CS339: Abstractions and Paradigms for Programming

Recursion and Iteration

Manas Thakur
CSE, IIT Bombay

Autumn 2024

Let's look at the processes generated by procedures

- Factorial of a number:

$$fact(n) = \begin{cases} 1 & , n=1 \\ n * fact(n-1) & , o/w \end{cases}$$

- A procedure to compute the same:

```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

The generated process for `fact(5)`

```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

Time: O(n)
Space: O(n)

```
(fact 5)
(* 5 (fact 4))
(* 5 (* 4 (fact 3)))
(* 5 (* 4 (* 3 (fact 2))))
(* 5 (* 4 (* 3 (* 2 (fact 1)))))
(* 5 (* 4 (* 3 (* 2 1))))
(* 5 (* 4 (* 3 2)))
(* 5 (* 4 6))
(* 5 24)
120
```

Recursive Process

How about this one?

➤ Another way to compute factorial:

```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```

The generated process for `fact(5)`

```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```

Time: O(n)
Space: O(1)

Iterative Process

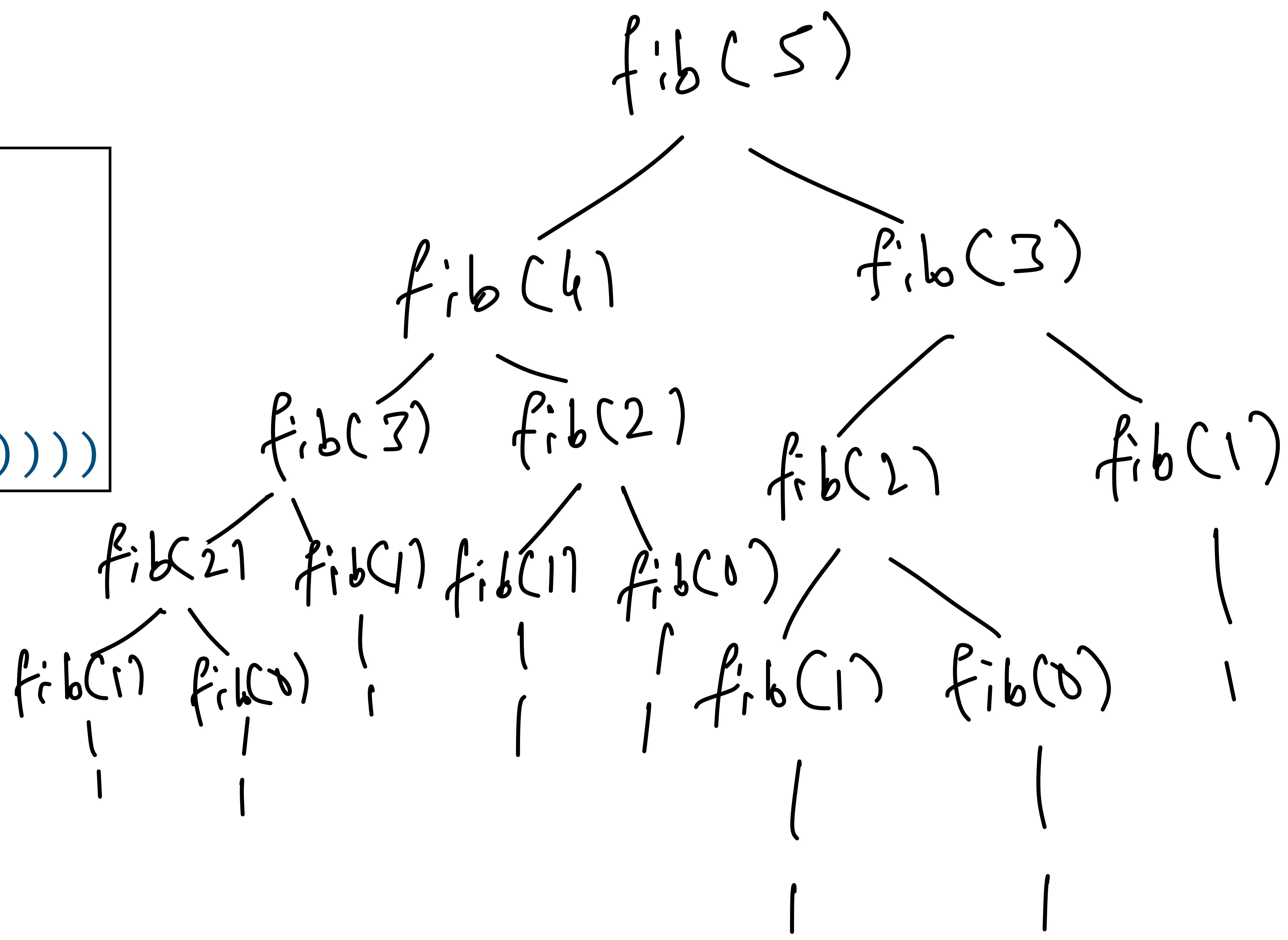
```
(fact 5)
(fact-iter 1 2 5)
(fact-iter 2 3 5)
(fact-iter 6 4 5)
(fact-iter 24 5 5)
(fact-iter 120 6 5)
120
```

Another recursive process

- Fibonacci numbers:

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

Tree-Recursive Process



Recursive vs Iterative Processes

```
(fact 5)
(* 5 (fact 4))
(* 5 (* 4 (fact 3)))
(* 5 (* 4 (* 3 (fact 2))))
(* 5 (* 4 (* 3 (* 2 (fact 1)))))
(* 5 (* 4 (* 3 (* 2 1))))
(* 5 (* 4 (* 3 2)))
(* 5 (* 4 6))
(* 5 24)
120
```

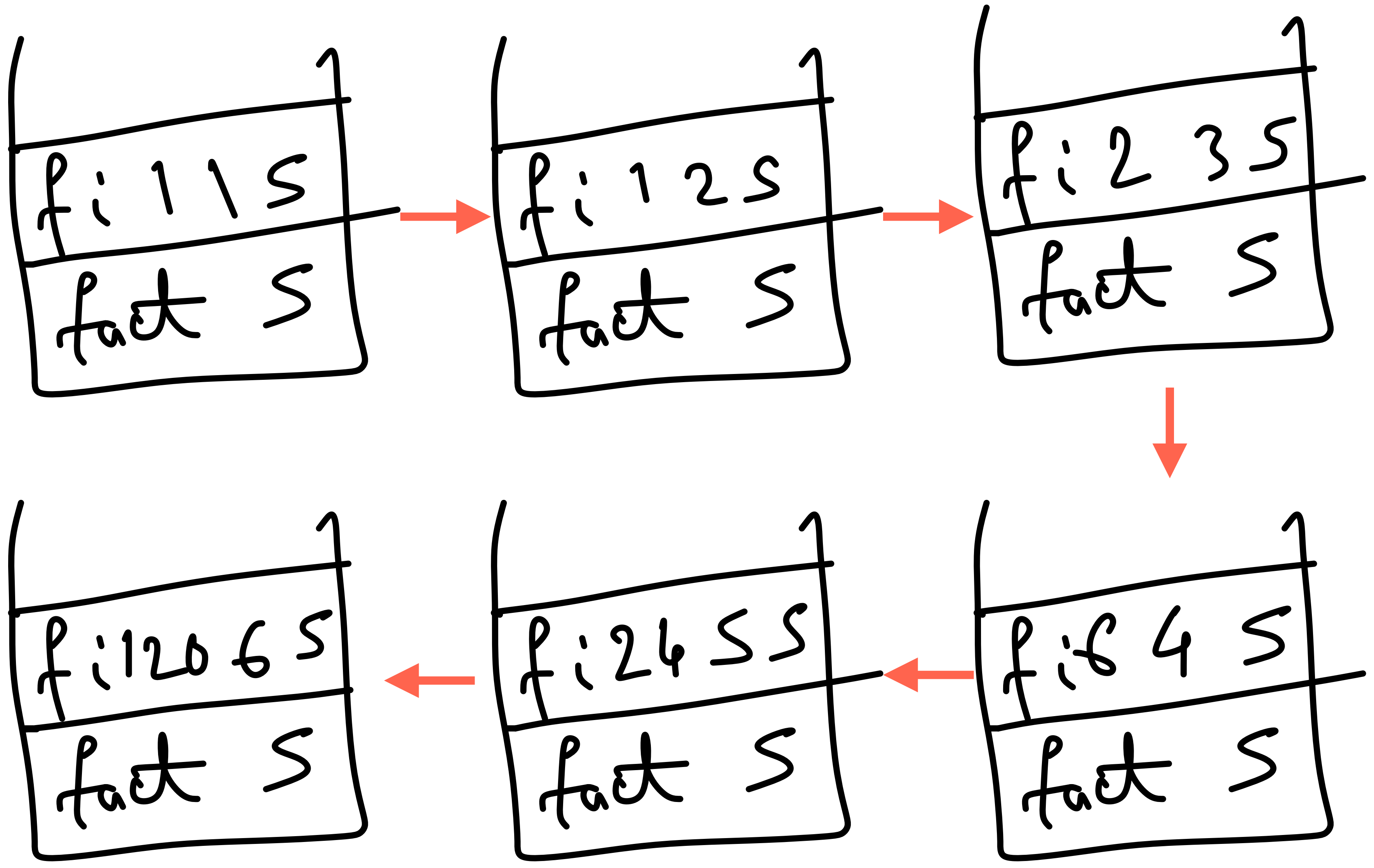
- Recursive: Grow then shrink.
- Recursive: Require more space.
- Iterative: State variables.
- Iterative: Can be resumed easily.
- Recursive: More *bureaucratic*.
- But even an iterative process generated by a recursive procedure requires more space!

```
(fact 5)
(fact-iter 1 2 5)
(fact-iter 2 3 5)
(fact-iter 6 4 5)
(fact-iter 24 5 5)
(fact-iter 120 6 5)
120
```

Tail-Call Optimization

- Iteration without looping constructs is expensive in space.
- But we can avoid returning when the recursive call is the tail!
- Saves stack space and makes iteration (nearly) as efficient as imperative languages with looping constructs.

```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```



Lab Modus Operandi

- Each lab has to be done individually.
- Only lab desktops. Fixed seat. No mobile phones.
- TAs would clarify your doubts and evaluate by seeing your code as well as asking questions. Their judgment would be final. We would rotate TAs.
- You can skip one lab; more than that would cause loss of marks.
- Maintain a silent atmosphere in the lab.
- DO NOT CHEAT.
- First day may be a bit confusing; bear with us and coordinate.