Day 3 - Python Programming

Recursion
State Machines
Classes
From Idea to Business
Project Time

Recursion

What is Recursion?

A programming technique whereby a function calls itself either directly or indirectly.

Recursion Example

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, . . .

Mathematically:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2)$$

Recursion Example

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, . . .
Mathematically:

F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Recursion vs Iteration

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        a = 1
        b = 1
        for i in range(1, int(n)-1):
            temp = a + b
            a = b
            b = temp
        return b
```

Benefits of Recursion

1. Simpler and more elegant solutions
2. Easier to visualize problems
3. Sometimes more efficient

Laws of Recursion

1. Requires a base case
2. Requires a recursive case
3. Recursive call must progress problem towards base case

Laws of Recursion

1. Requires a base case

   Path through the code which does not call the function again

2. Requires a recursive case
3. Recursive call must progress problem towards base case

Laws of Recursion

1. Requires a base case
2. Requires a recursive case
   Code that calls the function from within the function
3. Recursive call must progress problem towards base case

Laws of Recursion

1. Requires a base case
2. Requires a recursive case
3. Recursive call must progress problem towards base case

i.e. must reduce problem

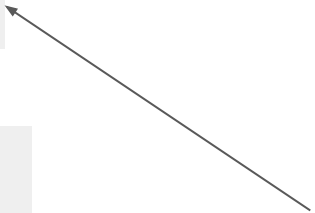
Recursion Example

```
def fibonacci(n):
    if n == 0:
        return 0

    elif n == 1:
        return 1

    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Base case

Base case

Recursive case



Drawbacks of Recursion

Function calls can be costly

- use up memory
- use up time

Common Pitfalls...

1. No base case
2. No progress towards base case
3. Infinite loop
4. Too costly

Common Pitfalls...

1. No base case
2. No progress towards base case
3. Infinite loop
4. Too costly

```
def fib(n):
    return fib(n-1) + fib(n-2)
```

What does this function know?

Common Pitfalls...

1. No base case
2. No progress towards base case
3. Infinite loop
4. Too costly

```
def fib(n):

    if n == 0 or n == 1:
        return n
    else:
        return (n-1) + (n-2)
```

Is the base case being used by the else?

Common Pitfalls...

1. No base case
2. No progress towards base case
3. Infinite loop
4. Too costly

```
def fib(n):

    if n == 0 or n ==1:
        return n
    else:
        return f(n+1) + f(n+2)
```

When will this end?

Common Pitfalls...

1. No base case
2. No progress towards base case
3. Infinite loop
4. Too costly

Especially when function being recursed:

- has many operations
- has many variable assignments

Recursion Optimization: Memoization

"Memorized" results from previous runs used in subsequent repeated calls.

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

How can we optimize this solution?

Recursion Optimization: Memoization

"Memorized" results from previous runs used in subsequent repeated calls.

```
memo = {}
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        if n not in memo:
            memo[n] = fibonacci(n-1) + fibonacci(n-2)
        return memo[n]
```

Recursion Exercise 1: Factorial

Given that n≥1, return the factorial of n. Compute the result recursively (without loops).

```
factorial(n) = n * (n-1) * (n-2) * ... * 1
factorial(1) → 1 (1)
factorial(2) → 2 (2 * 1)
factorial(3) → 6 (3 * 2 * 1)
```

Factorial Solution

```
factorial(n) = n * (n-1) * (n-2) * ... * 1
```

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Recursion Exercise 2: PowerN

Given a base and n and both are ≥ 0, compute recursively (no loops) the value of base to the n power, so is 9 (3 squared).

```
powerN(3, 2) = 3 ^ 2 = 9
powerN(2, 3) = 2 ^ 3 = 8
powerN(9, 0) = 9 ^ 0 = 1
```

PowerN Solution

`powerN(n) = base ^ n`

```
def powerN (b, n):
    if n == 0:
        return 1
    else:
        p = b * powerN(b, n-1)
```

PowerN Solution

powerN(n) = base ^ n

```
def powerN (b, n):
    if n == 0:
        return 1
    else:
        p = powerN(b, n//2)
        if n%2 == 0:
            return p * p
        else:
            return b * p * p
```

~Break~

State Machines

State Machines

Mathematical computation model

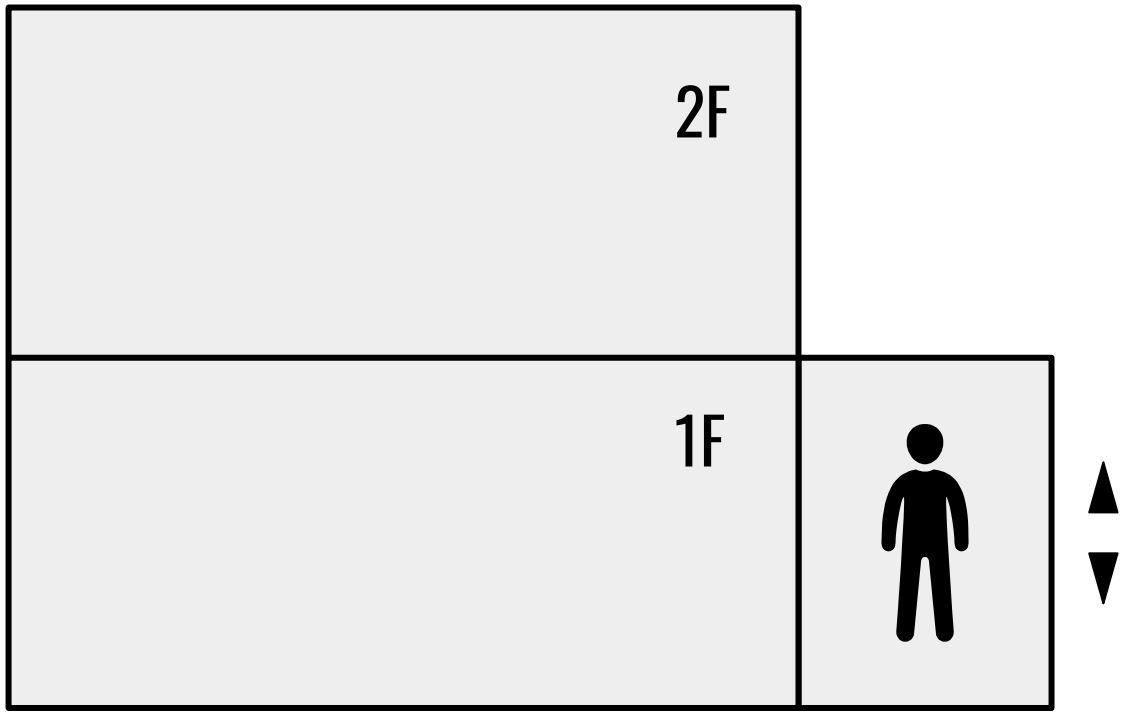
Abstract machine (theoretical idea of model)

Change states by specific inputs

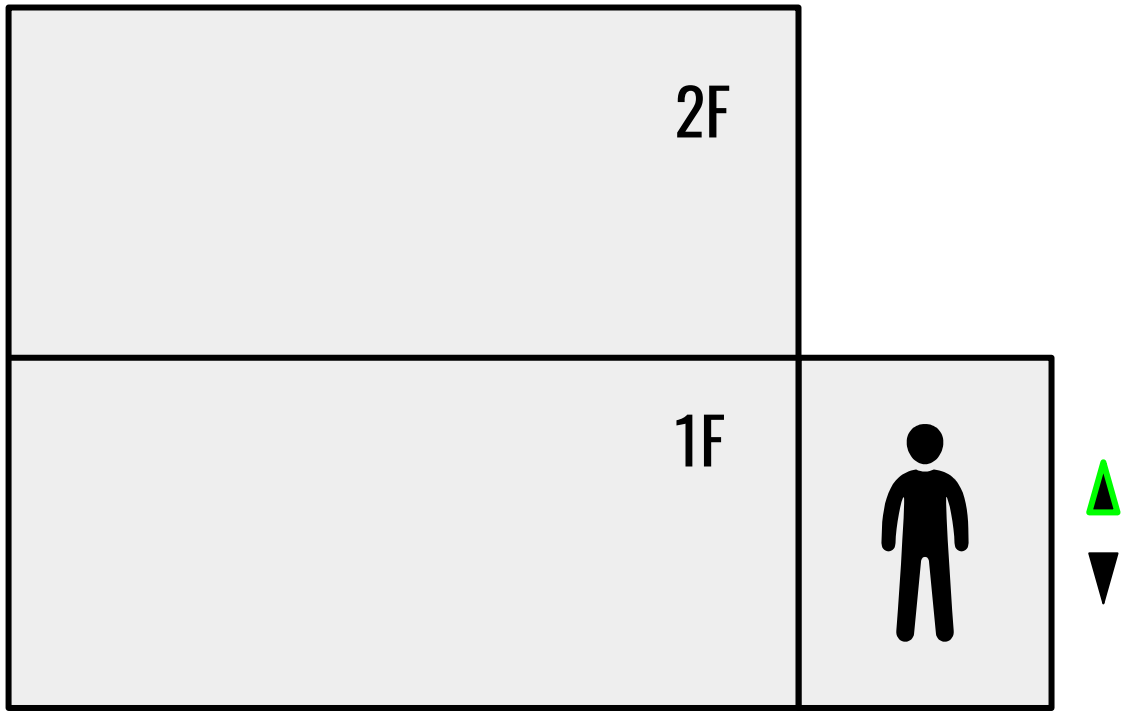
Examples of State Machines:

- Vending Machine
- Traffic Lights
- Combination Locks
- Elevators

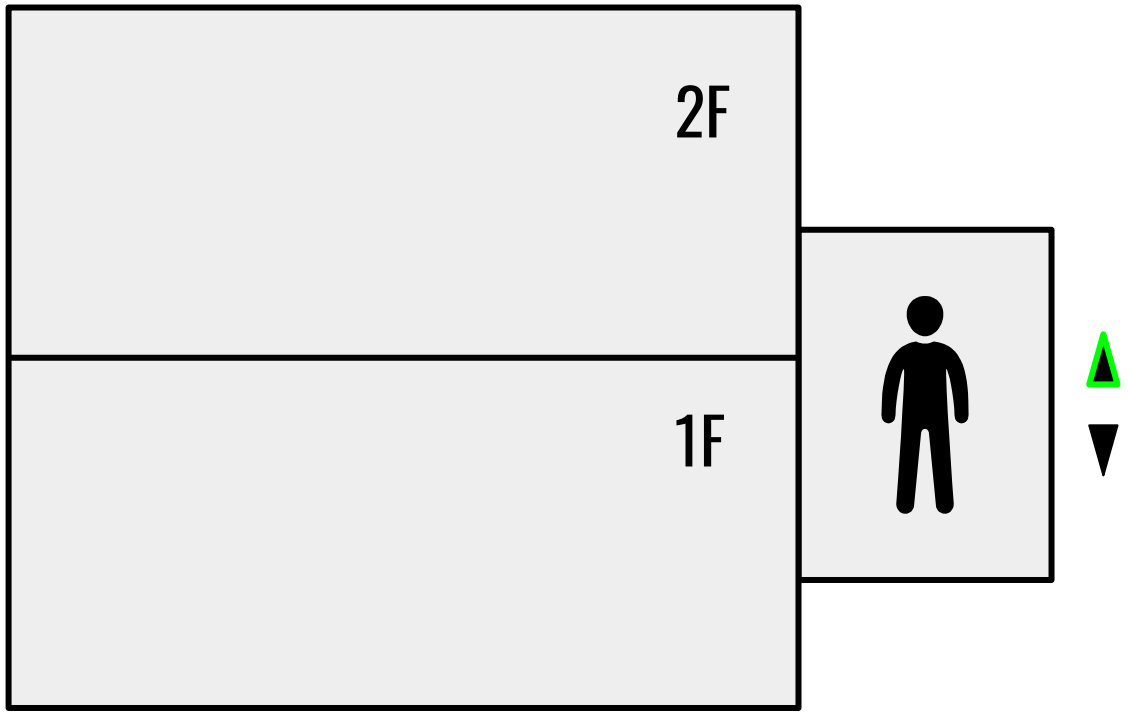
Elevator



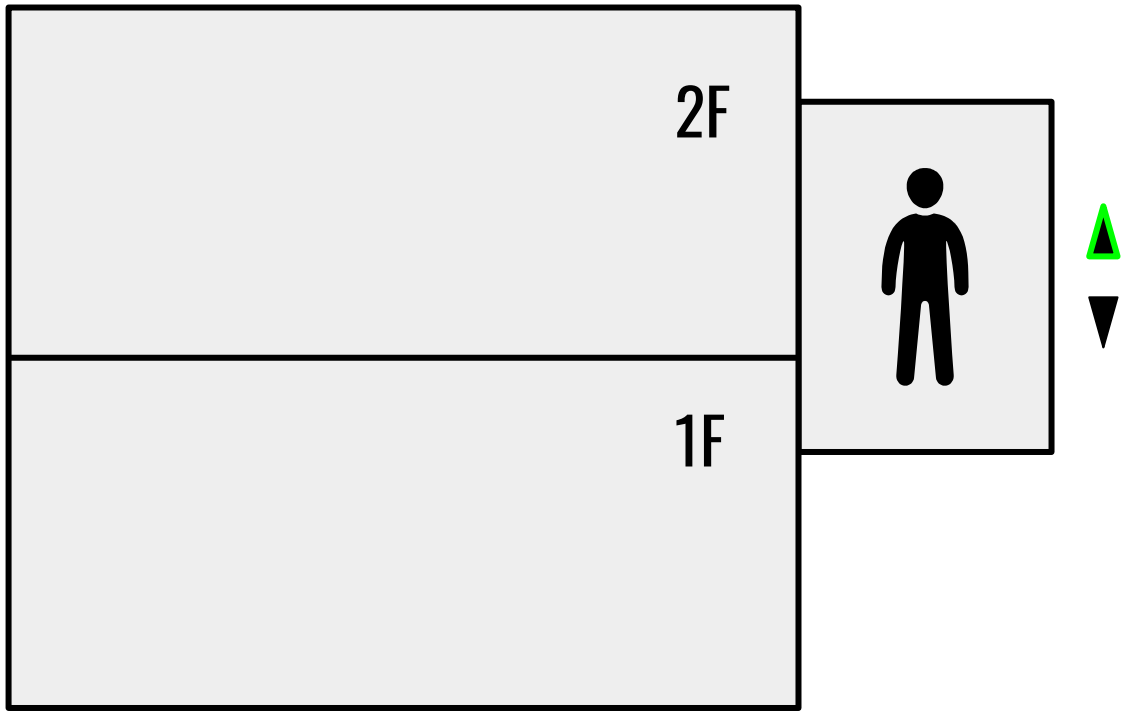
Elevator



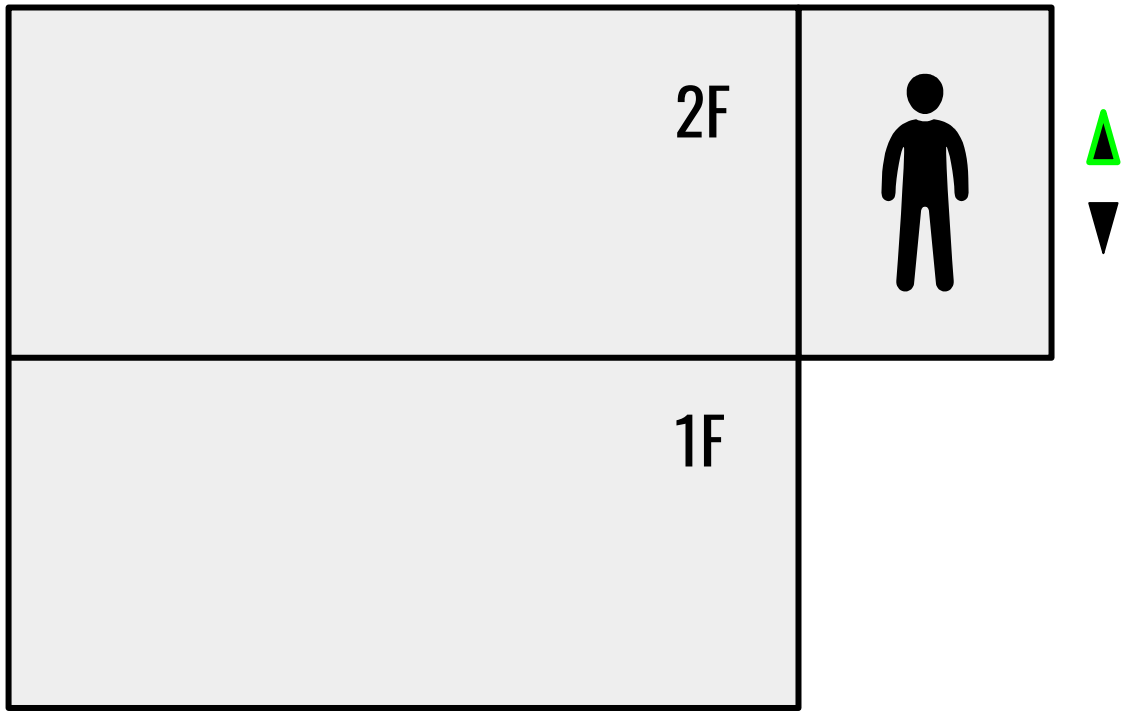
Elevator



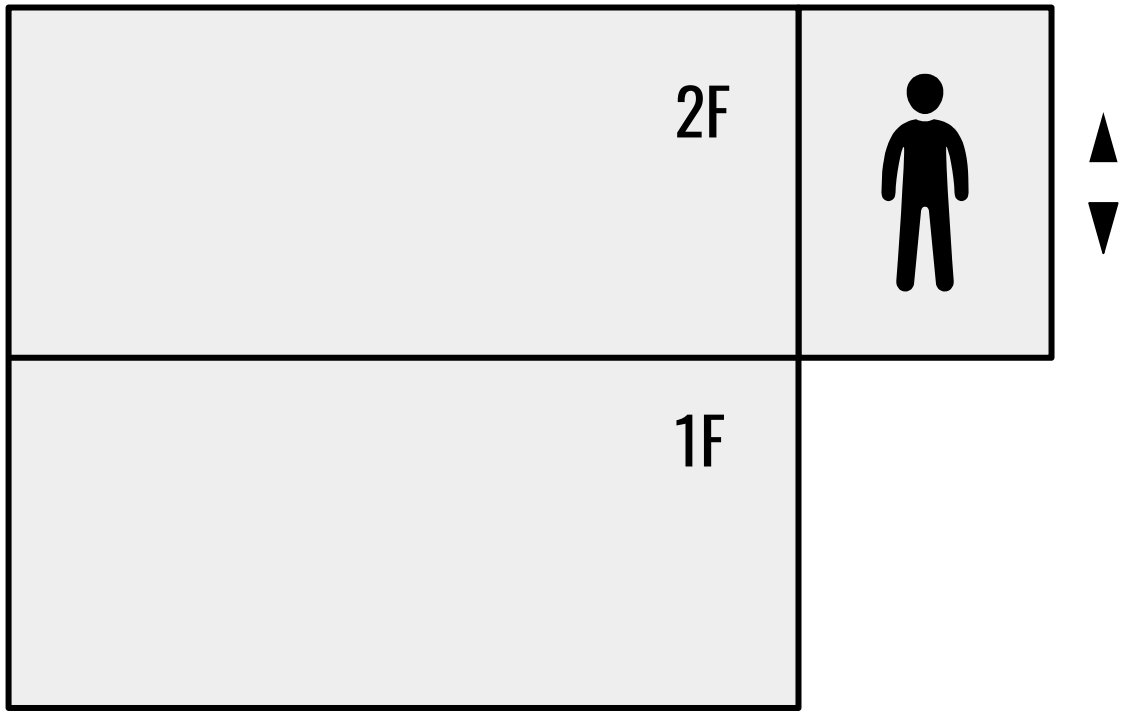
Elevator



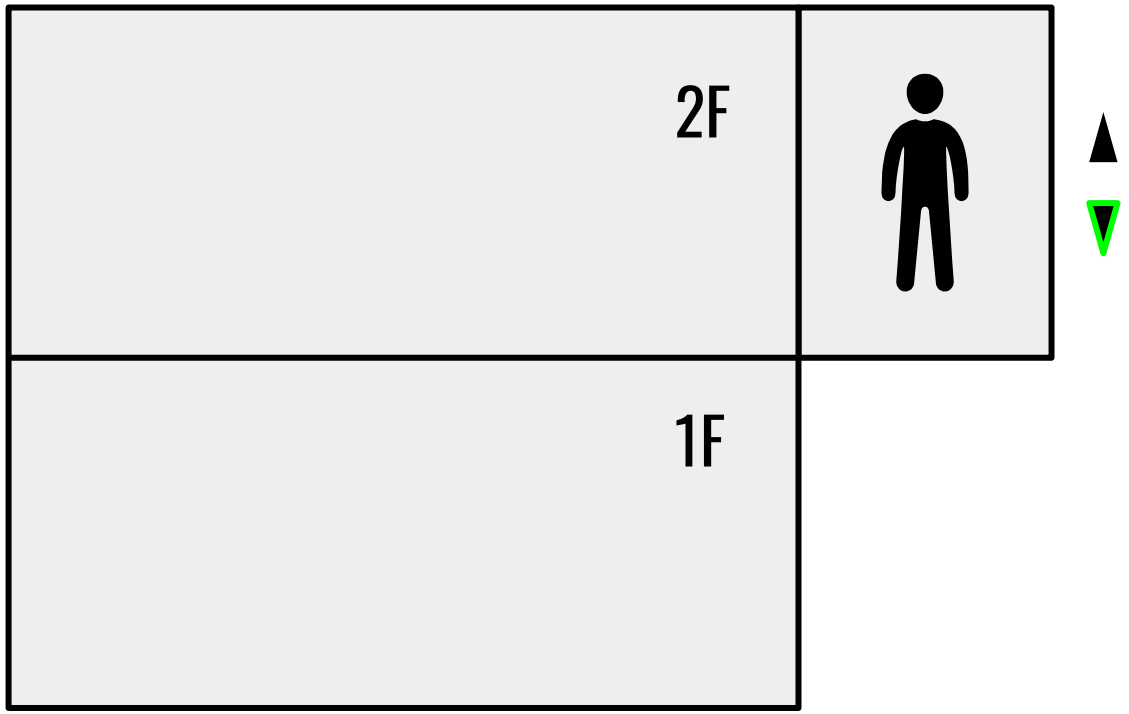
Elevator



