Go ahead and get today's code in Eclipse as shown on next few slides…

COMP110

Jump Around

Fall 2015
Sections 2 & 3
Sitterson 014

November 19th, 2015

🐑

Kris Jordan
kris@cs.unc.edu
Sitterson 238

Classroom Materials

https://github.com/comp110/materials

Classroom Materials

1. Open Eclipse, File > Import…
2. Git > Projects from Git > Clone URI
3. URI is **https://github.com/comp110/materials**
4. MACs: Install Command Line Utilities OK
5. Next, Next, Browse to your COMP110 Projects Directory, Next
6. Select Import Existing Projects, Next, Finish
7. Open
    1. 110 Class Materials > src > com.comp110.com.lecture20
    2. FindTheCash.java
8. Try Running! If you have errors, see next slide…
9. ***Check in on PollEverywhere!*** pollev.com/comp110

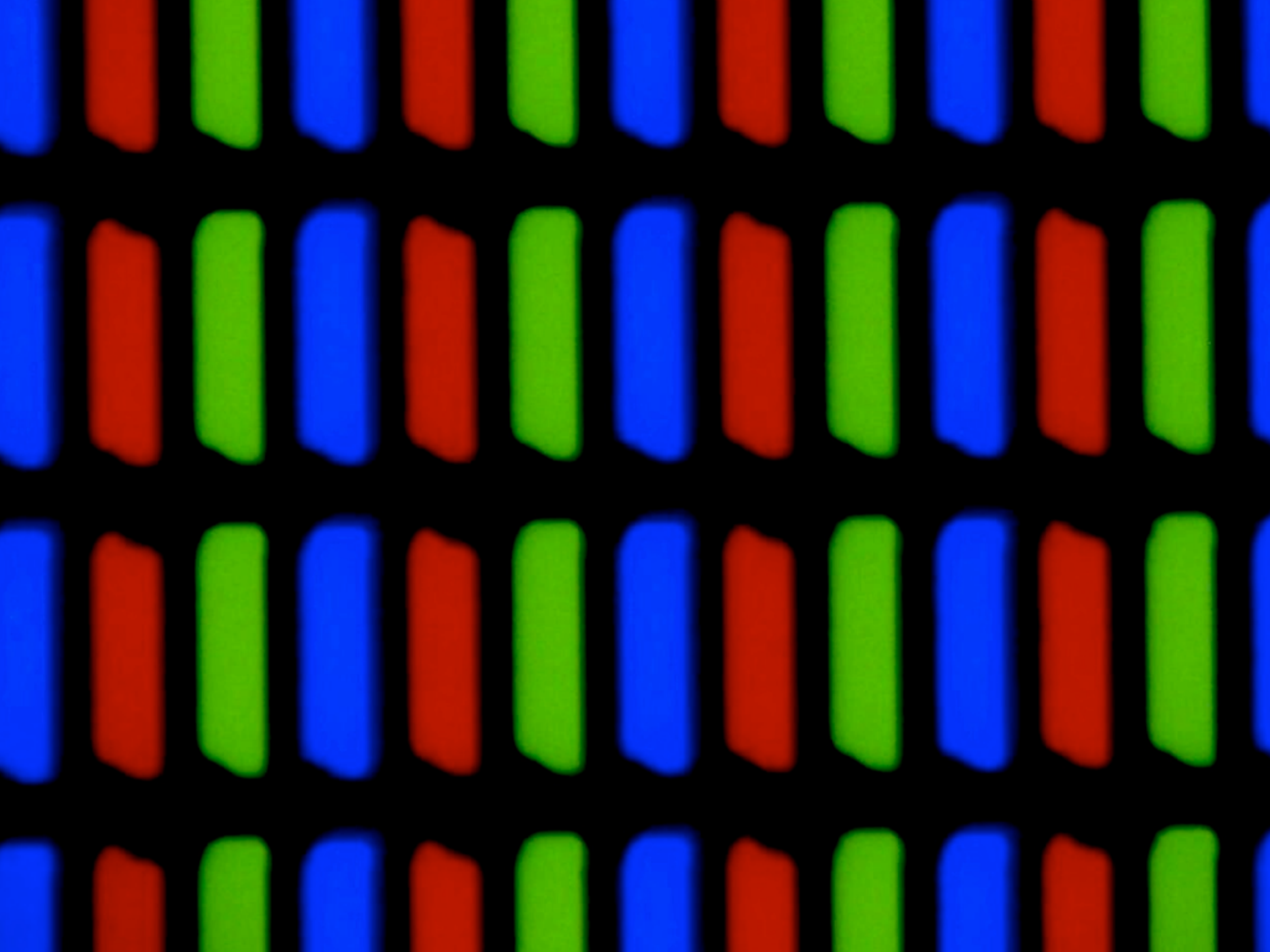
Classroom Materials: Errors?

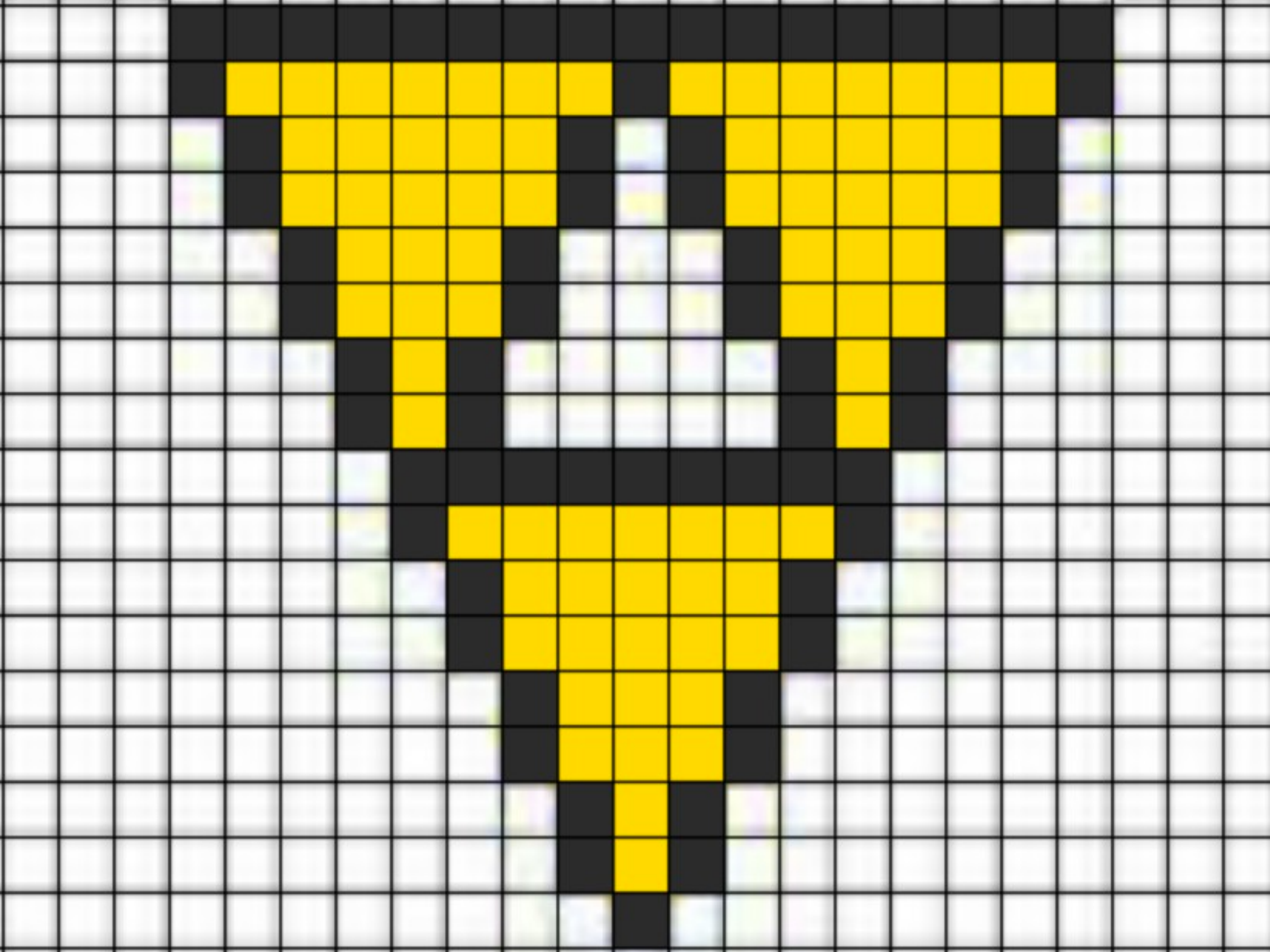
If Eclipse won't let you run:

