Lecture 5: Recursion Part 2

Kaleel Mahmood

Department of Computer Science and Engineering

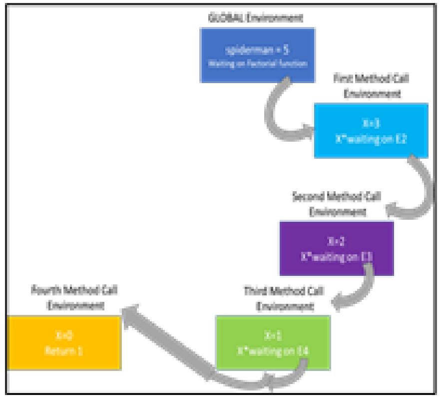
University of Connecticut

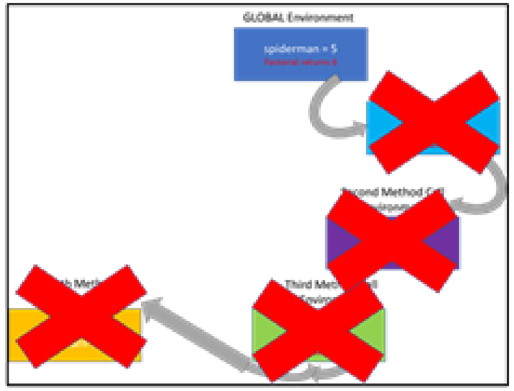
Last time on CSE 1729...

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

Traced through the recursion using pictures...

```
(define (factorial x)
  (if (= x 0)
      1
      (* x (factorial (- x 1))))
  )
```





ANOTHER EXAMPLE: THE FIBONACCI NUMBERS

- The *Fibonacci numbers* are defined by the rule:

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases}$$

- Note, then, that the sequence F_0, F_1, F_2, ... is

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

each is the sum of the previous two.

THE FIBONACCI NUMBERS IN SCHEME

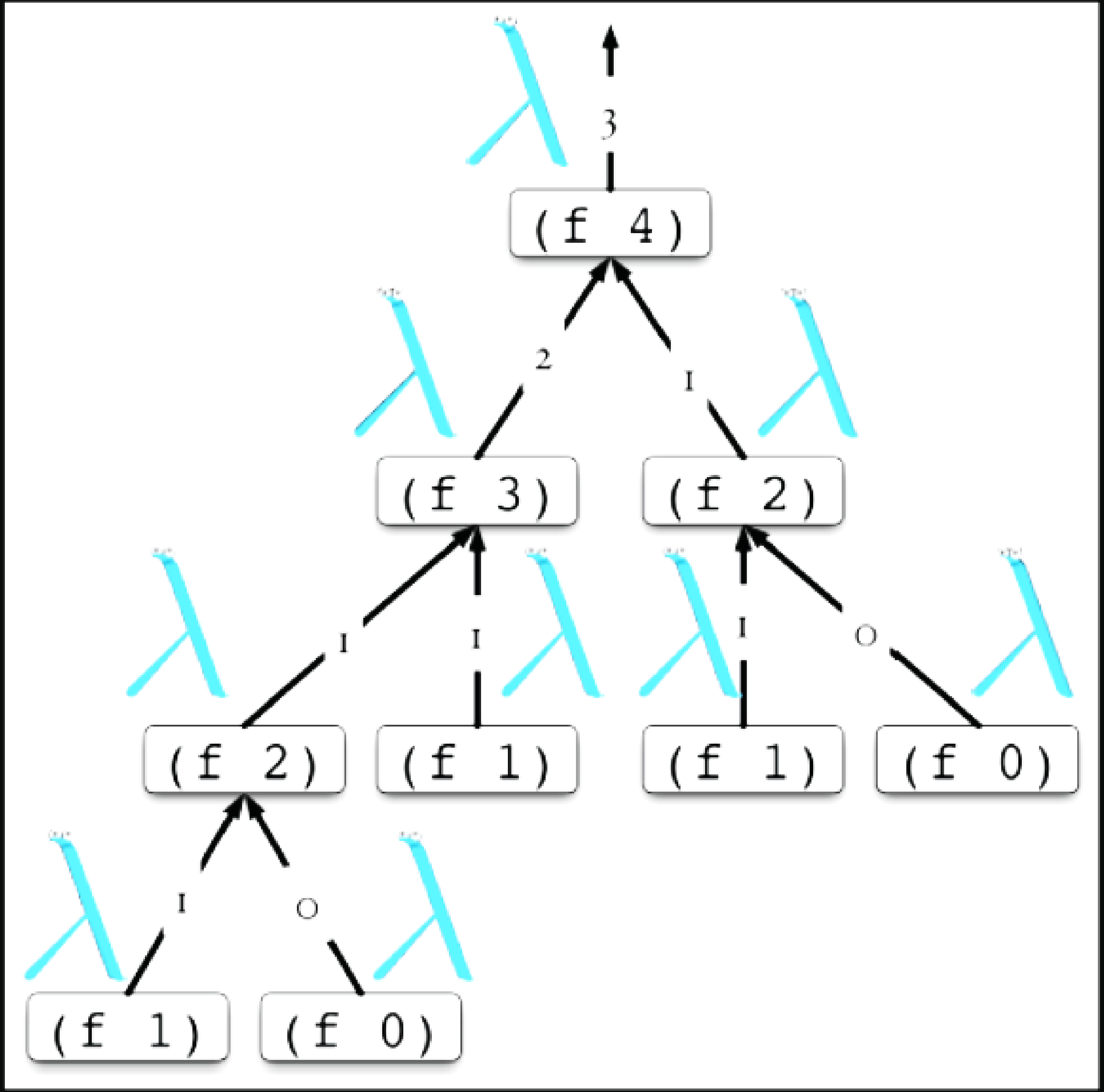
- As with the factorial function, we can naturally capture this definition in Scheme.

```
(define (fib n)
        (cond ((= n 0) 0)
              ((= n 1) 1)
              ((> n 1) (+ (fib (- n 1))
                          (fib (- n 2))))
        )
)
```

