There and back again

Buggle recursion

CS111 Computer Programming

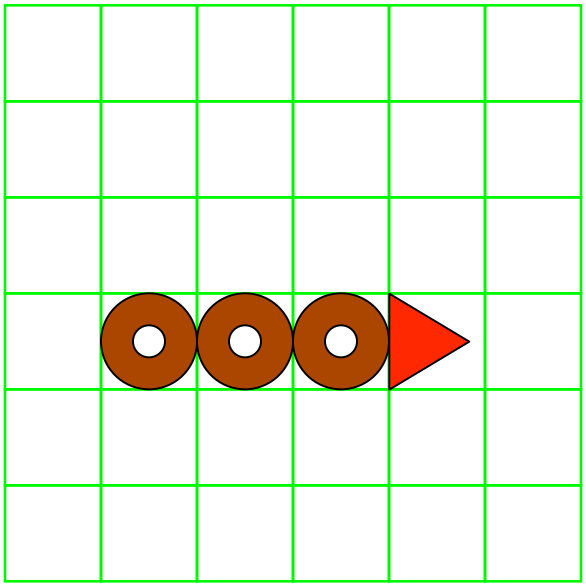
Department of Computer Science
Wellesley College

Recursion

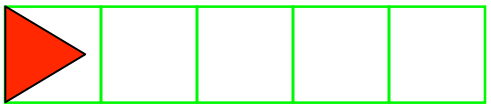
- Recursion solves a problem by solving a smaller instance of the same problem.
- Think divide, conquer, and glue when all the subproblems have the same "shape" as the original problem.

bagelForward(3)

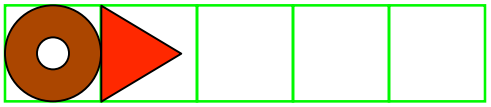
Write a method that teaches a buggle to leave behind a trail of bagels of a given length. Assume brushUp().



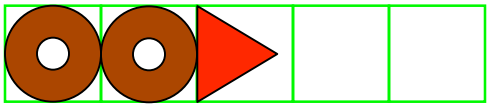
We could solve …



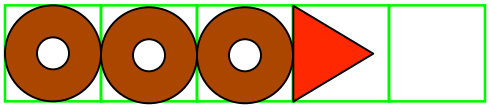
`bagelForward(3); by`
`dropBagel();`
`forward();` and solving …



`bagelForward(2); by`
`dropBagel();`
`forward();` and solving …



`bagelForward(1); by`
`dropBagel();`
`forward();` and solving …



`bagelForward(0); by`
Doing nothing at all

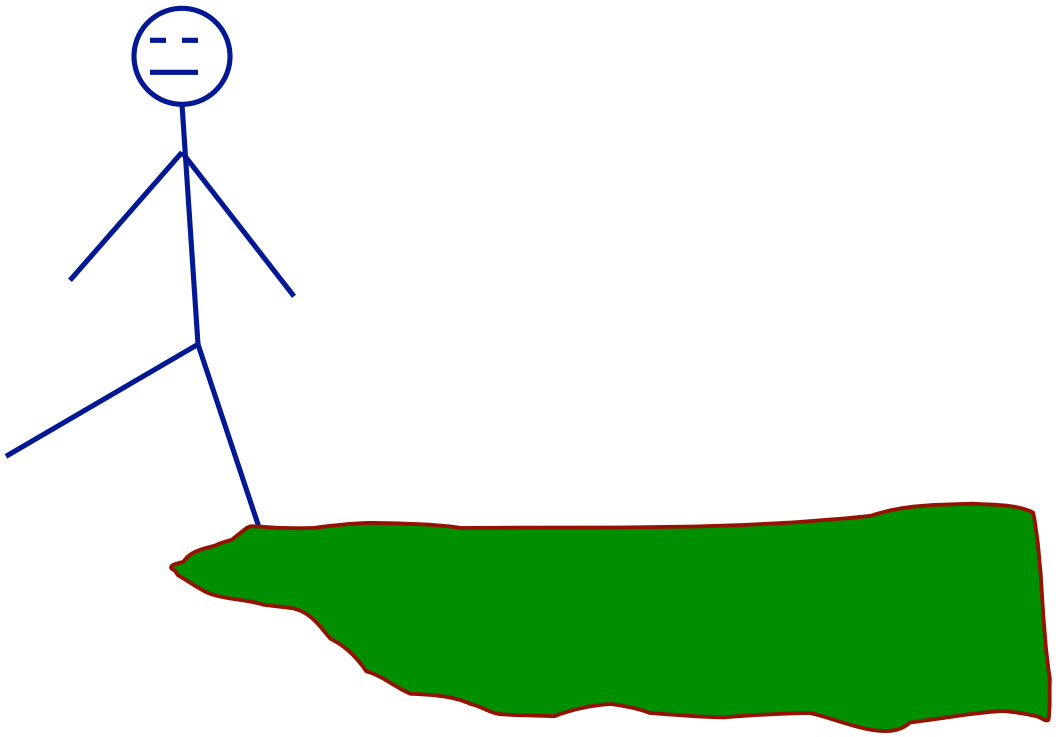
bagelForward(int n)

```
public void bagelForward(int n) {
  if (n == 0) {                          // base case
                                         // do nothing

  } else {                               // recursive case
    dropBagel();
    forward();
                                         // Now what?
  }
}
```

Recursive leap of faith

Assume we have a working method called **`bagelForward()`** and then use it …



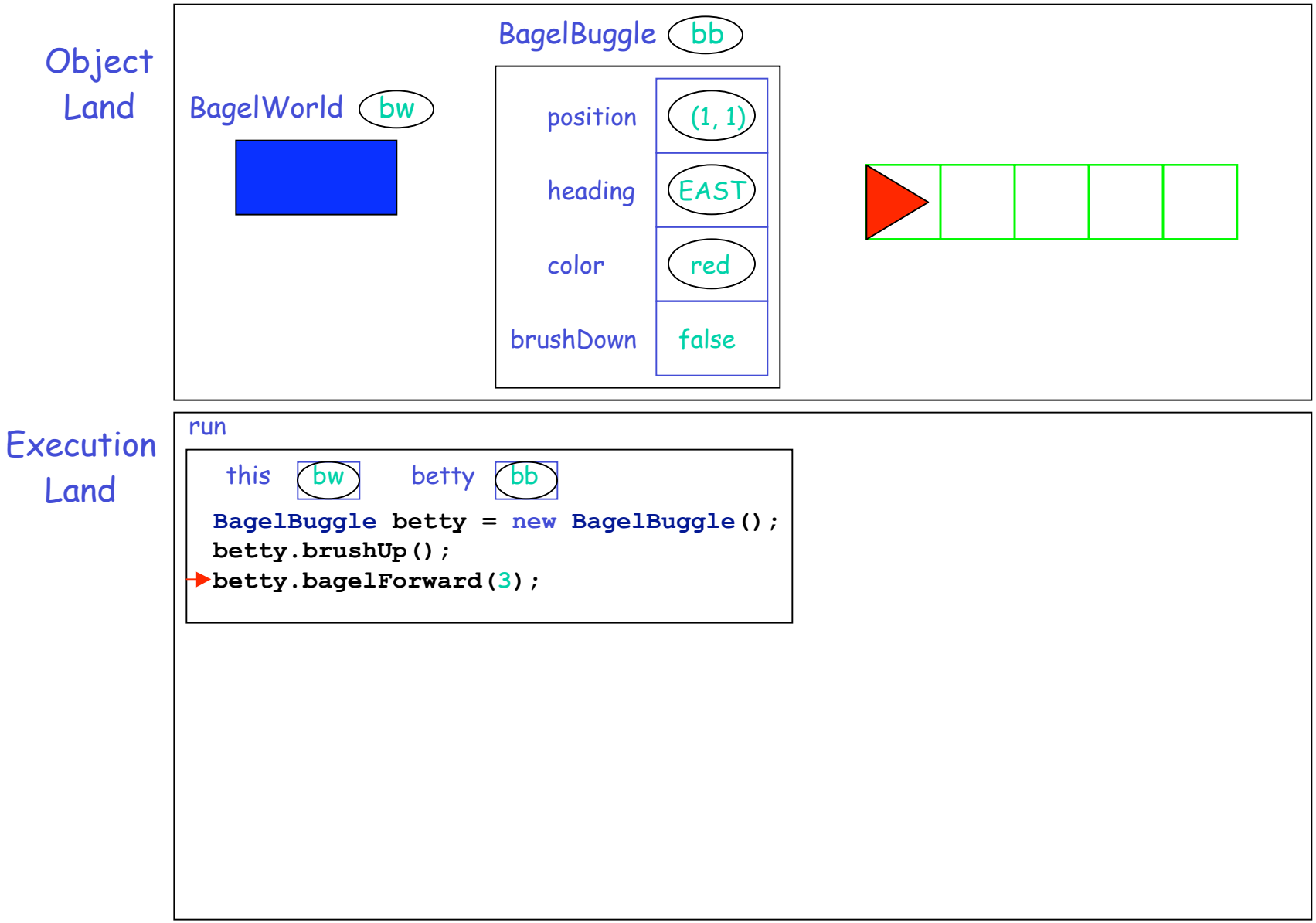
bagelForward(int n)

```
public void bagelForward(int n) {
  if (n == 0) {                          // base case
                                         // do nothing
  } else {                               // recursive case
    dropBagel();
    forward();
    bagelForward(n-1);                   // recursive invocation
  }
}
```

Putting a wrapper around it

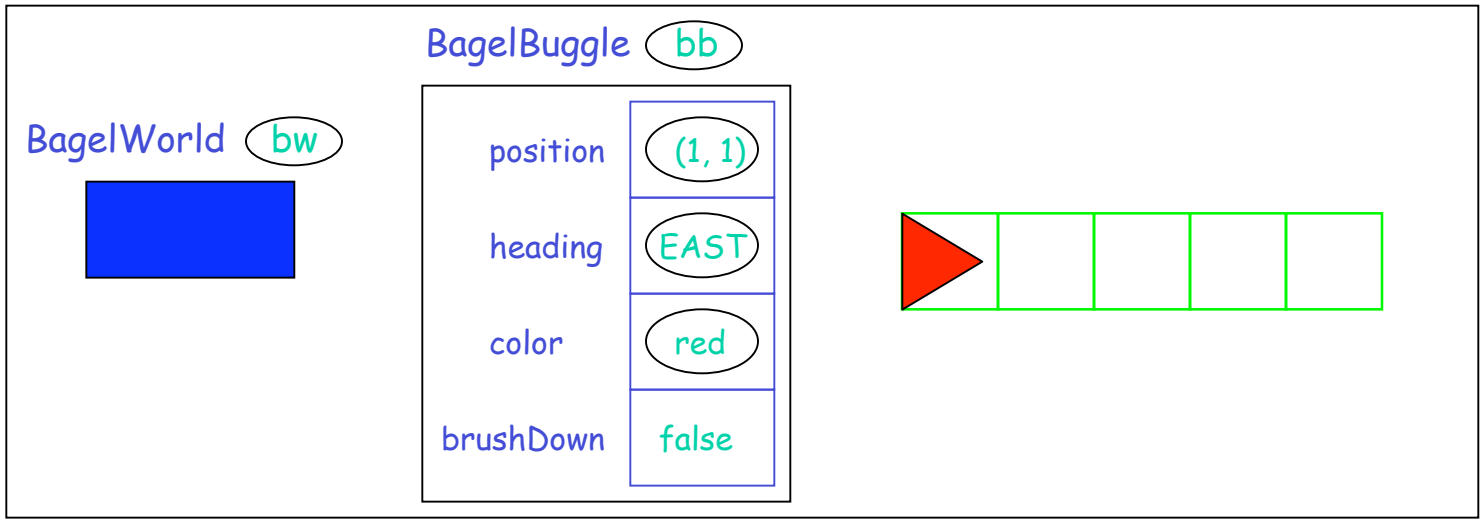
```
public class BagelWorld extends BuggleWorld {
  public void run() {
    BagelBuggle betty = new BagelBuggle();
    betty.brushUp();
    betty.bagelForward(3);
  }
}
class BagelBuggle extends Buggle {
  public void bagelForward(int n) {
    if (n == 0) {                          // base case
                                           // do nothing
    } else {                               // recursive case
      dropBagel();
      forward();
      bagelForward(n-1);                   // recursive invocation
    }
  }
}
```

Java execution model (joined in progress)



betty bagels forward

Object Land



Execution Land

run

this bw betty bb

```
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelForward(3);
```

