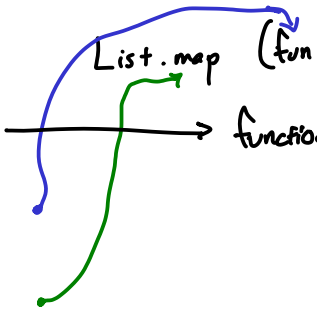
Functional Language ← I don't like this term.

- Partial application : List.map (fun x → x + 1) ⇒ a fn which adds 1 to everything in the list you pass it
- First-class functions : → functions are values
- Anonymous functions :
- Higher-order functions :



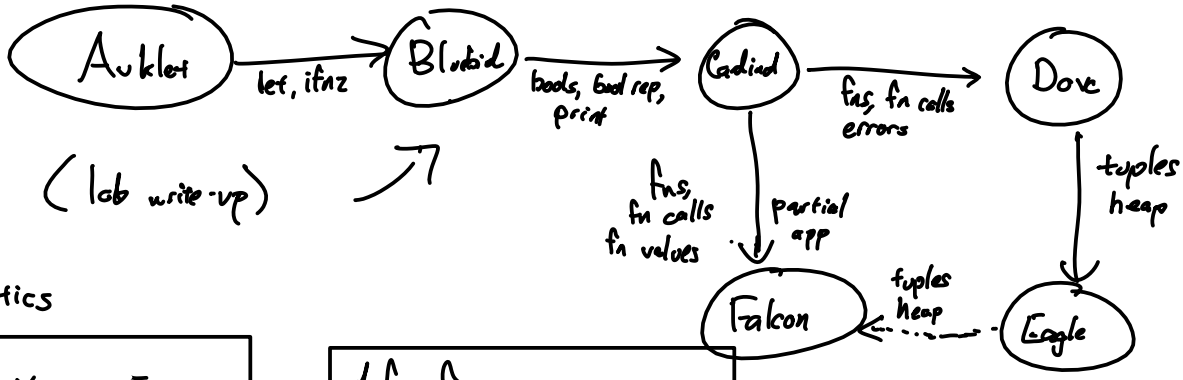
List of nums : [1; 2; 4]

List of fns : [(fun x → x+1); (fun x → x*2)] : (int → int) list

	OCaml	C	Python	JS	Java
Partial Application	Yes	No	No	No	No
First-class functions	Yes	No (not really)	Yes	Yes	8?
Anonymous functions	Yes	No	Yes	Yes	8?

λ ← lambda

Falcon



1. Syntax (lab write-up)

2. Semantics

```
def f x y =
    x - y
end
f 6 4
```

```
def f x y =
    x + y
end
let inc = f 1 in
inc 3
```

```
def twice f x =
  f (f x)
end
def inc x =
  x + 1
end
twice inc 2
```

3. Compilation

1. Which fn?
2. How many args we have?
3. How many args we need?
4. Which args?

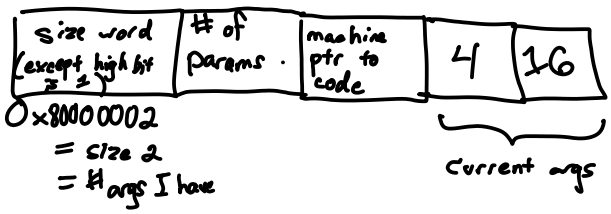
New heap entry: "closure"

Any fn value which is waiting for args will at runtime be represented by a closure.

Tuples

Size
n + 1

Closures



n + 3

```
let x = f in
```

What about errors?

Dove

wrong # args

call non-existent

unbound vars

dup defn fn

dup defn params

Falcon

call non-function (runtime)

unbound var

unbound var

dup defn fn

dup defn params

```
def f x =
  if x > 10 then 0 else f x
end
def f x =
  f (x+1)
end
f 2
```