Schedule for rest of term

Tail call optimization

↳ Last assignment (part 9) due last of classes, no late tokes can be used (college policy)
[also, no resubmissions]

Exam days

W last day of class

Th } reading days
F }

Sa
Su } exam days
M

2.5 hour slot

→ 8:30am - 11:30am

↳ our scheduled Celebration time (exam)

- CoK #4 (60-70 mins)
- retake options

↳ either during our exam period on Monday
- or self-scheduled

If you do it on Mon morning with us, bring your notecard w/ you

If you do self-scheduled, it's due end of day Tuesday week 10.

Late arrival to class

Tail call optimization
(making interpreter faster/use
less memory)

- you're not implementing this, but
you could

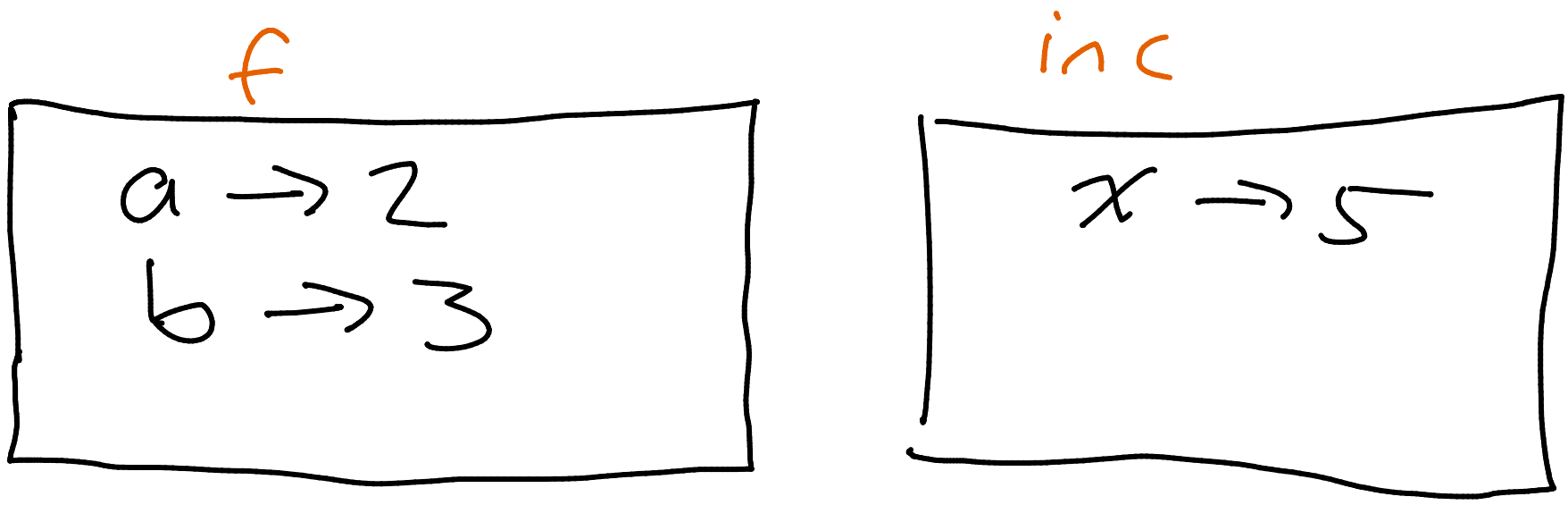
```
(define inc
  (lambda (x)
    (+ x 1)))

(define f
  (lambda (a b)
    (inc (+ a b))))

(f 2 3)
```

6

6

The reason this works is
because inside f, the
call to inc is the
last thing that happens
"tail call"

(f 2 3)



Once f called inc, we didn't need the frame that f created any more.

All f did, after calling inc, was just relay the answer back.

In principle, we could have had f "go away" after calling inc, — and have inc just give the answer to me directly.

Some language implementations optimize tail calls. Some don't.

In the case of recursion, it can matter a lot.

