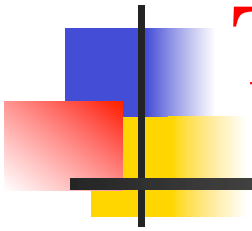
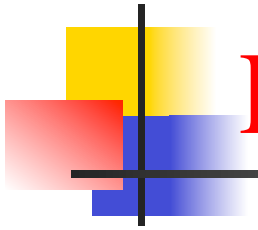


Trees

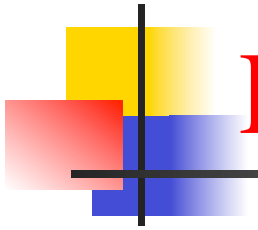
Corky Cartwright
Vivek Sarkar
Department of Computer Science
Rice University



Recap of Previous Lecture

Data-directed design

- Start with *data definition*
  - Specifies structure of data
- Derive *function template* from data definition
  - Model for any function that can be performed on data
  - Use generic name (e.g., *f*) for function
- Create *template instantiation* for a specific function and primary argument
  - Use specific name (e.g., big-class?) for function
  - Define separate auxiliary functions for other arguments if needed
- Develop *code* based on template instantiation



Example of Data-Directed Design

- **Data definition**

  ;; A natural-number (`n` for short) is either `0`,

  ;; or `(add1 n)`, where `n` is a natural-number

- **Function template**

```
;; f : natural-number -> ...
;; (define (f ... n ...)
;;   (cond [(zero? n) ...]
;;         [(positive? n) ...
;;          (f ... (sub1 n) ...)...]))
```

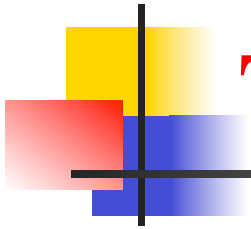
Example of Data-Directed Design (contd)

- **Template instantiation**

```
;; right-add : natnum, natnum -> natnum
;; (define (right-add m n)
;;  (cond [(zero? n) ...]
;;        [(positive? n) ...
;;         (right-add ... (sub1 n) ...)...]))
```

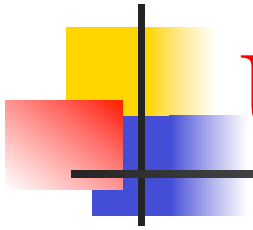
- **Code**

```
(define (right-add m n)
  (cond[(zero? n) m]
  [(positive? n) (add1 (right-add m (sub1 n)))]))
```



Today's Goals

- Loose Ends
  - Catching mistakes and raising errors
  - `and` & `or` operations
- Trees
  - Significantly more expressive type
  - "Lists with many tails"
- Examples:
  - Family tree
  - Binary search tree



Using and & or

- Scheme `and` abbreviates a conditional and takes an arbitrary number of arguments (our student dialects require at least 2)

  `(and arg1 ... argn)` abbreviates

```
(cond [(not arg1) false]
      ...
      [else argn])
```

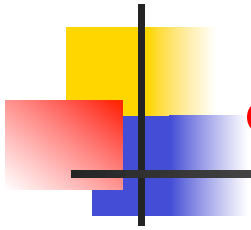
  Hence,

```
(and true true false true (zero? (/ 2 0)) ...)
 => false
```

- This behavior is called "short-circuit" or "non-strict" semantics for `and`
- What does `or` do? It is the "dual" of `and`.

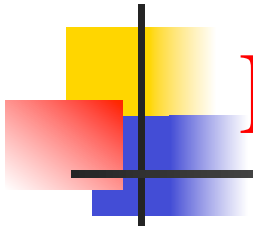
  `(or arg1 ... argn)` abbreviates

```
(cond [arg1 true]
      ...
      [else argn])
```



`and` & `or` cont.

- What are the reduction rules (laws) for `and`?
  - `(and false ... argn) => false`
  - `(and true arg2 ... argn) => (and arg2 ... argn)`
  - `(and v) => v`
- What are the reduction rules (laws) for `or`?
  - `(or true ... argn) => true`
  - `(or false arg2 ... argn) => (or arg2 ... argn)`
  - `(or v) => v`



Error Reporting

- To report an error in Scheme invoke:

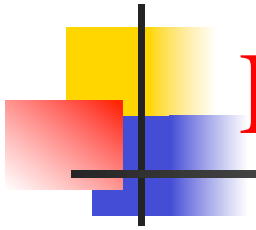
  `(error msg v1 … vn)`

  where `msg` is a string enclosed in quotation marks and `v1, …, vn` are arbitrary Scheme values.

