* First-class functions
* Partial applications
* Anonymous Functions

} Falcon } Finch

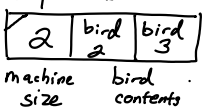
Falcon

Falcon values

true
5
(2, 3)

Machine representation

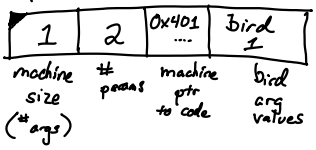
0xFF ··· FF
0x00 ··· 0A
0xPP ··· P[p001] → 0xPP···P[p000]



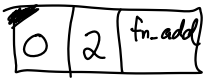
* `def add x y = x+y end`

add 1

0xPP···P[p001] → 0xPP···P[p000]



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



(3, 2, 9)

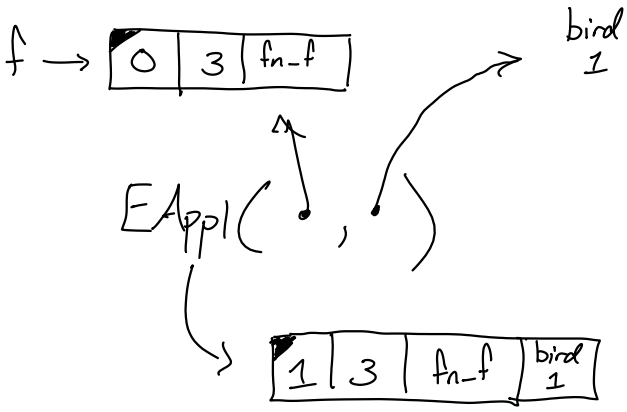
```
def inc y =
  1 + y
end
```

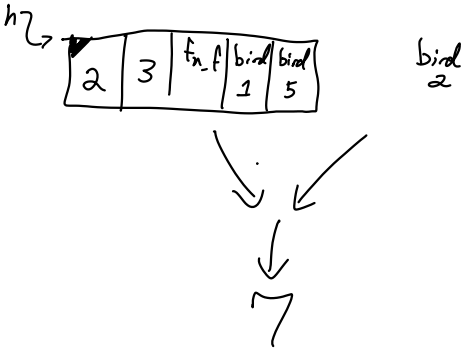
```
def dec y =
  -1 + y
end
```

good concept
bad implementation

Goal: give algorithm for function application (EAppl)

```
def f x y z =
  x * y + z
end
let g = f 1 in
let h = g 5 in
h 2
```





Falcon function application

If #args in closure = #params for func - 1 Then

IT'S GO TIME

Put args (including new one) on stack

Call fn

Tear down args

Else

Copy closure into a new heap location

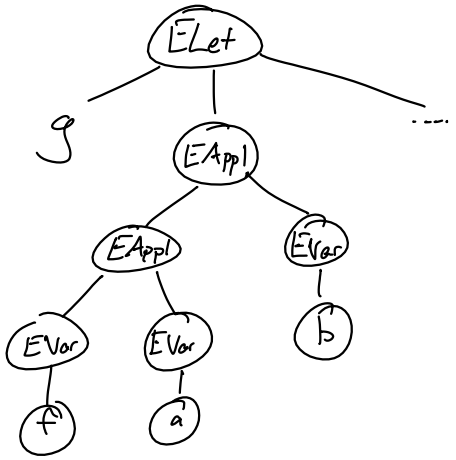
In copy:

increment args

copy application arg onto end of closure

Return ptr to copy

```
let g = (f a) b in
...
```

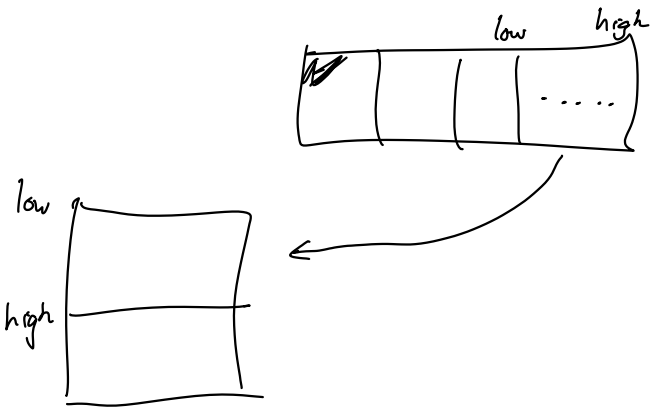


Given a call site (EAppl), you don't know at compile time how many parameters there are.

```
def inc x =
    x + 1
end
def mult x y =
    x * y
end
let n = .... in
let f = if n = 0 then mult 2 else inc in
f 3
```

rep movsq

rsi — pointer to source memory
rdi — pointer to destination memory
rcx — count of words to copy



Errors in

```
def f(x)
...
end
def f(y)
...
end
```

```
f(2,3,4)
badfn(5)
```

Dove		Falcon
Duplicate fn defns	→	still an error
incorrect # of args	→	ignore / runtime error
undeclared fn	→	ignore (unbound variable)
unbound variable	→	still an error (update)
duplicate param defn	→	still an error

```
def f x =
end
def f y =
end
```

```
f 2 3 4
badfn 5
```

```
def g x y =
  x+y
end
def f z =
  g
end
```