Lecture 3: More Procedures

Review

- What does lambda do?
- Names/tables

Details

- (define m 2)
  ((lambda (m) (* m m)) 5)

- What happens?
- Why?

Remainder

- Write the function (`remainder x y`) that computes the remainder of x/y
- `(remainder 5 2)` should evaluate to 2
- Plan:
  - Base Case?
  - Recursive Case?

Practice Problem

- Using string-append, write a procedure pad, which takes a string and a number, that returns the string with that number of spaces added to the end.

```
(pad "yay" 0)
;Value: "yay"
(pad "yay" 1)
;Value: "yay "
(pad "yay" 3)
;Value: "yay "
(define pad
```

Practice Problem

- Write a procedure that uses Euclid's algorithm to compute the GCD of two numbers. Euclid's algorithm (according to Knuth it's the oldest known algorithm) goes as follows: if r is the remainder of a divided by b, then the common divisors of a and b are the same as those of b and r. Additionally, the gcd of a number and 0 is the number.

```
(gcd 206 40)
;Value: 2
(define gcd
(lambda (a b)
```

Practice Problem – Guessing Game

- The classic, “I’m thinking of a number between 1 and 100”
- Here’s how the game goes:
  - You think of a number
  - I make guess
  - You tell me
    - Too low
    - Too high
    - Just right
  - I guess again

Algorithm

- Let’s play

Now let's do it in Scheme

- Let's make a plan:
  - What's the base case?
  - What are the recursive cases?

Coding it up

- First, let's write the code to make a new guess:

```
(define (make_new_guess low high)
….)
```

My answer

```
(define (make_new_guess low high)
  (quotient (+ low high) 2))
```

Coding it up

- Here's a start

```
(define (find-answer guess low high)
    ……)
```

- find_answer should return the correct answer when it finds it.

Basic Steps

- Here are the new steps
  - 1. Get a response
  - 2. Using the response, check for the right answer
  - 3. If it’s too low, modify guess

We'll need a new special form – **let**

- (**let** ((*name1 expr1) (name2 expr2))
  expr)
- This *binds* the value of *expr1* to *name1* when evaluating *expr*
  - But nowhere else!