  Semantics: the entire computation is aborted and an error message consisting of `msg v1 … vn` is printed.

- Example:

```
(define (len aloa)
  (cond [(empty? aloa) 0]
        [(cons? aloa) (add1 (len (rest aloa)))]
        [else (error "length: expected <list>; given"
                     aloa)]))
```

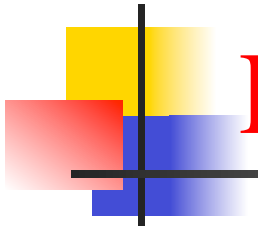


Error Reporting (contd)

- Questions:
  - Is error reporting a good idea.
  - Should error behavior be documented?

- Answers to questions are surprising subtle and lacking in consensus. In the case above, it is probably a good idea but it is often not done because it clutters the code and the error is caught anyway with a slightly less informative diagnostic. (DrScheme libraries do perform such checks.

- Error checking should not be included in a contract (purpose) unless the client code can depend on it and use it (by "catching" the error). We will cover error catching in Java.



Reductions for Errors

- First, note how errors work for functions you already know. In any context, erroneous primitive function applications like `(/ 1 0)` abort the computation and return an error *at the top level:*

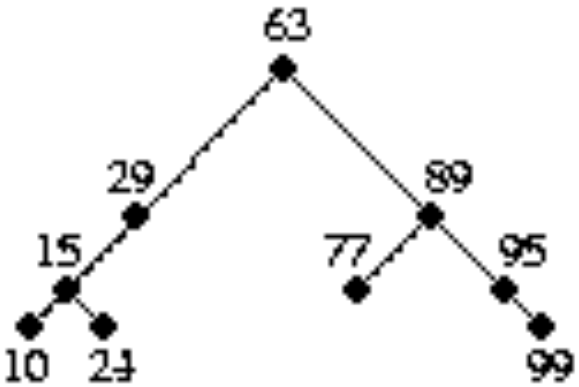
  `(/ 1 0) => /: divide by zero`
  at top level. This is unique among our rules.
- The error construct gives program text access to this mechanism. In any context

