

Swarmfest '99 Tutorial

Session II: Simplebug demo

Benedikt Stefansson

<benedikt at ucla.edu>

revised version 0.3

by YABUKI Taro for Java

Second session

- Tutorial: Simplebug
 - Start: From non-OO to OO to Swarm
 - Adding GUI: Probes, raster image, graphs
 - Creating and controlling full experiment run
- Topics covered:
 - Managing objects
 - Swarms, activity library, probes, GUI

Chapters in Simplebug tutorial

- `simpleCBug` Simple non-OO code
- `simpleObjCBug` Bug is now Java class
- `simpleObjCBug2` Adds grid holding food
- `simpleSwarmBug` Adds a ModelSwarm
- `simpleSwarmBug2` Many bugs...
- `simpleSwarmBug3` Reads parameters from file
- `simpleObserverBug` Adds ObserverSwarm & GUI
- `simpleObserverBug2` Adds more sophisticated GUI
- `simpleExperBug` Runs multiple experiments

Bug in non-OO

1
2

```
import swarm.*;
public class simpleCBug{
    public static void main(String[] args){
        int worldXSize = 80; // Maximum X value
        int worldYSize = 80; // Maximum Y value
        int xPos = 40; // Bug's starting position
        int yPos = 40;
        int i;
```

3

```
Globals.env.initSwarm("bug", "0.1", "t_yabuki at nifty dot com", args);

System.out.println("I started at X = " + xPos + " Y = " + yPos + "\n");

for(i = 0; i < 100; i++) {
    // Random movement in X and Y (possibly 0)
```

4

```
xPos = xPos + Globals.env.uniformIntRand.getIntegerWithMin$withMax(-1,1);
yPos = yPos + Globals.env.uniformIntRand.getIntegerWithMin$withMax(-1,1);

// Take move modulo maximum coordinate values

xPos = (xPos+ worldXSize) % worldXSize;
yPos = (yPos + worldYSize) % worldYSize;

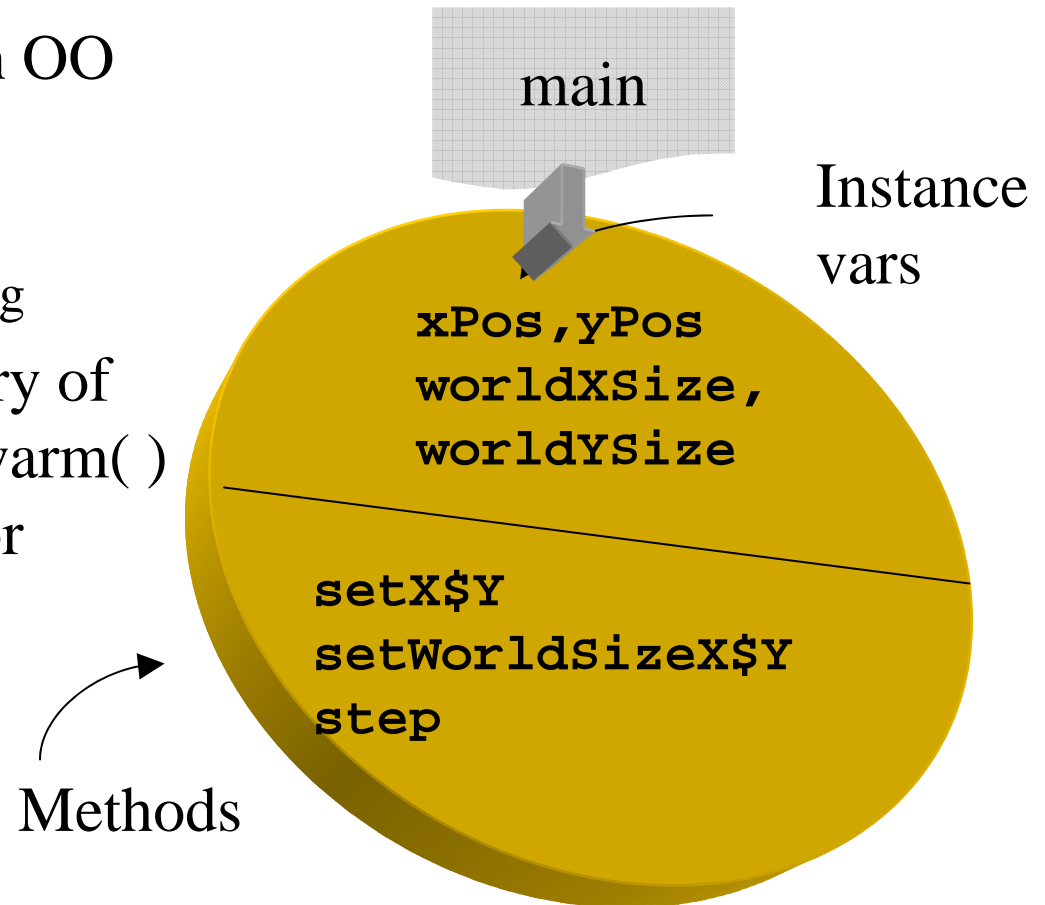
System.out.println("I moved to X = " + xPos + " Y = " + yPos);
}
}
}
```

Bug in non-OO explanations

- 1 Importing package gives us access to random number obj etc.
- 2 Each Java program must have **main** function, which can take an argument:
 - args: value of command line args
- 3 `initSwarm()` function checks command line args and revs up Swarm engine
- 4 The program uses default random number distributions to choose new location for bug

Bug in OO

- The simplest version of an OO program
 - A main() function
 - One class, one instance: Bug
- main() imports base library of Swarm and fires up initSwarm() which gives us libraries for memory allocation etc.



Bug in OO: The bug as object

- Inheriting from the SwarmObjectImpl class Bug knows these tricks:
- First instruct class to create instance of itself. Associate instance with aBug

- construction: Allocate memory
- drop: Deallocate and die

```
Bug aBug=new
```

```
Bug(Globals.env.globalZone);
```

- Then set parameters in the instance, aBug

```
aBug.setX$Y(xPos,yPos);
```

```
aBug.setWorldXSize$Y(worldXSize,w  
orldYSize);
```

Typical create phase

Allocate memory etc.

