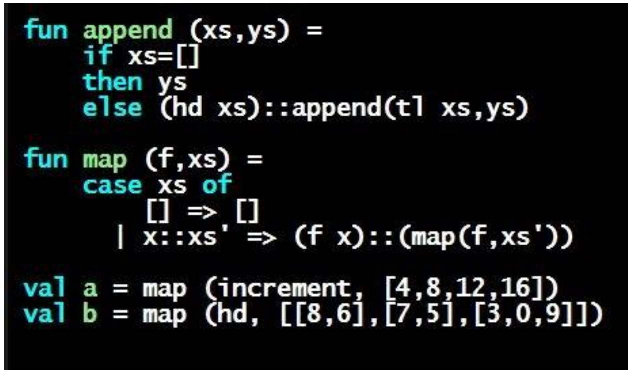


```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

fun map (f,xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f,xs'))

val a = map (increment, [4,8,12,16])
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman
2013

Let Expressions to Avoid Repeated Computation

Avoid repeated recursion

Consider this code and the recursive calls it makes

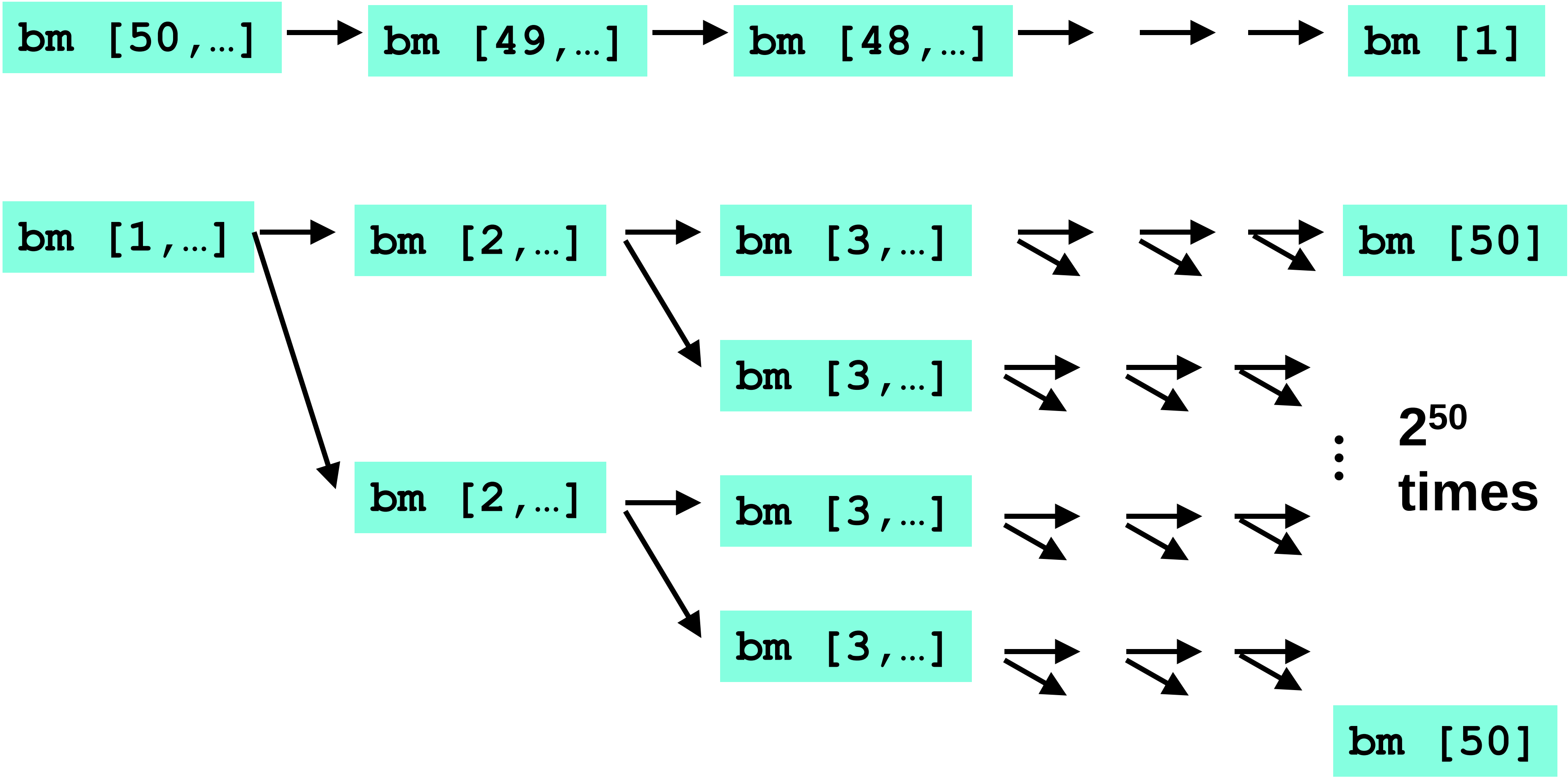
- Don’t worry about calls to **`null`**, **`hd`**, and **`tl`** because they do a small constant amount of work

