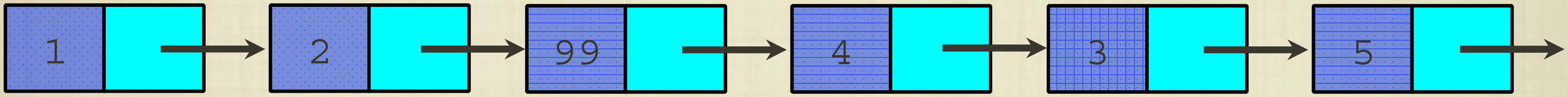
Functional Programming IV

Spring 2014
Carola Wenk

Data Structures in Scheme

- Scheme does not appear to have arrays, references, or pointers. Can we still represent the data structures we’ve talked about?

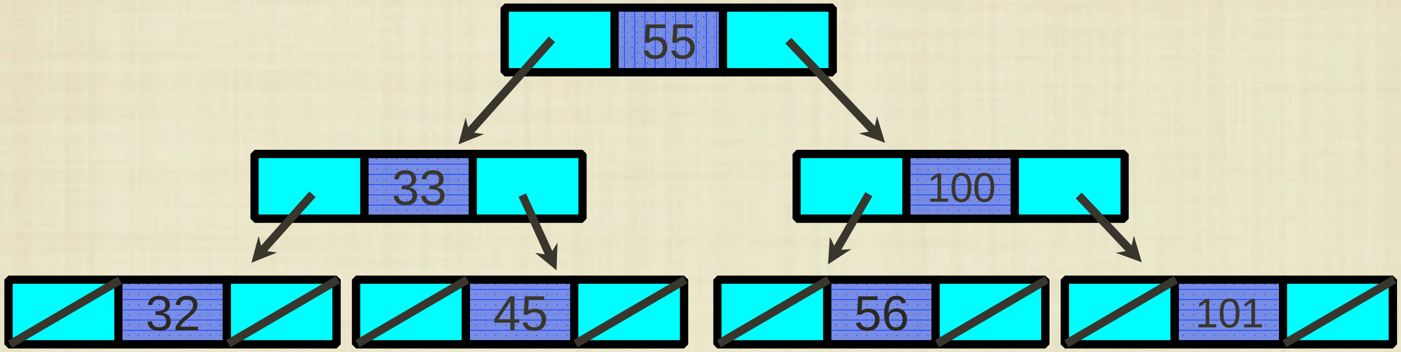


```
(1 2 99 4 3 5)
```

Abstractly, this is just a list, and that’s all we need to know.

Data Structures in Scheme

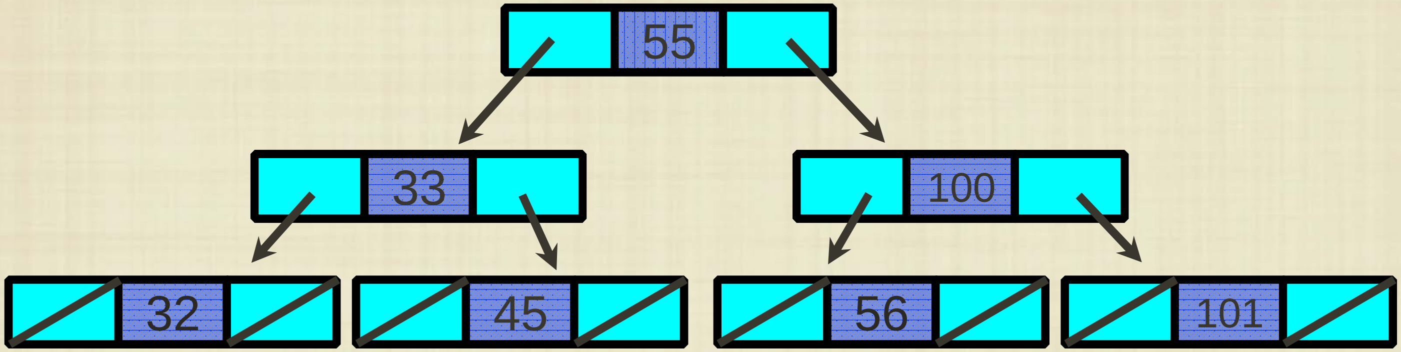
- Scheme does not appear to have arrays, references, or pointers. Can we still represent the data structures we've talked about?