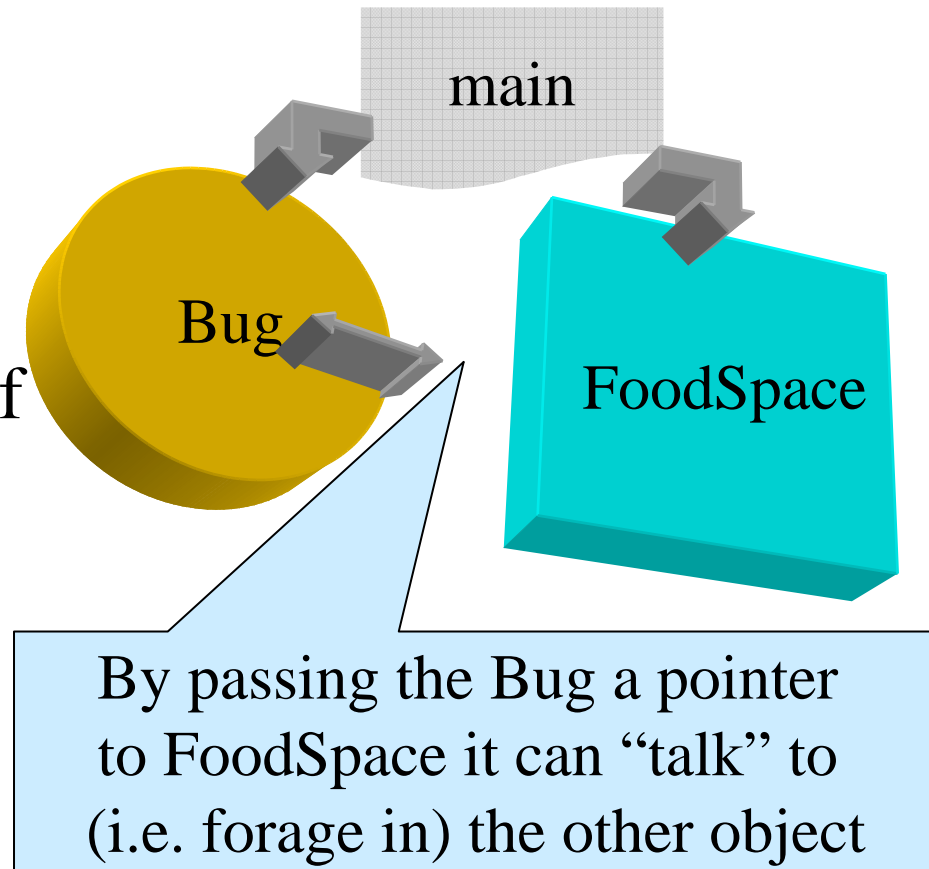
```
Bug aBug =new Bug(Globals.env.globalZone);
```

Set all necessary
parameters

```
aBug.setWorldSizeX$Y(xsize,ysize);  
aBug.setFoodSpace(foodSpace);  
aBug.setX$Y(xPos,yPos);
```

Bug in OO II: The FoodSpace

- Here create a world for the bug to roam around in and eat
- The world is defined in FoodSpace, a subclass of Discrete2dImpl
- Discrete2d manages a lattice of integer values



The FoodSpace class

- By subclassing from Discrete2dImpl FoodSpace inherits several Ivars and about 16 methods to store and retrieve values in 2d space
- The only addition is a method to fill space with “food”

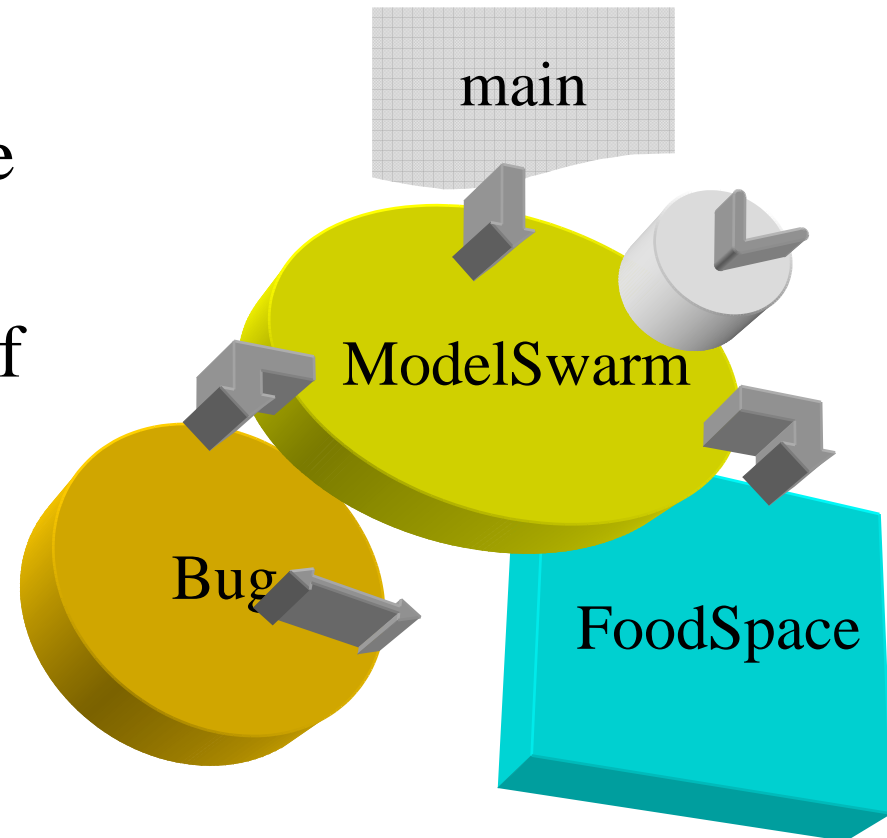
```
public Object seedFoodWorld(Prob prob)
(double seedProb){
int x,y;
int xsize,ysize;
xsize=this.getSizeX();
ysize=this.getSizeY();
for (y = 0; y < ysize; y++)
for (x = 0; x < xsize; x++)
if (Globals.env.uniformDb1Rand
.getDoubleWithMin$withMax
(0.0,1.0)<=seedProb)
this.putValue$atXY(1,x,y);
return this;
}
```

Inherited method

Inherited method

Bug in Swarm: The ModelSwarm

- An instance of the Swarm class can manage a model world
- Facilitates the creation of agents and interaction model
- Model can have many Swarms, often nested



Creating a Swarm

I. construction

- Initialize memory and parameters

II. buildObjects

- Build all the agents and objects in the model

III. buildActions

- Define order and timing of events

IV. activate

- Merge into top level swarm or start Swarm running

Step I: Initializing

1

```
public ModelSwarm(Zone aZone) {
```

2

3

```
    super(aZone);
```

4

```
    worldXSize = 80;
```

```
    worldYSize = 80;
```

```
    seedProb = 0.1;
```

```
    xPos = 40;
```

```
    yPos = 40;
```

```
}
```

Details on constructional method

1 This is a constructor. It can't return a value or object.

3 super

– Executes a constructor of the super class of obj (SwarmImpl). It allocates memory.

4

• varName=val

– It is same to “this.varName=val”. The name of itself can be omitted.

Memory zones

- The interface `swarm.defobj.Drop` provides facilities to drop an object - deallocate memory
- Each object is created in a memory zone
- Effectively this means that the underlying mechanism provides enough memory for the instance, its variables and methods.
- The zone also keeps track of all objects created in it and allows you to reclaim memory simply by dropping a zone. It will signal to all objects in it to destroy themselves

Where did that zone come from?

In main : `initSwarm (...);`

Executes various functions
which create a global memory zone
among other things

In main: `modelSwarm=new ModelSwarm(Globals.env.globalZone);`

Call the constructor of **ModelSwarm**

In ModelSwarm.java: `modelSwarm():`

Step II: Building the agents

```
public Object buildObjects(){  
    foodSpace =new FoodSpace  
        (Globals.env.globalZone,worldXSize,worldYSize);  
    foodSpace.seedFoodWithProb(seedProb);  
  
    Bug aBug =new Bug(Globals.env.globalZone);  
    aBug.setWorldSizeX$Y(worldXSize,worldYSize);  
    aBug.setFoodSpace(foodSpace);  
    aBug.setX$Y(xPos,yPos);  
  
    return this;  
}
```

Details on the buildObjects phase

- The purpose of this method is to create each instance of objects needed at the start of simulation, and then to pass parameters to the objects
- It is good OO protocol to provide setX: methods for each parameter X we want to set, as in:
`aBug.setFoodSpace(foodSpace)`

