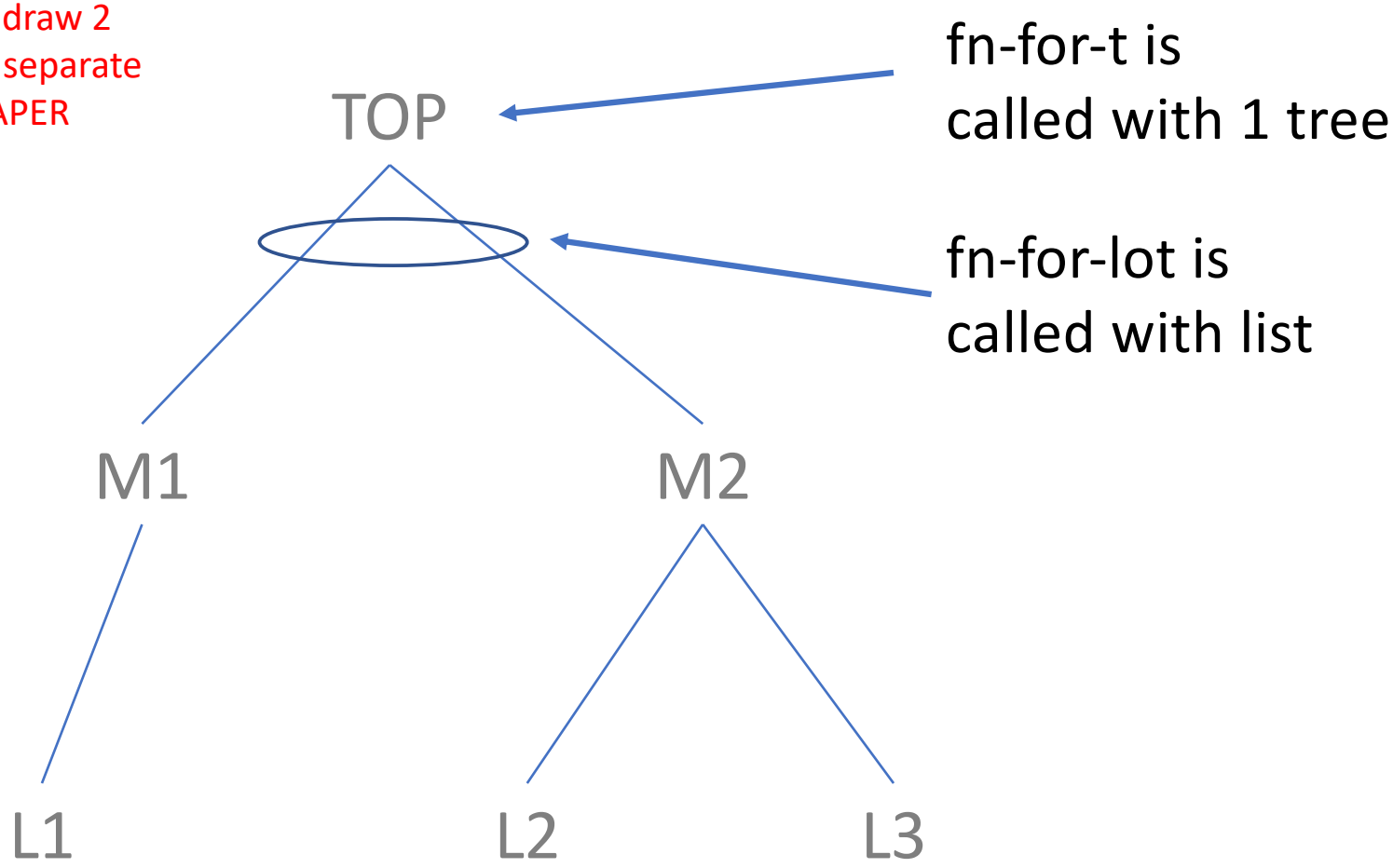
