OOPs i was having fun

(no i wasn't)

Ishan Dasgupta (ishandas), Sohan Subudhi (ssubudhi),
Uzair Nawaz (uzairn), Sathvik Chemudupati (sathvikc)

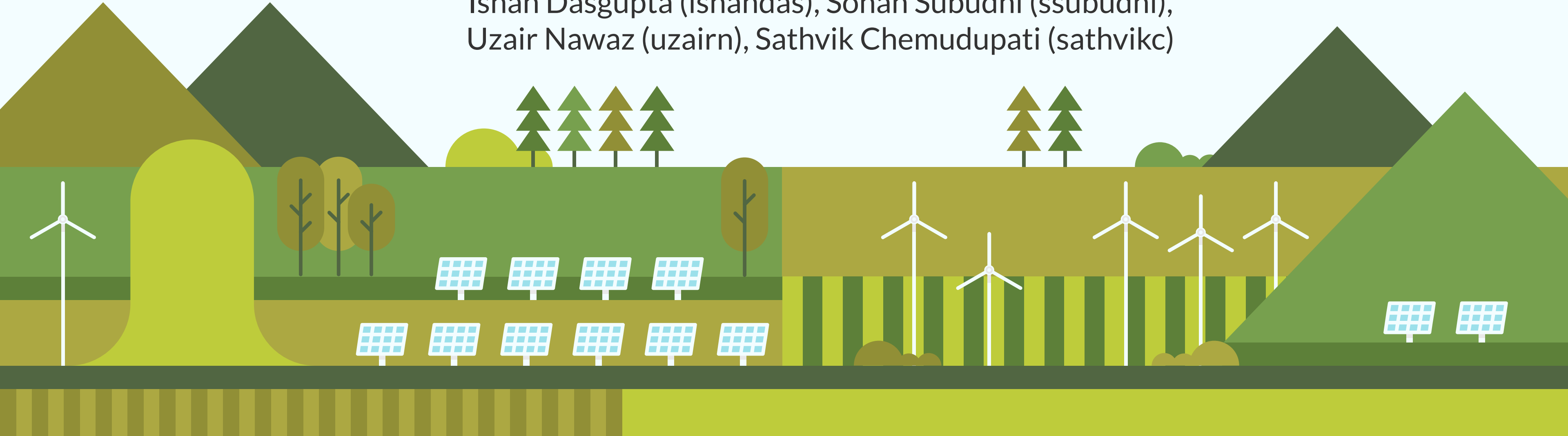


Table of contents

01

Heap

02

Core OOP Features

03

Additional Features

04

Libraries

05

Demo

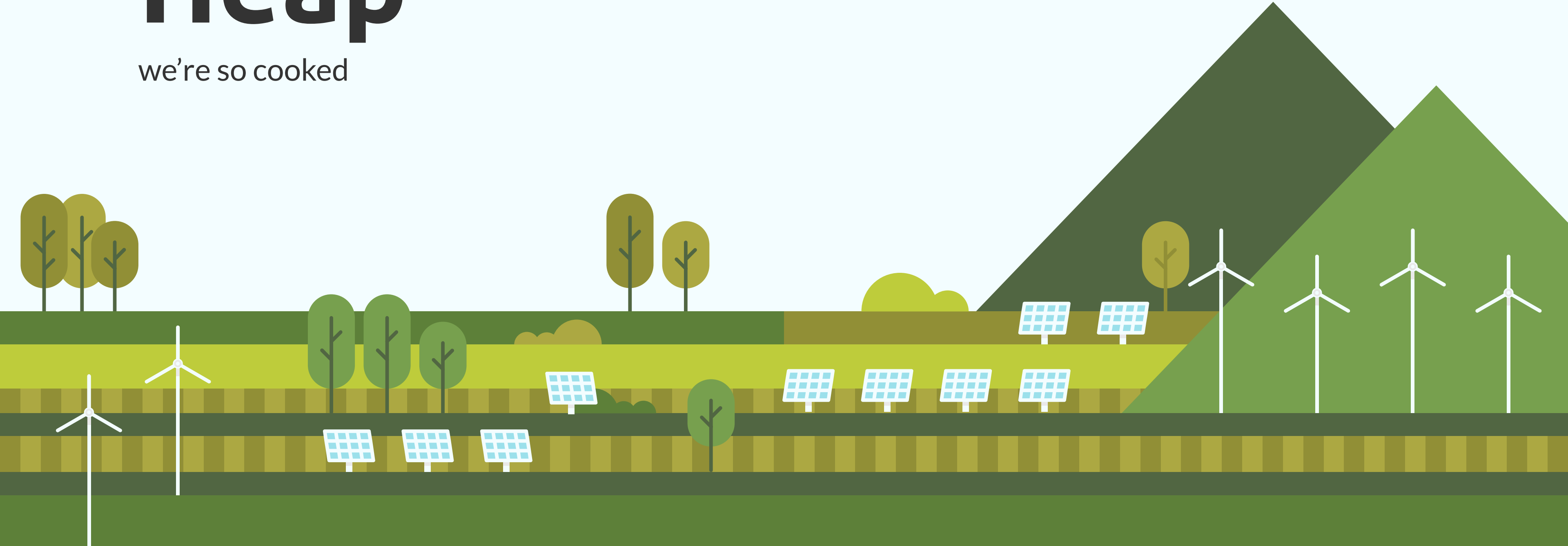
06

Analysis + Next Steps

01

Heap

we’re so cooked



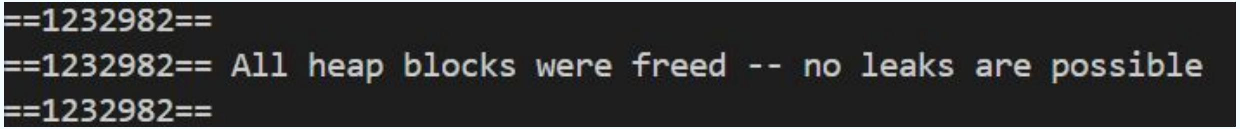
We are Environmentally Friendly

No “Leaks”

