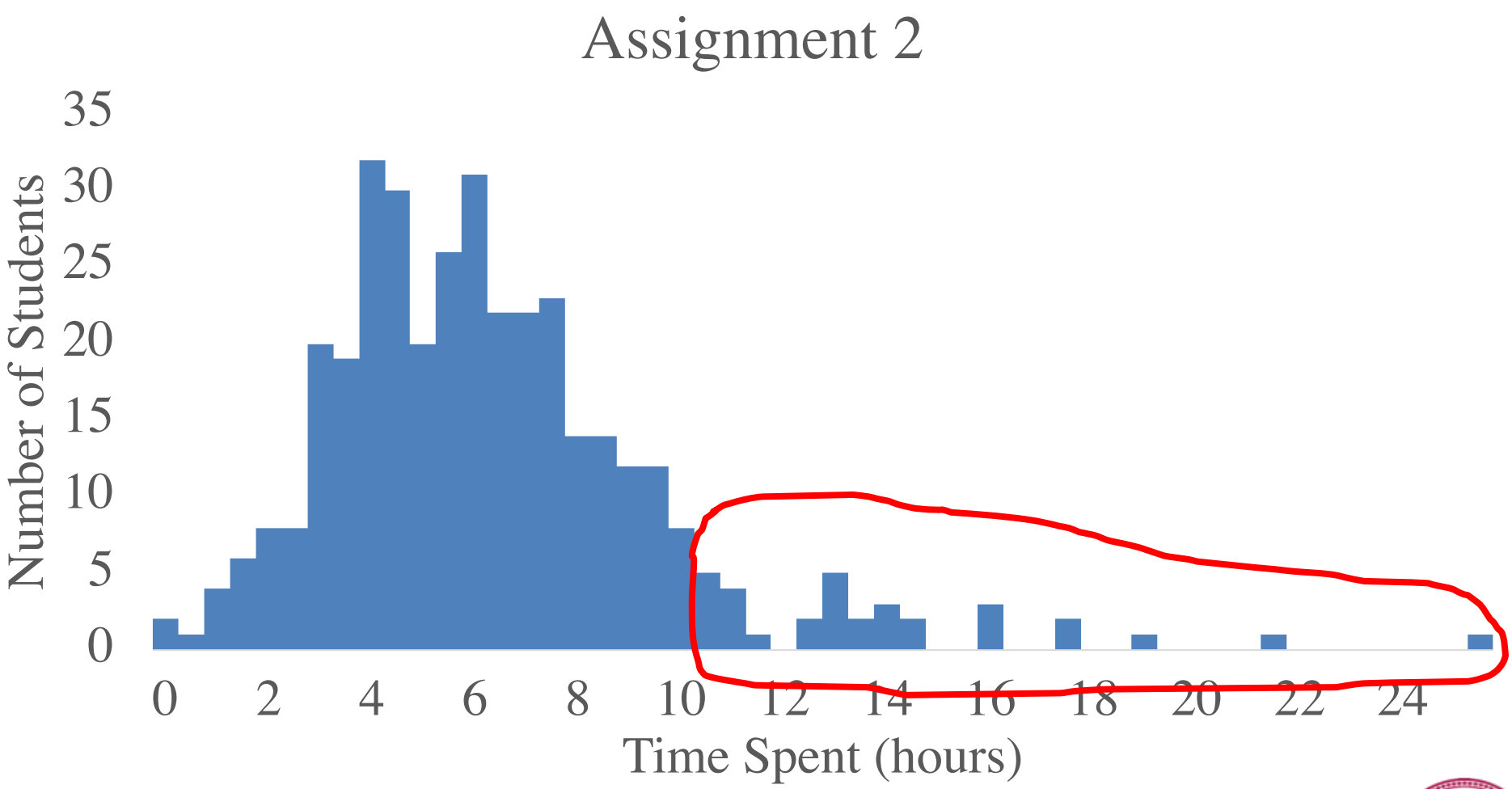
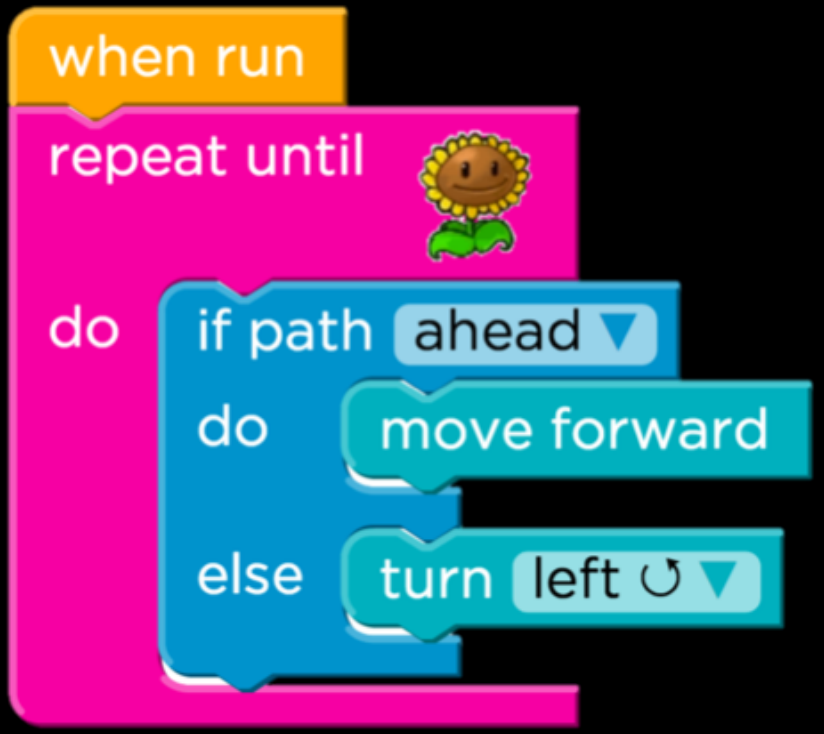
Memory

Chris Piech
CS106A, Stanford University

Learn by Doing



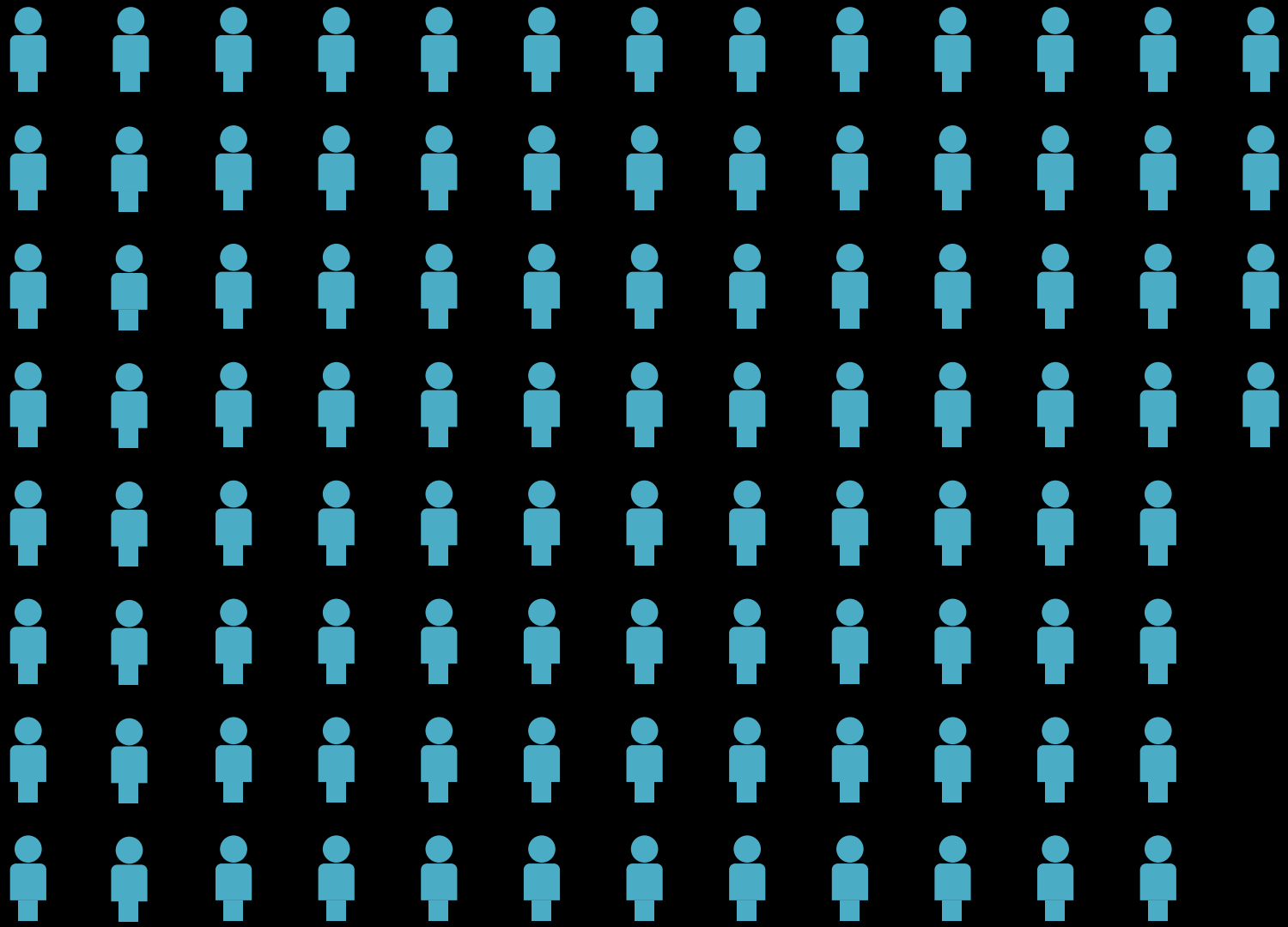
Learning to Program on the Internet



Task

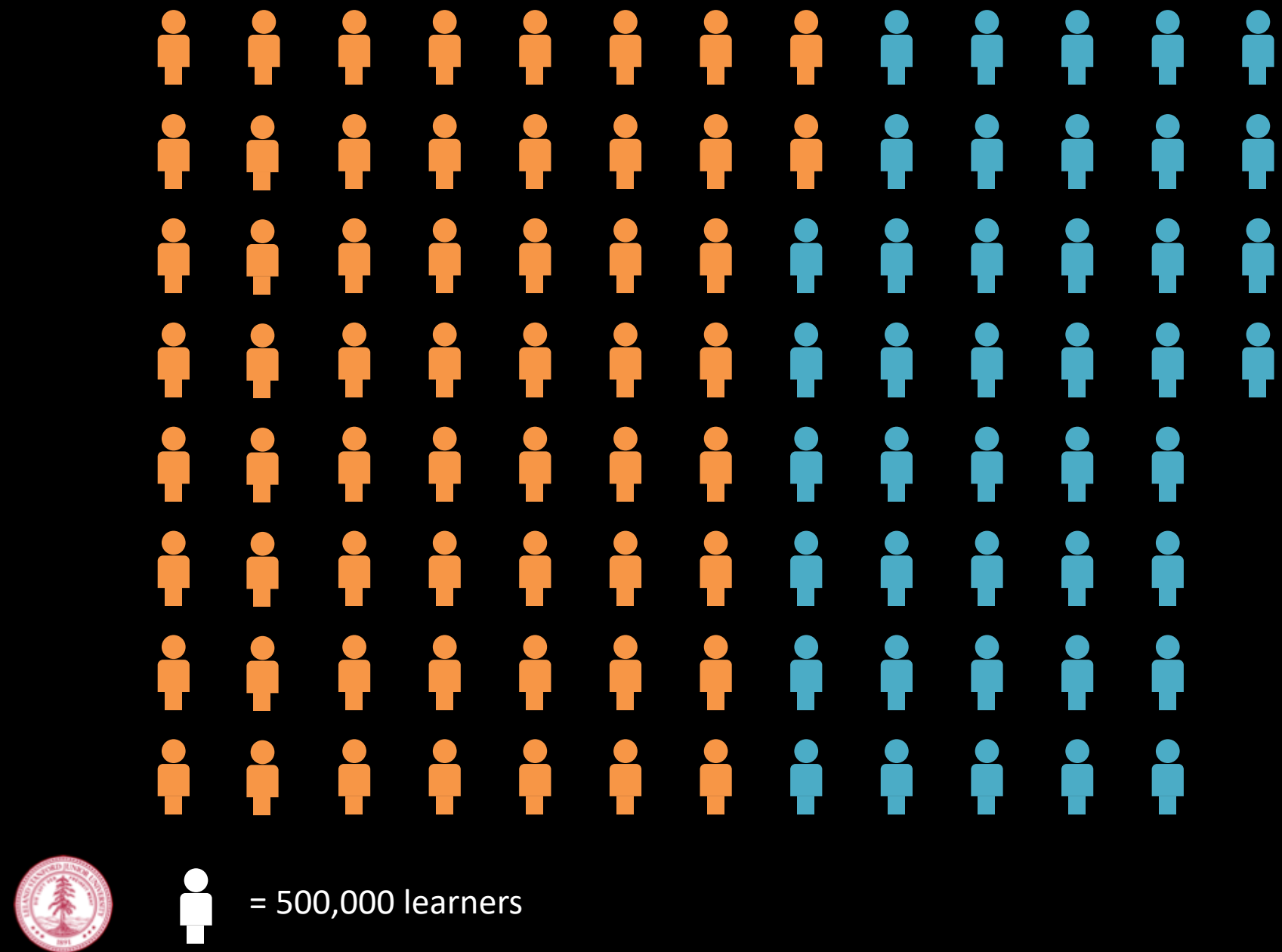
Almost a hundred thousand unique solutions

