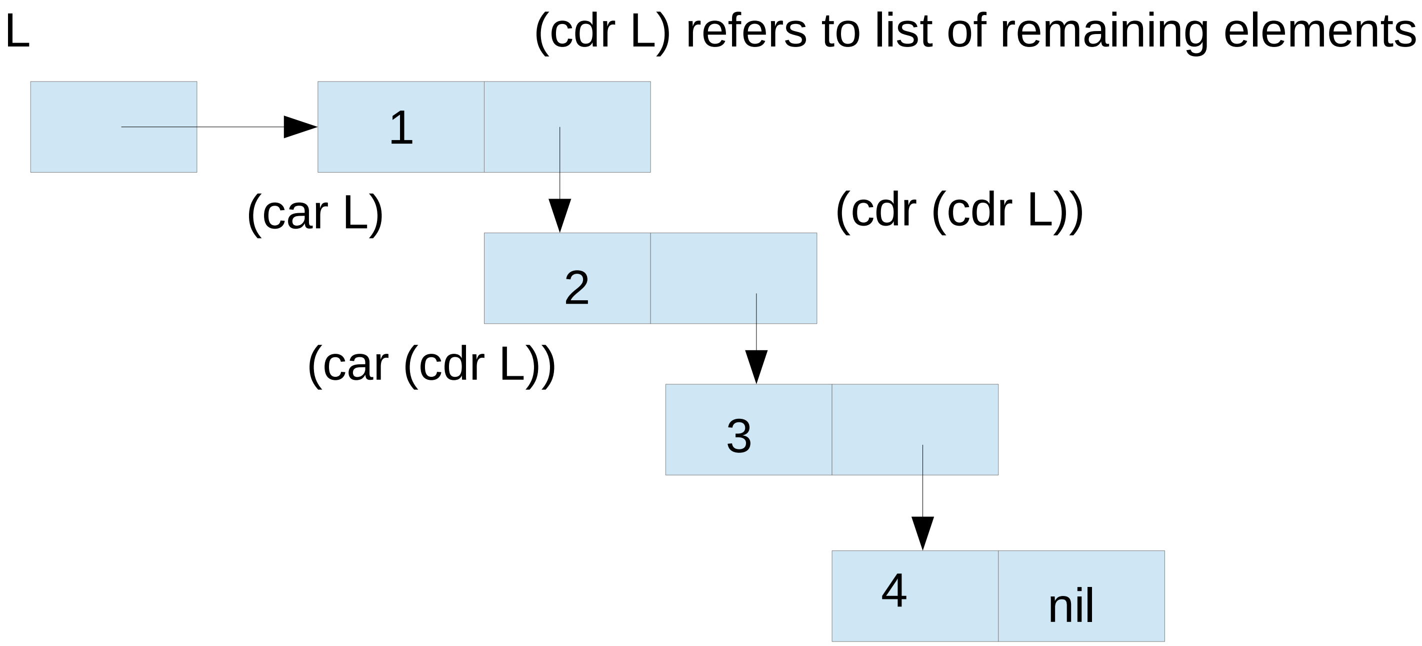
Lists: implementation/implications

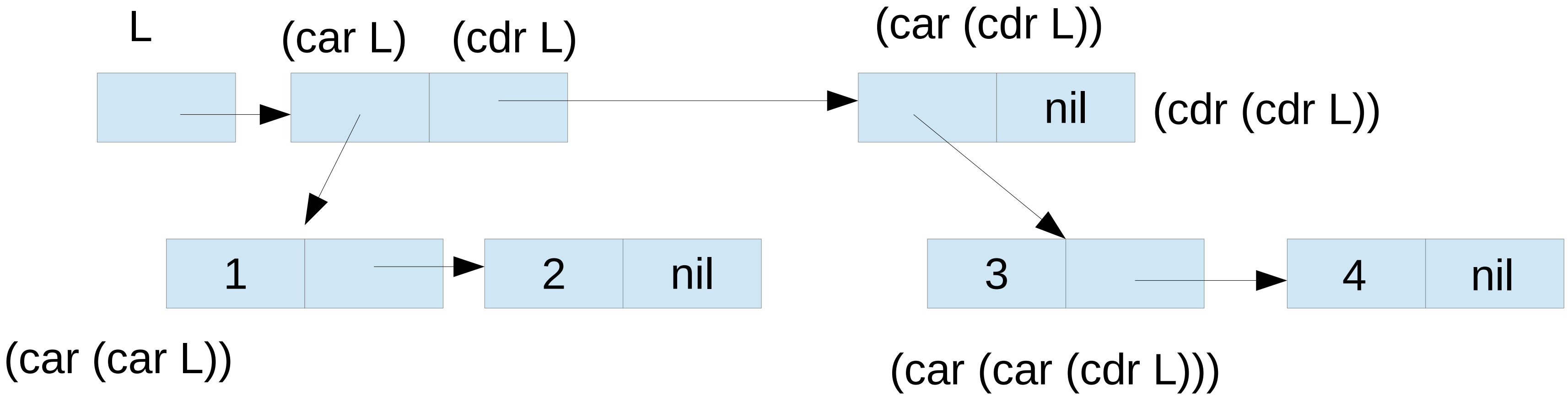
- For primitive data types (e.g. characters, integers, reals, booleans), items can be held in a simple 32- or 64-bit cell
- For lists, however, lisp adopts a linked-list approach, where it stores a pointer to the front of the list (nil if the list is empty)
- Each list element is represented as two parts: the value of that element (accessible through car) and a pointer to the next element (accessible through cdr)
- If a list element is itself a list, then the “value of the element” would be a pointer to the front element of that list

Pointer-based representation

- Consider (defvar L ‘(1 2 3 4))