1. Right click on '110 Class Materials' project
2. Build Path > Add Libraries…
3. Select 'JRE System Library'
4. Next > Finish
5. Try running again

Announcements

- Assignment 4
  - Part B - Effects and Filters
  - http://comp110.com/assignments/effects-and-filters
  - Demo





What is the output? (1)

```
public static void main(String[] args) {

  double  dollars = 20.0;

  // Going to the caffeine store
  boolean isCaffeinated;

  if(dollars >= 2.50) {
    dollars = dollars - 2.50;
    isCaffeinated = true;
  } else {
    isCaffeinated = false;
  }

  if(isCaffeinated) {
    System.out.print(":) ");
  } else {
    System.out.print(":( ");
  }
  System.out.println("Dollars left: " + dollars);

}
```

PollEv.com/comp110

What is the output? (2)

```
public static void main(String[] args) {

  double  dollars = 20.0;

  System.out.println("Going to the caffeine store...");
  boolean isCaffeinated = buyCoffee(dollars);

  if(isCaffeinated) {
    System.out.print(":) ");
  } else {
    System.out.print(":( ");
  }
  System.out.println("Dollars left: " + dollars);

}

static boolean buyCoffee(double dollars) {
  if(dollars >= 2.50) {
    dollars = dollars - 2.50;
    return true;
  }
  return false;
}
```

PollEv.com/comp110

Call Stack Overview

- When your program begins, the **call stack** has one **frame** for *main()*
- *So far*, when we've drawn boxes for variables in memory, it's been in this frame.
- A **frame** is a segment of memory storing the method's **parameters** and all **variables** declared within the method.
- When you *call a method*, a frame for the **callee** is "*pushed*" onto the stack.
- When you *return from a method* to its **caller**, that frame "pops" off the stack.

Call Stack on Pen & Paper

A trick and a treat for you... 

```
public static void main(String[] args) {
  System.out.println( scary(3) );
}
```

```
static String scary(int howScary) {
  if(howScary > 1) {
    return scary(howScary - 1) + "O";
  } else {
    return "B";
  }
}
```

Recursion

A Recursive Selfie

- Pull out your phones
- Open up your camera app and set it to selfie mode
  - If you don’t have a phone / selfie mode: photobomb
- Pair up
- Turn your phone toward your neighbor’s and try to get an “infinite mirror” effect photo
- Challenge: one at a time, make it a selfie, too
- Keep this photo for Part B of Assignment 4!

So that's recursion in a nutshell...

It's a little awkward and hard to get right.

It's *really* cool when it works.

It'll stretch your mind.

What is recursion?

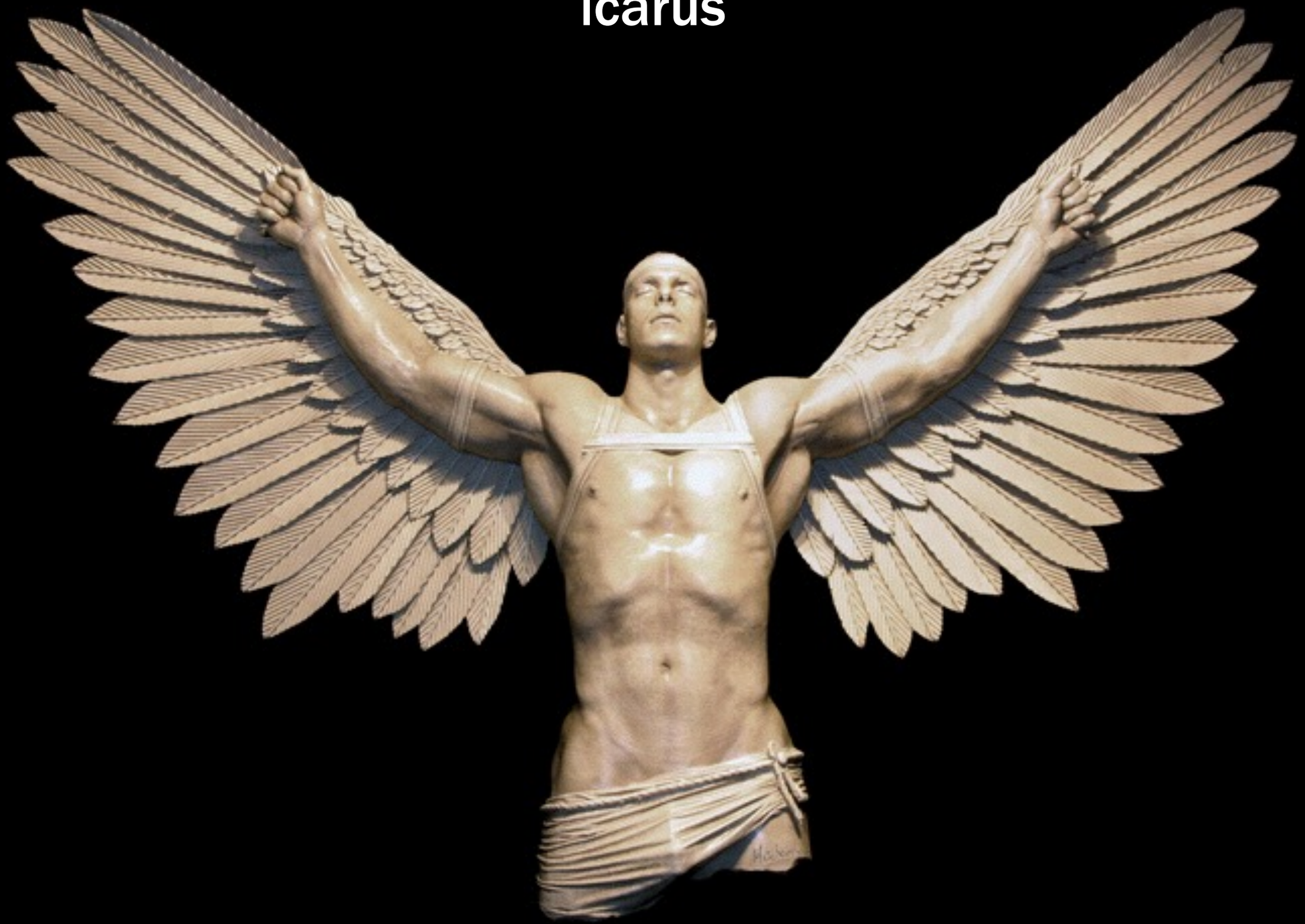
- It's the idea that something can be self-referencing
- In Computer Science it shows up commonly in 2 ways:
  - Recursive Data - A Data Type can *refer* to itself
    - Example: organization chart of people
    - A Person's "boss" is just another Person
  - Recursive Methods - A Methods can *call* itself
- You'll most commonly use recursive methods *with* recursive data types, but you can use either independently