→

Save the world

- Since objects can be of a variable size, they need to be dynamically allocated. We won’t know the size at compile time.
- Enter malloc
- But instead, we malloc onto our own simulated heap from p5
- All those memory leaks are sealed out of sight
- We just have to optimize our heap to make it comparably more space efficient.

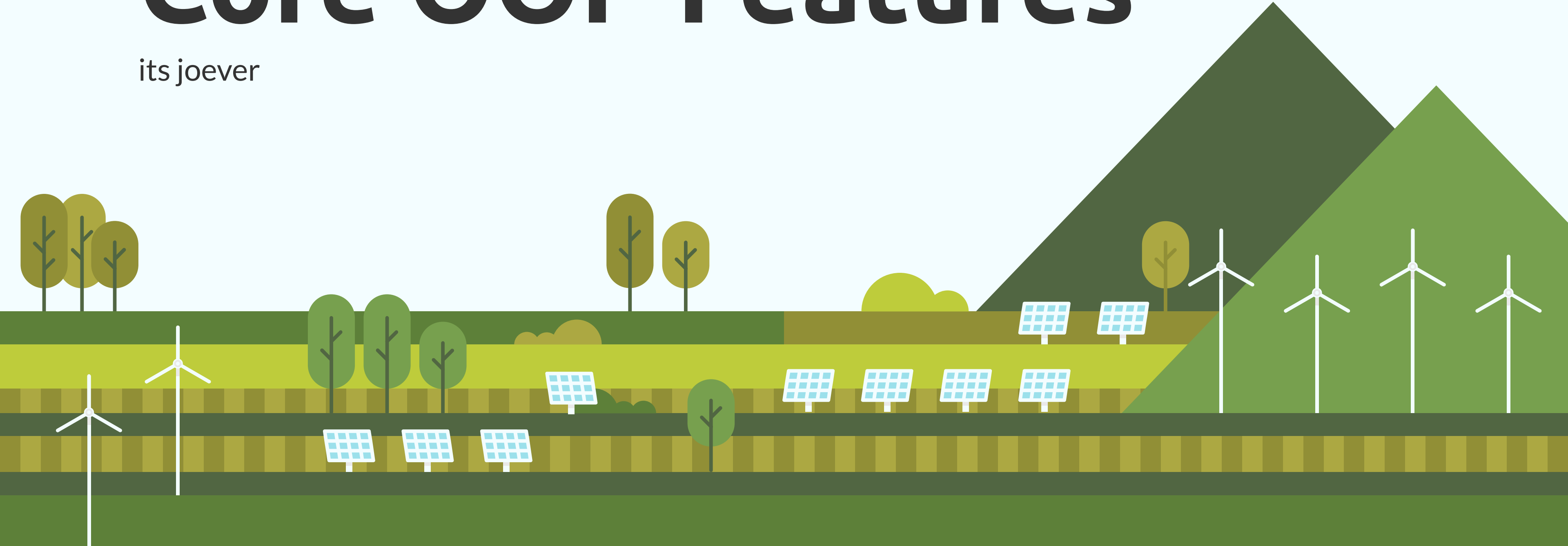


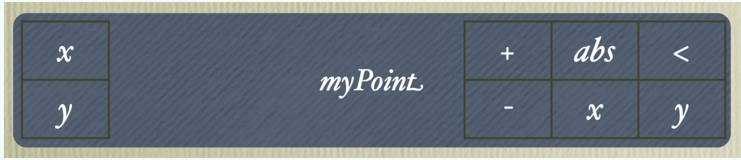
```
==1232982==
==1232982== All heap blocks were freed -- no leaks are possible
==1232982==
```