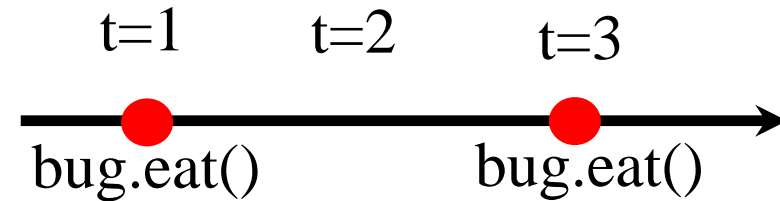
Step III: Building schedules

```
public Object buildActions() {  
    modelSchedule=new ScheduleImpl(this,1);  
    try {  
        modelSchedule.at$createActionTo$message  
            (0,aBug,new Selector(Class.forName("Bug"),  
                "step",false));  
    } catch (Exception e){  
        System.exit(1);  
    }  
    return this;  
}
```

Schedules

- Schedules define event in terms of:

- 1 – Time of first invocation
- 2 – Target object
- 3 – Method to call
- 4 – Repeat interval

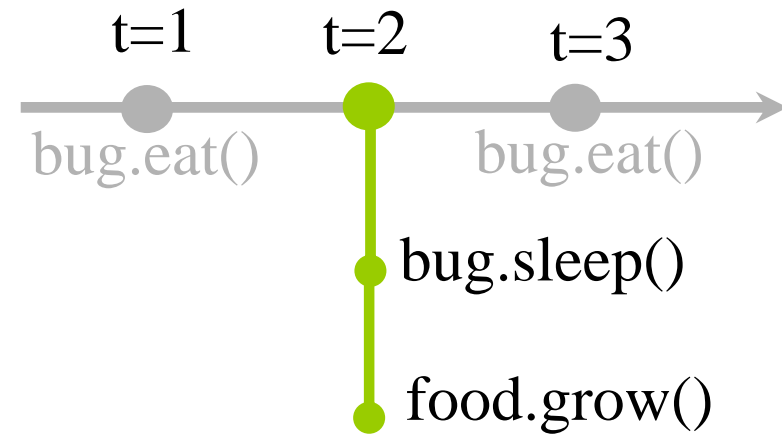


```
modelSchedule.at$  
createActionTo$  
message(t,agent, method)
```



ActivityGroups

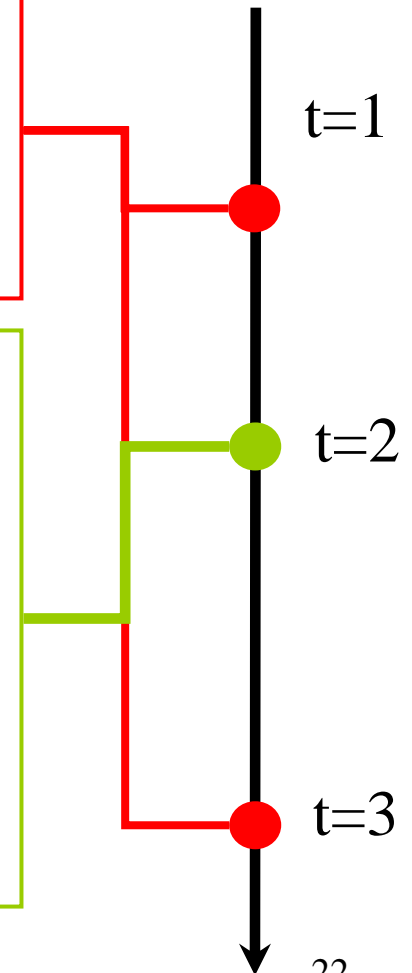
- ActivityGroups group together events at same timestep
- Schedule then “executes” group



Implementation

```
schedule1=new ScheduleImpl(this,2);  
schedule1.at$createActionTo$message  
  (0,bug,new Selector(Class.forName("Bug"),"eat",false));
```

```
actionGroup=new ActionGroupImpl(this);  
actionGroup.createActionTo$message  
  (bug,new Selector(Class.forName("Bug"),"sleep",false));  
actionGroup.createActionTo$message  
  (bug,new Selector(Class.forName("Food"),"grow",false));  
schedule2=new ScheduleImpl(this,1);  
schedule2.at$createAction(2,modelActions);
```



Step IV: Activating the Swarm

```
public Activity activateIn(Swarm context) {  
  
    super.activateIn(context);  
  
    modelSchedule.activateIn(this);  
  
    return getActivity();  
  
}
```

Activation of schedule(s)

In main: `modelSwarm.activateIn(null);`

There is only one Swarm so we activate it in null

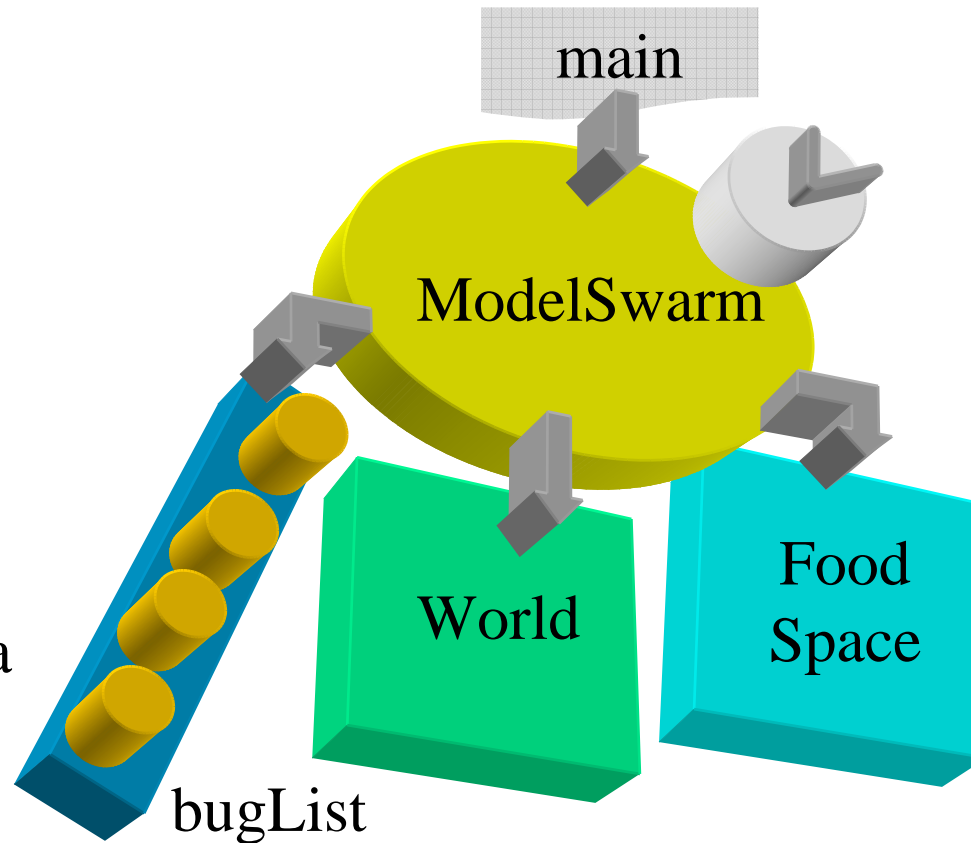
This one line could set in motion complex scheme of merging and activation

`activateIn(Swarm context)`

`modelSchedule.activateIn(this)`

Bug in Swarm II: More bugs

- To manage more than one bug add
 - World, a Grid2d instance that enforces one bug per location
 - bugList, to keep track of bugs as a collection



Creating a population of agents

- Create collection objects (Grid2d, List) to keep track of agents
- In for loop create each instance & initialize
- Then put each agent on list, grid