Recursion in 110

A *gentle* introduction to the concept

with some hands on exploration.

Let's write a recursive method...
icarus



Let's write a recursive method...

- Open RecursiveExamples.java
- Write a **public static void** method named **icarus**
- The icarus method should println "MUST FLY HIGHER" and then call the icarus method again from within itself.
- Try running. Let icarus fly toward that Sun...

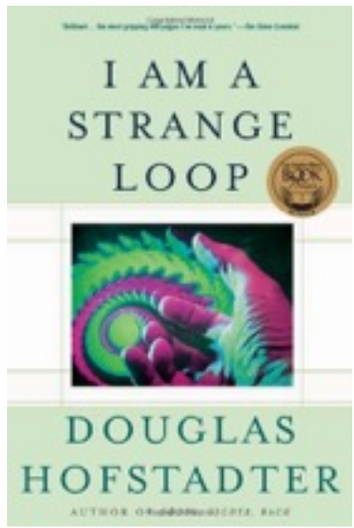
So what's happening?

- Remember the call stack?
- Each time we call a method a new frame gets added to the top of the call stack.
- Here we're calling and calling and calling the icarus method
- When the stack grows too tall, we run out of memory and crash
- This is a **Stack Overflow**

How do we prevent Stack Overflows with recursive methods?

- What differentiates an infinite loop from a non-infinite loop?

- Recursive methods are like "Strange Loops"



We need to reach some condition at which point we'll stop recurring.

This is called the **base case.**

```
int aRecursiveMethod(int n) {
  if(base case condition) {
    return 1;
  } else {
    return aRecursiveMethod(n - 1);
  }
}
```

Every Recursive Method *Needs* a Base Case

- The base case is the end of a recursive method call
- Once the base case is reached, we pop a frame off the stack and return to the method from whence we came
- Think of the base case as similar to the boolean expression in a loop: it is a condition that ends the repetition

Aside: Loops & Recursive Methods

- Anything you can express with a loop you can express with recursive method

- Some programming languages don't have loops and you have to use recursion for *everything* (LISP, Scheme, Haskell)

- Learning one of these "functional" languages after learning Java will stretch your mind

Let’s write Factorial!

What is 3!

4!

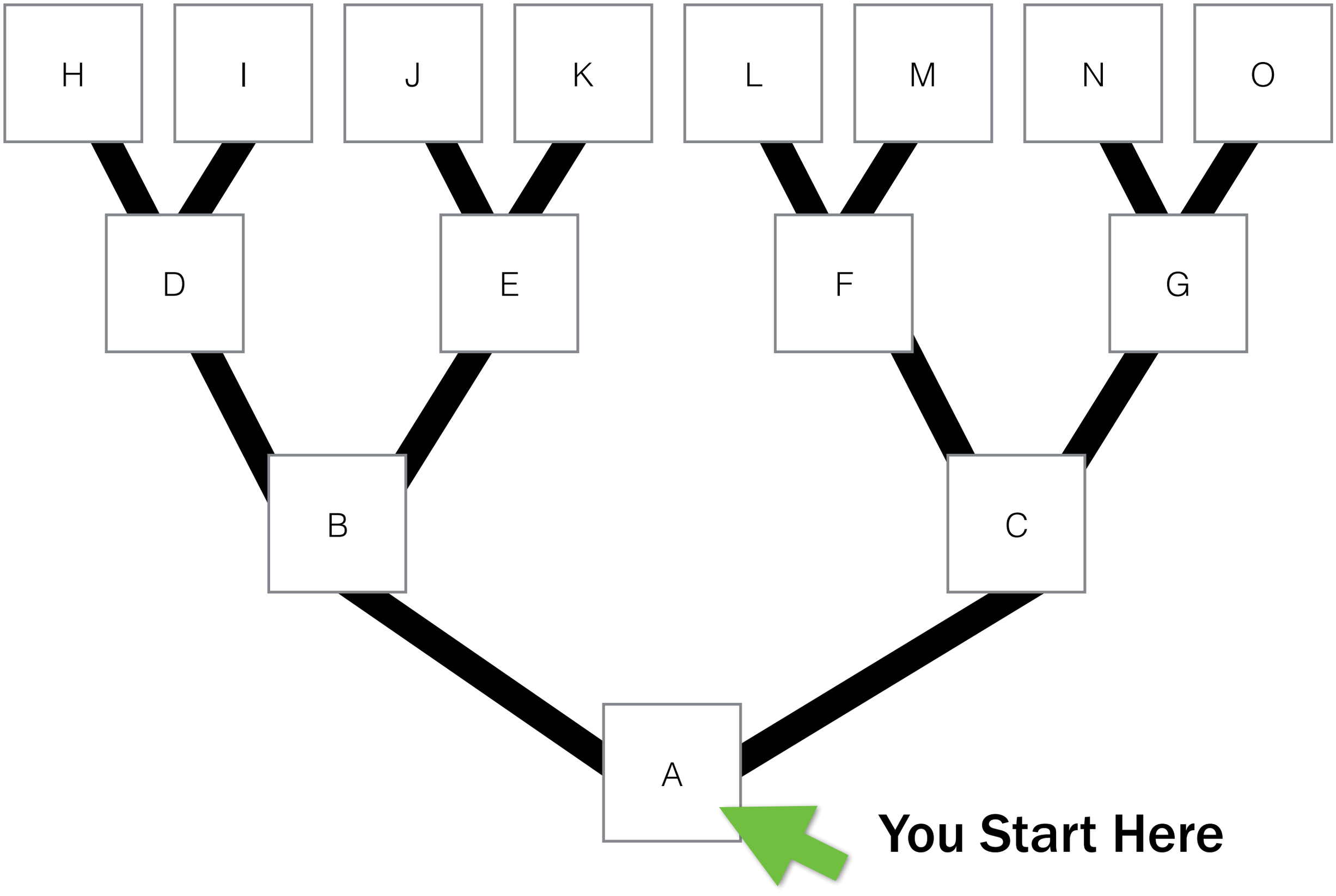
5!