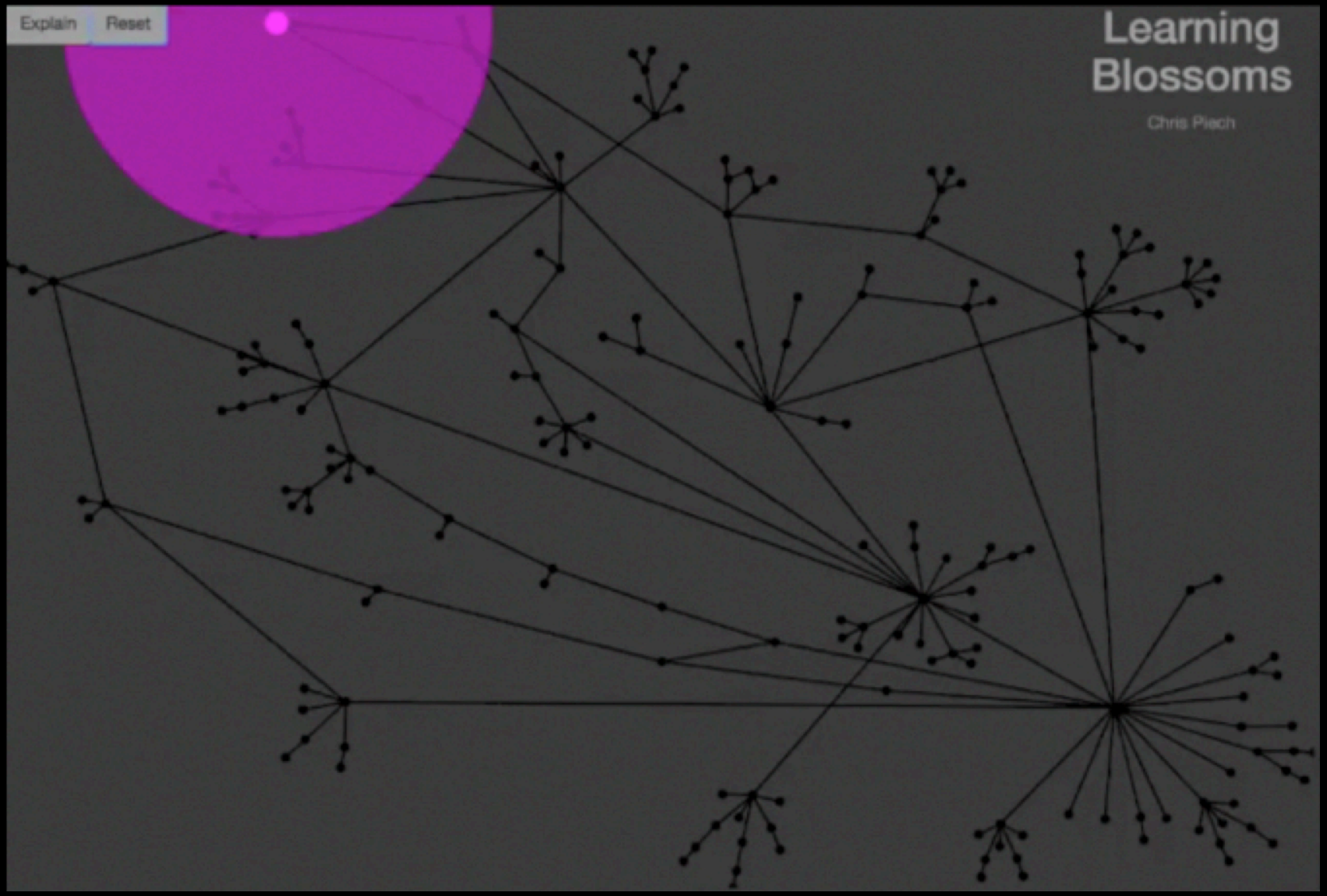
US K-12 Students



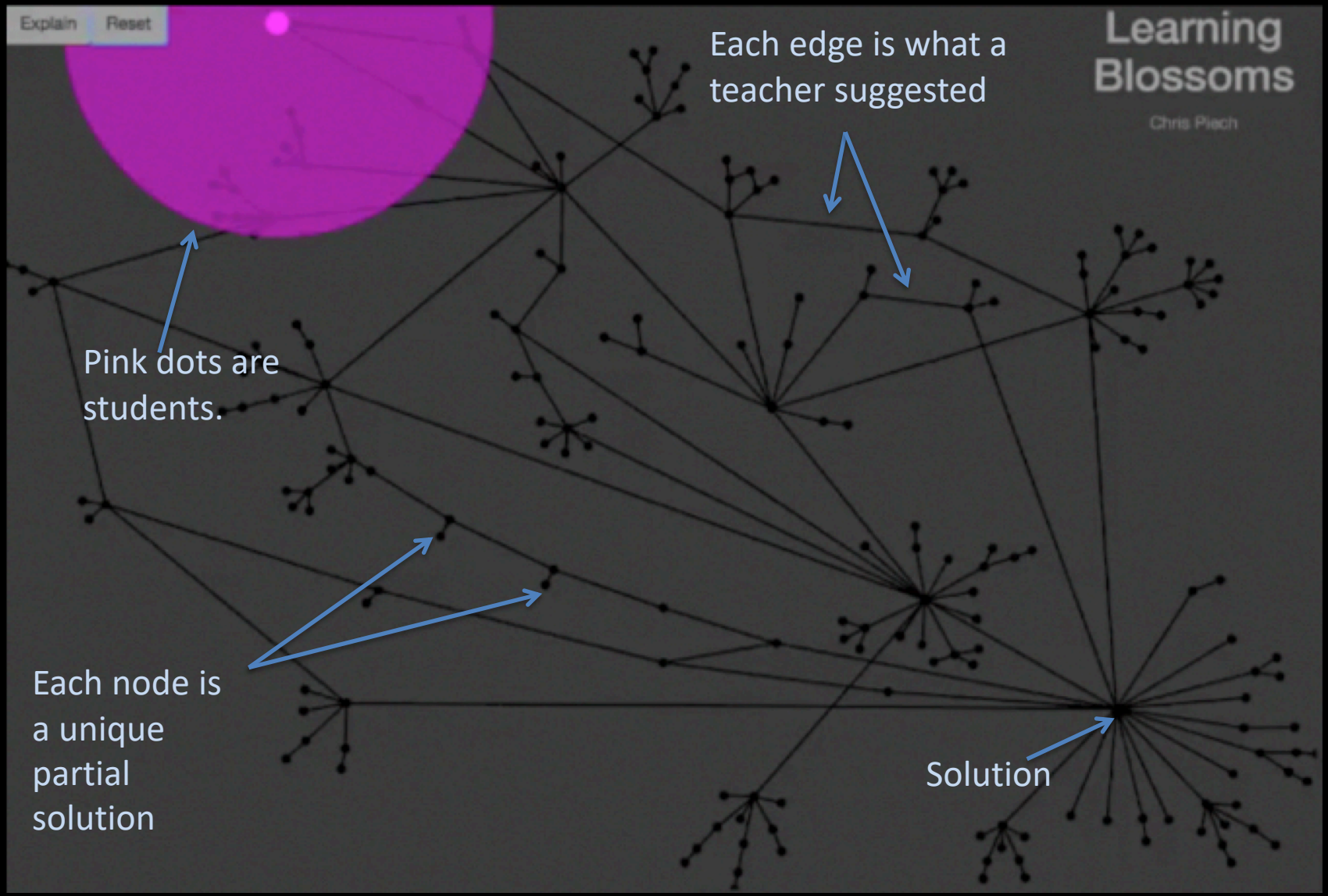
= 500,000 learners

Code.org Students

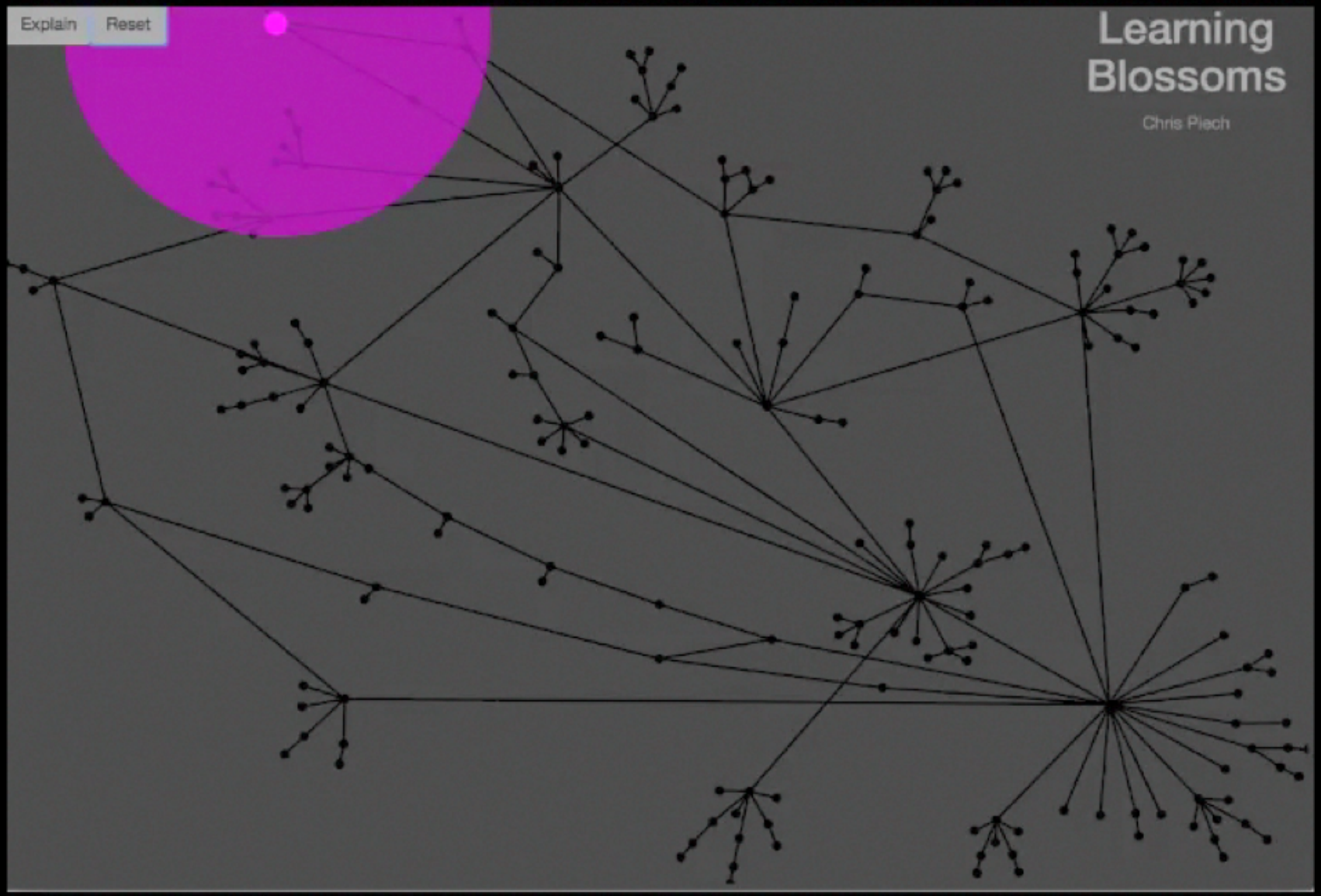




Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.



Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.



Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.

Desirable Path Algorithm

Poisson Common Path

$$\gamma(s) = \arg\min_{p \in Z(s)} \sum_{x \in p} \frac{1}{\lambda_x}$$

Path Cost

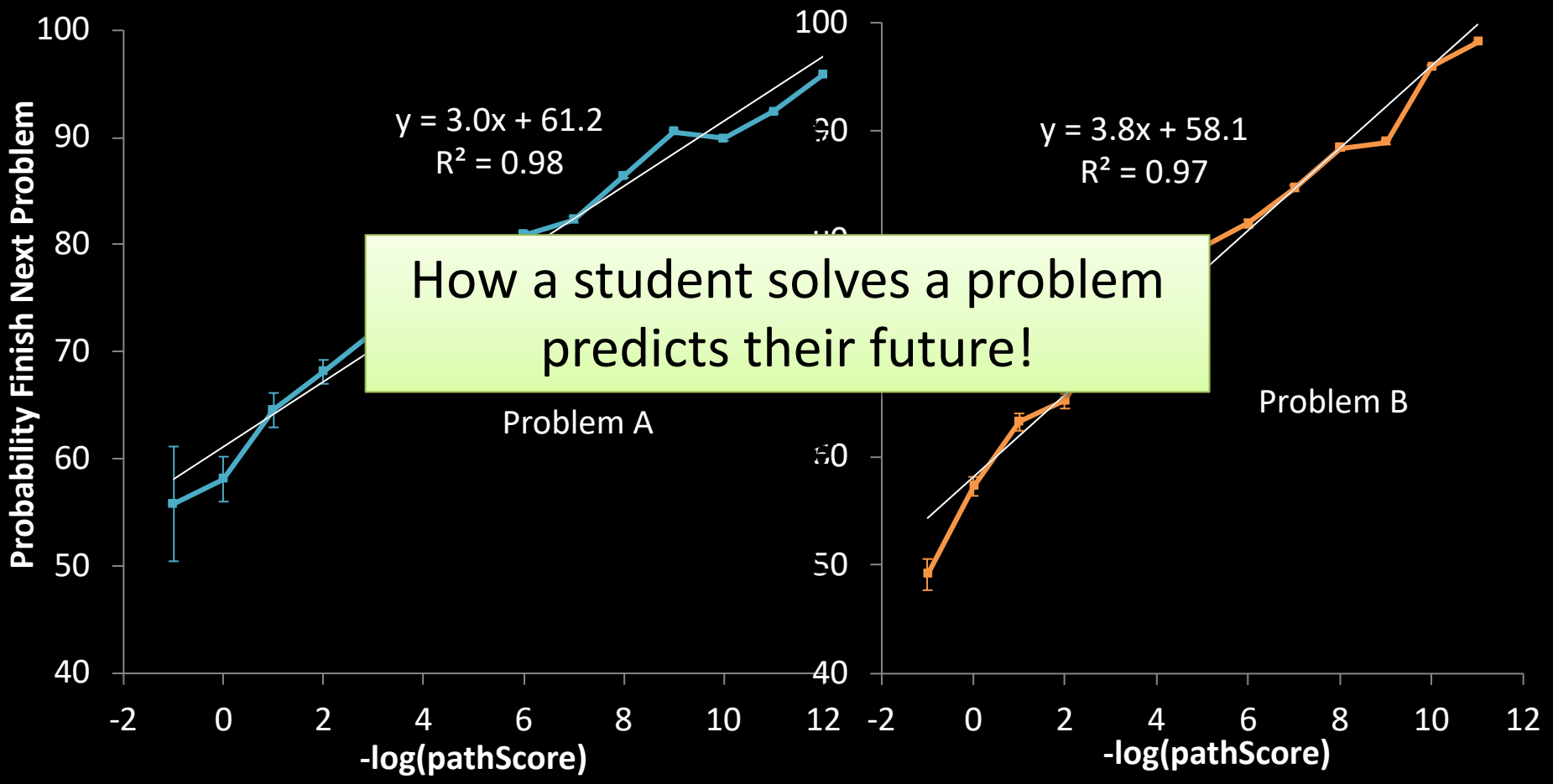
Predicted next partial solution

Paths to solution

Partial solutions in the path

Submission count of partial solution

Predicts Future Success

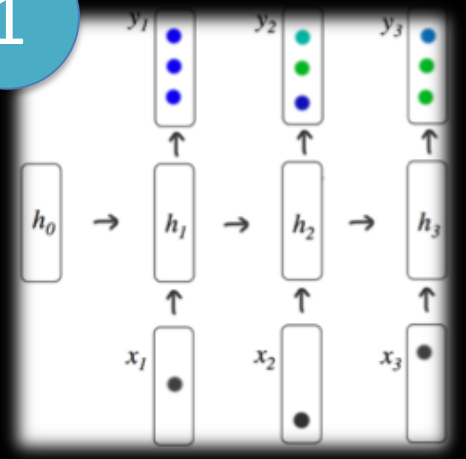


How a student solves a problem predicts their future!

Effect is large and logrithmic.