1

```
(create world)
bugList=new ListImpl(Globals.env.getZone);
for(y=0;y<worldYSize;y++)
  for(x=0;x<worldXSize;x++)
    if(...uniformDblRand...<=bugDensity){
```

2

```
    Bug aBug=new Bug(this);
    aBug.setWorld$Food(world,foodSpace);
    aBug.setX$Y(x,y)
```

3

```
    world.putObject$atX$Y(aBug,x,y);
    bugList.addLast(aBug);
```

Activity and collections

- We now have a collection of agents - bugs
- Schedule now takes as target the collection object - a List instance
- The collection object passes on the message to each bug

```
public Object buildActions(){
    modelActions=new
        ActionGroupImpl(this);
    modelActions.createActionForEach
        $message(bugList,new Selector
            (Class.forName("Bug"),"step",
            false));
    modelActions.createActionTo
        $message(reportBug,new Selector
            (Class.forName("Bug"),"report",
            false));

    modelSchedule=new ScheduleImpl(this,1);
    modelSchedule.at$createAction
        (0,modelActions);

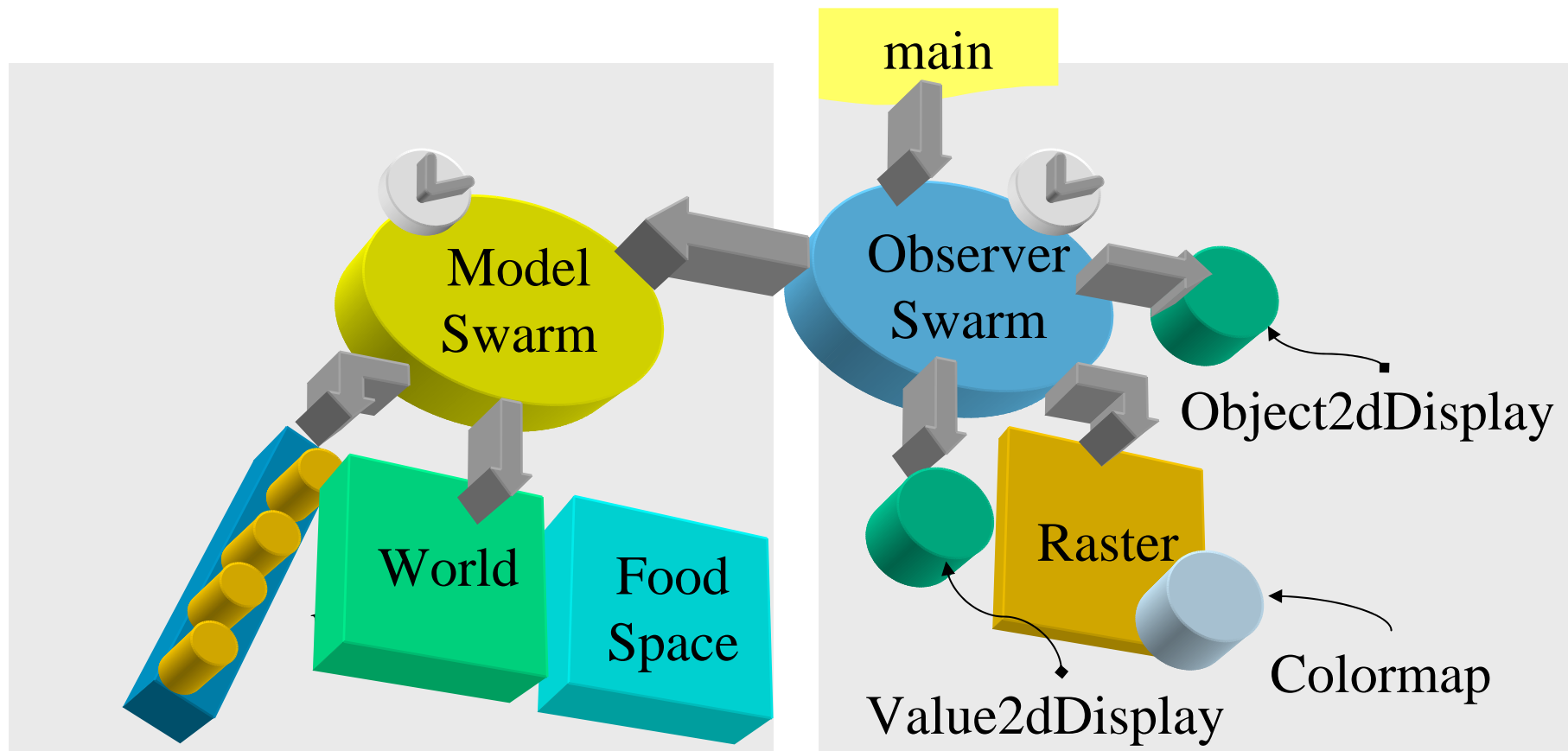
    return this;
}
```

Bug in Swarm III: Loading state

- `lispAppArchiver`
 - Reads values of ivars from file
- Any instance vars not mentioned in `.scm` unchanged
- Instance vars must be public to be set by `lispAppArchiver`
- The `bug.scm` file:

```
(list (cons 'modelSwarm
(make-instance
'ModelSwarm
#:worldXSize 80
#:worldYSize 80
#:seedProb 0.9
#:bugDensity 0.01)))
```

Bug with Observer: Adding GUI



Creating an ObserverSwarm

- constructor
 - Initialize memory and parameters
- buildObjects
 - Build ModelSwarm
 - Build graphs, rasters and probes
- buildActions
 - Define order and timing of GUI events
- activate

Step I: Initializing

```
public observerSwarm(Zone aZone) {  
    super(aZone);  
  
    displayFrequency = 1;  
}
```

Step II: Creating objects

```
public Object buildObjects() {
    super.buildObjects();

    modelSwarm=new ModelSwarm(this);
    getControlPanel().setStateStopped();
    modelSwarm.buildObjects();

    (create Colormap)
    (create Raster)
    (create foodDisplay)
    (create bugDisplay)
    return this;
}
```

**Boldfaced items are
only placeholders**

Step III: Building schedules

```
public Object buildActions() {  
    super.buildActions();  
    modelSwarm.buildActions();
```

```
    (create ActionGroup)
```

```
        (create foodDisplay action)
```

```
        (create bugDisplay action)
```

```
        (create worldRaster action)
```

```
        (create GUI update action)
```

```
    (create Schedule)
```

