

## STRUKTUR DATA

# Double Linked List

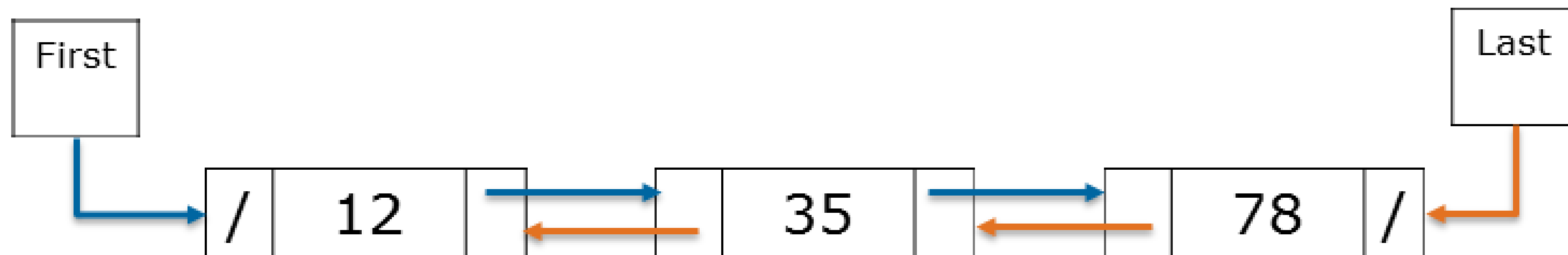
Rita Rismala, Dade Nurjanah,  
Febryanti Sthevanie, Bambang Ari Wahyudi

Prodi Informatika – Fakultas Informatika



## DOUBLE LINKED LIST

- Linked list dengan dua pointer yang menghubungkan sebuah elemen dengan elemen sebelumnya dan setelahnya.



## STRUKTUR

- Setiap elemen dibagi menjadi tiga bagian.



- Biasanya memiliki dua pointer kepala.



## ADT DOUBLE LINKED LIST

```
Type infotype : integer
Type address : pointer to ElmList
```

```
Type ElmList <
  info : infotype
  next : address
  prev : address
```

```
>
```



Terdapat dua pointer untuk menunjuk ke elemen sebelumnya dan setelahnya.

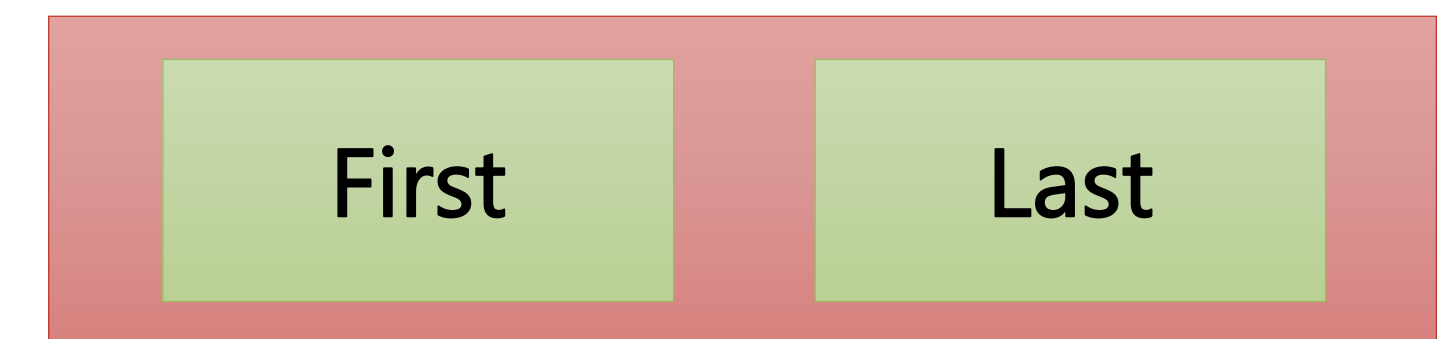


## ADT DOUBLE LINKED LIST

Type List : <  
    First : address  
    Last : address  
>

Kamus

L : List





L

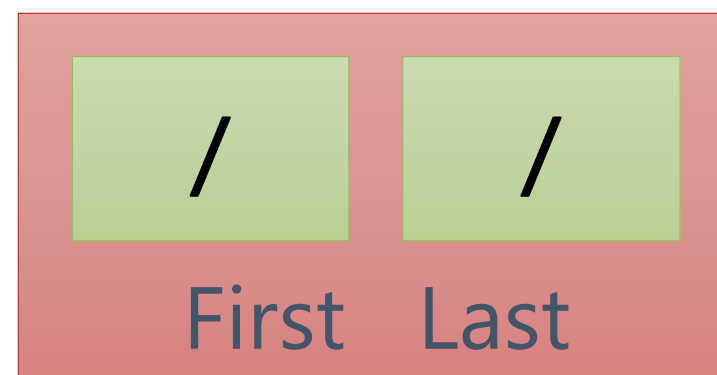
Terdapat dua pointer kepala yang menunjuk elemen pertama dan terakhir pada list.

A cluster of red triangles of various sizes and orientations in the top-left corner.

## PRIMITIF-PRIMITIF DASAR

- 
- 
- Pembuatan list baru
  - Pembuatan elemen baru
  - Penyisipan elemen
  - Penghapusan elemen

## PEMBUATAN LIST BARU



$L$

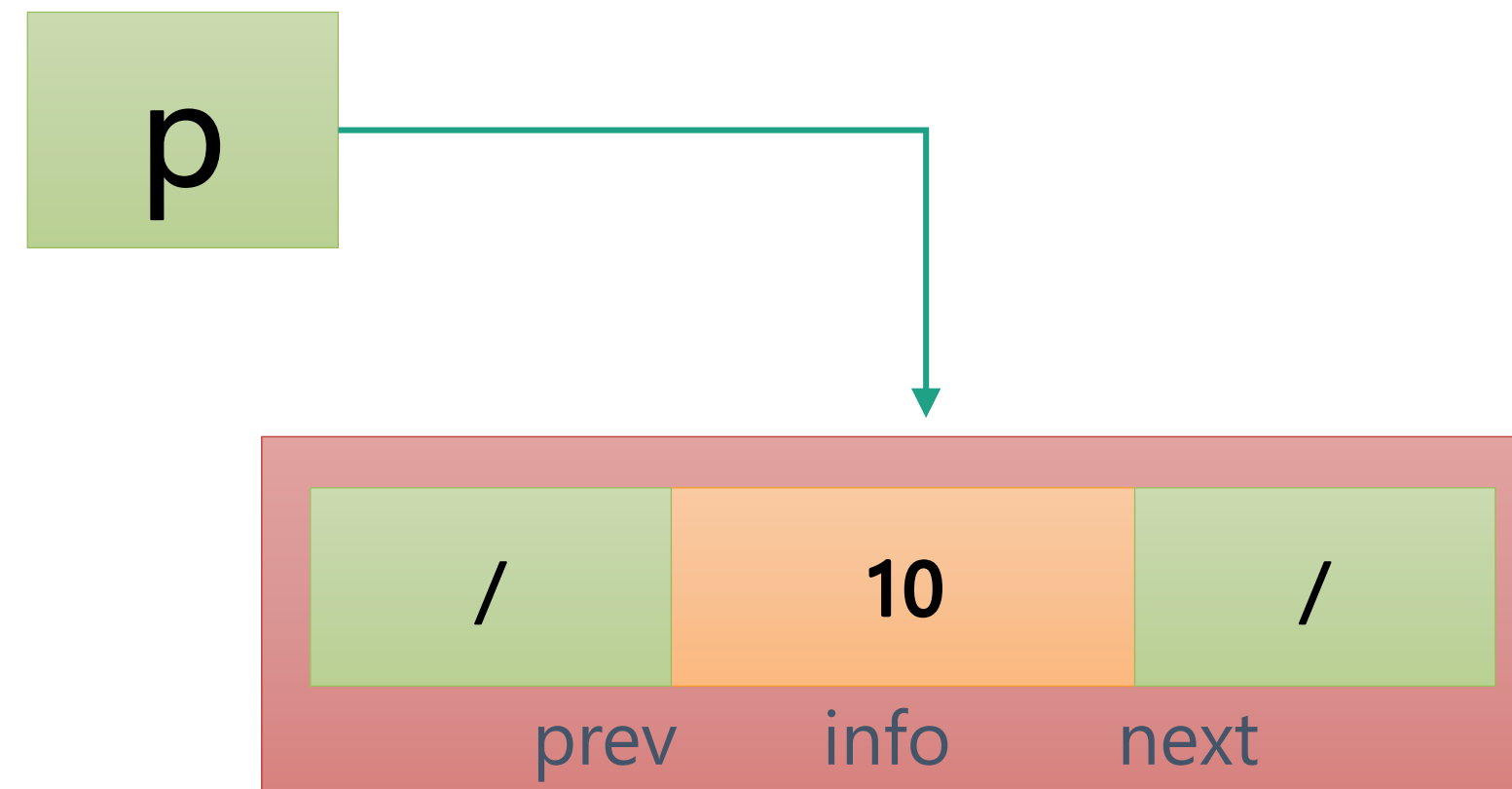
### Algoritma

$First(L) \leftarrow Nil$

$Last(L) \leftarrow Nil$

- **First(X)** merupakan kata kunci untuk menyatakan pointer yang menunjuk ke elemen pertama dari list X.
- **Last(X)** merupakan kata kunci untuk menyatakan pointer yang menunjuk ke elemen terakhir dari list X.
- List yang baru dibuat belum memiliki elemen, oleh karena itu  $First(L)$  dan  $Last(L)$  adalah Nil / Null.

## PEMBUATAN ELEMEN BARU






### Algoritma

```
Alokasi(p)
info(p) ← 10
next(p) ← Nil
prev(p) ← Nil
```

- **next(y)** merupakan kata kunci untuk menyatakan pointer yang menunjuk ke elemen setelah elemen y.
- **prev(y)** merupakan kata kunci untuk menyatakan pointer yang menunjuk ke elemen sebelum elemen y.
- **info(y)** merupakan kata kunci untuk menyatakan info dari elemen y.
- Elemen yang baru dibuat belum terhubung ke elemen yang lain, oleh karena itu next dan prev-nya adalah Nil.



## KATA KUNCI

- 
- A cluster of overlapping red triangles in various sizes and shades of red, located in the top-left corner of the slide.
- 
- A small red triangle pointing to the right, used as a bullet point.
- 
- A small red triangle pointing to the right, used as a bullet point.
- First(X)
  - Last(X)
  - next(y)
  - prev(y)
  - info(y)

## PENYISIPAN ELEMEN BARU

### Insert first

Menyisipkan elemen baru sebagai elemen pertama.

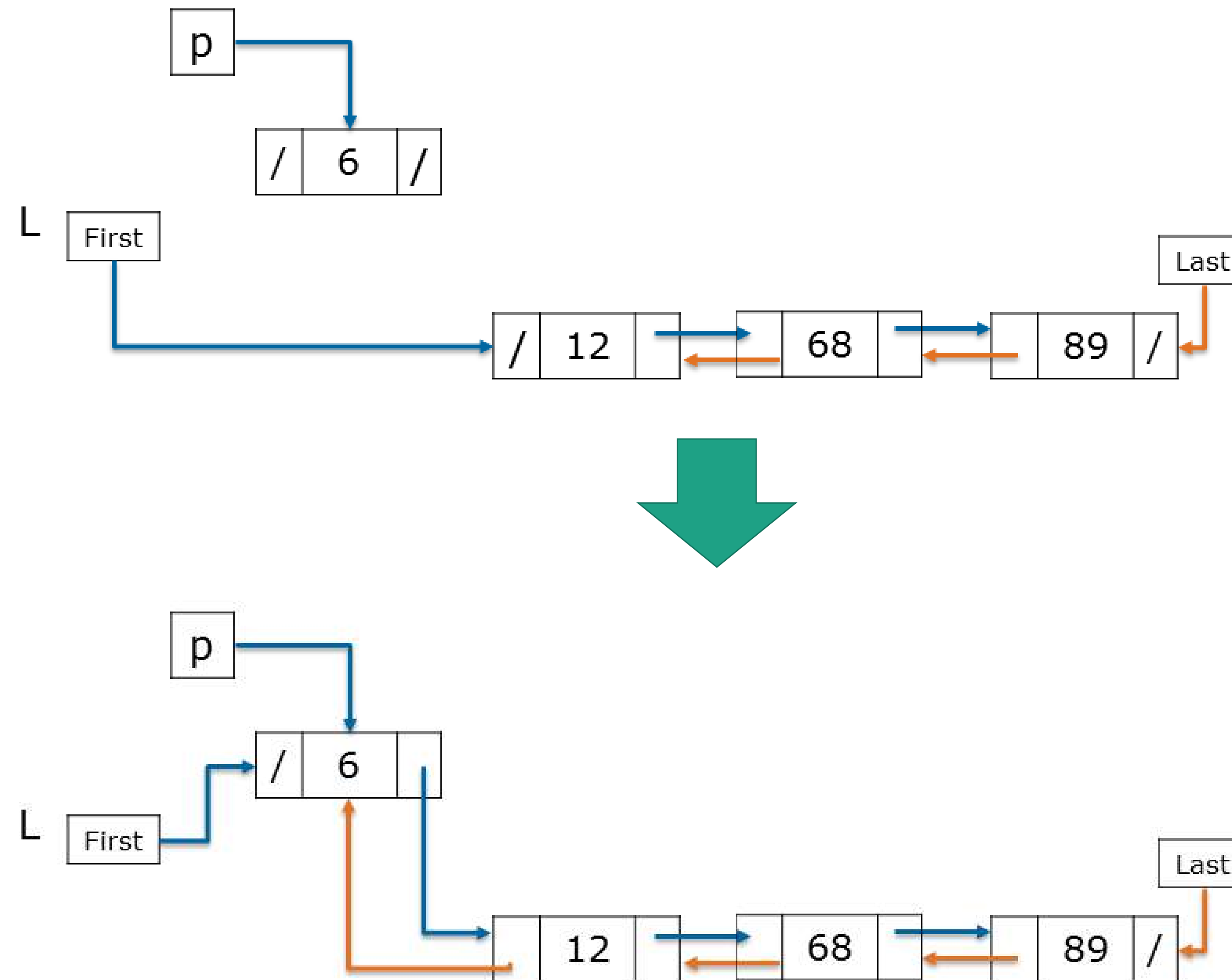
### Insert last

Menyisipkan elemen baru sebagai elemen terakhir.