Nested lists

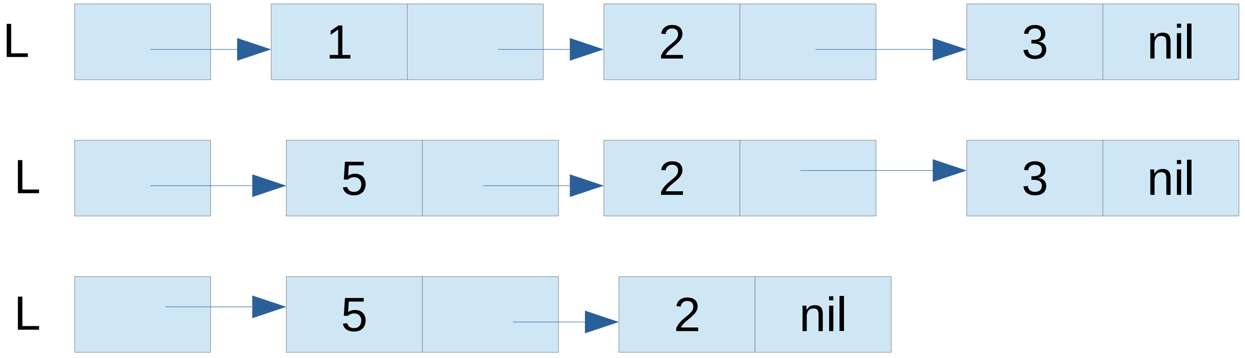


- (defvar L ‘((1 2) (3 4)))

setf on car, cdr

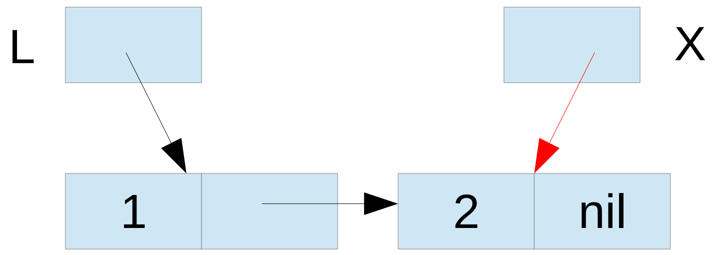
- (car X) and (cdr X) can be altered with setf

```
(defvar L ‘(1 2 3))
(setf (car L) 5)
(setf (cdr (cdr L)) nil)
```



Shallow copy of a list

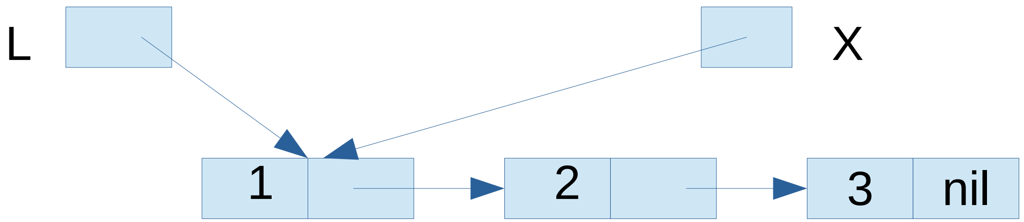
- (defvar L ‘(1 2)) (defvar X L)



- (setf (car X) 10) changes front element to 10 for L as well, since L and X really refer to the same internal list
- (setf X 10) changes X to 10, has no effect on list L refers to

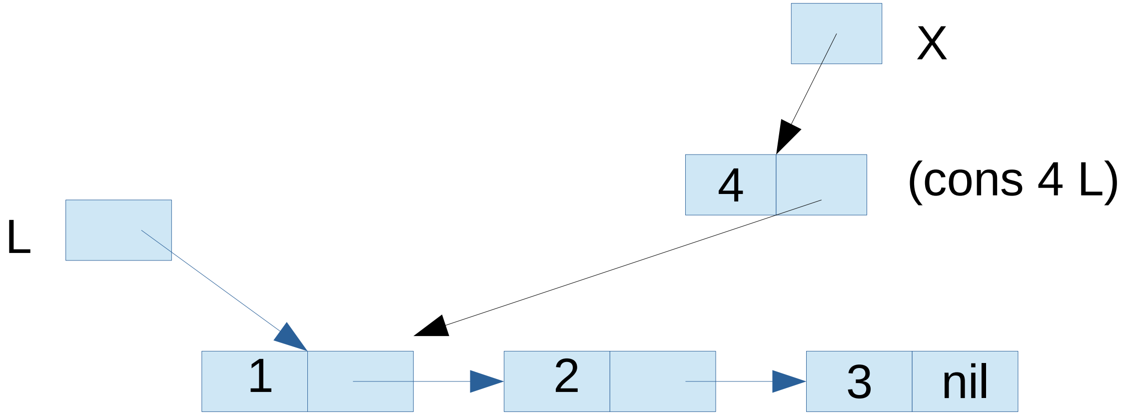
Passing a list to a function

```
(defvar L '(1 2 3))
(defun f ( X ) (setf X 10))
(defun g ( X ) (setf (car X) 10))
(f L) ; no effect
(g L) ; changes first element to 10
```



(cons e L)

```
(defvar L '(1 2 3))
(defun X (cons 4 L))
```



Circular lists

- we can create circular lists

```
(defvar L '(10 20 30))
(setf (cdddr L) L)   ; make end of L point to front of L
(nth 3 L)            ; returns 10
(nth 4 L)            ; returns 20, etc
(format "~A~%" L)    ; goes into infinite loop
```

- Can turn on cycle-detection so it doesn't infinite loop

```
(let ((*print-circle* t)) (format t "~A~%" L))
; actually prints "#0=(10 20 30 . #0#)
```