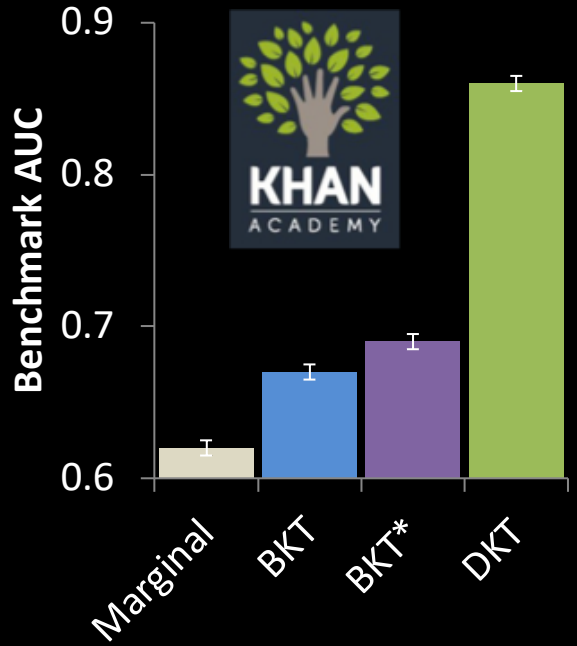
Deep Learning Algorithms

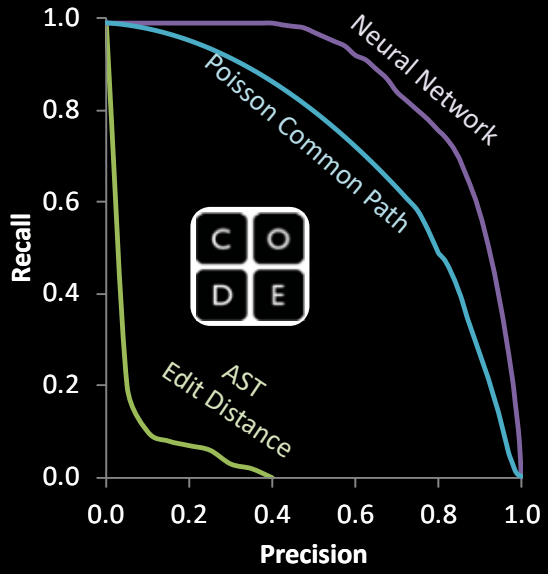
1



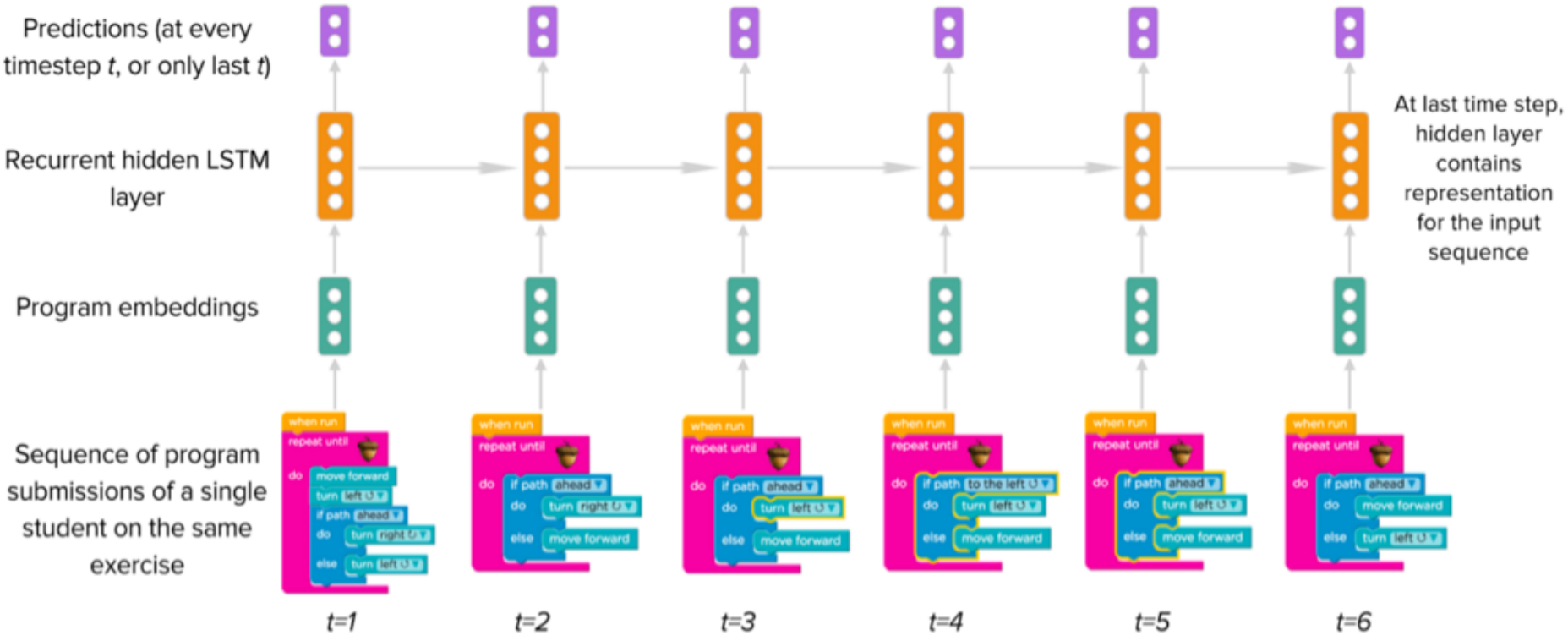
2

Program $\rightarrow$ $\mathbb{R}^n$



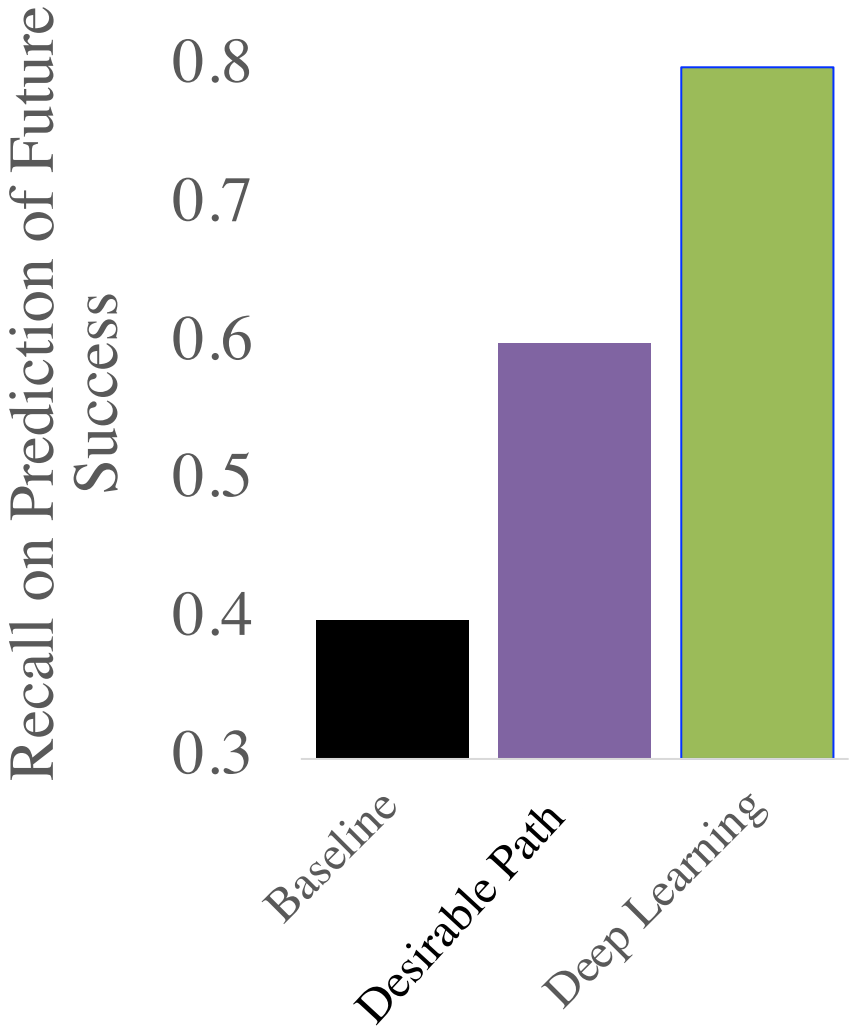


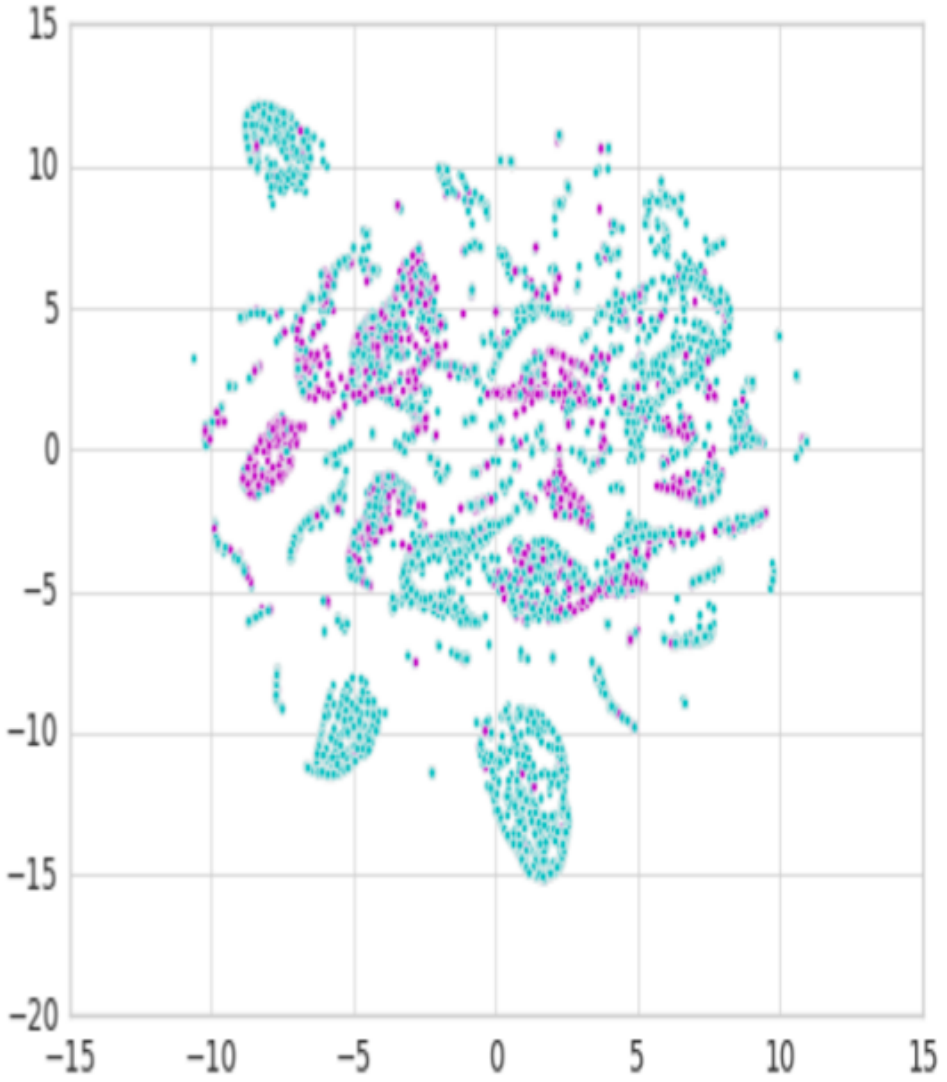
Deep Learning on Trajectories



Research in collaboration with Lisa Wang

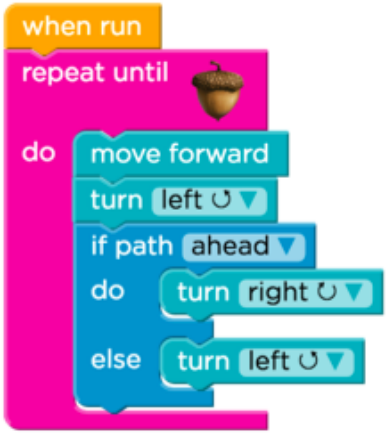
Deep Learning on Trajectories



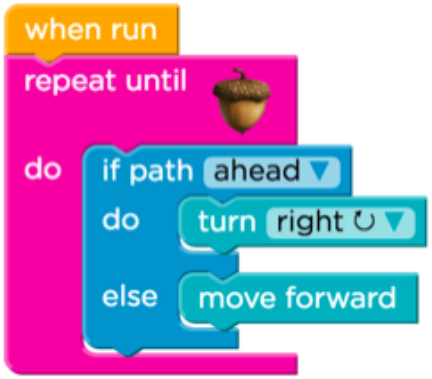


Highly Rates Grit

1. Two compound errors

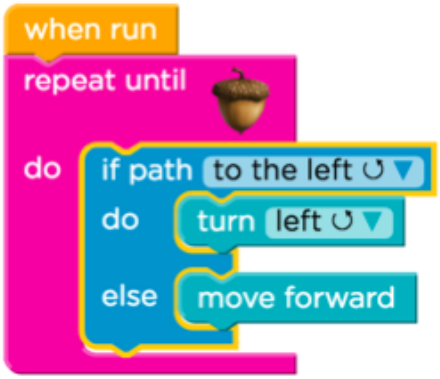


2. Solves first error



3. Starts reasonable attempt

4. Completes attempt



5. Backtracks

6. Finds solution

Back to our regularly scheduled programming

Warmup

The TSA spent $336,000 on an iPad app that decides what line you go in at security *

* TSA says it was either $50k or $340k

Learning Goals

1. Be able to trace memory for objects

Who thinks this prints **`true`**?

```
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```


Who thinks this prints **`true`**?

```
public void run() {
    int x = 5;
    int y = 5;
    println(x == y);
}
```


Who thinks this prints **`true`**?

```
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```


Deep Understanding is Key

```
private GRect brick;
public void update() {
    GObject collider = getCollidingObject();
    if(collider == brick) {
        remove(brick);
    }
}
```


[suspense]

Review

Primitives vs Classes

Primitive Variable Types

```
int
double
char
boolean
```

Class Variable Types

```
GRect
GOval
Gline
Color
```

Class variables (aka objects)

1. Have upper camel case types
2. You can call methods on them
3. Are constructed using `new`
4. Are stored in a special way

How do you share wikipedia articles?

Antelope Canyon Article

Antelope Canyon is a slot canyon in the American Southwest. It is located on Navajo land east of Page, Arizona. Antelope Canyon includes two separate, photogenic slot canyon sections, referred to individually as *Upper Antelope Canyon* or *The Crack*; and *Antelope Canyon* or *The Corkscrew*.[2]

The Navajo name for Upper Antelope Canyon is Tsé bighánílíní, which means "the place where water runs through rocks." Lower Antelope Canyon is Hazdistazí (advertised as *"Hasdestwazi"* by the Navajo Parks and Recreation Department), or "spiral rock arches." Both are located within the LeChee Chapter of the Navajo Nation.[4]

Contents [hide]

1 Geology
2 Tourism and photography
2.1 Upper Antelope Canyon

Antelope Canyon

Tsé bighánílíní dóó Hazdistazí (Navajo)

A beam of light in Upper Antelope Canyon

https://en.wikipedia.org/wiki/Antelope_Canyon

What does an object store?

Objects store addresses
(which are like URLs)

```
public void run() {
    GRect r = null;
}
```

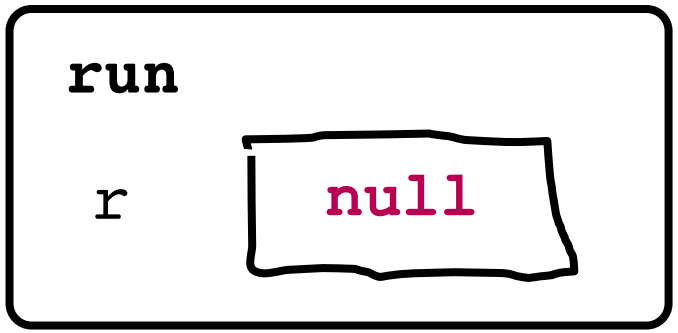
---

stack

```
public void run() {
   GRect r = null;
}
```

---

stack

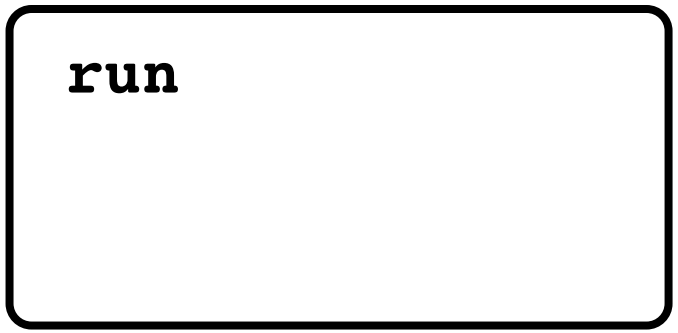


Wahoo!

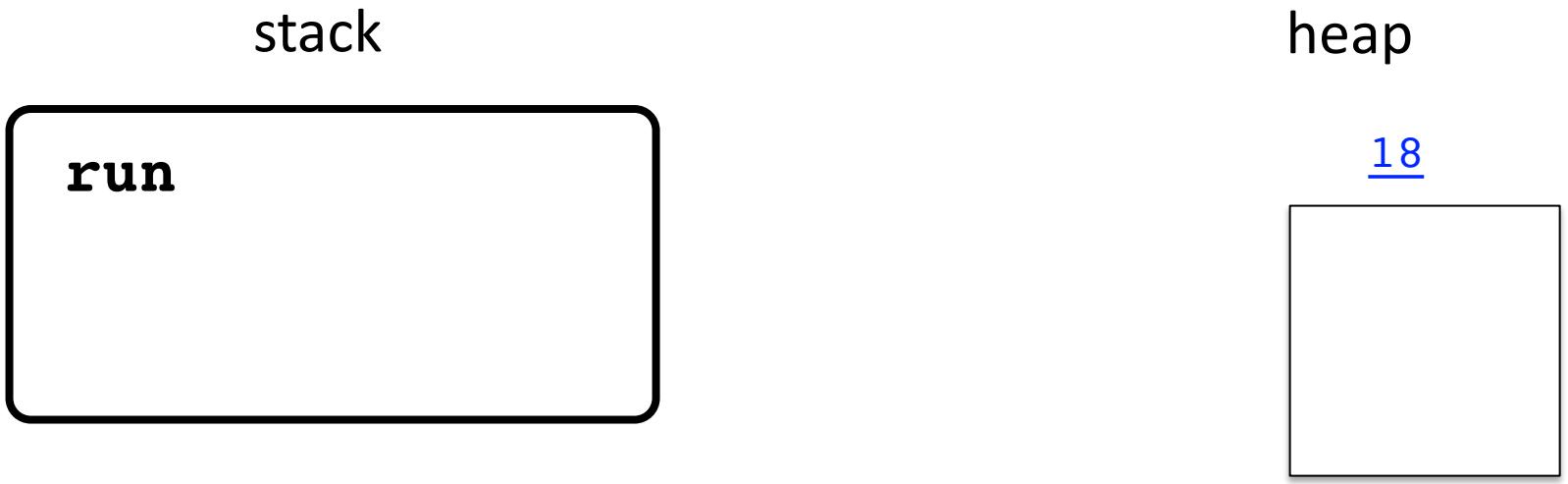
```
public void run() {
    GRect r = new GRect(50, 50);
}
```

Method Memory

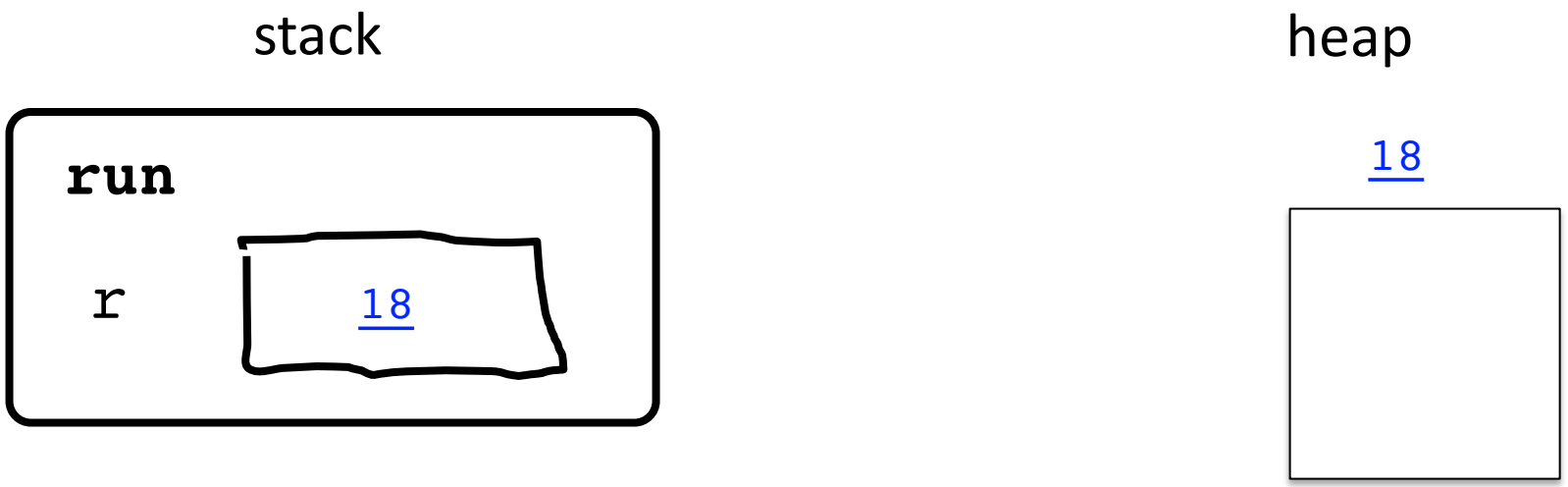
Object Memory



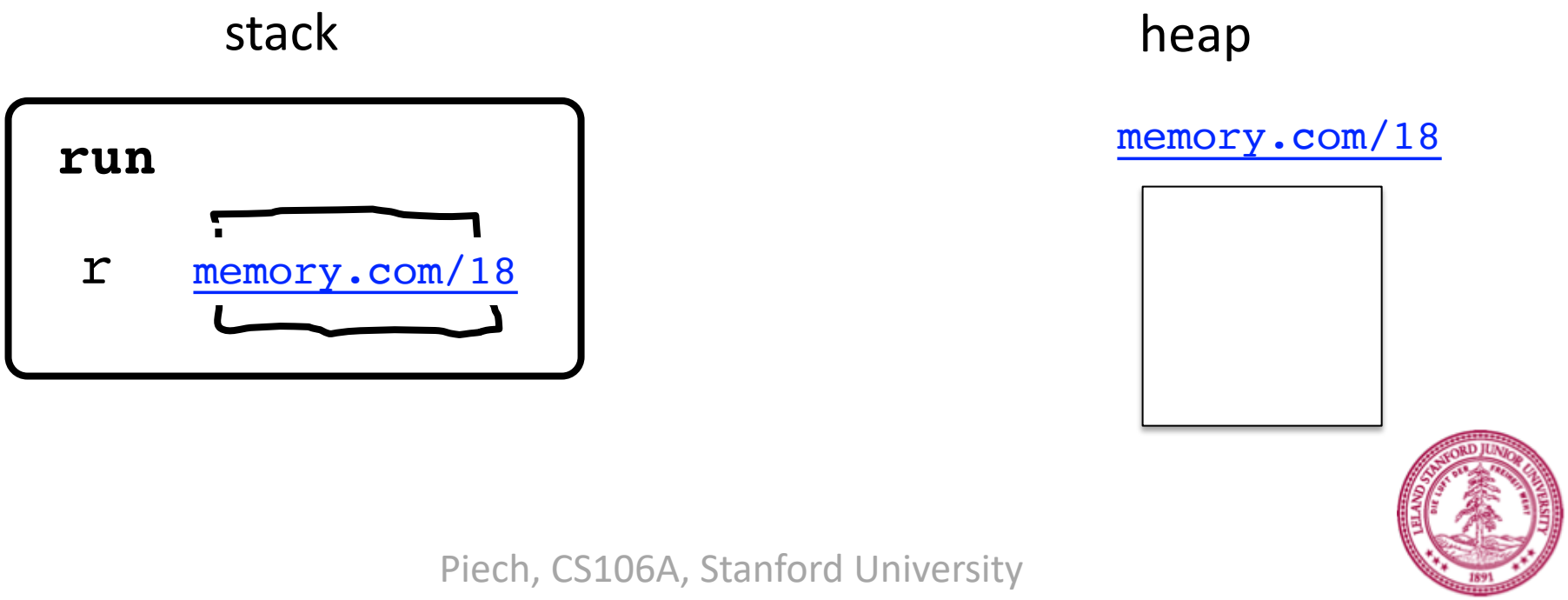
```
public void run() {
    GRect r = new GRect(50, 50);
}
```



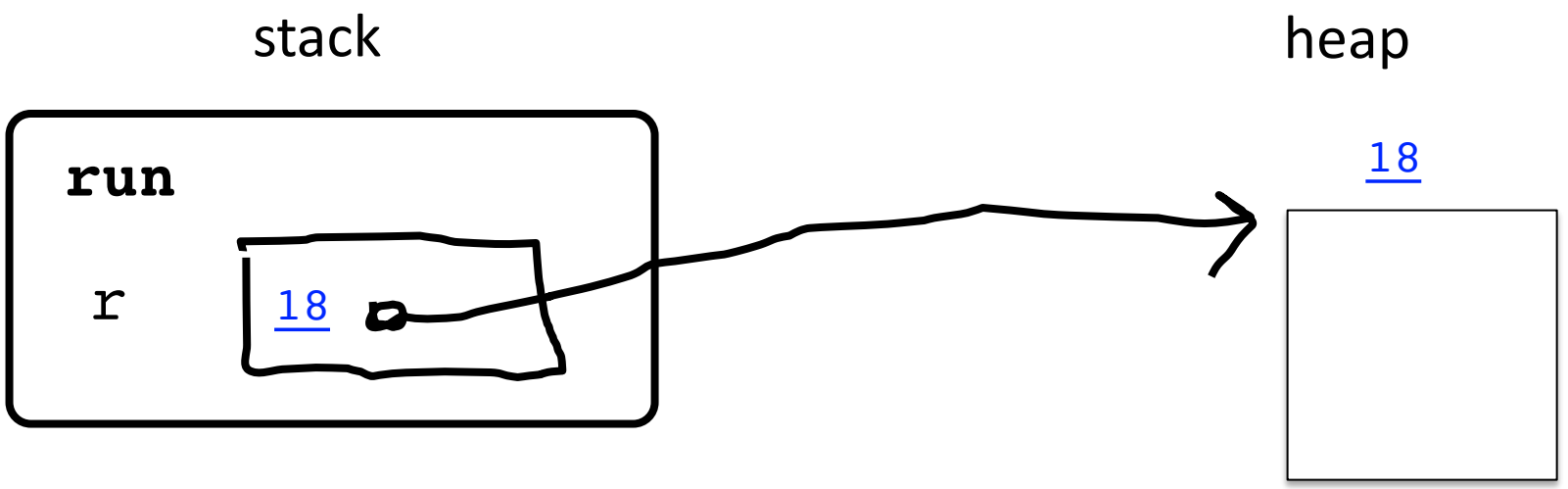
```
public void run() {
   GRect r = new GRect(50, 50);
}
```