- Notice, as with factorial, how closely the Scheme definition can mirror the mathematical definition.

THE FIBONACCI EVALUATION TREE

- The Fibonacci function gives rise to an "evaluation tree" as shown. Here each node returns the sum of the value of its children.
- Note that some "sub"-problems are evaluated many times.
- Question: How many times is `(f 1)` evaluated, in total?



Be careful with recursion!

Spiderman’s reaction when you forget to include a base case

```
1 > (define (recurse x) (recurse x))
2 > (recurse 1)
```

- Recursion is a lot like nuclear energy. It is powerful but can lead to serious problems if not handled correctly.

“ITERATIVE” CONSTRUCTS IN SCHEME

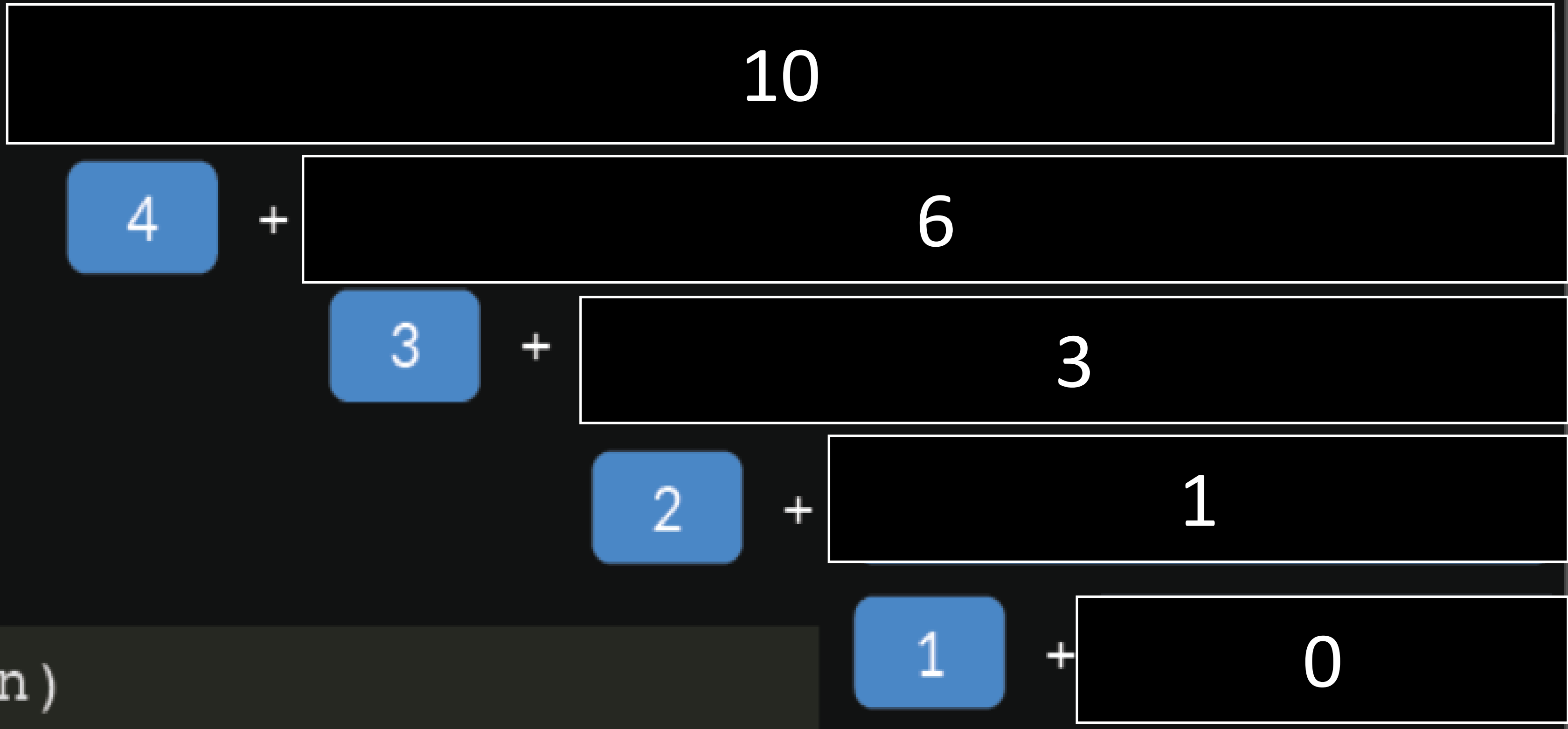
- Consider computing the sum of the first $n+1$ natural numbers in Scheme.
- Note that