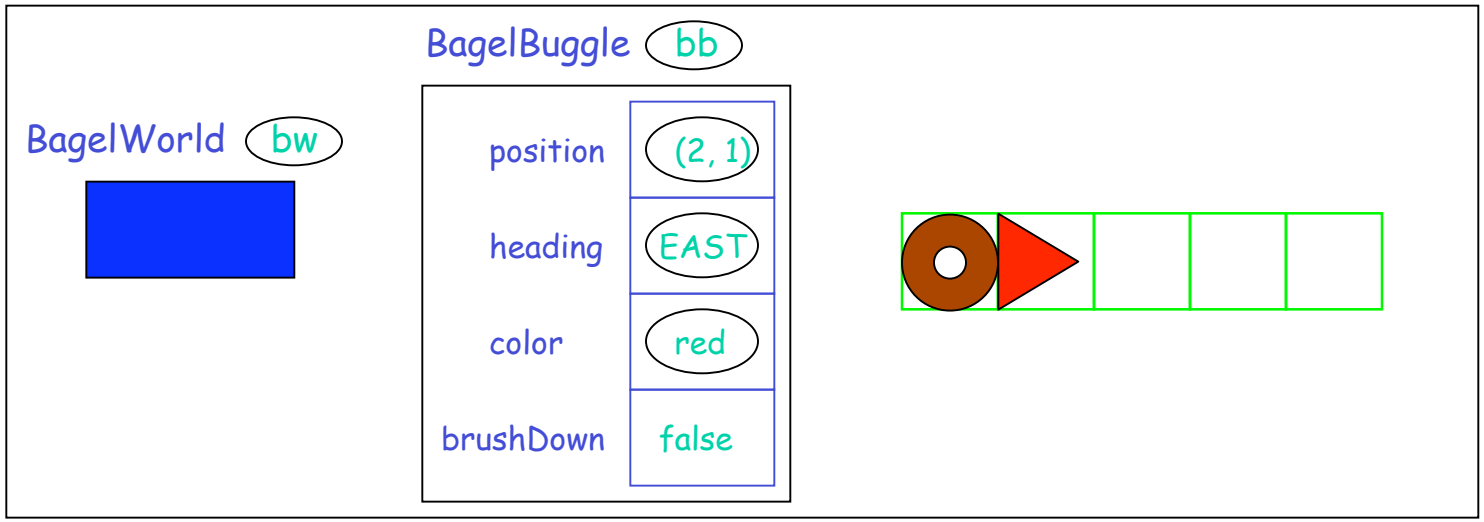
bagelForward(3)

this bb n 3

```
if (n==0) {
} else {
  dropBagel();
  forward();
  bagelForward(n-1);
}
```

Drop the bagel & step away from the wall

Object Land



Execution Land

run

this bw betty bb

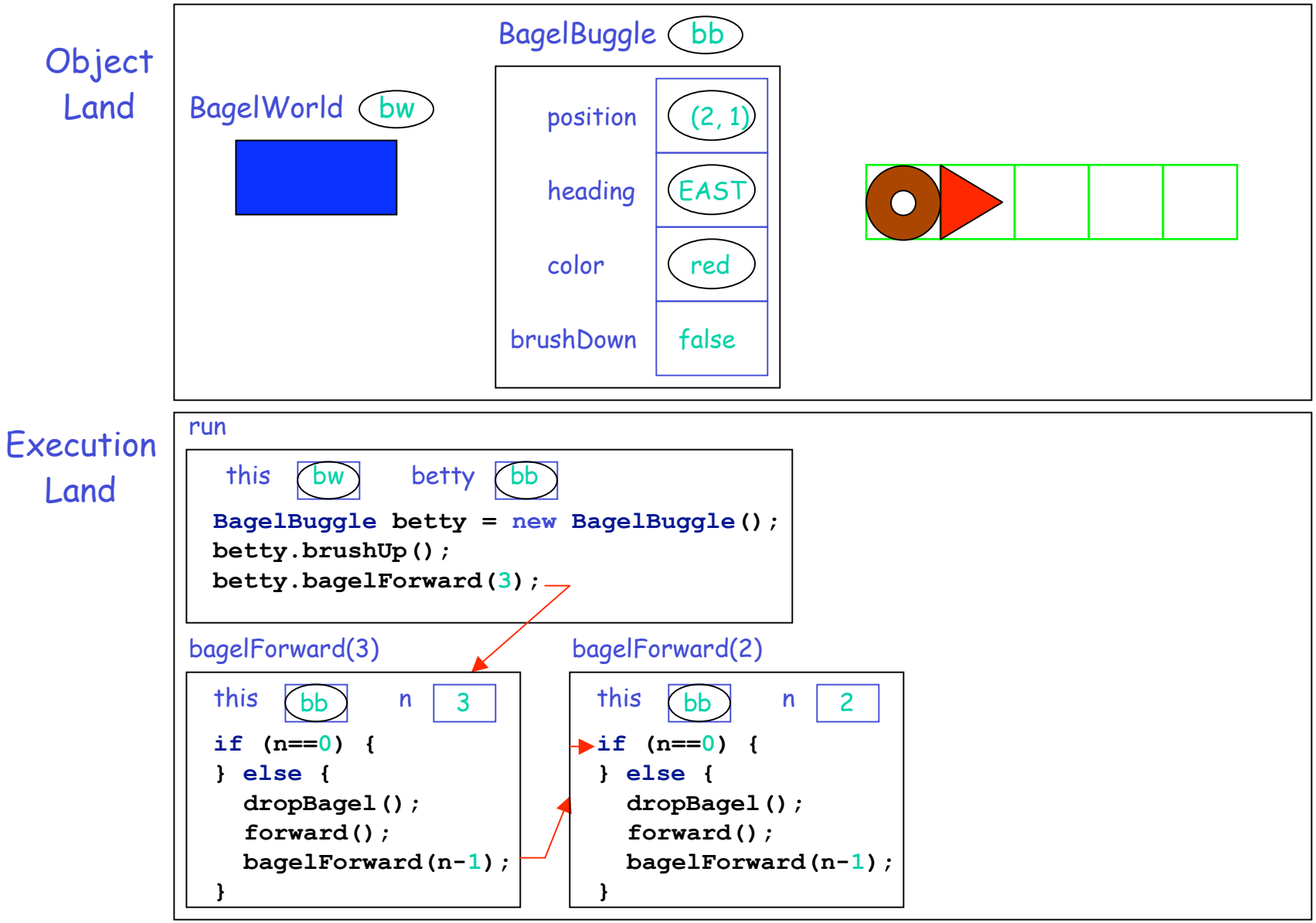
```
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelForward(3);
```

bagelForward(3)

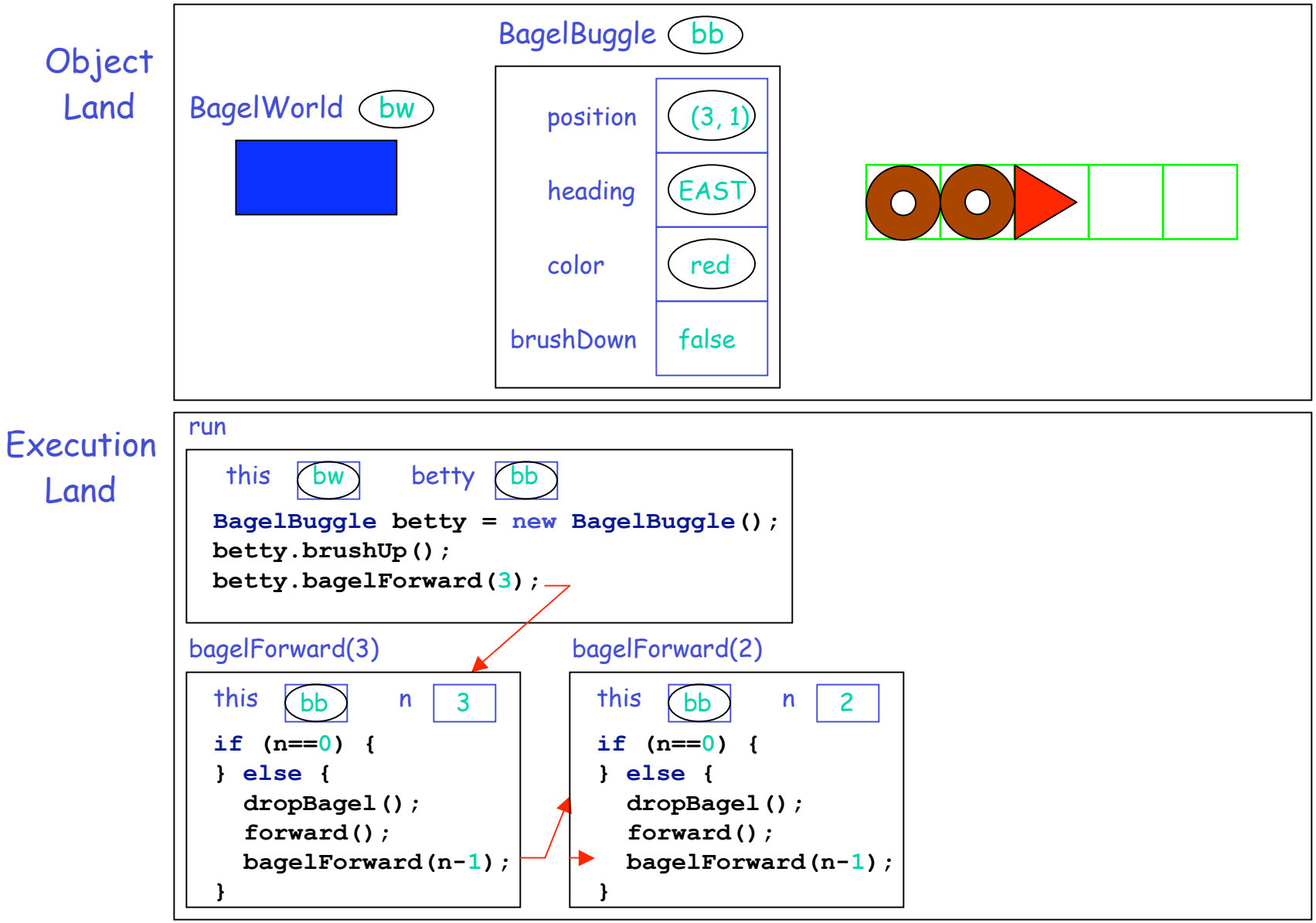
this bb n 3

```
if (n==0) {
} else {
  dropBagel();
  forward();
  bagelForward(n-1);
}
```