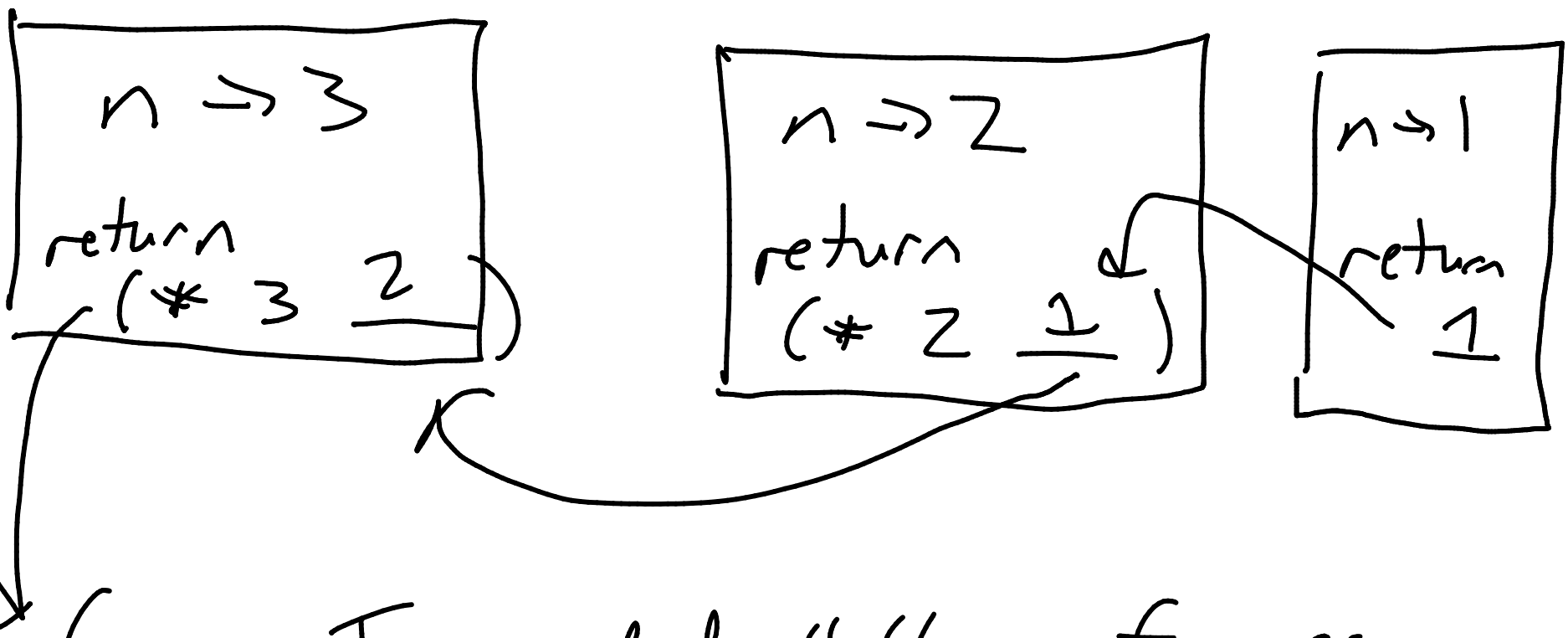
Classic recursion example

```
(define fact
  (lambda (n)
    (if (equal? n 1)
        1
        (* n (fact (- n 1))))))
```

last thing that happens, which is not the recursive call, so not optimizable.

This function is not tail recursive.

(fact 3)

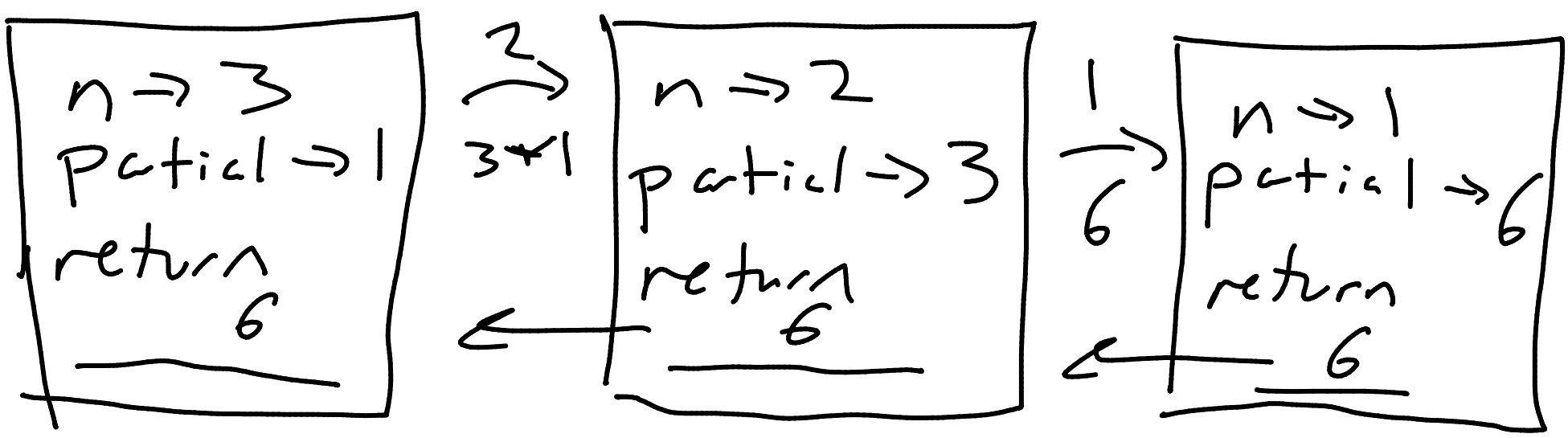


6 I needed all those frames.

What if I wrote factorial differently?

```
(define fact
  (lambda (n partial)
    (if (equal? n 1)
        partial
        (fact (- n 1)
              (* n partial)))))
```

(fact 3 1) always call w/ a 1



we could have eliminated each frame on the way down as we were done with it