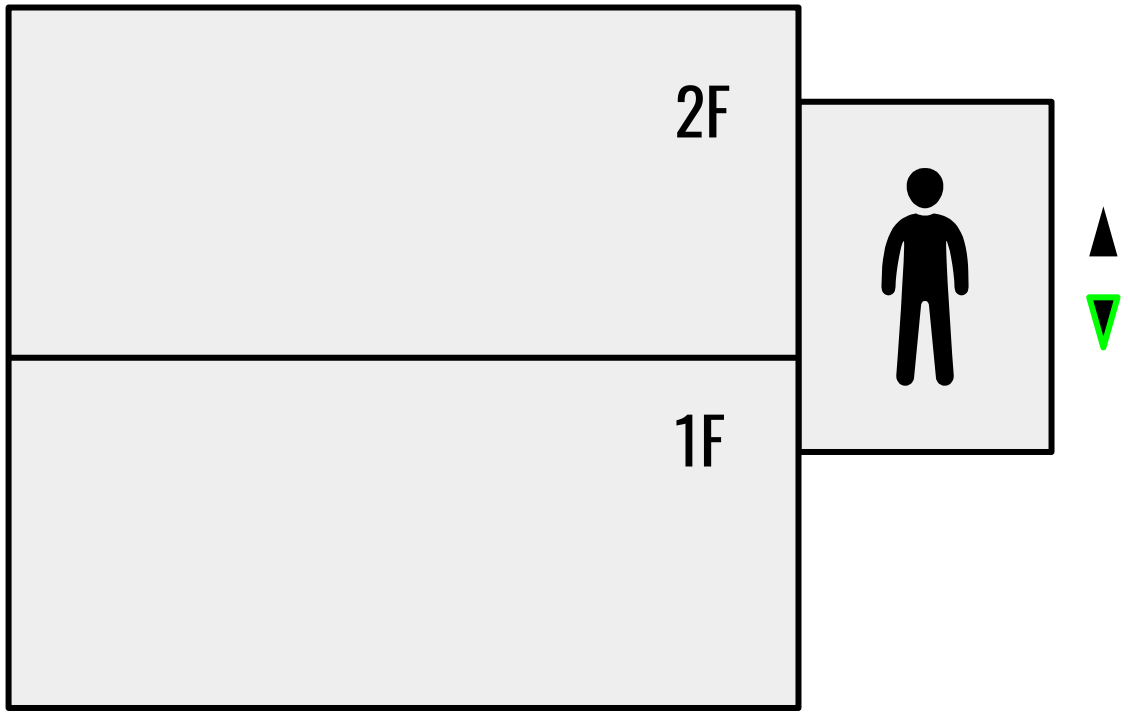
Elevator



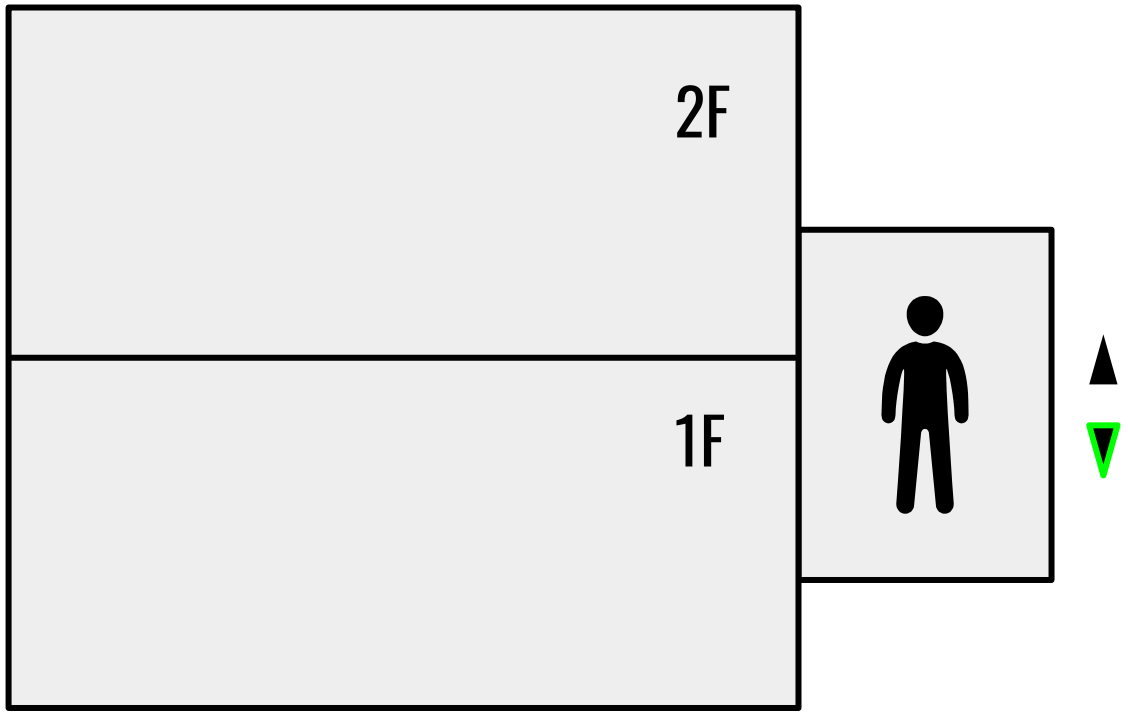
Elevator



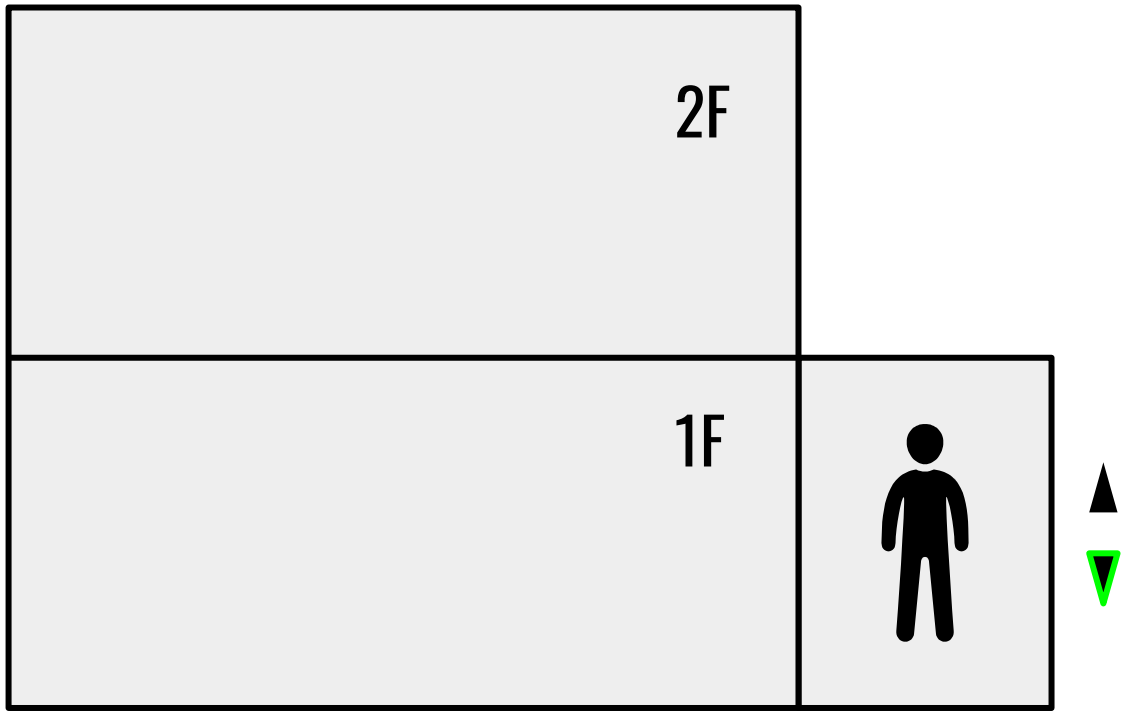
Elevator



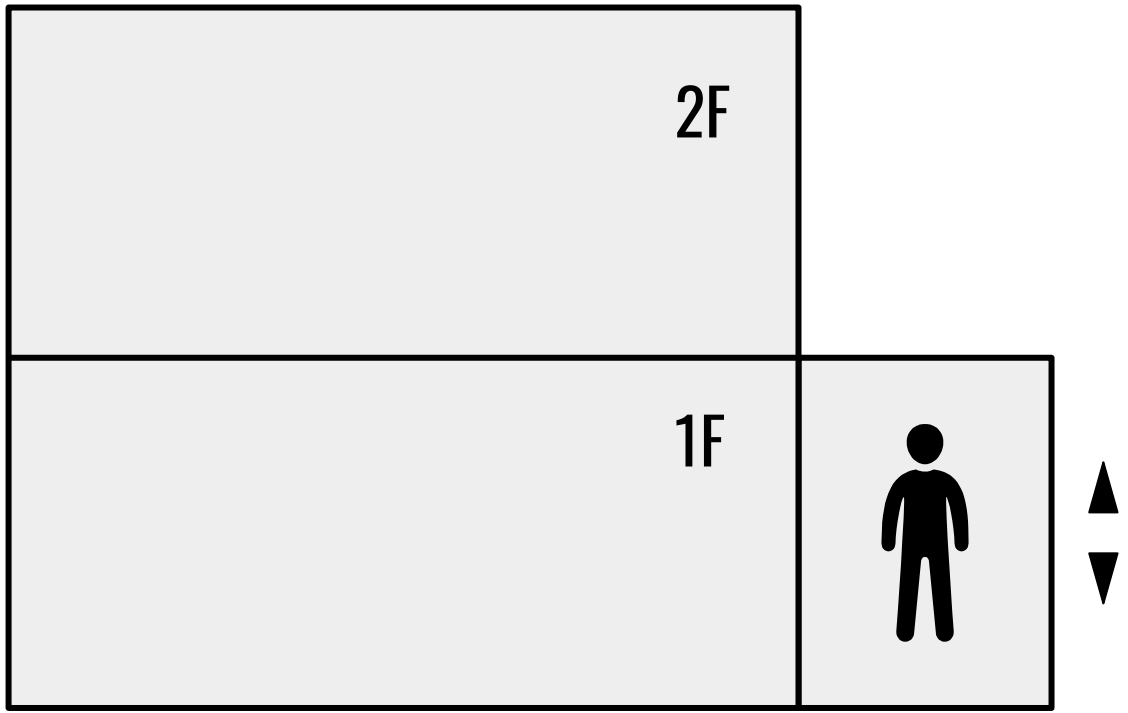
Elevator



Elevator

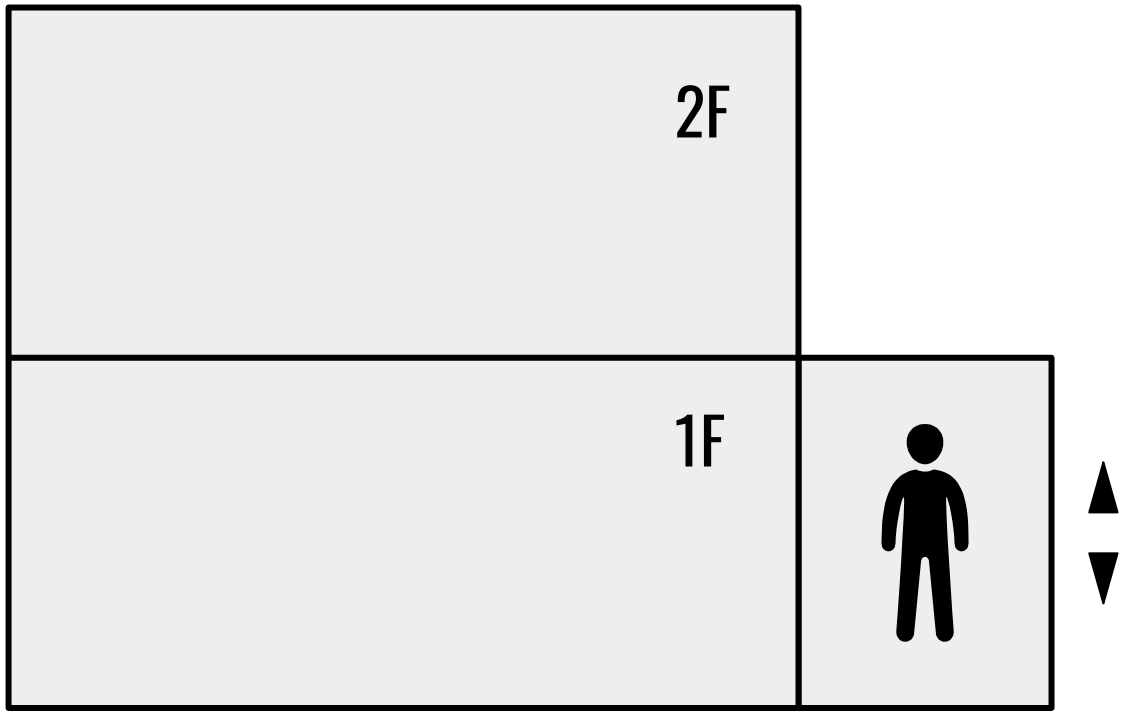


Elevator



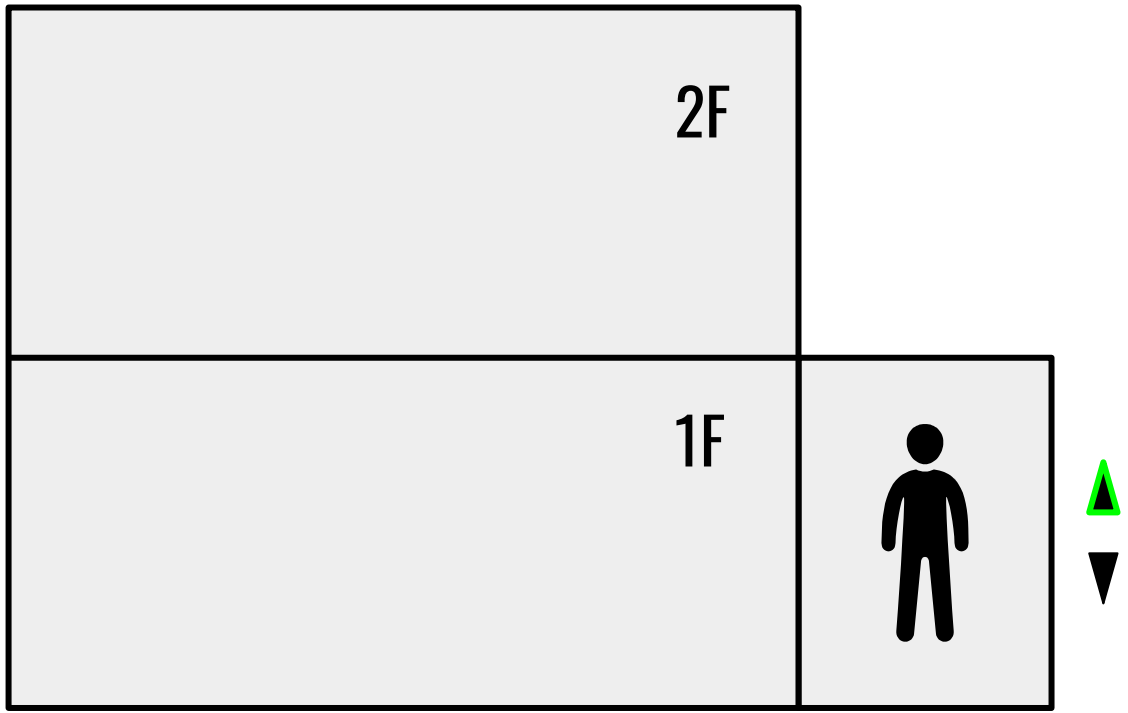
Elevator as a State Machine!