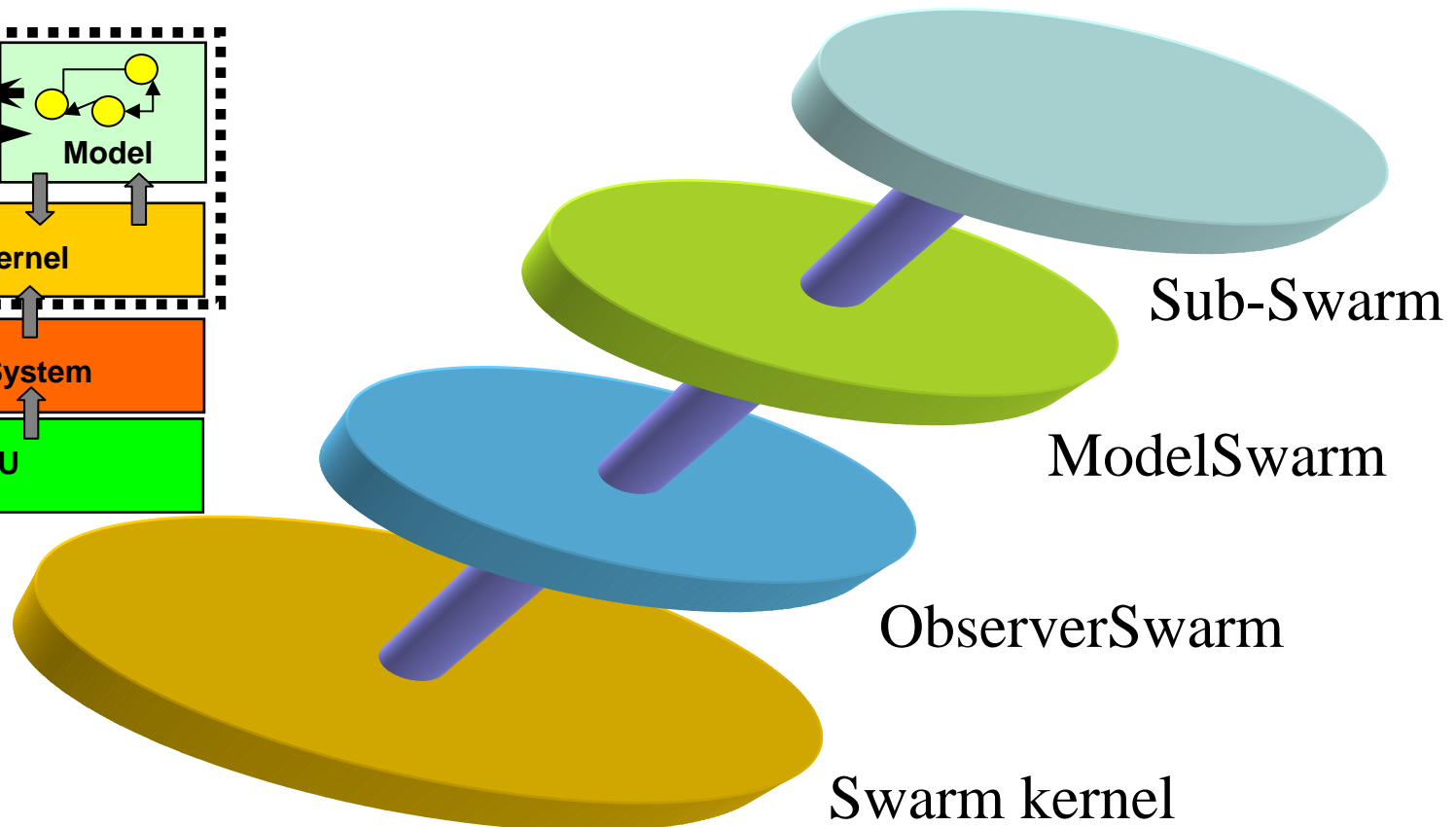
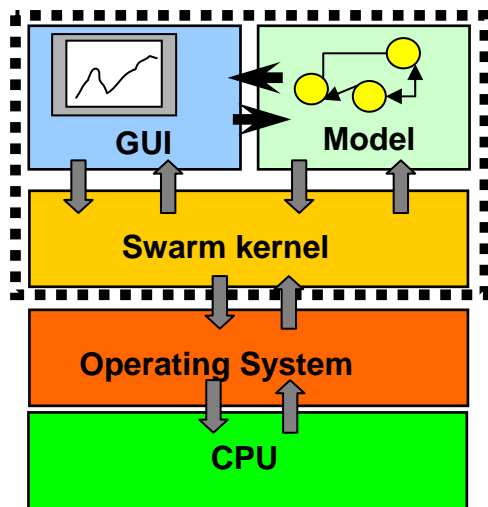
```
        (create action to ActionGroup)
```

```
return this;}
```

Step IV: Activating the Swarms

```
public Activity activateIn(Swarm context) {  
    super.activateIn(context);  
  
    modelSwarm.activateIn(this);  
    displaySchedule.activateIn(this);  
  
    return getSwarmActivity();  
}
```

Integration of Swarm activities



Multilevel activation

In main: `topSwarm.activateIn(null);`

`activateIn(Swarm context)`

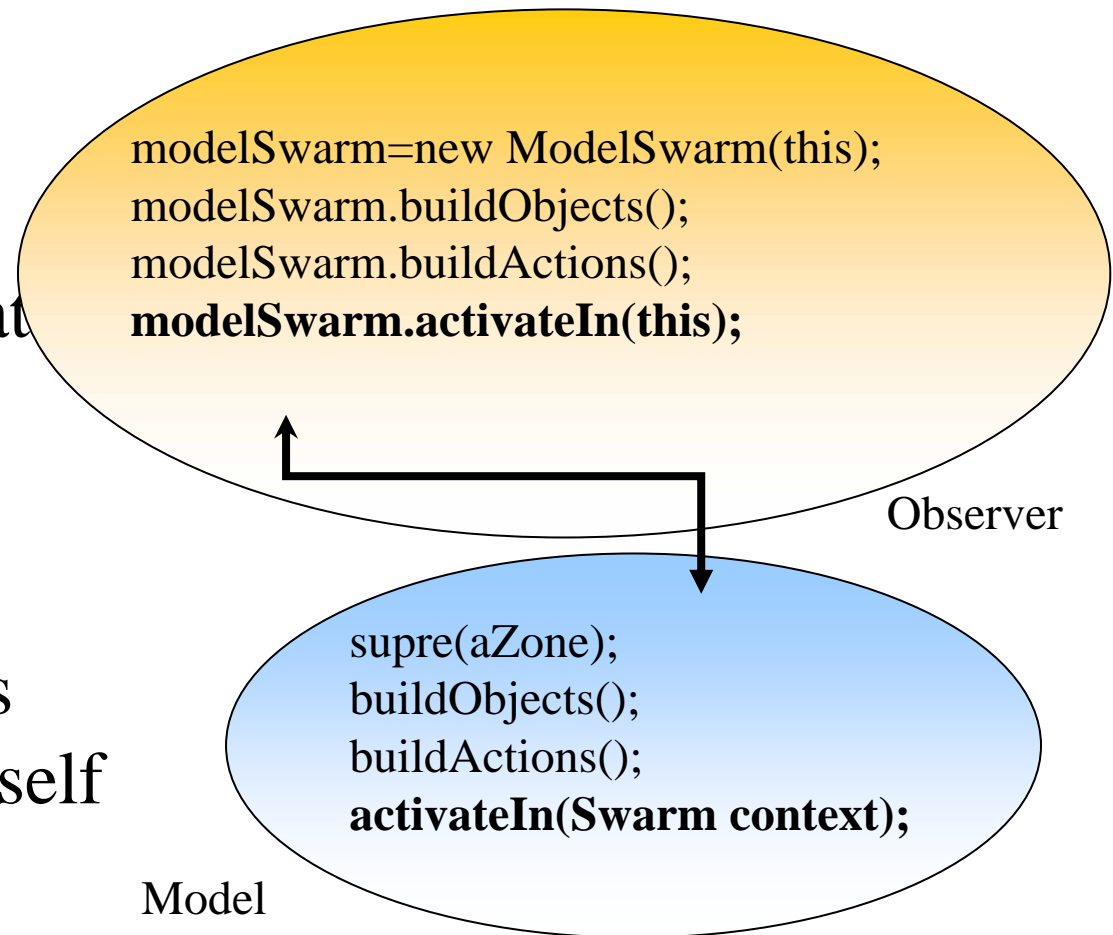
`schedule.activateIn(this);`
`subSwarm.activateIn(this);`