### Insert after/ before

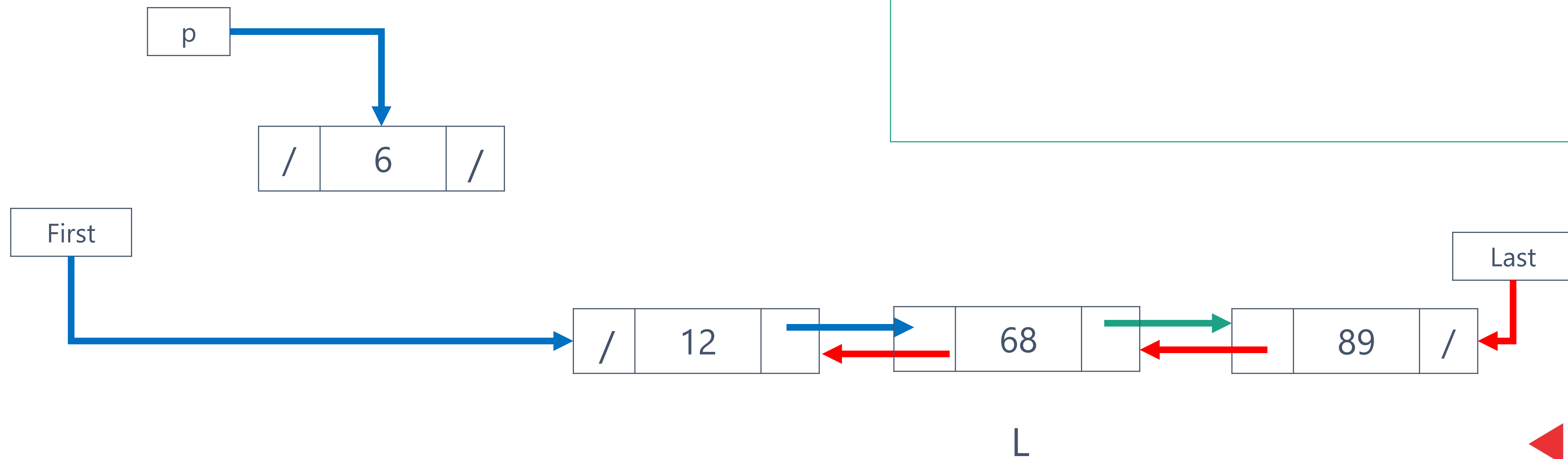
Menyisipkan elemen baru setelah atau sebelum elemen yang lain.