```
fun bad_max (xs : int list) =
    if null xs
    then 0 (* horrible style; fix later *)
    else if null (tl xs)
    then hd xs
    else if hd xs > bad_max (tl xs)
    then hd xs
    else bad_max (tl xs)

let x = bad_max [50,49,…,1]
let y = bad_max [1,2,…,50]
```

Fast vs. unusable

```
if hd xs > bad_max (tl xs)
then hd xs
else bad_max (tl xs)
```



bm [50,…]

bm [49,…]

bm [48,…]

bm [1]

bm [1,…]

bm [2,…]

bm [3,…]

bm [50]

bm [3,…]

bm [2,…]

bm [3,…]

bm [3,…]

2^{50} times

bm [50]

Math never lies

Suppose one **`bad_max`** call's if-then-else logic and calls to **`hd`**, **`null`**, **`tl`** take 10^{-7} seconds

- Then **`bad_max [50,49,…,1]`** takes 50 x 10^{-7} seconds
- And **`bad_max [1,2,…,50]`** takes 1.12 x 10^{8} seconds
  - (over 3.5 years)
  - **`bad_max [1,2,…,55]`**takes over 1 century
  - Buying a faster computer won't help much ☺

The key is not to do repeated work that might do repeated work that might do…

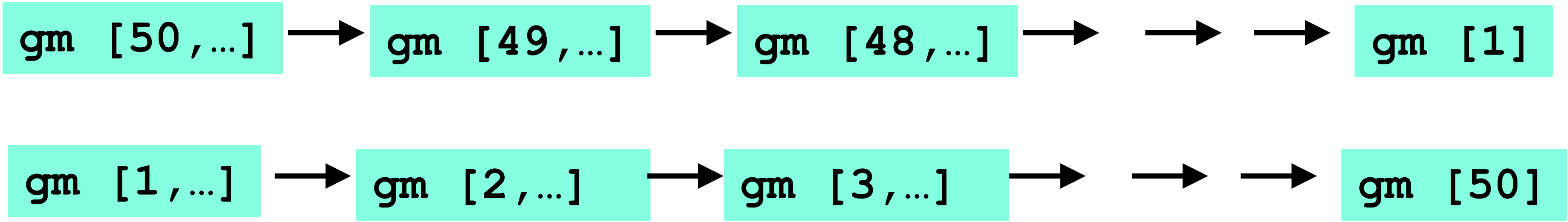
- Saving recursive results in local bindings is essential…

Efficient max

```
fun good_max (xs : int list) =
    if null xs
    then 0 (* horrible style; fix later *)
    else if null (tl xs)
    then hd xs
    else
         let val tl_ans = good_max(tl xs)
         in
             if hd xs > tl_ans
             then hd xs
             else tl_ans
         end
```

Fast vs. fast

```
let val tl_ans = good_max(tl xs)
in
     if hd xs > tl_ans
     then hd xs
     else tl_ans
end
```



gm [50,…] → gm [49,…] → gm [48,…] → → → gm [1]

gm [1,…] → gm [2,…] → gm [3,…] → → → gm [50]