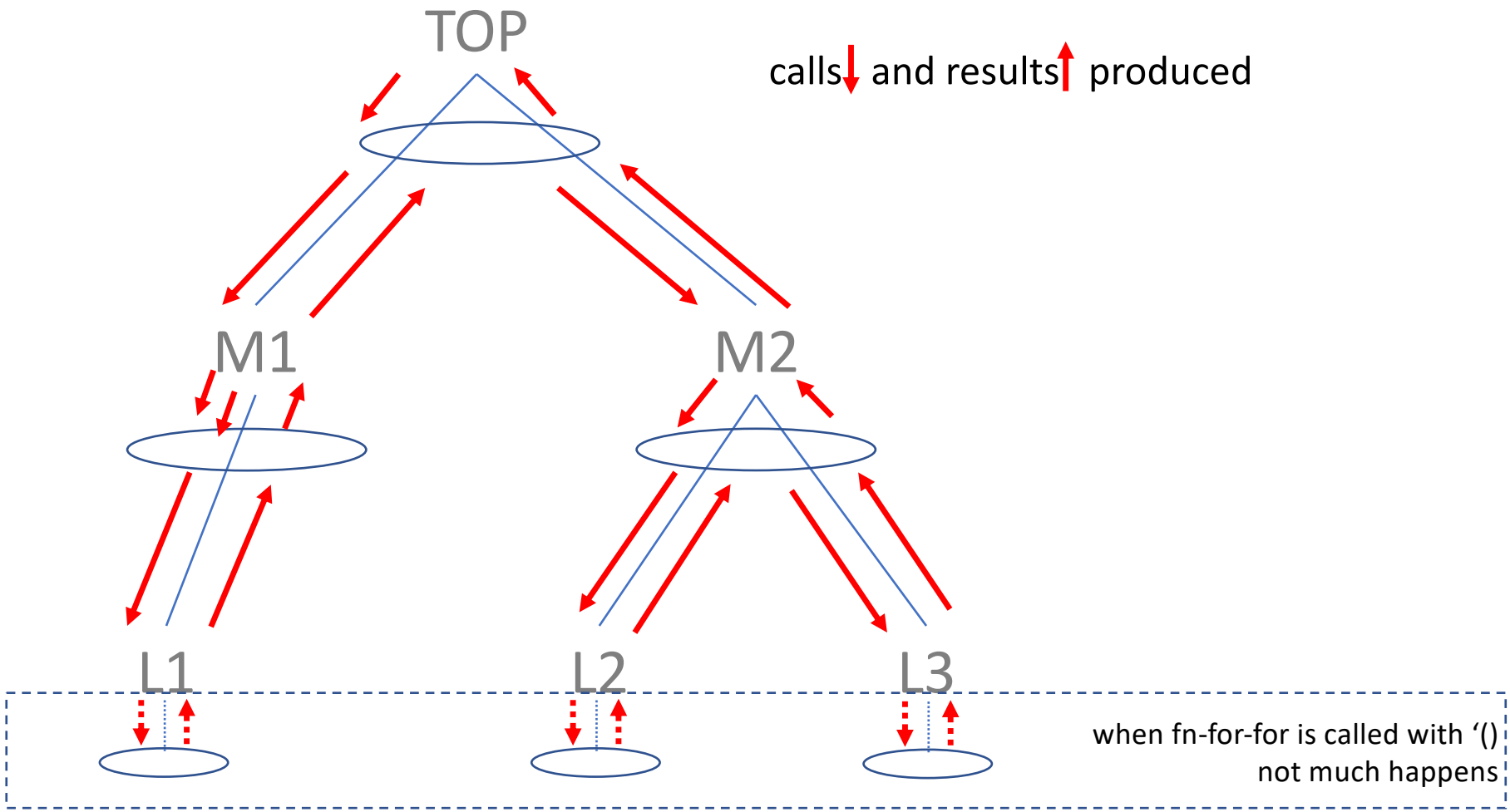
Lecture 19