`activateIn(Swarm context)`

`Schedule.activateIn(this);`

Merging two Swarms

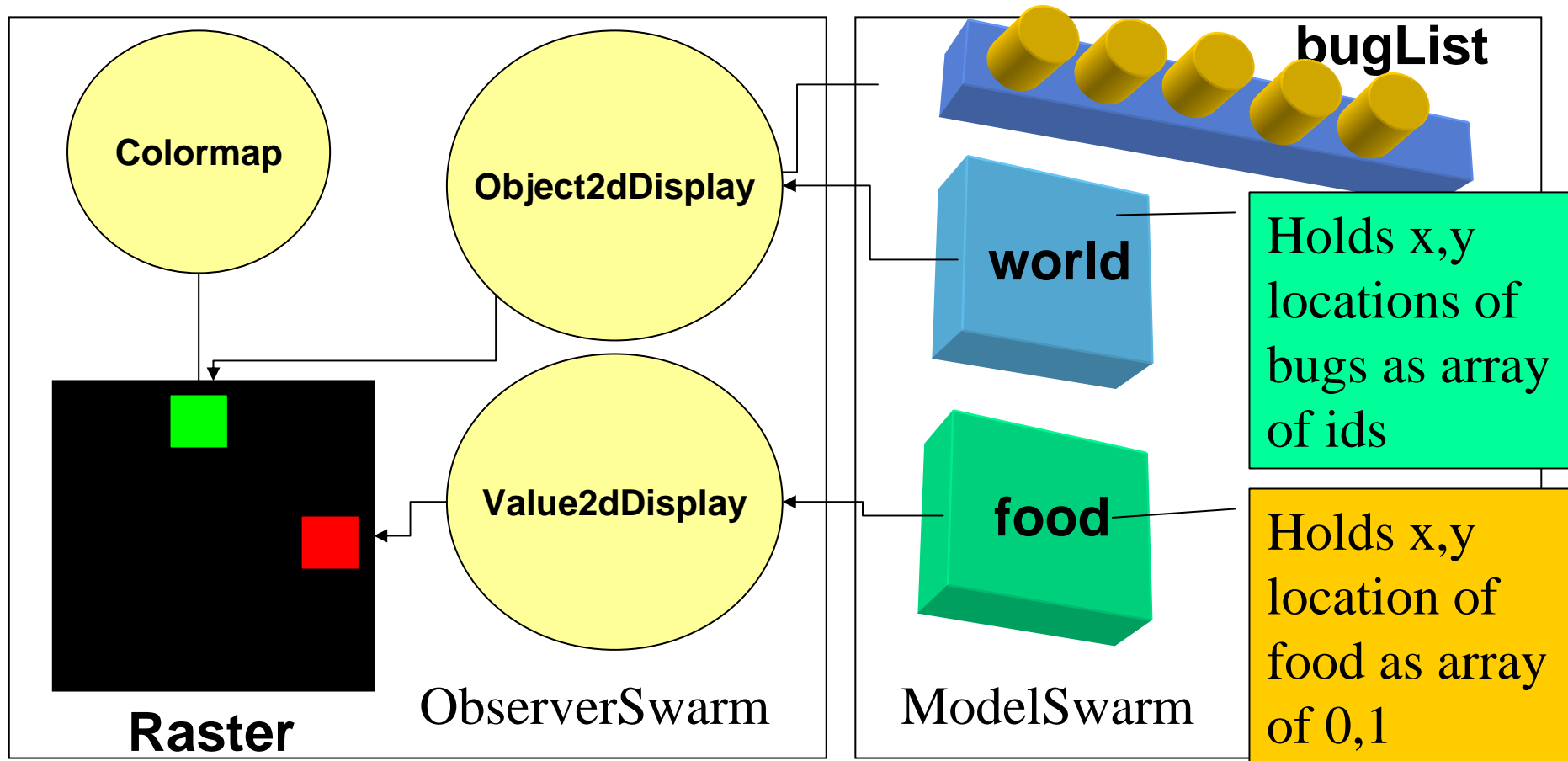
- main() creates ObserverSwarm
- ObserverSwarm creates ModelSwarm as a subswarm in own memory zone
- ModelSwarm creates agents and activates self in ObserverSwarm



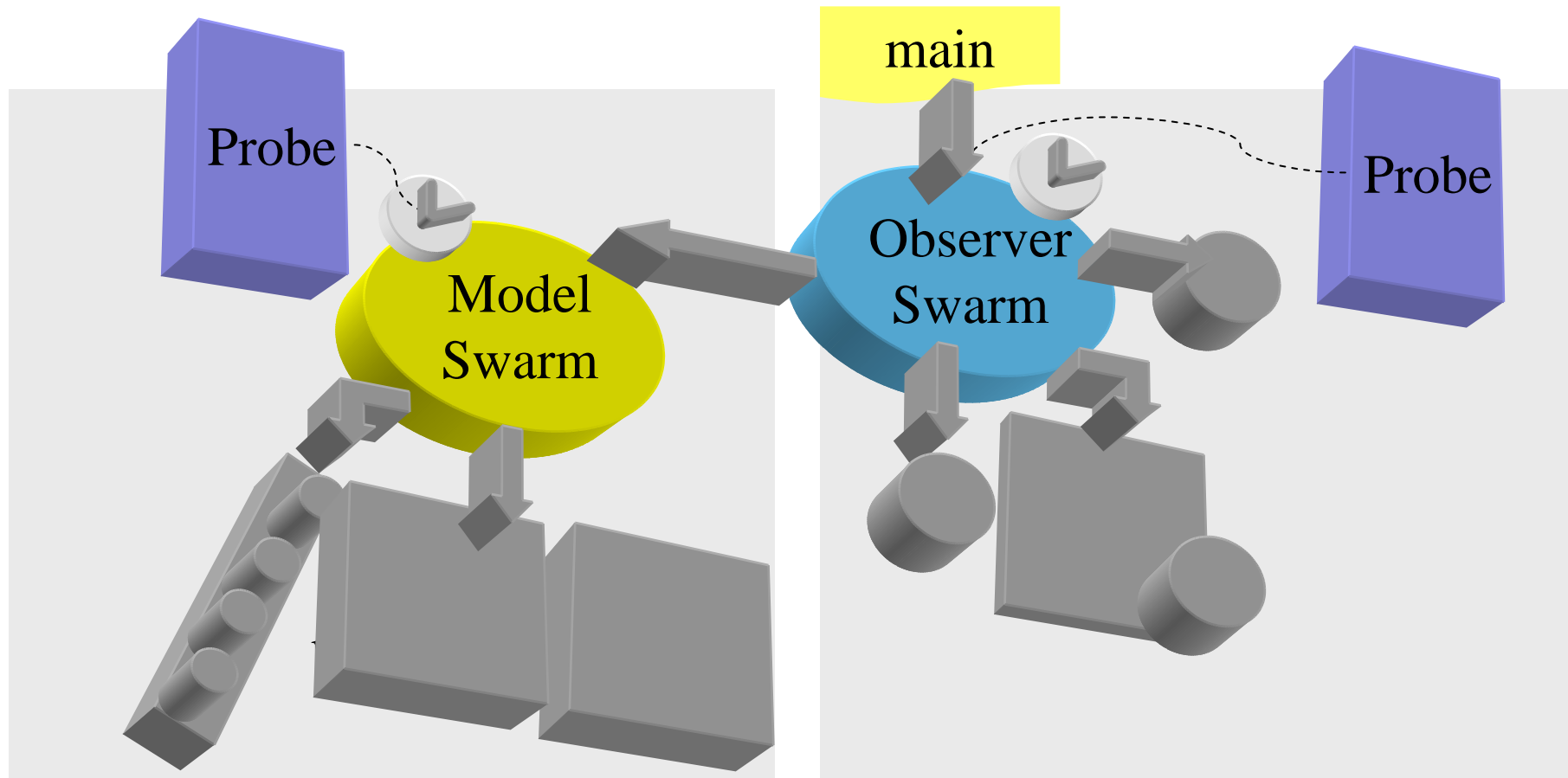
Managing the Raster display

- ZoomRaster displays data from lattice
- We will display the food distribution from FoodSpace and ask bugs to draw location
- In addition to ZoomRaster need the 3 classes to the right:
 - ColorMap:
 - Associates a number with a color in palette
 - Value2dDisplay:
 - Maps array of x,y int data to raster
 - Object2dDisplay:
 - Feeds data from agents & captures mouseclick

