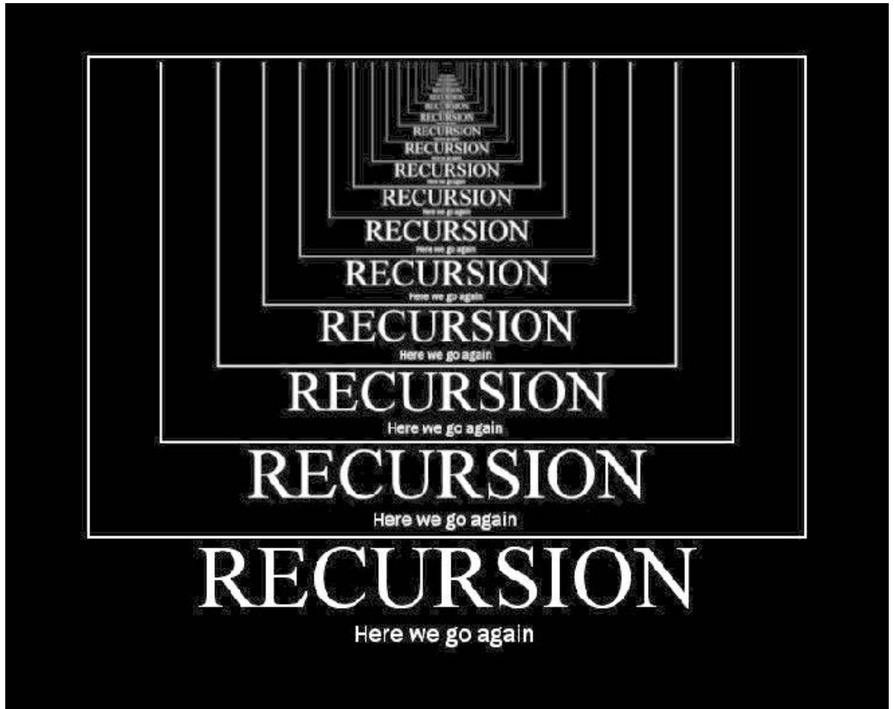
Functional programming



Recursion occurs when a thing is defined **in terms of itself** or of its type. The most common application of recursion is in mathematics and computer science, where a function being defined is applied within its own definition.

Loop

C/C++

```
int fibonacci(int n) {
  if (n < 2) {
    return 1;
  }
  int twoBefore = 1;
  int oneBefore = 1;
  int current;
  for (int i = 2; i <= n; i++) {
    current = oneBefore + twoBefore;
    twoBefore = oneBefore;
    oneBefore = current;
  }
  return current;
}
```

Haskell

Recursion

C/C++

```c
int fib(int n) {
  if (n < 2) {
    return 1;
  } else {
    return fib(n - 1) + fib(n - 2);
  }
}
```

Haskell

```haskell
fib n =
  if n < 2
  then 1
  else fib(n - 1) + fib(n - 2)
```

Pattern guards


```haskell
fib n =
  if n < 2
  then 1
  else fib(n - 1) + fib(n - 2)
```

```haskell
fib' :: (Integral a) => a -> a
fib' 0 = 1
fib' 1 = 1
fib' n = fib'(n - 1) + fib'(n - 2)
```

```haskell
fib'' :: (Integral a) => a -> a
fib'' n
  | n < 2      = 1
  | otherwise = fib''(n - 1) + fib''(n - 2)
```