## INSERT FIRST



# INSERT FIRST

**INITIAL STATE**

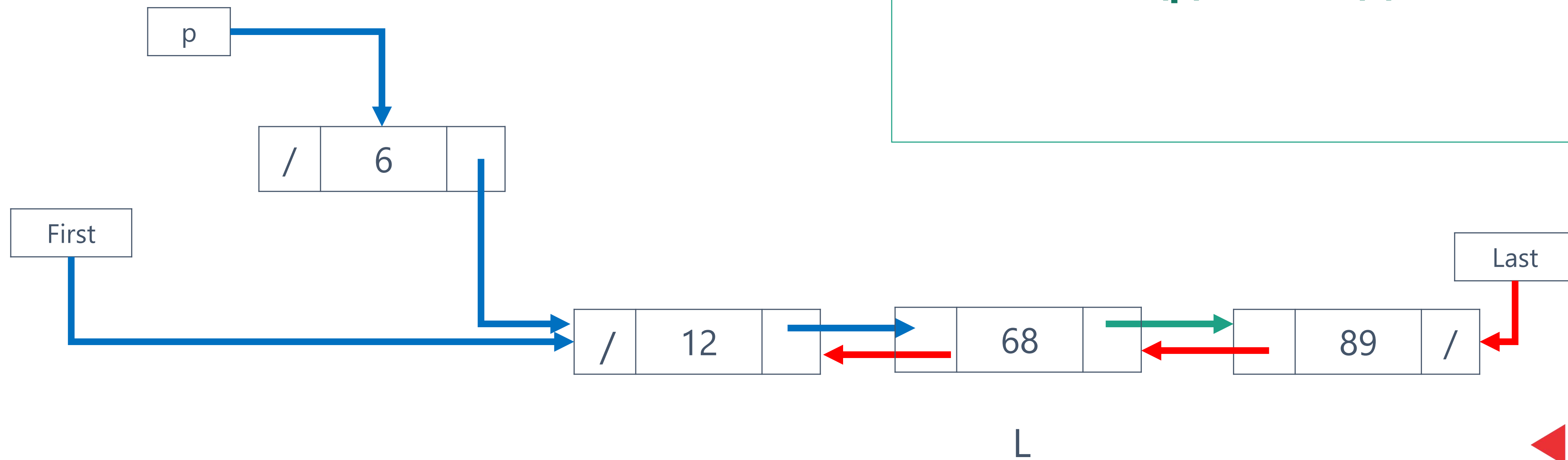


Algoritma

# INSERT FIRST

Algoritma

**$\text{next}(p) \leftarrow \text{First}(L)$**



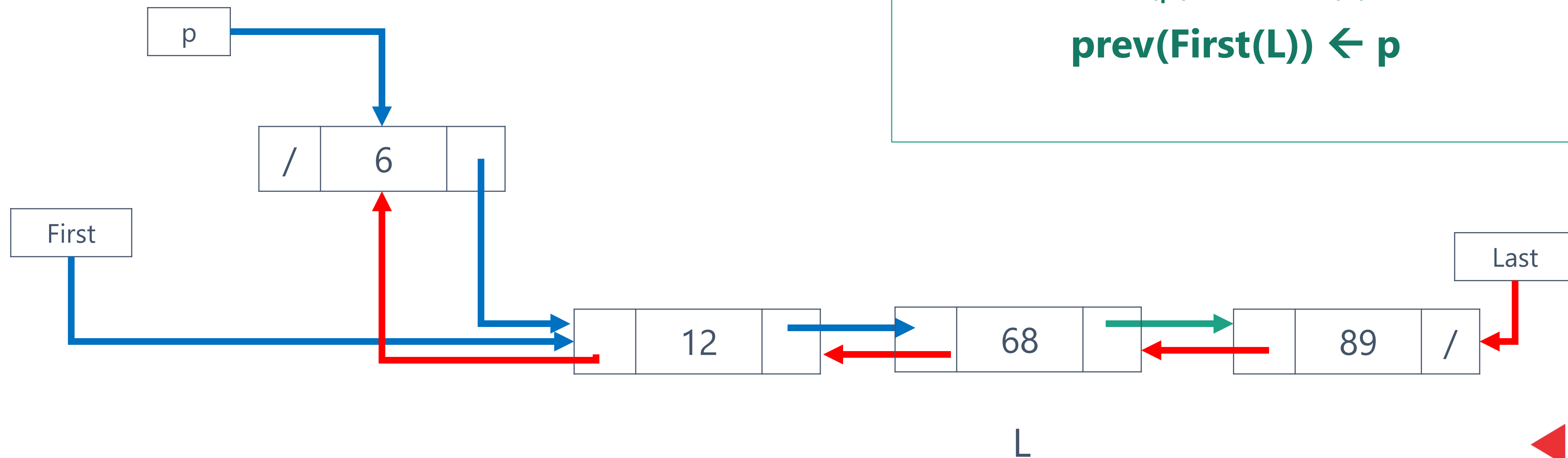


# INSERT FIRST

## Algoritma

$\text{next}(p) \leftarrow \text{First}(L)$

**$\text{prev}(\text{First}(L)) \leftarrow p$**



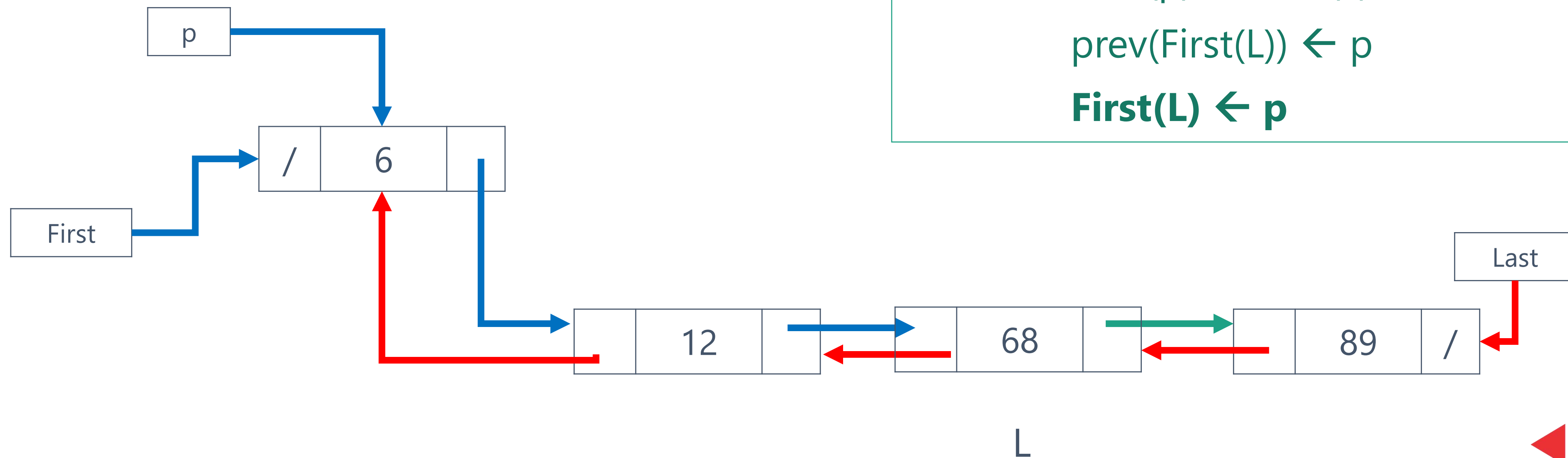
# INSERT FIRST

## Algoritma

$\text{next}(p) \leftarrow \text{First}(L)$

$\text{prev}(\text{First}(L)) \leftarrow p$

**$\text{First}(L) \leftarrow p$**



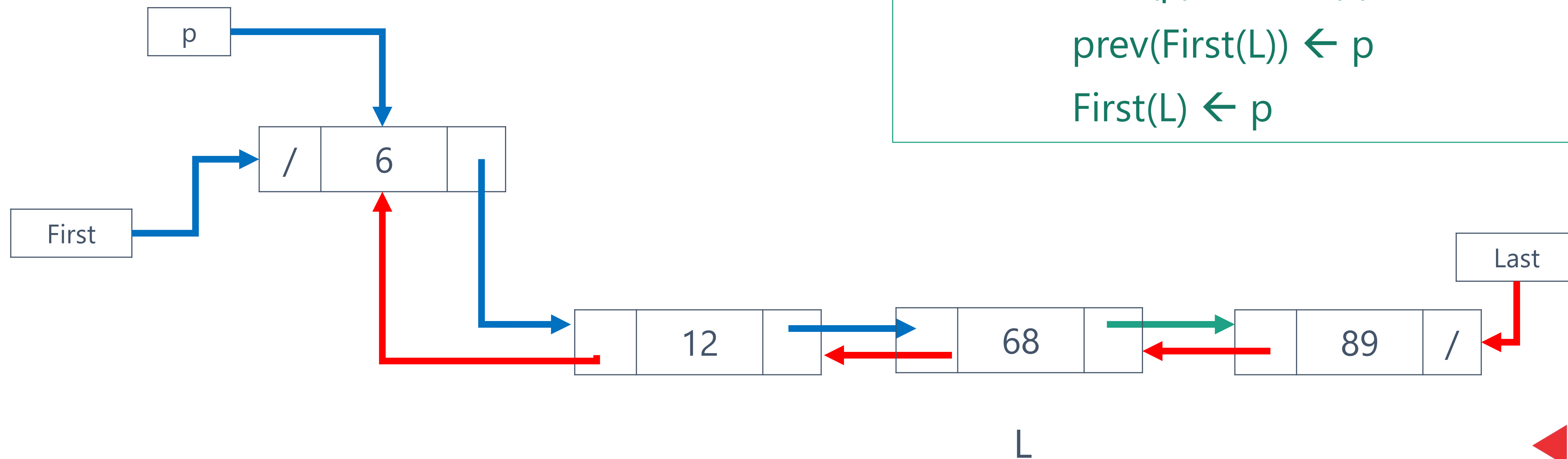
# INSERT FIRST

## Algoritma

$\text{next}(p) \leftarrow \text{First}(L)$

$\text{prev}(\text{First}(L)) \leftarrow p$

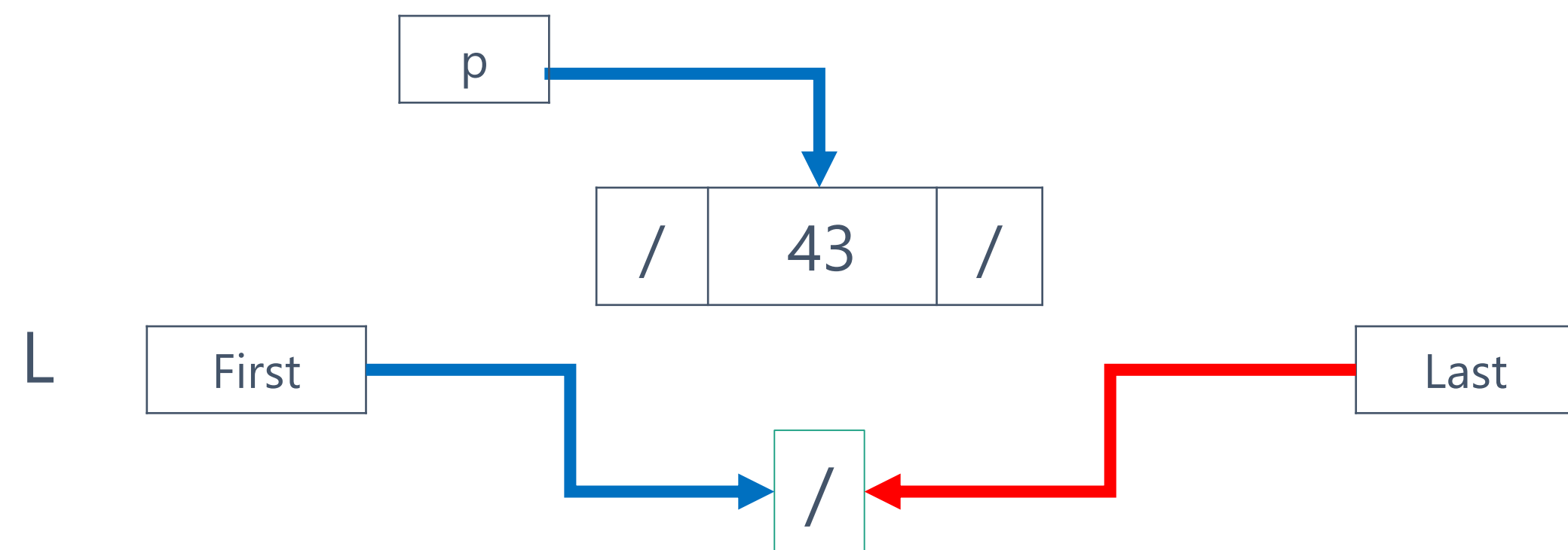
$\text{First}(L) \leftarrow p$



**BAGAIMANA JIKA LIST AWALNYA KOSONG?**

# INSERT PADA LIST KOSONG

## INITIAL STATE



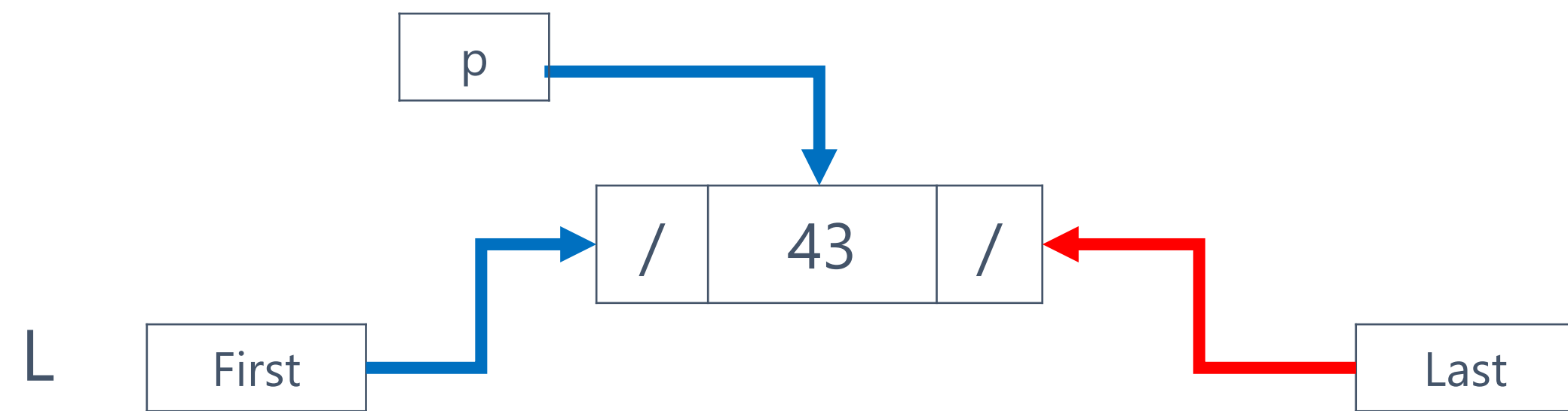
## Algoritma

$\text{next}(p) \leftarrow \text{First}(L)$

$\text{prev}(\text{First}(L)) \leftarrow p$

**ERROR**

## INSERT PADA LIST KOSONG



### Algoritma

**First(L)  $\leftarrow$  p**

**Last(L)  $\leftarrow$  p**



## INSERT FIRST

**Procedure** InsertFirst (In p: address, In/Out L: List)

*{ IS: p adalah elemen baru,  $p \neq \text{nil}$ . List L mungkin kosong.*

*FS: Elemen p menjadi elemen pertama dari list L. }*

**Kamus**

**Algoritma**

if (First(L)  $\neq$  Nil and Last(L)  $\neq$  Nil ) then *{jika list awal tidak kosong}*

next(p)  $\leftarrow$  First(L)

prev(First(L))  $\leftarrow$  p

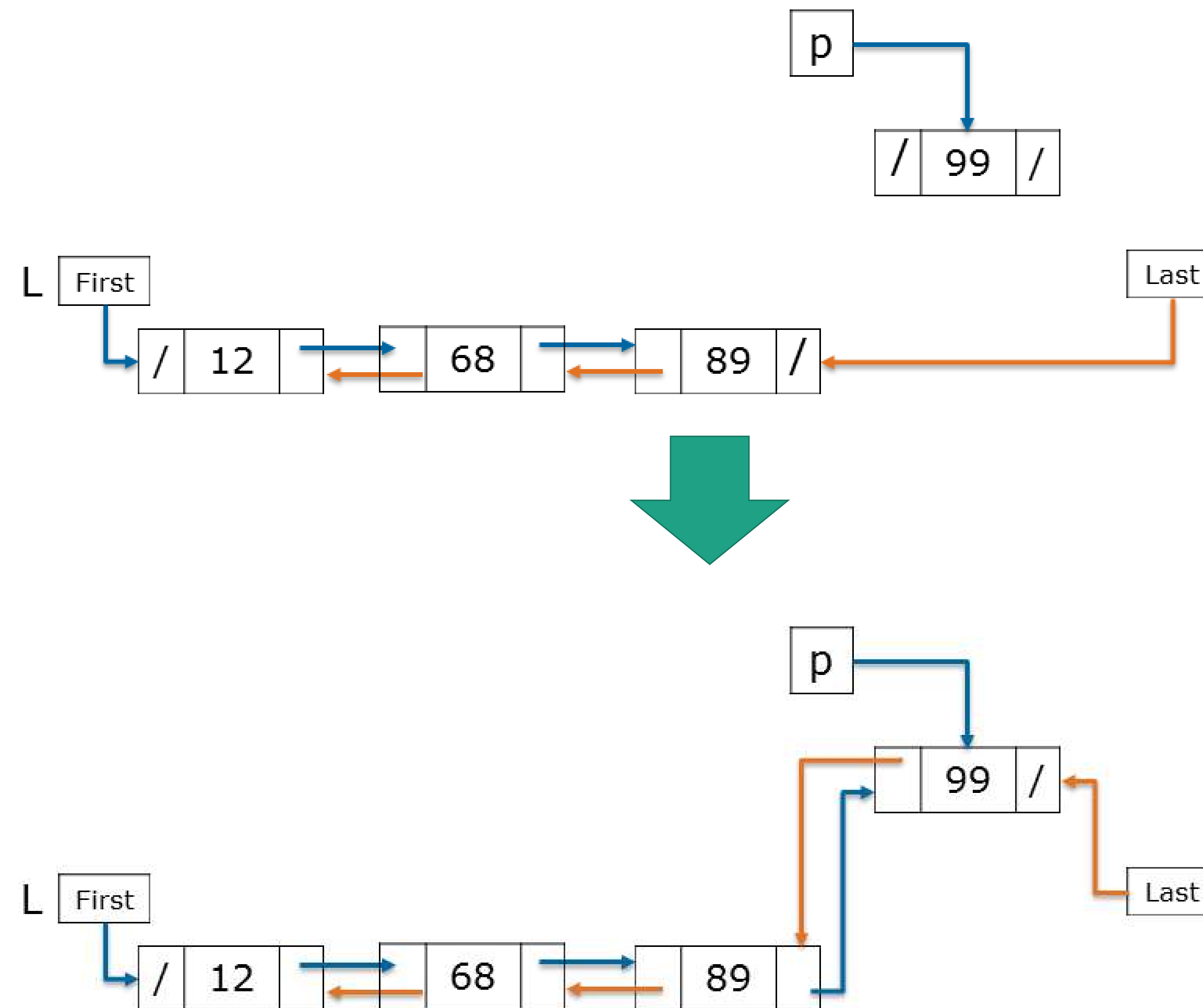
First(L)  $\leftarrow$  p

else *{jika list awal kosong}*

First(L)  $\leftarrow$  p

Last(L)  $\leftarrow$  p

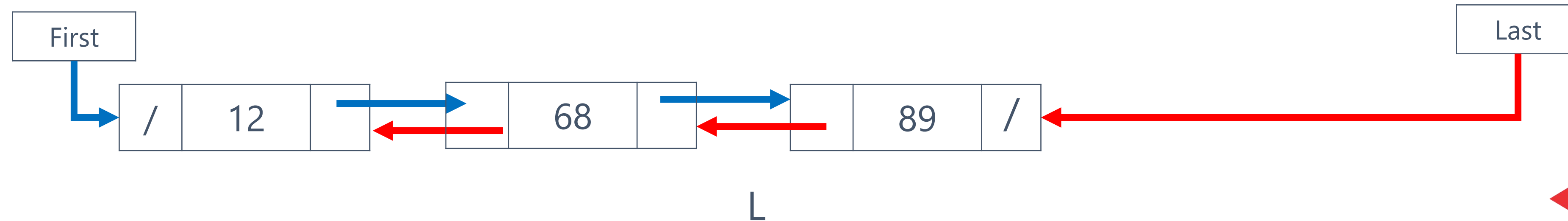
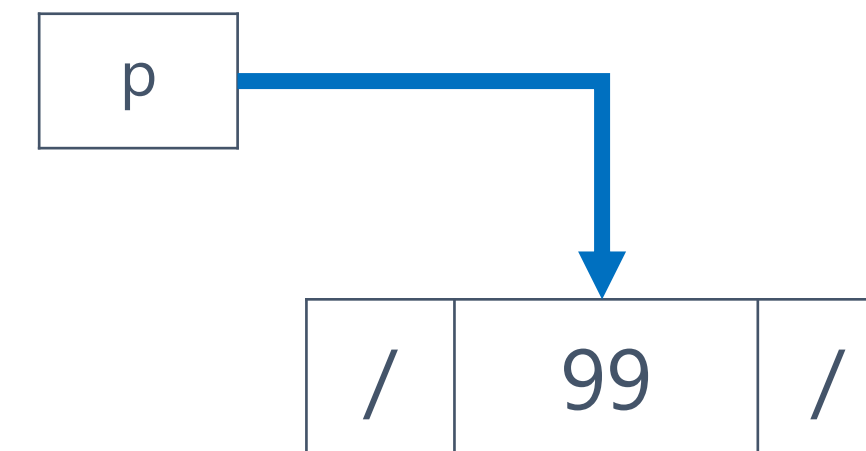
# INSERT LAST