$$\underbrace{(0+1+\cdots+n)}_{\sum_{i=0}^{n} i} = n + \underbrace{(0+1+\cdots+(n-1))}_{\sum_{i=0}^{n-1} i}$$

- And thus:

```
> (define (number-sum n)
    (if (= n 0)
        0
        (+ n (number-sum (- n 1)))))
> (number-sum 10)
55
```

THE EVALUATION TREE FOR `number-sum`




```
> (define (number-sum n)
    (if (= n 0)
        0
        (+ n (number-sum (- n 1)))))
> (number-sum 10)
55
```

How to write and debug recursion quickly?

The story of the space pen…

- This story has since been shown to be a myth but it illustrates an important point.
- In the 1950s the Americans and Soviet were locked in a space race.
- Normal pens need gravity to operate. So <u>no</u> normal pens in space.
- What to do?

1. Spend millions developing a space pen (American solution)

2. OR…use a pencil (Soviet Solution)

EXAMPLE: MULTIPLICATION IN TERMS OF ADDITION

- Consider the definition of multiplication as repeated addition:

$$a \times b = \underbrace{b + b + \cdots + b}_{a \text{ times}}$$

- We can express this in Scheme:

```
(define (mult a b)
    (if (= a 0)
        0
        (+ b (mult (- a 1) b))))
```

EFFICIENCY CONSIDERATIONS

- How many recursive calls are generated by

`(mult 200 2)` ← `(mult 199 2)` ← `(mult 198 2)` ←

- How about

`(mult 2 200)` ← `(mult 1 200)` ← `(mult 0 200)`

- We could write a more *efficient* program by "recursing on the smaller of a and b." Thus

A MORE EFFICIENT MULTIPLY...

- We could write a new program to exploit this...

```
(define (fmult a b)
    (cond ((= a 0) 0)
          ((= b 0) 0)
          ((<= a b) (+ b (mult (- a 1) b)))
          ((>  a b) (+ a (mult a (- b 1))))))
```

- Now it will only recurse min(a,b) times. Alternatively,

```
(define (fmult a b)
    (if (> a b) (mult b a) (mult a b)))
```

TO BE REALLY FANCY, WE COULD REDUCE BOTH A AND B AT THE SAME TIME...

- Remember that $ab = (a-1)(b-1) + a + b - 1$. Thus we could also express multiply as...

```
(define (fmult a b)
    (cond ((= a 0) 0)
          ((= b 0) 0)
          (else (+ -1
                   a
                   b
                   (fmult (- a 1) (- b 1))))))
```

- This will also recurse min(a,b) times.

ACTUALLY, ALL THREE OF THESE ALGORITHMS ARE TERRIBLE...WHY?

- With paper and pencil, how long would it take you to multiply two 16 digit numbers? Perhaps a few hours?
  With the program above, the computation

```
(fmult 1000000000000000 1000000000000000)
```

  - Well, 1000000000000000 will generate a call to
    - 999999999999999, and hence to
    - 999999999999998, and hence to
    - 999999999999997, and hence ....
  - In total 10000000000000000 calls must be completed. If the computer could carry out a million calls per second, this would take 1000000000 seconds, a little over 30 years.

WE CAN FIX THIS BY USING MORE INFORMATION ABOUT MULTIPLICATION...

- On a computer dividing by 2 and multiplying by 2 can be done very quickly--we can improve our program:
- **Observation**: Suppose we wish to multiply x and y.
  - If we're lucky, x is even, and we have

$$x \times y = 2 \times \boxed{[\tfrac{x}{2} \times y]}$$

These operations can be done quickly

x has been reduced by **half** in this recursive call

FAST MULTIPLICATION WITH DIVISION & MULTIPLICATION BY 2

- On a computer dividing by 2 and multiplying by 2 can be done very quickly--we can improve our program:
- **Idea**: To multiply x and y (positive whole numbers):
  - If x is odd, fix it! The answer is: $y + [(x-1) * y]$
  - Now, x-1 is even in the recursive call [...]
- If x is even: the answer is: $2 * [\frac{x}{2} * y]$

Recursive calls

Now, one of the numbers in the recursive call [...] has been significantly reduced--it's only half the previous size!

