...
  }
  public boolean isAlive(){
    ...
  }
  public void play(Revolver r){
    ...
  }
}
```



The objects: Revolver

- Variables: Revolver knows
  - Number of bullets
- Behavior: Gun can
  - Empty bullets
  - Load new bullet
  - Engage trigger and return result (fired, didn't fire)

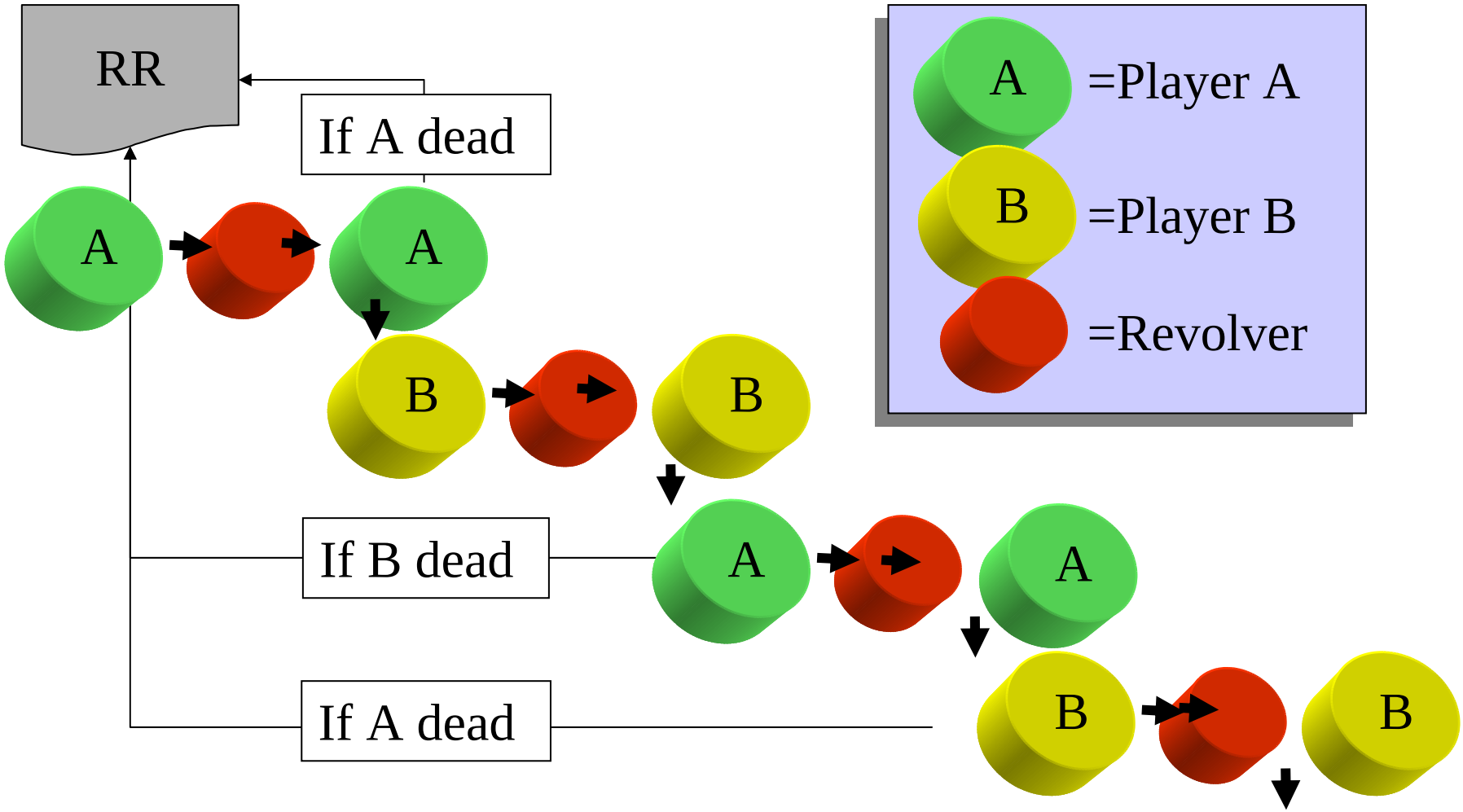
```
public class Revolver extends
   SwarmObjectImpl{
  int bullets;
  public Revolver(Zone
   aZone){...}
  public void empty(){
    ...