While waiting, draw 2 copies of this on separate sheets on PAPER



typical structural recursion

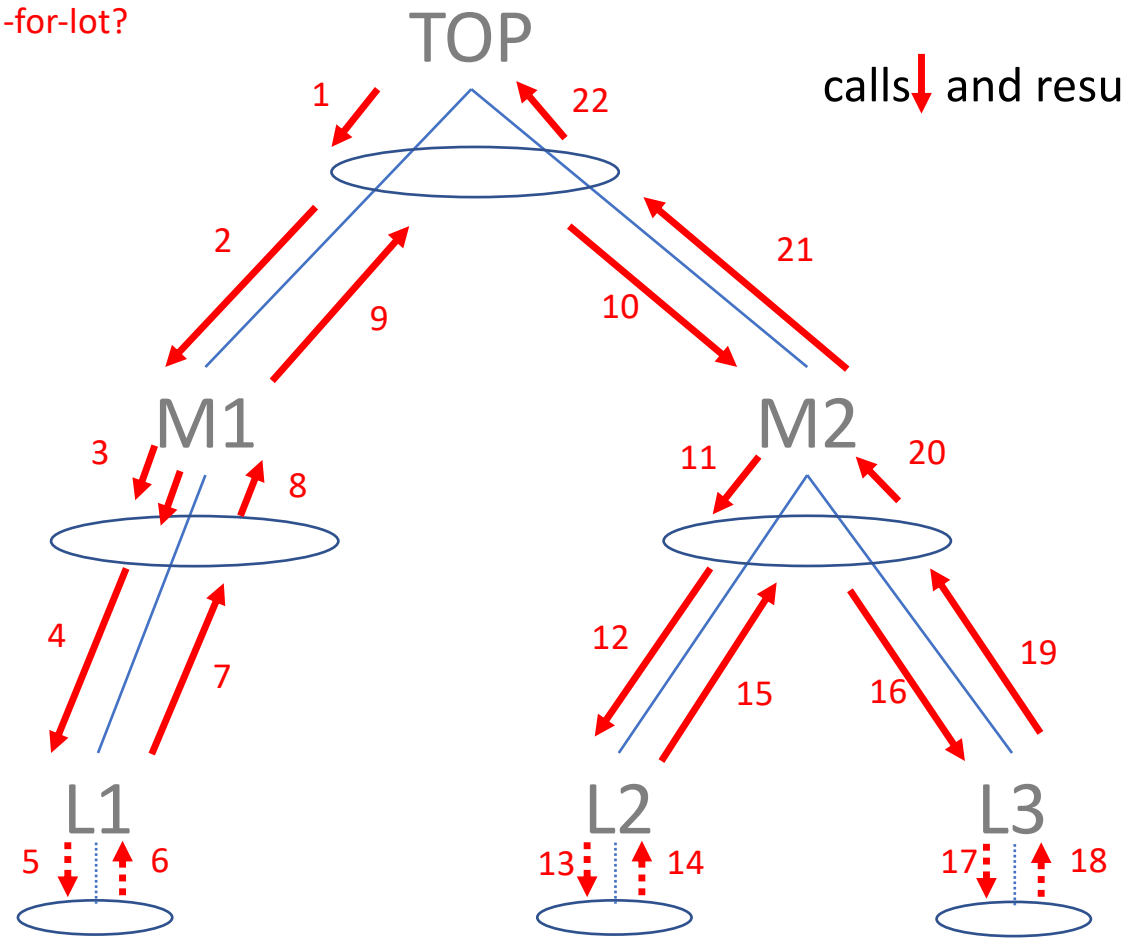
calls↓ and results↑ produced



What is value of path at each call to fn-for-lot?