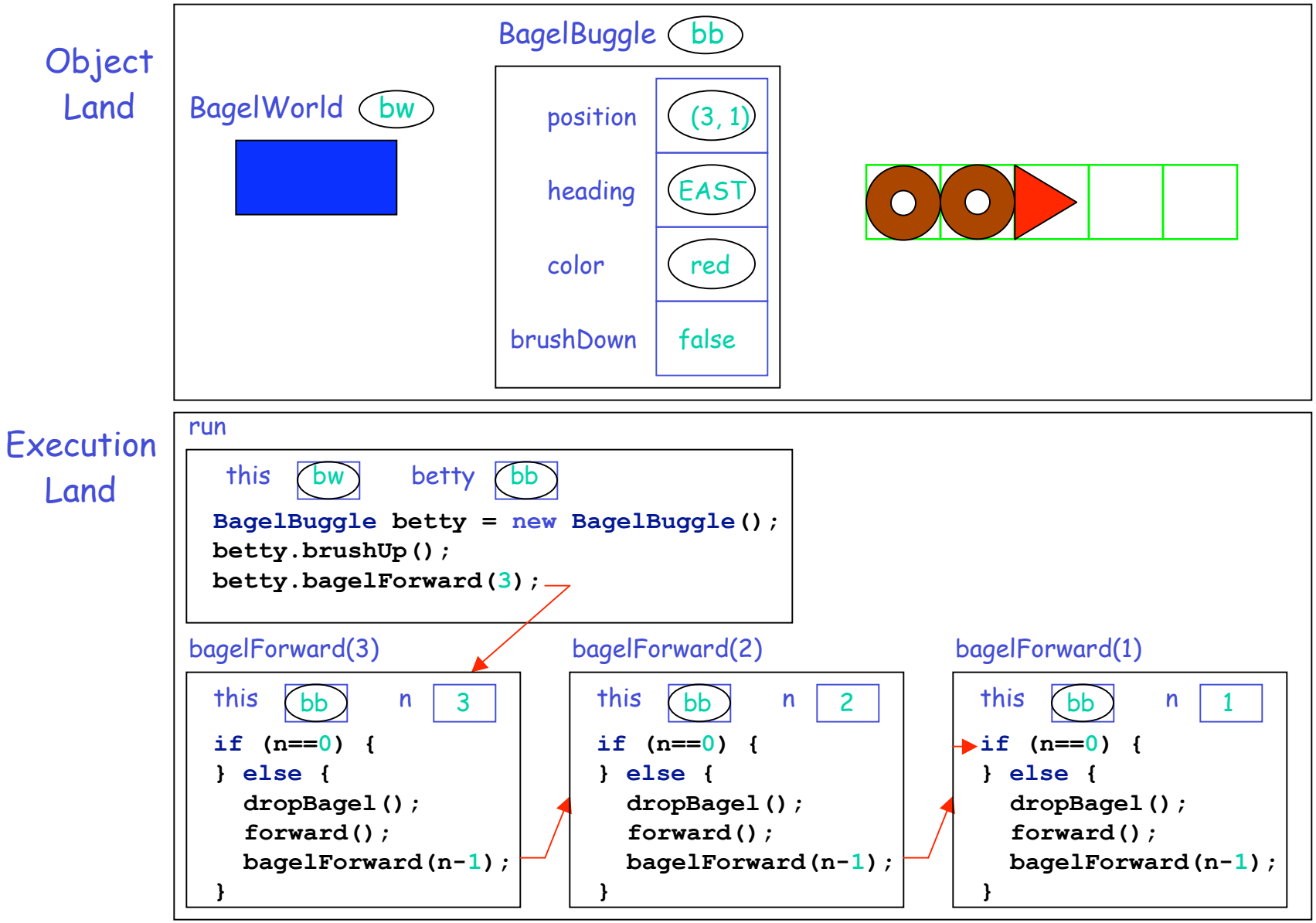
The spark of life moves on



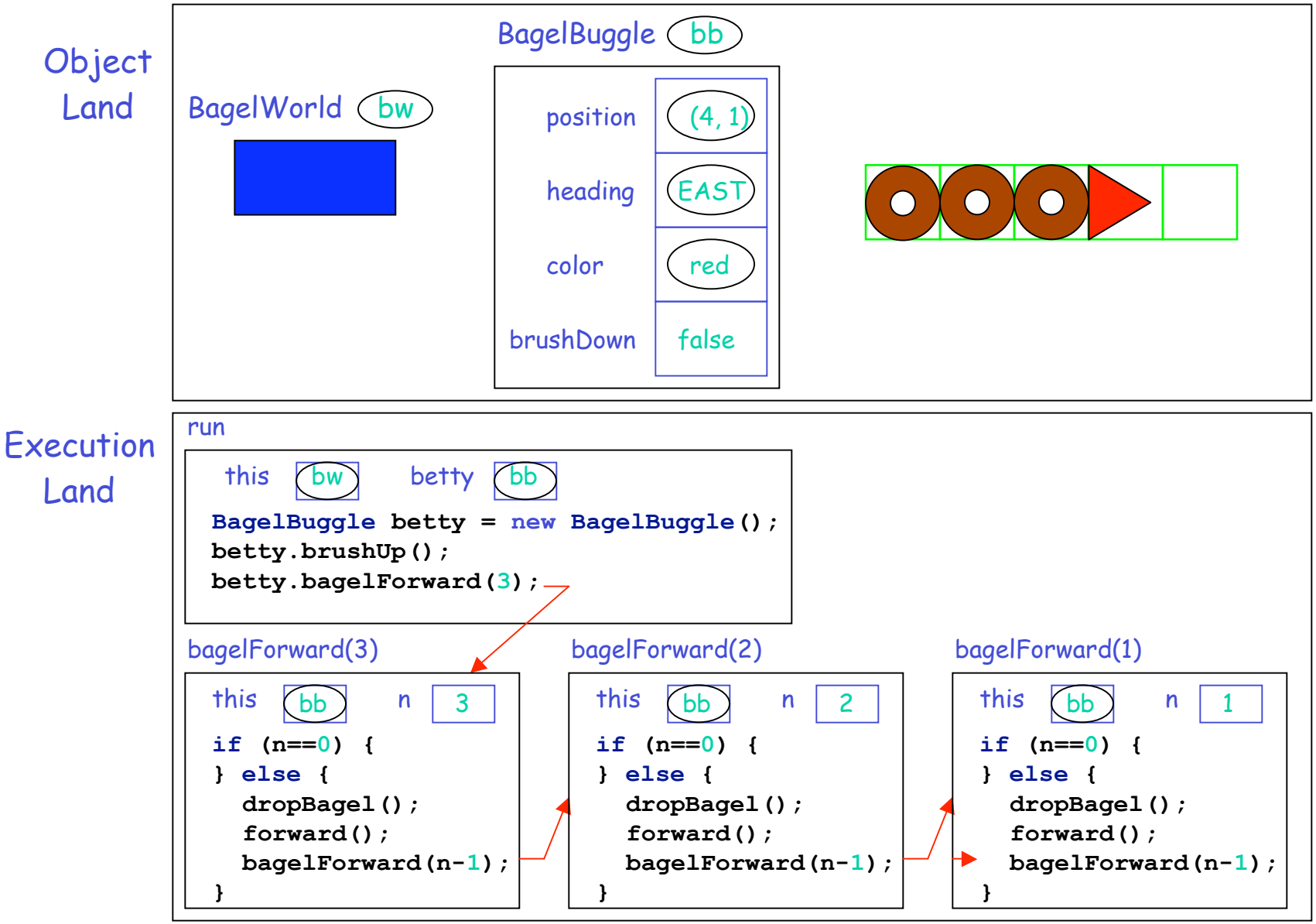
Another cell, another bagel



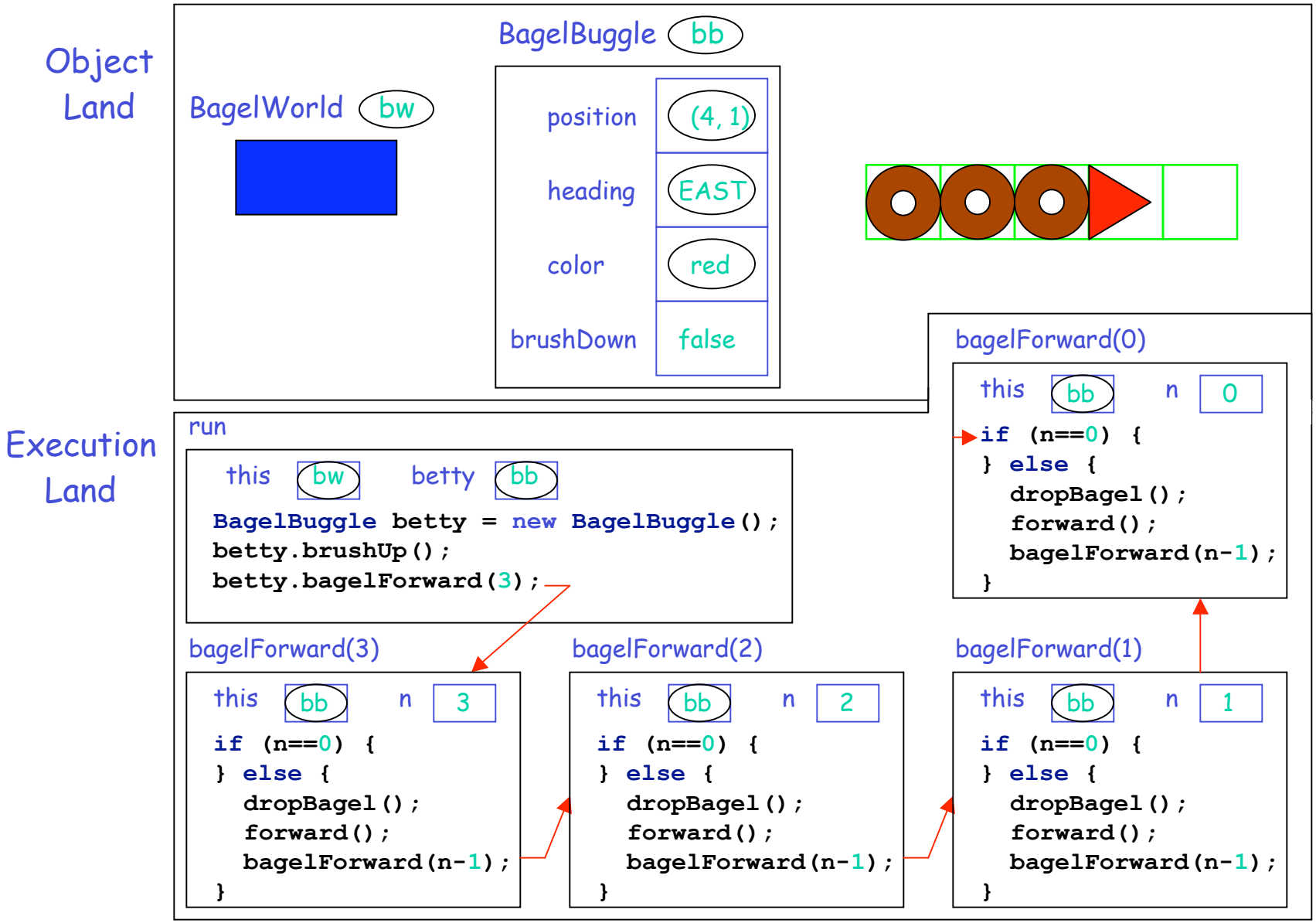
... and another invocation



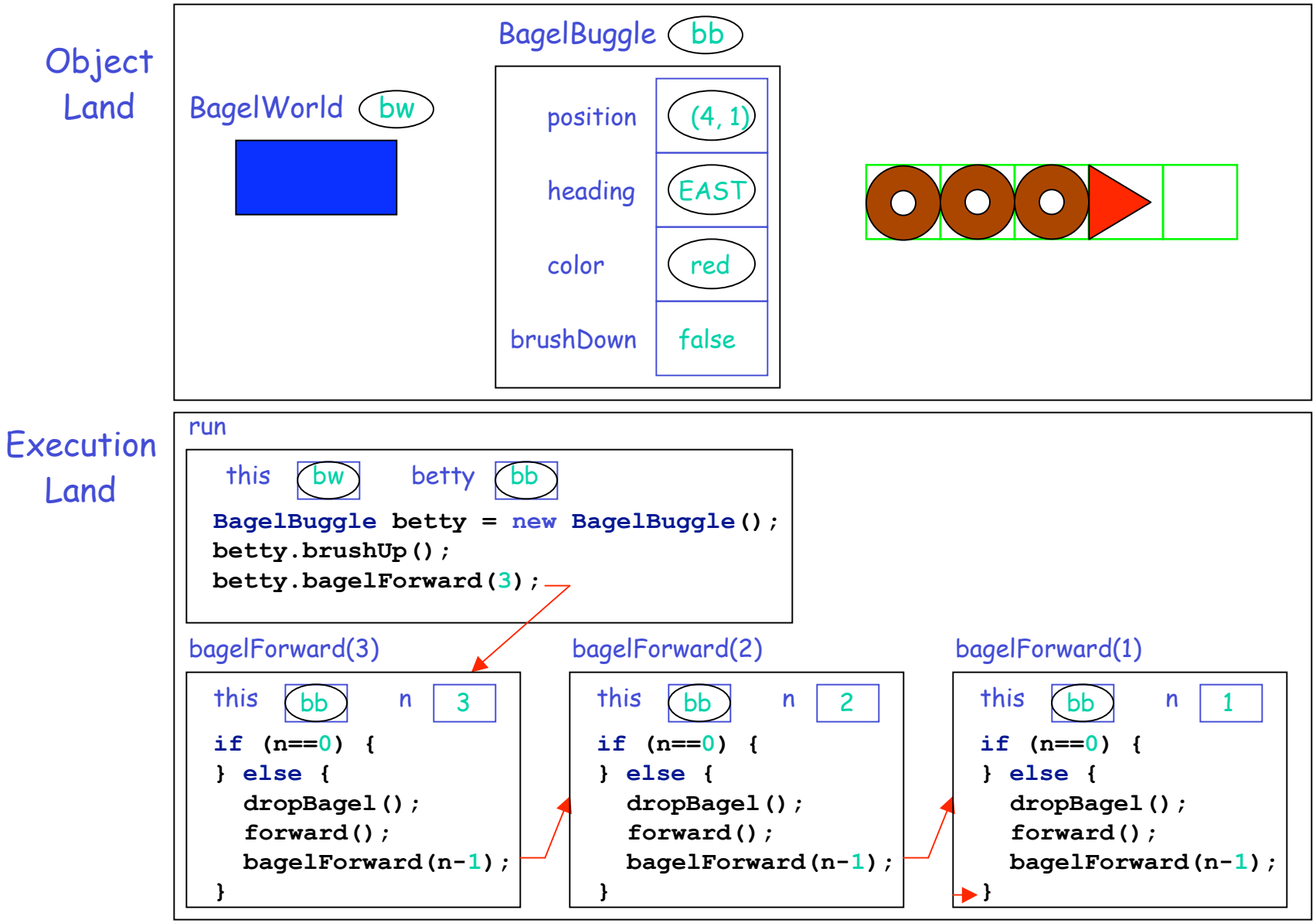
Bagels away


Object Land
BagelBuggle
bb
BagelWorld
bw
position
(4, 1)
heading
EAST
color
red
brushDown
false
Execution Land
run
this
bw
betty
bb
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelForward(3);
bagelForward(3)
this
bb
n
3
if (n==0) {
} else {
dropBagel();
forward();
bagelForward(n-1);
}
bagelForward(2)
this
bb
n
2
if (n==0) {
} else {
dropBagel();
forward();
bagelForward(n-1);
}
bagelForward(1)
this
bb
n
1
if (n==0) {
} else {
dropBagel();
forward();
bagelForward(n-1);
}