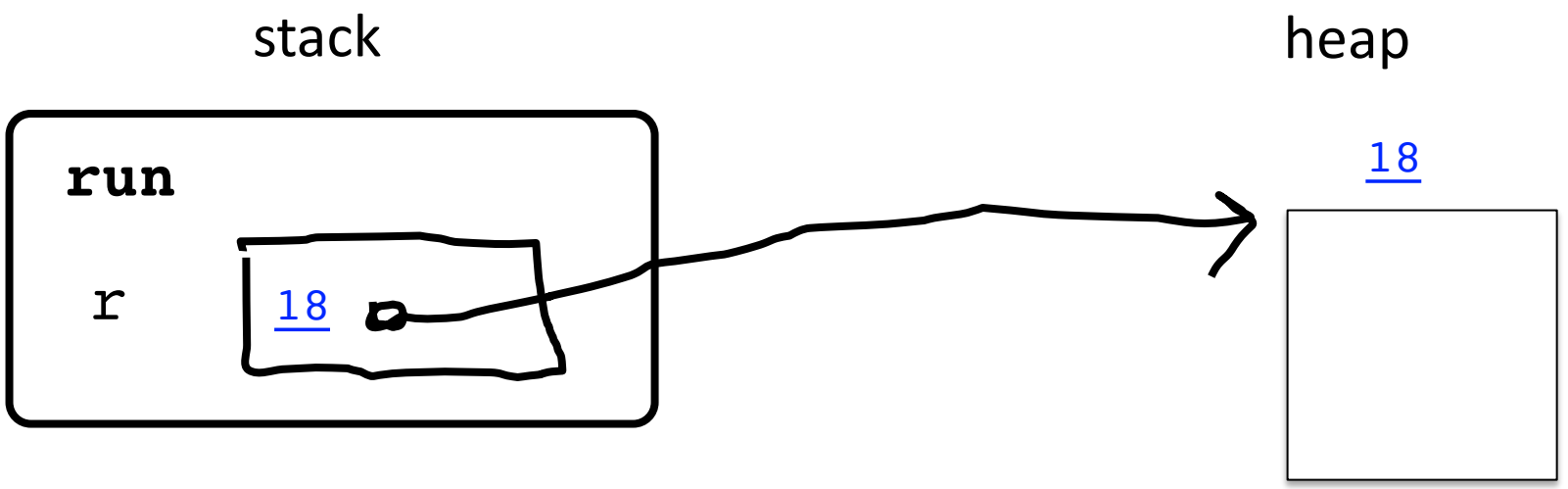
```
public void run() {
   GRect r = new GRect(50, 50);
}
```



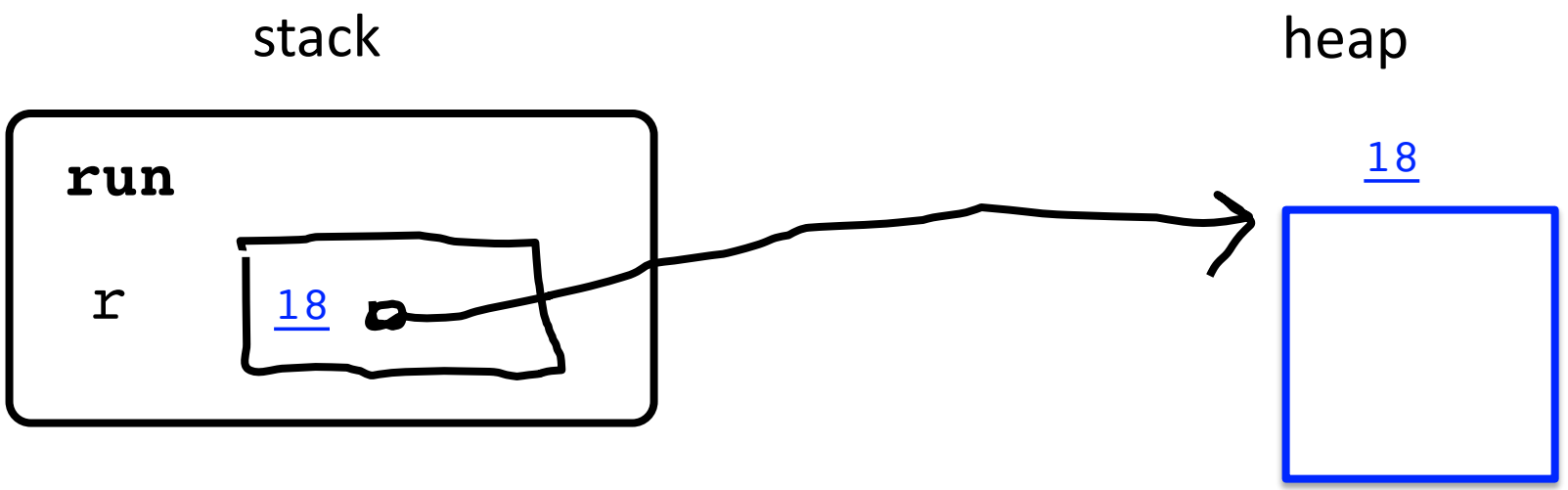
```
public void run() {
   GRect r = new GRect(50, 50);
}
```



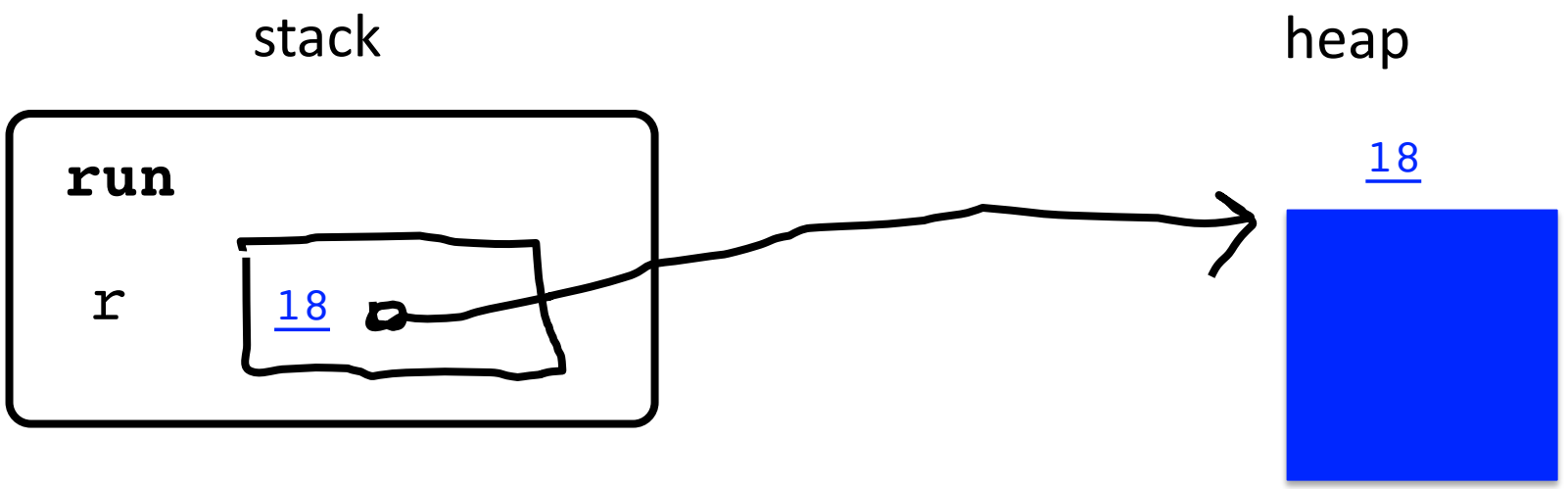
```
public void run() {
   GRect r = new GRect(50, 50);
   r.setColor(Color.BLUE);
   r.setFilled(true);
}
```



```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```



```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```



#1: `new` allocates memory on the heap

#2: object variables store heap addresses

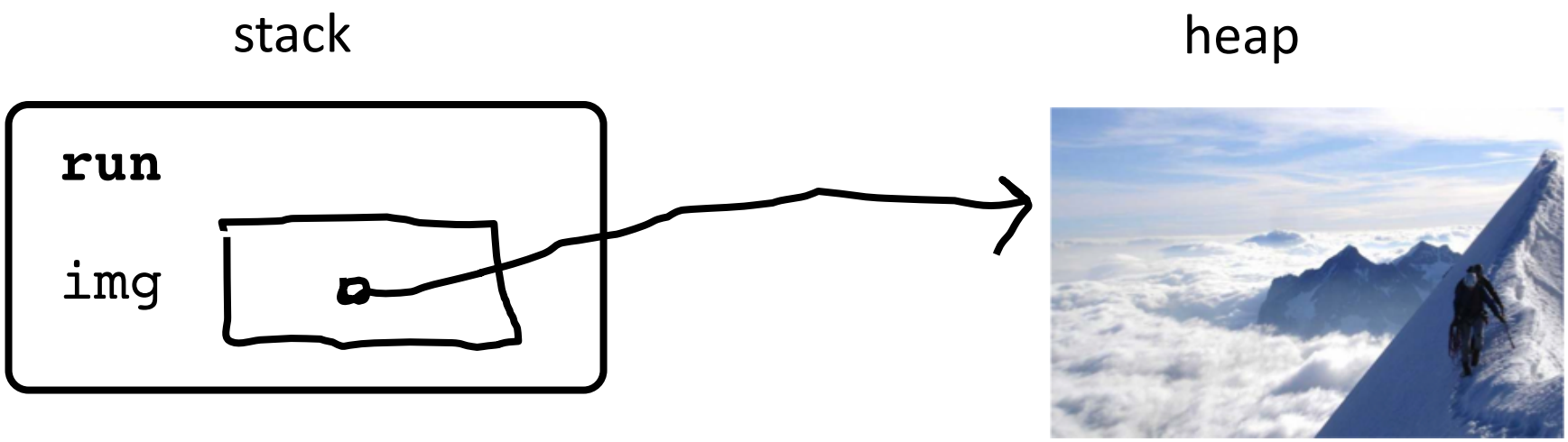
#ultimatekey

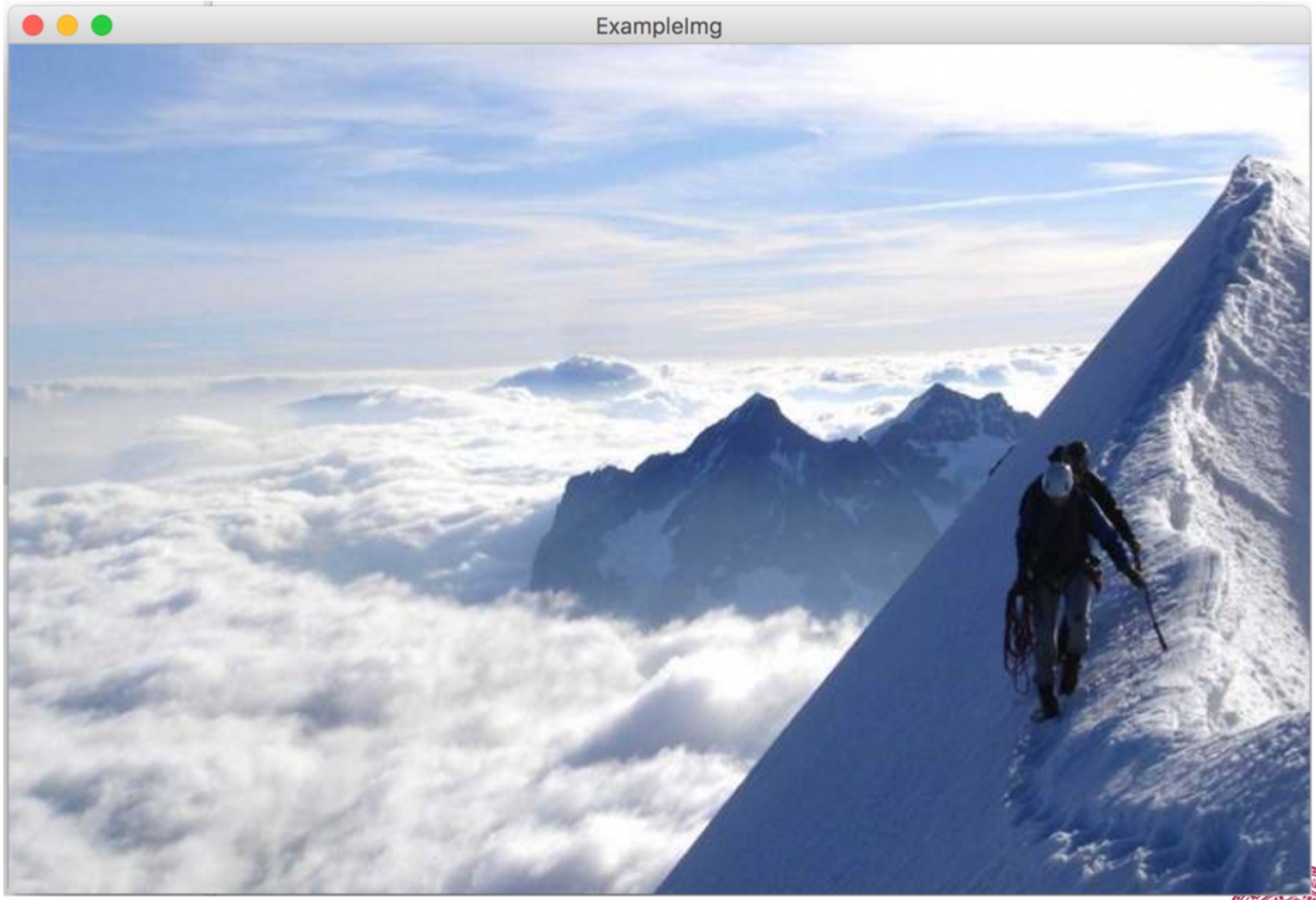
End Review

What does an object store?

Objects store addresses
(which are like URLs)

```
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}
```