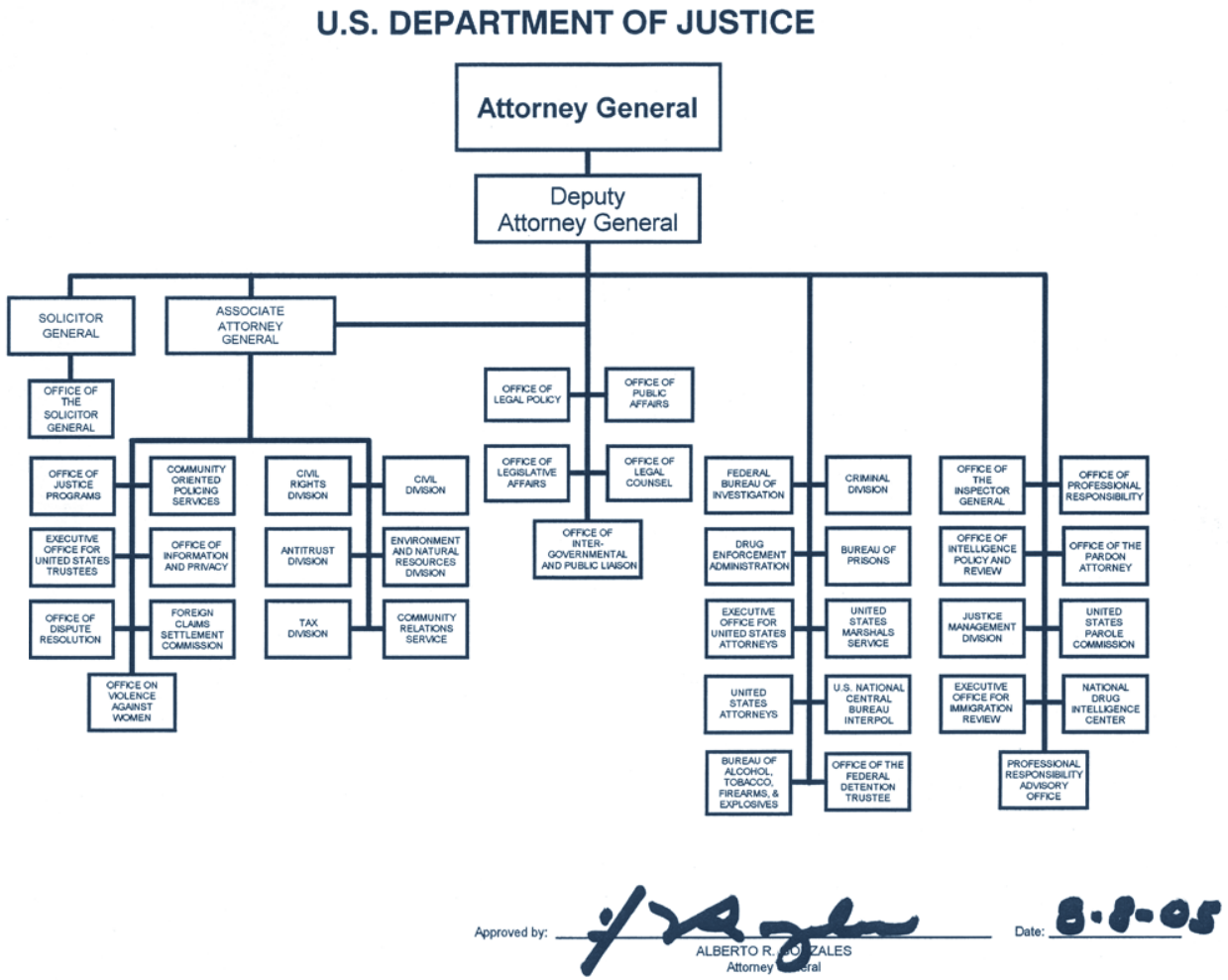
  `(error 'append "first argument not a list") =>`
  `append: first argument not a list` [at top level]
- Use errors **only** as required by the problem or recipe

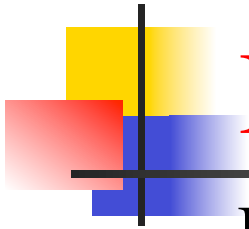
Another Inductive Type: Trees (Structures in Structures)

- Labeled trees
- Organizational charts
- Decision trees
- Search trees

and many more!





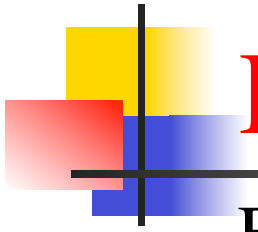


From Lists to Trees

Example of a List Data Definition

```
; A list-of-symbols is
;       empty, or
;       (cons s los)
; where s is a symbol and los is a list-of-symbols
```

A list has one embedded structure (rest)