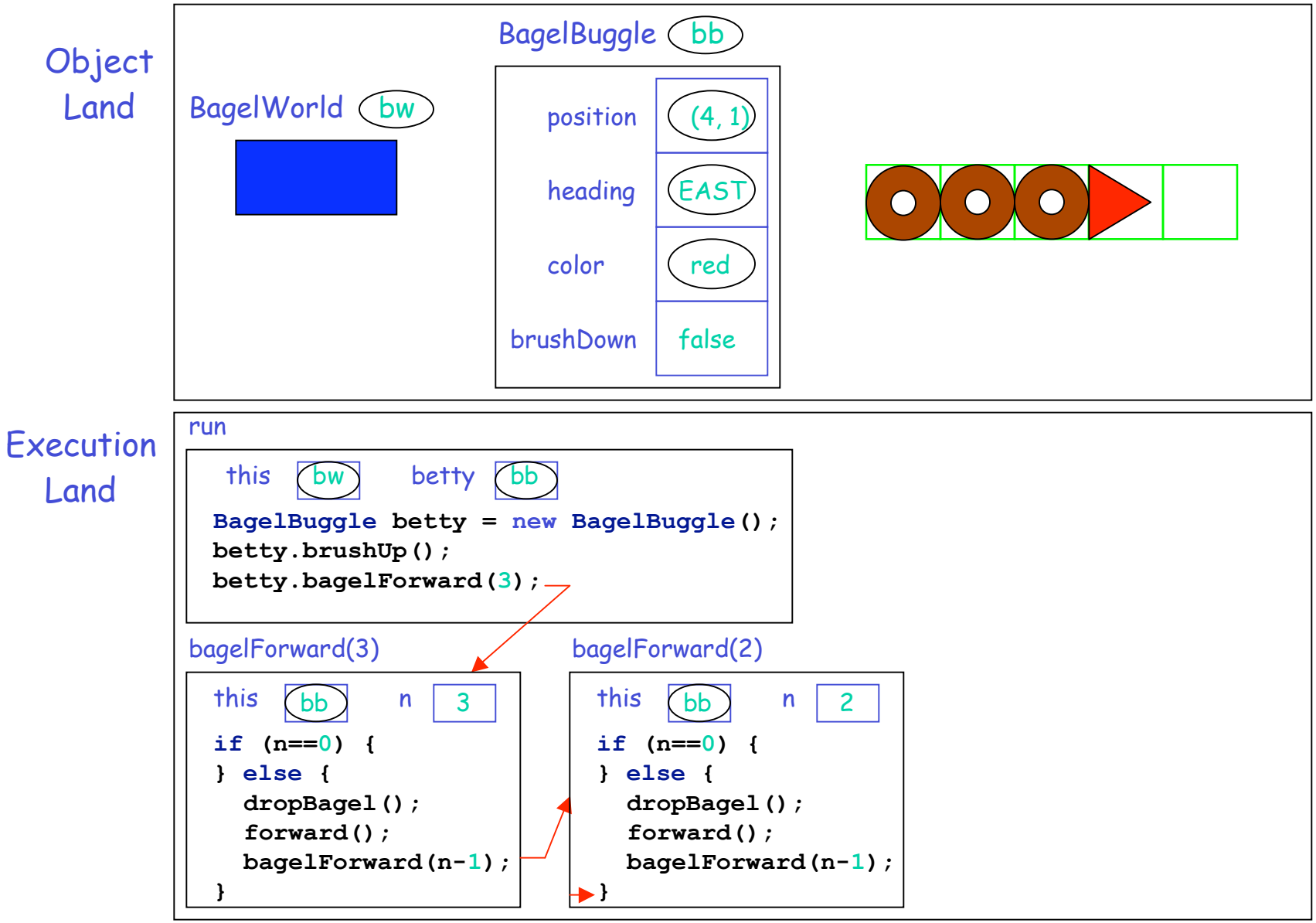
The base case



bagelForward(0) completes



bagelForward(1) completes



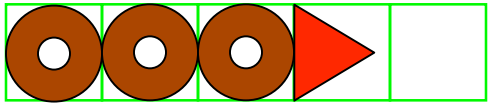
bagelForward(2) completes

Object Land

BagelBuggle bb

BagelWorld bw

position	(4, 1)
heading	EAST
color	red
brushDown	false



Execution Land

run

this bw betty bb

```
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelForward(3);
```

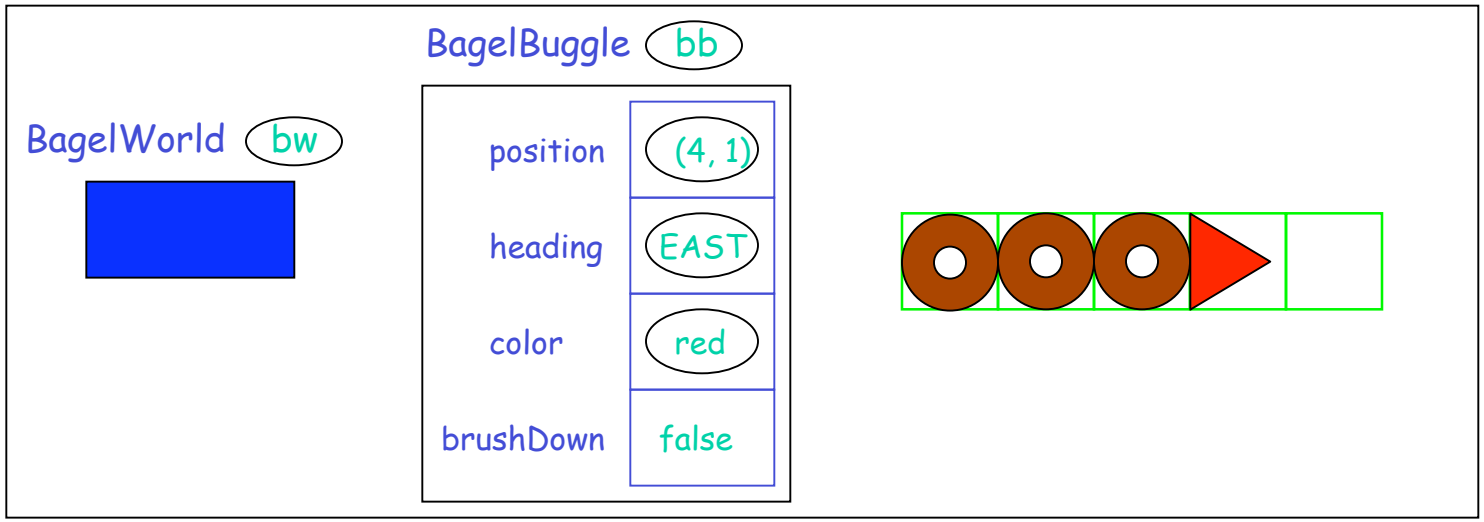
bagelForward(3)

this bb n 3

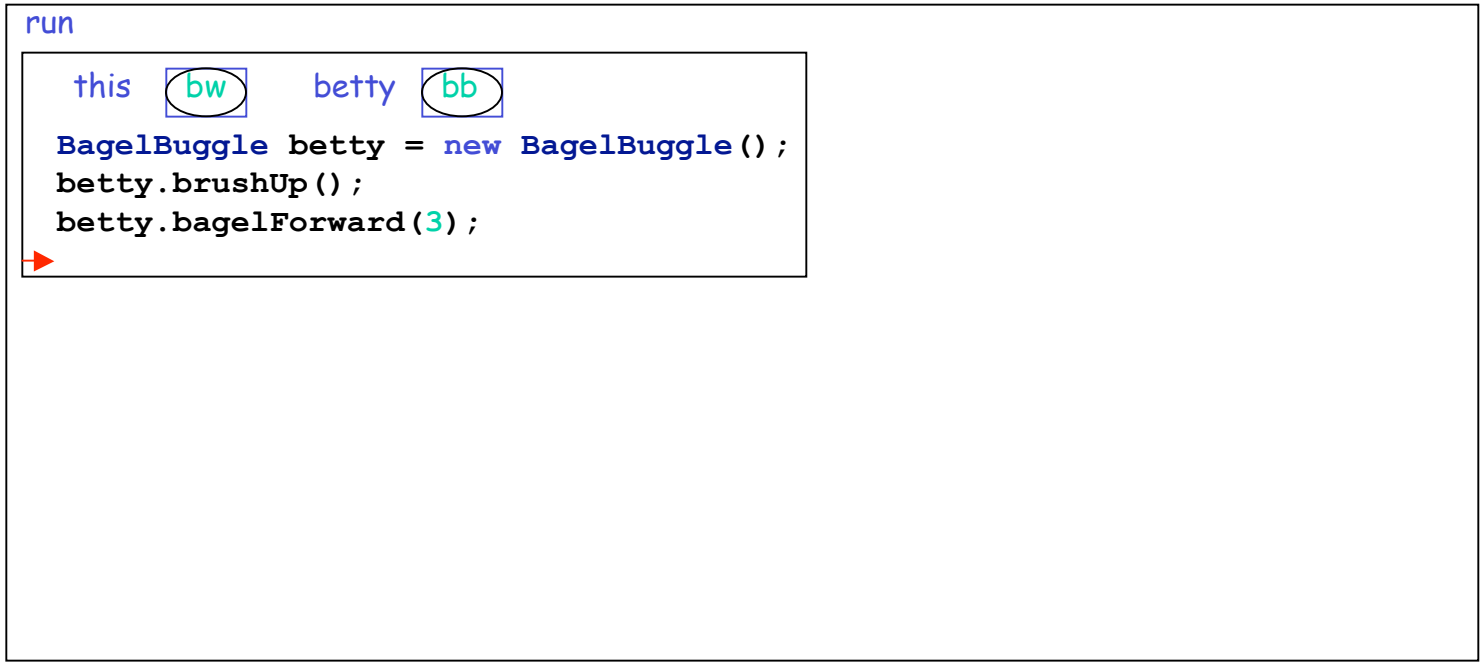
```
if (n==0) {
} else {
  dropBagel();
  forward();
  bagelForward(n-1);
}
```

betty.bagelForward(3) completes

Object Land

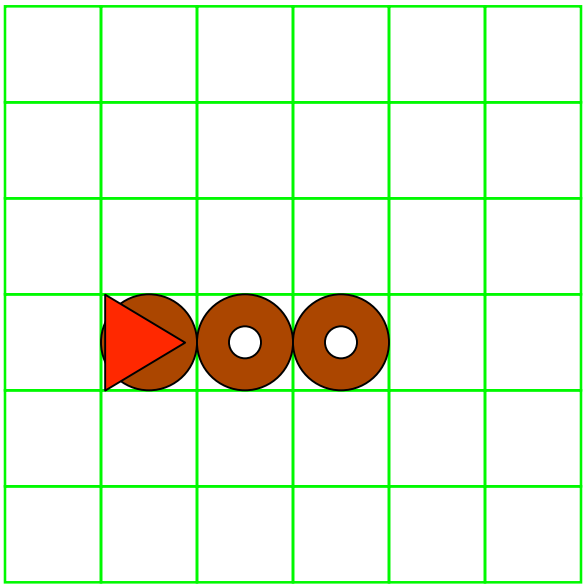


Execution Land

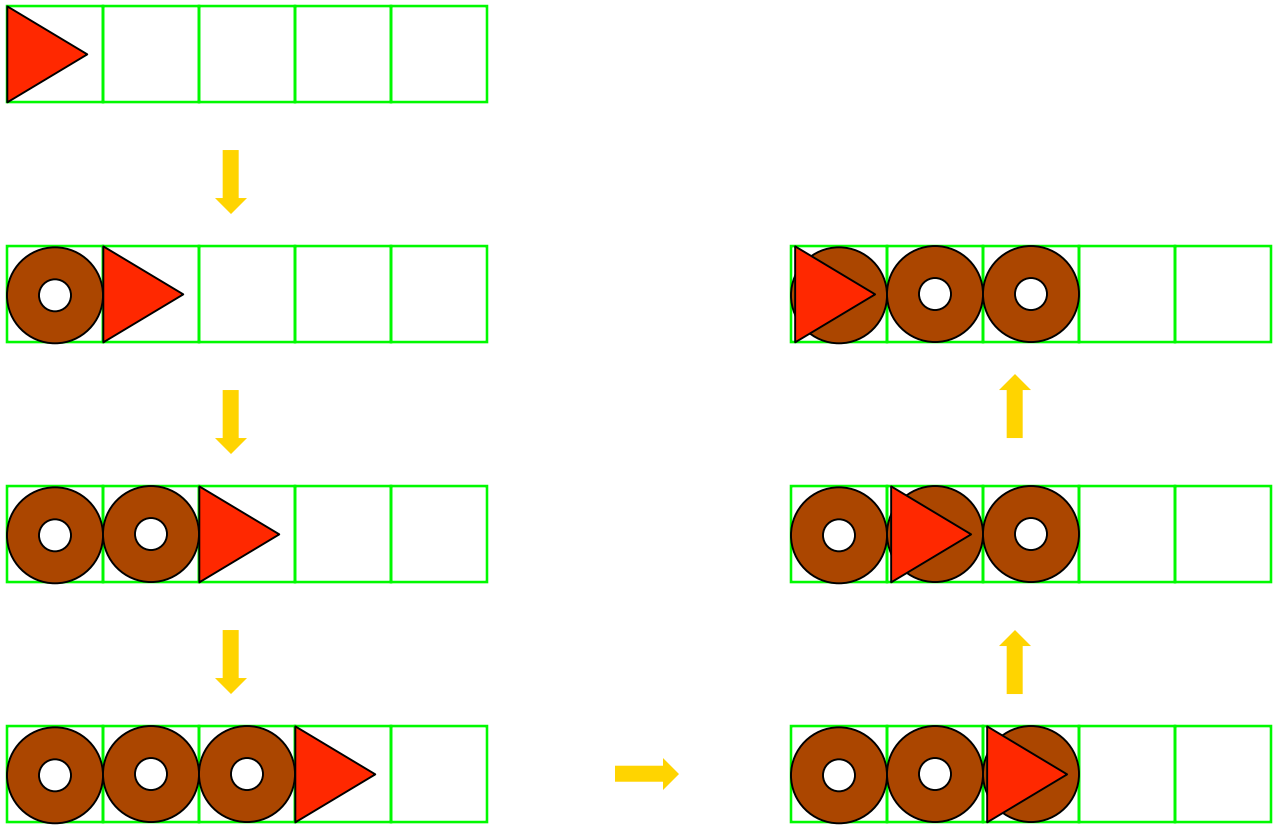


thereAndBack(3)

Write a method that teaches a buggle to leave behind a trail of bagels of a given length, then returns to starting point. Assume brushUp().