ExampleImg

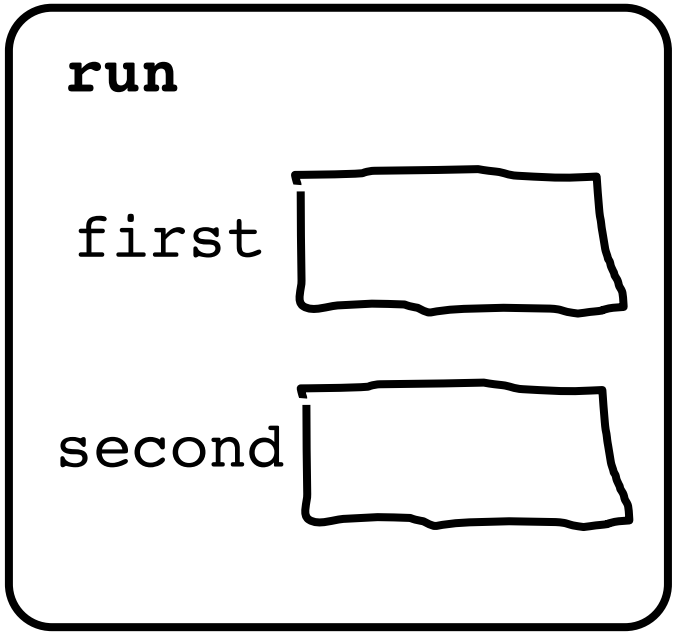
#3: **`GImage`**s look impressive but don't take much extra work

```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```

---

stack

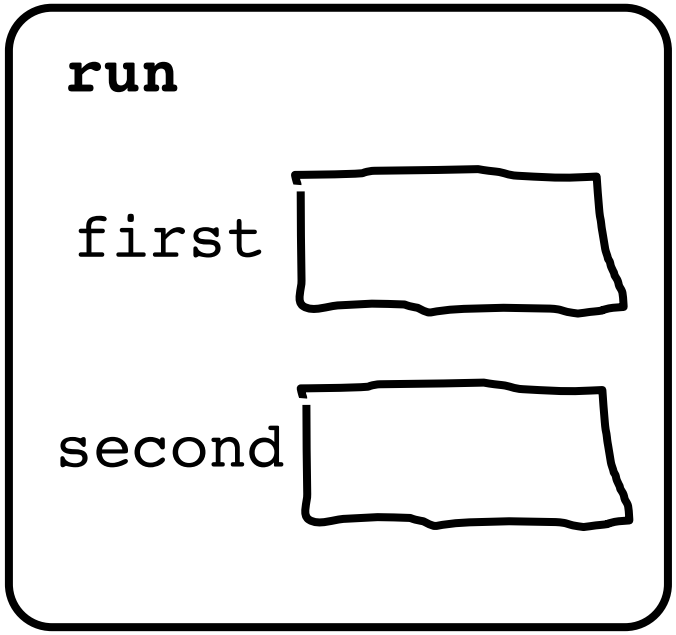
heap



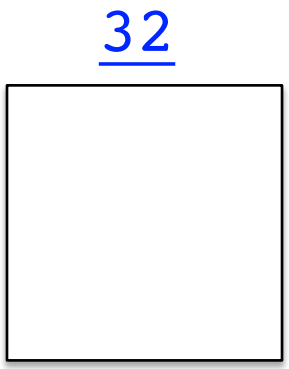
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```

---

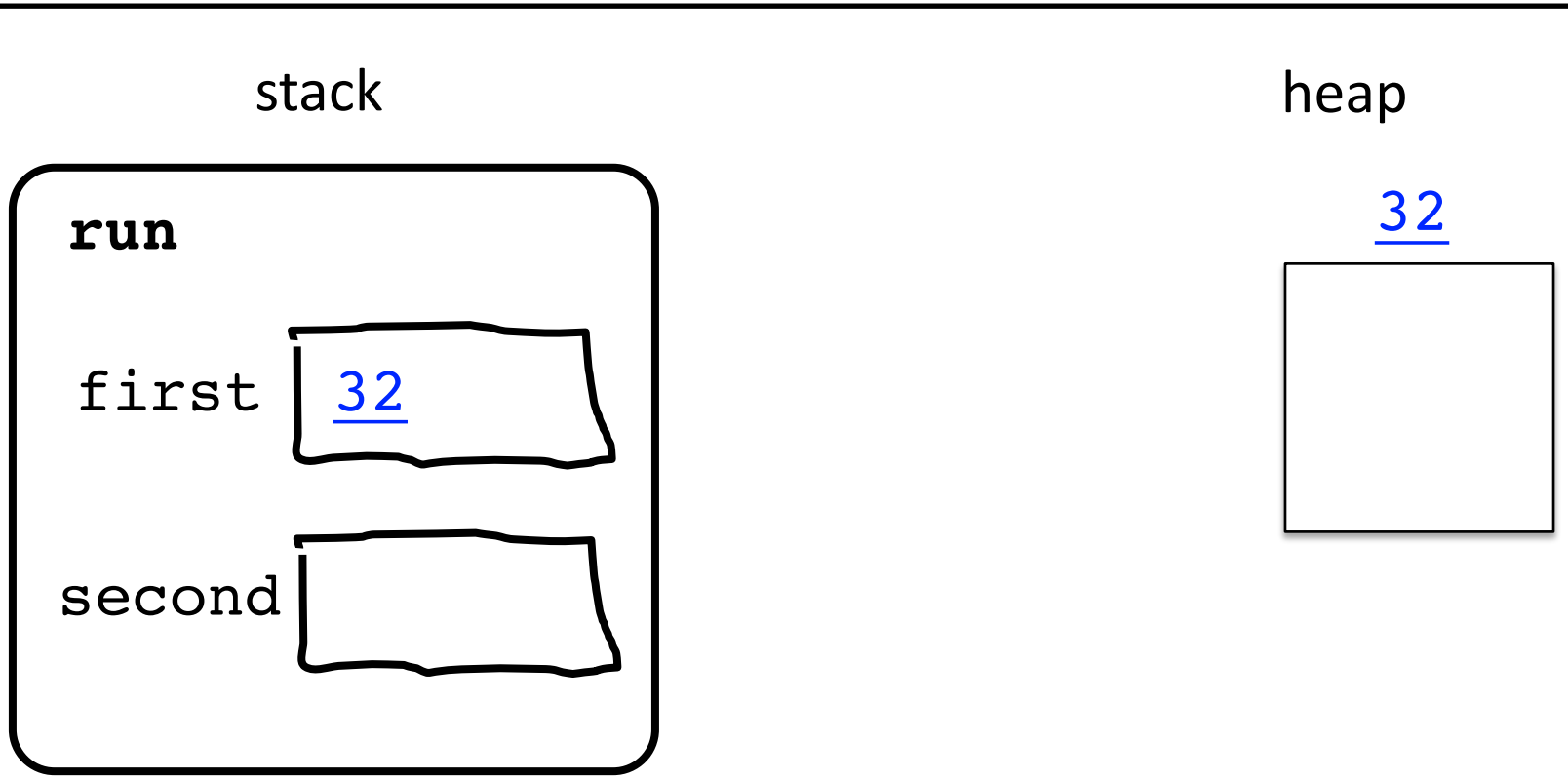
stack



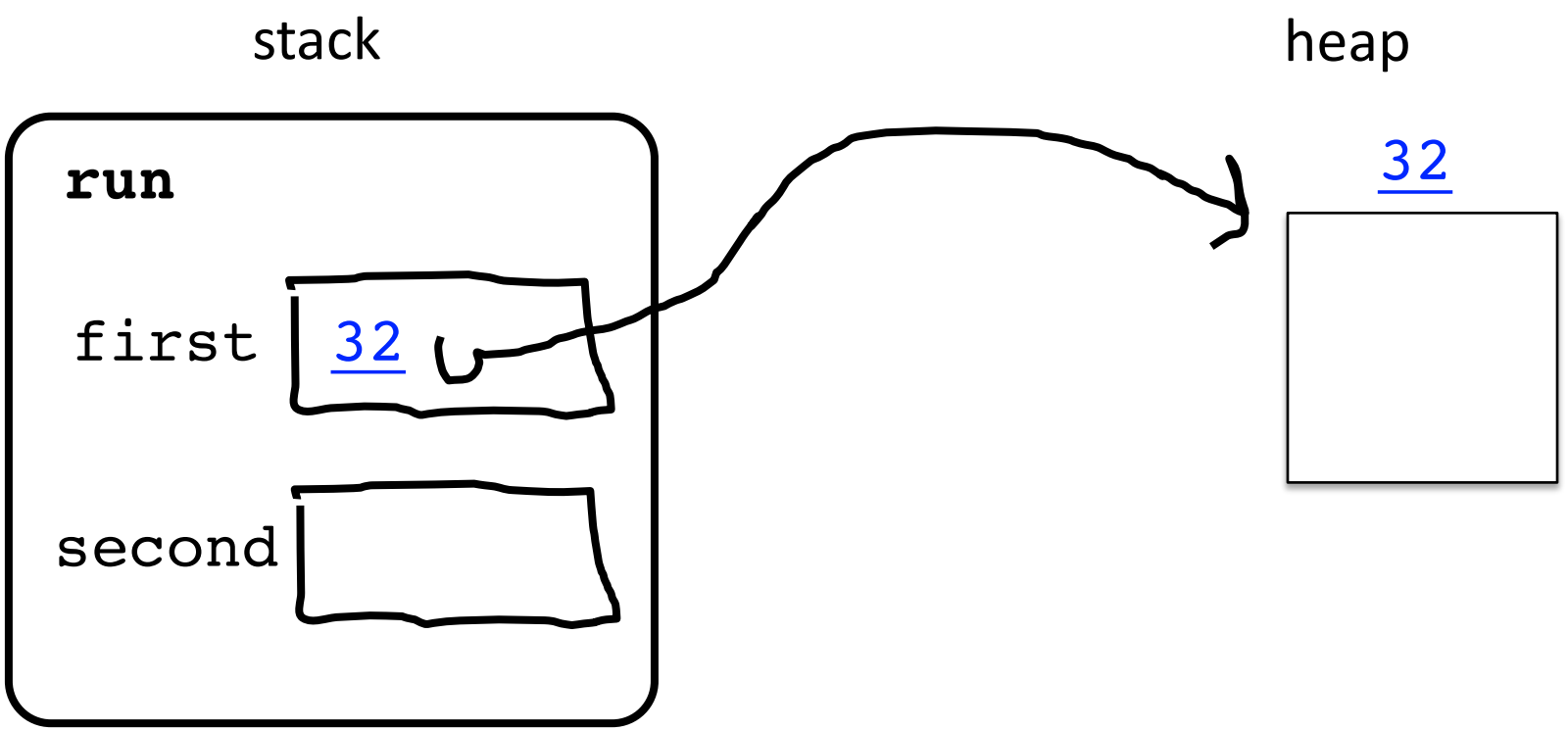
heap



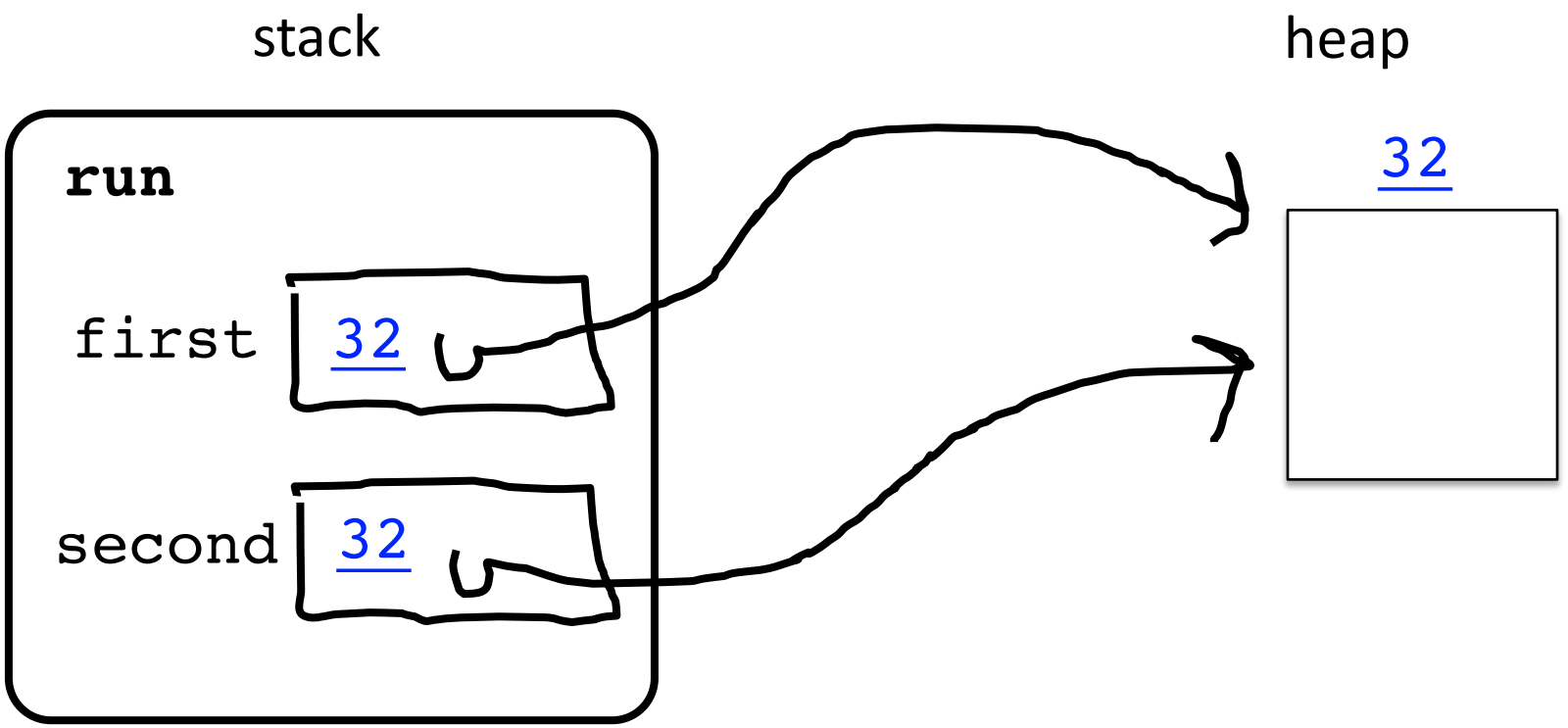
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



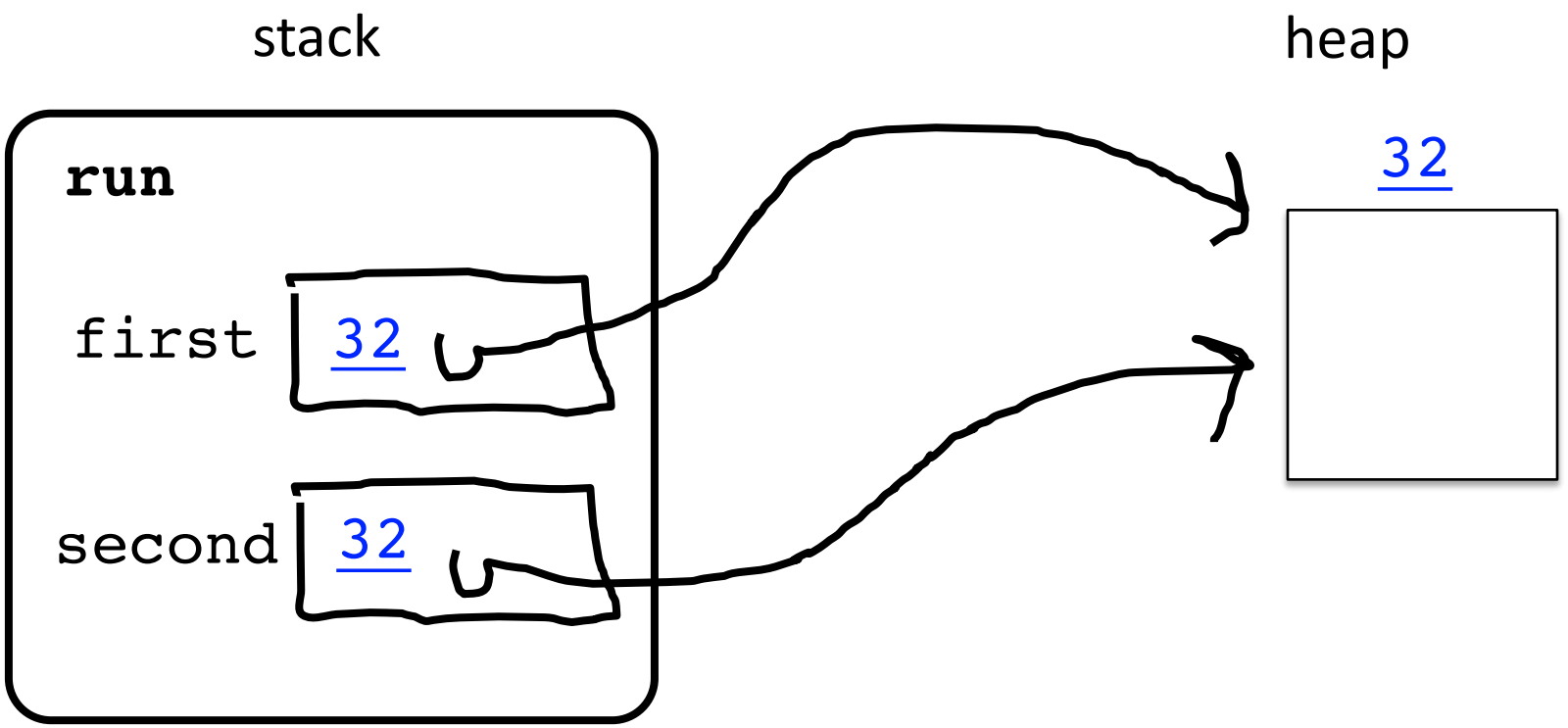
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



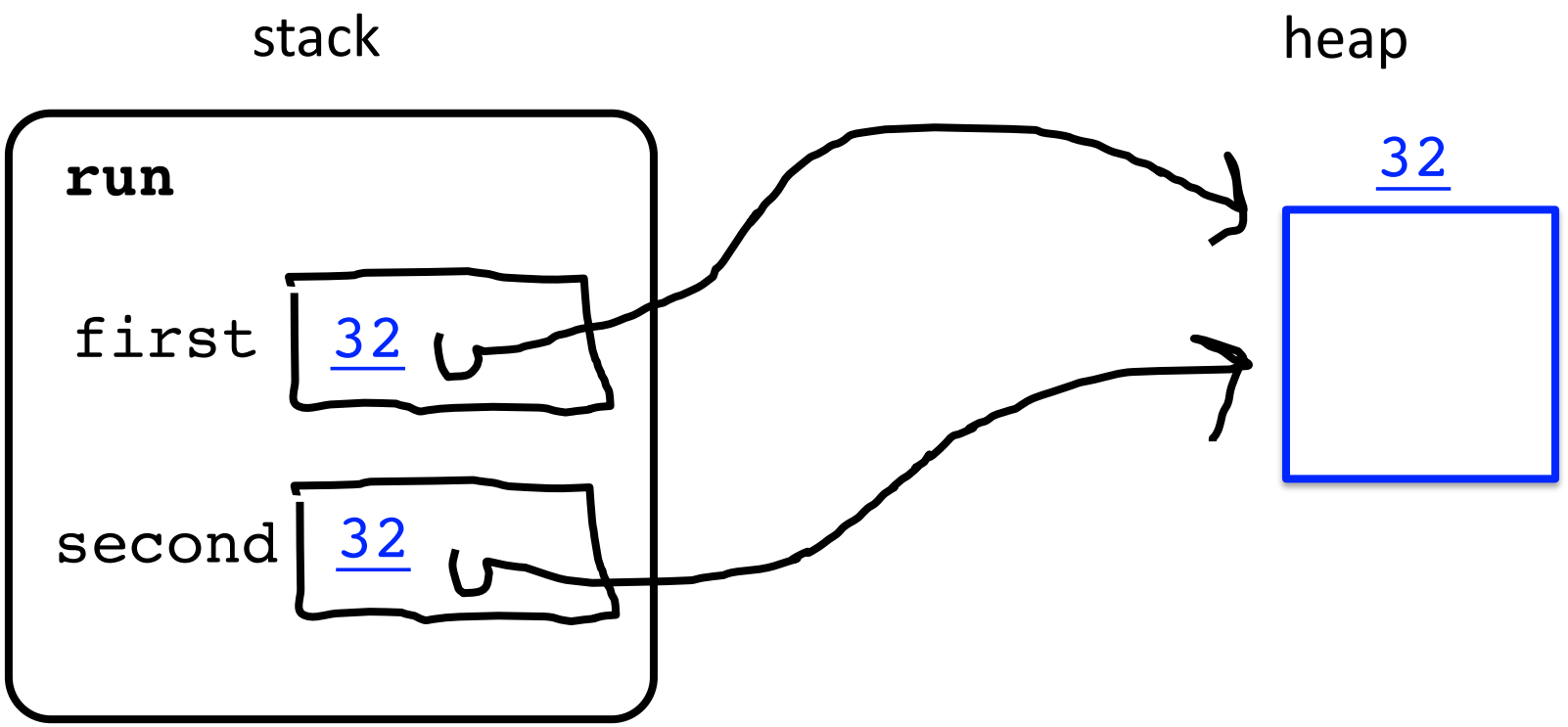
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



#4: when you use the = operator with objects, it copies the *address*

What does an object store?

Objects store addresses
(which are like URLs)

Passing by “Reference”

Primitives pass by value

```
// NOTE: This program is buggy!!

public void run() {
   int x = 3;
   addFive(x);
   println("x = " + x);
}

private void addFive(int x) {
   x += 5;
}
```

* This is probably the single more important example to understand in CS106A

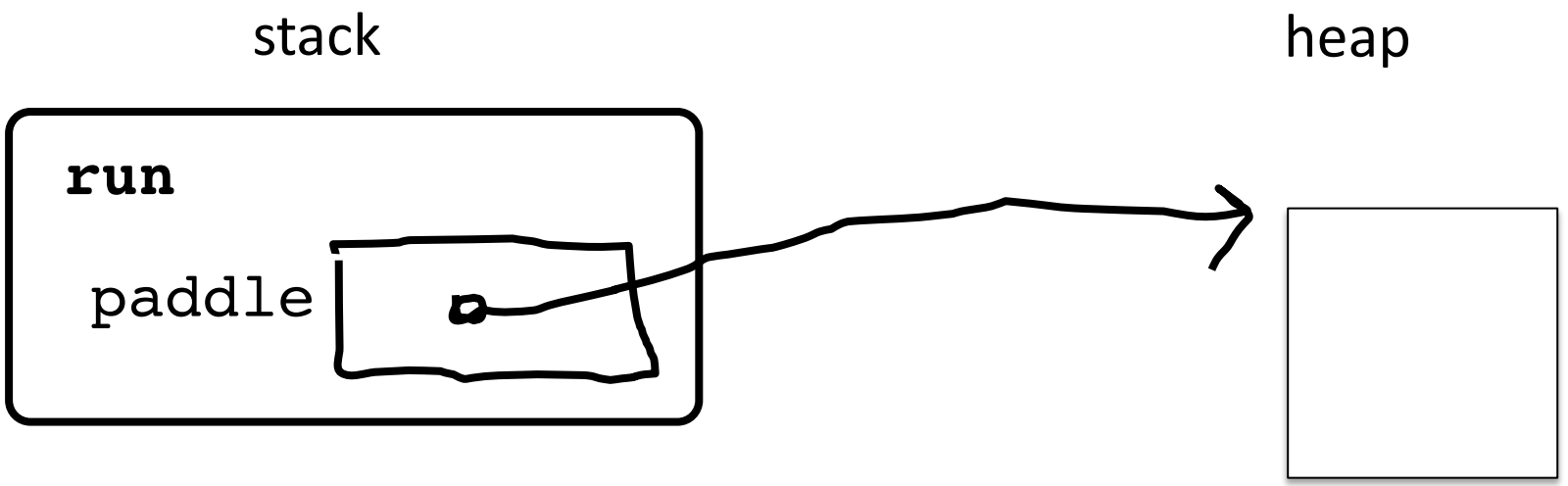
Objects pass by reference

```
// NOTE: This program is awesome!!