Family Trees

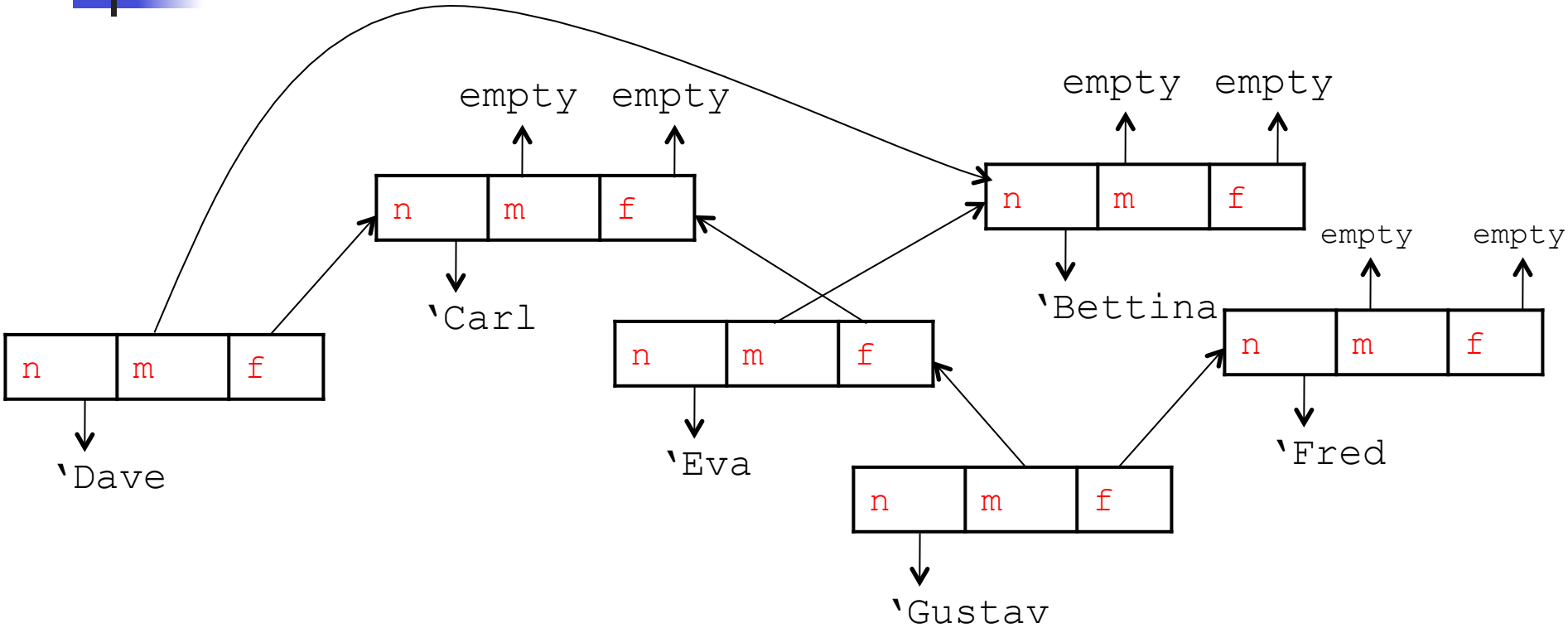
Example of a Family Tree Data Definition

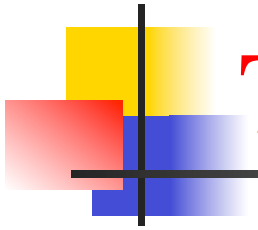
```
; A child is
;      empty                          // Represents "unknown"
;      (make-child n m f)             // Two self-references
; where n is a symbol, m is a child and f is a child

(define-struct child (name mother father))
```

A child has two embedded structures (mother, father)

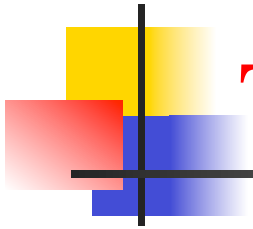
Example Family Tree (variant of Figure 35 in textbook)





Tree Depth (in class ex.)

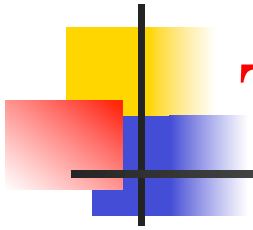
- Consider the following problem
  - "Given an ancestry tree, compute the maximum number of generations for which we know something about this person."
- Contract (or "type") is
  - `child -> natural`
- Examples (next slide)



Tree Depth Examples

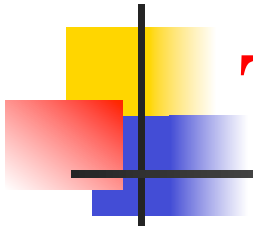
```
(define cat (make-child 'Cat empty empty))
(define tom (make-child 'Tom cat empty))
(define jane (make-child empty tom))
(define johnny (make-child 'Johnny empty empty))
(define ray (make-child 'Ray empty johnny))
(define sue (make-child 'Sue empty ray))
(define rob (make-child 'Rob empty sue))
(define bob (make-child 'Bob jane rob))

;(max-depth cat) = 1
;(max-depth tom) = 2
;(max-depth jane) = 3
;(max-depth johnny) = 1
;(max-depth ray) = 2
;(max-depth sue) = 3
;(max-depth rob) = 4
;(max-depth bob) = 5
```



Tree Depth Template Instantiation

```
max-depth : child -> natural
(define (max-depth c)
 (cond
   [(empty? c) ...]
   [else ...
     ... (max-depth (child-mother c)) ...
     ... (max-depth (child-father c)) ...)))
```



