Homework 12: HOFs in Haskell

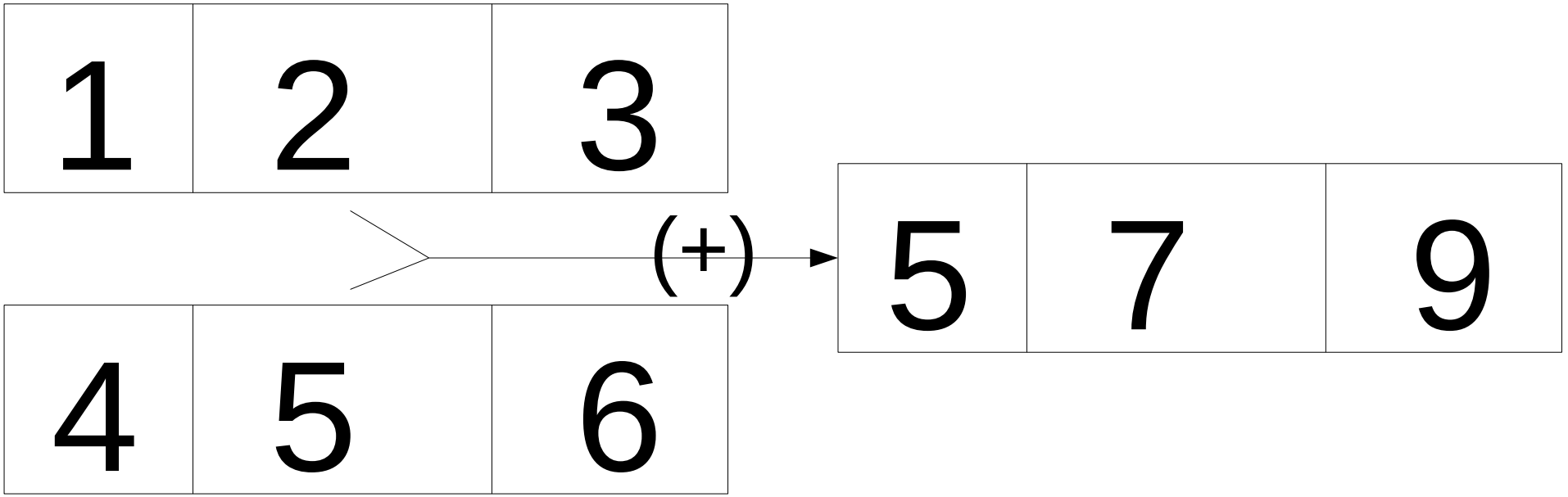
- Higher Order Functions (HOFs) are functions that take functions as arguments to perform functions with common patterns. The resulting functions usually operate on lists.
- Hopefully this is just review and Prof. Liu has gone through this.

Lambda expressions

- Sometimes we'll want to create a simple function to use only once.
- Using with HOFs is a typical case.
- Format:
  (\arguments -> expression)
- Examples:
  (\x -> x+1)
  (\x y z -> x+y+z) – multiple arguments are passed this way
  (\[x,y,z] -> x+y) – lists can be passed
  (\x:xs -> x) - this WON'T work
  (\(x,y,z) -> x) – tuples can be passed

zipWith

- zipWith takes a binary operator and 2 lists, then uses the operator to combine the lists together
- Example:
  zipWith (+) [1,2,3] [4,5,6] returns [5,7,9]