CAPTURING THIS IDEA IN A SCHEME PROGRAM

- On a computer dividing by 2 and multiplying by 2 can be done very quickly--we can improve our program:

```
(define (even x)  (= (modulo x 2) 0))
(define (twice x) (* x 2))
(define (half x)  (/ x 2))

(define (rfmult a b)
    (cond ((= 0 a) 0)
          ((= 0 b) 0)
          ((even a) (twice (rfmult (half a) b)))
          (else     (+ b (twice (rfmult (half (- a 1))
                                         b)))
    )
)
```

HOW HAS THE EVALUATION TREE CHANGED?

- Well, `(rfmult` 2^{k} `x)` will generate a call to
  - `(rfmult` 2^{k-1} `x)`, and hence to
  - `(rfmult` 2^{k-2} `x)`, and hence to
  - `(rfmult` 2^{k-3} `x)`, ...
- In total, if called on $10000000000000000 < 2^{54}$, only 54 calls must be completed. Your computer can do this in a fraction of a second. (Try it.)

How does this relate to Scheme?

“The space pen”

```
(define (even x)  (= (modulo x 2) 0))
(define (twice x) (* x 2))
(define (half x)  (/ x 2))

(def

)
```

“The pencil”

```
(define x 5)
```

Conclusion: A lot of these examples are exercises in DESIGN. They are created to make you think about programming in constrained or resource limited manners.

__However-__ The tips of pencils can break off, which is hazard to personal and equipment in space. Pencils are flammable, something NASA wanted to avoid after a fire on the Apollo 1. Looks like “just” using a pencil wasn’t actually viable.

Source: https://www.scientificamerican.com/article/fact-or-fiction-nasa-spen/

Let’s play a simple game…

- I am thinking of an integer number x between range a and b.
- Every time you guess a number y

  I will tell you two things:
  1. If $y == x$ (you win)
  2. If $y = ! \; x$ I will tell you either: y is bigger than my number or y is smaller than my number.

  Every time you guess wrong I will charge you $1. Fun game…for me!

  How to play this game effectively?

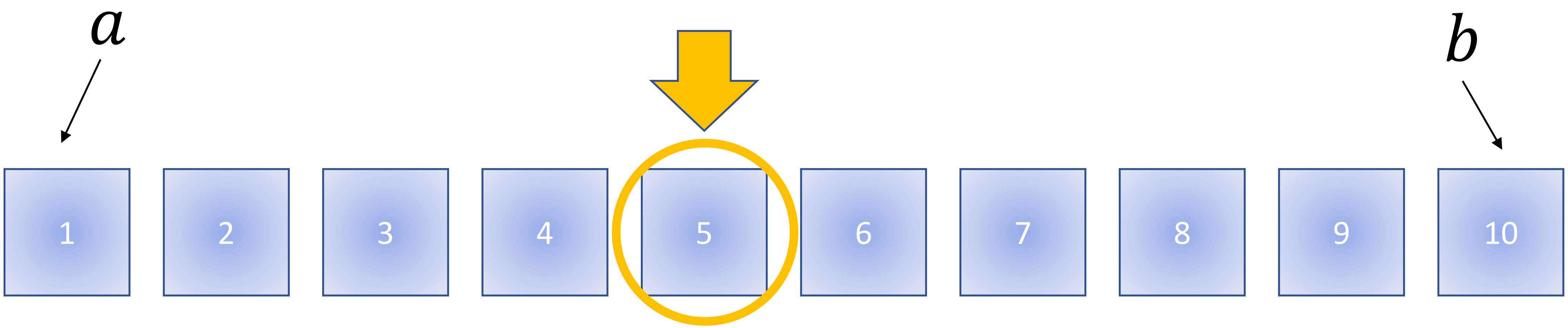
1st way to play (the stupid way)

- Let's say my number is between 1 and 10. $(a = 1,\ b = 10)$
- The stupid way: Start at 1 and go up. If my number is 10 you will have lost $9 playing my game.

2^{nd} way to play (the smart way)

- Let's say my number is between 1 and 10. $(a = 1,\ b = 10)$
- Use binary search.