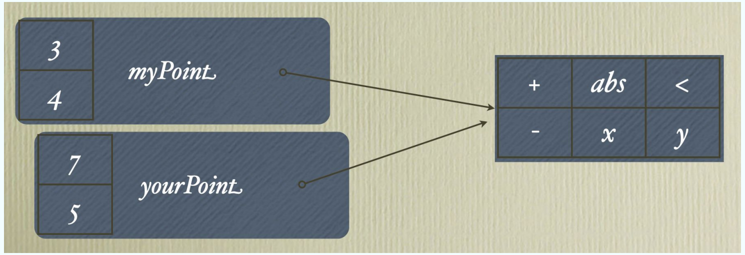
02

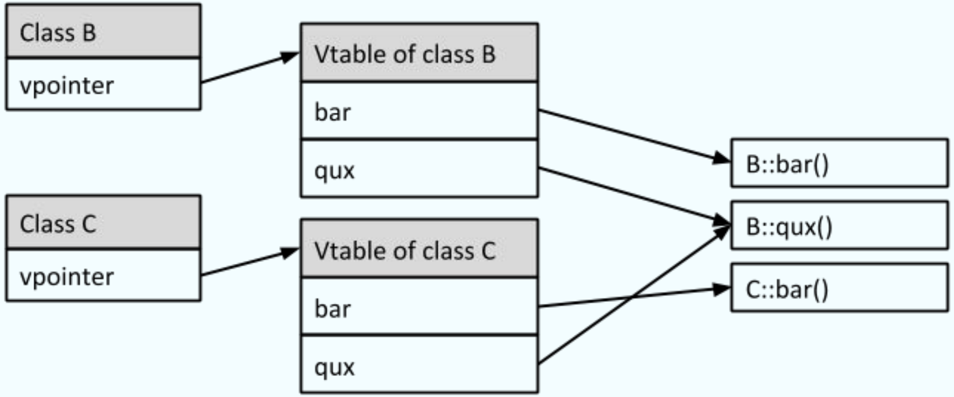
Core OOP Features

its joever









Class B

vpointer

Vtable of class B

bar

qux

Class C

vpointer

Vtable of class C

bar

qux

B::bar()

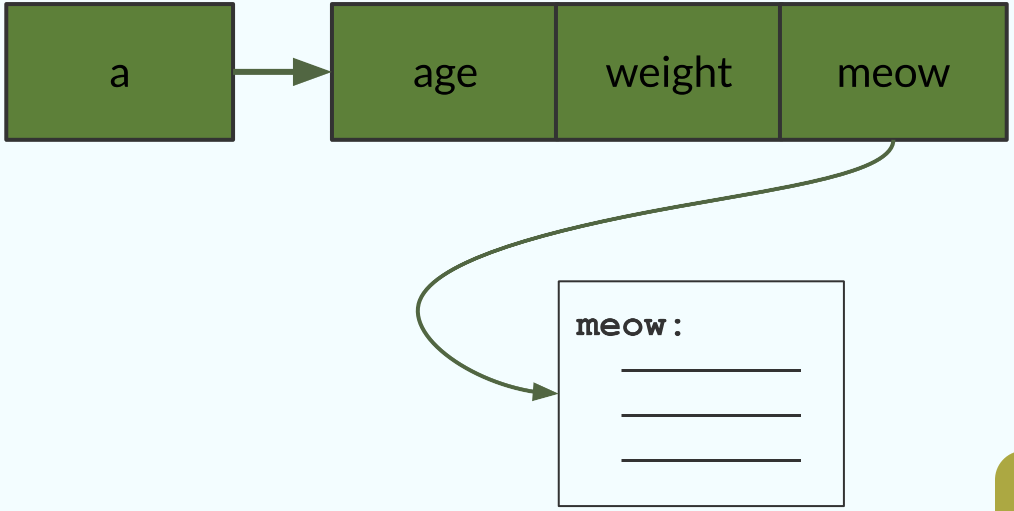
B::qux()

C::bar()

Object Creation

Now that we have a heap, we can dynamically allocate objects!

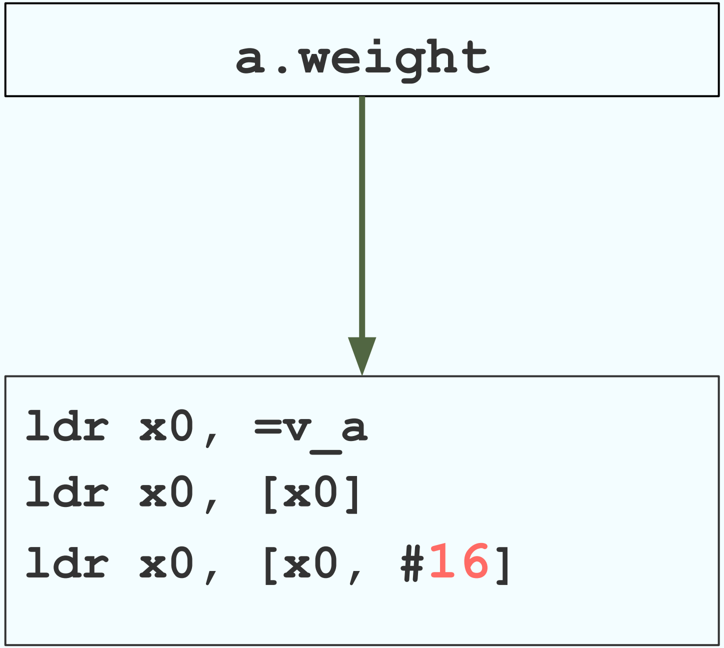
```
class Cat {
    age = 0
    weight = 0
    meow = fun {
        print self.age
    }
}

Cat a = new Cat()
```



Accessing Object Members

```
class Cat {
    age = 0
    Toy favToy
    weight = 0
    meow = fun {
        print self.age
    }
}

Cat a = new Cat()
```

16 bytes



```
a.weight
```

```
ldr x0, =v_a
ldr x0, [x0]
ldr x0, [x0, #16]
```

Storing Types

- Compiler maintains an internal hierarchy of class nodes
- Global variables are mapped to a specific class node to track types
- Class nodes contain:
    - Pointer to the parent class node (that we inherit from)
    - Data for each member variable
        - Position in the class (used to calculate memory index offset)
        - Whether it’s private/public
        - Type for each member variable