  }
  public void load(){
    ...
  }
  public boolean trigger(){
    ...
  }
}
```



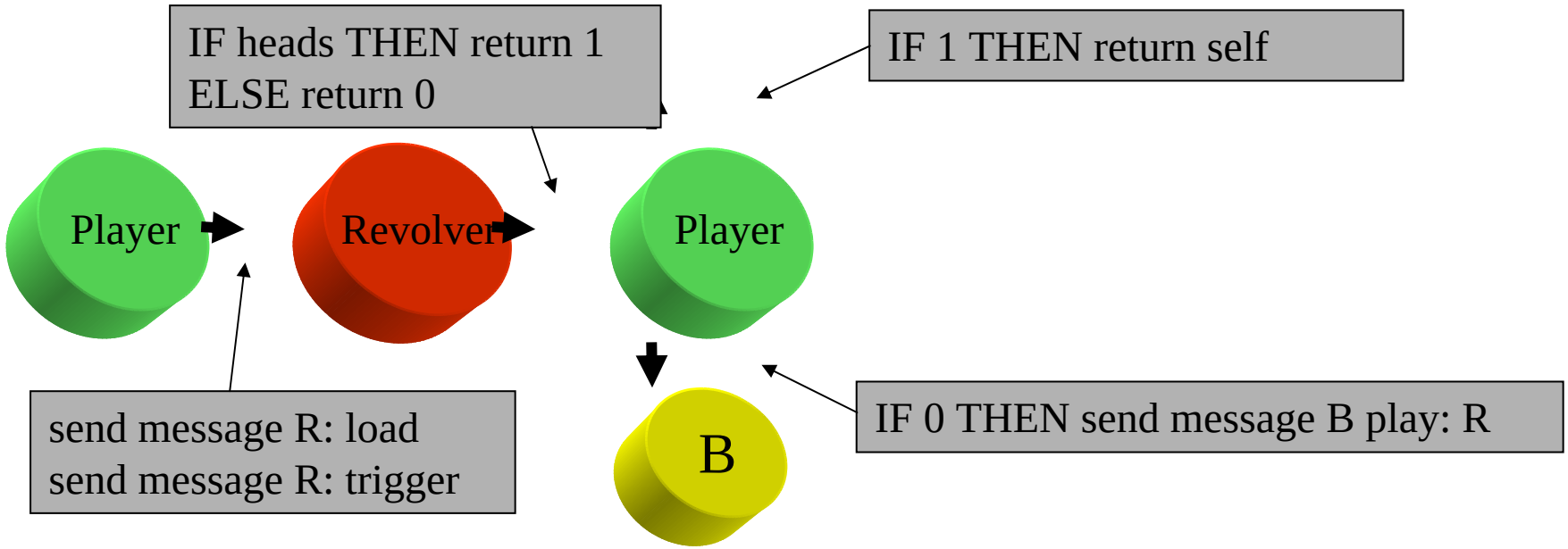
How it works

- Objects are created in RR.java
- RR hands empty revolver object off to player A
- Player A loads bullet and pulls trigger
- **If** player A is alive hands gun to B **else** returns herself to RR
- Player B loads bullet and pulls trigger
- **If** player B is alive hands gun to A **else** returns herself
- etc...

How it works (cont)



How it works (cont)



Other resources

- Java
  - Many textbooks available because…
  - Main resource: Sun website (http://java.sun.com/)
- Swarm
  - Main website at SDG: Code, manuals, links
  - User community, mailinglists (and archives)
  - Beg, steal and borrow other people's code!

Where to find help

- Main Swarm website
  - http://www.swarm.org/
- The Java manual
  - http://java.sun.com/
- Subscribing to mailinglist
  - Send mail to majordomo at santafe.edu with

    subscribe swarm-support <Your E-mail>