Elevator



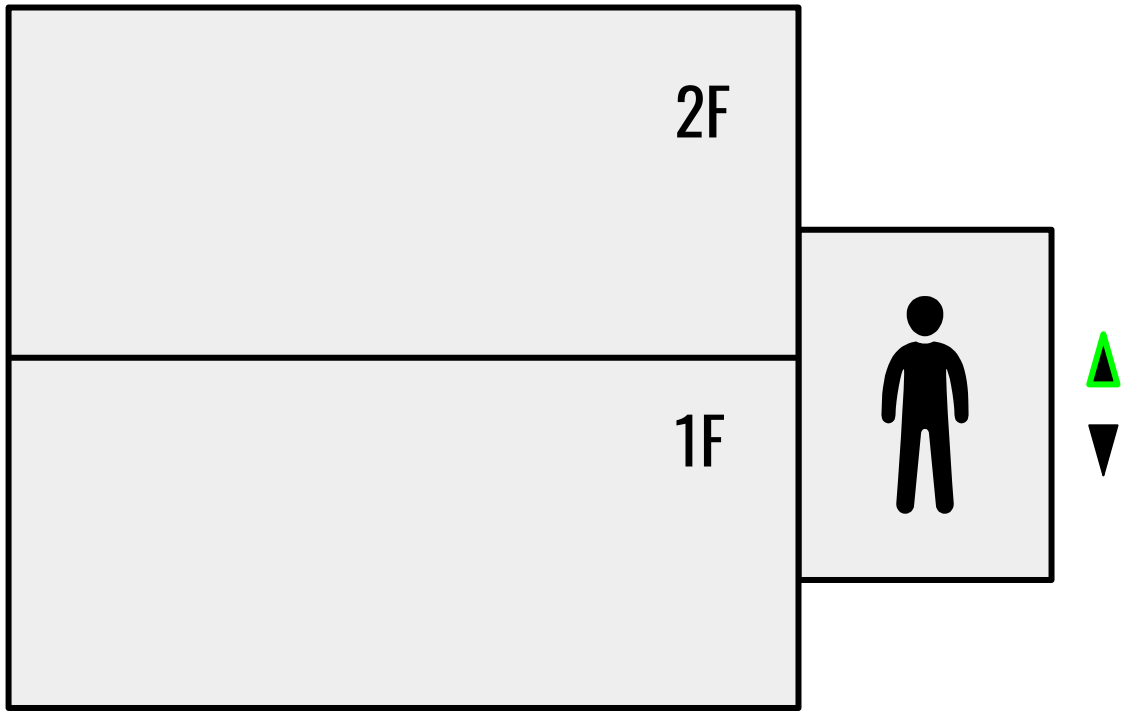

```
moving = 0
up = 0
down = 0
```

Elevator



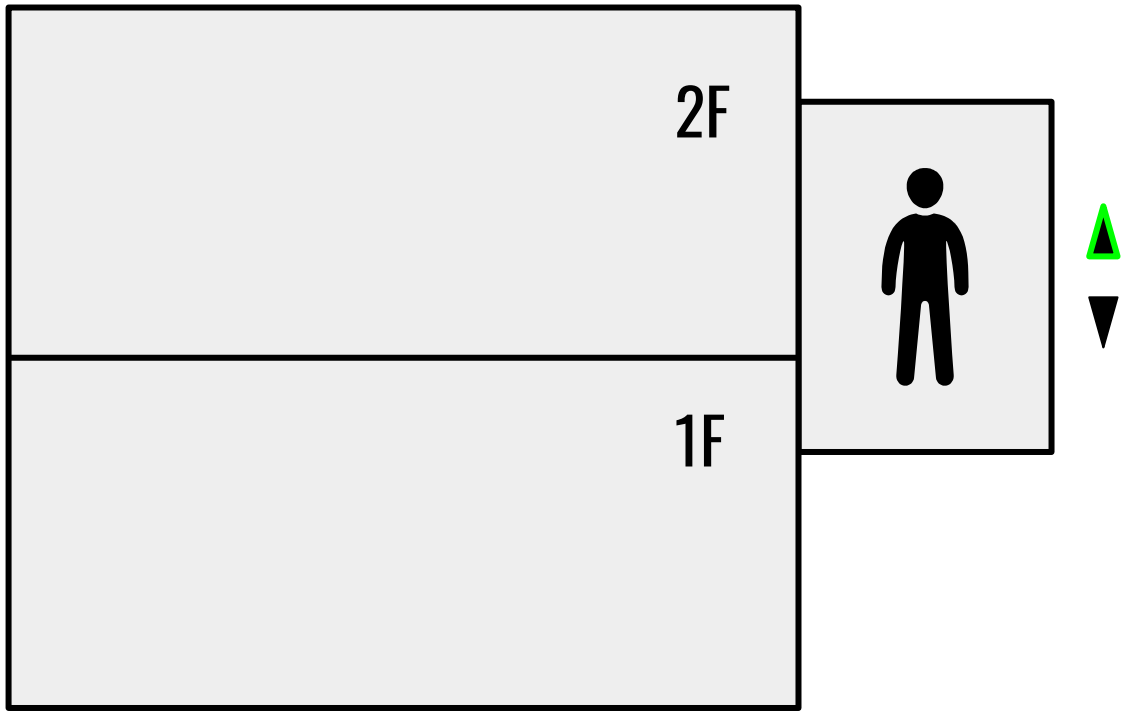

```
moving = 0
up = 1
down = 0
```

Elevator



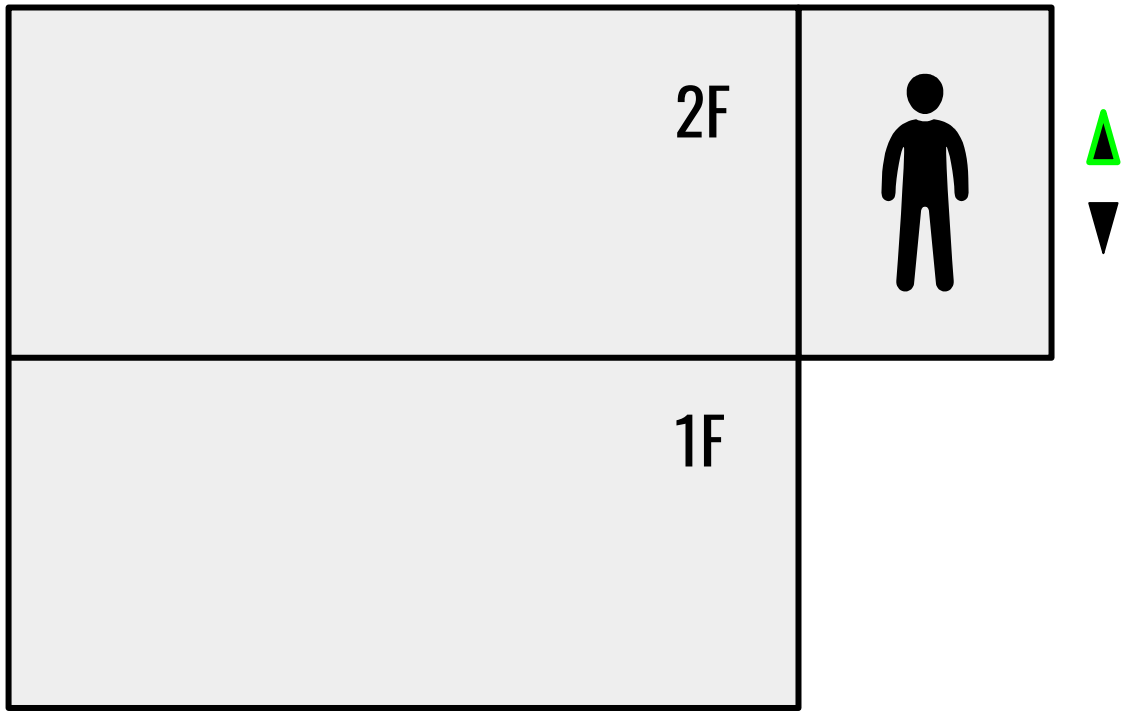

```
moving = 1
up = 1
down = 0
```

Elevator



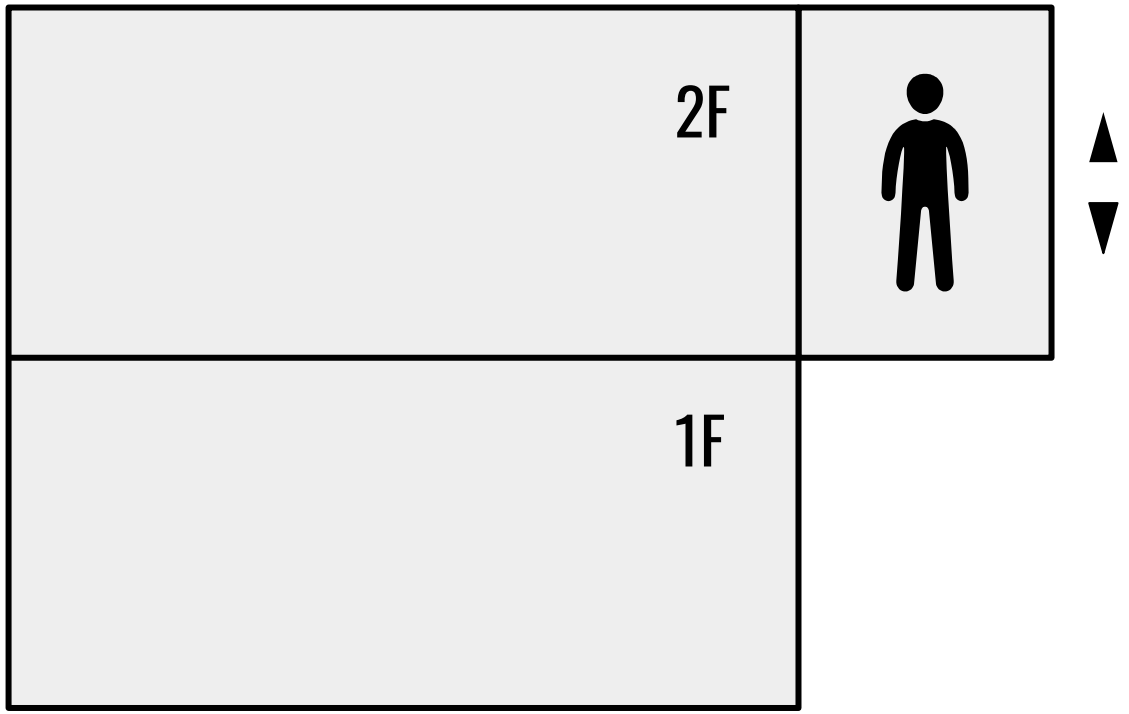

```
moving = 1
up = 1
down = 0
```

Elevator



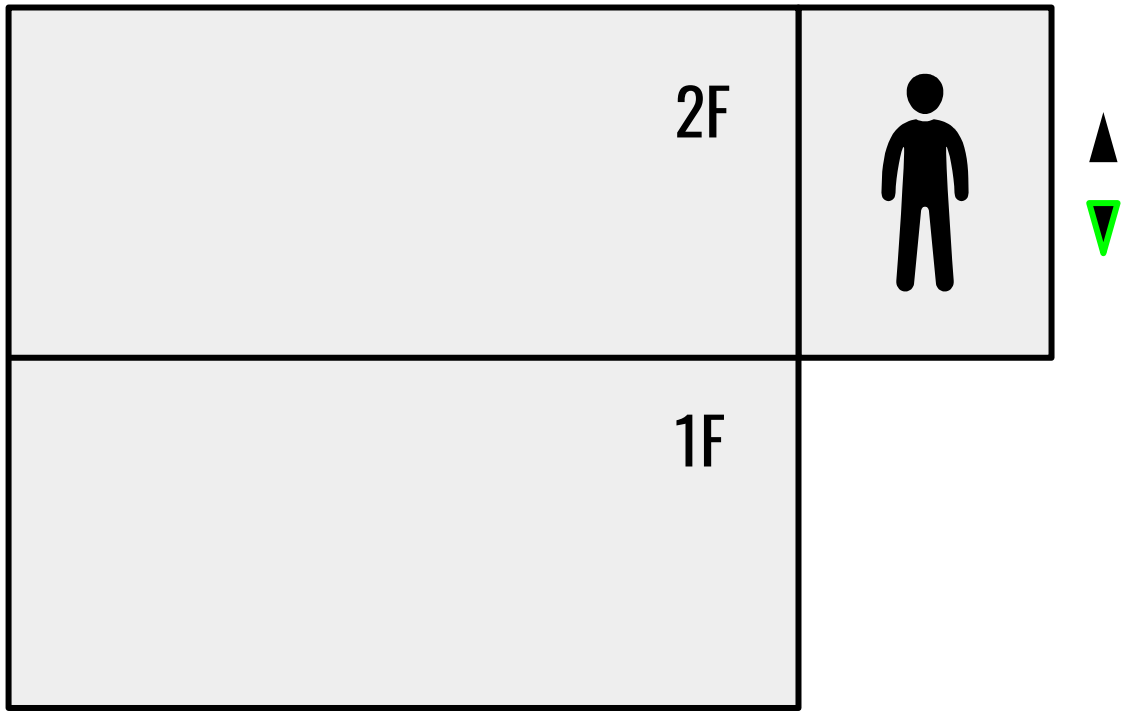

```
moving = 0
up = 1
down = 0
```

Elevator



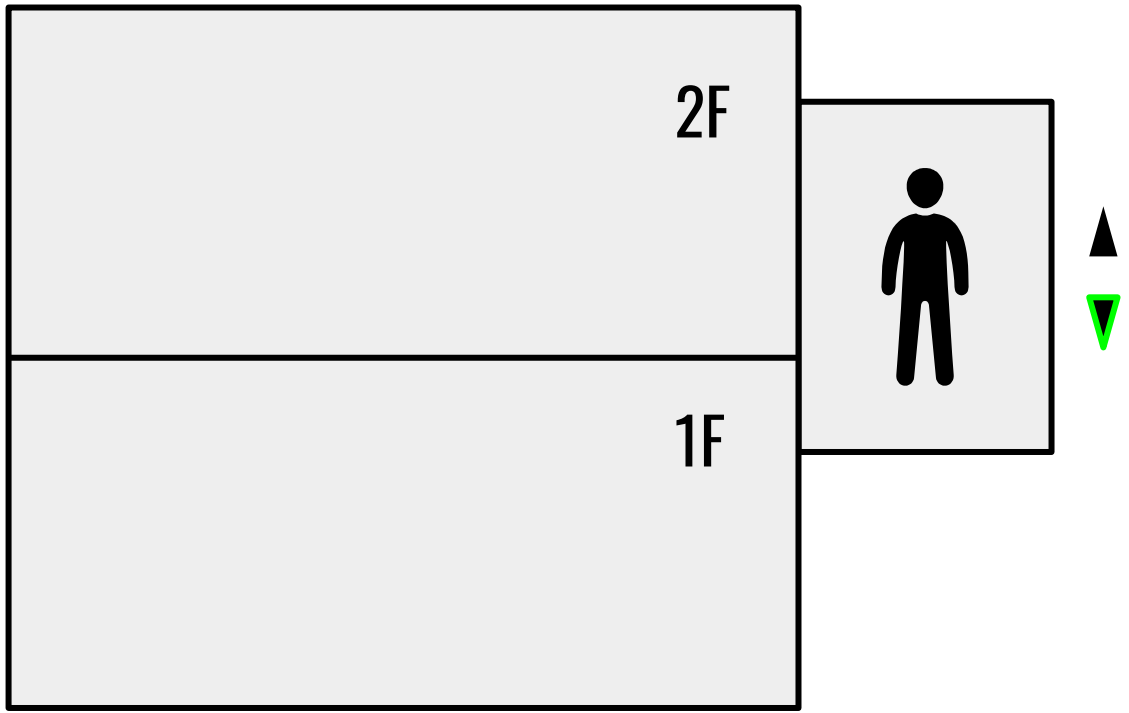

```
moving = 0
up = 0
down = 0
```

Elevator



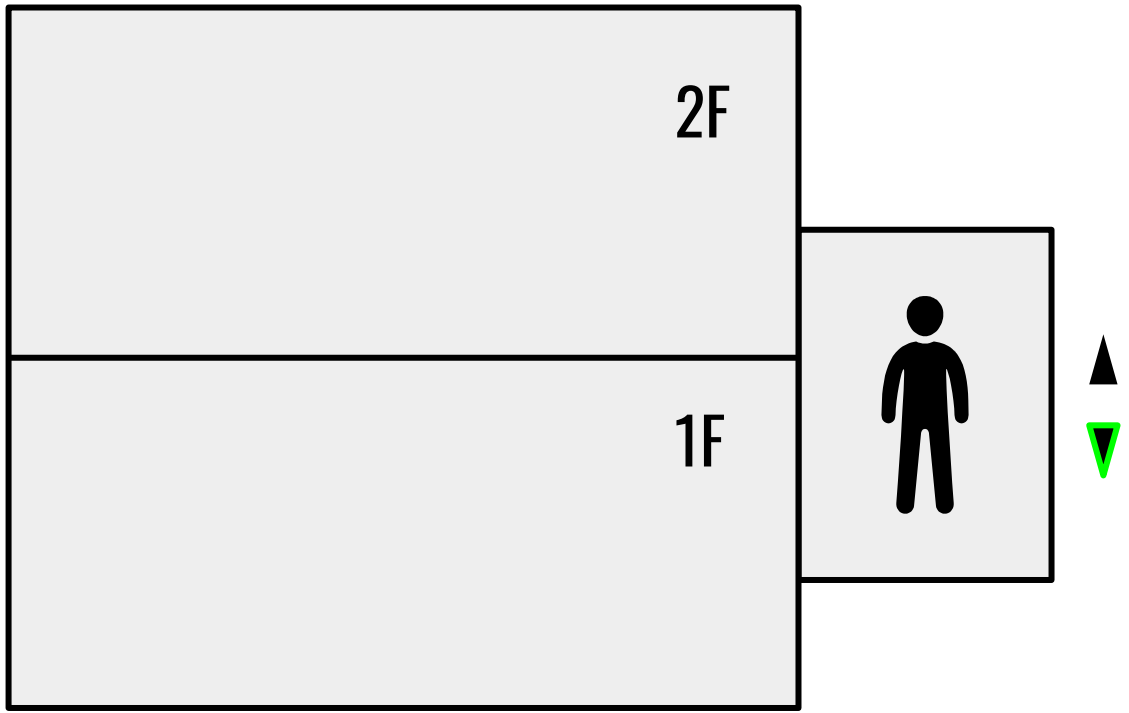

```
moving = 0
up = 0
down = 1
```

Elevator



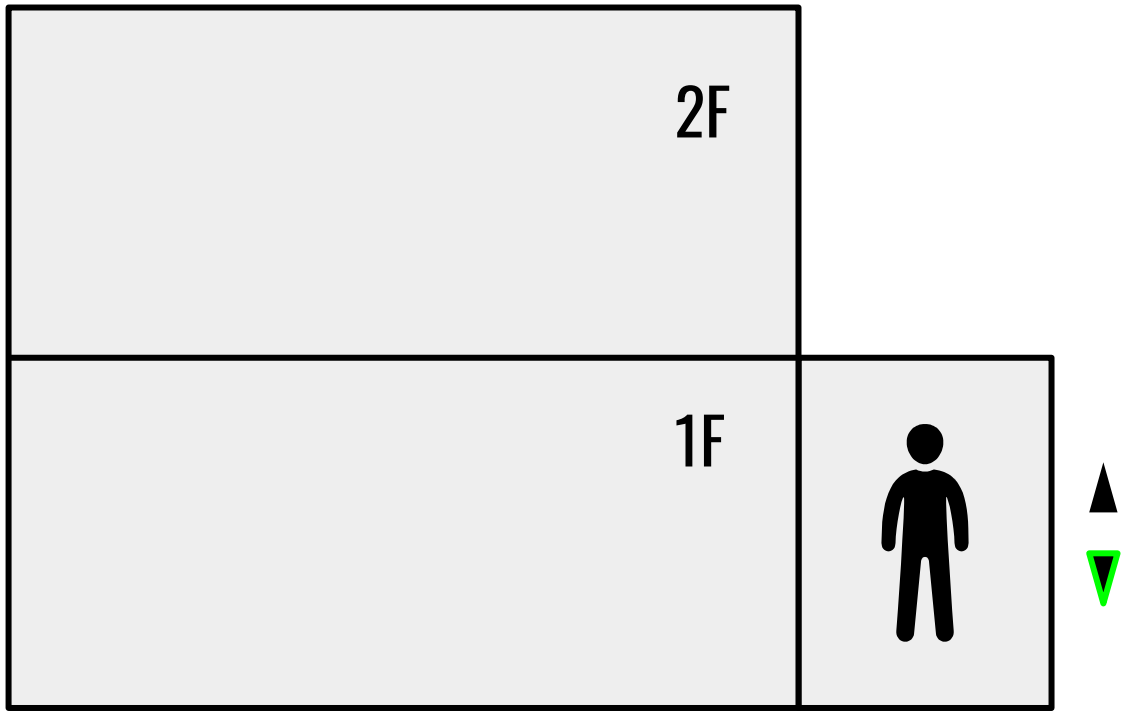

```
moving = 1
up = 0
down = 1
```

Elevator



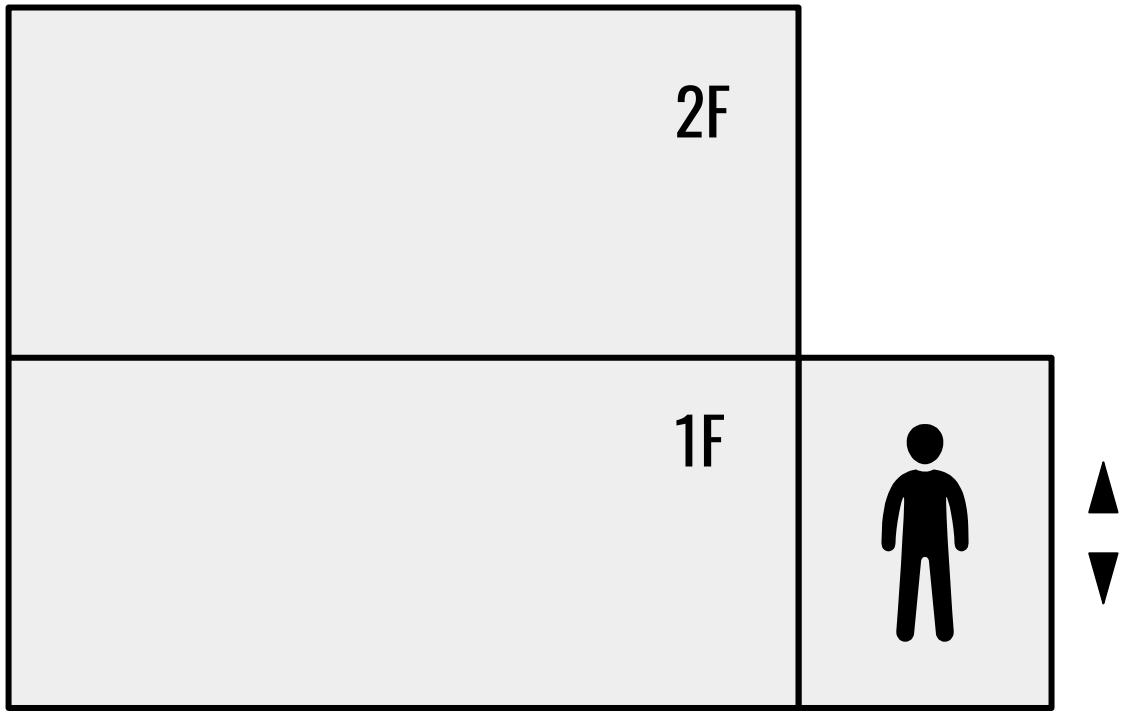

```
moving = 1
up = 0
down = 1
```

Elevator

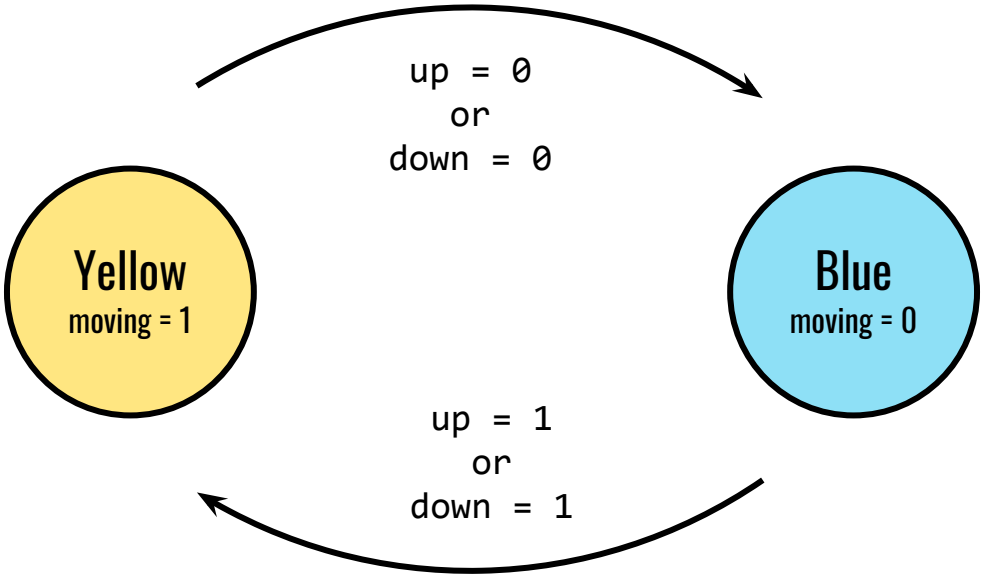



```
moving = 0
up = 0
down = 1
```

Elevator



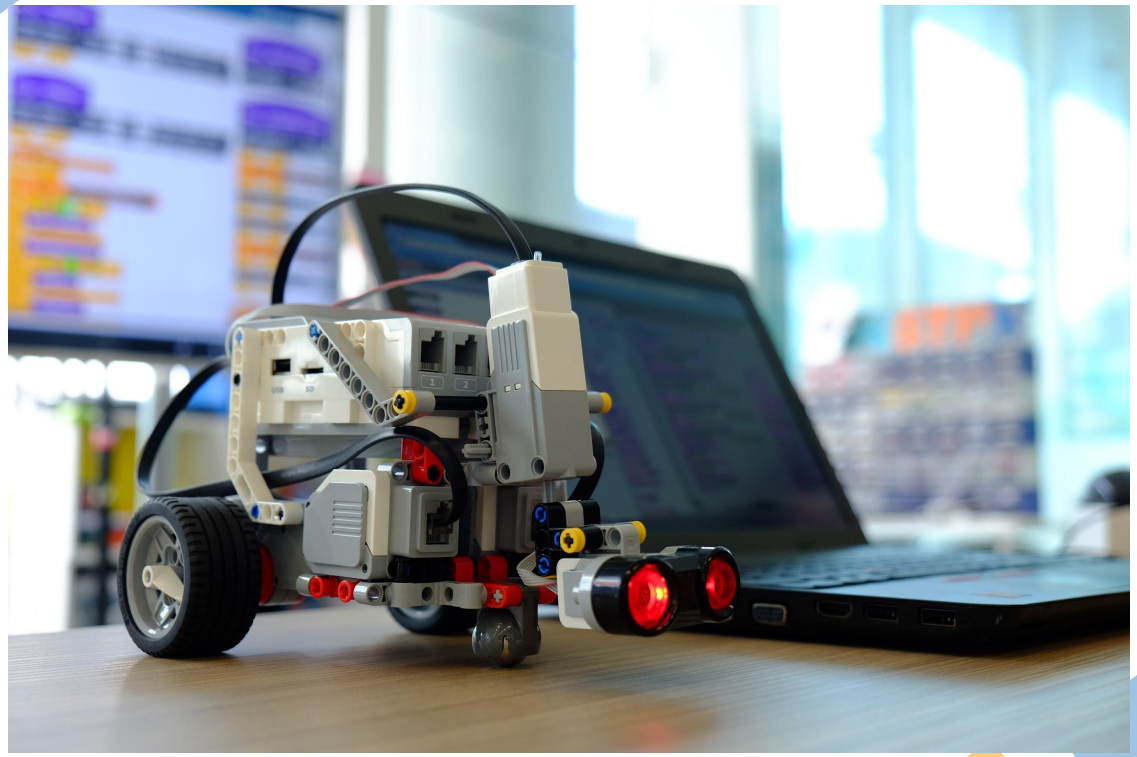

```
moving = 0
up = 0
down = 0
```

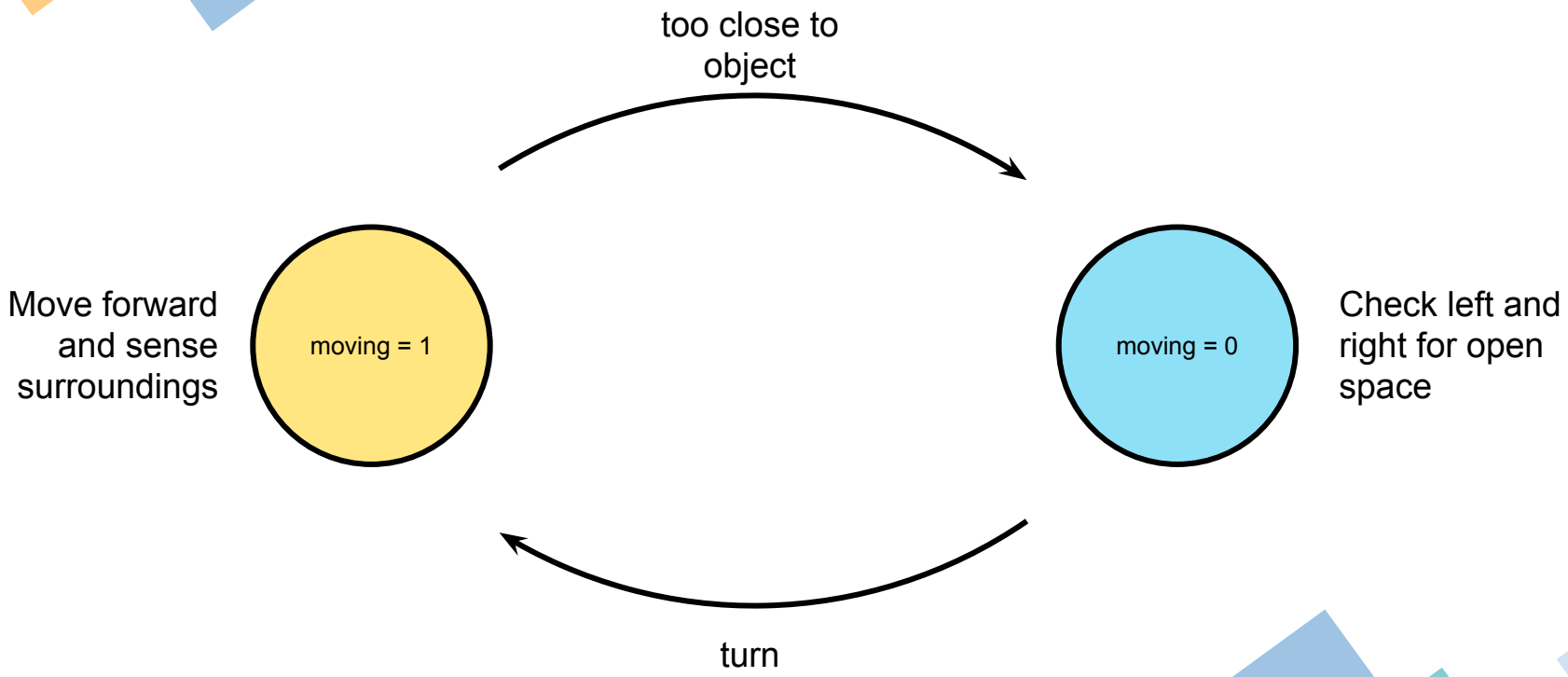

up = 0
or
down = 0
Yellow
moving = 1
Blue
moving = 0
up = 1
or
down = 1

Your Turn!

Write a state machine for being hungry

Why is this relevant?




too close to object
Move forward and sense surroundings
moving = 1
moving = 0
Check left and right for open space
turn

```
state = 0

if state == 0:
      game.reset()
elif state == 1:
      #level one
      player.battle(snakes)
      if player.win():
            print('next level!')
            state += 1
      else:
            print('start over')
            state = 1
elif state == 2:
      #level two
      player.battle(dinosaurs)
      if player.win():
            print('next level!')
            state += 1
      else:
            print('start over')
            state = 1
elif state == 3:
      #level one
      player.battle()
      if player.win():
            print('yay!')
      else:
            print('game over')
      state = 0
```

```
state = 0
lives = 3

if state == 0:
      game.reset()
elif state == 1:
      #level one
      player.battle(snakes)
      if player.win():
            print('next level!')
            state += 1
      else:
            lives -= 1
            if lives > 0:
                  level.reset()
            else:
                  print('start over')
                  state = 1
```

```
elif state == 2:
      #level two
      player.battle(dinosaurs)
      if player.win():
            print('next level!')
            state += 1
      else:
            lives -= 1
            if lives > 0:
                  level.reset()
            else:
                  print('start over')
                  state = 1
```

```
elif state == 3:
      #level one
      player.battle()
      if player.win():
            print('yay!')
      else:
            lives -= 1
            if lives > 0:
                  level.reset()
            else:
                  print('game over')
                  lives = 3
      state = 0
```

Classes and Objects

```python
class Charleen:
    food = "Watermelon"

    def dance(self):
        print("Catch a crayon")

# Create a new class object
charleen1 = Charleen()
print(charleen1)
print(charleen1.food) # Access public variables
print(charleen1.dance()) # Call functions

# Creata another object
charleen2 = Charleen()
print(charleen2)
charleen2.food = "Mango"

# Two objects
# Same Class, Different Values
print(charleen1.food)
print(charleen2.food)

```

```python
class Sprite:
    name = "default_name"
    hometown = "default_hometown"

    # Make sure to use 'self' as a parameter
    # And use it when calling class variables
    def get_description(self):
        result = ""
        result += "Name: " + self.name + "\n"
        result += "Hometown: " + self.hometown + "\n"
        return result

    # Different from 'self' class vars and input parameter
    def set_name(self, name):
        self.name = name

    def set_hometown(self, hometown):
        if (hometown == "NYC"):
            hometown = "NYC (da best)"
        self.hometown = hometown

    # Without 'self' parameter, it is a Static class function
    def generic_greeting():
        return "Hi, how are you?"

```

```python
class Player:
    name = ""
    power = 0
    speed = 0

    # Class Constructors in python
    def __init__(self, name, power, speed):
        self.name = name
        self.power = power
        self.speed = speed

    # Calling methods from other objects in the same class
    def attack(self, other):
        if (self.speed < other.speed):
            return "Attack failed! Too slow!"
        else:
            if (self.power >= other.power):
                return self.name + " successfully attacked " + other
            else:
                return self.name + "'s attack on " + other.name + "

    def speed_boost(self):
        self.speed *= 2


Spiderman = Player("Peter Parker", 50, 2)
Hawkeye = Player("Clint Barton", 20, 3)

battle1 = Spiderman.attack(Hawkeye)
print(battle1)

Spiderman.speed_boost()
battle2 = Spiderman.attack(Hawkeye)
print(battle2)
```

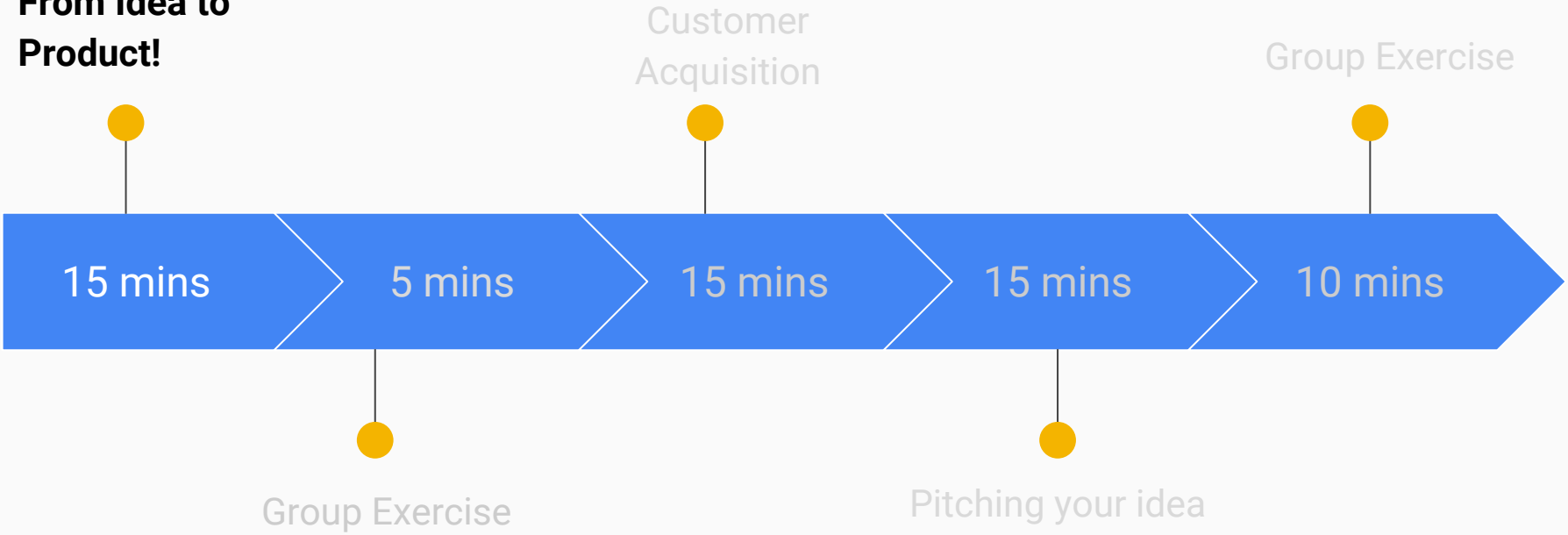
Turning an Idea into a Business

Objectives

Learn how to get from an idea into a product with a customer.