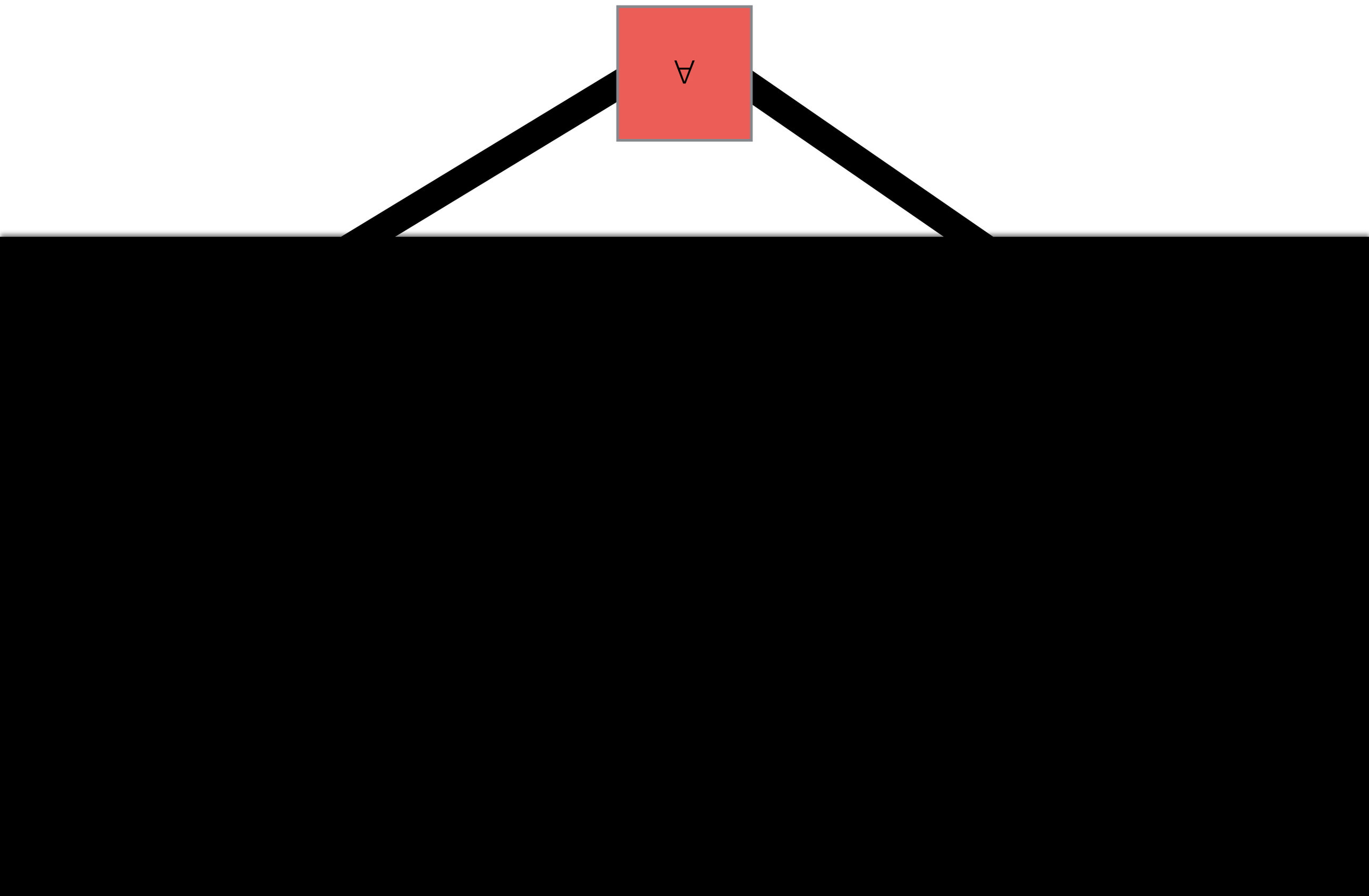
Changing Gears to a Game

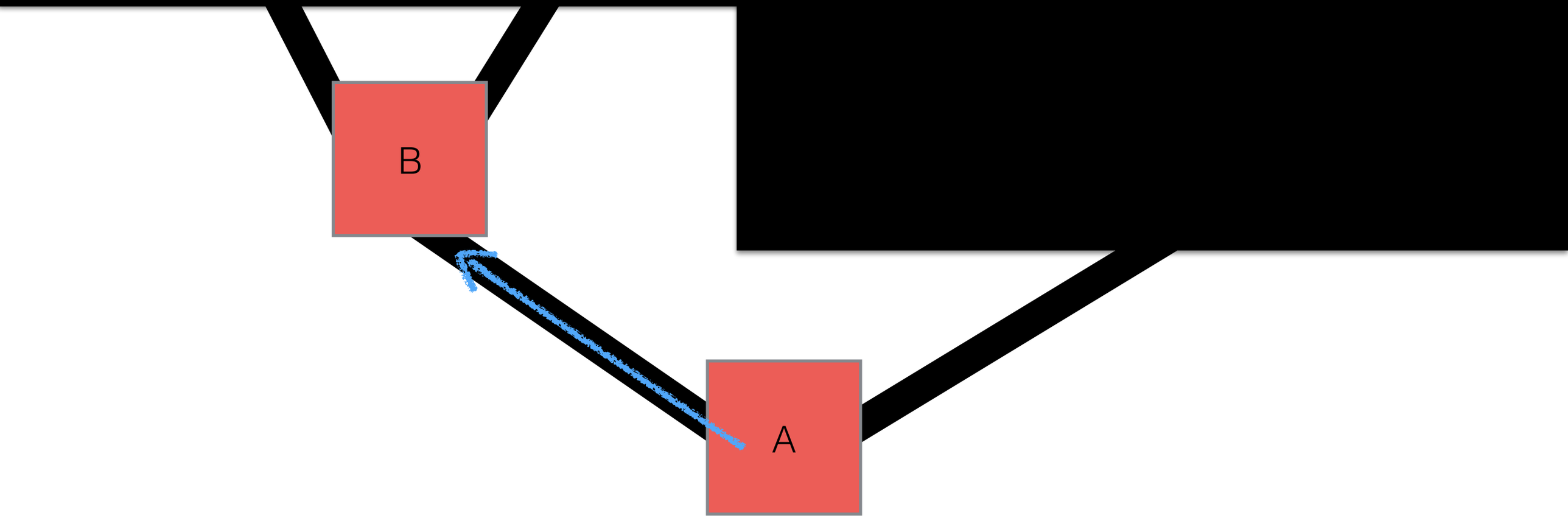
Find the Cash Game

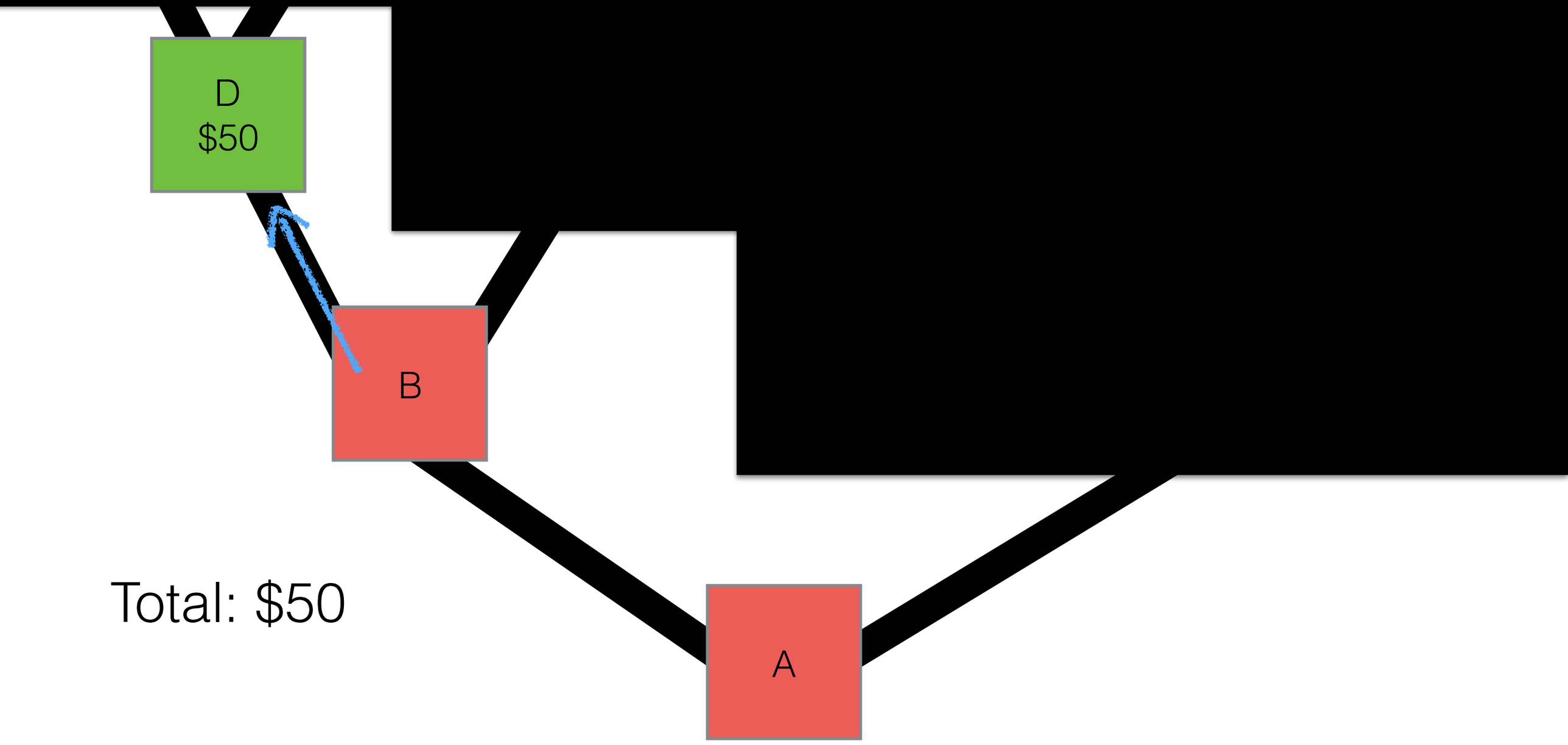
- Imagine a bunch of rooms connected by hallways.
- Each room has a right door and a left door or no doors at all.
- There is cash spread out through the rooms at random. You want to collect all the cash.
- What strategy would you use for walking through each room?

As you walk through each room, write down its letter...