First turn: Guess $y = (a + b)/2$

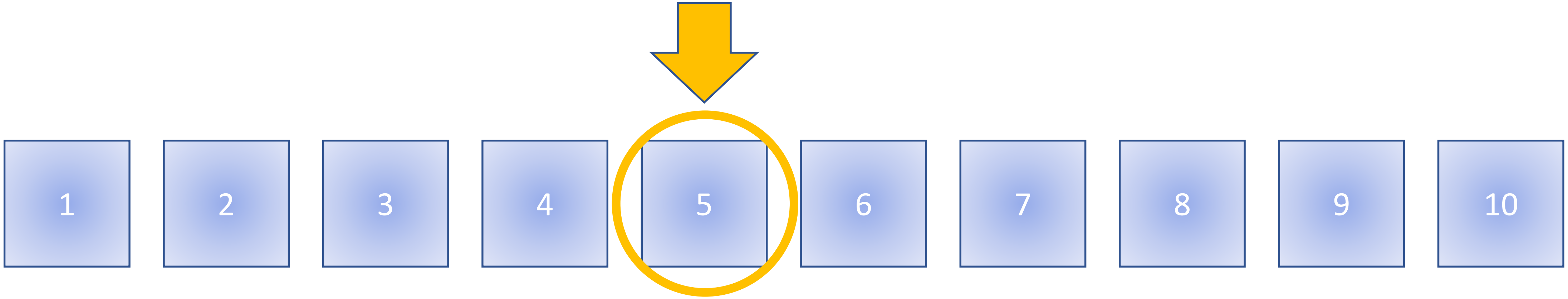


2^{nd} way to play (the smart way)

First turn: Guess $y = (a + b)/2$

I tell you it is not 5 AND my number is bigger.

What info have we gained?

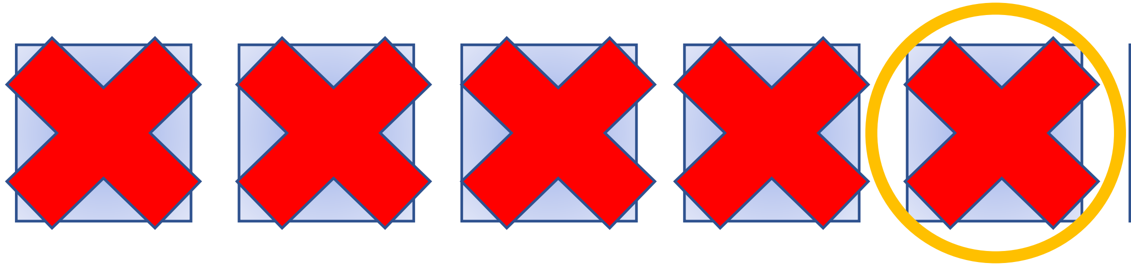


2^{nd} way to play (the smart way)

First turn: Guess $\mathrm{y} = (a+b)/2$

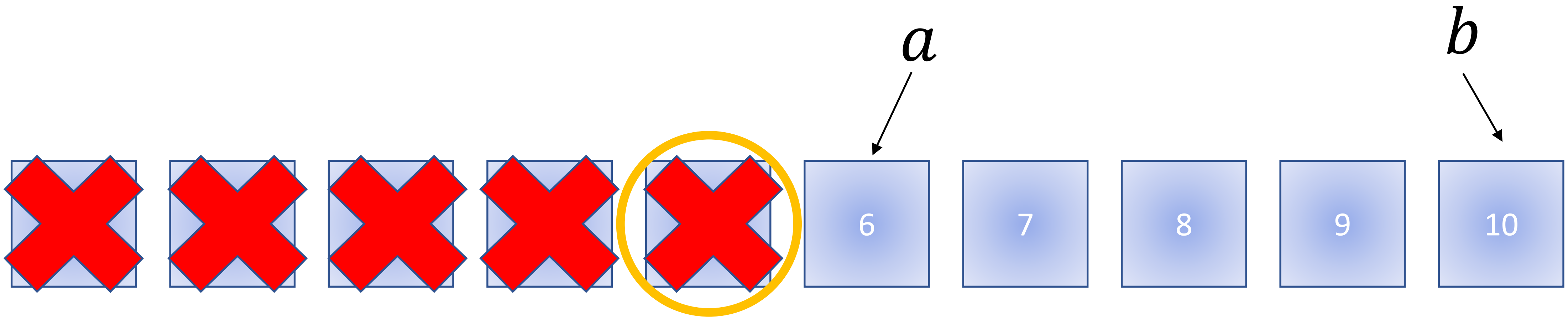
I tell you it is not 5 AND my number is bigger.

What info have we gained?



2^{nd} way to play (the smart way)

