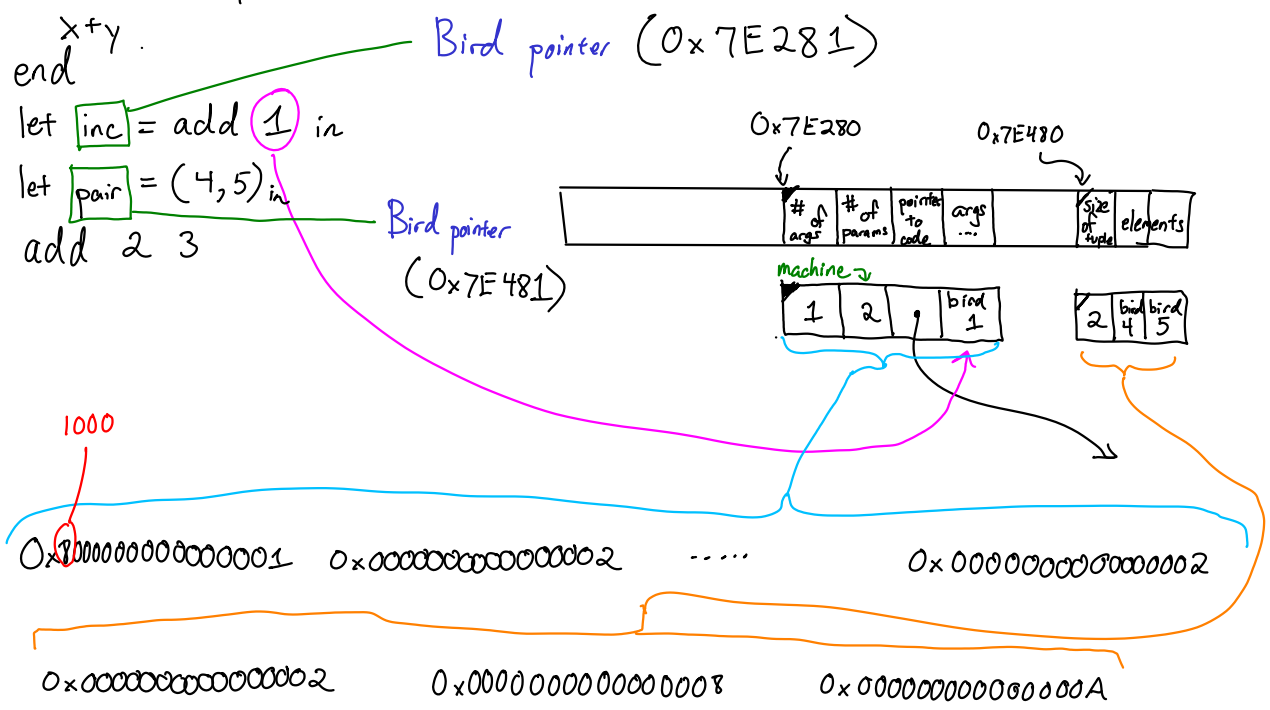
Falcon

- First-class functions — functions are values
- Partial application — functions can be called with fewer arguments producing a function waiting for more
- Anonymous function — functions don't need names (fun s -> s)

Finch

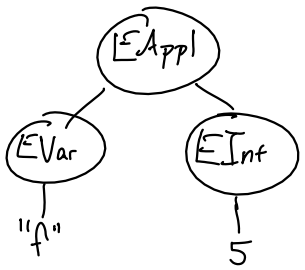
```
let add x y =
  x + y
;;
let inc = add 1;;
```

```
def add x y =
  x+y
end
let inc = add 1 in
let pair = (4,5) in
add 2 3
```



Application

f 5

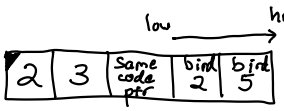



```
def foo x y z =
  x + (y * z)
end
let f = foo 2 in
  ⋮
let g = f 5 in
let g' = f 6 in
  ⋮
let n1 = g 7 in
```

37

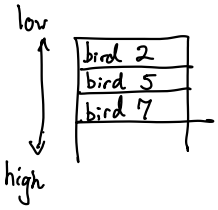
Assume f contains a bird ptr to

let g = f 5 in ⟹



g 7 ⟹ 37

Call foo with 2, 5, 7 just like Dave



Algorithm for Application

e_1 e_2

- Evaluate both
- Check first expr is a closure
- Is # params of fn > # args in closure + 1? If so
  - Copy closure to a new heap location
  - Increase # args in copy
  - Append new arg to end
  - Produce a pointer to copy
- Otherwise

GO TIME

Have: pointer to a closure with all but last arg, also have last arg

Want: to call fn w/ all args & give result

Option 1:

push last arg

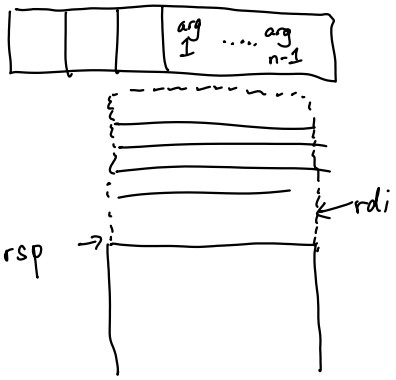
loop from right to left in closure, pushing each arg

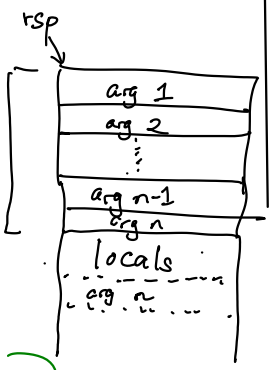
call

pop args

must be assembly because we don't know at compile time how many args there are

Option 2:





1. Allocate stack memory
   sub rsp, ... 8·n
2. Copy all args from closure
   rep movsq
3. Copy last arg
   mov [rsp + ...], ...

rep movsq — copies memory

put into rdi the destination (machine ptr)
put into rsi the source (machine ptr)
put into rcx the # of 8-byte words (machine int)
rep movsq ← modifies all three registers

f 5 7