What is the recursive definition of a binary search tree?

Data Structures in Scheme

- Scheme does not appear to have arrays, references, or pointers. Can we still represent the data structures we’ve talked about?



```
(55 (33 (32) (45))(100 (56) (101)))
```

Any linked structure can actually be represented by a nested list.

How do we find an element in a binary search tree?

Binary Search Trees

- In Scheme, a binary search tree can be treated as a nested list. Does our method to find an element in a binary search tree need to be updated?

```
(define (bst-find T x)
  (if (equal? T '()) #false
      (cond ((= x (car T)) #true)
            ((< x (car T)) (bst-find (car (cdr T)) x))
            (else (bst-find (car (cdr (cdr T))) x)))))
```

Actually, for some inputs this might cause a runtime error. Why?

Binary Search Trees

- In Scheme, a binary search tree can be treated as a nested list. Does our method to find an element in a binary search tree need to be updated?

```
(define (bst-find T x)
  (if (equal? T '()) #false
      (cond ((= x (car T)) #true)
            ((< x (car T)) (if (null? (cdr T))
                                      #f
                                      (bst-find (car (cdr T)) x)))
            (else (if (null? (cdr T))
                      #f
                      (bst-find (car (cdr (cdr T))) x))))))
```

What is the running time to evaluate this function?

As before, it is dependent on the (nesting) depth or height of the tree.

Higher Order Functions

Scheme’s methodology for evaluating functions allows us to actually pass functions as parameters to other functions.

```
(define (add4 x) (+ 4 x))
(define (add5 x) (+ 5 x))

(define (mult2 f y) (* 2 (f y)))

(mult2 add5 7)
```

Functions can also be declared “anonymously” using the `lambda` keyword:

```
(mult2 (lambda (x) (+ 6 x)) 7)

(mult2 (lambda (x) (* x x)) 7)
```

Defining Functions

In fact, the syntax for defining functions:

```
    (define (f x) body)
```

is a shortcut for:

```
    (define f (lambda (x) body))
```

Map and Reduce

The `map` function in Scheme takes a function and a list as arguments and applies the function to each element of the list.

```
(map (lambda (x) (+ x 1)) ‘(1 2 3 4 5 6))
```

The `foldr` function in Scheme takes a (binary) function, an initial value and a list as arguments and applies the function “right-to-left”.

```
(foldr + 0 ‘(1 2 3 4 5 6))

(foldr (lambda (x y) (+ x y)) 0 ‘(1 2 3 4))
```

Map and Reduce

The `map` function in Scheme takes a function and a list as arguments and applies the function to each element of the list.

```
(map (lambda (x) (+ x 1)) ‘(1 2 3 4 5 6))
```

The `foldr` function in Scheme takes a (binary) function, an initial value and a list as arguments and applies the function “right-to-left”.

```
(foldr cons ‘() ‘(1 2 3 4 5))
  = (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 ‘())))))
  = ‘(1 2 3 4 5)
```