# INSERT LAST

**INITIAL STATE**

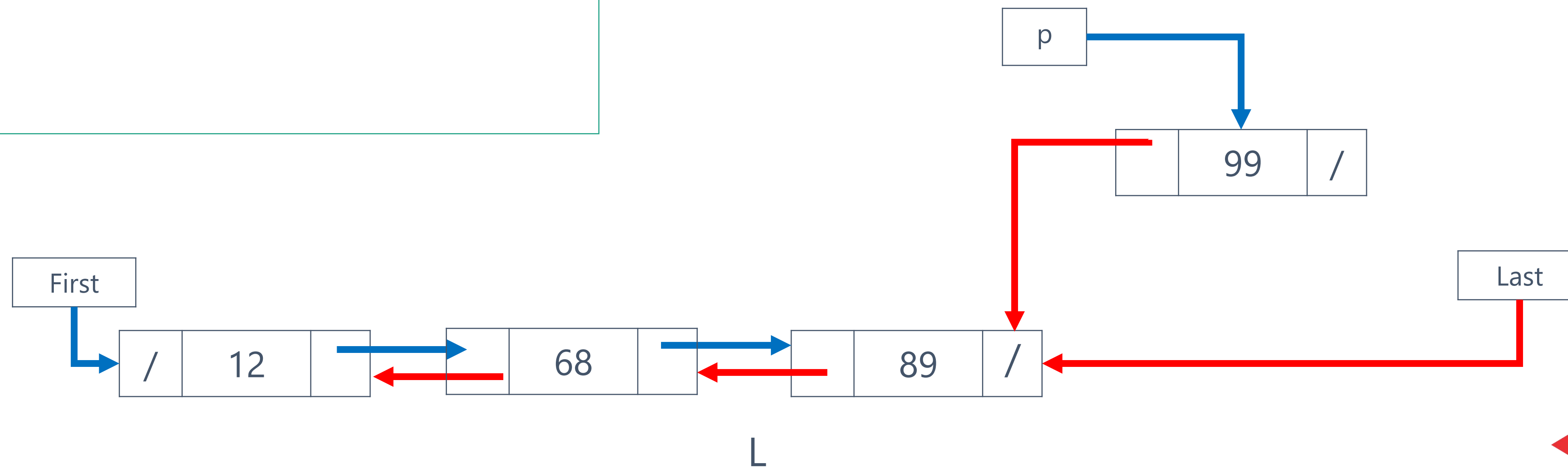
Algoritma



# INSERT LAST

Algoritma

**$\text{prev}(p) \leftarrow \text{Last}(L)$**

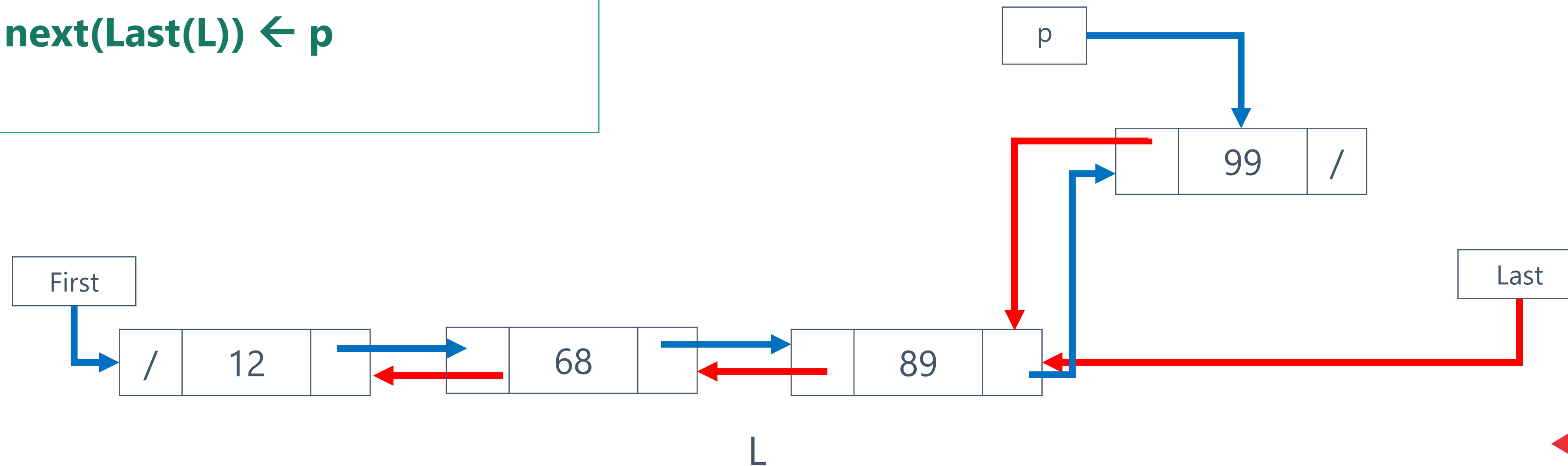


# INSERT LAST

## Algoritma

$\text{prev}(p) \leftarrow \text{Last}(L)$

$\text{next}(\text{Last}(L)) \leftarrow p$





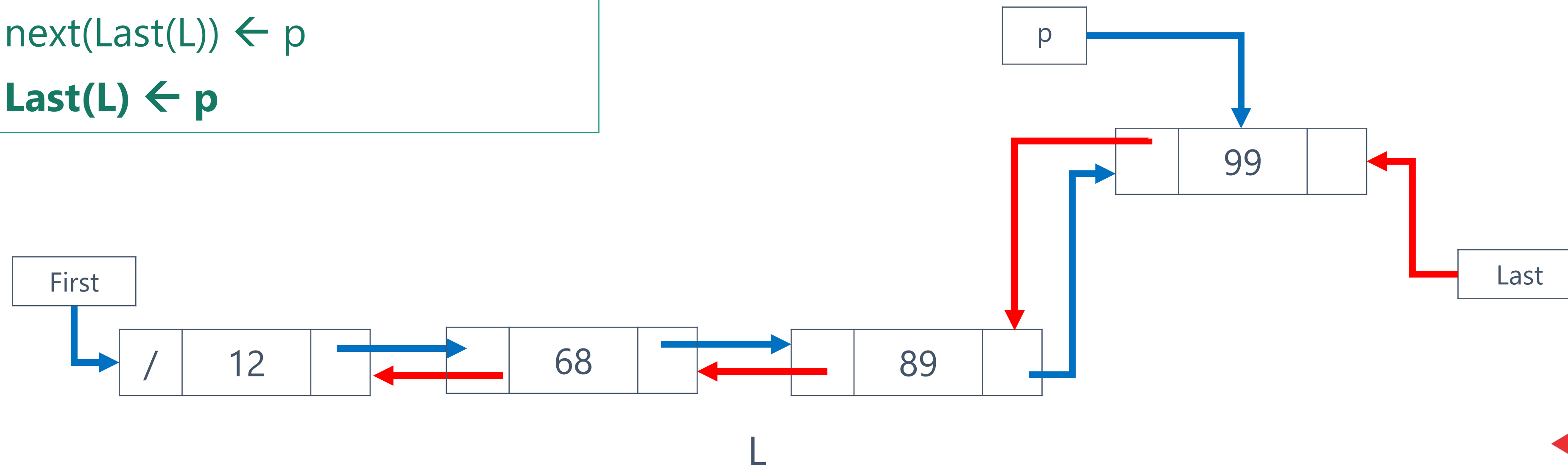
# INSERT LAST

## Algoritma

$\text{prev}(p) \leftarrow \text{Last}(L)$

$\text{next}(\text{Last}(L)) \leftarrow p$

**$\text{Last}(L) \leftarrow p$**



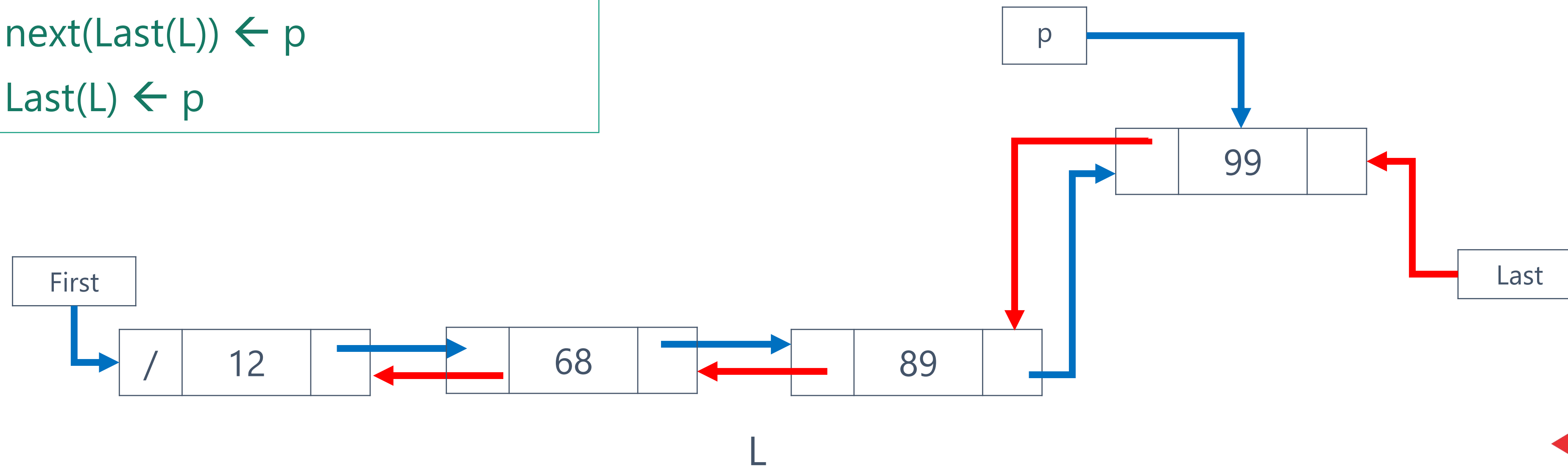
## INSERT LAST

### Algoritma

$\text{prev}(p) \leftarrow \text{Last}(L)$

$\text{next}(\text{Last}(L)) \leftarrow p$

$\text{Last}(L) \leftarrow p$



**HATI-HATI JIKA LIST AWAL KOSONG**

## INSERT LAST

**Procedure** InsertLast (In p: address, In/Out L: List)  
{ *IS: p adalah elemen baru,  $p \neq nil$ . List L tidak kosong.*  
  *FS: Elemen p menjadi elemen terakhir dari list L. }*

**Kamus**

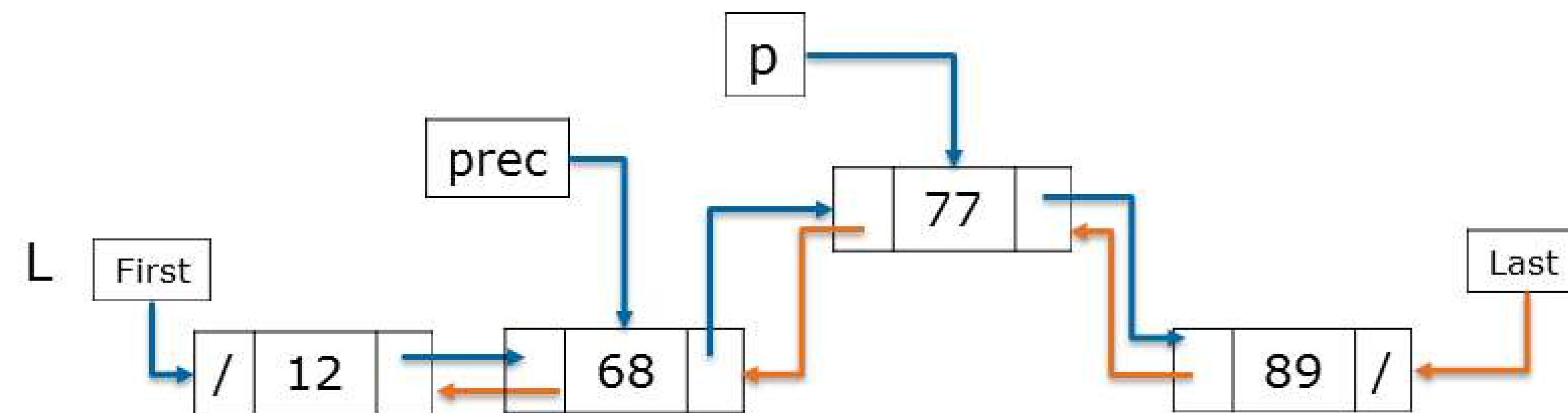
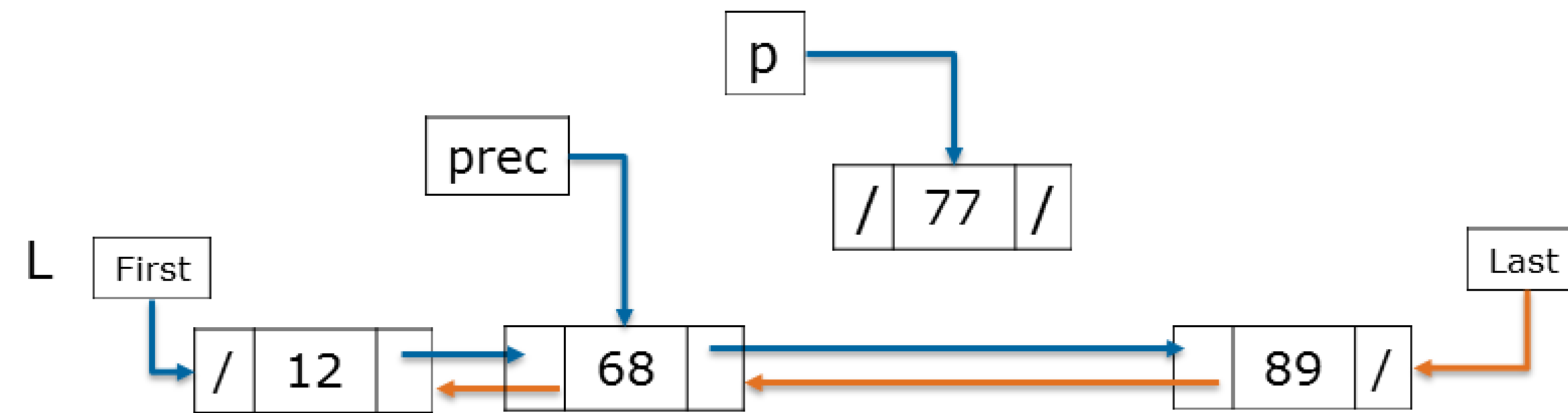
**Algoritma**

$prev(p) \leftarrow Last(L)$

$next(Last(L)) \leftarrow p$

$Last(L) \leftarrow p$

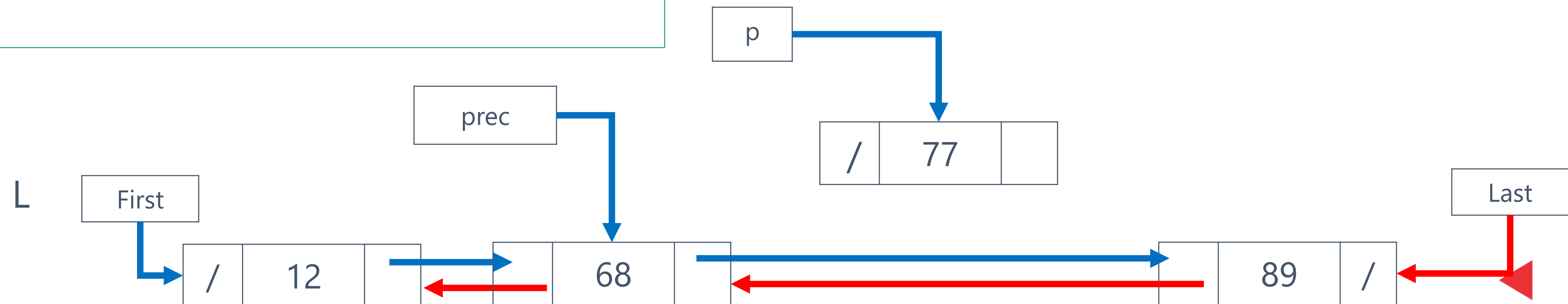
# INSERT AFTER



# INSERT AFTER

Algoritma

**INITIAL STATE**

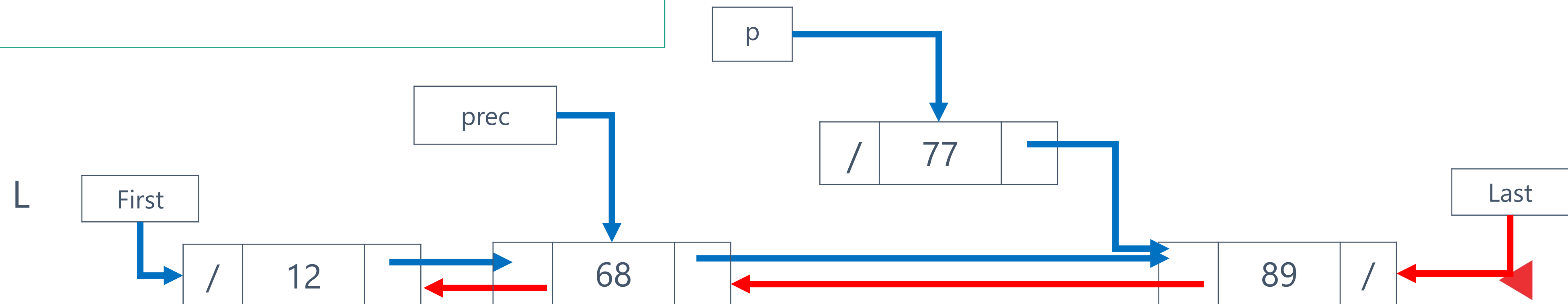




# INSERT AFTER

Algoritma

**$\text{next}(p) \leftarrow \text{next}(\text{prec})$**

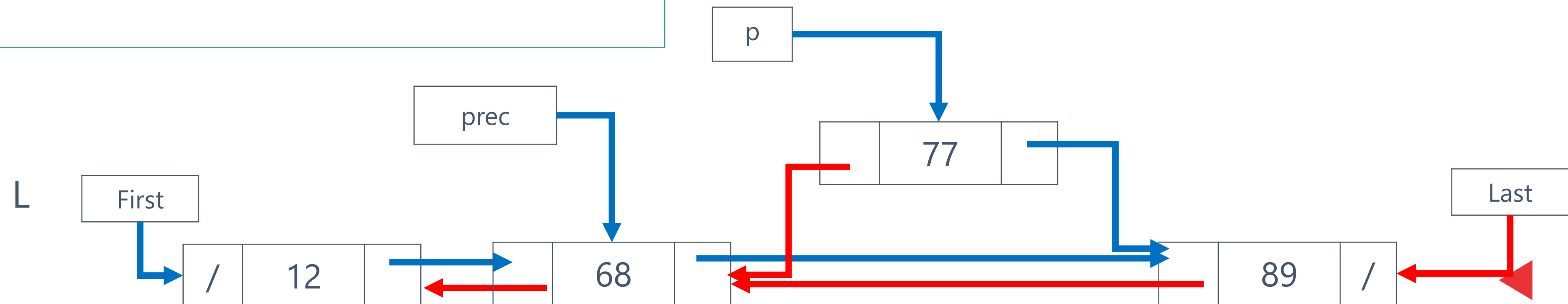


# INSERT AFTER

## Algoritma

$\text{next}(p) \leftarrow \text{next}(\text{prec})$

**$\text{prev}(p) \leftarrow \text{prec}$**



## INSERT AFTER

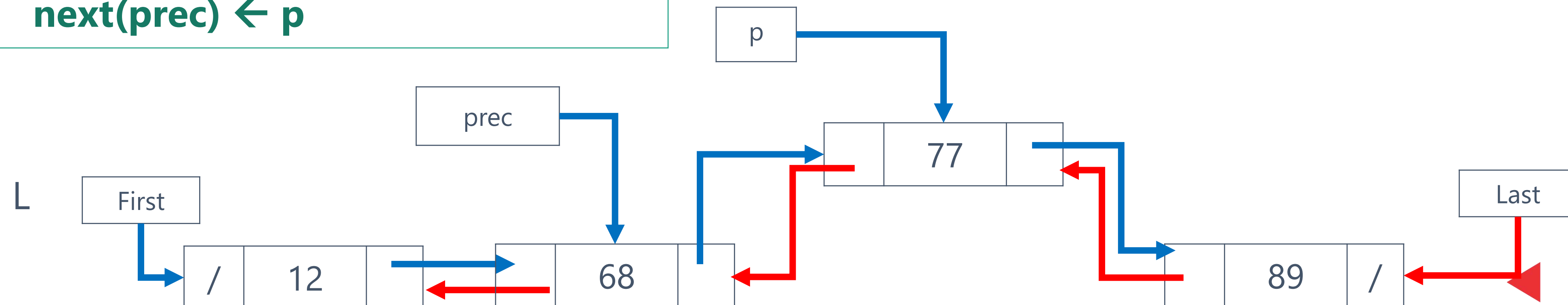
### Algoritma

$\text{next}(p) \leftarrow \text{next}(\text{prec})$

$\text{prev}(p) \leftarrow \text{prec}$

**$\text{prev}(\text{next}(\text{prec})) \leftarrow p$**

**$\text{next}(\text{prec}) \leftarrow p$**



## INSERT AFTER

**Procedure** InsertAfter (In p, prec: address)

*{ IS: p adalah elemen baru,  $p \neq \text{nil}$ . prec bukan elemen yang terakhir. }*