Learn how to pitch/sell your product.

From Idea to Product!



From an Idea into a Product with a Customer

What is a Product?

- It is not an idea!

What is a Product?

- It is not an idea
- It is not code!

What is a Product?

- It is not an idea
- It is not code
- It is not a prototype!


product
A good, idea, method, information, object or service created as a result of a process and serves a need or satisfies a ...
BD
BusinessDictionary

Example: Is this a product?

- I think that an app that notifies you of free cool events near you would be very helpful!

NO!

Idea

- I have code that can compute if I am meeting my saving/retirement goals.

NO!

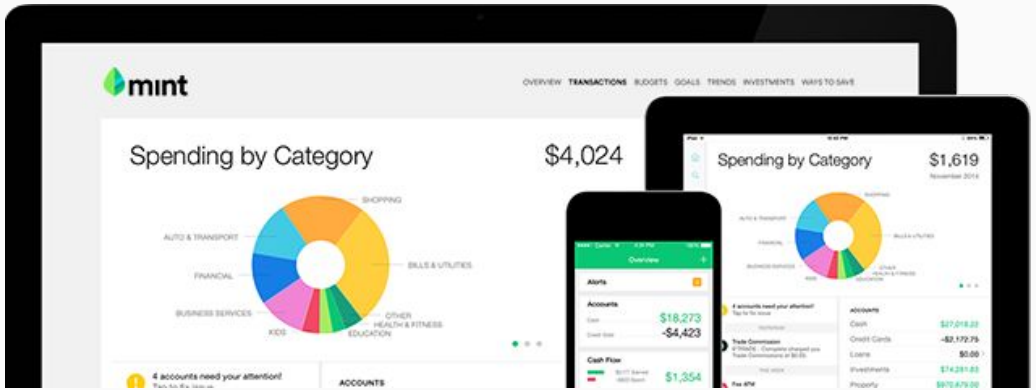
Code

Idea → Product

- I think that an app that notifies you of free cool events near you would be very helpful!

Code → Product

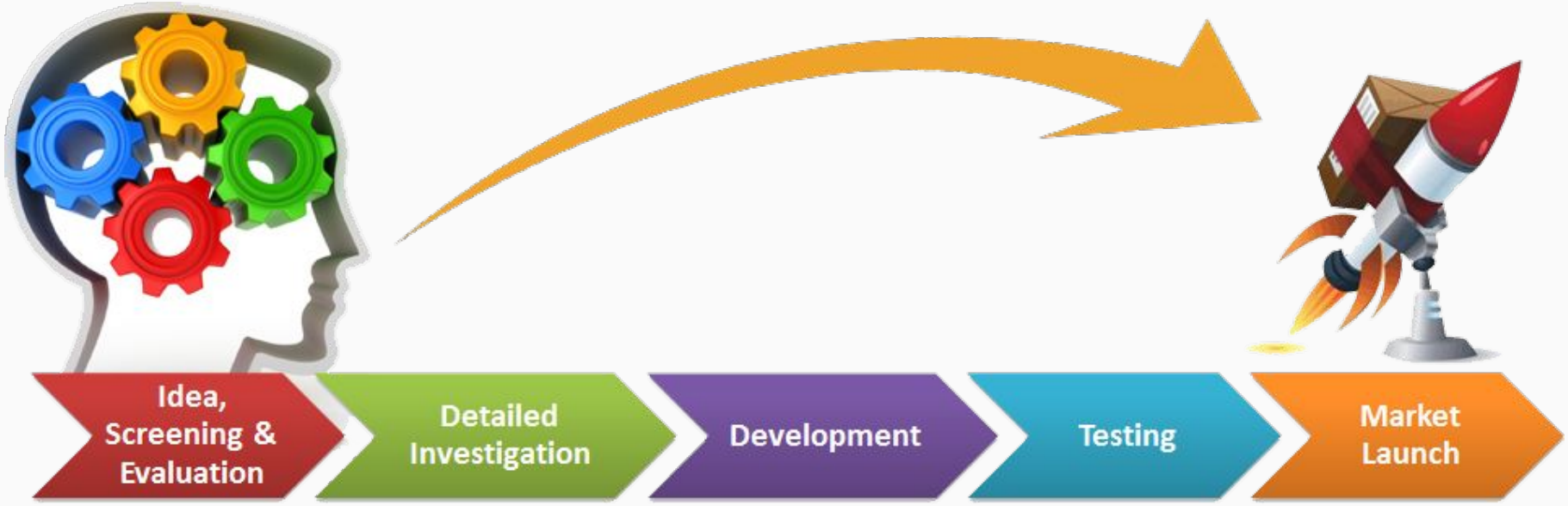
- I have code that can compute if I am meeting my saving/retirement goals.



A product is the result of a Process

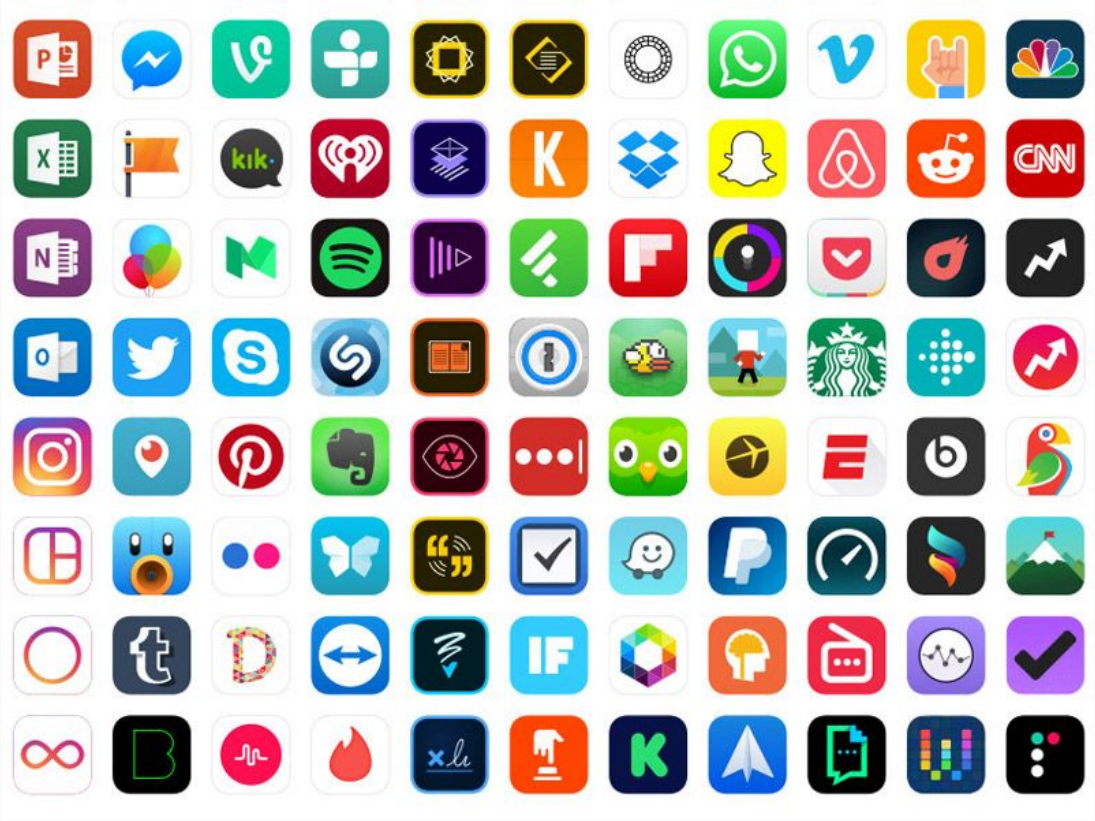
Example: Mint

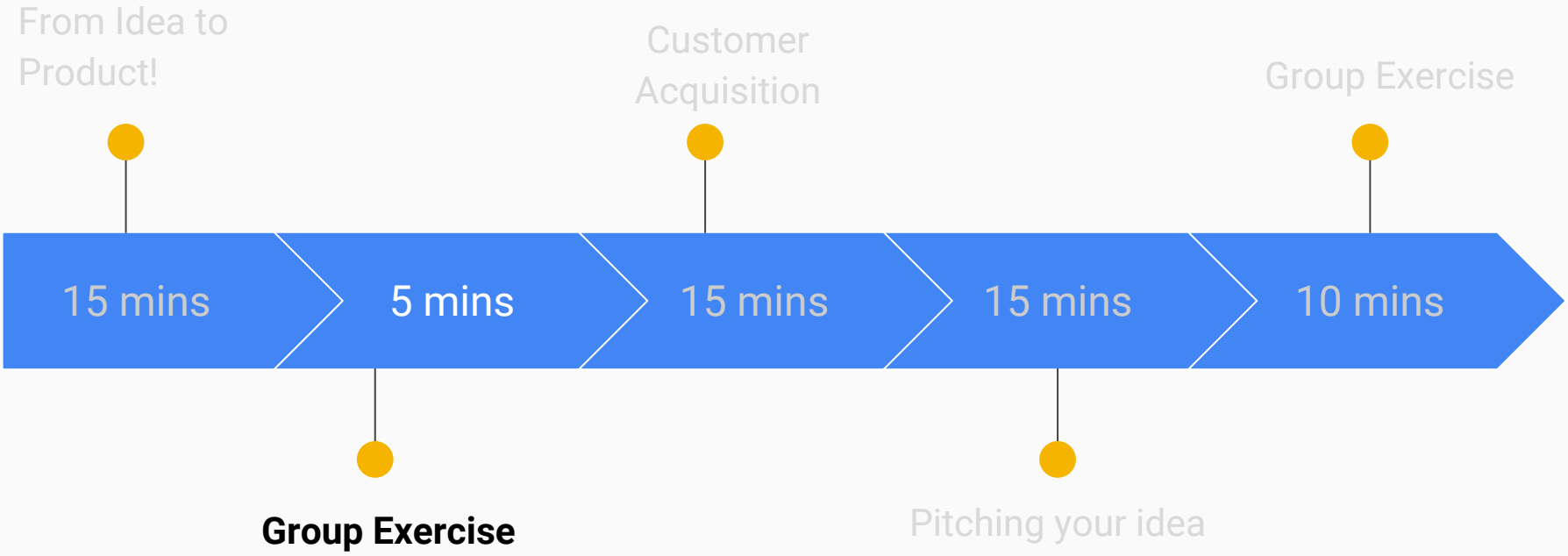
Idea	It would great if people could track and visualize their saving goals and progress.
Concept	The application would collect all of one's bank accounts data and analyze against user set budgets and saving goals.
Prototype	Code that can get data from all of one's back accounts.
Product	**Mint app** (android & iphone)



Product Development Process

Once, they were ideas!




From Idea to Product!
Customer Acquisition
Group Exercise
15 mins
5 mins
15 mins
15 mins
10 mins
Group Exercise
Pitching your idea

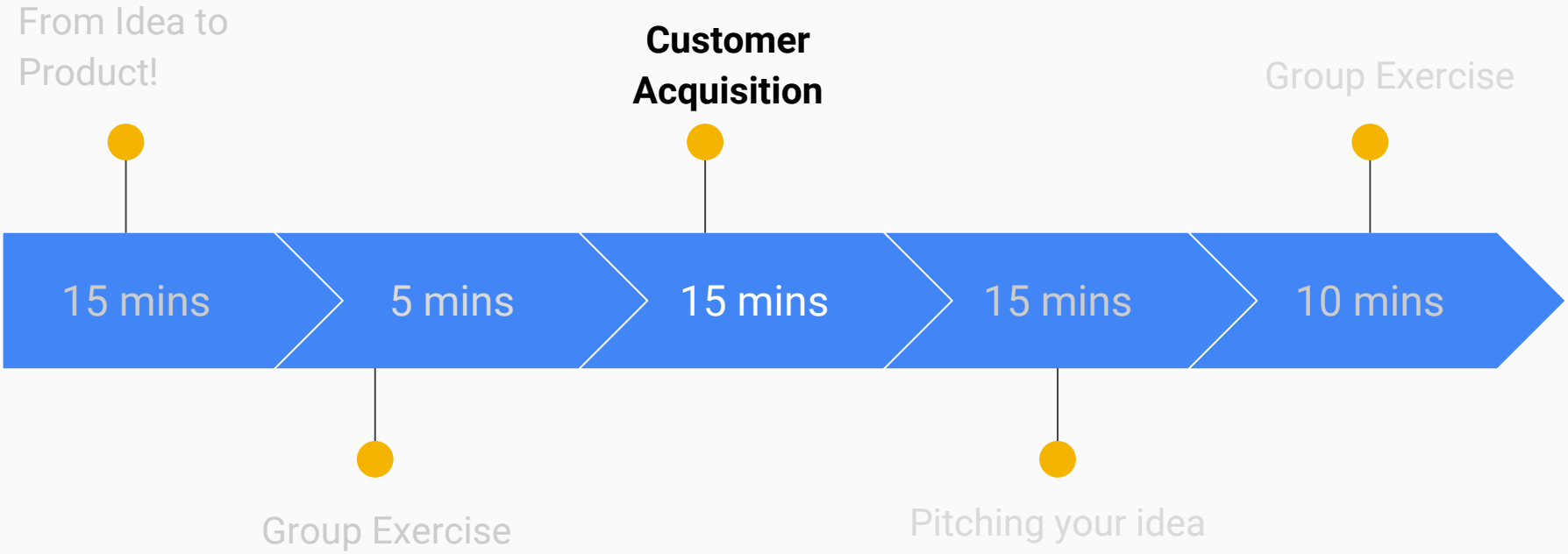
Group Exercise!!!

In your groups, take 5 minutes to:

1) Clearly state your final idea.

2) Determine a product from your idea.

3) Discuss a potential prototype.


From Idea to Product!
Customer Acquisition
Group Exercise
15 mins
5 mins
15 mins
15 mins
10 mins
Group Exercise
Pitching your idea

Process of Acquiring a Customer

Who is a Customer?