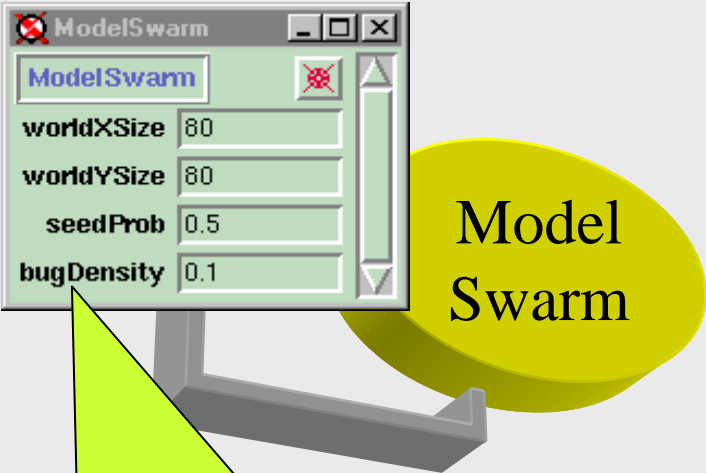
What do these new objects do?



Bug with Observer II: Add probes

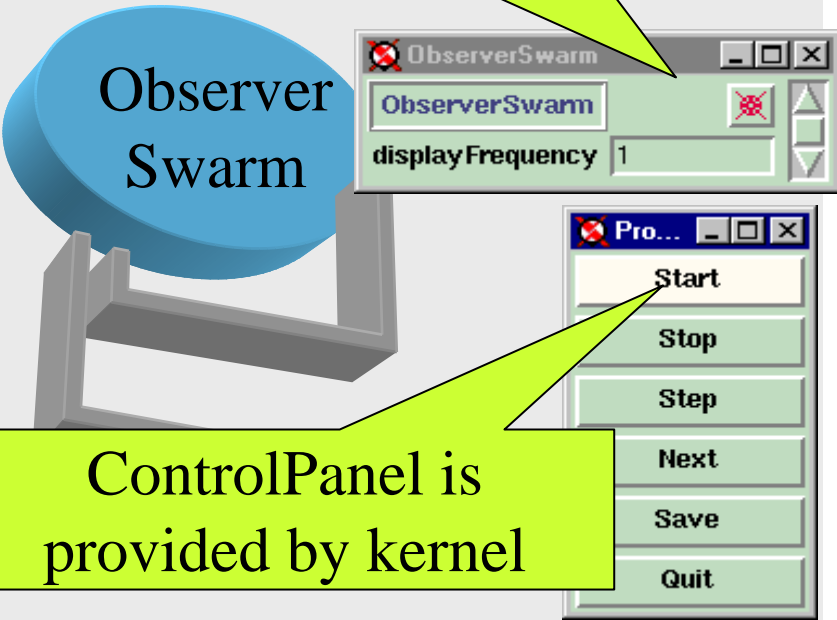


How the probes come in...



Model Swarm

A custom ProbeMap with 4 VariableProbes



Observer Swarm

A custom ProbeMap with 1 MessageProbes

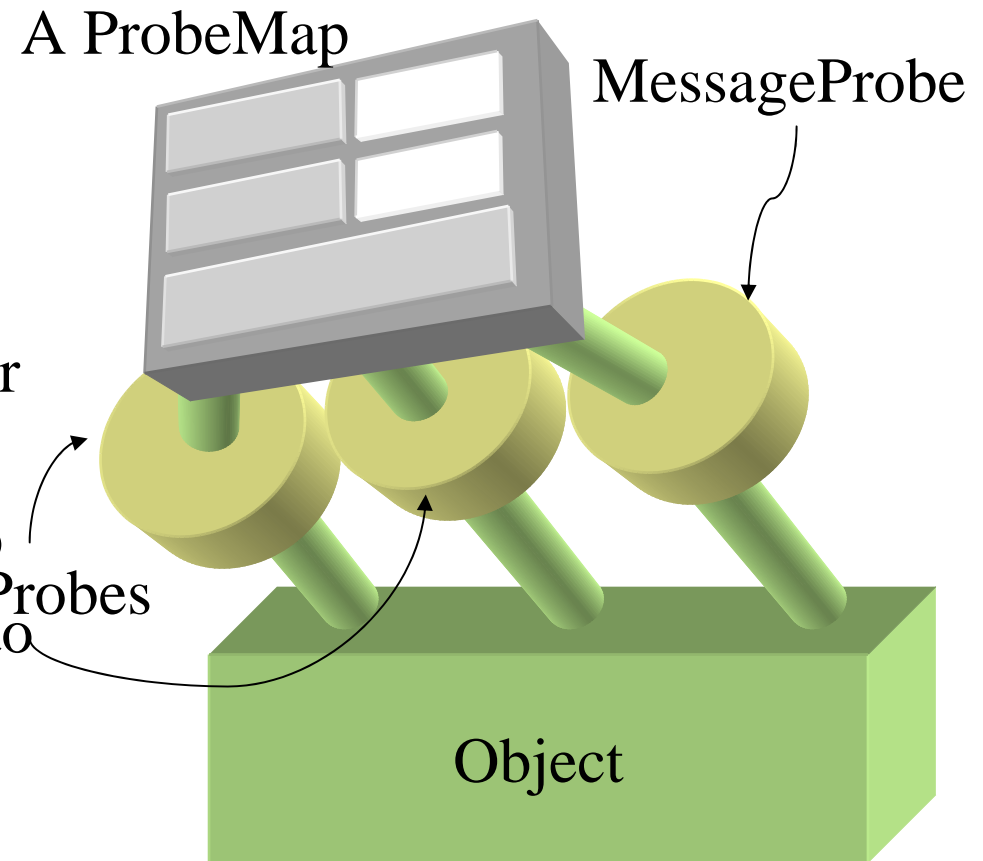
ControlPanel is provided by kernel

Brief overview of probes

- Two major uses for probes
 - To interface with an object
 - To create a GUI to an object
- Interface with an object of two types
 - VarProbe: Probes an instance variable
 - MessageProbe: Probes a method
- GUI utilities:
 - ProbeMap: Collection of Var and MessageProbes