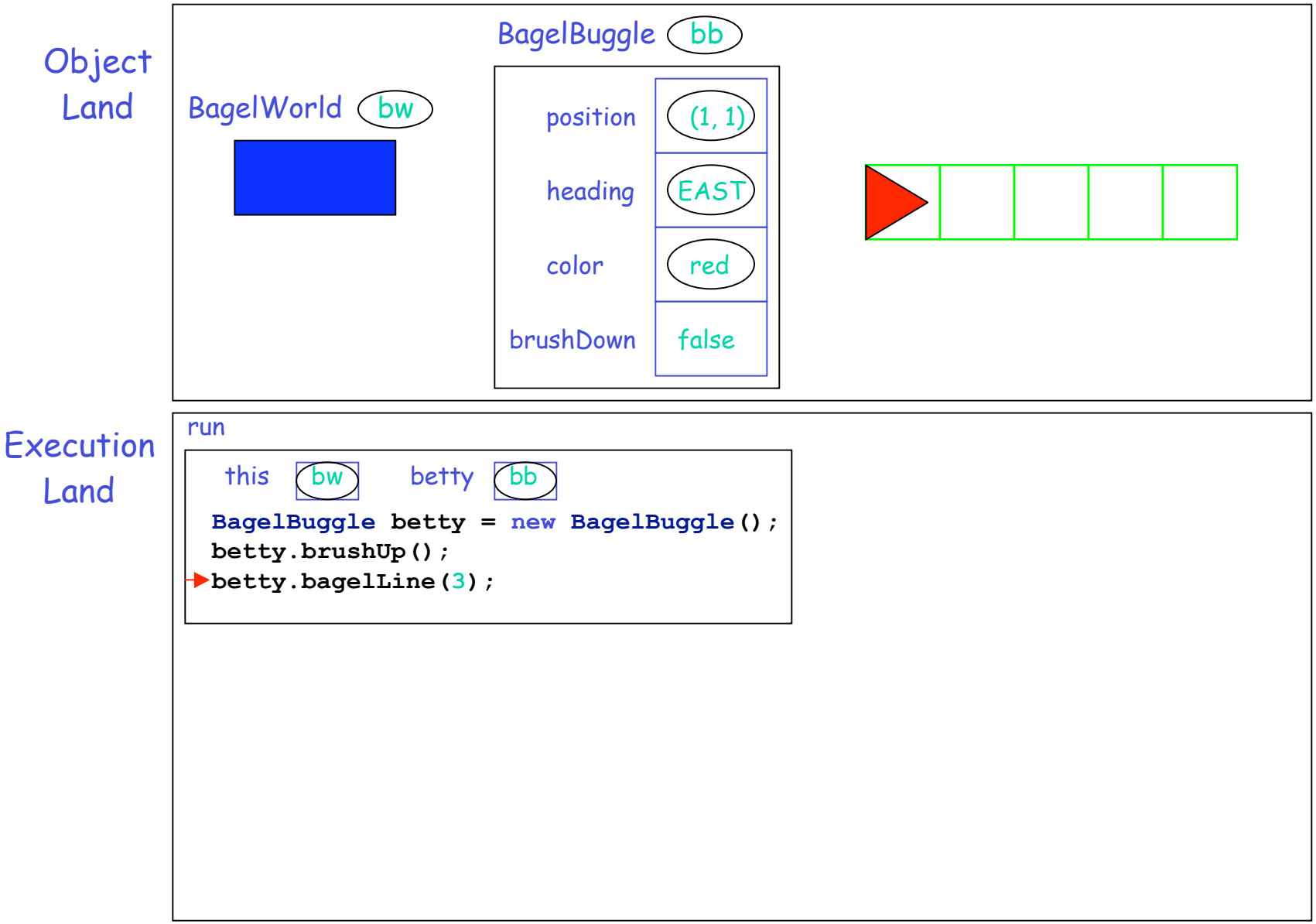
Roundtrip BagelBuggle



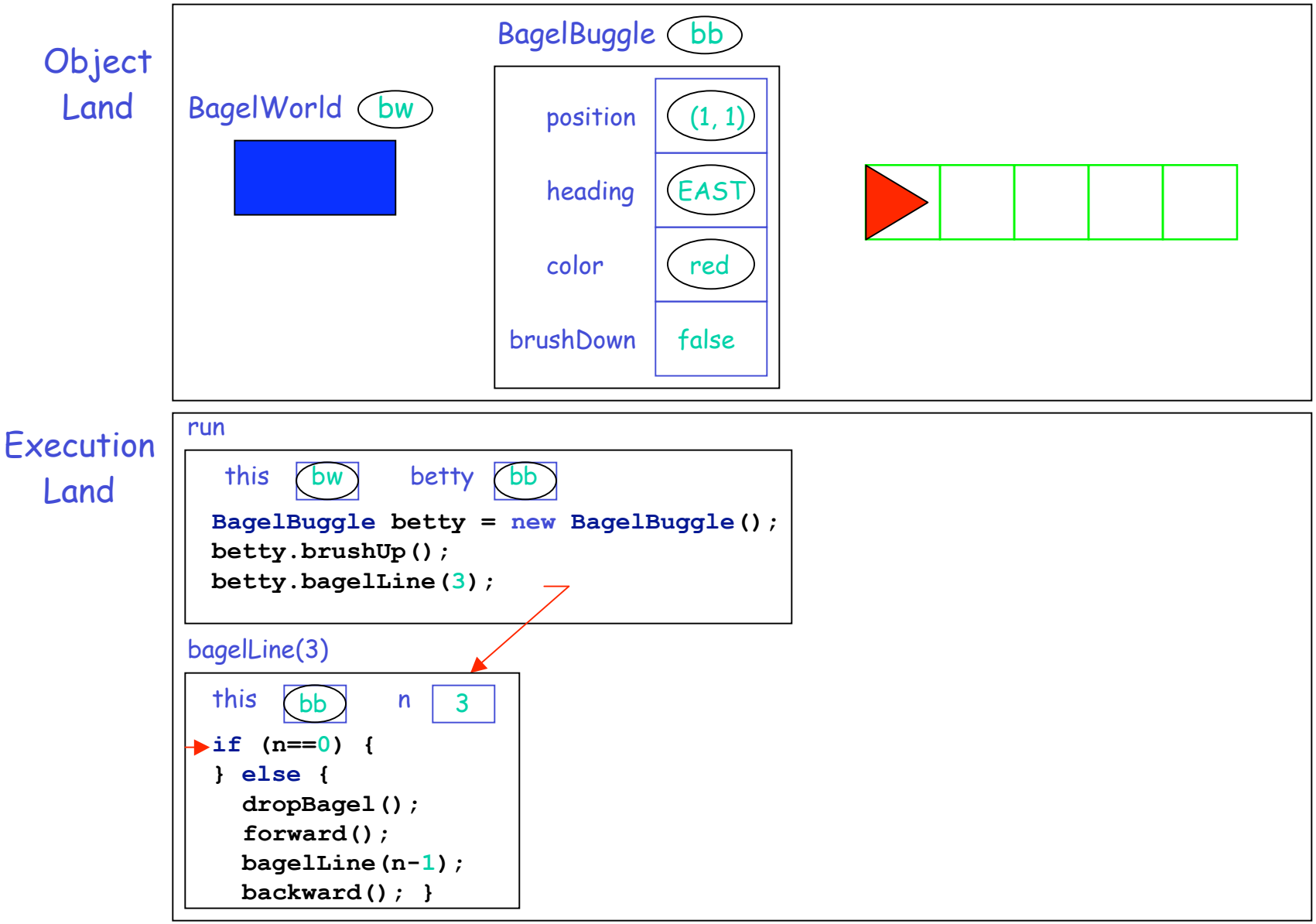
bagelLine(int n)

```
public void bagelLine(int n) {
  if (n == 0) {                          // base case
                                         // do nothing
  } else {                               // recursive case
    dropBagel();
    forward();
    bagelLine(n-1);                      // drop n-1 bagels
    backward();                          // and backup
  }
}
```

Java execution model (joined in progress)

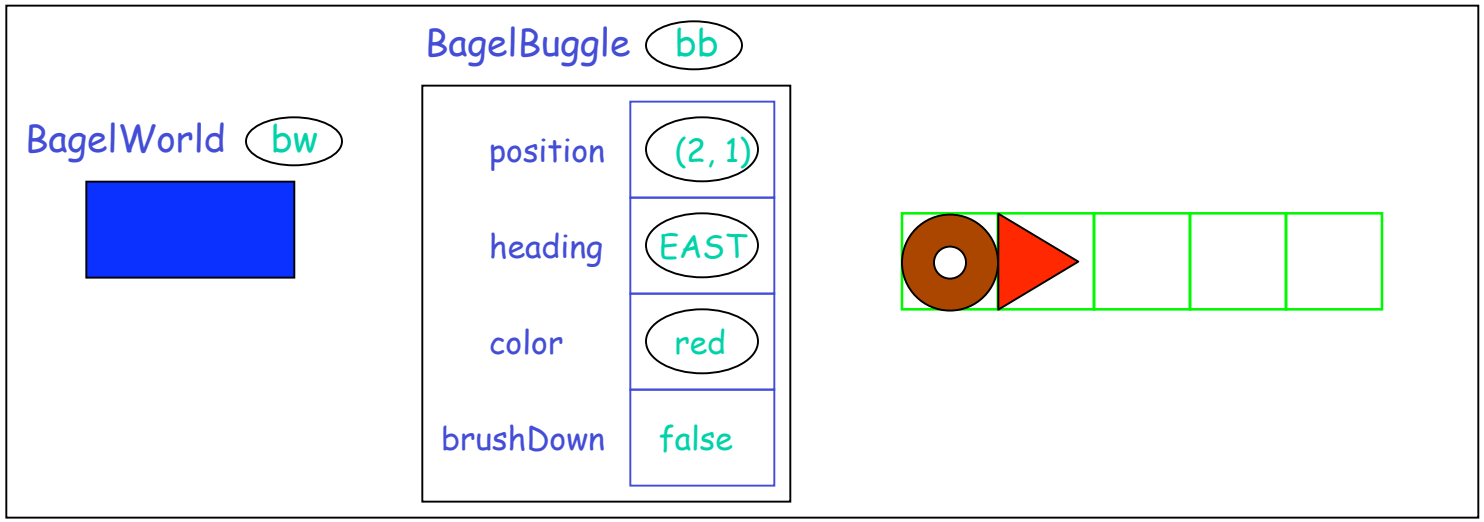


betty invokes bagelLine(3)



Dropping a bagel and moving forward

Object Land



Execution Land

run

this bw betty bb

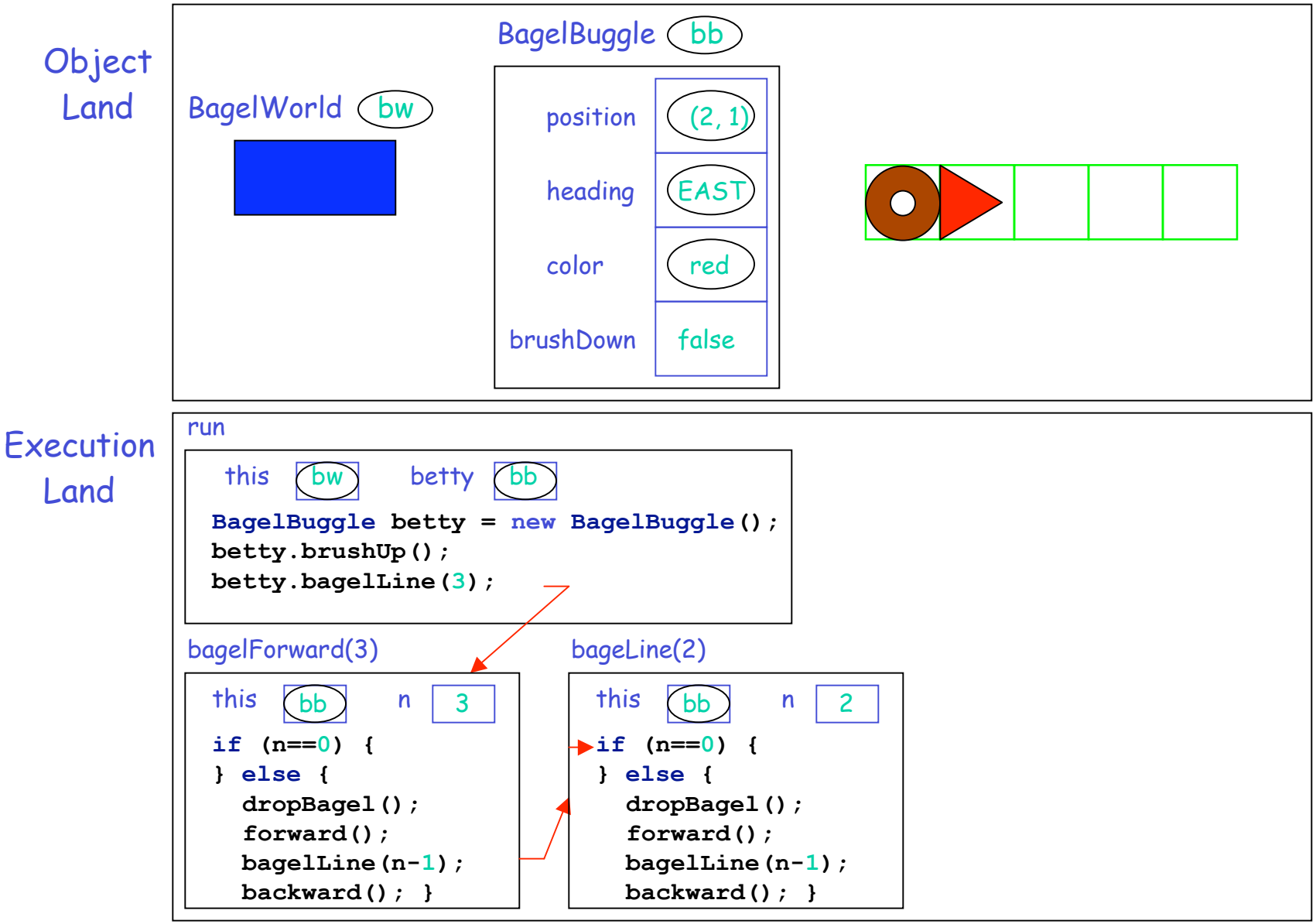
```
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelLine(3);
```

bagelForward(3)