Object Oriented Features

Inheritance

Allow new classes to build on existing classes, extending their features

Polymorphism

Allowing objects to “morph” into different types

Encapsulation

Hiding implementation details from the user of a class

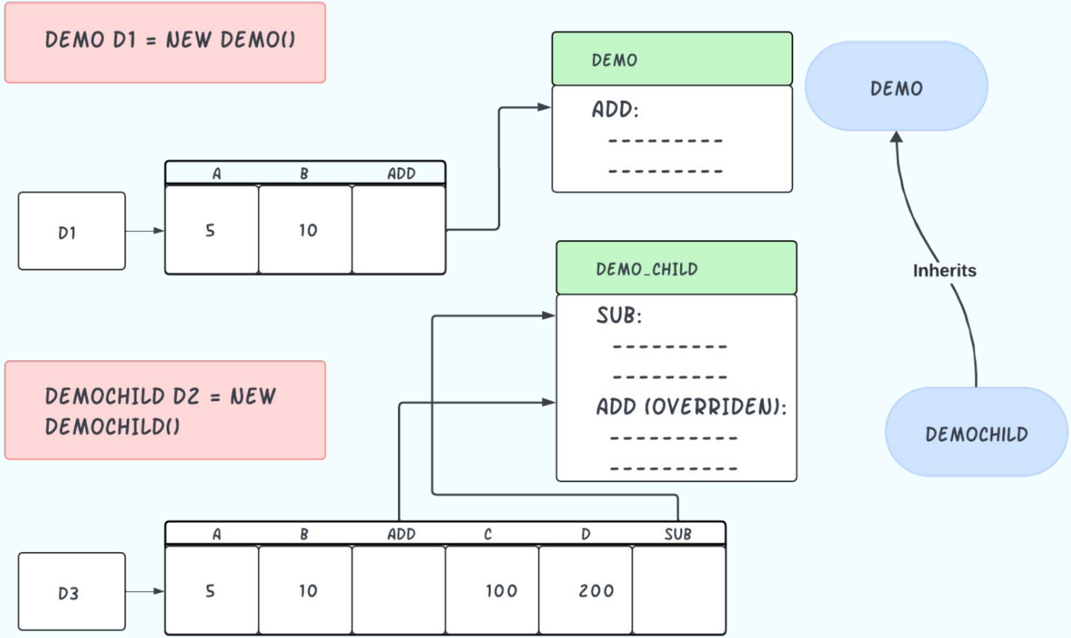
Inheritance + Polymorphism

```
CLASS DEMO {
    A = 5
    B = 10
    ADD = FUN {
       RETURN A + B
    }
}
```

```
DEMO D1 = NEW DEMO()
```

```
CLASS DEMOCHILD
EXTENDS DEMO {
    C = 100
    D = 200
    SUB = FUN {
       RETURN C - D
    }
    ADD = FUN {
       RETURN C + D
    }
}
```

```
DEMOCHILD D2 = NEW
DEMOCHILD()
```



D1

A	B	ADD
5	10	

DEMO

```
ADD:
  ---------
  ---------
```

D3

A	B	ADD	C	D	SUB
5	10		100	200	

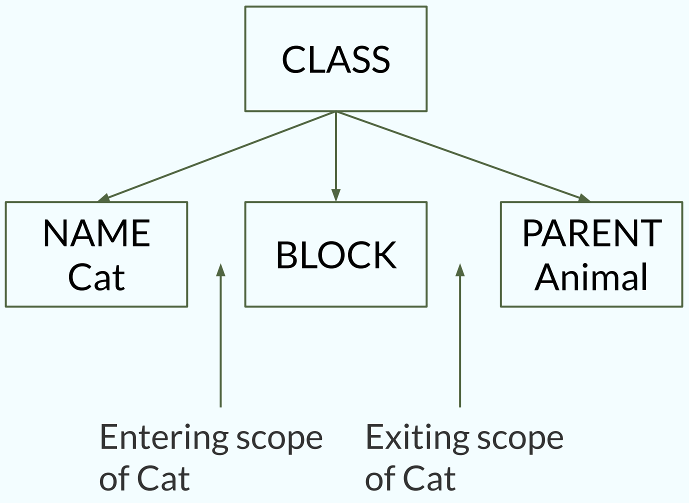
DEMO_CHILD

```
SUB:
  ---------
  ---------
ADD (OVERRIDEN):
  ----------
  ----------
```

Encapsulation

Issue Cases	
A private member of an object’s inherited class is accessed	A private member of an object’s own class is accessed and we are not in the scope of the class