typical structural recursion

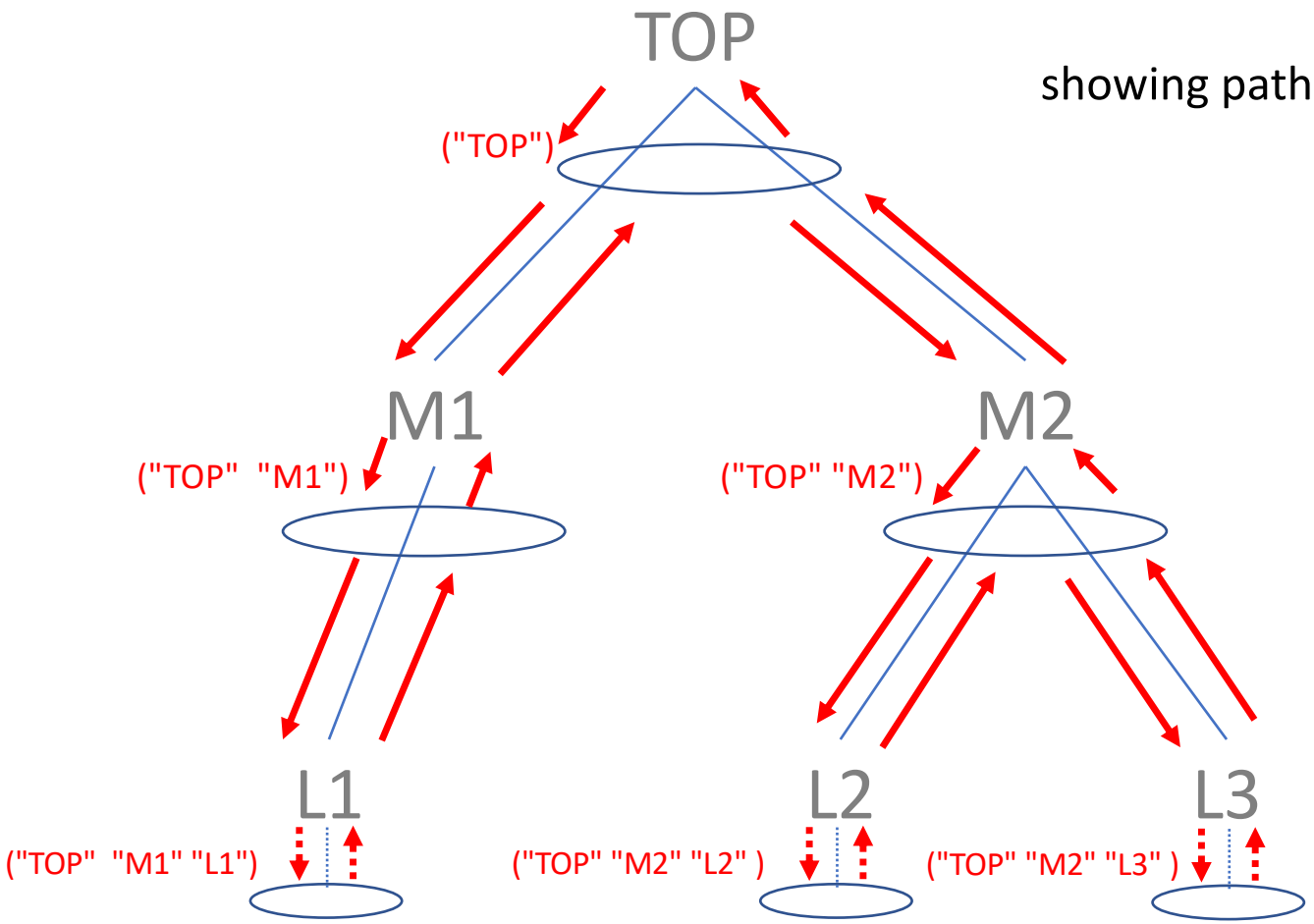
calls↓ and results↑ produced



path; names of parents to here

With path accumulator

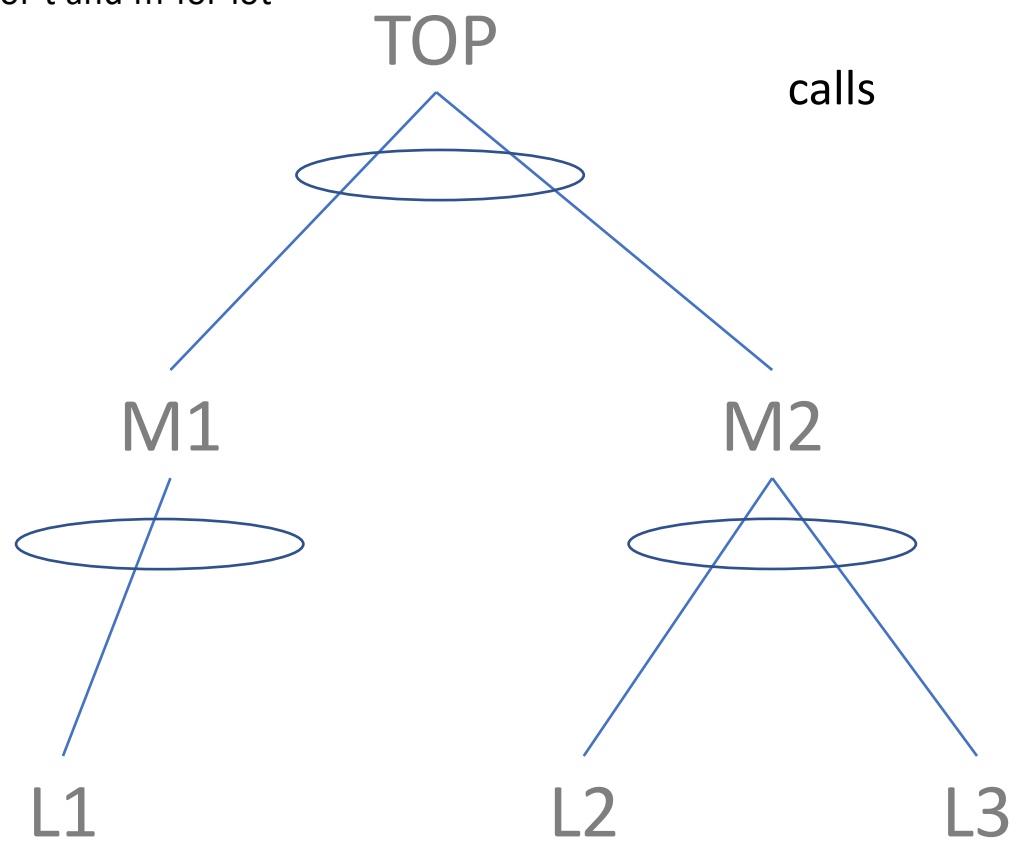
showing path at each call to fn-for-lot



tree worklist at fn-for-t and fn-for-lot

tail recursion with worklist

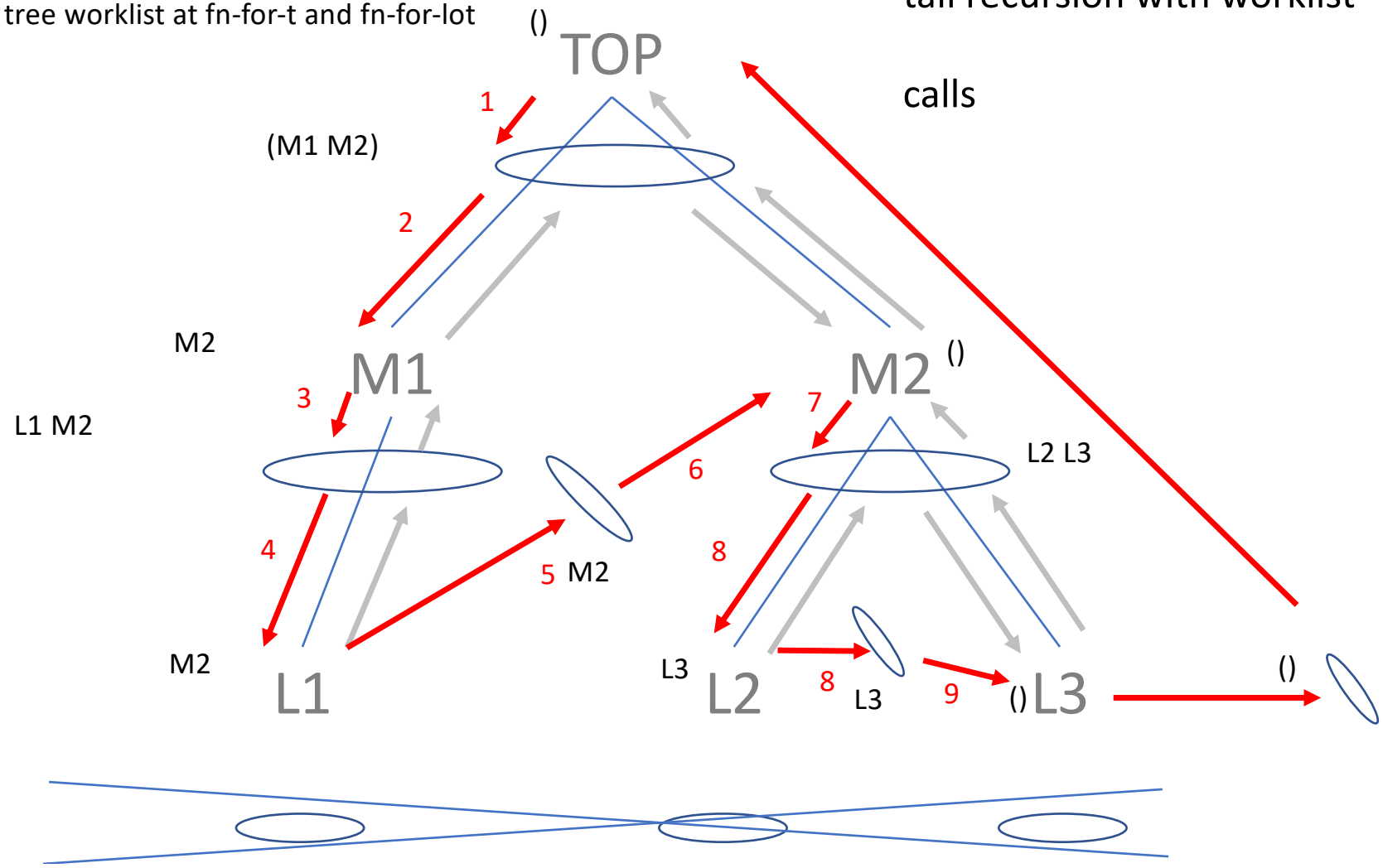
calls



tree worklist at fn-for-t and fn-for-lot

tail recursion with worklist

calls



arb arity tree structural recursion templates

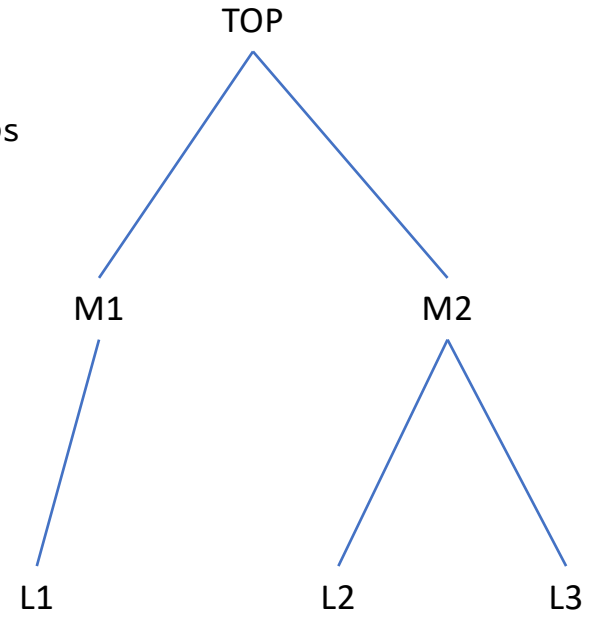
```
(define-struct node (name subs))
;; Tree is (make-node String (listof Tree))
;; interp. a bare bones arbitrary arity tree, each node has a name and subs