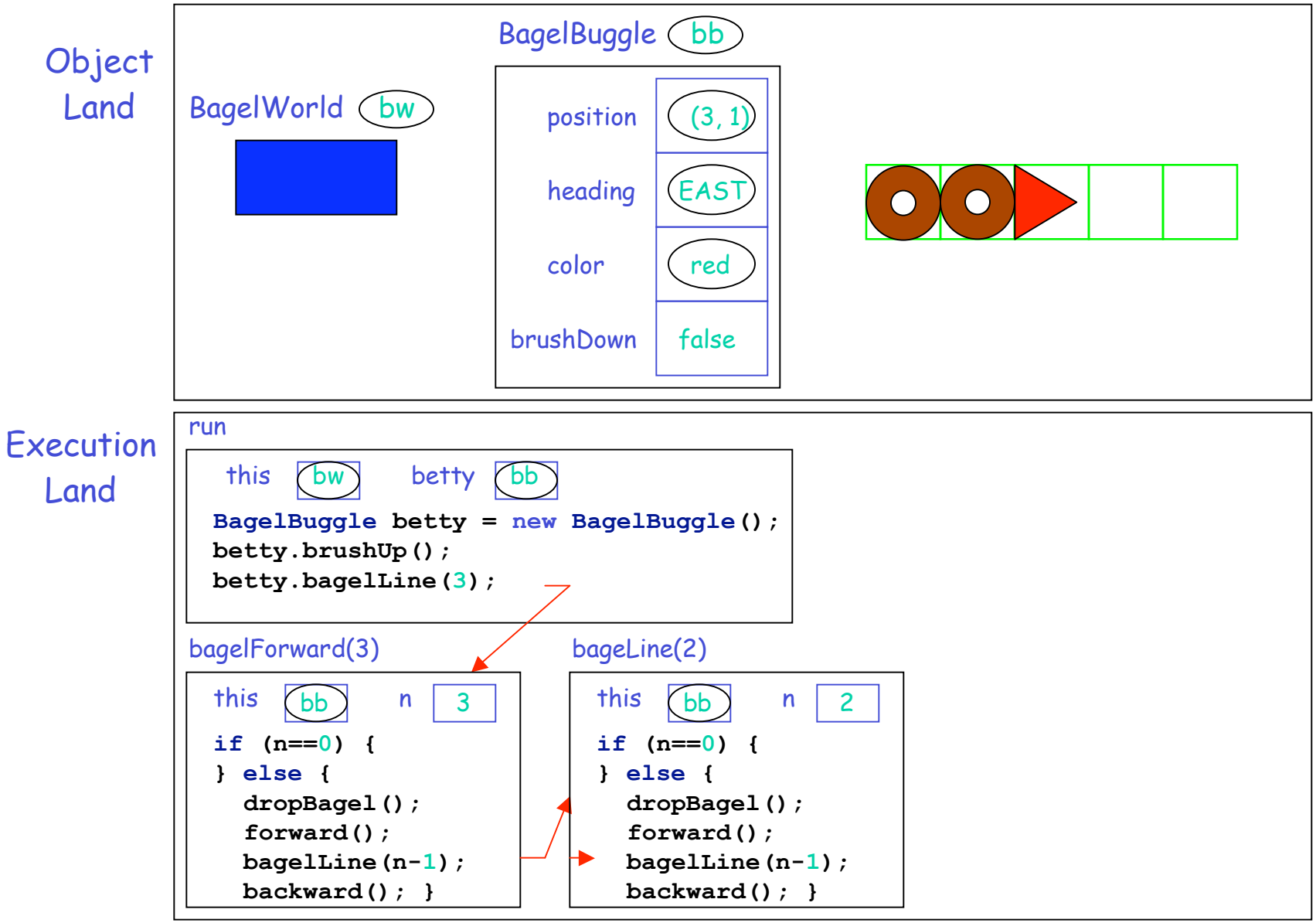
this bb n 3

```
if (n==0) {
} else {
  dropBagel();
  forward();
  bagelLine(n-1);
  backward(); }
```

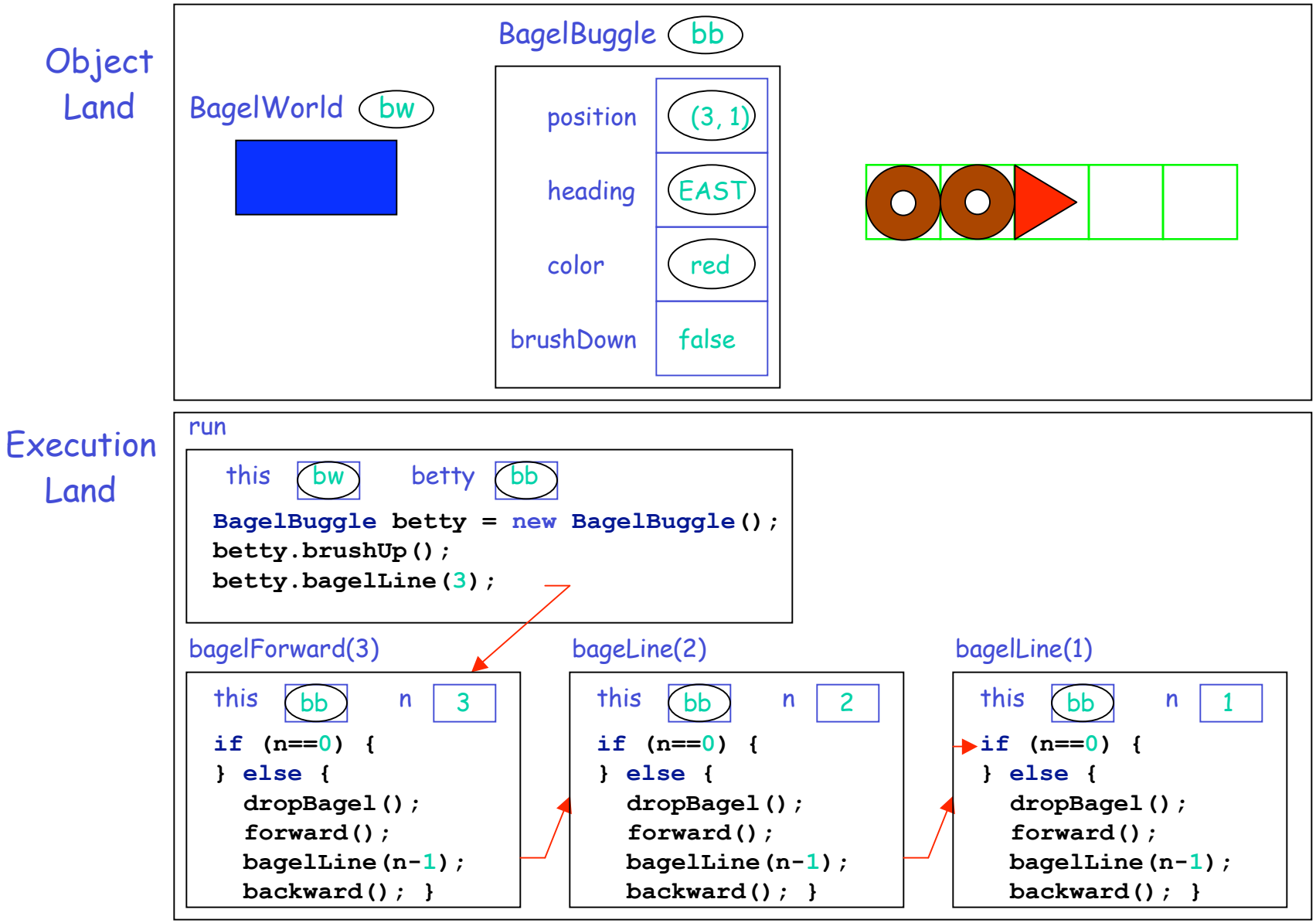
bagelLine(2)


Object Land
BagelBuggle
bb
BagelWorld
bw
position
(2, 1)
heading
EAST
color
red
brushDown
false
Execution Land
run
this
bw
betty
bb
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelLine(3);
bagelForward(3)
this
bb
n
3
if (n==0) {
} else {
dropBagel();
forward();
bagelLine(n-1);
backward(); }
bageLine(2)
this
bb
n
2
if (n==0) {
} else {
dropBagel();
forward();
bagelLine(n-1);
backward(); }