Encapsulation

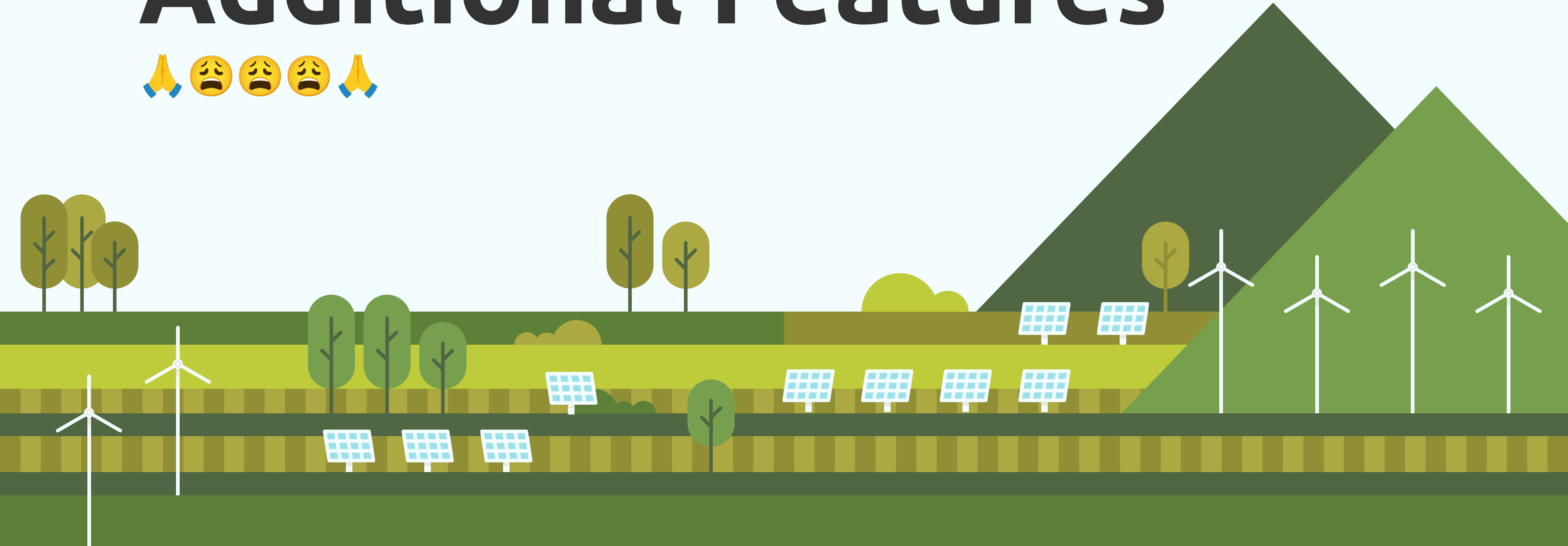


```
class Animal {   ← Current scope = Animal
    private age

}
class Cat extends Animal{ ←
    age = 0   ← Uh oh      Current scope = Cat

    private meow = fun {
        print self.age
    }
}

                        Current scope = null
Cat a = new Cat()
print a.meow()      ← Uh oh
```

03

Additional Features

🙏😩😩😩🙏



Arrays

Fun didn’t feel COMPLETE without the most fundamental data structure: arrays

```
int[] numbers = new int[10]

Object objects[] = new Object[10]
```

Just like Java, we support both array declaration syntaxes!!!!

What else do we support 🤔

```
[][]int[] array[] = new i[]nt[10]
```

Compilation Errors 👺👺👺

- I can’t write programs in fun 😭
- ERROR MESSAGES!!!!!
- We print instructions to a buffer and output it all at once at the end of the program

```
if 1 {
  print 3
```

UNMATCHED CLOSE_CURLY ERROR:

No associated CLOSE_CURLY with the OPEN_CURLY (token #2)

```
class Cat {
    private weight = 10
}

Cat c = new Cat()
print c.weight
```

Private access detected.
Compilation rejected.

```
class Cat {
    weight = 10
}

class Tiger extends Cat {
  danger = 10
}

Cat c = new Tiger()
print c.danger
```

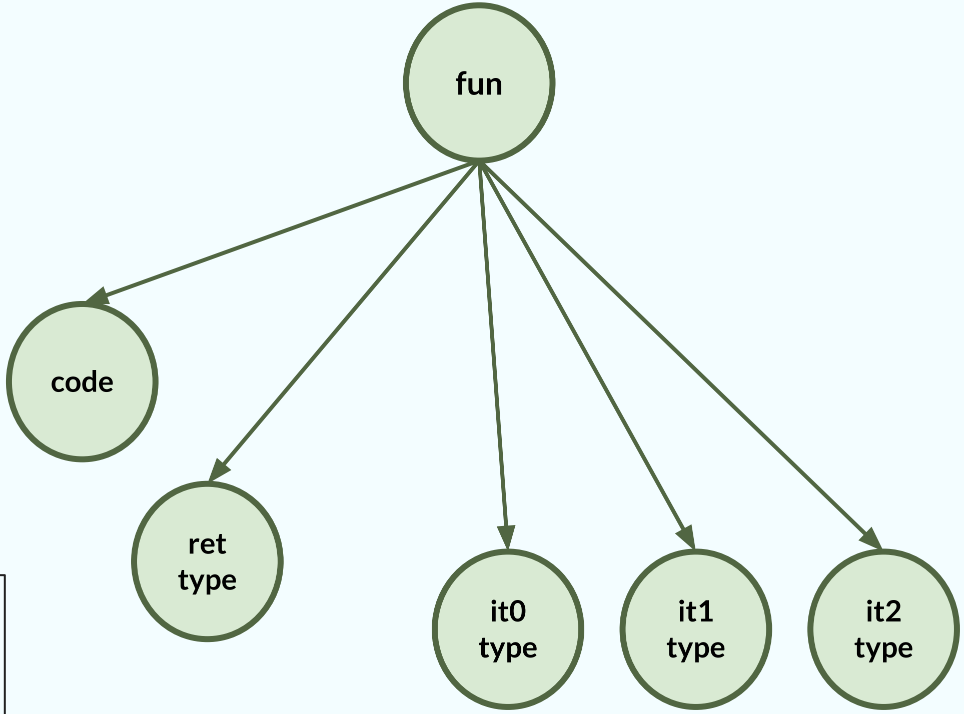
UNDEFINED MEMBER ERROR
----
an object had an undefined member trying to be accessed: danger

Multiple Parameters!