public void run() {
  GRect paddle = new GRect(50, 50);
  makeBlue(paddle);
  add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
   object.setColor(Color.BLUE);
   object.setFilled(true);
}
```

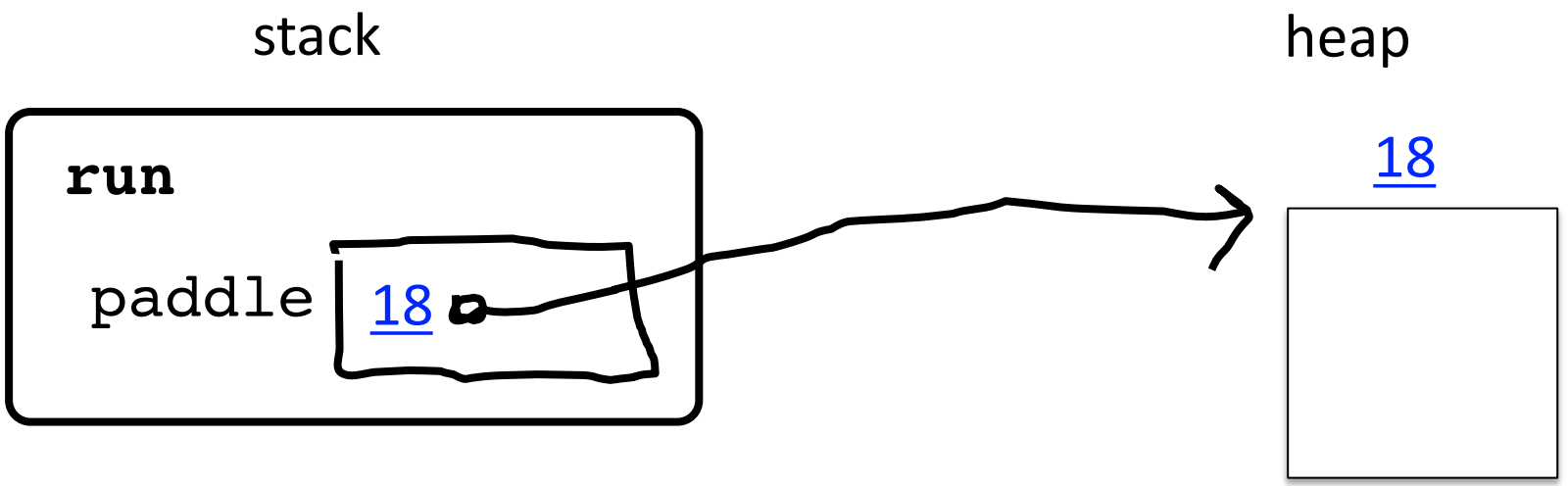
* This is probably the single more important example to understand in CS106A

```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```

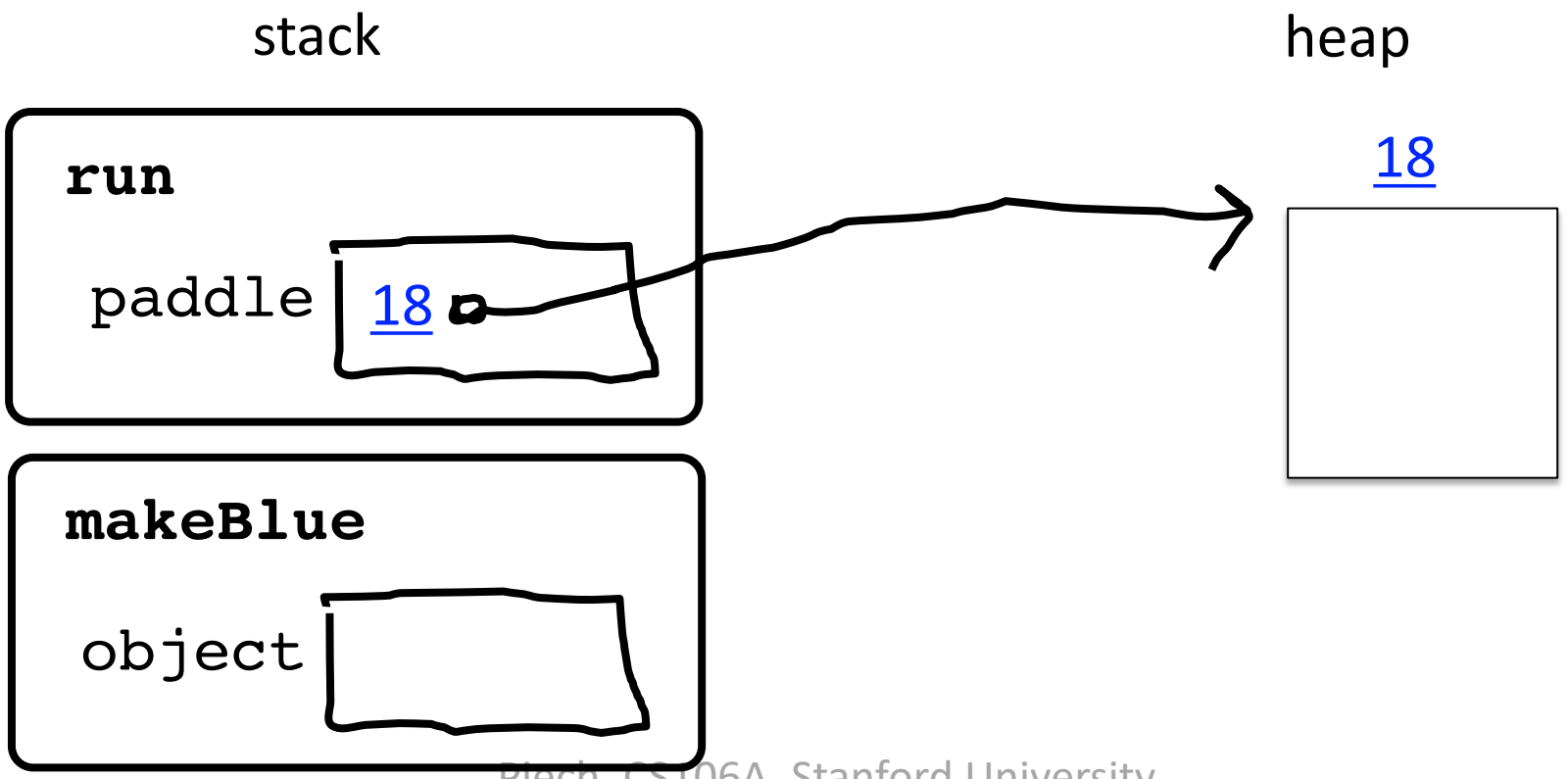

```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



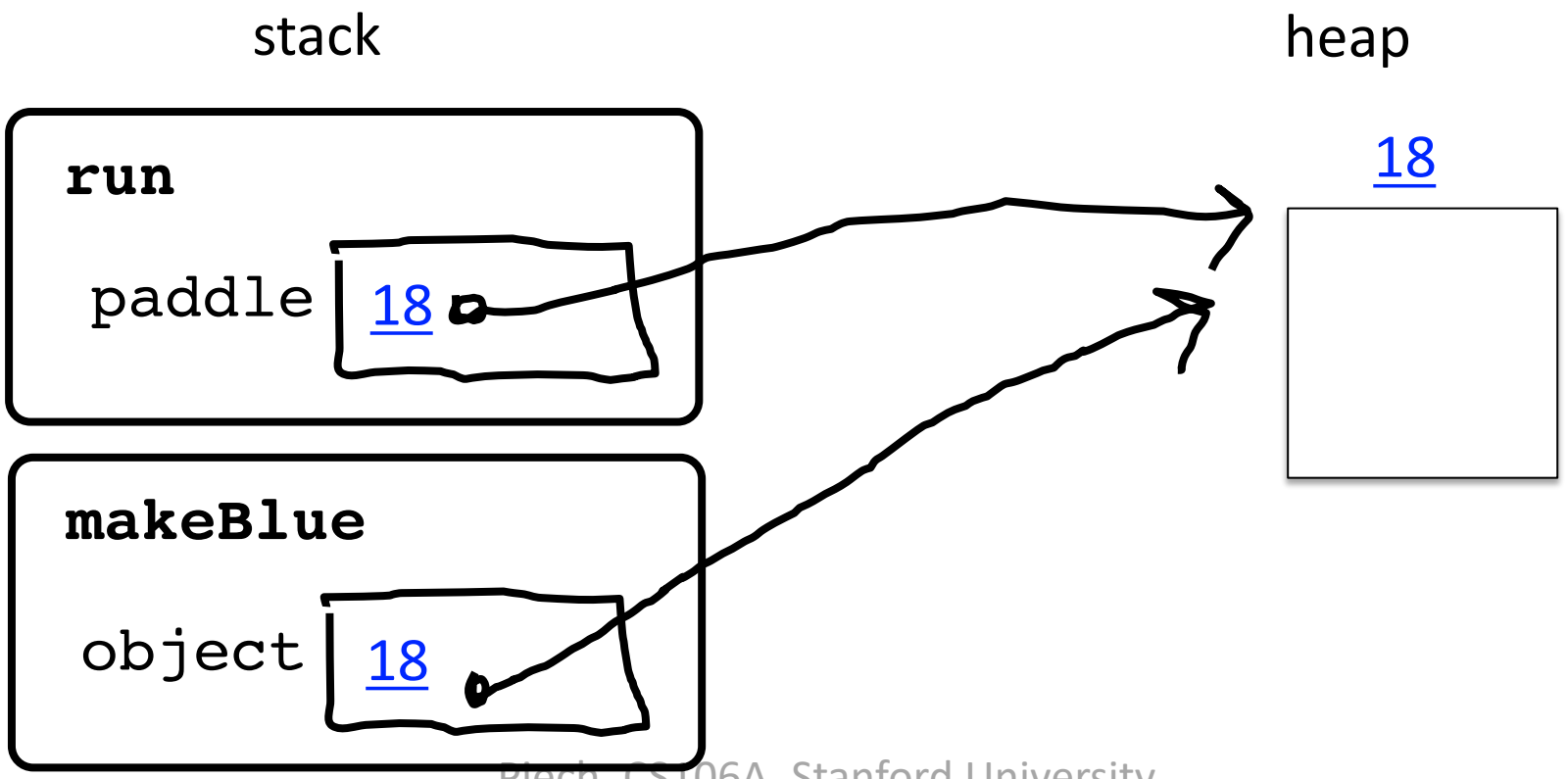
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



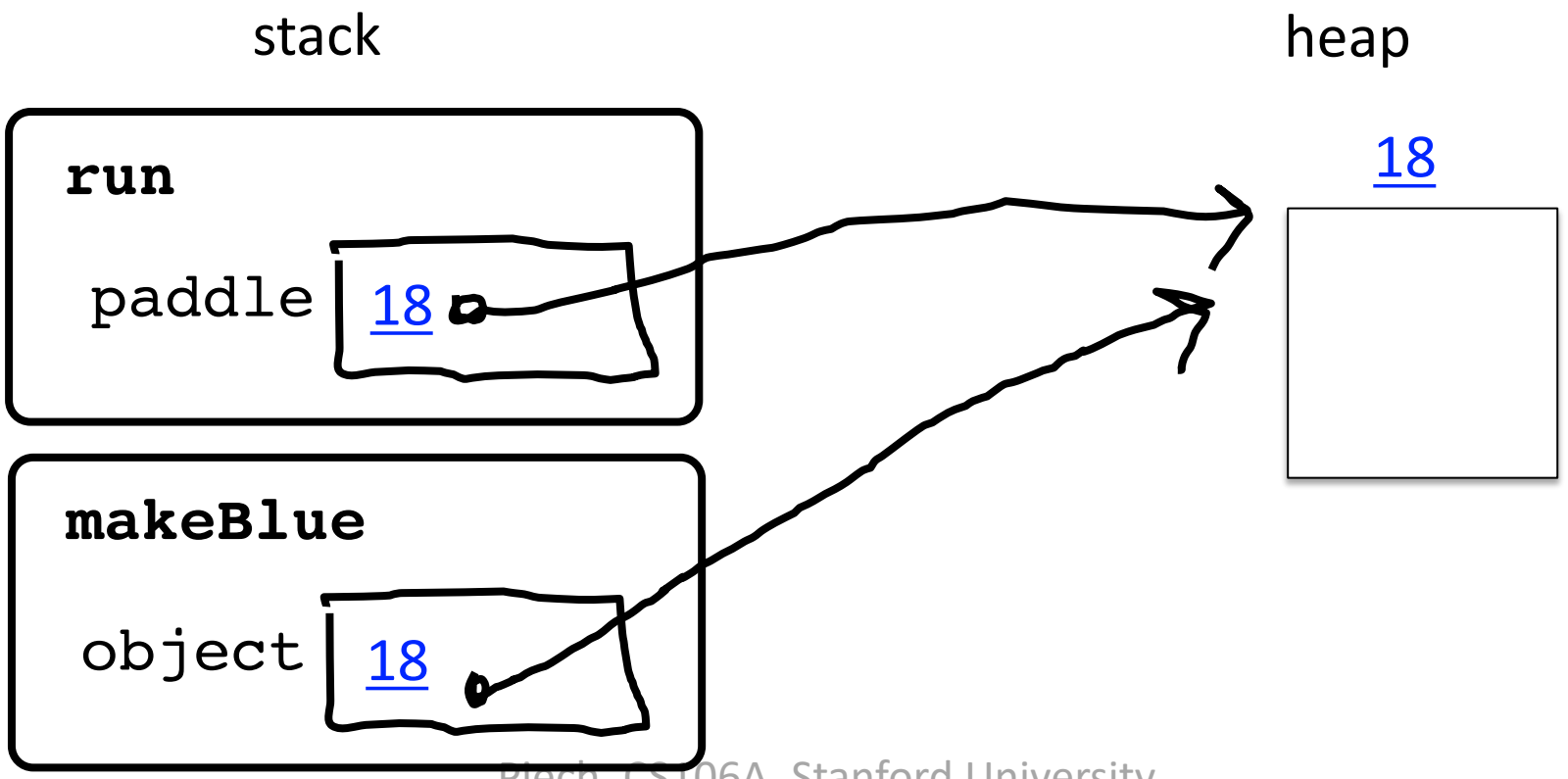
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



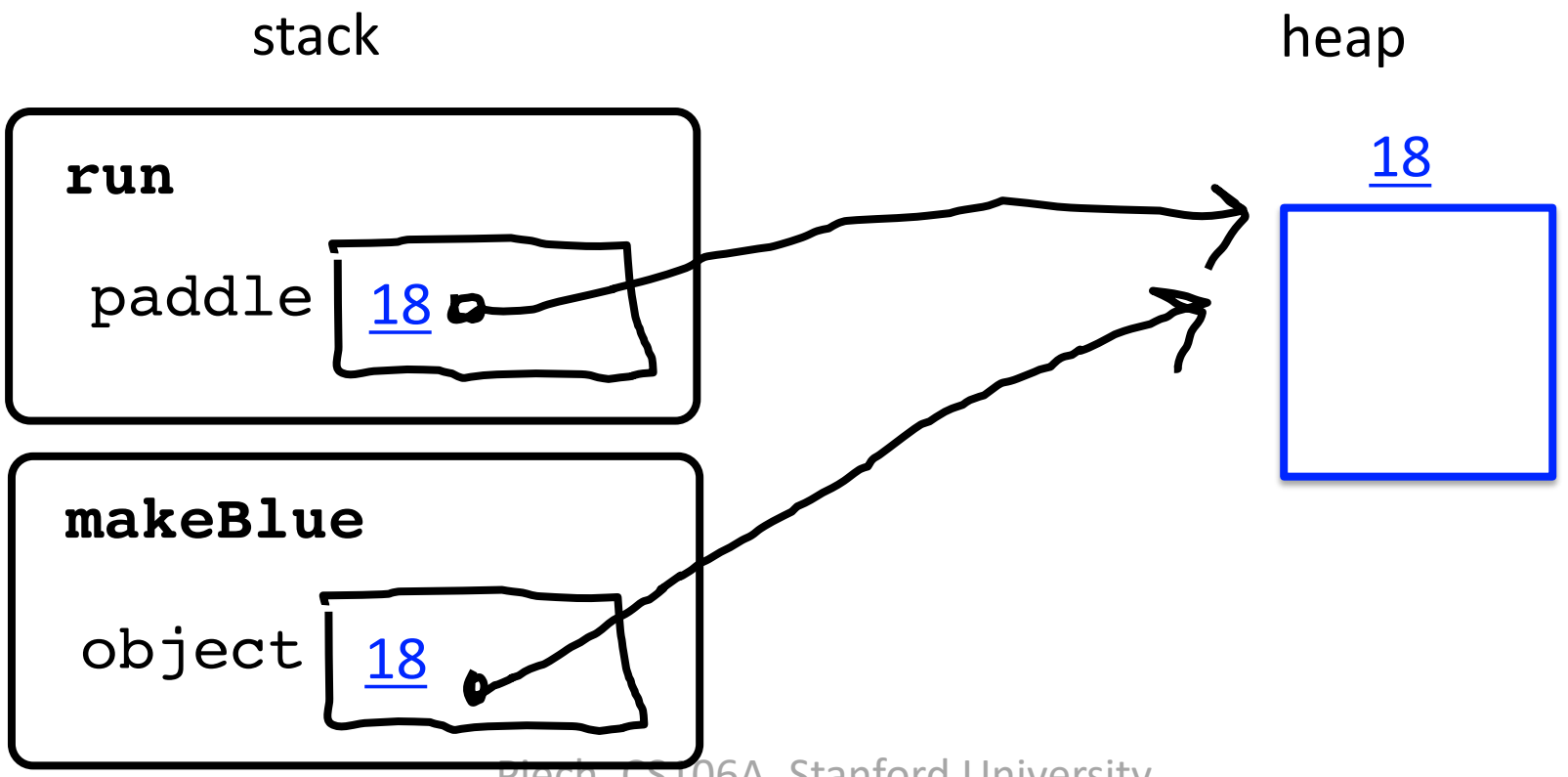
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



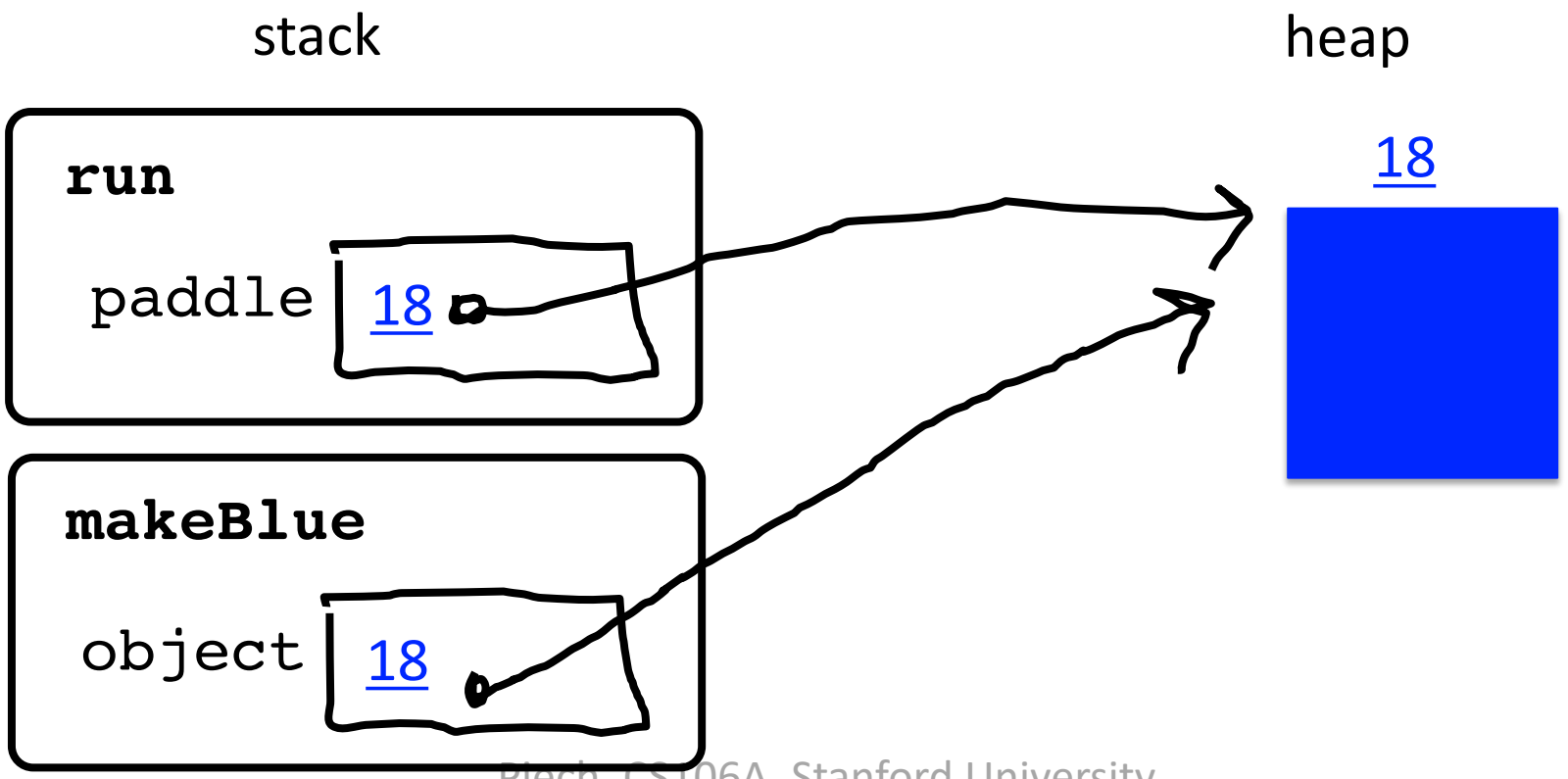
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



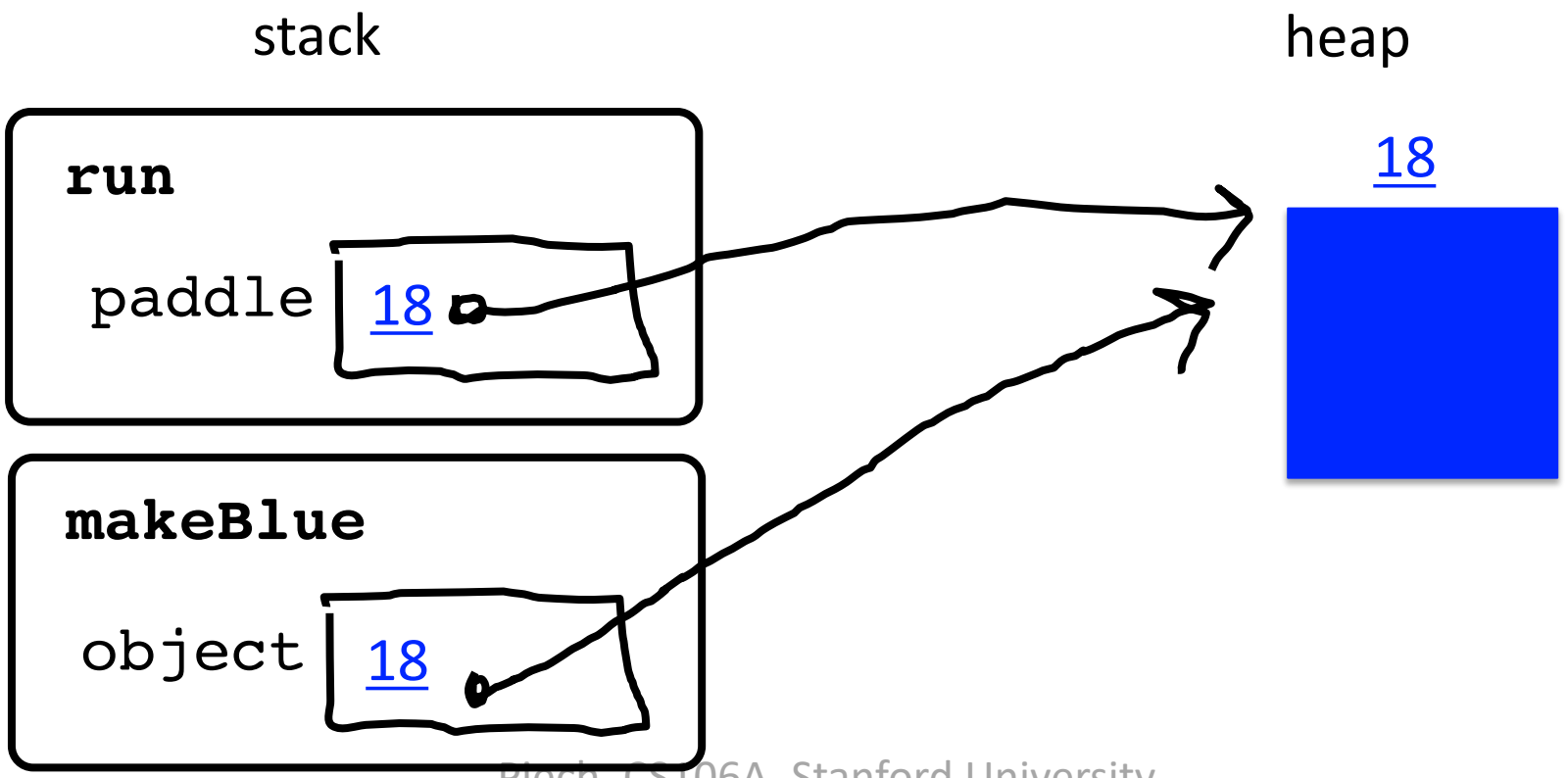
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



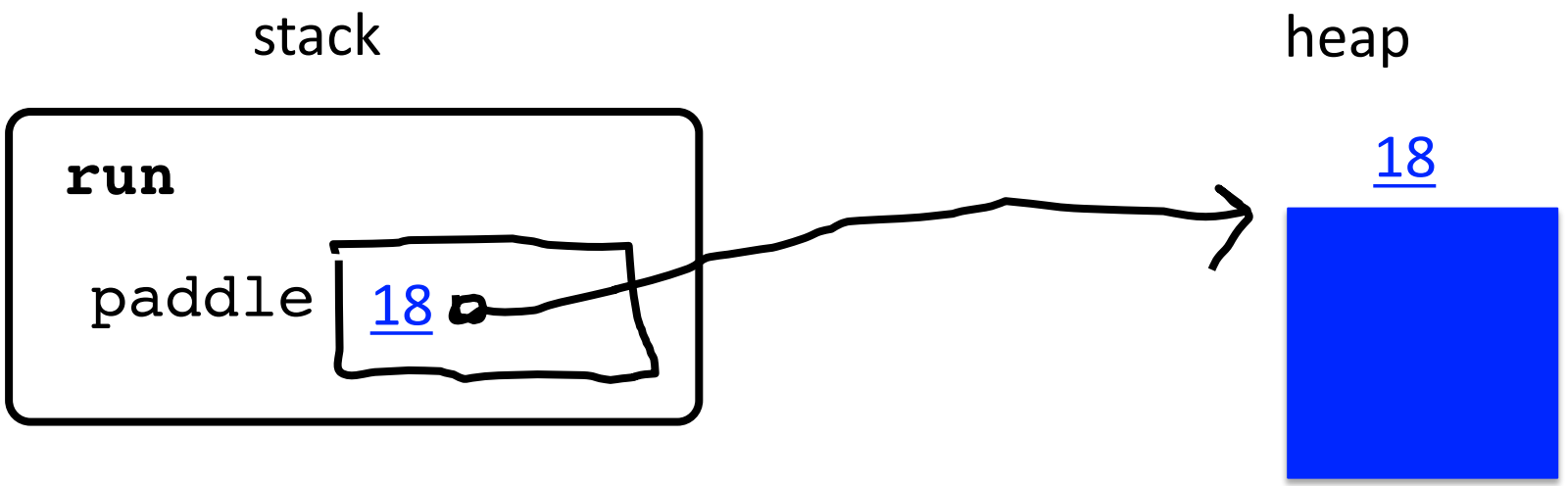
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



#5: when you pass (or return) an object, the address is passed.

Aka reference

What does an object store?

Objects store addresses
(which are like URLs)

Instance Variables

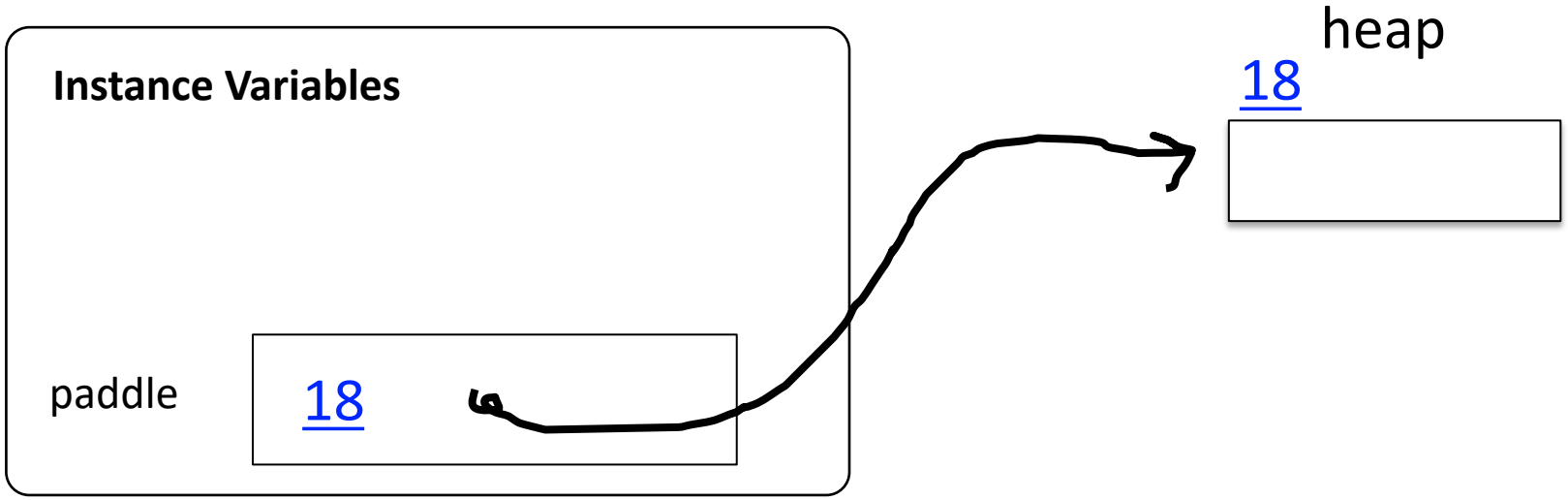
```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```