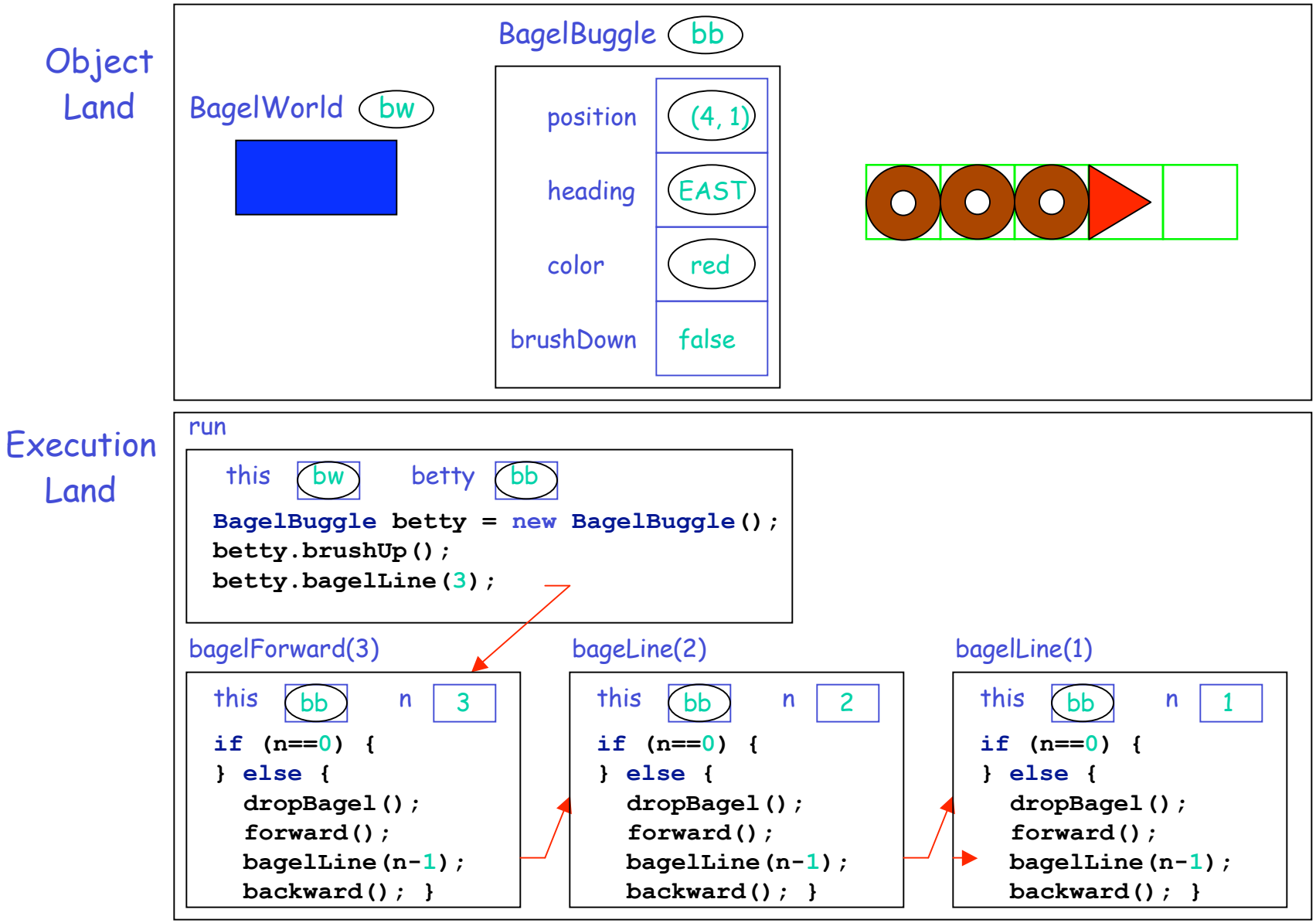
drop bagel and move forward



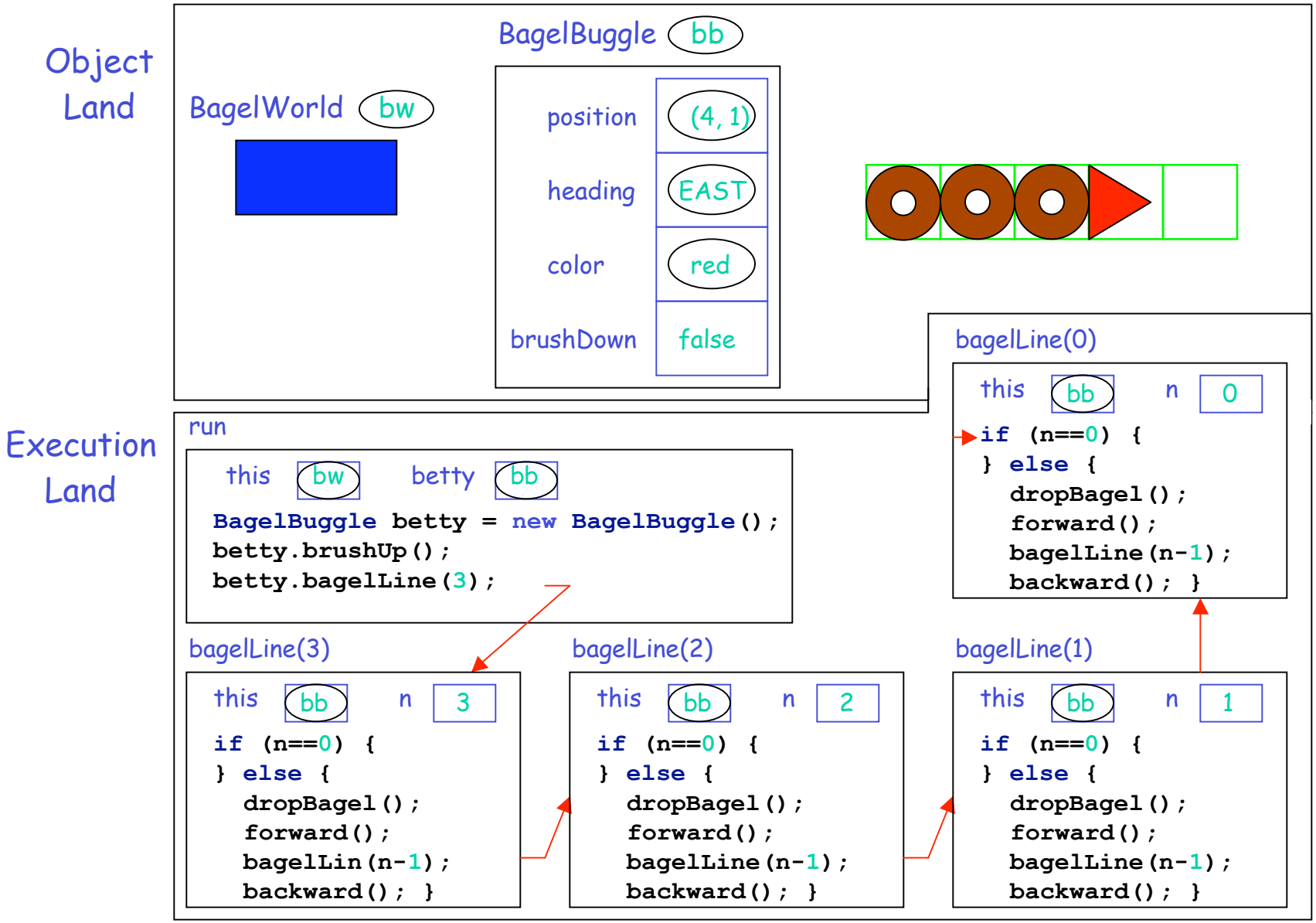
bagelLine(1)


Object Land
BagelBuggle
bb
BagelWorld
bw
position
(3, 1)
heading
EAST
color
red
brushDown
false
Execution Land
run
this
bw
betty
bb
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelLine(3);
bagelForward(3)
this
bb
n
3
if (n==0) {
} else {
dropBagel();
forward();
bagelLine(n-1);
backward(); }
bageLine(2)
this
bb
n
2
if (n==0) {
} else {
dropBagel();
forward();
bagelLine(n-1);
backward(); }
bagelLine(1)
this
bb
n
1
if (n==0) {
} else {
dropBagel();
forward();
bagelLine(n-1);
backward(); }