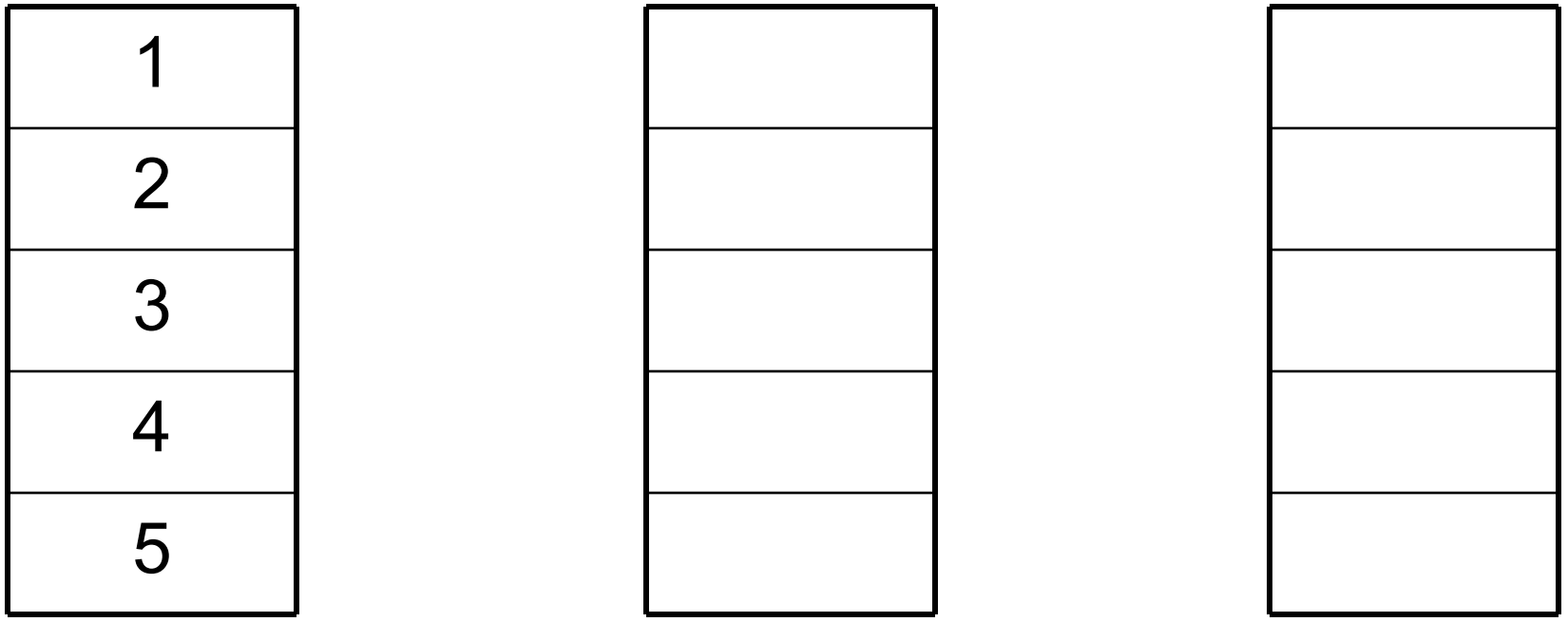
Let's try it

- Type
- (let ((a 1)
  (b 2))
  (+ a b))

- Now add (+ a b) at the bottom

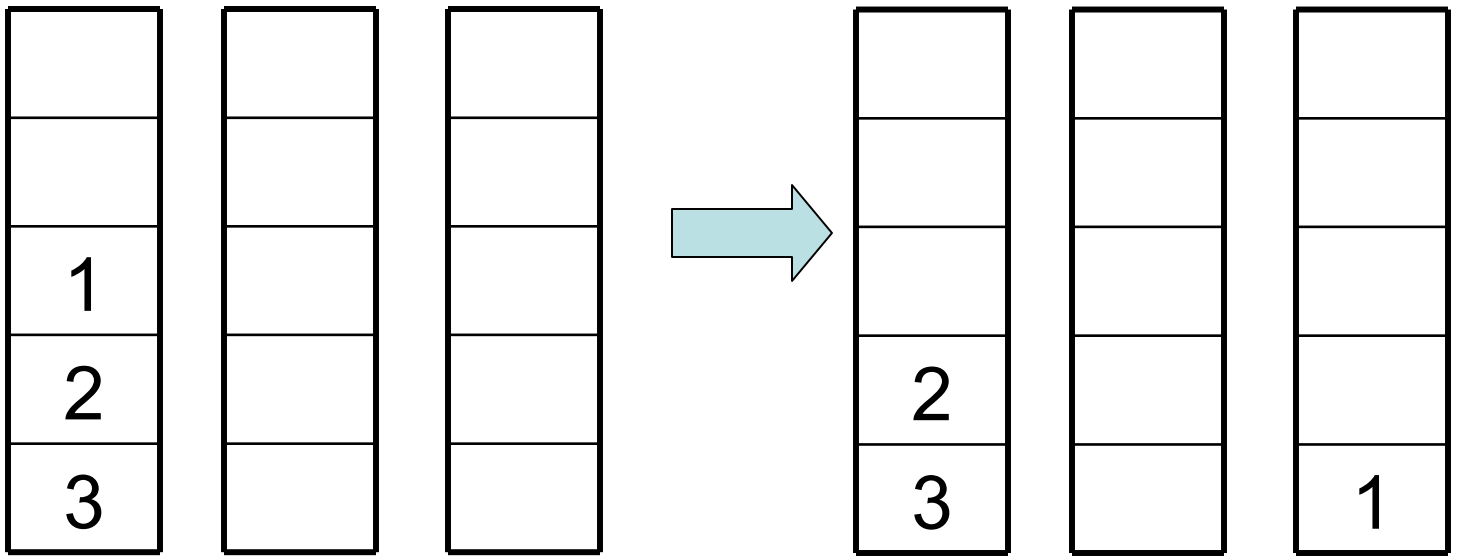
- Please load lec3-guess.scm from the website

Towers of Hanoi (My version)

