EAppl(name, args)

$\boxed{f(2, inc(3))}$

EAppl("f", [EInt 2, EUnaryOp(OpInc, EInt 3)])

```
let names = List.map (fun e -> freshname "$")
                args in
```

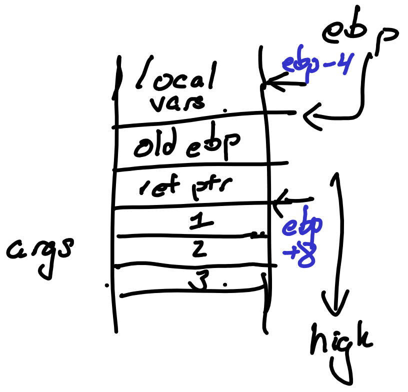
```
List.combine : 'a list -> 'b list ->
                   ('a * 'b) list
```

```
let $1 = 2 in
let $3 =
        let $2 = 3 in
         inc($2)
in
 f($1, $3)
```

# Diamondback

def f(params...) body end

| AFunction(name, params, body) →
  let flabel = XLabel("_" ^ name) in



modify/create environment to refer to parameters

compile body w/ new environment

x, y, z

{ x: +8
  y: +12
  z: +16 }

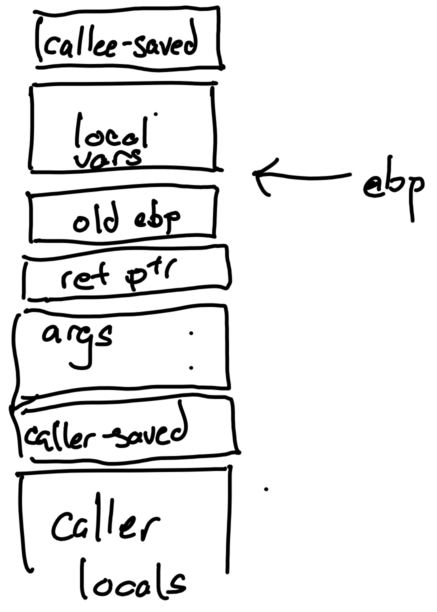
callee [ Save ebp, move ebp
         Count vars, reserve stack space

body [

callee [ free up stack
         restore ebp, esp
         ret

type environment =
    int StringMap.t *
    int * int

call:
- push next instruction addr onto stack
- jump



```
⋮
call printValue
add esp, 4
mov eax, [ebp-4]
```

| Caller-saved | Callee-saved |
|---|---|
| are not safe | are to be restored |
| eax | ebx |
| ecx | esi |
| edx | edi |

```
push edx
push eax  ← arg
call f
add esp, 4
pop edx
```

# Diamondback

- refer to unbound variable
- use a function as a parameter/value
- call a function that doesn't exist?
- arity mismatch: wrong number of args?
- 2 decls w/ same fn name?
- 2 params w/ same name?

```
def f(x, x, x, y; z, z, x)
  . . .
end
```

x + y

if true then 2 else 4