- Each parameter is pushed onto the stack
- Each parameter is reloaded into the data section variables after the return statement
- Number of parameters a given function has is implicitly stored by the AST as the number of children

```
fun (Animal, int, Dog) -> (Animal) {
    return it0
}
```

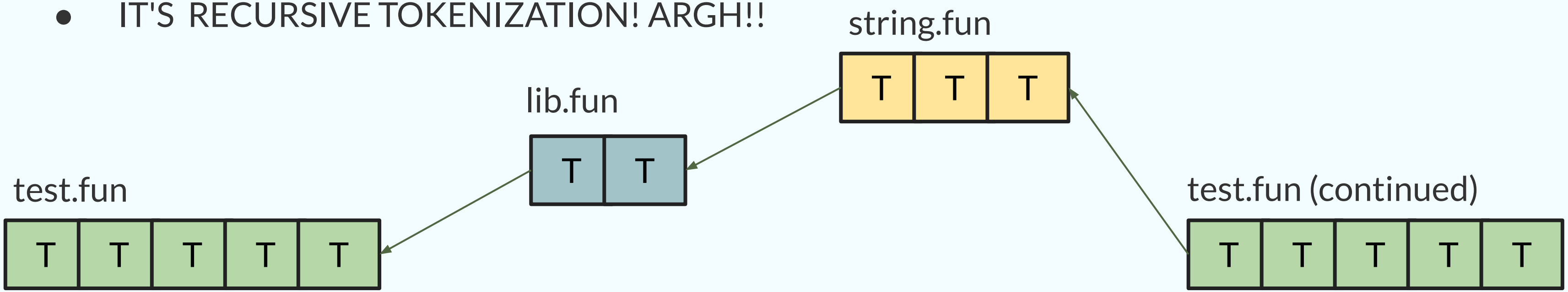
The AST node to support this



#include

#include lib/hashmap.joy

- Useful for open sourcing plans in the future to take over the world
- Useful for fun libraries
- IT'S RECURSIVE TOKENIZATION! ARGH!!



string.fun

T T T

lib.fun

T T

test.fun

T T T T T

test.fun (continued)

T T T T T

Further Miscellaneous Features

- `printc` allows us to print characters
- Standalone function calls as statements
- Multidimensional Arrays

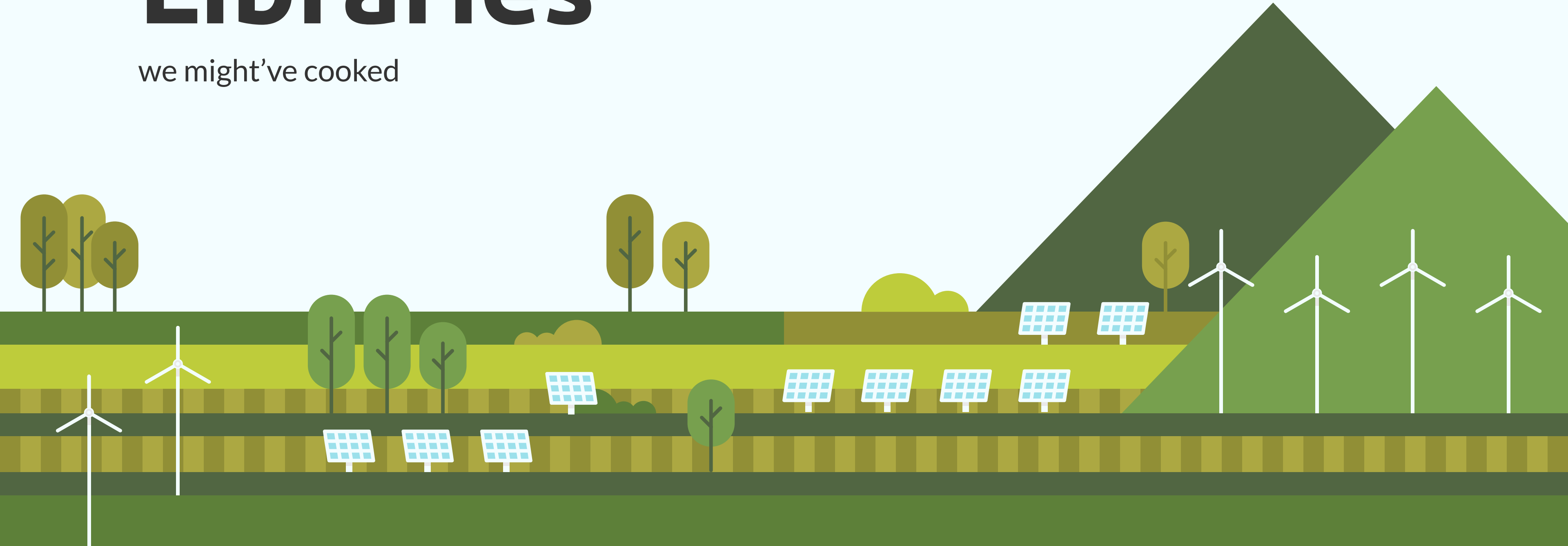
Multidimensional Arrays

```
int[] arr = new int[3]
arr[0] = new int[4]
arr[1] = new int[4]
arr[2] = new int[4]

arr[1][2] = 3
print arr[1][2]
```

04

Libraries

we might’ve cooked



Fun Libraries!!! 🔥🔥🔥

- Generics are gross 🤢🤮🤮 → How can we make general fun data structures?
- Everything is stored as an int internally as in we only have int primitives (int primitives and pointers)
- Therefore, each data structure can take input it as ints, and programmers can cast results to their needed object type