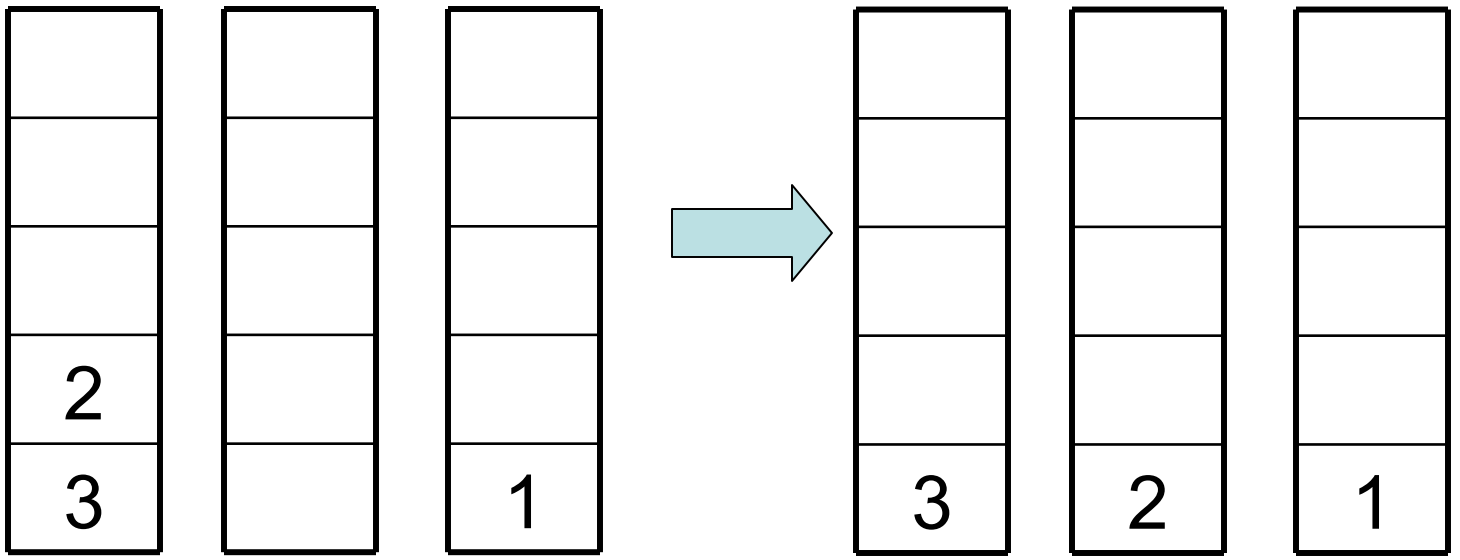


We want to move all of the numbers from one stack to the other
Numbers must always be in increasing order!!!!