*FS: Elemen p disisipkan setelah elemen prec. }*

**Kamus**

**Algoritma**

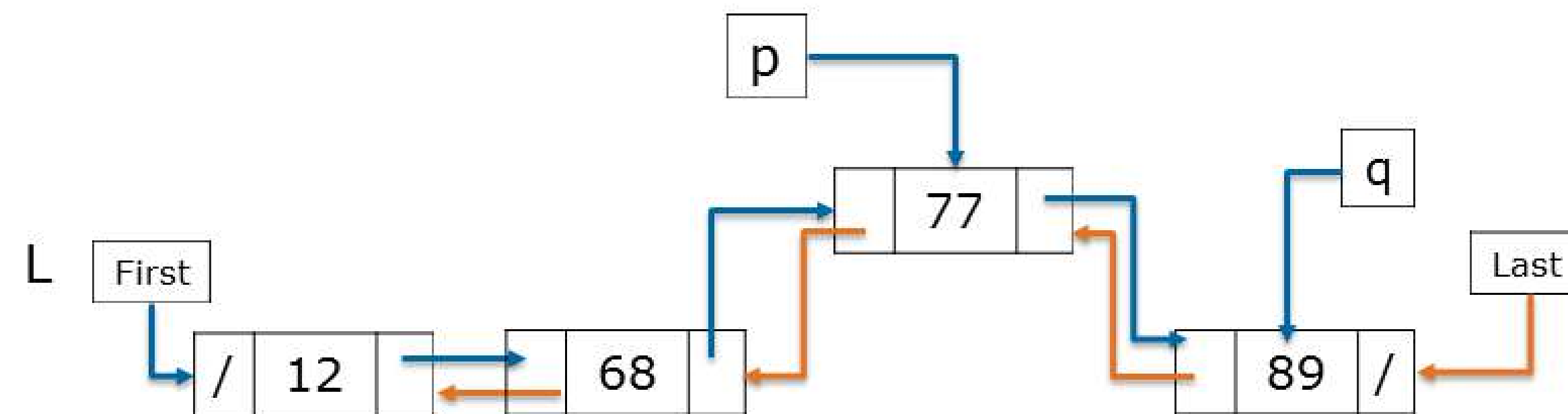
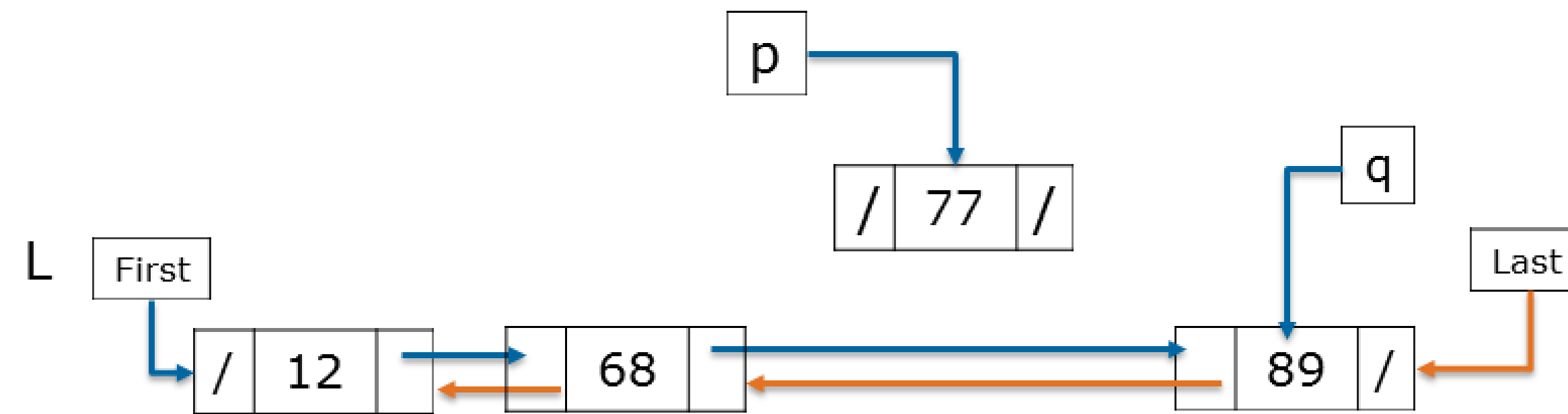
$\text{next}(p) \leftarrow \text{next}(\text{prec})$

$\text{prev}(p) \leftarrow \text{prec}$

$\text{prev}(\text{next}(\text{prec})) \leftarrow p$

$\text{next}(\text{prec}) \leftarrow p$

# INSERT BEFORE



A cluster of red triangles of various sizes and orientations in the top-left corner.

## PENGHAPUSAN ELEMEN

A small red triangle pointing right.

### Delete first

Menghapus elemen yang pertama.

### Delete last

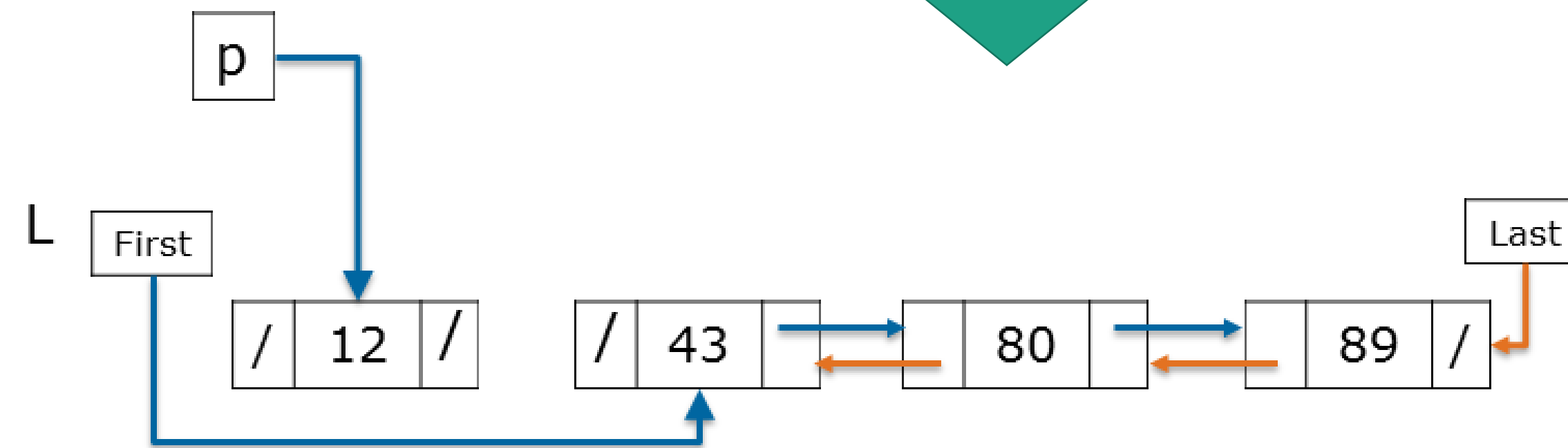
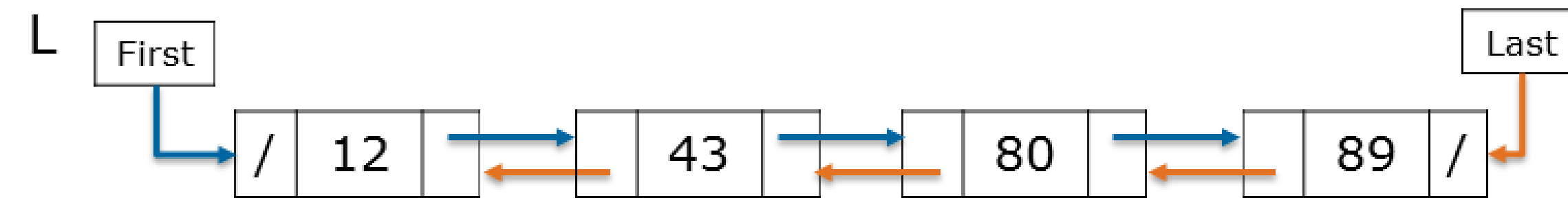
Menghapus elemen yang terakhir.

### Delete after/ before

Menghapus elemen yang berada setelah/ sebelum elemen lain.



