Process

- A sequence of execution of a procedure
- How a procedure consumes the computational resources
Simple Process

- def square(x) = x * x
- This process is fairly simple
- One level on the call stack

\[
\begin{align*}
square(5) & \quad 5 * 5 \\
5 * 5 & \quad \text{square}(5)
\end{align*}
\]

Iterative vs. Recursive

**Iterative**

- A procedure or process is considered iterative when it loops through a set of values to perform a certain operation
- State is determined by fixed set of variables and rules

**Recursive**

- It is considered recursive when it calls itself to fulfill the operation
- There is a phase of expansion with deferred evaluation followed by a phase of contraction
- Interpreter or runtime needs to keep track of deferred computations
Procedure vs. Process

- Just because a procedure is recursive, it does not mean it’s process is recursive!
- You can have a recursive procedure with an iterative process
- We’ll examine this further

Iterative vs. Recursive Process

- Iterative Process carries a series of steps
- You can stop at anytime and resume later even on another processor with the current state
- Recursive Process are highly expressive
- Recursive process, on the other hand, requires that you carry the current chain or sequence of calls with it
- Often not desirable for a large sequence since the demand on resource is high
- However, do not use that as a reason to shy away from recursion, you can write a recursive procedure that runs as iterative process
Ways to find factorial

- Recall 5! is 5 * 4 * 3 * 2 * 1 and is equal to 120
- There are a few different ways to find factorial of a number
- You can write it iterative or recursive
- You can process it iterative or recursive

Option 1

```scala
def fact(n: Int) = {
  var factorial = 1
  for(i <- 1 to n) factorial *= i
  factorial
}
println(fact(5))
```
Option 1: Iterative/Iterative

Procedure and Process are iterative in this case

\[
\text{fact(5)} \\
factorial = 1 \times 1 \\
factorial = 1 \times 2 \\
factorial = 2 \times 3 \\
factorial = 6 \times 4 \\
factorial = 24 \times 5 \\
120
\]

Option 2

```scala
def fact(n: Int) : Int = {
  if(n <= 1) 1 else n * fact(n-1)
}
println(fact(5))
```
Option 2: Recursive/Recursive

- Procedure and Process are recursive in this case

```
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
```

Same sequence and number of steps as before

Much more expensive in resource usage, however

Tail Recursion

- There is a trick some languages can perform

- If the recursive call is the last expression to be evaluated, they can roll the call into a simple iteration instead of a jump!

- So, what appears to a recursion ends up being an iteration

- This leads to recursive procedure—iterative process
Option 3

def fact(n: Int) : Int = {
    def fact(partialResult: Int, currentNumber : Int) : Int = {
        if (currentNumber == 1)
            partialResult
        else
            fact(partialResult * currentNumber, currentNumber - 1)
    }
    fact(1, n)
}

Option 3: Recursive/Iterative

Procedure is iterative; Process is Recursive

fact(5)
fact(5, 4)
fact(20, 3)
fact(60, 2)
fact(120, 1)
120