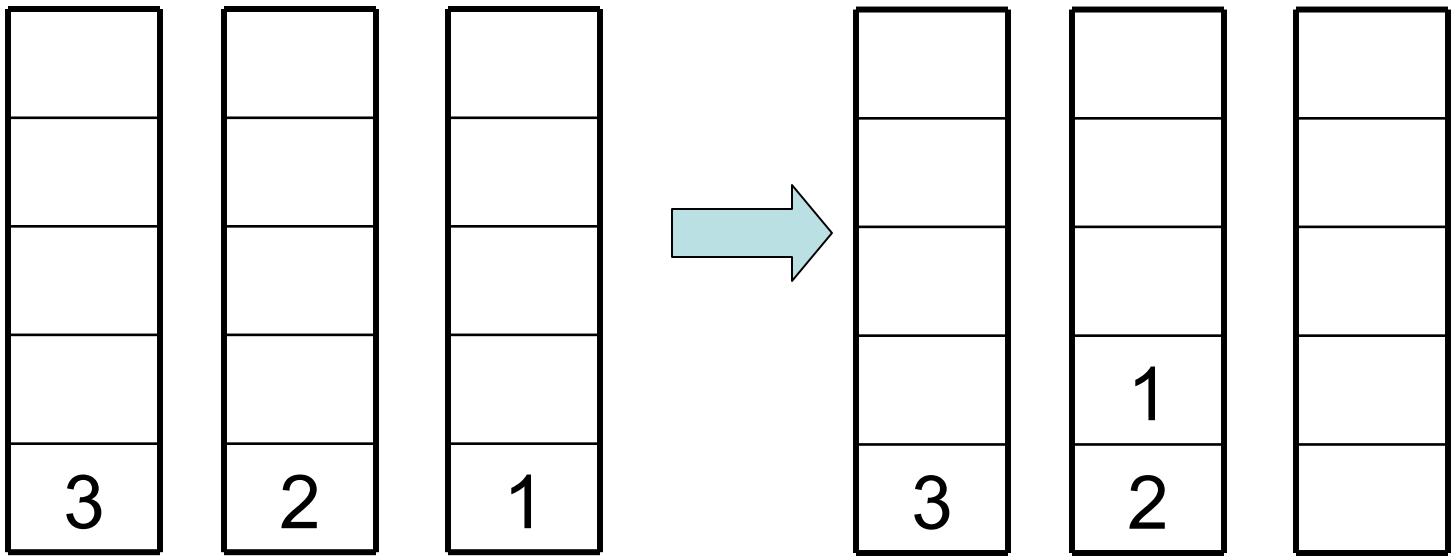
Easy Example



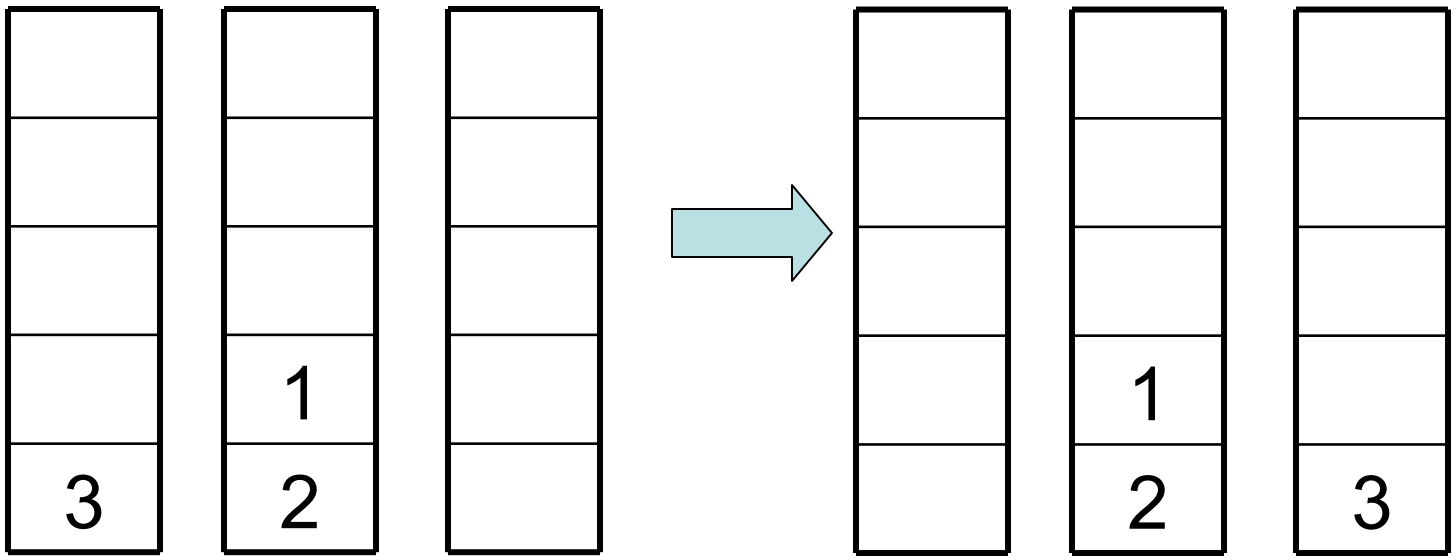
Easy Example



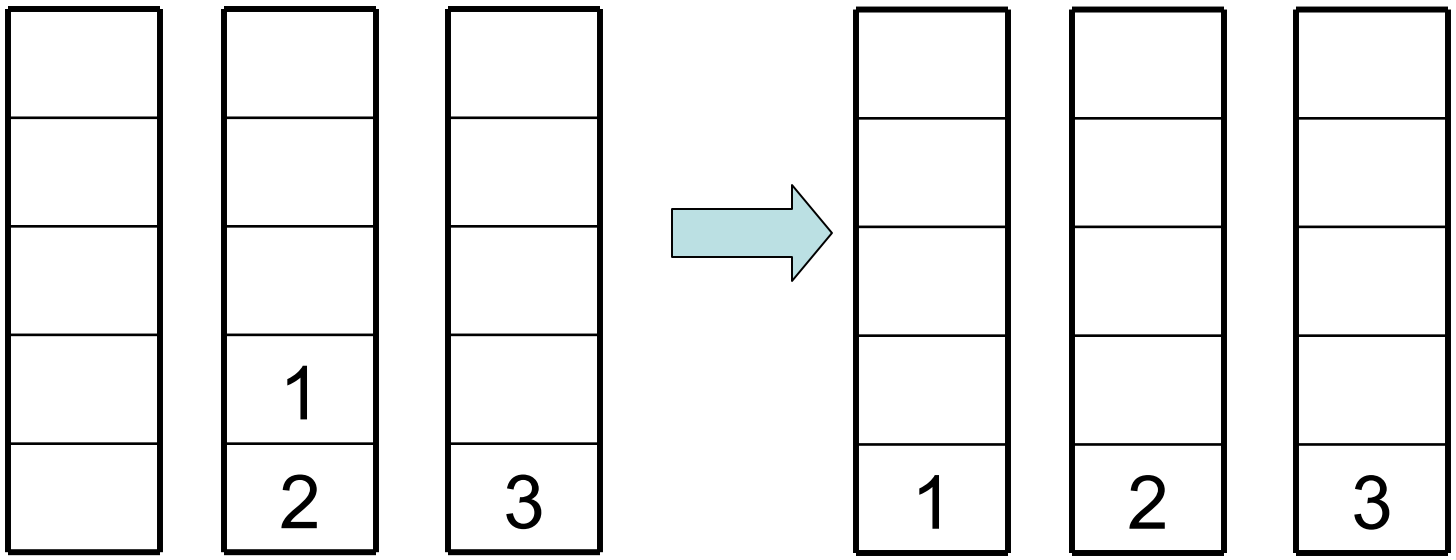