String

ArrayList

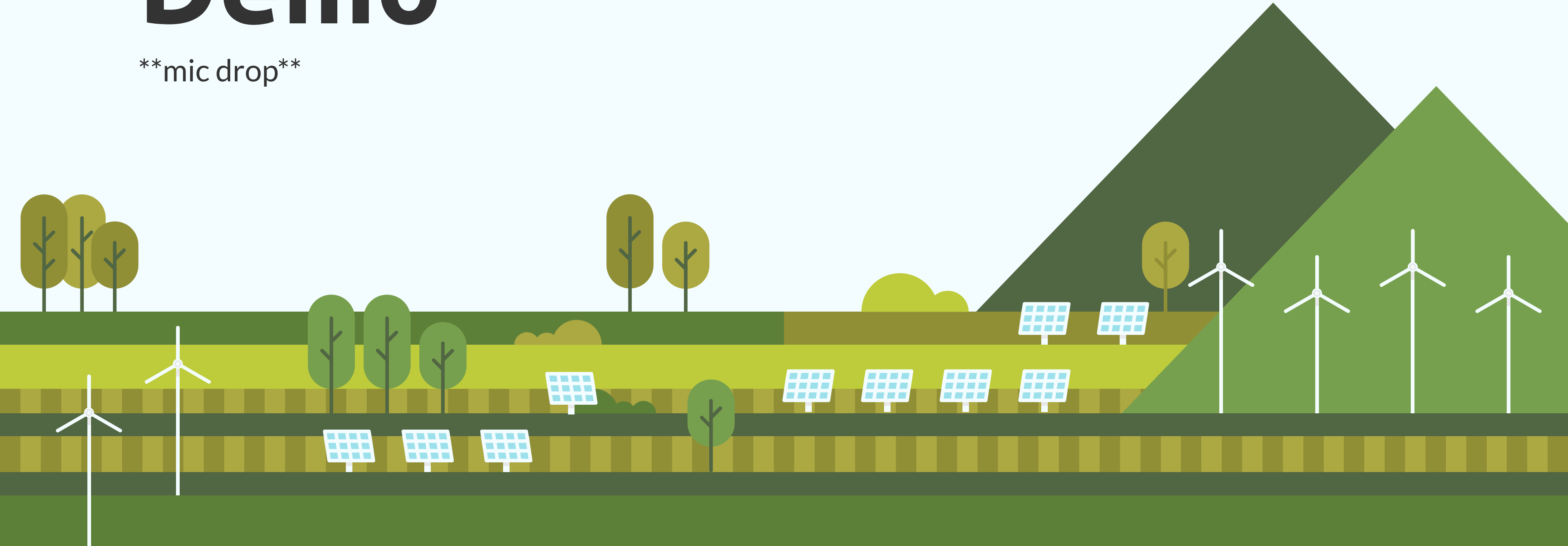
HashMap

LinkedList

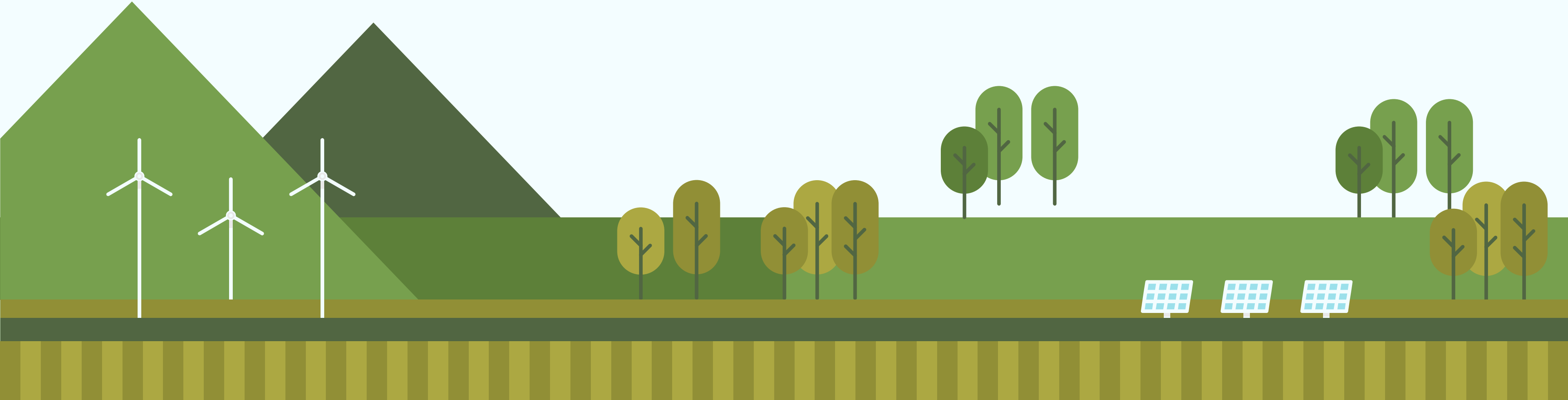
05

Demo

mic drop



“Let's do a live demo”