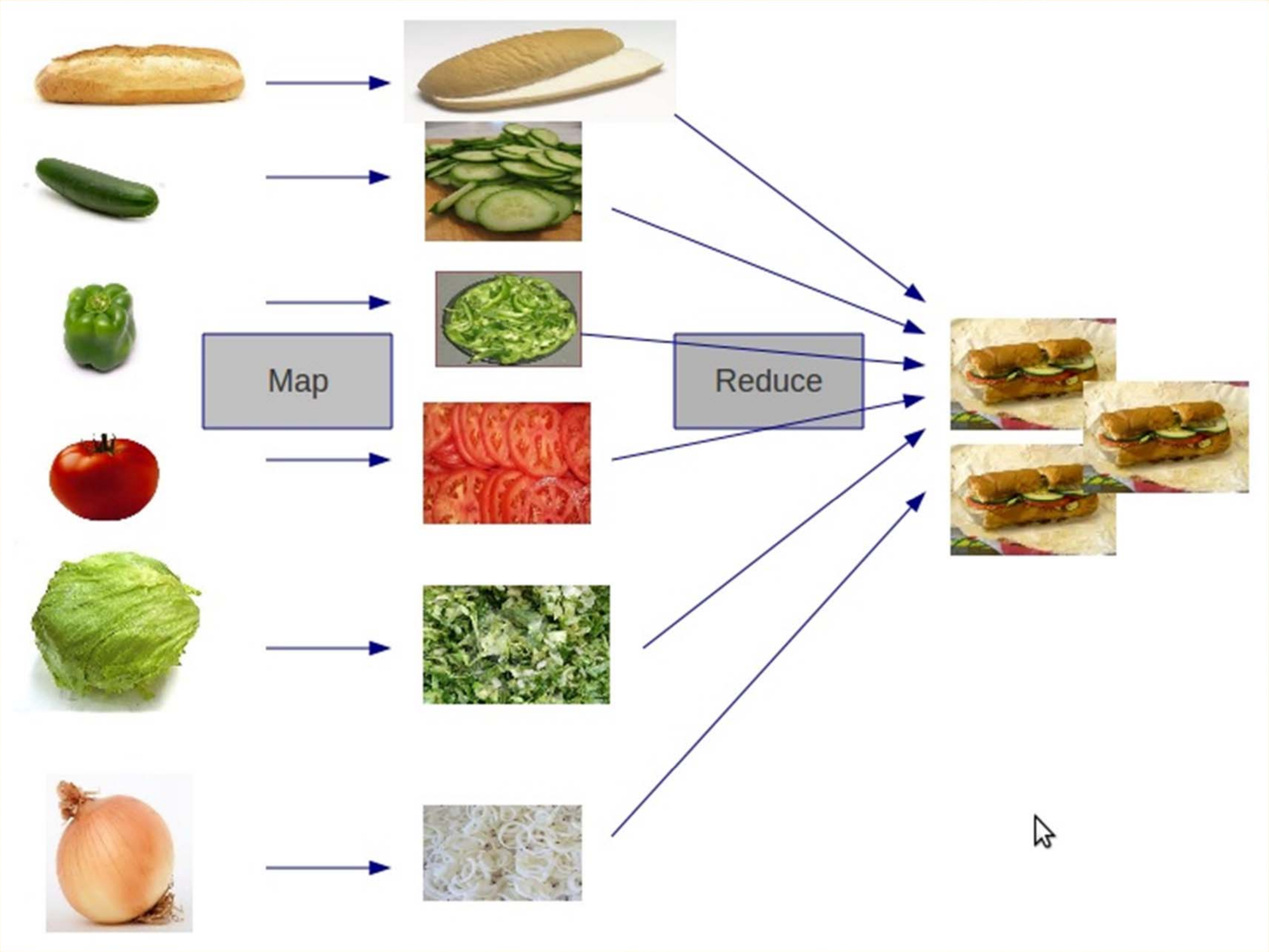
Map and Reduce

The `map` function in Scheme takes a function and a list as arguments and applies the function to each element of the list.