Creating graphic probe to object

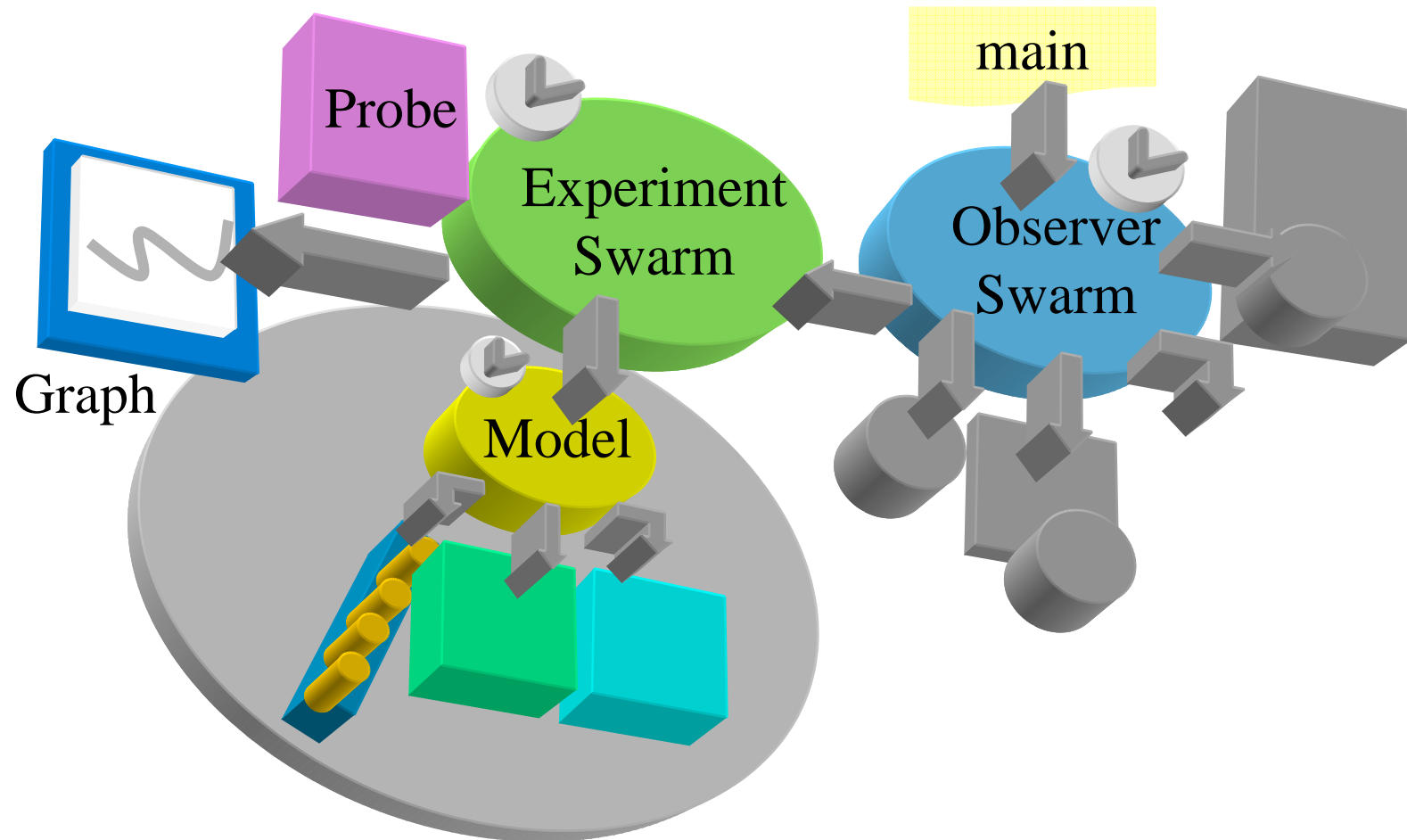
- 1 Check out instance of EmptyProbeMap
- 2 Attach VarProbe or MessageProbe to each variable or message to appear on GUI
- 3 Put each probe on ProbeMap
 - Ask probeDisplayManager to
- 4 create actual widget



Creating ModelSwarm's probe

- 1 `EmptyProbeMap probeMap=new
EmptyProbeMapImpl(aZone,this.getClass());`
- 2 `probeMap.addProbe(Globals.env.probeLibrary.
getProbeForVariable$inClass
("worldXSize",this.getClass()));`
- 3 `Globals.env.probeLibrary.setProbeMap$For
(probeMap,this.getClass());`
- 4 (then call `Globals.env.createArchivedProbeDisplay
(modelSwarm,"modelSwarm") in Observer`)

Final frontier: Experiment Bug



Multilevel activation

In main: `observerSwarm.activateIn(null);`

ObserverSwarm

`activateIn(Swarm context)`

`schedule.activateIn(this);`
`experimentSwarm.activateIn(this);`

`activateIn(Swarm context)`

`schedule.activateIn(this);`
`modelSwarm.activateIn(this);`

ExperSwarm

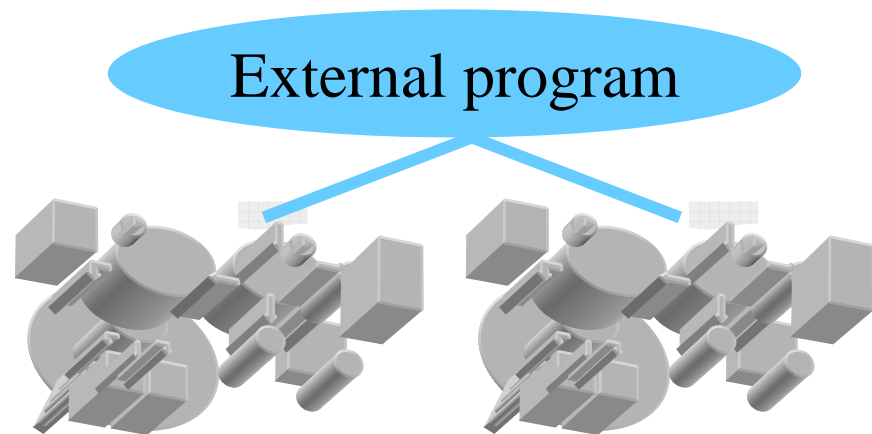
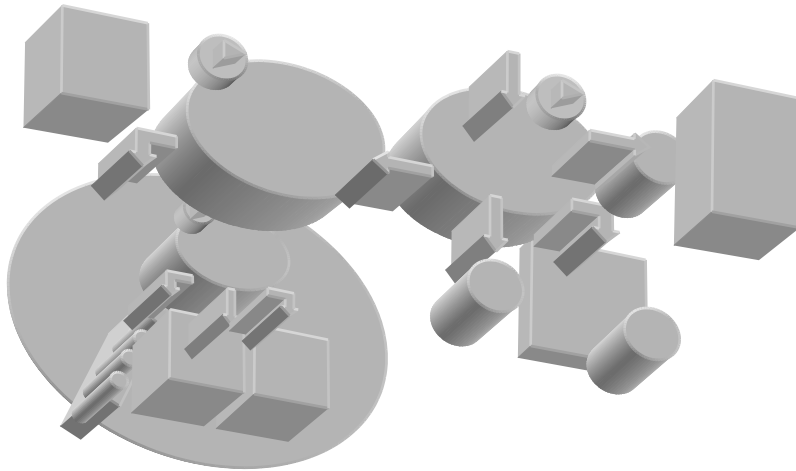
`activateIn(Swarm context)`

ModelSwarm

`Schedule.activateIn(this)`

Two approaches to experiments

- Use Swarm to control experiment runs
- Use external program to execute Swarms



Two approaches: Pros and cons

- Use Swarm to control experiment runs
- Pros:
 - All processing is done within Swarm framework
 - Can use GUI interactively
- Cons
 - Sequential - can't parallelize in present form
- Use external program to execute Swarms
- Pros:
 - Can do “poor mans” parallelization on multiprocessor machines
- Cons:
 - Requires knowledge of tools outside Swarm framework