- It is a user willing to pay for your product

Who is a Customer?

- It is a user willing to pay for your product
- It is not someone who says “your idea is so cool!”

Who is a Customer?

- It is a user willing to pay for your product
- It is not someone who says “your idea is so cool!”
- It is not someone who just listens to you and nods yes yes yes!

There are 4 Types of Customers

A) Early adopters

- Always quick/first to try new ideas/products/tech
- Do it for the reputation
- Very few
- Often have money/voice/power

There are 4 Types of Customers

A) Early adopters

B) Early majority

- Relatively quick to adopt new ideas/products/tech
- Make decisions based on utility and practical benefits over coolness

There are 4 Types of Customers

A) Early adopters

B) Early majority

C) Late majority

- Average at adopting new ideas/products/tech
- Share a lot of traits with early majority but tend to be overly cautious before committing

There are 4 Types of Customers

A) Early adopters

B) Early majority

C) Late majority

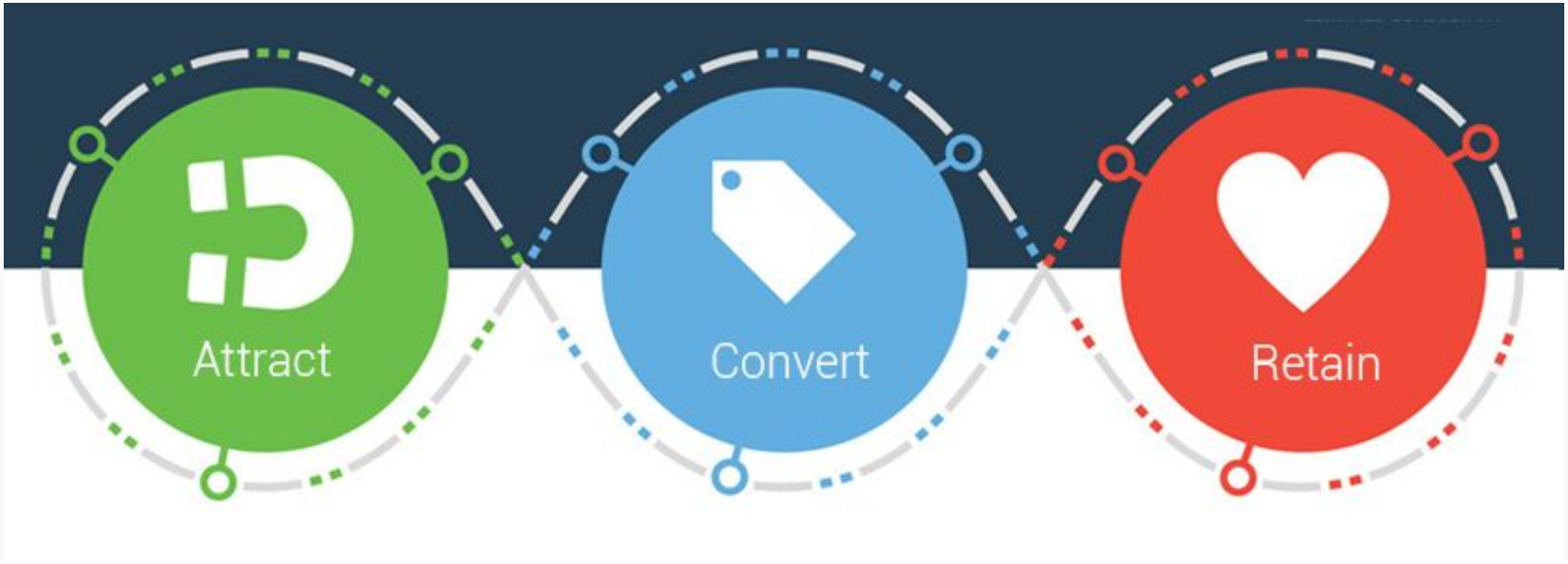
D) Laggards

- Most people
- Last to adopt new ideas/products/tech
- When forced to or product is now famous/proved to be a great product

Who is your Customer?

Your customer . . .

- Is not everyone!
- Is not all your friends and family and classmates and ...
- Must be very specific, for example:
    1. Children from Asian middle-class families who know/speak very limited English.
    2. Women who like reading Steve Harvey’s books.
    3. Couples that frequently travel together to tropical destinations and spend about $1000 on vacation annually.



Customer Acquisition → Customer Engagement

Customer Acquisition: COCA & LTV

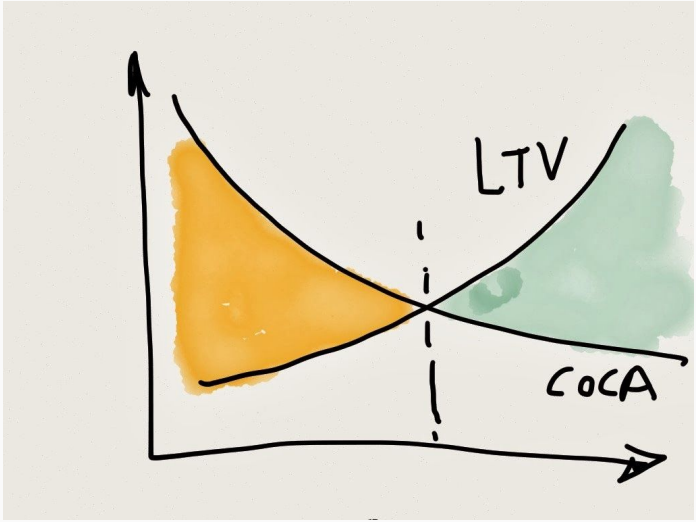
COCA:

- Cost Of Customer Acquisition
- $\frac{\text{sales \& marketing costs}}{\text{\# of new customers}}$

LTV:

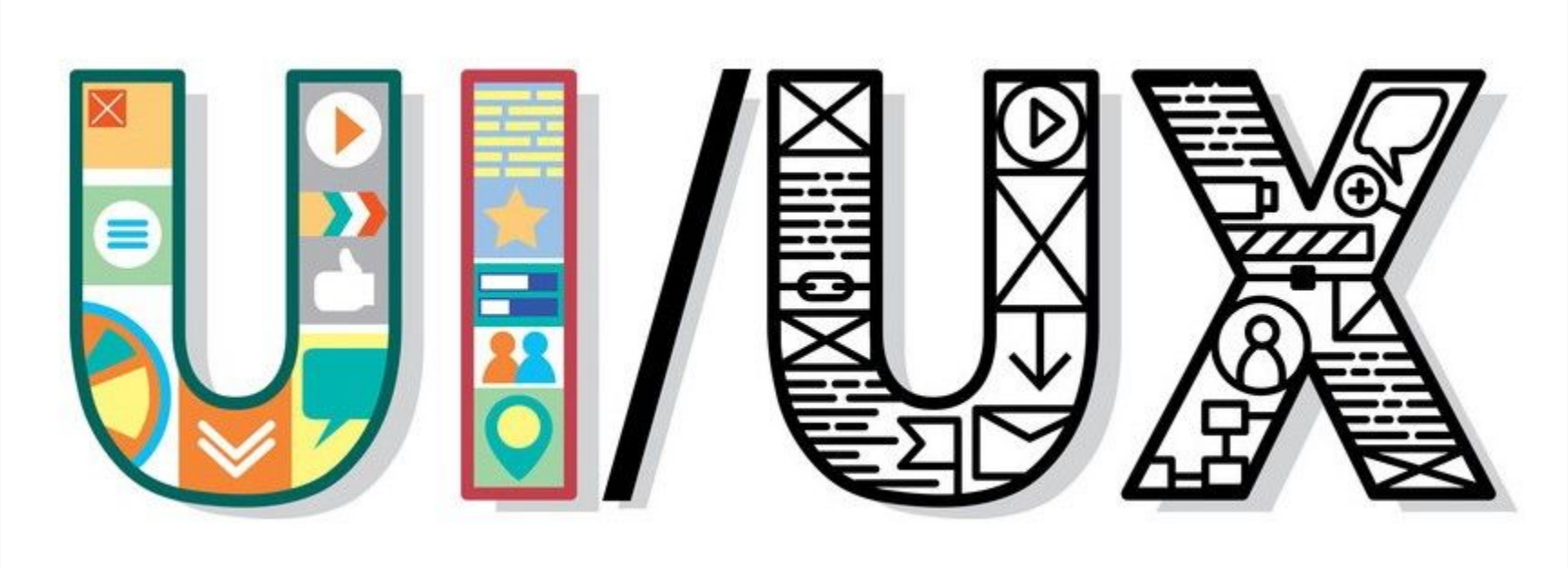
- Life Time Value of a customer
- Total lifetime profit you expect to make from a customer

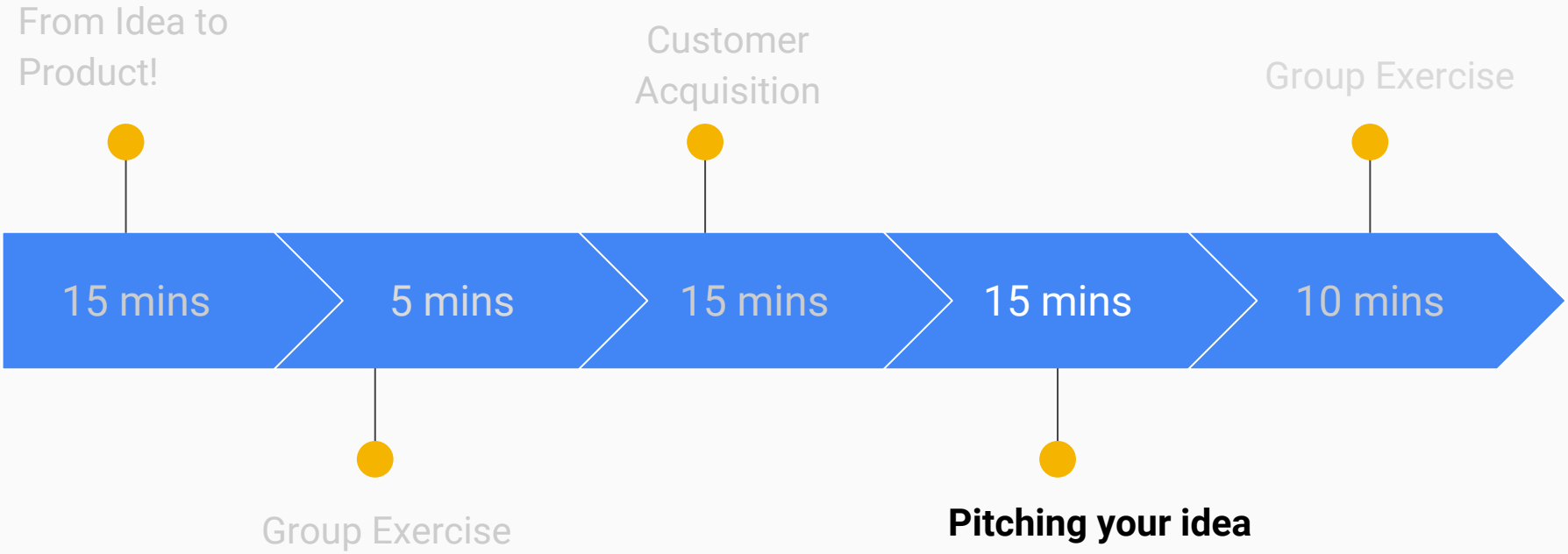
Product successful when . . .