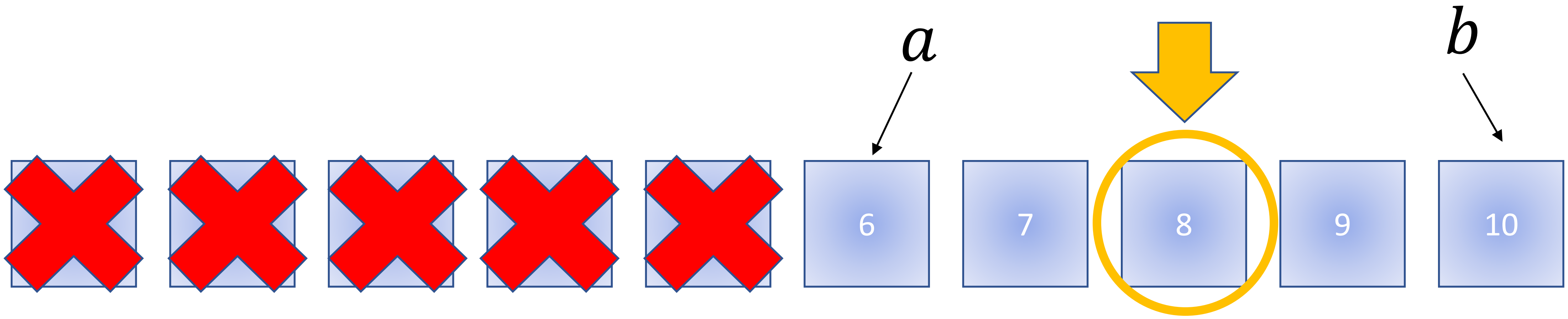
Second turn: We have a new range, we know the number is between $6\ and\ 10$. Now let $a = 6\ and\ b = 10$



2^{nd} way to play (the smart way)

Second turn: We have a new range, we know the number is between $6 \; and \; 10$. Now let $a = 6 \; and \; b = 10$

Guess $y = (a + b)/2$

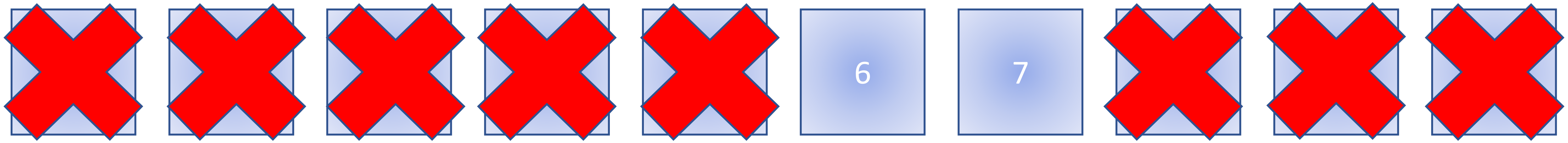


2^{nd} way to play (the smart way)

Second turn: We have a new range, we know the number is between $6 \text{ } and \text{ } 10$. Now let $a = 6 \text{ } and \text{ } b = 10$