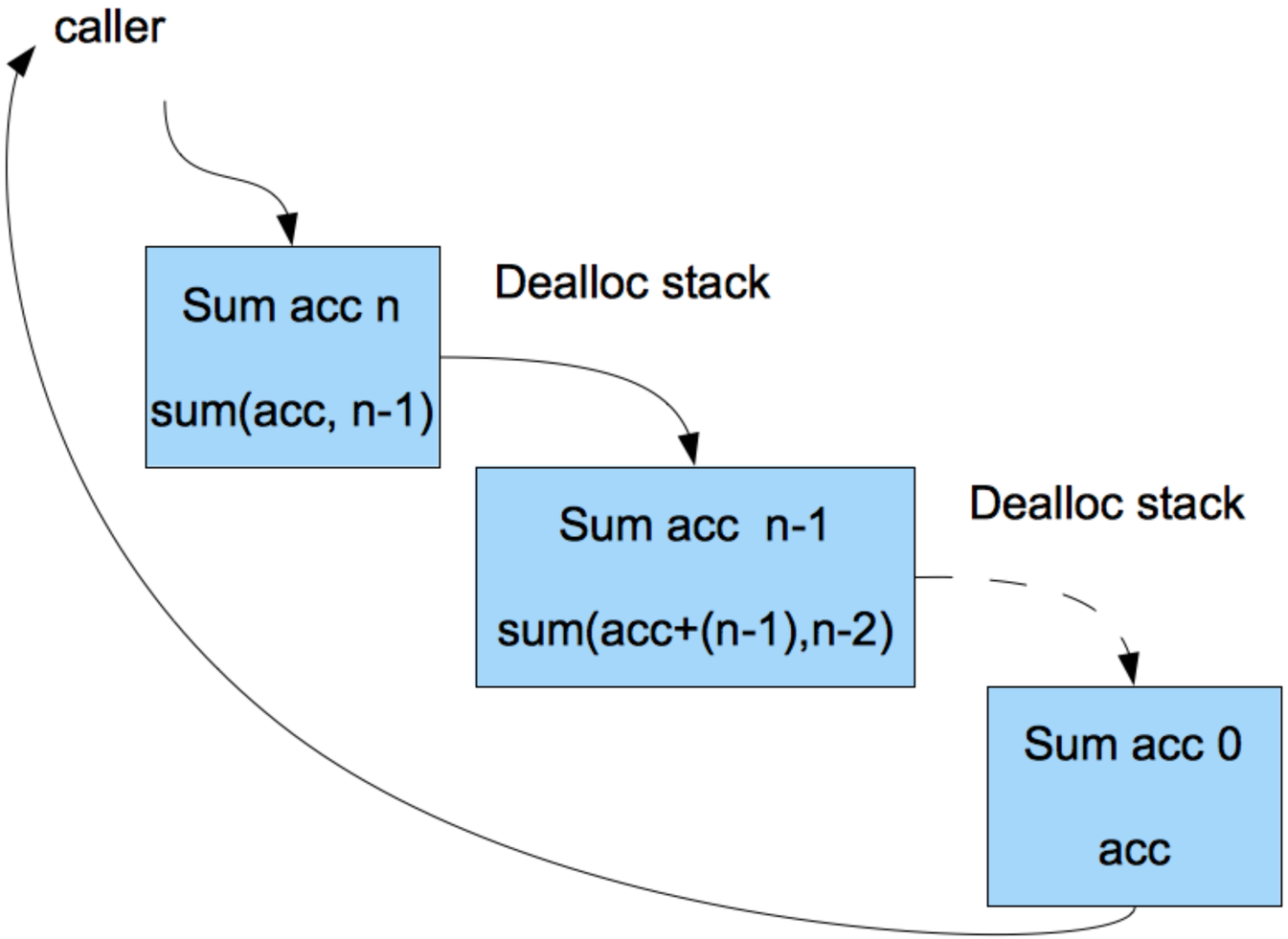
Tail recursion

The term *tail recursion* refers to a form of recursion in which the final operation of a function is a call to the function itself.

```
fib'' :: (Integral a) => a -> a
fib'' n
  | n < 2       = 1
  | otherwise = fib''(n - 1) + fib''(n - 2)
```

Not a final operation

Tail recursion



Tail recursion

```
sum' n
  | n == 0     = n
  | otherwise = n + sum'(n - 1)
```

final operation is n + sum’

```
sum'' n acc
  | n == 0 = acc
  | otherwise = sum''(n - 1) (acc + n)
```

final operation is sum”

Tail recursion


```haskell
tailFibs prev1 prev2 start end
  | start == end  = next
  | otherwise     = tailFibs next prev1 (start + 1) end
  where next = prev1 + prev2

fibTail n = tailFibs 0 1 0 n
```

Thanks

- A **thunk** is a value that is yet to be evaluated.
- **Thunk** is used in Haskell systems, that implement non-strict semantics by lazy evaluation.
- A lazy run-time system does not evaluate a **thunk** unless it has to.

Challenges

- Reverse numbers in a natural number *1234 -> [4, 3, 2, 1]*
- Check if a word is a palindrome. *“ABBA” -> True; “Queen” -> False*
- Find a maximum in the list. [1, 10, 9, 7, 15, 3] -> 15
- A natural number is given. Function should return **True** if a given number is a power of two and **False** if it’s not. *2 -> True; 14 -> False; 16 -> True*