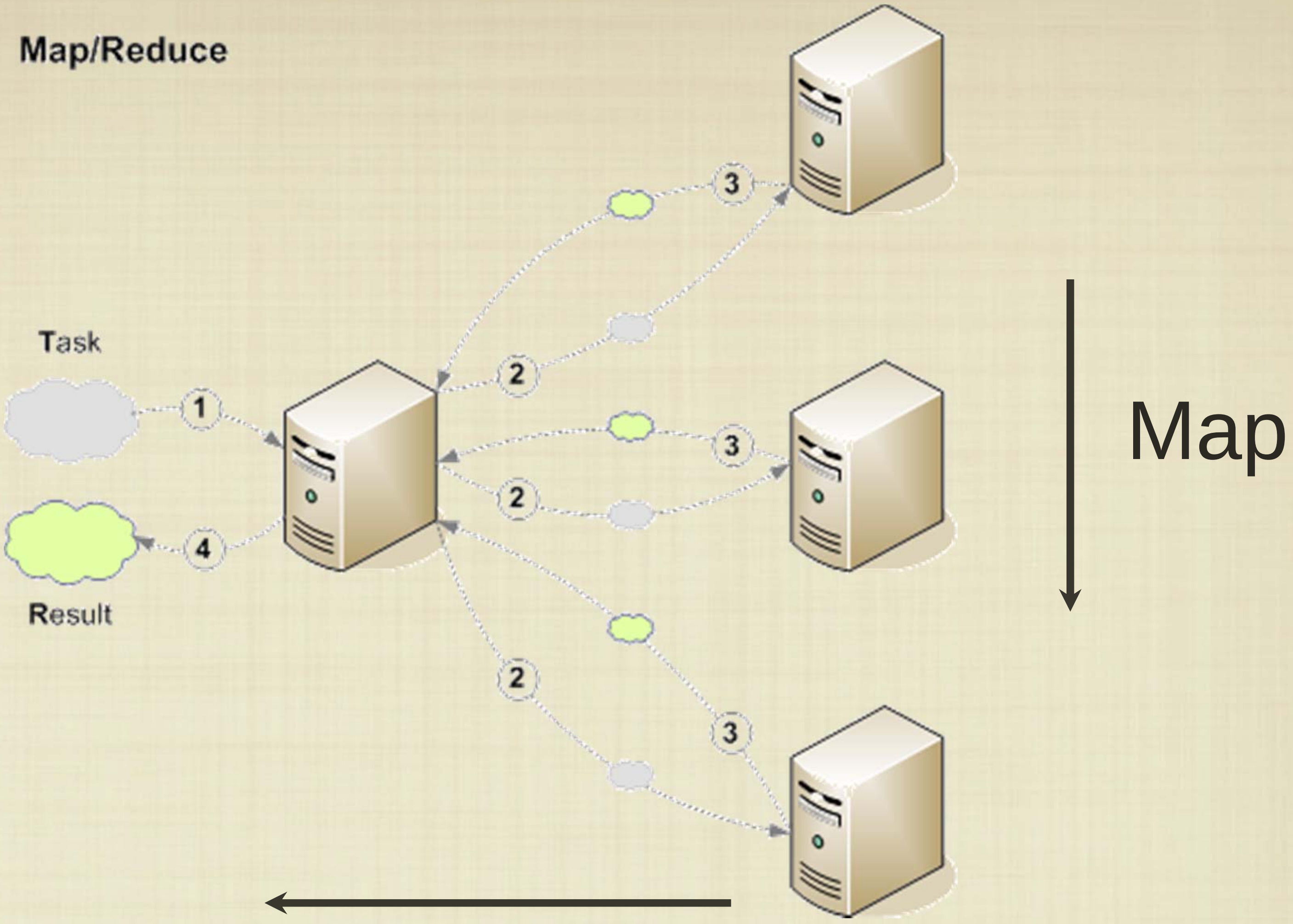
```
(map (lambda (x) (+ x 1)) ‘(1 2 3 4 5 6))
```

The `foldr` function in Scheme takes a (binary) function, an initial value and a list as arguments and applies the function "right-to-left". `foldl` simply works in the other direction.

```
(foldl cons ‘() ‘(1 2 3 4 5))
  = (cons 5 (cons 4 (cons 3 (cons 2 (cons 1 ‘()))))))
  = ‘(5 4 3 2 1)
```



Google’s MapReduce framework was inspired by constructs in LISP. The strength of this framework is that Map and Reduce can be done in parallel.



Map

Reduce

Google's MapReduce framework was inspired by constructs in LISP. The strength of this framework is that Map and Reduce can be done in parallel.