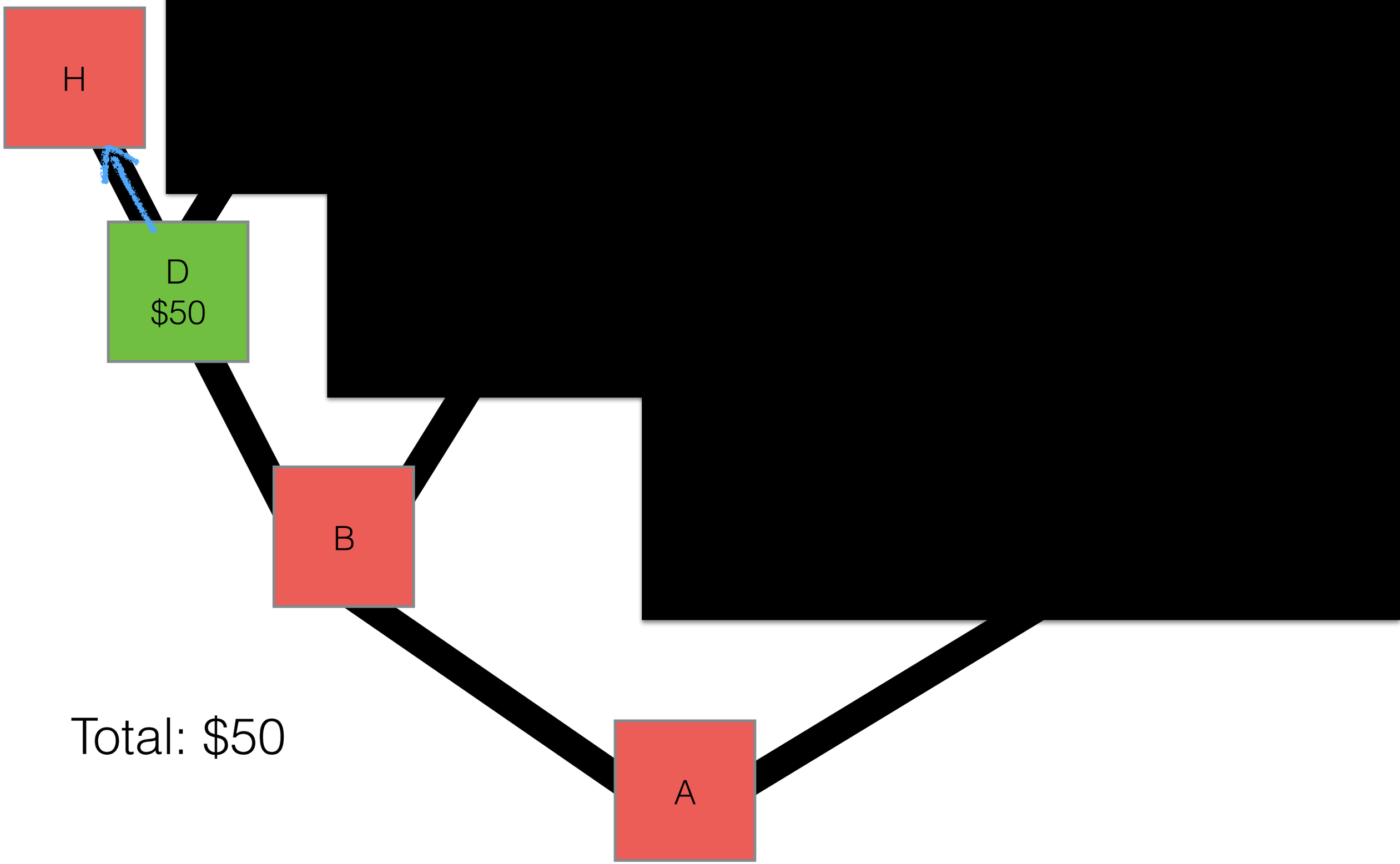




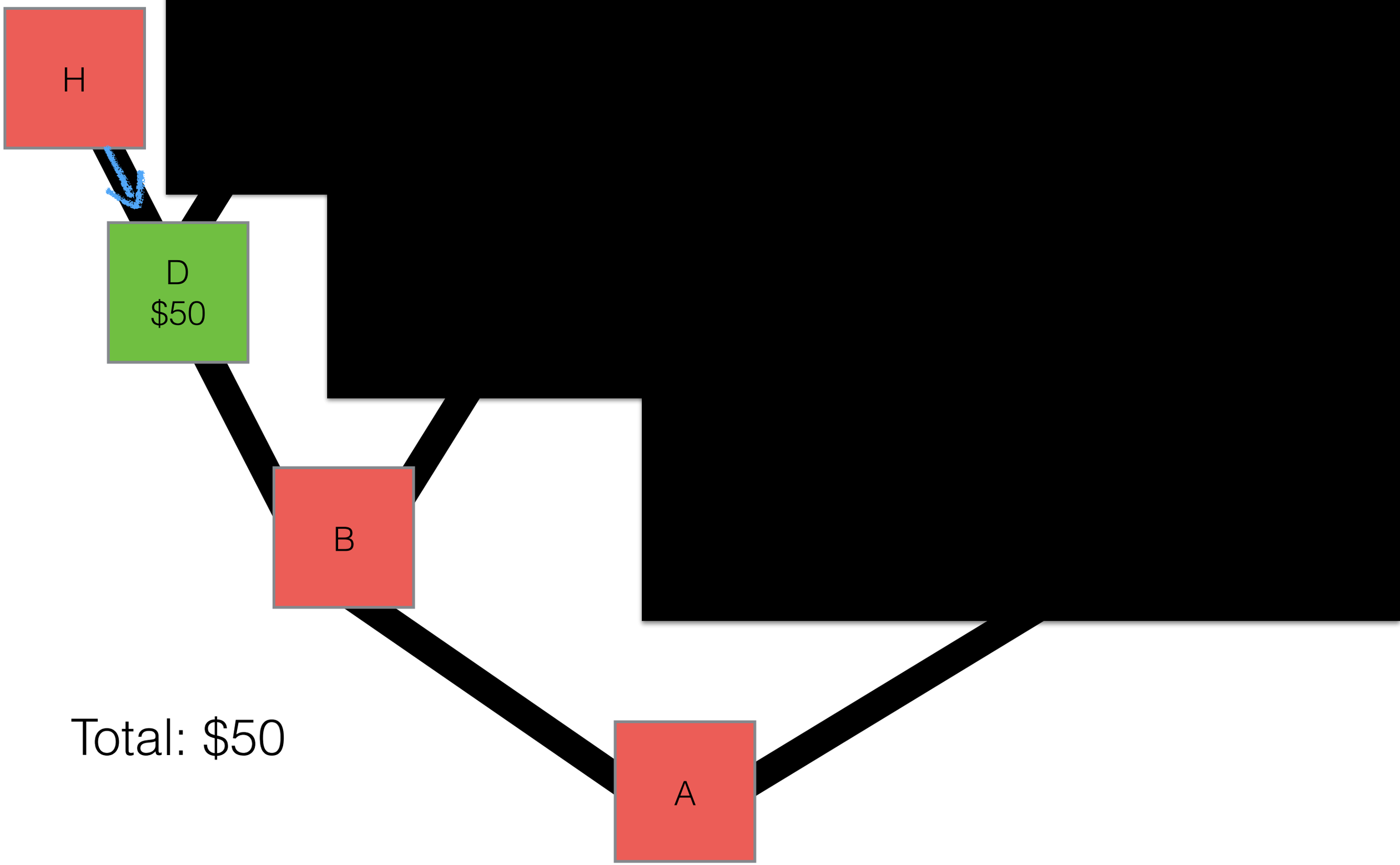

B
A



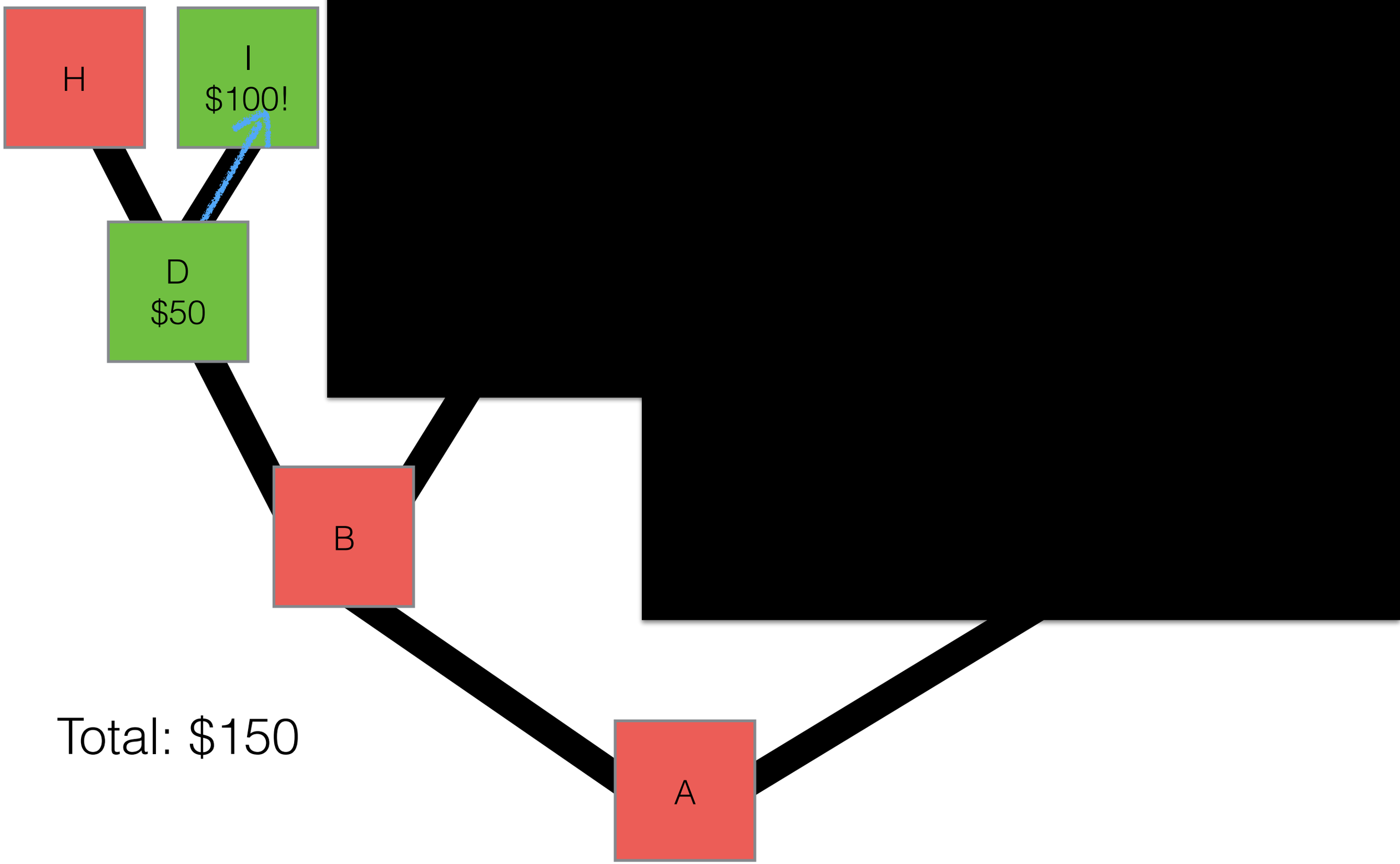
Total: $50



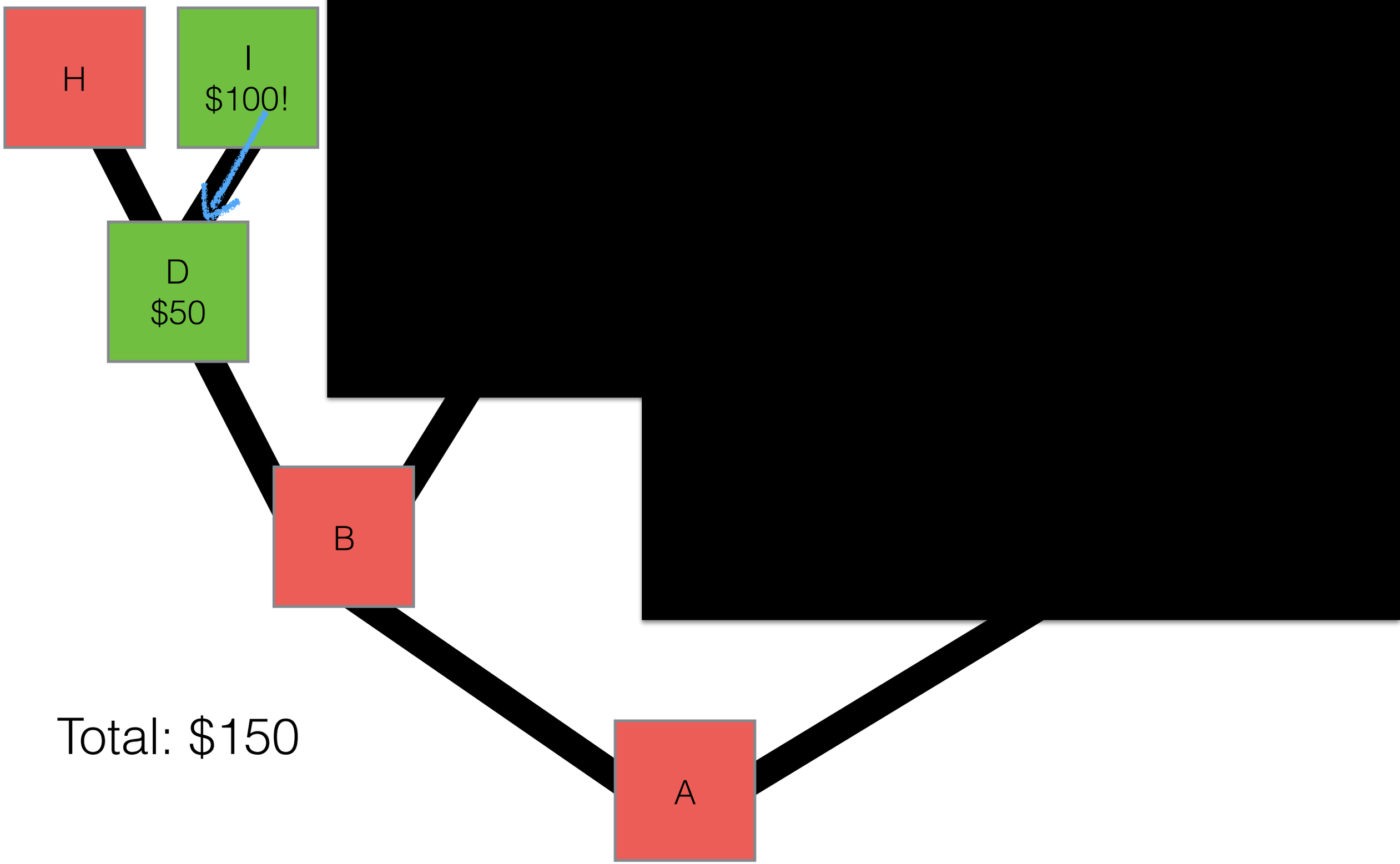
Total: $50



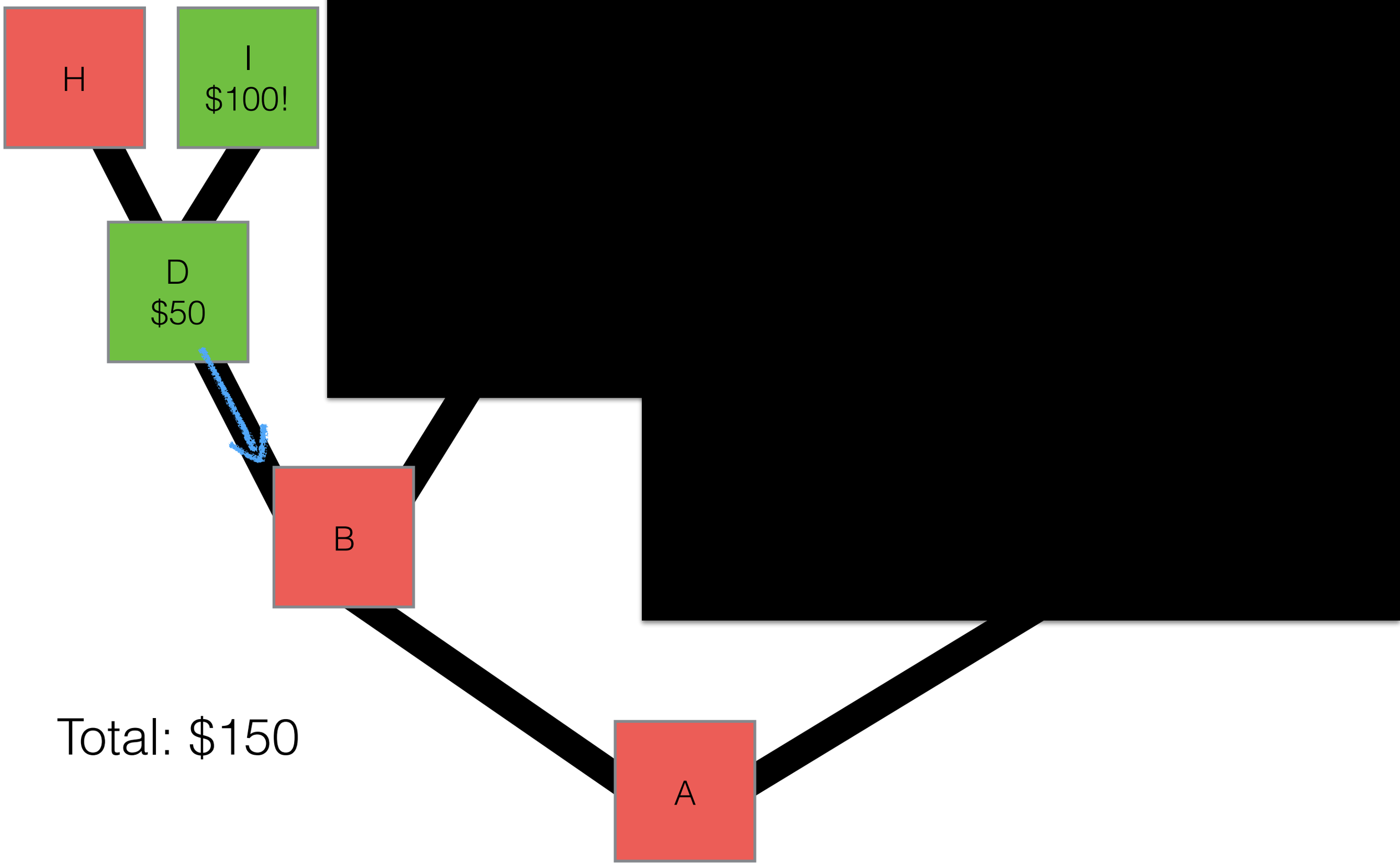
Total: $50



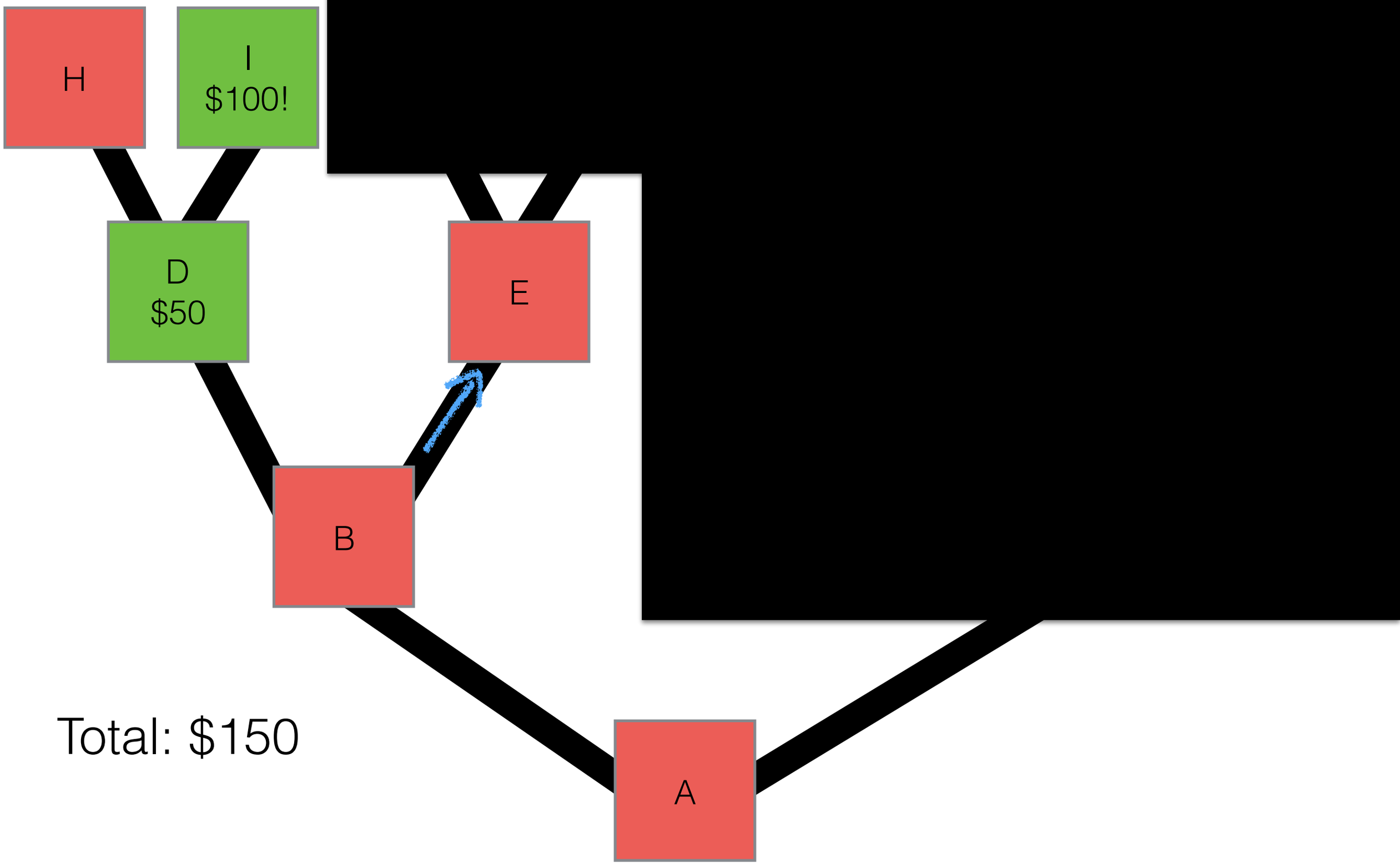
Total: $150



Total: $150



Total: $150



Total: $150

How Could We Do This with Code?

- Open FindTheCash.java and then Room.java
- Try running FindTheCash.java
- Let’s walk through the code…

Implement our Room Searching Algorithm

1. In Room.java's **searchForCash** method…
2. Check to see if it has a left door / right door using **this.hasLeftDoor()** / **this.hasRightDoor()**
3. If so, add the cash in the room behind each door to the cashFound variable.
4. How could you search each room for cash **recursively**?

Done? Check in on PollEverywhere pollev.com/comp110

This is called a recursive descent algorithm

1. Useful when “walking” or “traversing” a **tree** of objects

2. Imagine needing to write a program that finds a file in your Documents folder (supposing there are subfolders). Same idea!

3. Handy when working with a hierarchy (folders, web sites, programming language compilers, scene graphs)

Recursive Data Types

1. When data types can refer to themselves you can construct complex relationships between objects

2. “Trees” and “Graphs” of Objects are commonly used to model real-world concepts

    1. Organizational chart: Tree

    2. Follow / Followers on Instagram: Graph

3. Recursive methods often pair beautifully with recursive data types