Easy Example



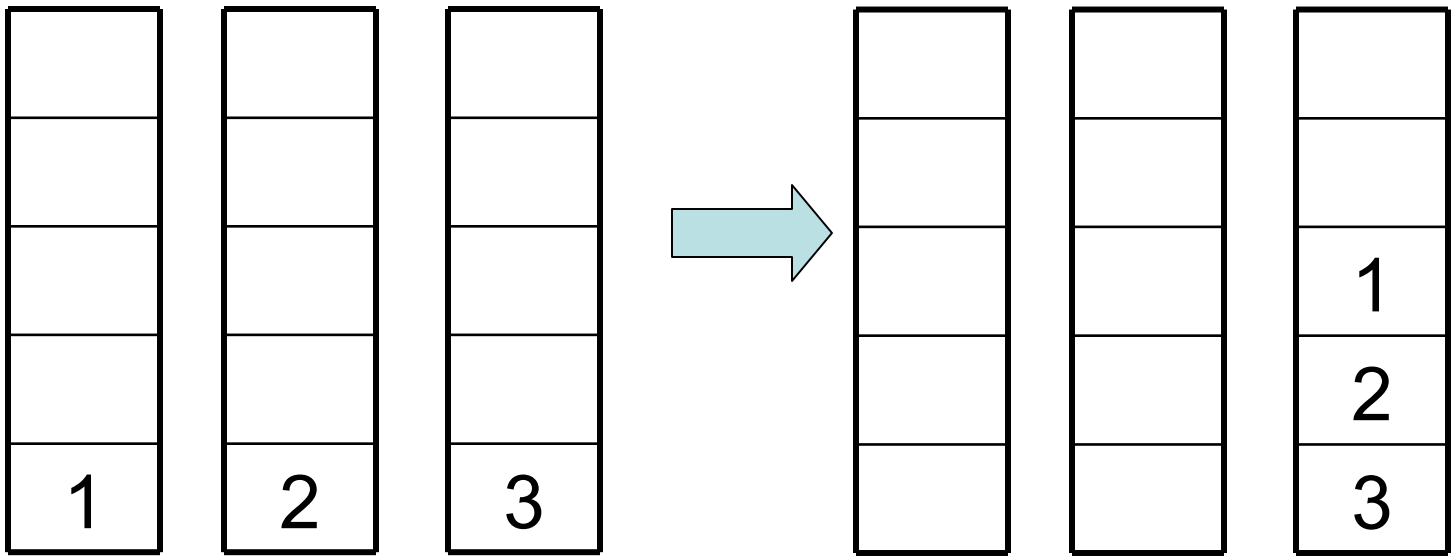
Easy Example



Easy Example



Easy Example



Now you try

Let’s solve this in Scheme

- What’s the base case?