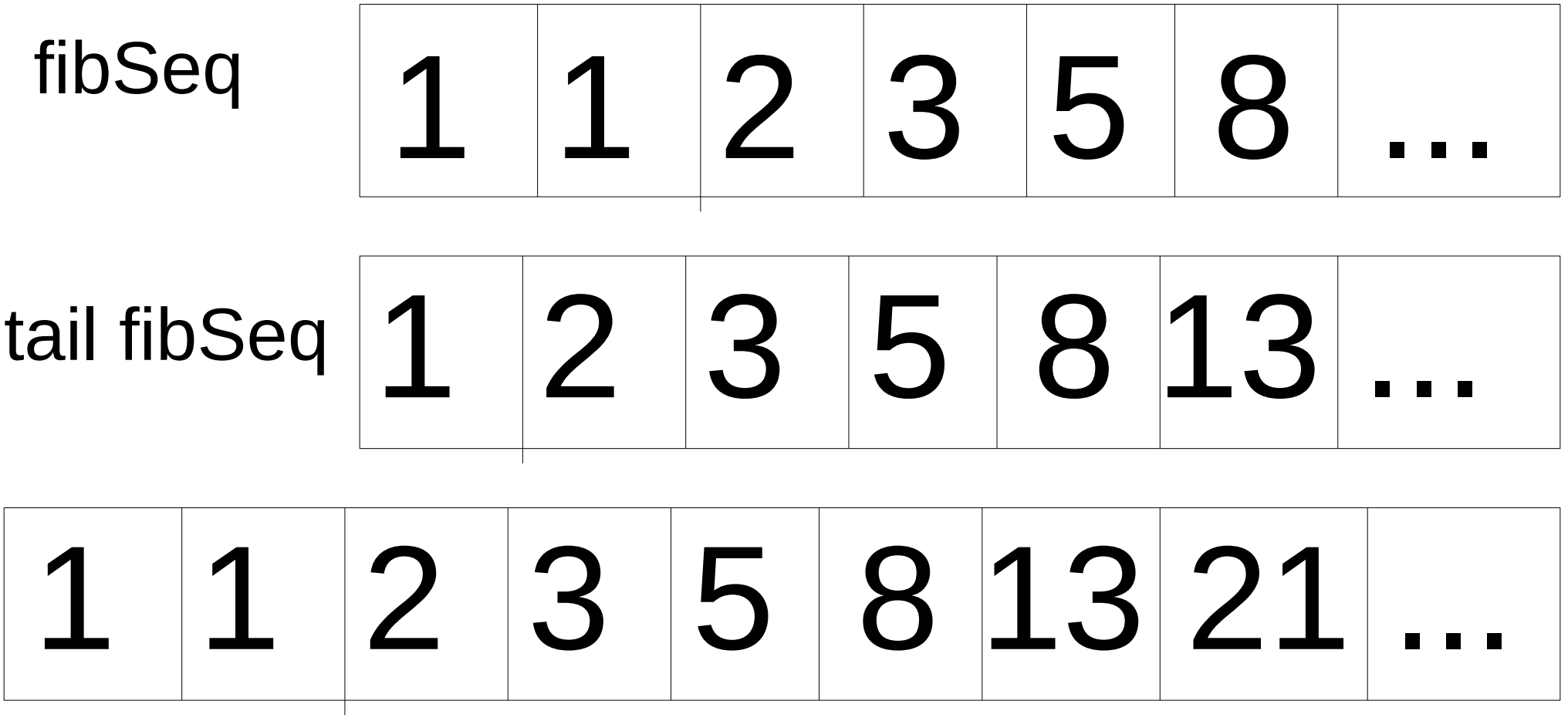


zipWith

- The input lists and output lists could all be of different types.
- zipWith (\x y -> (x == “g” && y == 7))
  [“f”, “g”, “h”, “i”] [6,7,7,8]
  returns [False,True,False,False].

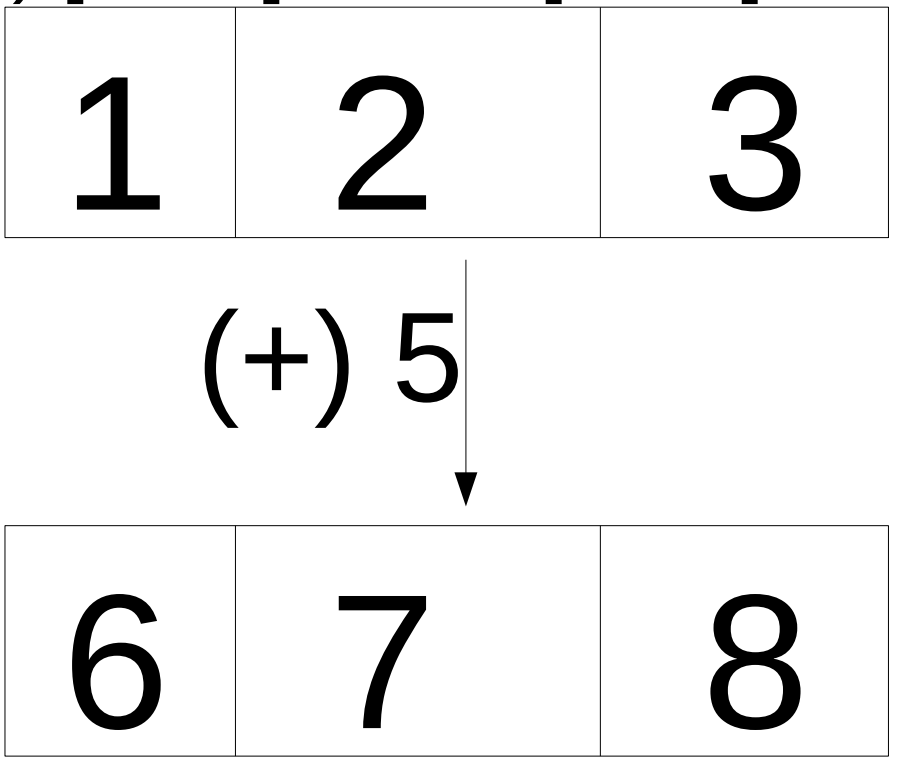
zipWith for Fibonnaci

- fibSeq = 1:1:[f | f <- zipWith (+) fibSeq (tail fibSeq)] will generate the infinite Fibonacci sequence.