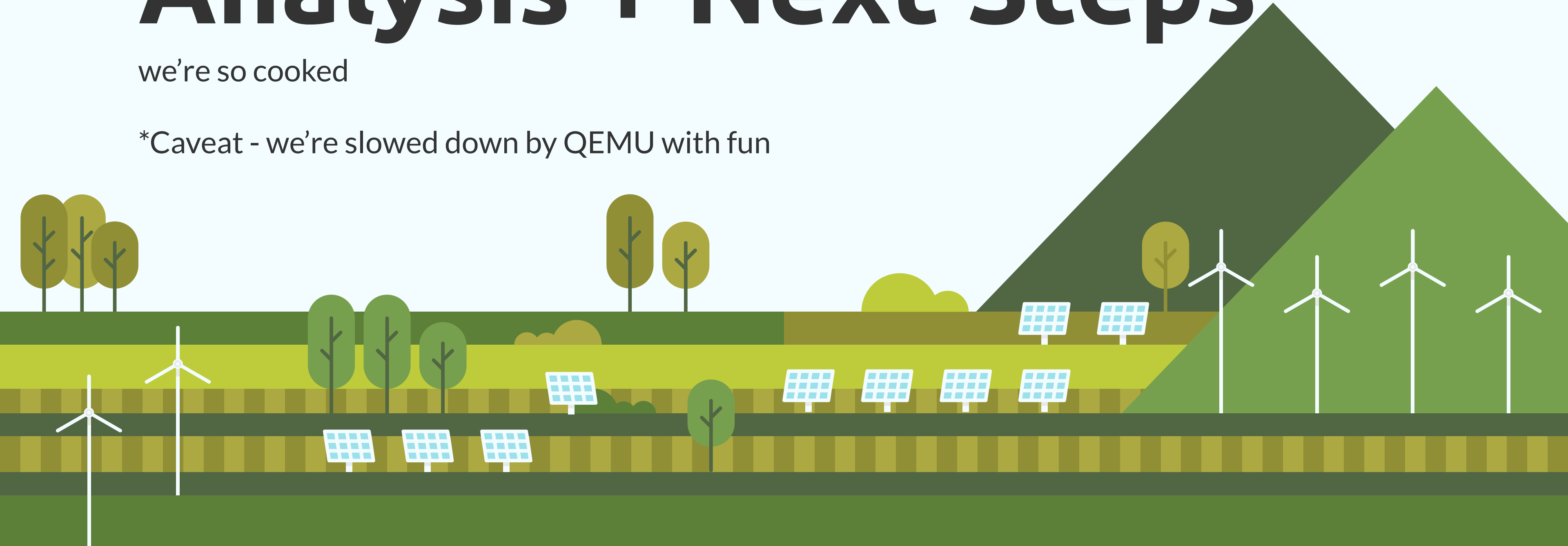


06

Analysis + Next Steps

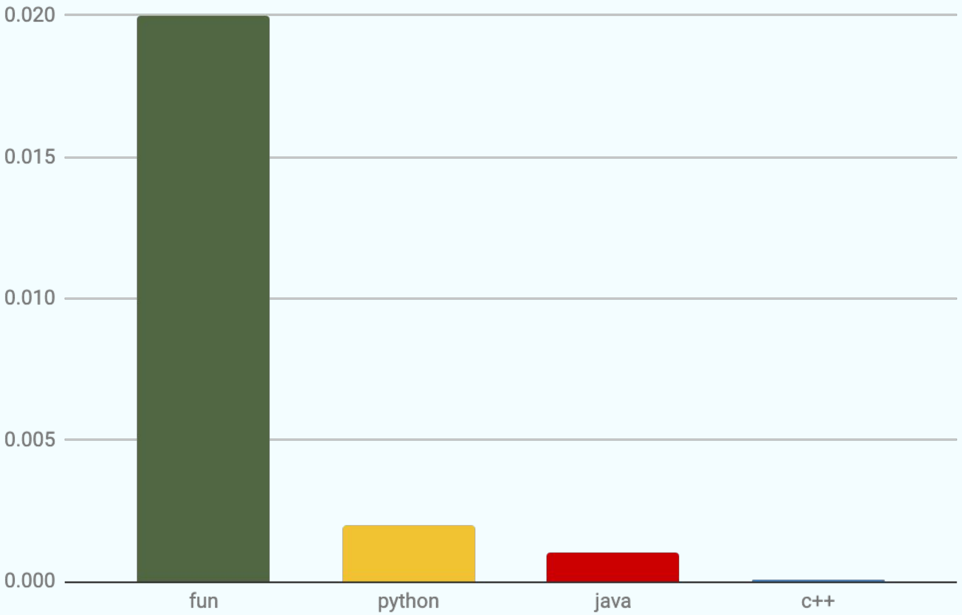
we’re so cooked

*Caveat - we’re slowed down by QEMU with fun



Analysis

```
class MyClass {
    x = 5
}

i = 0
while (i < 10000) {
    MyClass mc = new MyClass()
    i = i + 1
}
```



Analysis

```
class MyClass {
        x = 5
        myPrint = fun (int) -> (int) {
            self.x = self.x + 1
        }
}

i = 0
while (i < 10000) {
        MyClass mc = new MyClass()
        mc.myPrint()
        i = i + 1
}
```

fun	0.03 s
python	0.005 s
java	0.002 s
c++	.000098 s

Analysis

```
class MyClass {
        x = 5
        increment = fun (int) -> (int) {
            self.x = self.x + 1
        }
}

class MyClassChild extends MyClass {

}


i = 0
while (i < 10000) {
        MyClassChild mc = new MyClassChild()
        mc.increment()
        i = i + 1
}
```

fun	0.01 s
python	0.007 s
java	0.001 s
c++	.000024 s

Next Steps

Heap Improvements

Adding a Buddy Allocator for smaller allocations

Adding segregated free lists based on type sizes

Adding Char/String Syntax

Add some syntax sugar to allow using ‘letter’ format and for string object creation with “words” format, with accompanying utility functions

Adding Constructors

It's so much more convenient than manually calling a construction method (with params) after instantiating an object

Questions?