Let's solve this in Scheme

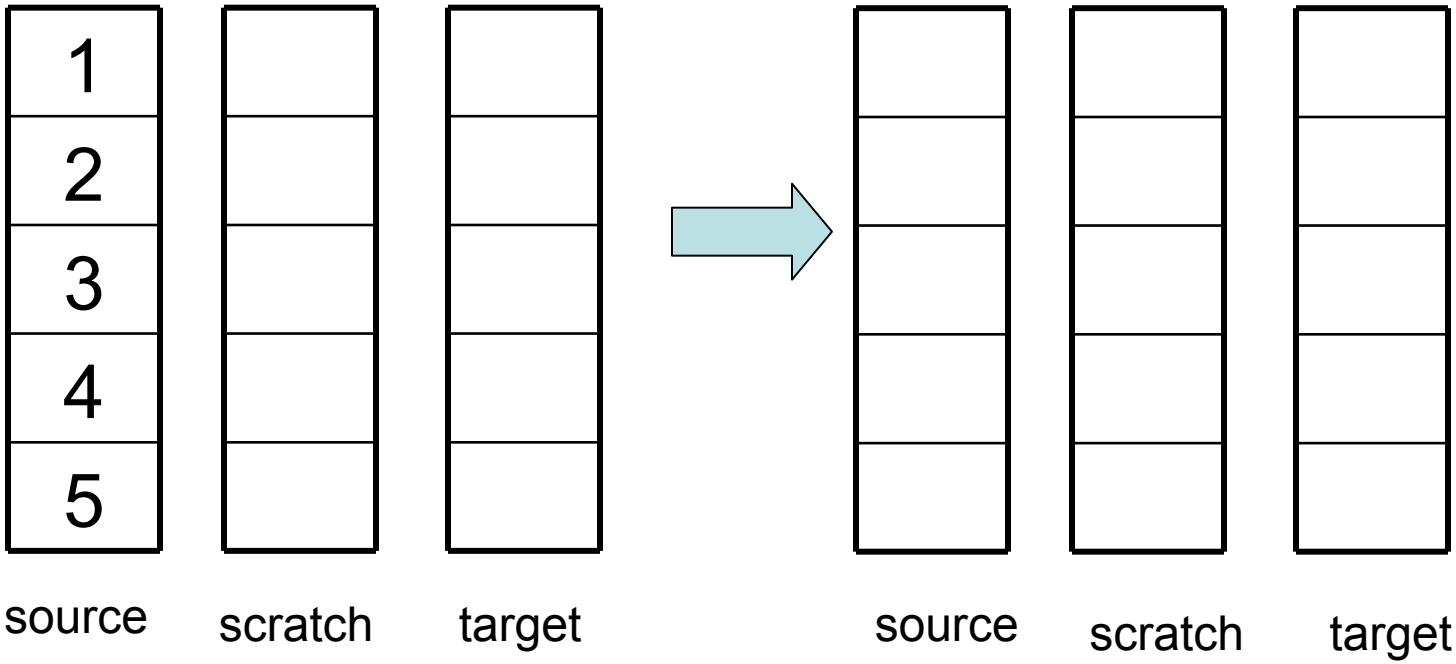
- What's the base case?
  - Move the first disc, (just move it)

- What are the operations that we'll use for the recursive case?