(define (fn-for-tree t)
  (local [(define (fn-for-t t)
            (local [(define name (node-name t))  ;unpack the fields
                    (define subs (node-subs t))] ;for convenience

              (... name (fn-for-lot subs))))

          (define (fn-for-lot lot)
            (cond [(empty? lot) (...)]
                  [else
                   (... (fn-for-t (first lot))
                        (fn-for-lot (rest lot)))]))]

    (fn-for-t t)))
```



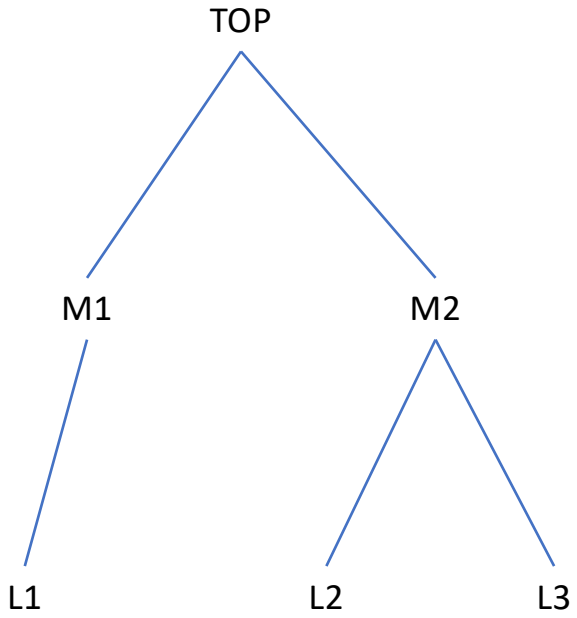
structural recursion with path accumulator.

```
(@template Tree (listof Tree) accumulator)

(define (find-path t n)
  ;; path is (listof String); names of ... grandparent, parent trees
  ;;                          (builds along recursive  calls)
  (local [(define (fn-for-t t path)
            (local [(define name (node-name t))
                    (define subs (node-subs t))
                    (define npath (append path (list name)))]
              (if (string=? name n)
                  npath
                  (fn-for-lot subs npath))))


          (define (fn-for-lot lot path)
            (cond [(empty? lot) false]
                  [else
                   (local [(define try (fn-for-t (first lot) path))]
                     (if (not (false? try))
                         try
                         (fn-for-lot (rest lot) path)))]))]

    (fn-for-t t empty)))
```



tail recursion with worklist

```
(@template backtracking Tree (listof Tree) accumulator)


;; Tail recursion
(define (find-tree t to)
  ;; todo is (listof Tree); worklist of pending trees to visit
  (local [(define (fn-for-t t t-wl)
            (local [(define name (node-name t))  ;unpack the fields
                    (define subs (node-subs t))] ;for convenience
              (if (string=? name to)
                  t
                  (fn-for-lot (append subs t-wl)))))


          (define (fn-for-lot t-wl)
            (cond [(empty? t-wl) false]
                  [else
                   (fn-for-t (first t-wl)
                             (rest t-wl))]))]

    (fn-for-t t empty)))
```