```
let rec sum n =
  if n=1 then 1 else
    n + sum (n-1)
;;
let z = sum 4;;
```

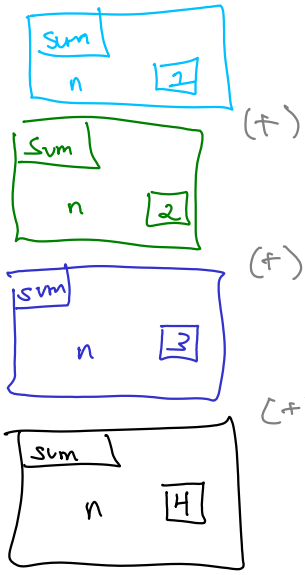
```
let z = if 4 = 1 then 1 else
          4 + sum (4-1) ;;
```

```
let z = 4 + sum (4-1) ;;
```

```
let z = 4 + sum 3;;
```

```
let z = 4 + if 3=1 then 1 else
            3 + sum (3-1)
        ;;
```

```
let z = 4 + (3 + (2 + (1)))
```



Which of these don't I need?

I need all of this!

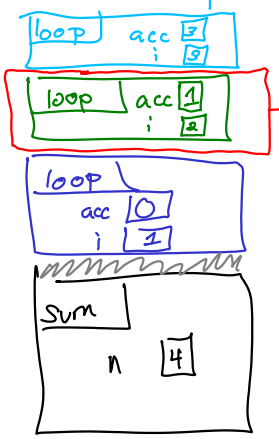
```
let sum n =
  let rec loop acc i =
      if i=n then
        acc + i
      else
        loop (acc+i) (i+1)
  in
  loop 0 1
;;
let z = sum 4;;
```

```
let z = let rec loop acc i =
            if i=4 then
              acc + i
            else
              loop (acc+i) (i+1)
        in
        loop 0 1
```

```
let z = let rec loop acc i =
            if i=4 then
              acc + i
            else
              loop (acc+i) (i+1)
        in if 1=4 then
             0 + 1
           else
             loop (0+1) (1+1)
```

```
let z = let rec loop acc i =
            if i=4 then
              acc + i
            else
              loop (acc+i) (i+1)
        in loop 1 2
```

```
let z = let rec loop acc i =
            if i=4 then
              acc + i
            else
              loop (acc+i) (i+1)
        in loop 3 3
```



do I need this?

No

- args (in boxes) — args + params : data
- base ptr
- return ptr — work left

g 0

1 + g 0

g 0 + 1

(f 0, g 0)

if f 0 then g 0 else h 0

let x = f 0 in g 0

(f 0) (g 0)

- Yes: we can discard caller's stack frame.
- No: we still need caller's stack frame.

For each of these calls: can I throw away the caller's stack frame when I make that call?

```
type expr =
  | EInt of int
  | EBool of bool
  | ETuple of expr list
  | EBinaryOp of binop * expr * expr
  | EUnaryOp of unop * expr
  | ELet of string * expr * expr
  | EAppl of expr * expr
  | EIf of expr * expr * expr
```

```
if i=n then
  acc + i
else
  loop (acc+i) (i+1)
```