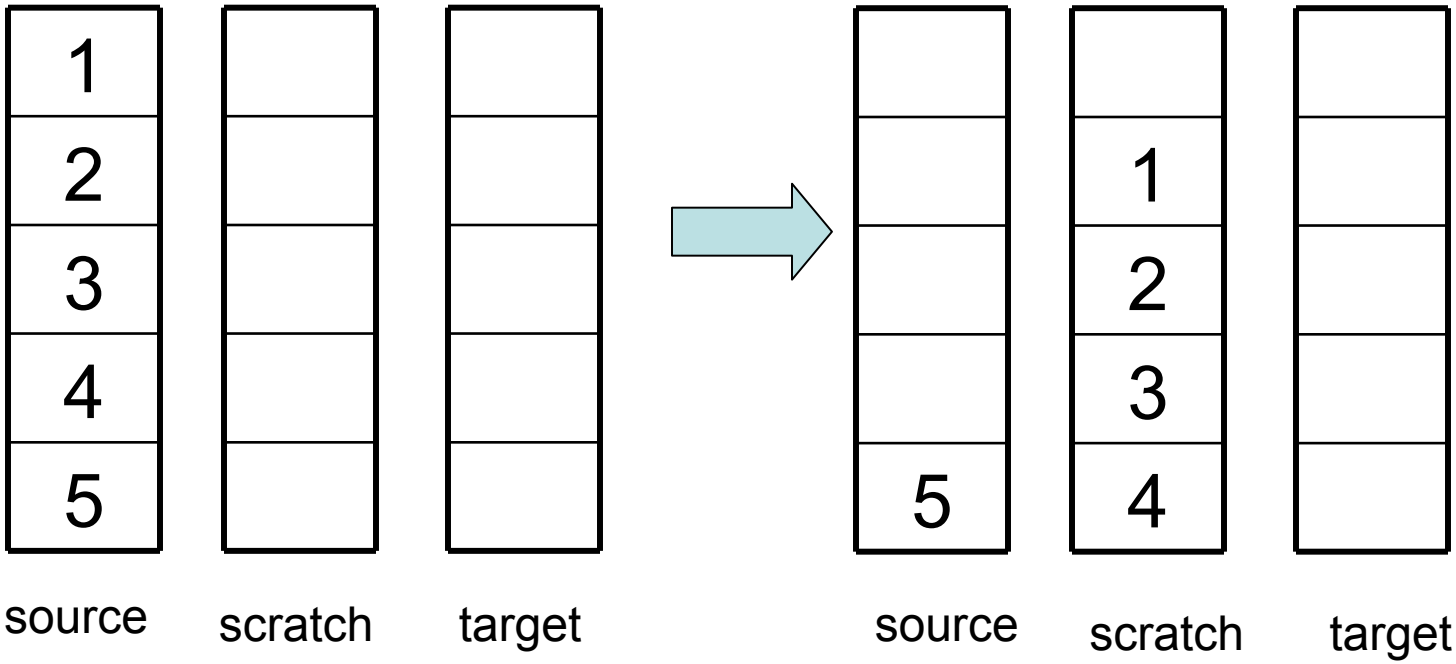
Let's solve this in Scheme

- What's the base case?
  - Move the first disc, (just move it)

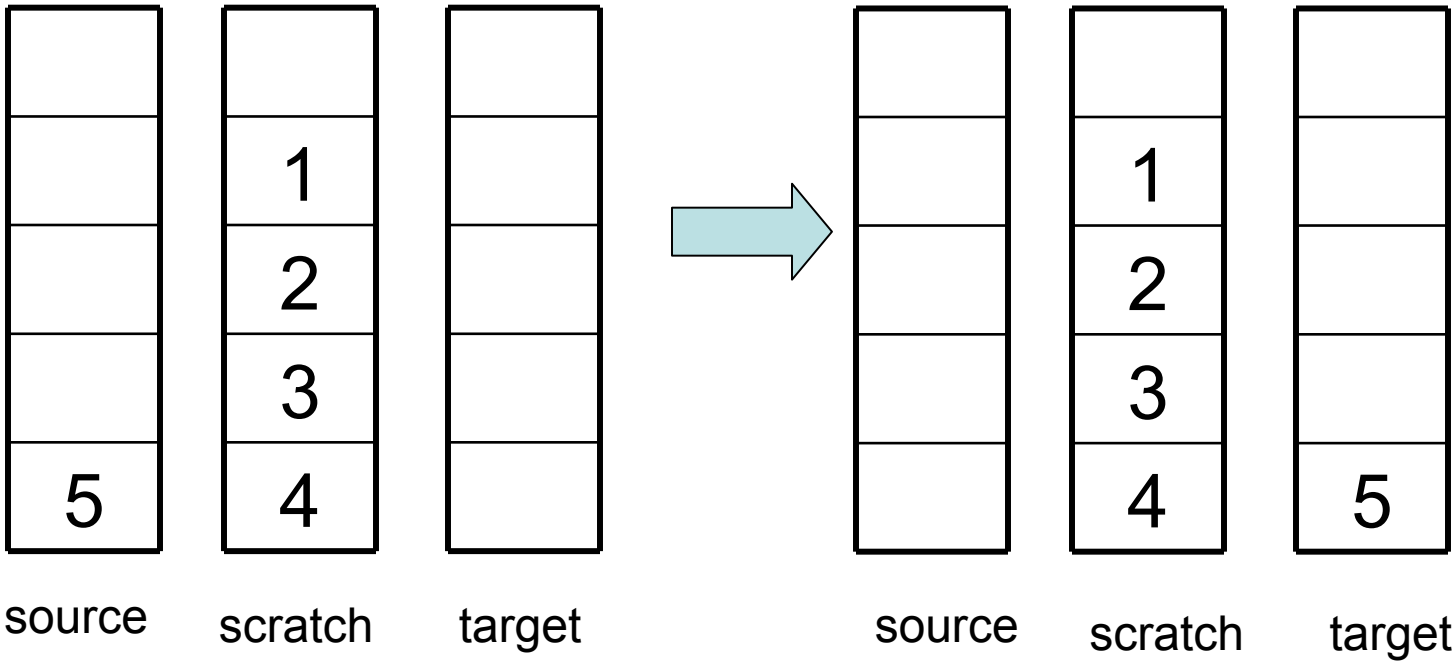
- What are the operations that we'll use for the recursive case?
  - 1. Assign a source, target, and scratch column