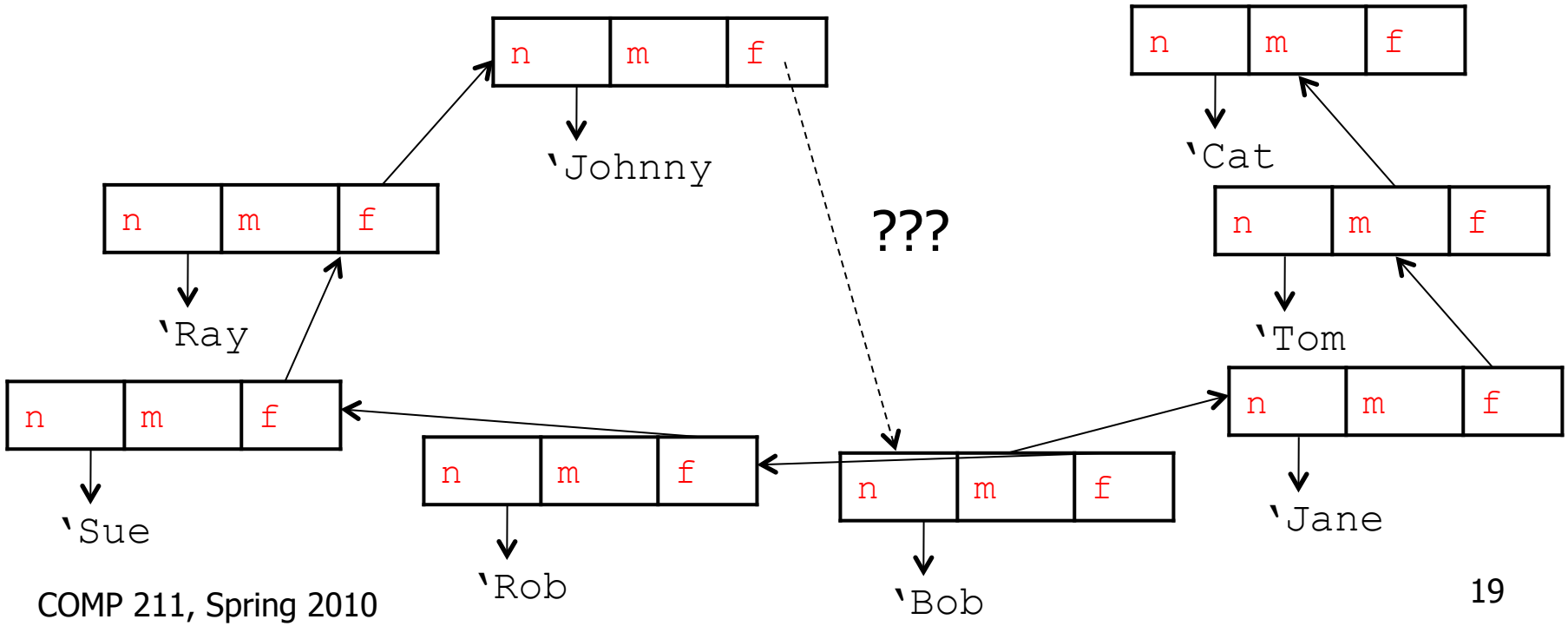
Tree Depth Code

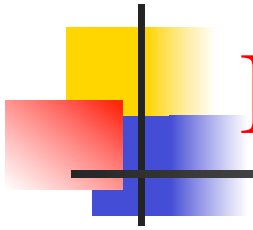
```
max-depth : child -> natural
(define (max-depth c)
 (cond
   [(empty? c) 0]
   [else (add1
    (max (max-depth (child-mother c))
         (max-depth (child-father c))))]))
```

Examples can help in writing code.

Challenge Problem

- Can you think of a Scheme program that can create a cycle among structures?





Binary Search Trees

```
;  A binary-search-tree (BST) is either
;     empty, or
;     (make-node n l r)
;  where n is a number, l and r are BTs.
;  Invariants:
;       1. Numbers in l are less than or equal to n
;       2. Numbers in r are greater than n
(define-struct node (num left right))
```

Binary Search Trees

Which tree satisfies the invariants of a binary search tree?