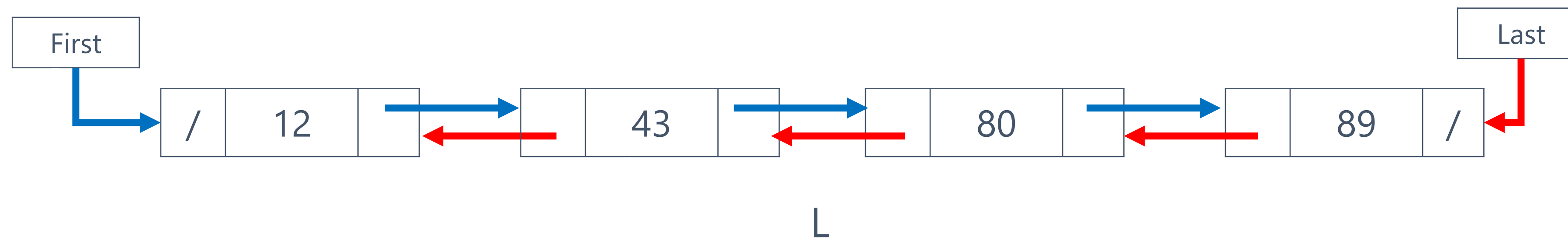
# DELETE FIRST



# DELETE FIRST

**INITIAL STATE**

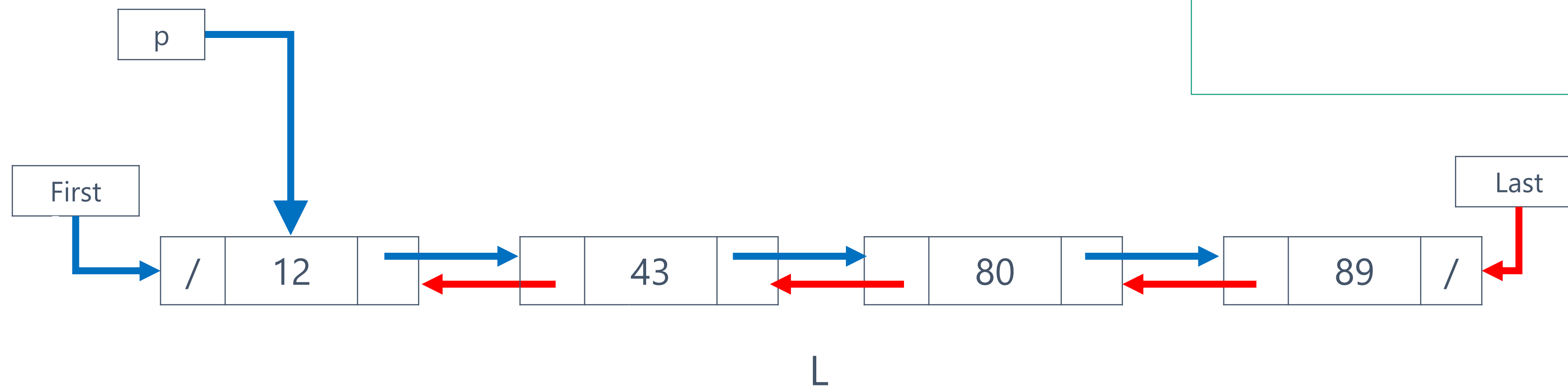
Algoritma



# DELETE FIRST

Algoritma

**$p \leftarrow \text{First}(L)$**

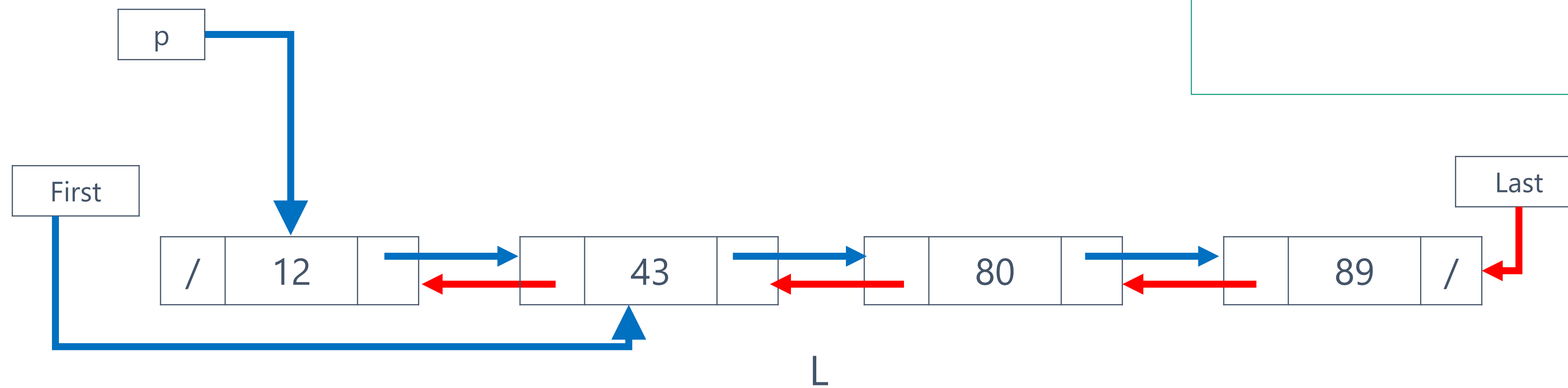


# DELETE FIRST

## Algoritma

$p \leftarrow \text{First}(L)$

**$\text{First}(L) \leftarrow \text{next}(p)$**

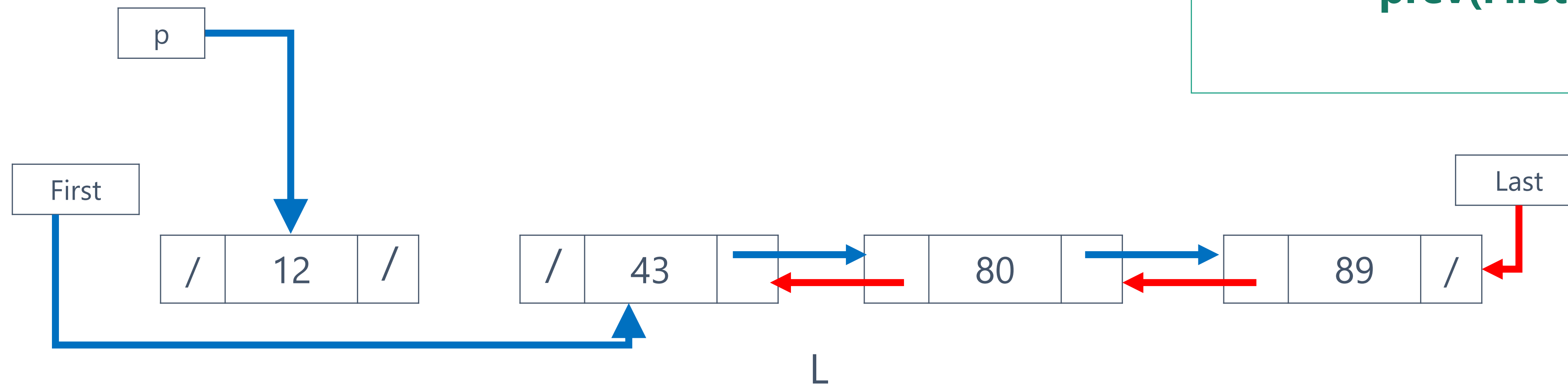


# DELETE FIRST

## Algoritma

```

p ← First(L)
First(L) ← next(p)
next(p) ← Nil
prev(First(L)) ← Nil
    
```

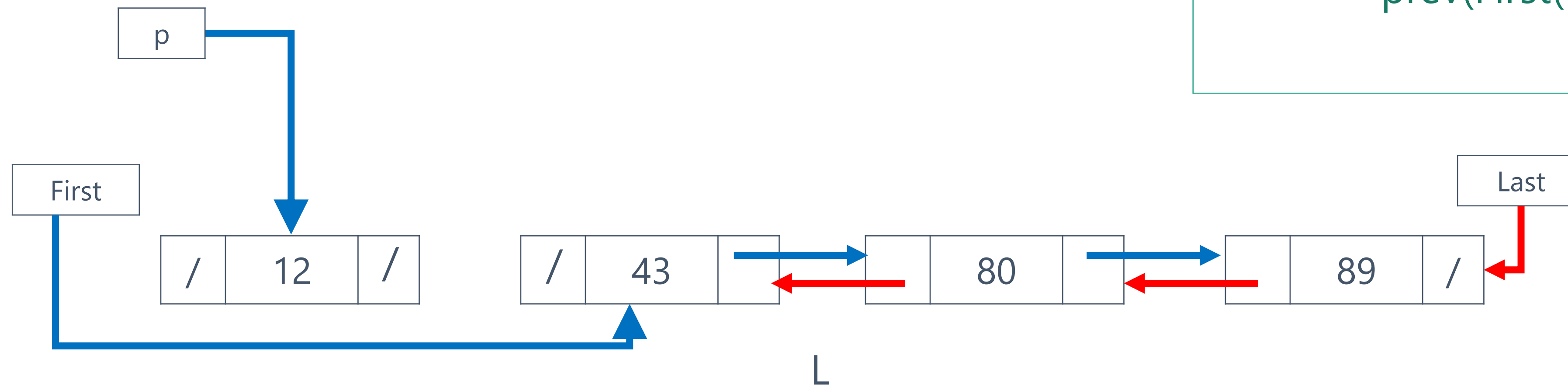


## DELETE FIRST

**BAGAIMANA JIKA LIST HANYA MEMILIKI SATU ELEMEN?**

### Algoritma

```
p ← First(L)
First(L) ← next(p)
next(p) ← Nil
prev(First(L)) ← Nil
```



# DELETE - SATU ELEMEN

**INITIAL STATE**



Algoritma

```

p ← First(L)
First(L) ← next(p)
next(p) ← Nil
prev(First(L)) ← Nil
  
```

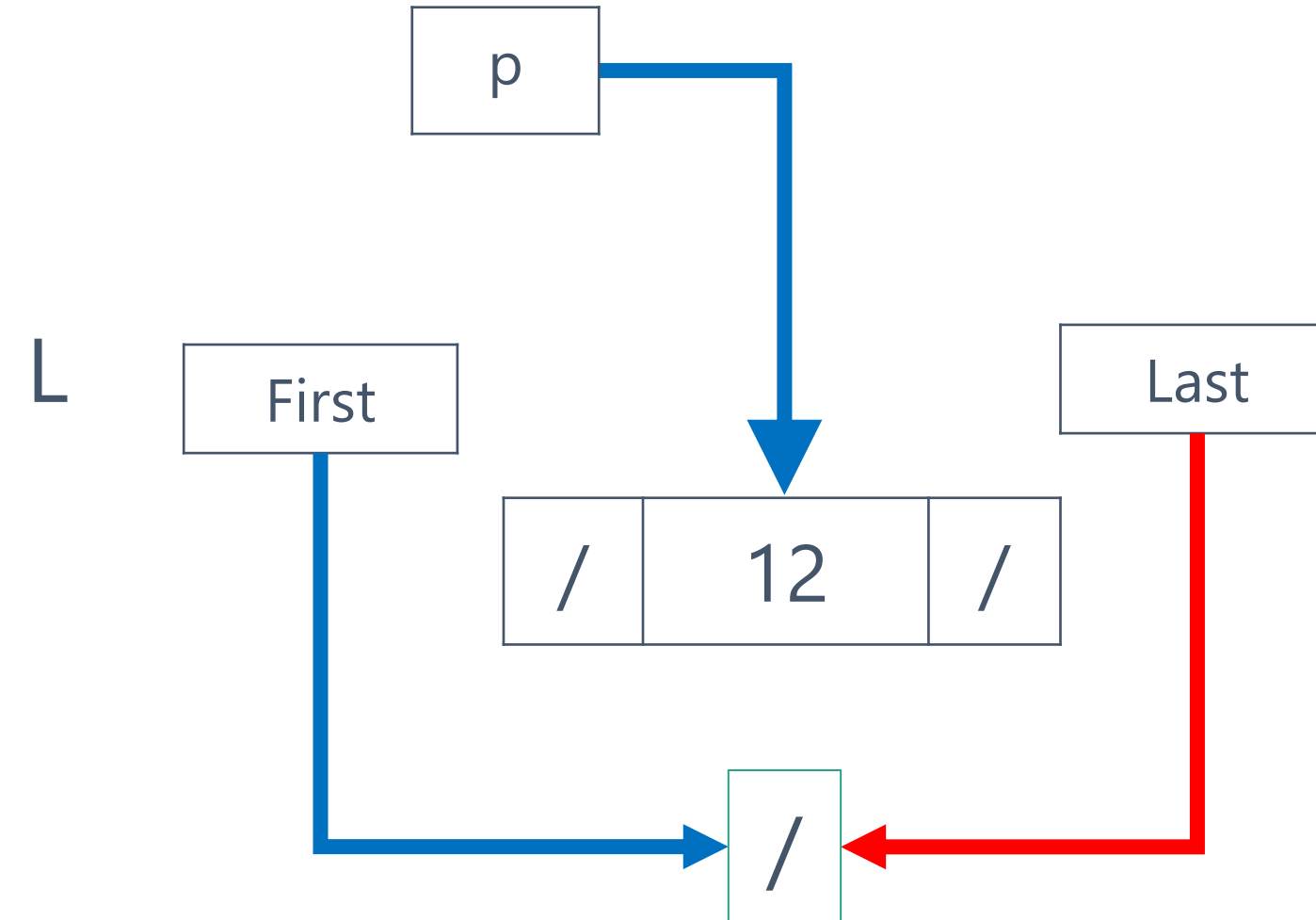
**ERROR**



## DELETE - SATU ELEMEN

Algoritma

$p \leftarrow \text{First}(L)$



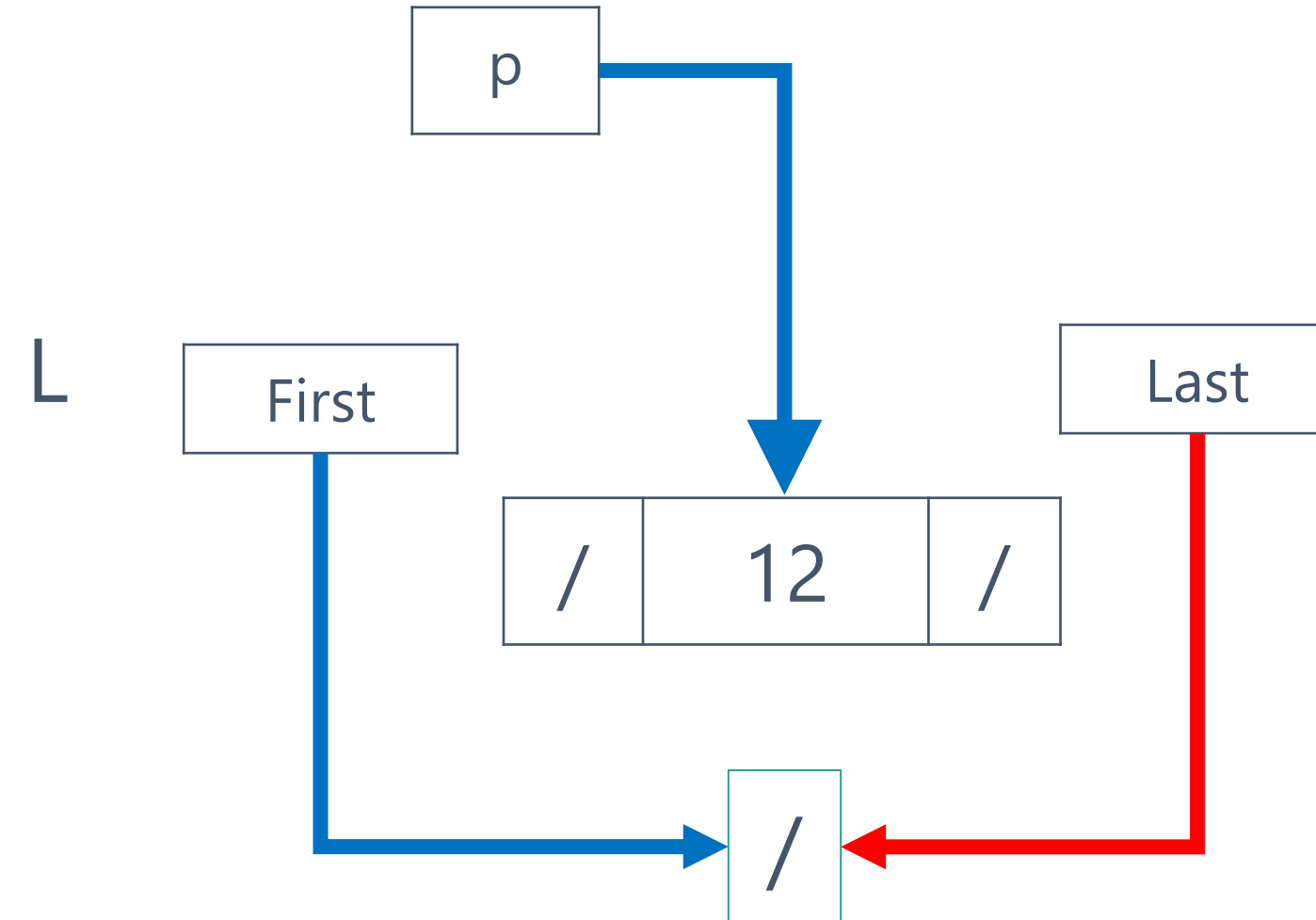
## DELETE - SATU ELEMEN

### Algoritma

$p \leftarrow \text{First}(L)$

**$\text{First}(L) \leftarrow \text{Nil}$**

**$\text{Last}(L) \leftarrow \text{Nil}$**



## DELETE FIRST

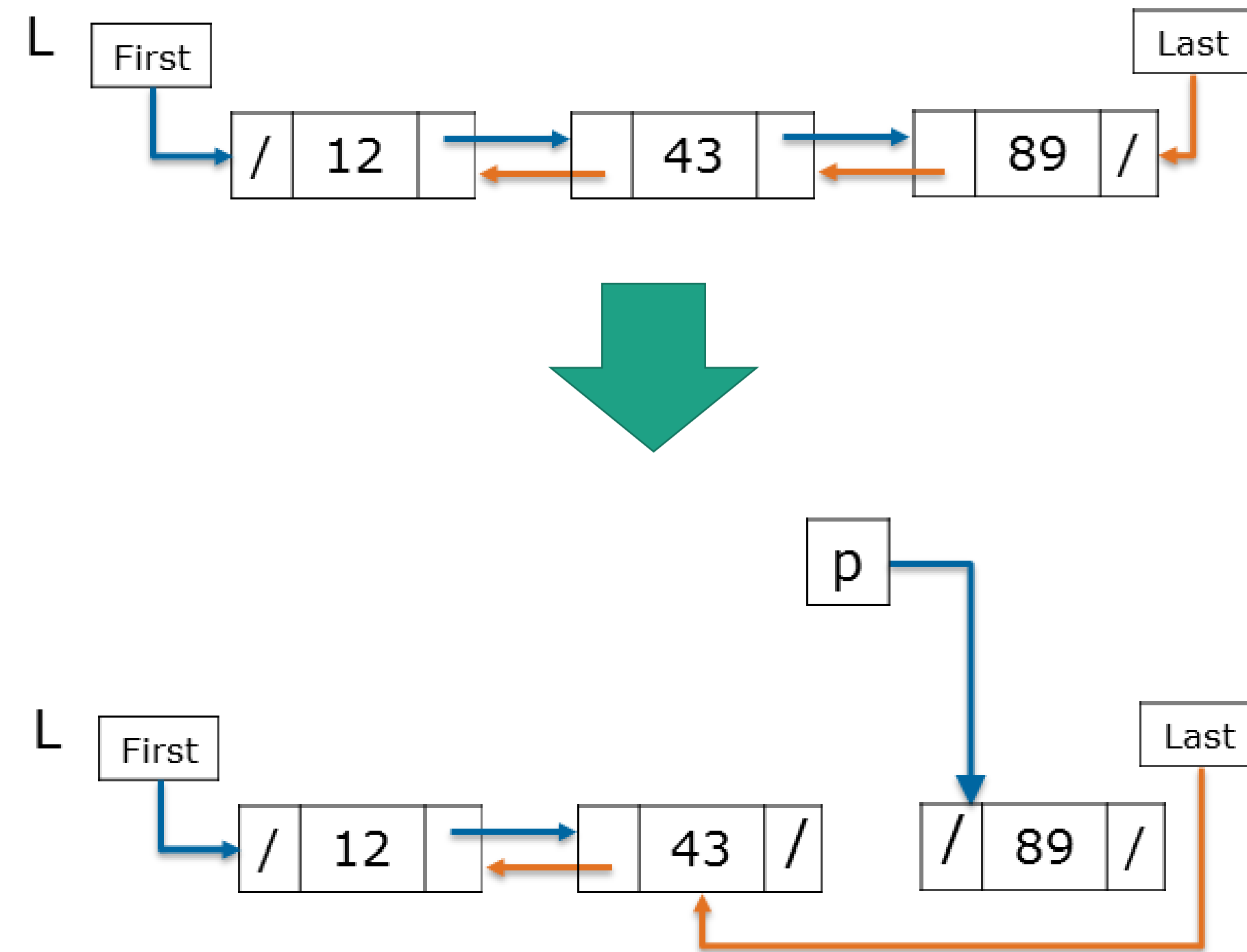
**Procedure** DeleteFirst(In/Out L: List, Out p: address)  
{ IS: List L tidak kosong. Minimal memiliki satu elemen.  
FS: p adalah elemen pertama yang dihapus dari list L. }