Guess $y = (a + b)/2$

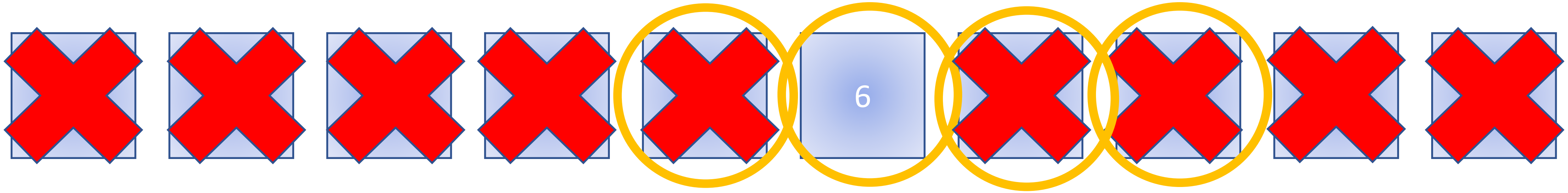
I tell you my number is smaller than y



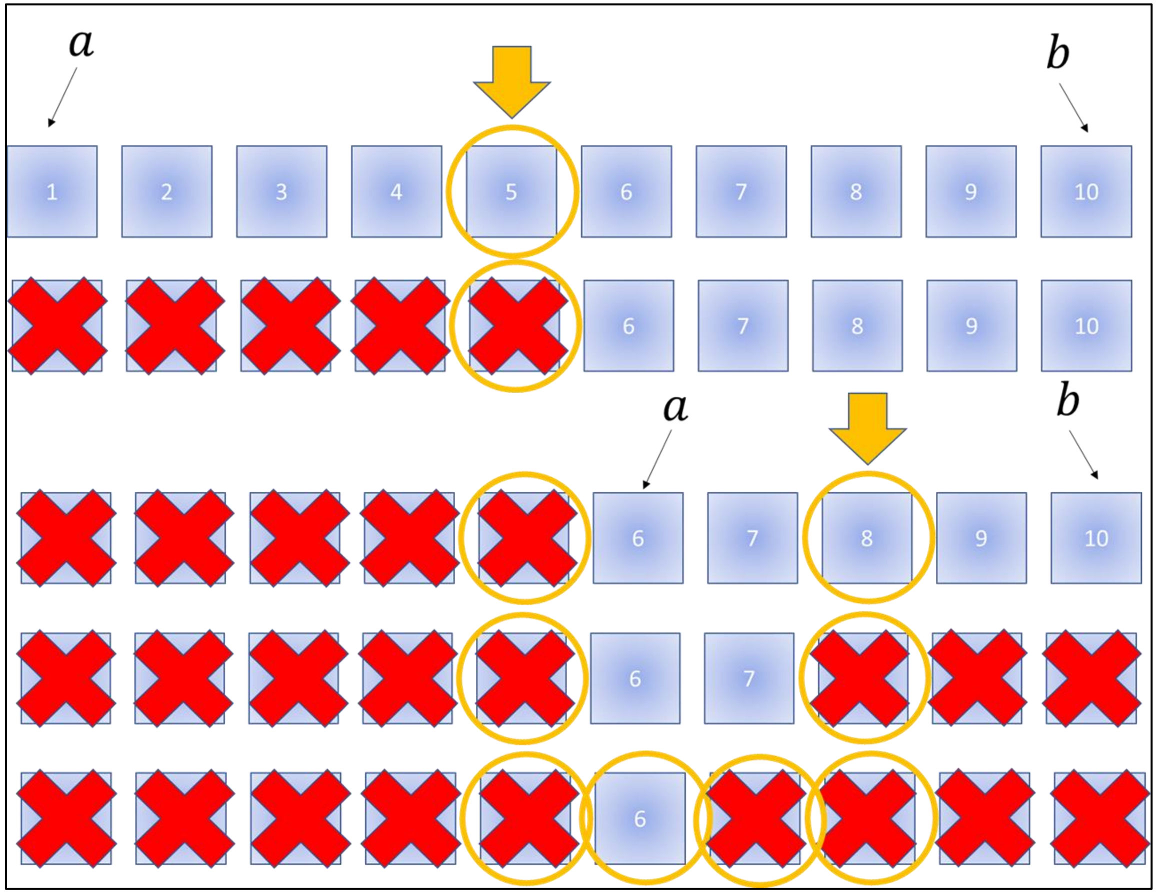
2nd way to play (the smart way)

Now it doesn't matter. We can guess 6 or 7. Let's say we guess 6 and it is wrong. Then we'll guess 7 and we have the answer. Worst case this means it took 4 guess.

Binary Search: Keep guessing the midpoint (and reducing the search space by half until) you reach a solution.



Comparison of strategies



- *Stupid Strategy*: Start at 1 and guess up. Worst Case: you lose $9 guessing.
- *Binary search strategy*: Start at midpoint and iteratively reduce. Worst Case: you lose $4.

Can you program binary search in Scheme?

Hint: First think about what the base case is.

When do we know that we guessed the right number? Well the midpoint is NOT less than x also the midpoint is NOT greater than x.

```
(define (guessMyNumber x a b)
  (define midpoint (round (* (+ a b) 0.5)))
  (display midpoint) (newline)
  (cond ((> x midpoint) (guessMyNumber x midpoint b))
        ((< x midpoint) (guessMyNumber x a midpoint))
        (#t midpoint)))
```

COMPUTING SQUARE ROOTS BY AVERAGING

- One simple way to compute an approximation to the square root of a number x is to
  - Start with two guesses, a and b, with the property that

$$a < \sqrt{x} < b$$

- (For example, if $x > 1$, we could start with $a = 1, b = x$.) Thus we know that the actual square root is between a and b.
- If $\frac{(a+b)}{2}$ is larger than the square root (which we can check by comparing $[\frac{(a+b)}{2}]^2$ with x) we know the real square root lies between a and $\frac{(a+b)}{2}$.
- Otherwise, the real square root lies between $\frac{(a+b)}{2}$ and b.

FOR EXAMPLE...

- To compute the square root of 10:
  - start with the window: [1, 10] (we know the square root lies in this range).
  - Consider $\frac{(1+10)}{2} = 5.5$. Since $5.5^2 > 10$, this is larger than sqrt(10).
  - Now we know the square root lies in $[1, 5.5]$.
- Repeating this process, we find that it lies in $[1, 3.25]$.
- Repeating again, we find that it lies in $[2.125, 3.25]$.
- ...