---

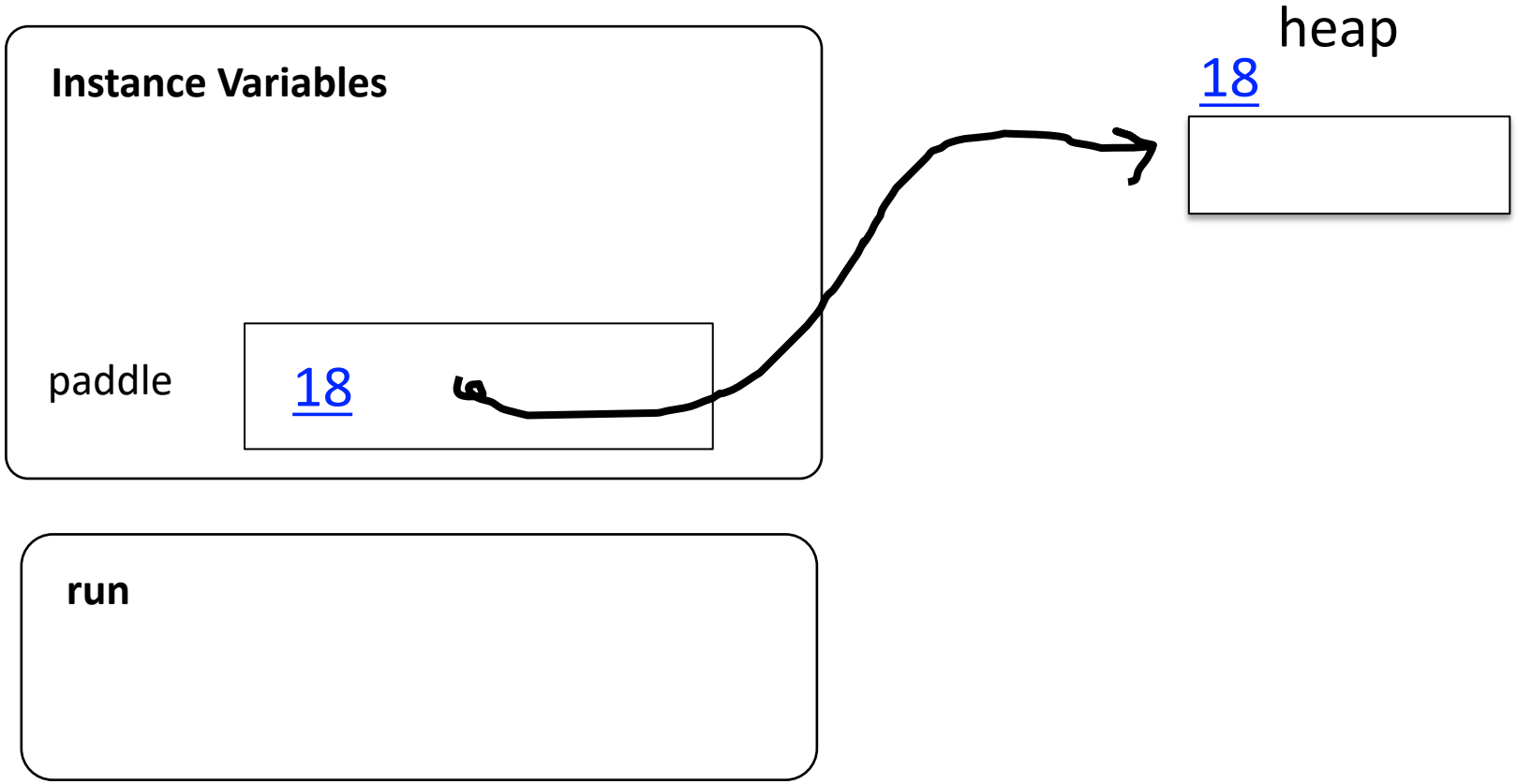
heap

Instance Variables

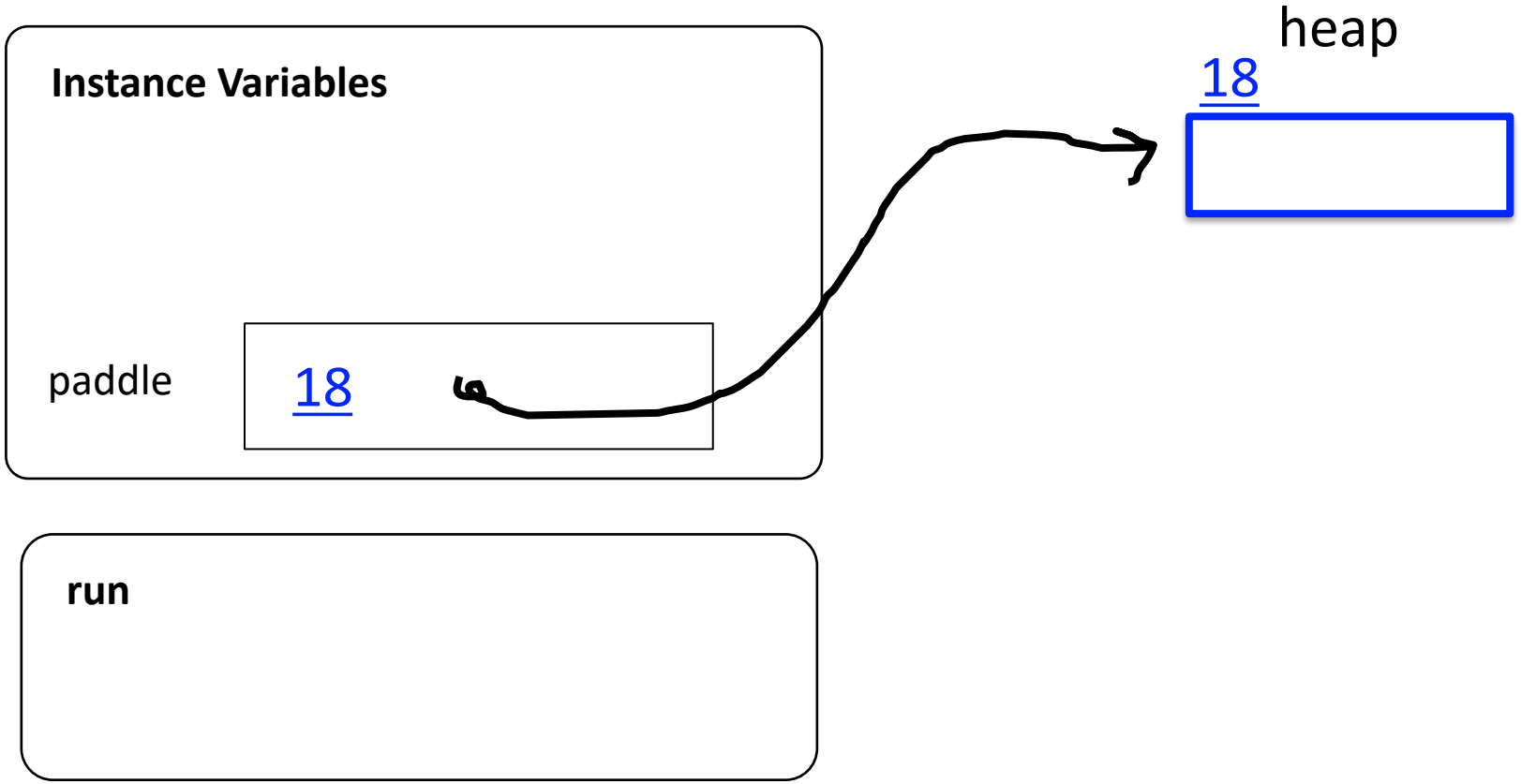
```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```