**Kamus**

**Algoritma**

```
p ← First(L)
if (First(L) ≠ Last(L)) then {jika list awal memiliki lebih dari satu elemen}
    First(L) ← next(p)
    next(p) ← Nil
    prev(First(L)) ← Nil
else {jika list awal memiliki satu elemen}
    First(L) ← Nil
    Last(L) ← Nil
```

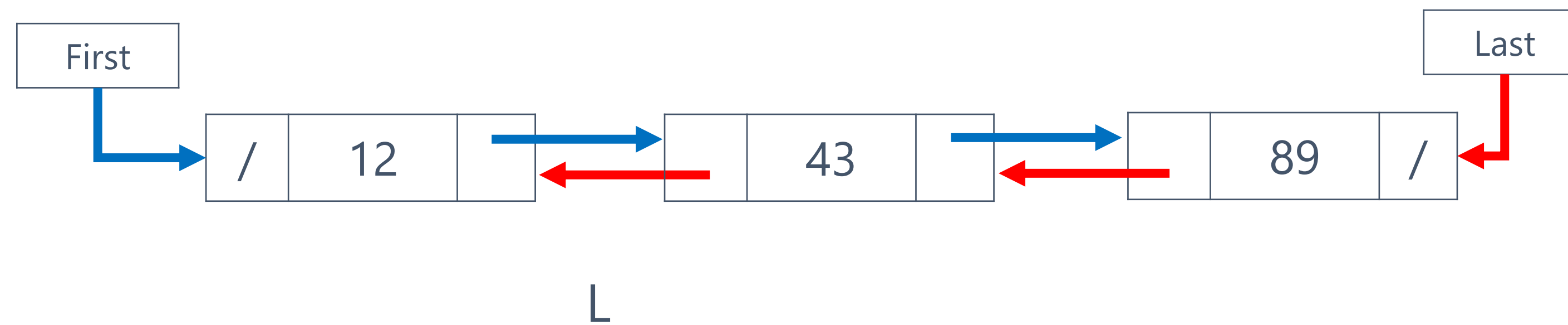
# DELETE LAST



# DELETE LAST

**INITIAL STATE**

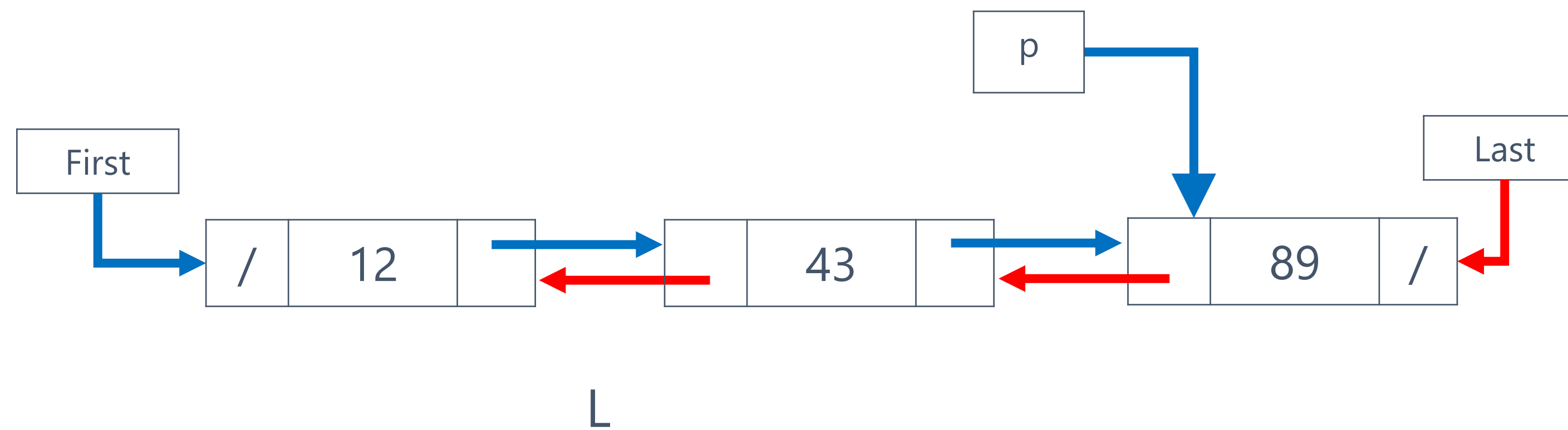
Algoritma



# DELETE LAST

Algoritma

$p \leftarrow \text{Last}(L)$

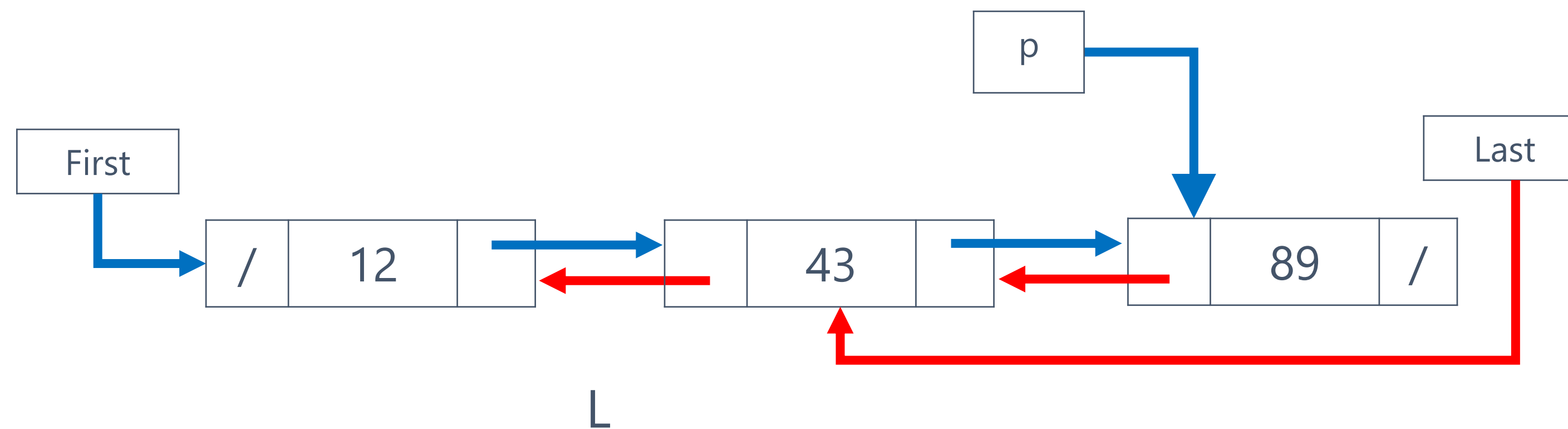


# DELETE LAST

## Algoritma

$p \leftarrow \text{Last}(L)$

**$\text{Last}(L) \leftarrow \text{prev}(\text{Last}(L))$**



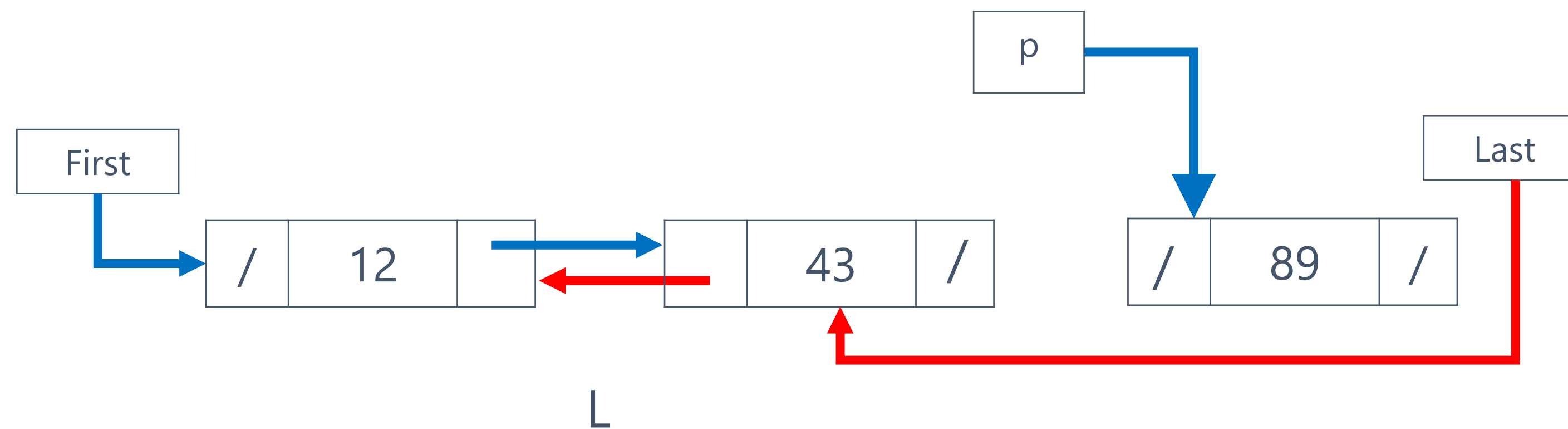


## DELETE LAST

### Algoritma

```

p ← Last(L)
Last(L) ← prev(Last(L))
prev(p) ← Nil
next(Last(L)) ← Nil
    
```



## DELETE LAST

**Procedure** DeleteLast(In/Out L: List, Out p: address)  
{ IS: List L tidak kosong, memiliki lebih dari satu elemen.  
FS: p adalah elemen terakhir yang dihapus dari list L. }

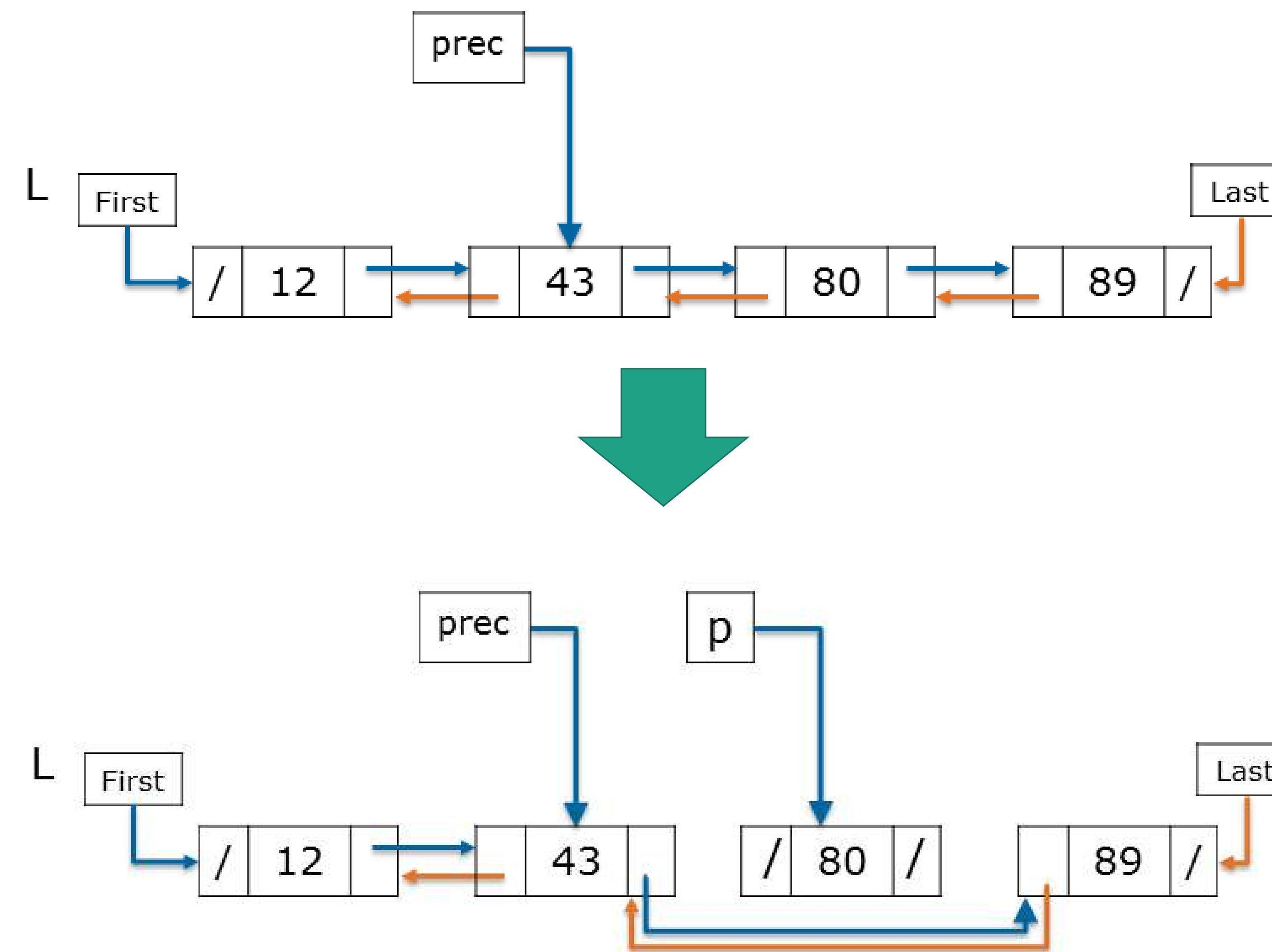
**Kamus**

**Algoritma**

$p \leftarrow \text{Last}(L)$   
 $\text{Last}(L) \leftarrow \text{prev}(\text{Last}(L))$   
 $\text{prev}(p) \leftarrow \text{Nil}$   
 $\text{next}(\text{Last}(L)) \leftarrow \text{Nil}$