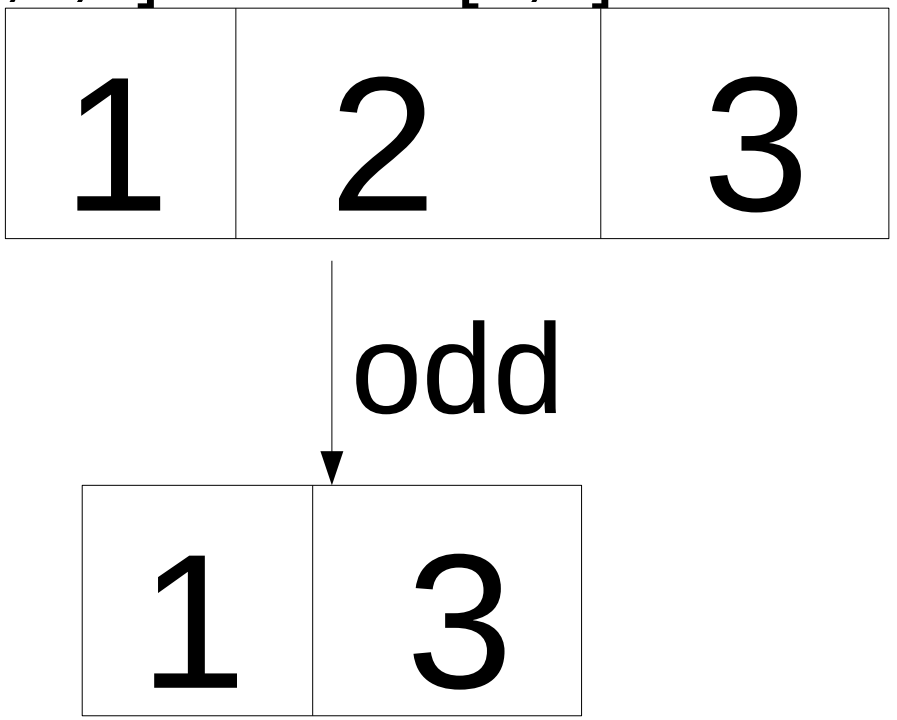
map

- map takes a unary function and applies to each element in a list, producing a list of the same size
- Simple example:
  map ((+) 5) [1,2,3] return [6,7,8]



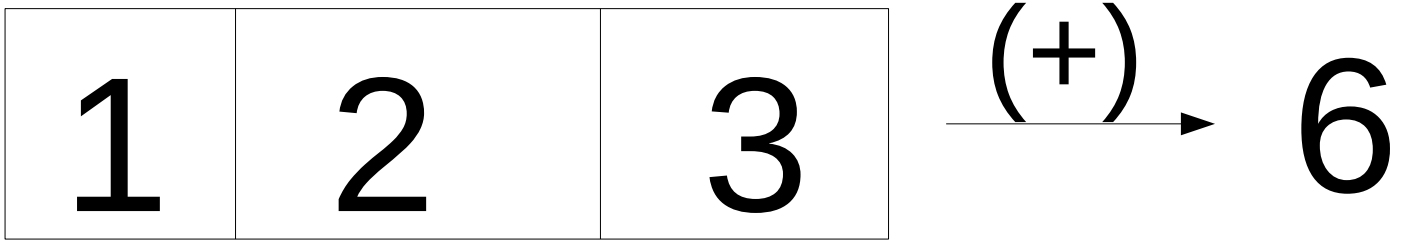
filter

- filter takes a unary function that returns a boolean, and uses it on a list to produce a smaller list
- Simple example:
  filter odd [1,2,3] returns [1,3]



folds

- folds use a binary operator to compress a list into a single element
- Simple example:
  foldl1 (+) [1,2,3] returns 6 - we've taken the values in the list and applied (+) between them
- We can do other things:
  foldl1 (++) [[1,2],[3,4],[5,6]] return the lists concatenated together: [1,2,3,4,5,6]



folds

- There are 4 fold functions: foldl, foldr, foldl1 and foldr1.
- foldl and foldl1 use left associativity, so
  foldl1 (-) [10,6,2] returns 2 and
  foldr1 (-) [10,6,2] returns 6.
- foldl and foldr require an initial accumulator, where foldl1 and foldr1 use the first value in the list
  foldl (+) 0 [1,2,3] is equivalent to foldl1 (+) [1,2,3]
- We can use other values for the initial value,
  foldl (+) 10 [1,2,3] will return 16.

Sometimes we can’t use foldl1

- foldl is necessary if the accumulator is not the same type as the elements of the list
- foldl (\str val -> if val < 0 then
  str ++ “n” else str ++ “p”) “” [1,(-2),3]
  will return “pnp”

More complex example

- dist p q = sqrt (foldl1 (+) (map (** 2)
  (zipWith (-) p q)))
- This function finds this distance between 2 points of arbitrary dimensions.
- dist [0,0] [1,1] returns sqrt(2)
- dist [0,0,0,0,0] [1,1,1,1,1] returns sqrt(5)