```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```



```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```



#7: there is space for all instance variables. They are accessible by the entire class

#8: instance variables are initialized before run is called

Common Bug

Question: what does this program do?

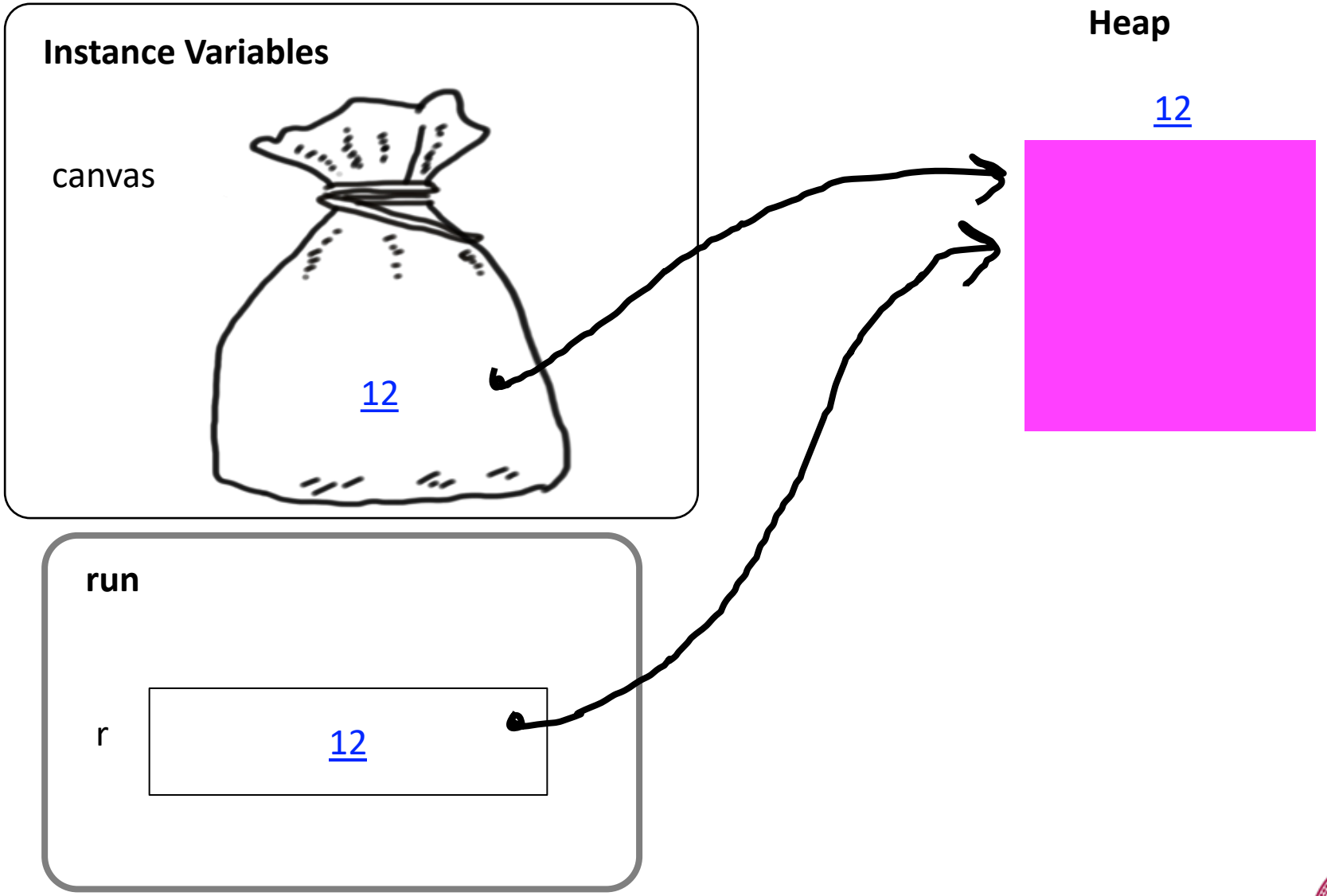
```
GRect paddle = new GRect(getWidth(), getHeight());
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```

Answer: makes a square that is 0 by 0 since **`getWidth`** is called before the screen has been made.

Canvas
(and getElementAt)

```
public class SimpleRect extends GraphicsProgram {

    public void run() {
        GRect r = null;
        r = new GRect(300, 300);
        r.setColor(Color.MAGENTA);
        add(r, 0, 0);
        addMouseListeners();
    }

    public void mousePressed(MouseEvent e) {
        GObject obj = getElementAt(1, 1);
        remove(obj);
    }

}
```


Canvas



```
public class SimpleRect extends GraphicsProgram {

    public void run() {
        GRect r = null;
        r = new GRect(300, 300);
        r.setColor(Color.MAGENTA);
        add(r, 0, 0);
        addMouseListeners();
    }

    public void mousePressed(MouseEvent e) {
        GObject obj = getElementAt(1, 1);
        remove(obj);
    }

}
```

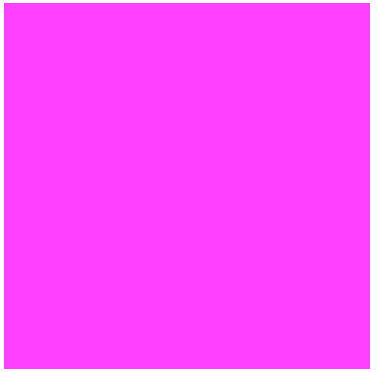

Canvas

Instance Variables

canvas

Heap

12



mousePressed

e | 94

obj | 12

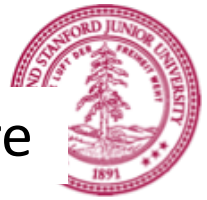
94

x = 72
y = 94
time = 192332123

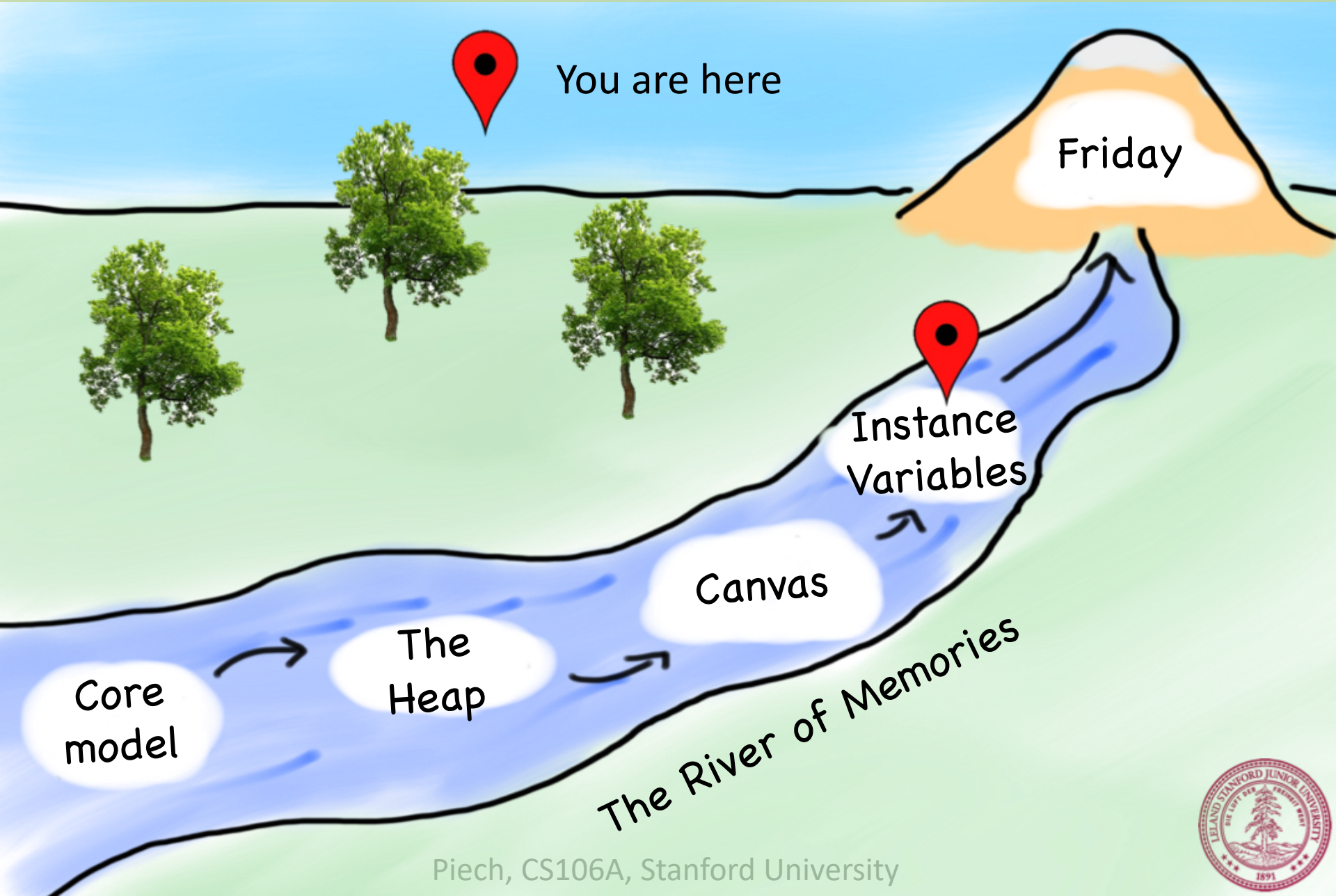
#6: graphics programs all have an instance variable "canvas" which keeps track of the objects on the screen

```
public void run() {
    GRect first = new GRect(50, 50);
    GRect second = first;
    add(first, 0, 0);
    add(second, 20, 20);
}
```

Intentionally left blank so that we can fill it in during lecture



Today's Route



#9: for objects == checks if the variables store the same address

Recall the start of class?

Who thinks this prints **`true`**?

```
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```


Who thinks this prints **`true`**?

```
public void run() {
    int x = 5;
    int y = 5;
    println(x == y);
}
```


Who thinks this prints **`true`**?

```
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```


What does an object store?

Objects store addresses
(which are like URLs)

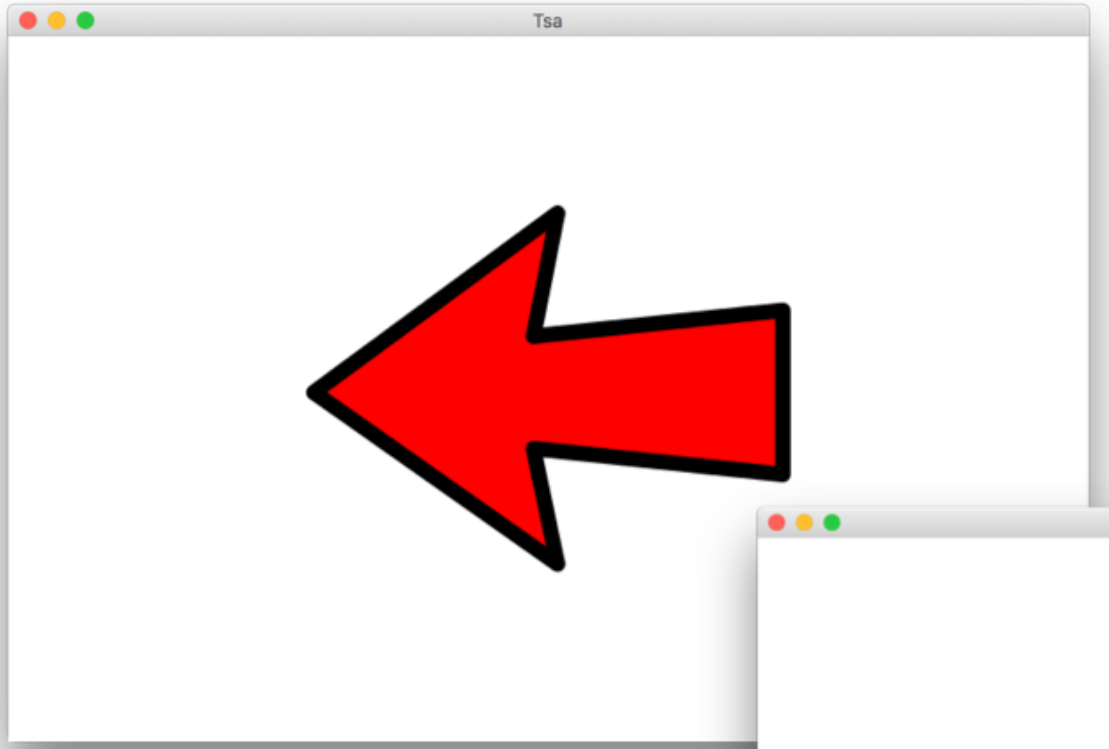
Primitives vs Objects

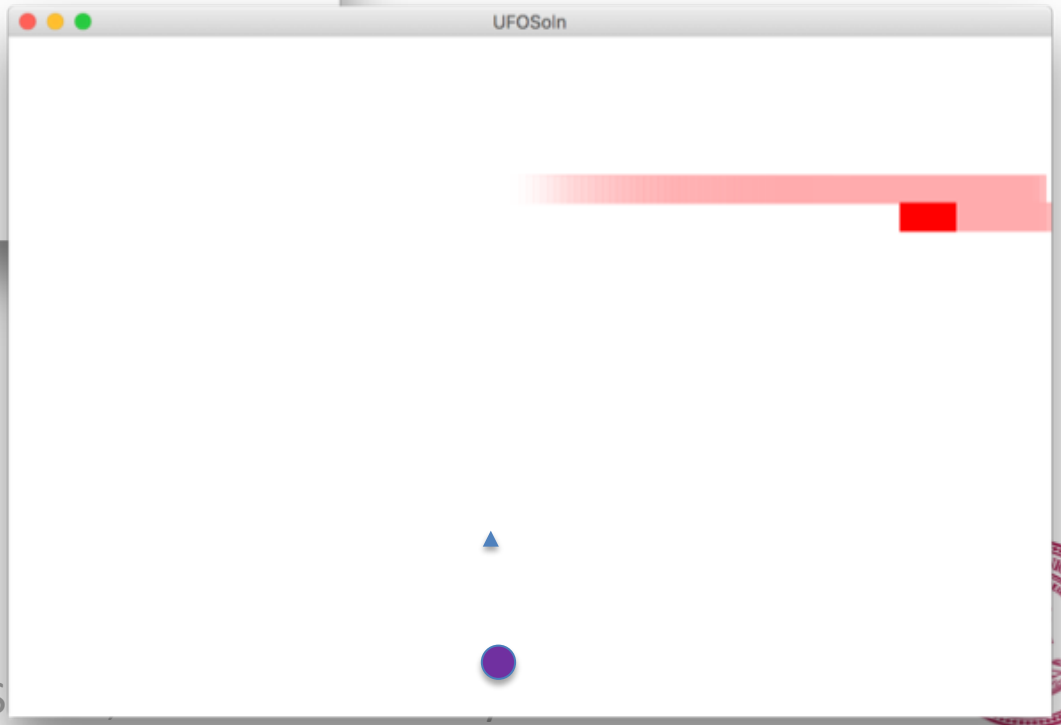
- **Primitives** store their *actual value* in their variable box. You can compare values with == and !=, and the original does not change when passed as a parameter and changed.
- **Objects** store their *address* in their variable box. You can't compare properties of an object via ==, and the original *does* change when passed as a parameter and changed.
- **Primitives** are *passed by value*, **Objects** are *passed by reference*

Learning Goals

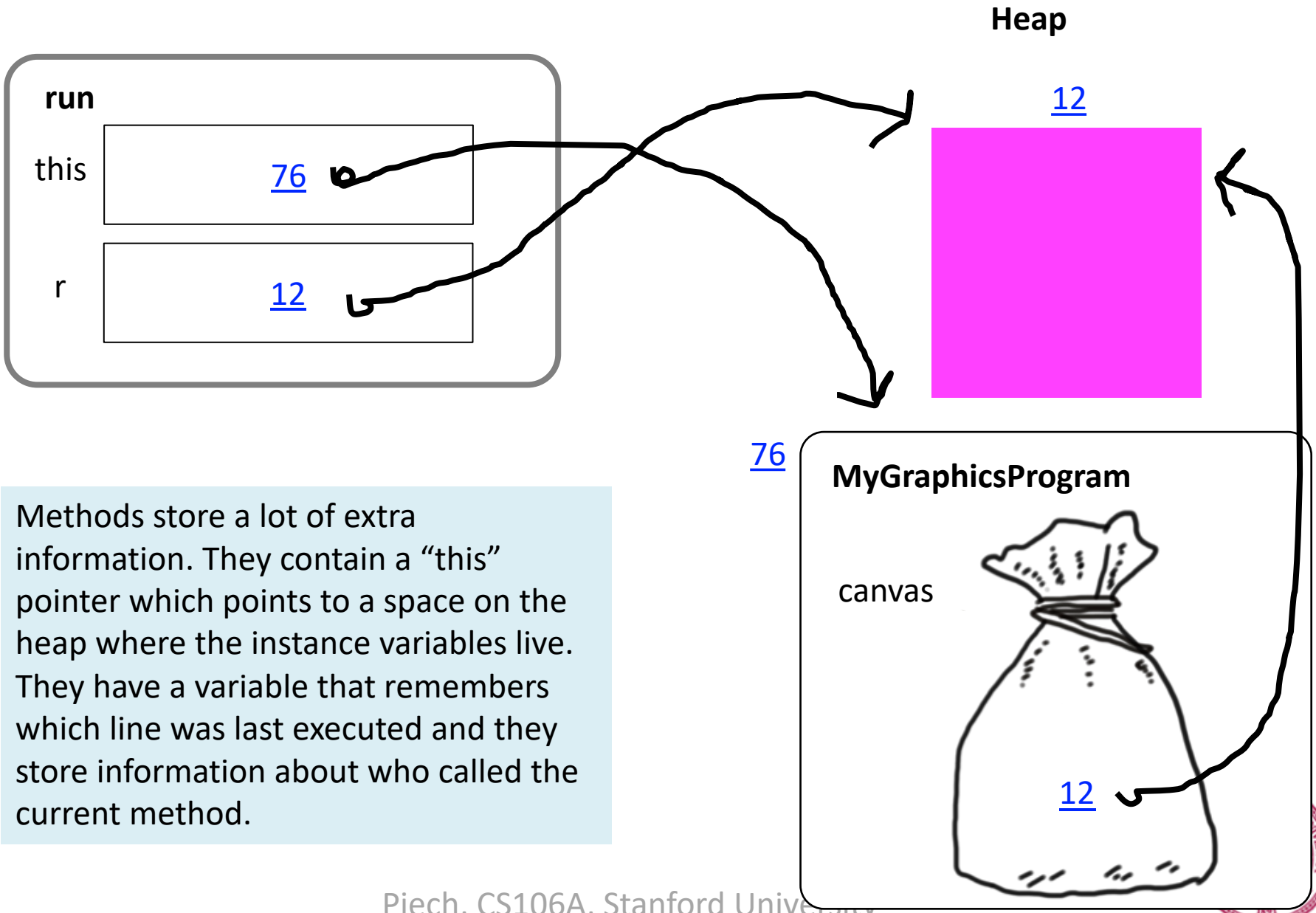
1. Be able to trace memory for objects

Revisit our Programs





Beyond CS106A



Methods store a lot of extra information. They contain a "this" pointer which points to a space on the heap where the instance variables live. They have a variable that remembers which line was last executed and they store information about who called the current method.