**HATI-HATI JIKA LIST AWAL HANYA  
MEMILIKI SATU ELEMEN**

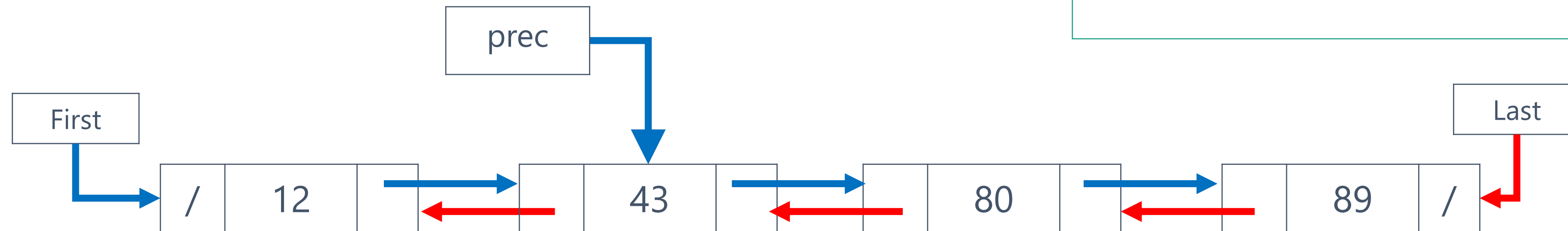
## DELETE AFTER



# DELETE AFTER

**INITIAL STATE**

Algoritma

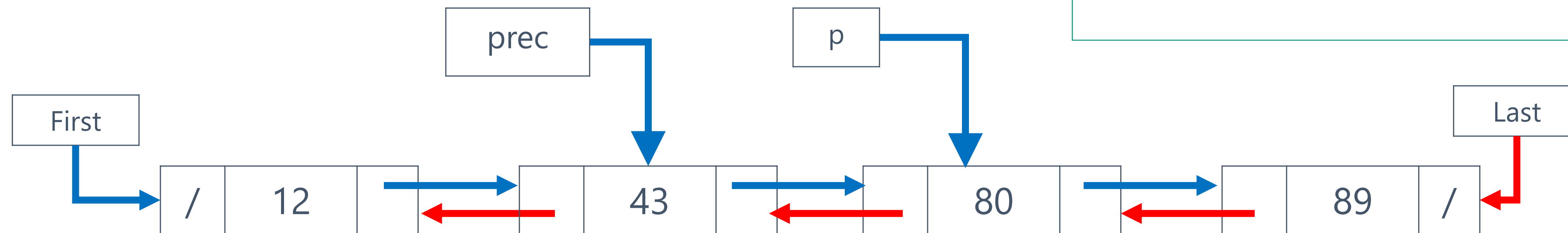


L

# DELETE AFTER

Algoritma

$p \leftarrow \text{next}(\text{prec})$



L

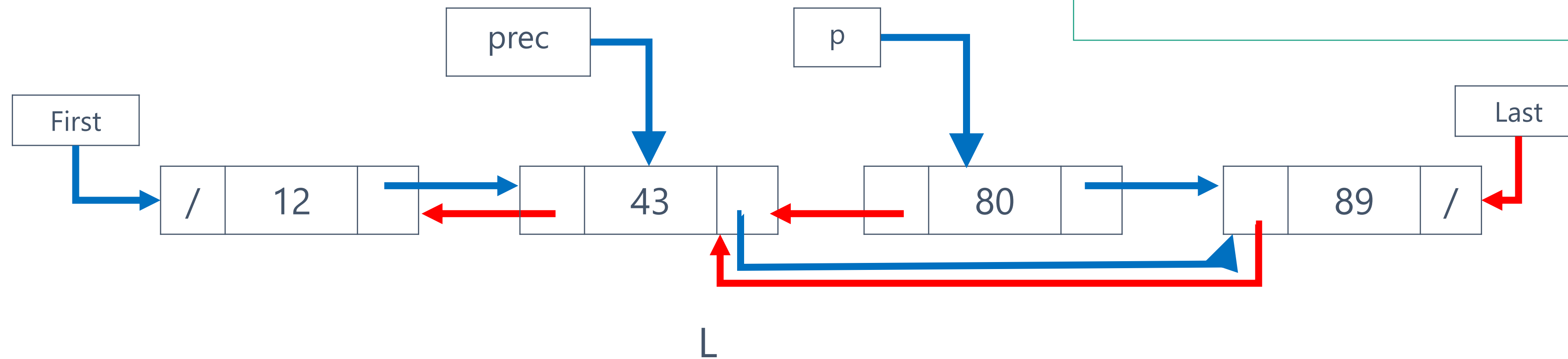
## DELETE AFTER

### Algoritma

$p \leftarrow \text{next}(\text{prec})$

**$\text{next}(\text{prec}) \leftarrow \text{next}(p)$**

**$\text{prev}(\text{next}(p)) \leftarrow \text{prec}$**

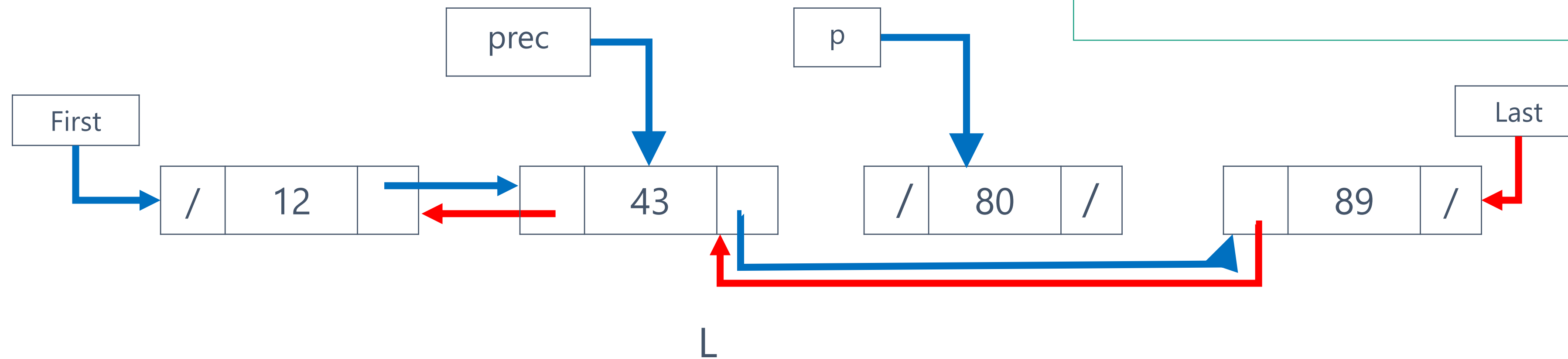


## DELETE AFTER

### Algoritma

```

p ← next(prec)
next(prec) ← next(p)
prev(next(p)) ← prec
prev(p) ← Nil
next(p) ← Nil
    
```





## DELETE AFTER

**Procedure** DeleteAfter(In prec: address, Out p: address)

*{ IS: prec adalah sebuah elemen dalam list,  $prec \neq nil$ , prec bukan elemen terakhir..*

*FS: p adalah elemen setelah prec yang dihapus dari list. }*

**Kamus**

**Algoritma**

$p \leftarrow \text{next}(\text{prec})$

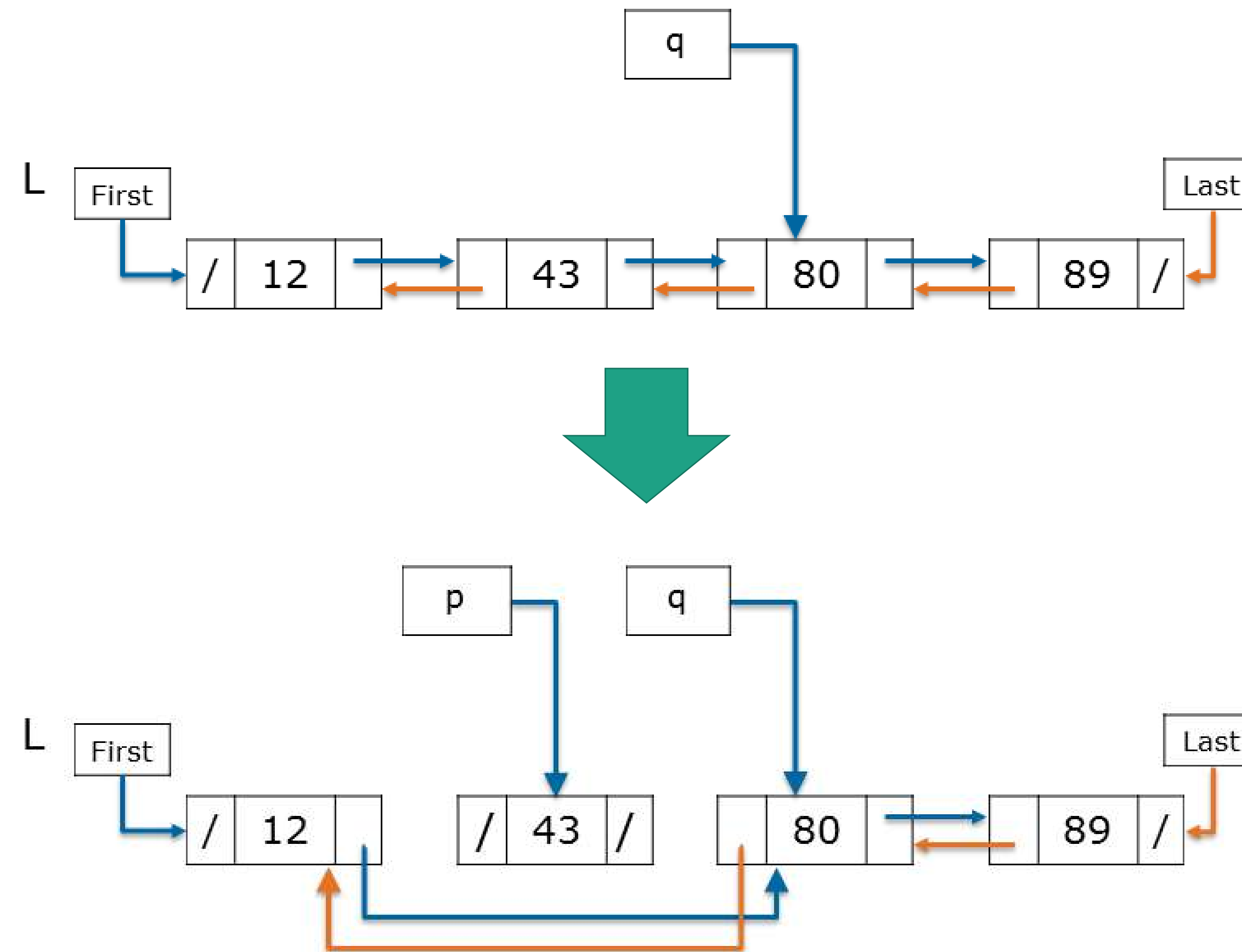
$\text{next}(\text{prec}) \leftarrow \text{next}(p)$

$\text{prev}(\text{next}(p)) \leftarrow \text{prec}$

$\text{prev}(p) \leftarrow \text{Nil}$

$\text{next}(p) \leftarrow \text{Nil}$

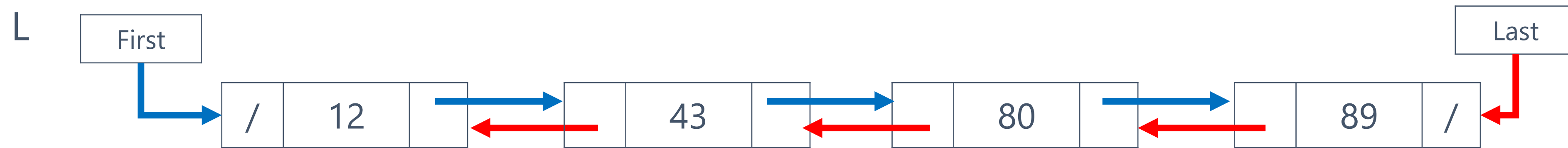
## DELETE BEFORE



## PENELUSURAN DOUBLE LINKED LIST

Double linked list bisa ditelusuri secara dua arah, baik secara maju dari satu elemen ke elemen selanjutnya, atau mundur dari satu elemen ke elemen sebelumnya.

## PENELUSURAN MAJU

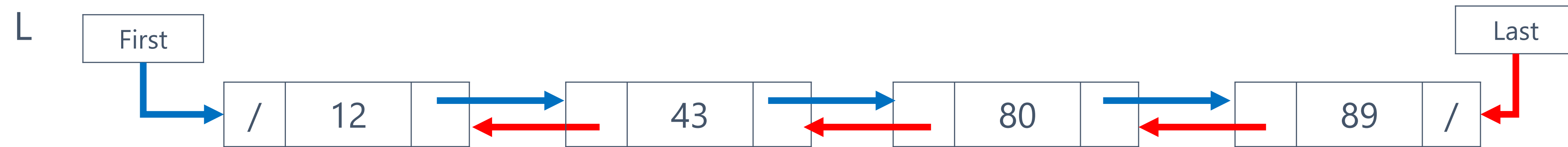


### Algoritma

```

p ← First(L)
while (p ≠ Nil) do
    p ← next(p)
    
```

## PENELUSURAN MUNDUR



### Algoritma

```

p ← Last(L)
while (p ≠ Nil) do
    p ← prev(p)
    
```

## PENCARIAN

- Pencarian diperlukan untuk menemukan data, penyisipan data di posisi tertentu, dan penghapusan data tertentu.
- Function SearchAddress(L: List, data: infotype) → address
- Function SearchData (L: List, data: infotype) → boolean
- Menerapkan algoritma sequential search.

## PENERAPAN

- Aplikasi yang memiliki list Most Recent Used (MRU) → List file
- Cache pada browser → List URL



TERIMA KASIH