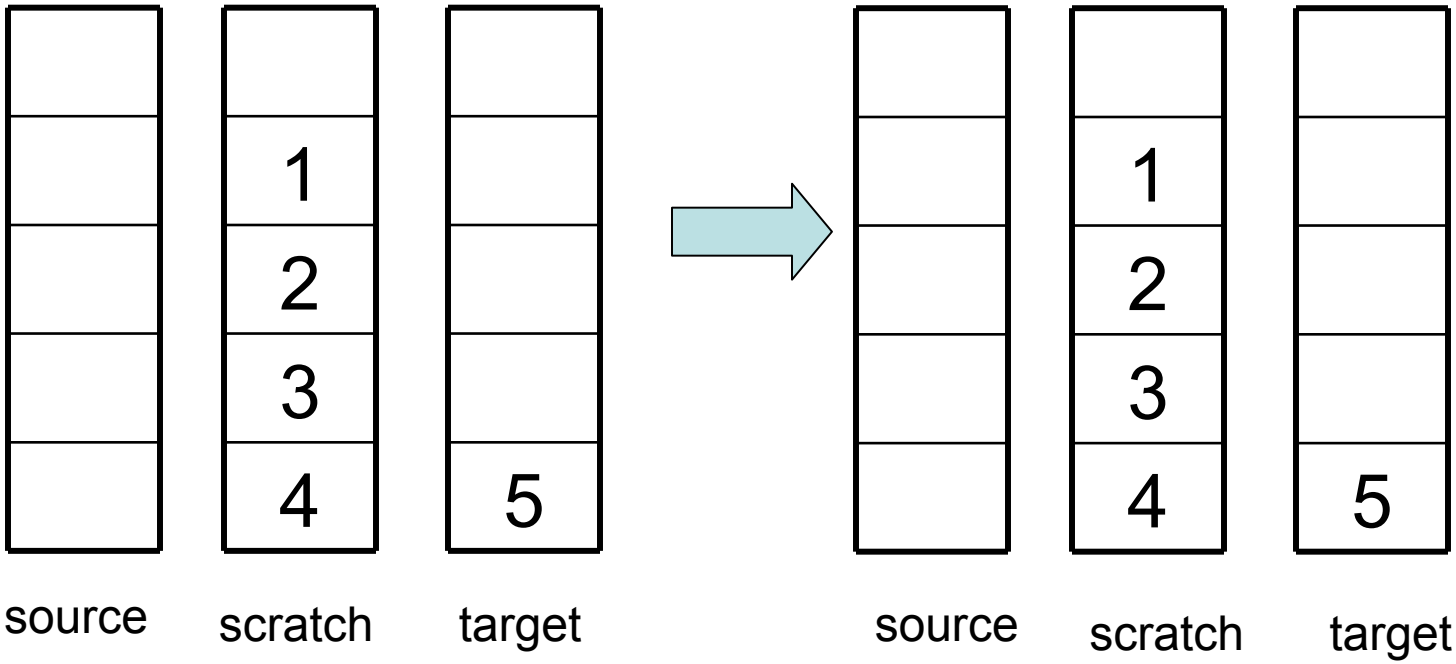
1. Assign a source, target, and scratch column



2. Move n-1 discs to scratch



2. Move nth disc to target



Now what?

Expressing this as Scheme

begin

- Sometimes need to be able to do several steps in order.
- Usually, this is to get “side-effects”, like printing something out
- (**begin** *state1*

  state2

  state3)

begin

- (**begin** *state1*
  state2
  state3)
- The value of this expression is the value of the last expression in the block

Examples:

```
(begin
  3
  4
  5)
;Value: 5
```

begin

- (**begin** *state1*
  state2
  state3)
- The value of this expression is the value of the last expression in the block

Examples:

```
(begin
        (display “yay”)
                5)
   yay
;Value: 5
```