LTV ≥ 3 * COCA

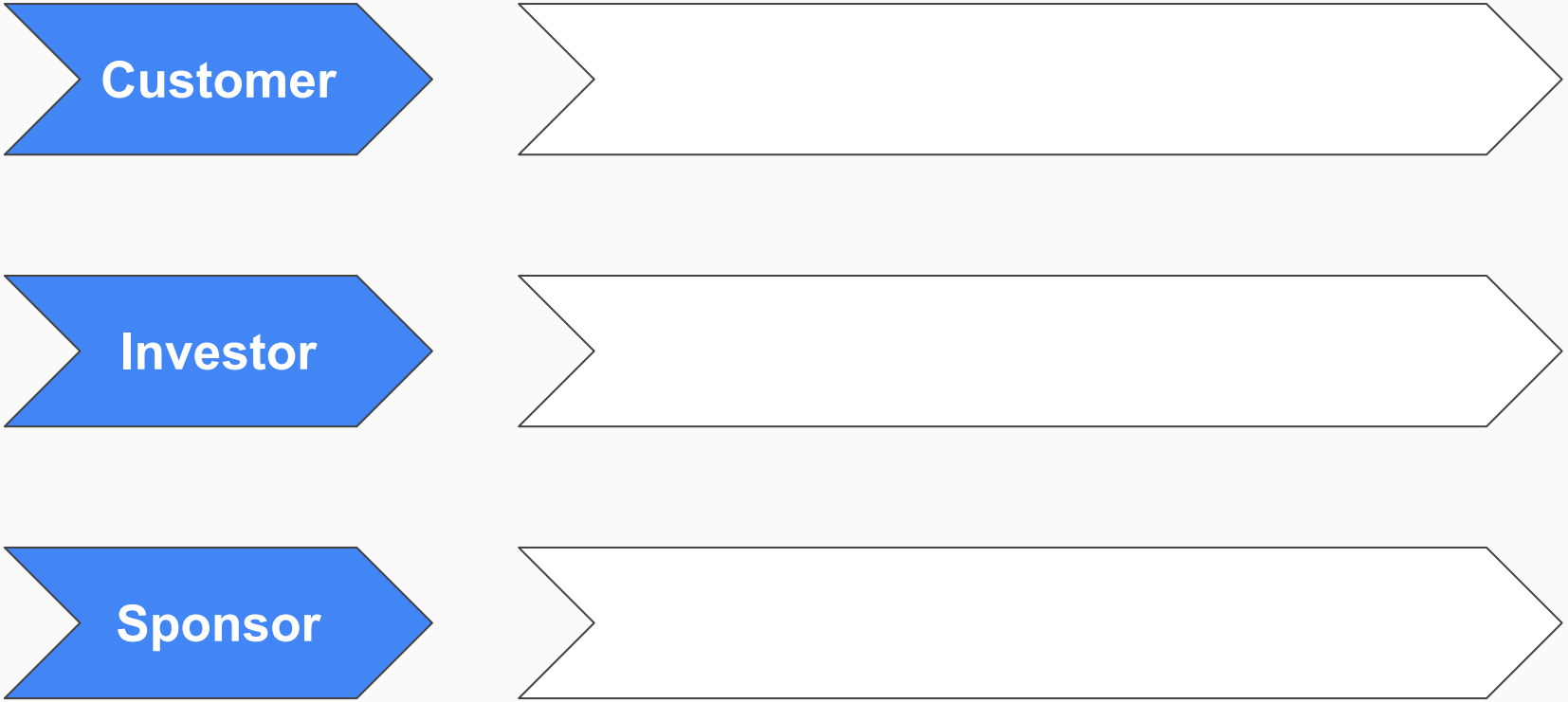
Companies spend millions



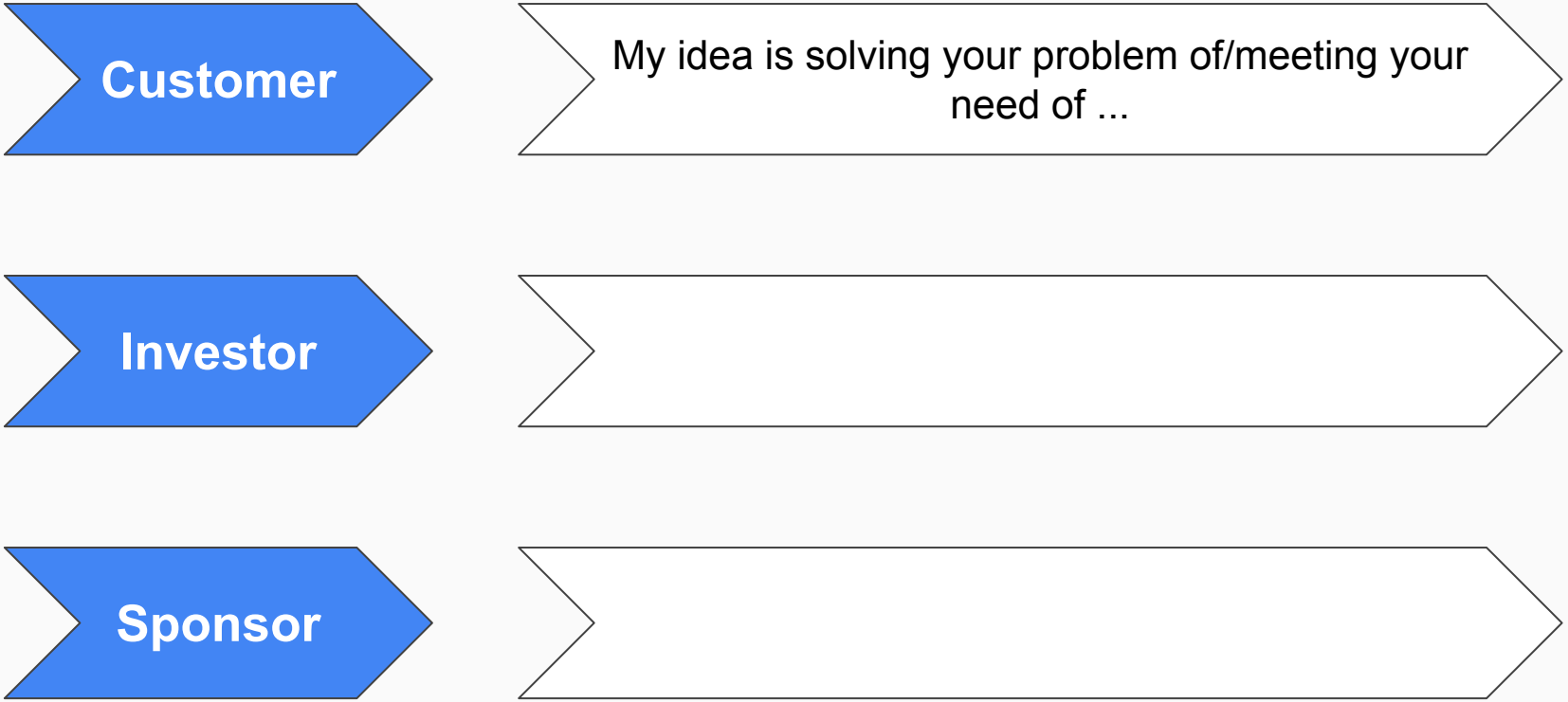

From Idea to Product!
Customer Acquisition
Group Exercise
15 mins
5 mins
15 mins
15 mins
10 mins
Group Exercise
Pitching your idea

Pitching your Idea/Product

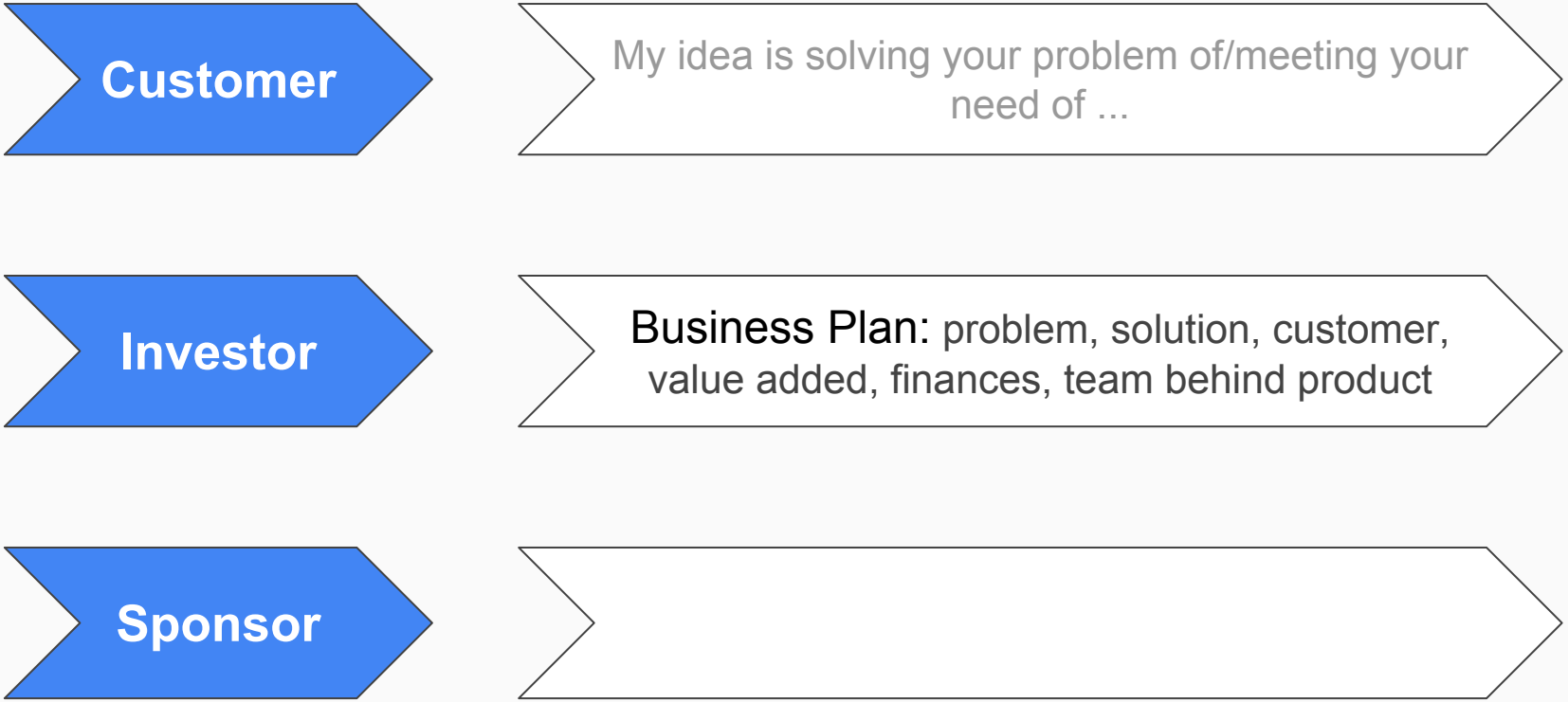
Different Pitch for Different Audiences



Different Pitch for Different Audiences

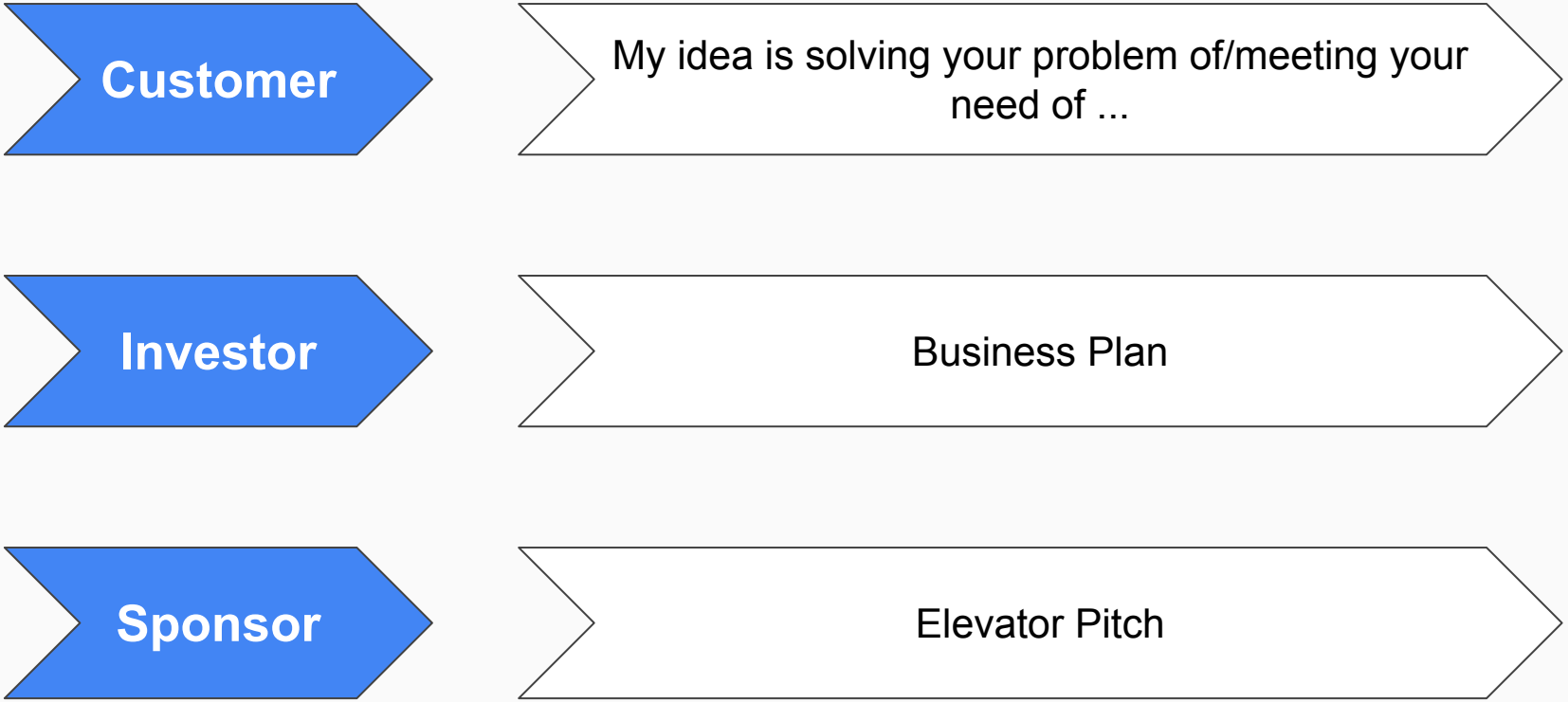


Different Pitch for Different Audiences



Different Pitch for Different Audiences

Different Pitch for Different Audiences



Any pitch should:

- State problem & solution

Any pitch should:

- State problem & solution
- Be clear & concise!

Any pitch should:

- State problem & solution
- Be clear & concise!
- Be persuasive!

Any pitch should:

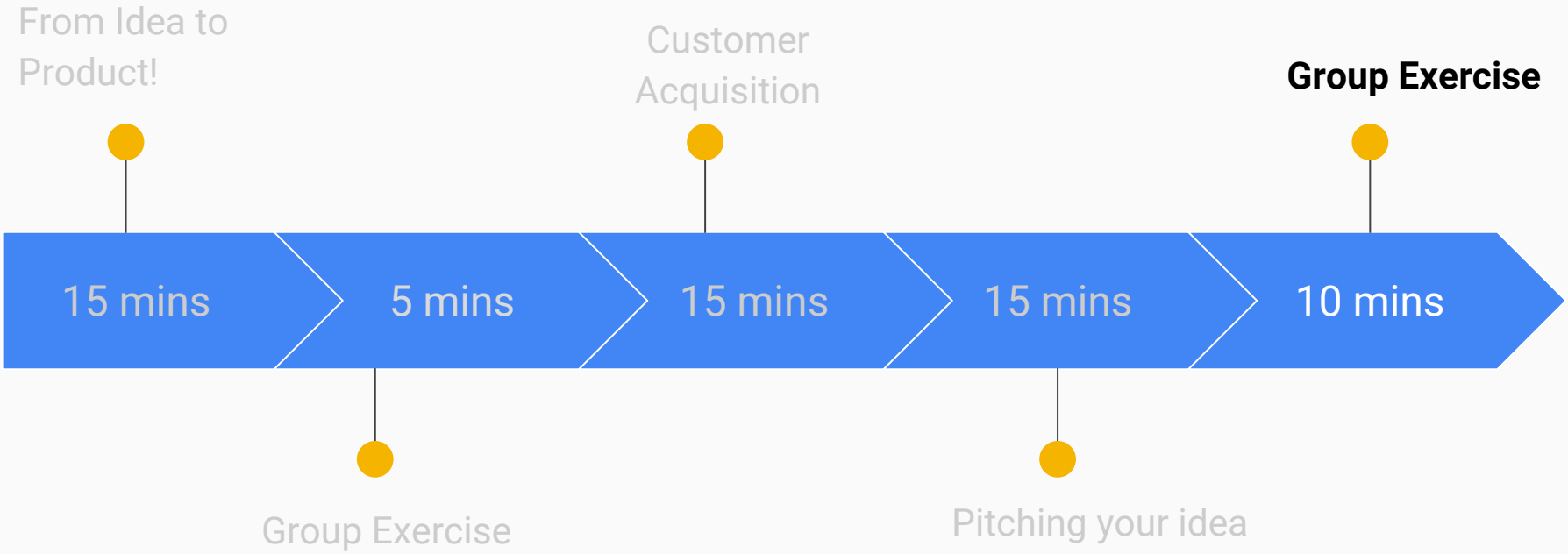
- State problem & solution
- Be clear & concise!
- Be persuasive!
- Be passionate!

Any pitch should:

- State problem & solution
- Be clear & concise!
- Be persuasive!
- Be passionate!
- Extend an invite to continue conversation!

Any pitch should:

- State problem & solution
- Be clear & concise!
- Be persuasive!
- Be passionate!
- Extend an invite to continue conversation!


From Idea to Product!
Customer Acquisition
Group Exercise
15 mins
5 mins
15 mins
15 mins
10 mins
Group Exercise
Pitching your idea

Friday Final Deliverables

1. **Presentation:** You will pitch your idea to us!
2. **Prototype:** A working portion of your product/idea!

THEME: Community

Group Exercise!!!

In your groups, take 10 minutes to:

1) Identify what need your idea is solving.

2) Define 2-3 potential customers. Be very specific!

3) Determine 3 things you'd tell an investor in order to get capital for your idea.

Project Work Time!

MIT-THEi-2018

Repositories 1 | People 2 | Teams 0 | Projects 0 | Settings

Search repositories... | Type: All | Customize pinned repositories | New

lesson-examples

Updated a day ago

People 2

charleenwang
Charleen

leoncheng57
Leon Cheng

Invite someone

Survey Time!

https://goo.gl/4NoUeq