IN SCHEME

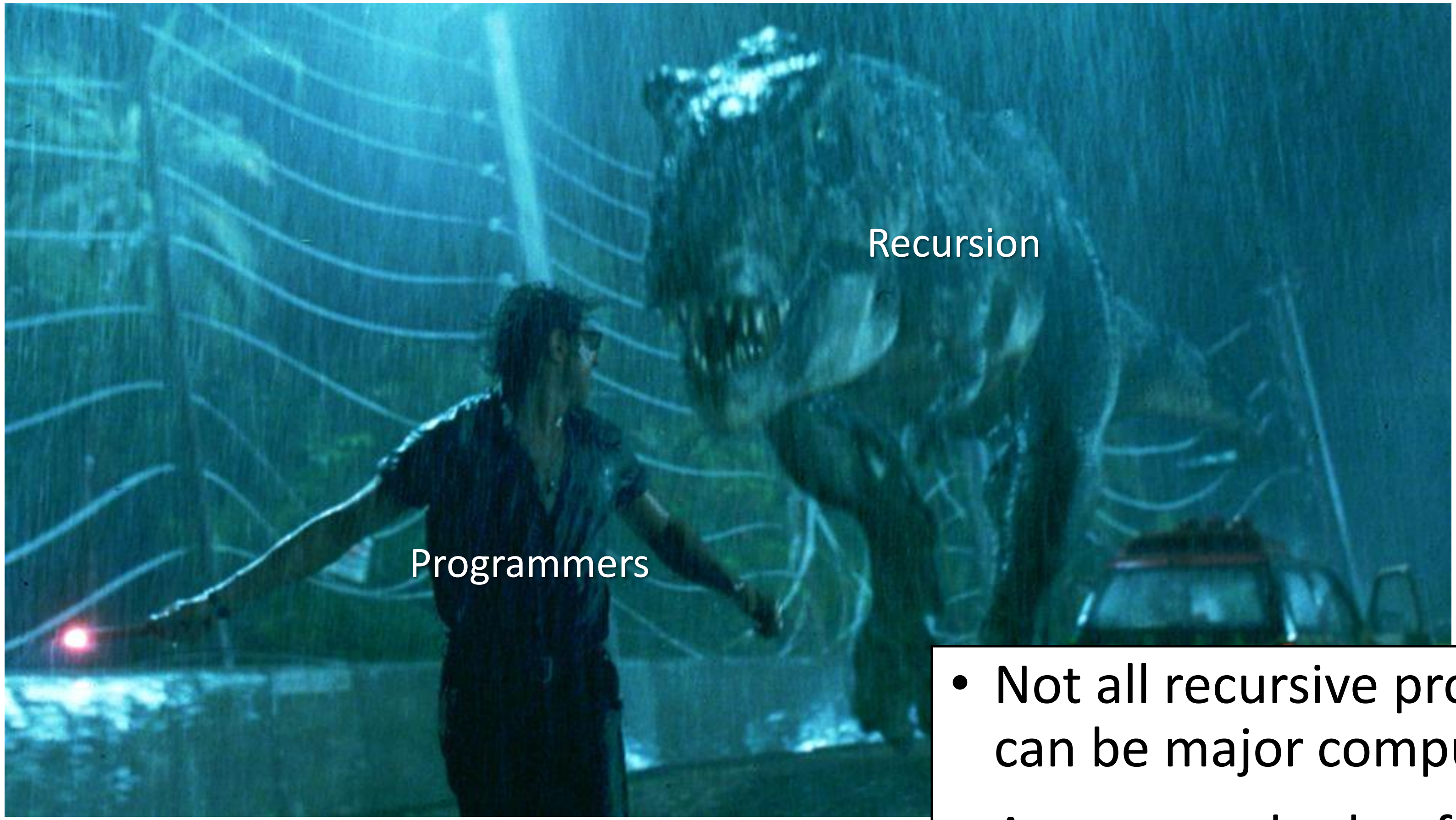
```
(define (average a b) (/ (+ a b) 2))
(define (square a) (* a a))

(define (sqrt-converge x a b)
  (if (< (abs (- a b)) .000001)
      a
      (if (> (square (average a b)) x)
          (sqrt-converge x a (average a b))
          (sqrt-converge x (average a b) b))))
```

Now, we might like to define a more attractive square root function that does not require choosing a and b:

```
(define (new-sqrt x) (sqrt-converge x 1 x))
```

Some other general strategies for dealing with recursion (beside running away)…



- Not all recursive problems decompose the same, there can be major computational differences.
- As a general rule of thumb, if possible try and think about what the **base case** would be first.
- If you are super-duper-duper stuck and have absolutely no clue…try to write it with a “for” loop. Then try and think about the recursive way to do it.

Figure Sources

- https://upload.wikimedia.org/wikipedia/commons/7/79/Operation_Upshot-Knothole_-_Badger_001.jpg
- https://www.memesmonkey.com/images/memesmonkey/b2/b2dd360b14b4f7d7680d90b3cd9376ba.jpeg
- https://en.meming.world/images/en/0/07/Drakeposting.jpg
- http://web.cs.ucla.edu/~klinger/dorene/Gif/escher-hands.gif
- https://ichef.bbci.co.uk/news/976/cpsprodpb/168A6/production/_101862329_a7a28e9d-b11b-46a7-aa57-b0b6dc8b5f19.jpg
- https://upload.wikimedia.org/wikipedia/commons/2/27/AG-7_Space_Pen.JPG
- https://static01.nyt.com/images/2020/03/24/arts/gaming-newbies-stadia/gaming-newbies-stadia-superJumbo.jpg
- https://www.themarysue.com/wp-content/uploads/2019/09/Ned-Was-Originally-in-the-Mid-Credit-Scenes-for-Far-from-Home-1200x675.jpg
- Greg Johnson’s Lecture Slides.