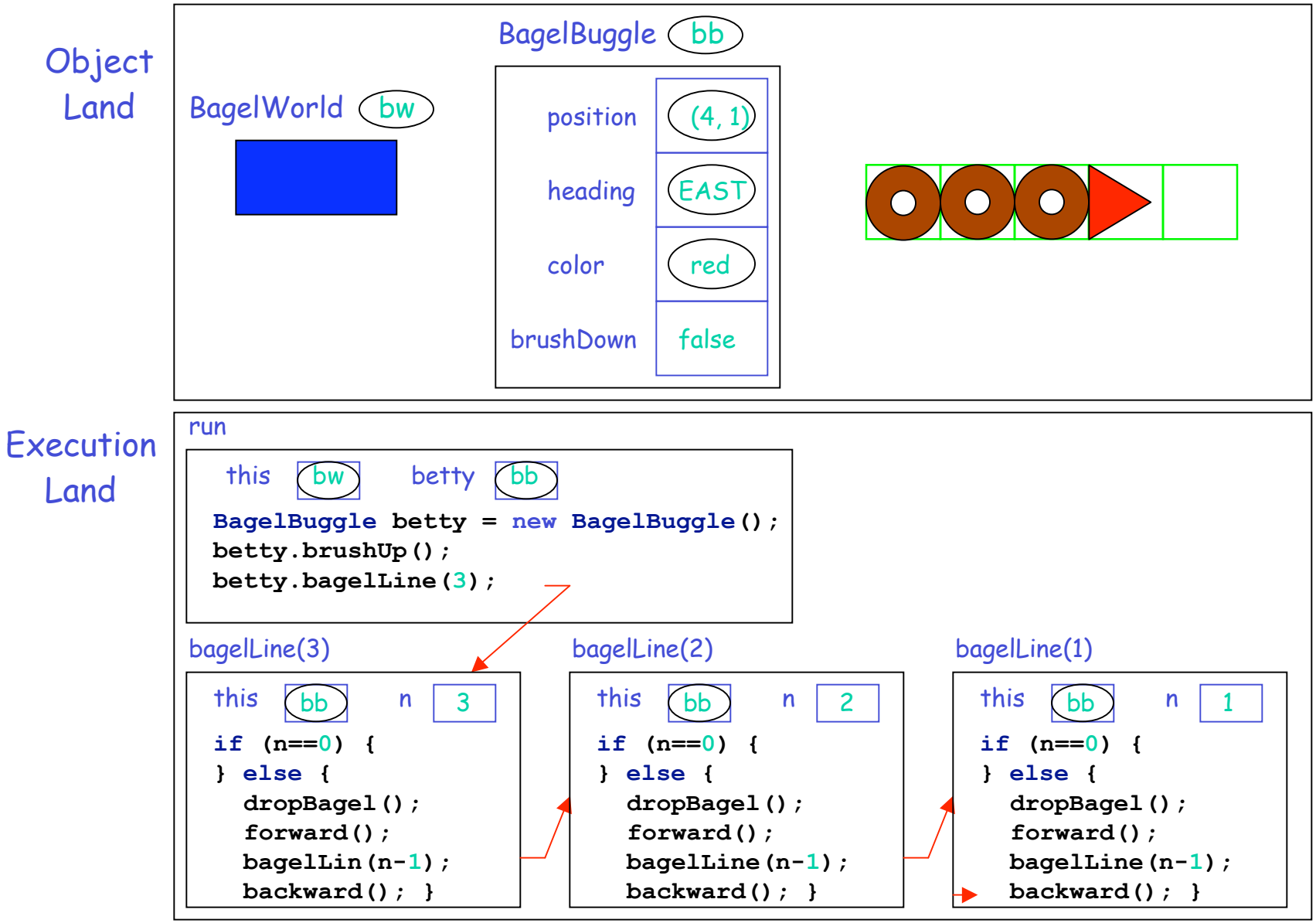
Dropping our last bagel



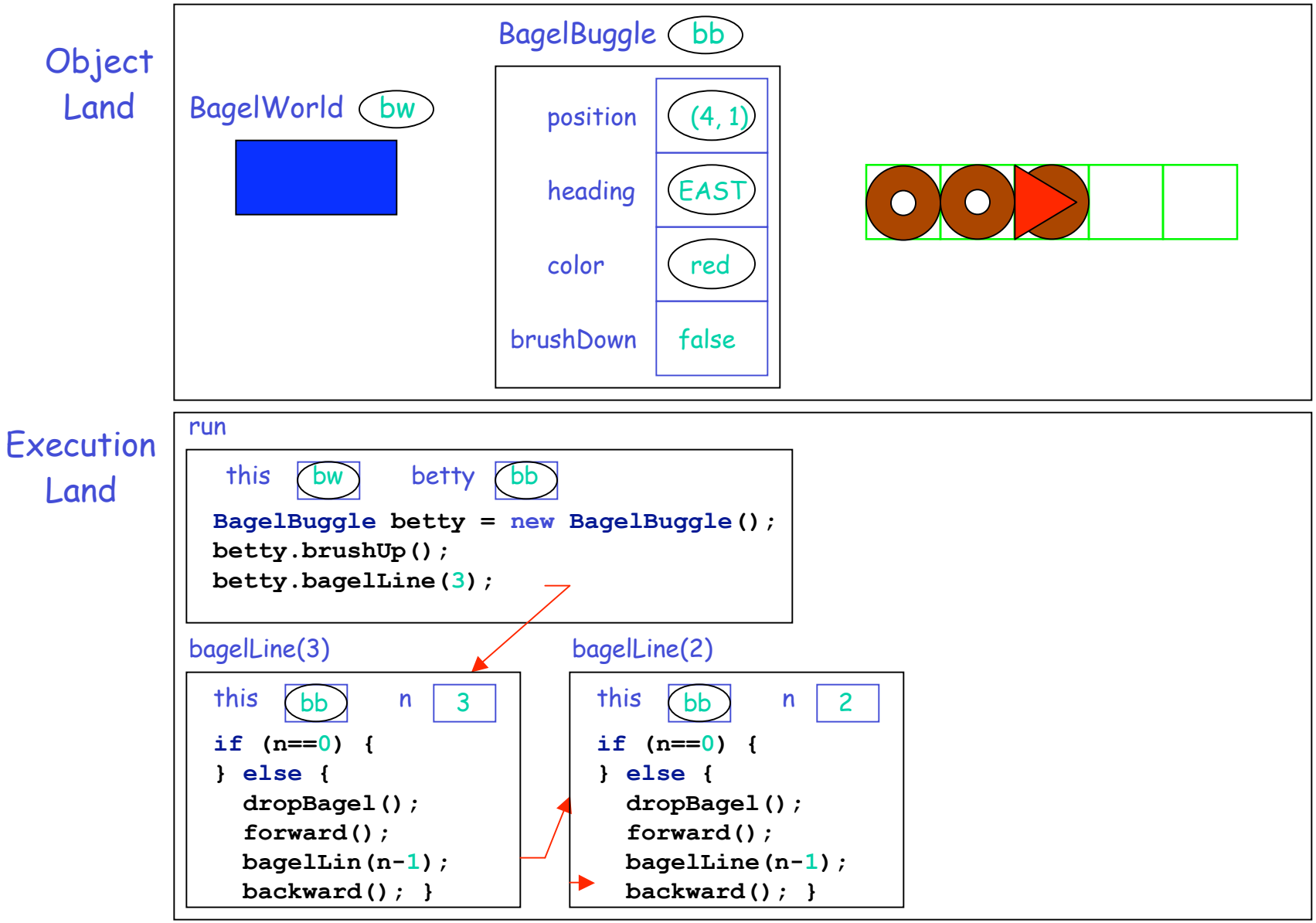
Back to basics



bagelLine(0) is done

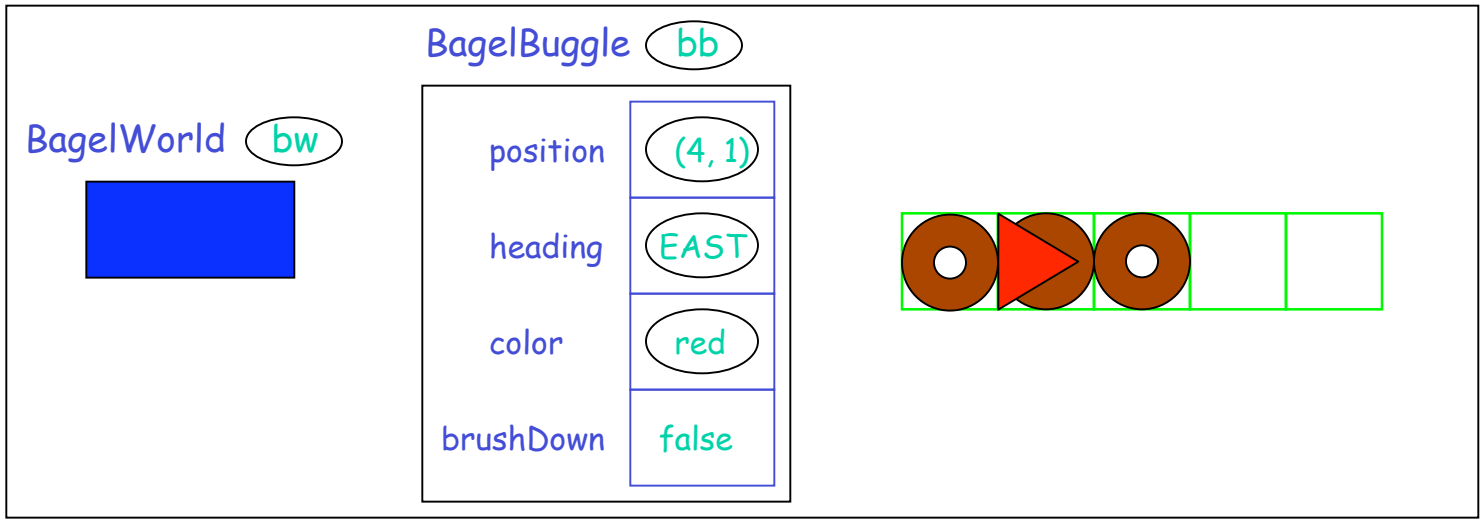


back up and complete



back up and complete again

Object Land



Execution Land

run

this bw betty bb

```
BagelBuggle betty = new BagelBuggle();
betty.brushUp();
betty.bagelLine(3);
```

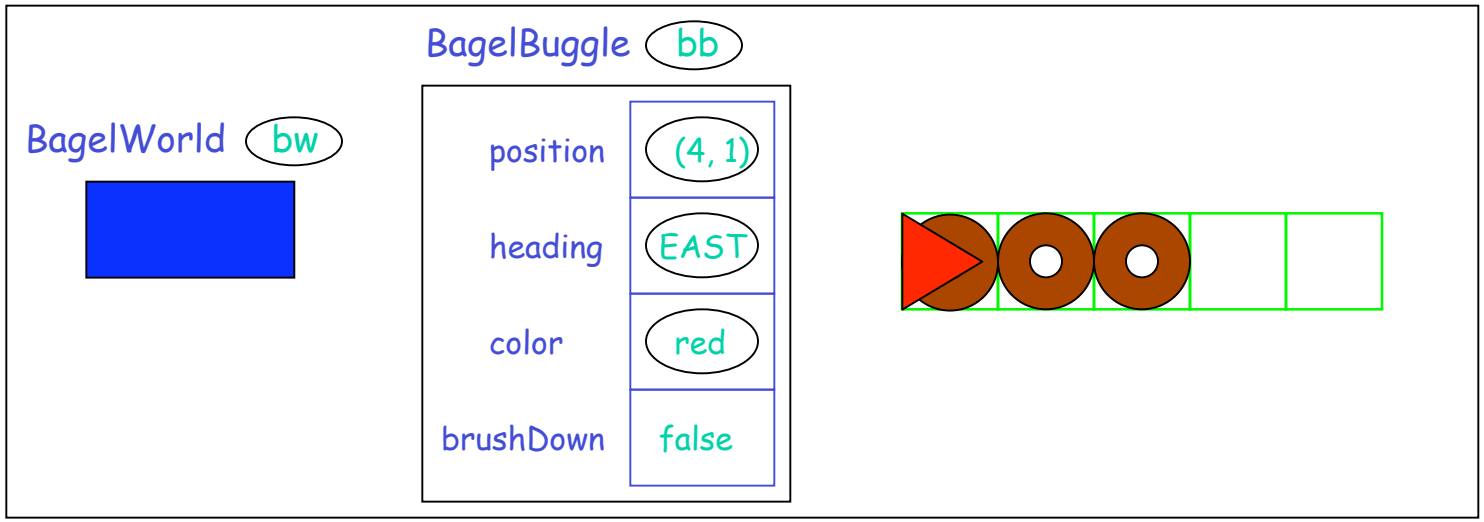
bagelLine(3)

this bb n 3

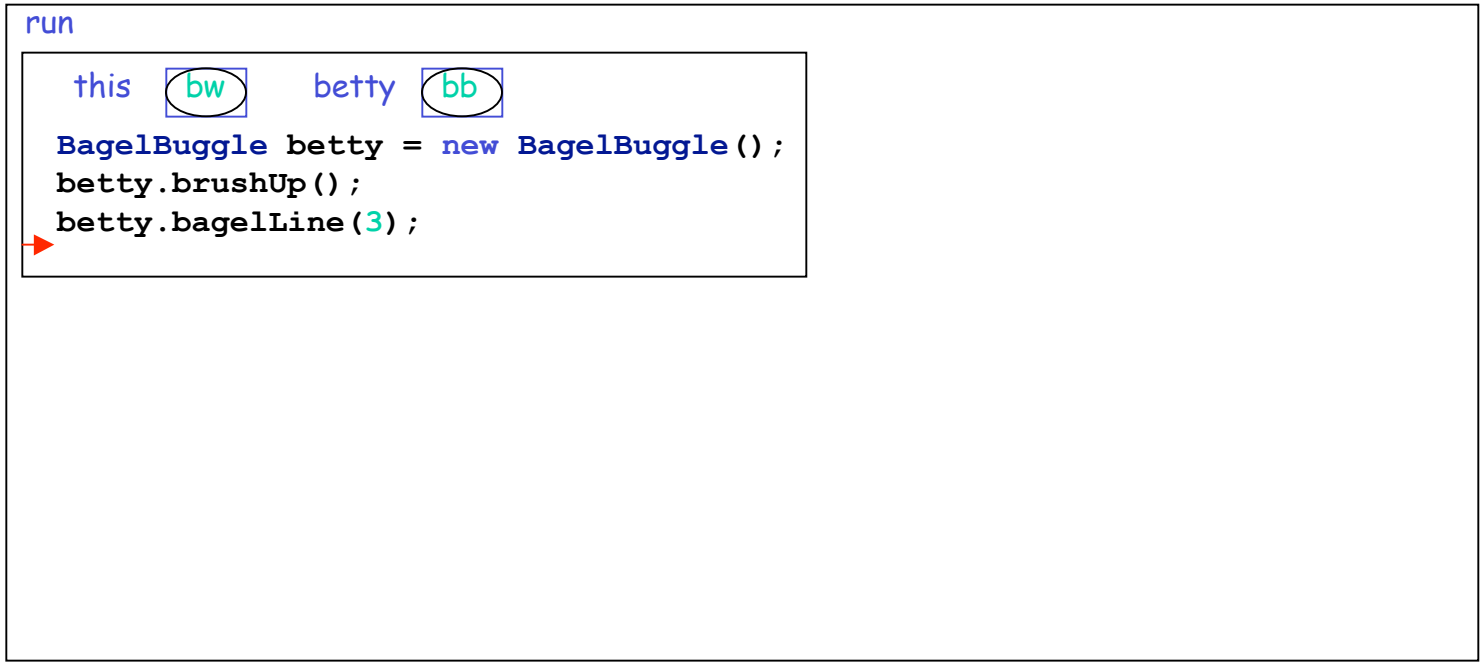
```
if (n==0) {
} else {
  dropBagel();
  forward();
  bagelLin(n-1);
  backward(); }
```

Done

Object Land

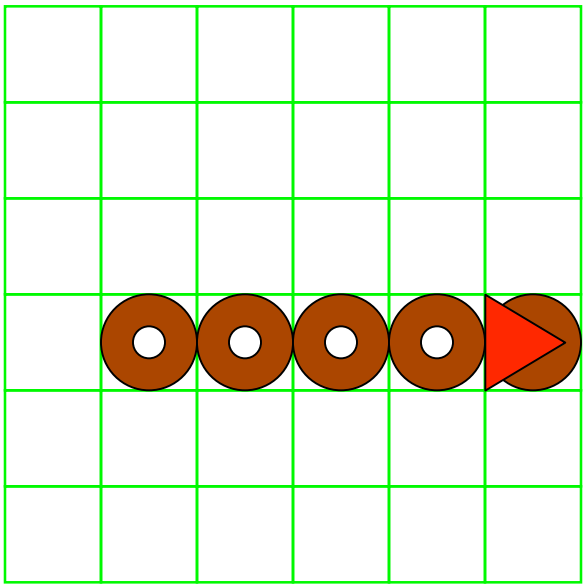


Execution Land



`bagelsToWall();`

Write a method that teaches a buggle to leave behind a trail of bagels all the way to the wall. Assume brushUp().

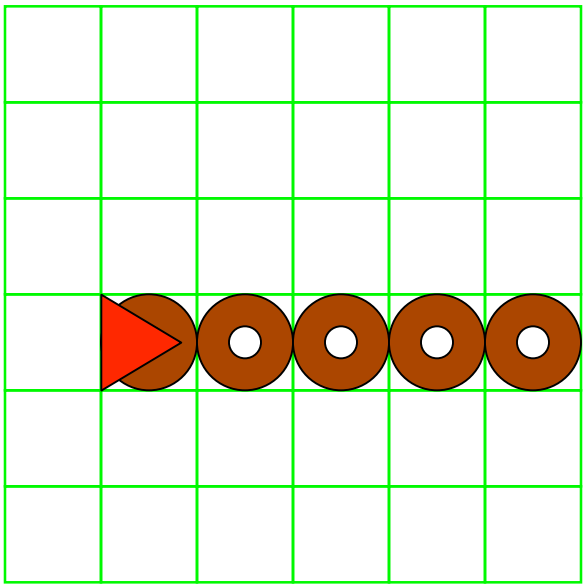


bagelsToWall();

```
public void bagelsToWall() {
  if (isFacingWall()) {              // base case
    dropBagel();
  } else {                           // recursive case
    dropBagel();
    forward();
    bagelsToWall();                  // drop rest of bagels
  }
}
```

`bagelsToWallAndBack();`

Write a method that teaches a buggle to leave behind a trail of bagels all the way to the wall, and returns to starting point. Assume brushUp().



bagelsToWallAndBack();

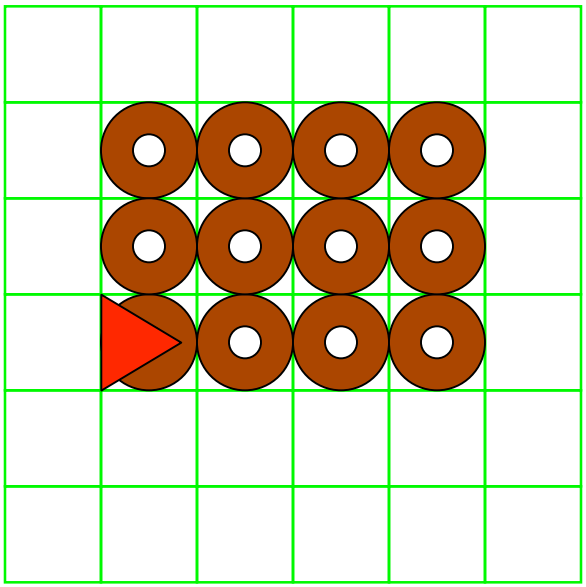
```
public void bagelsToWallAndBack() {
  if (isFacingWall()) {                  // base case
    dropBagel();
  } else {                               // recursive case
    dropBagel();
    forward();
    bagelsToWallAndBack();               // drop rest of bagels
    backward();                          // and backup
  }
}
```

Alternatively

```
public void bagelsToWallAndBack() {
  if (isFacingWall()) {              // base case
    dropBagel();
  } else {                           // recursive case
    forward();
    bagelsToWallAndBack();           // drop rest of bagels
    backward();                      // backup and
    dropBagel();                     // drop bagel at start
  }
}
```

Bagel Rectangle

Write a method that teaches a buggle to draw a w by h rectangle of bagels, and returns to starting point. Assume brushUp().



*How many parameters would such a method take?

bagelRect(int w, int h);

```
public void bagelRect(int w, int h) {
  if (h == 0) {                          // base case
                                         // nothing to do
  } else {                               // recursive case
    bagelLine(w);                        // draw bottom line

    left();                              // move to the
    forward();                           // next row
    right();

    bagelRect(w, h-1);                   // draw "subrect"

    left();                              // undoes state change
    backward();                          // undoes state change
    right();
  }
}
```