# ADATSZERKEZETEK ÉS ALGORITMUSOK

Szekvenciális adatszerkezetek

# Szekvenciális adatszerkezetek

- Definíció: A szekvenciális adatszerkezet olyan $\langle A, R \rangle$ rendezett pár amelynél az $R \subseteq (A \times A)$ reláció tranzitív lezártja teljes rendezési reláció
- Az $R \subseteq (A \times A)$ reláció tranzitív lezártja az a reláció, mely tranzitív, tartalmazza R-et, és a lehető legkevesebb elemet tartalmazza
- Megadása:
  1. $R' = R \cup (R \circ R)$
  2. Ha $R \neq R'$, akkor folyt. 1.-nél, különben $R' = R_T$, a tranzitív lezárt

# Szekvenciális adatszerkezetek

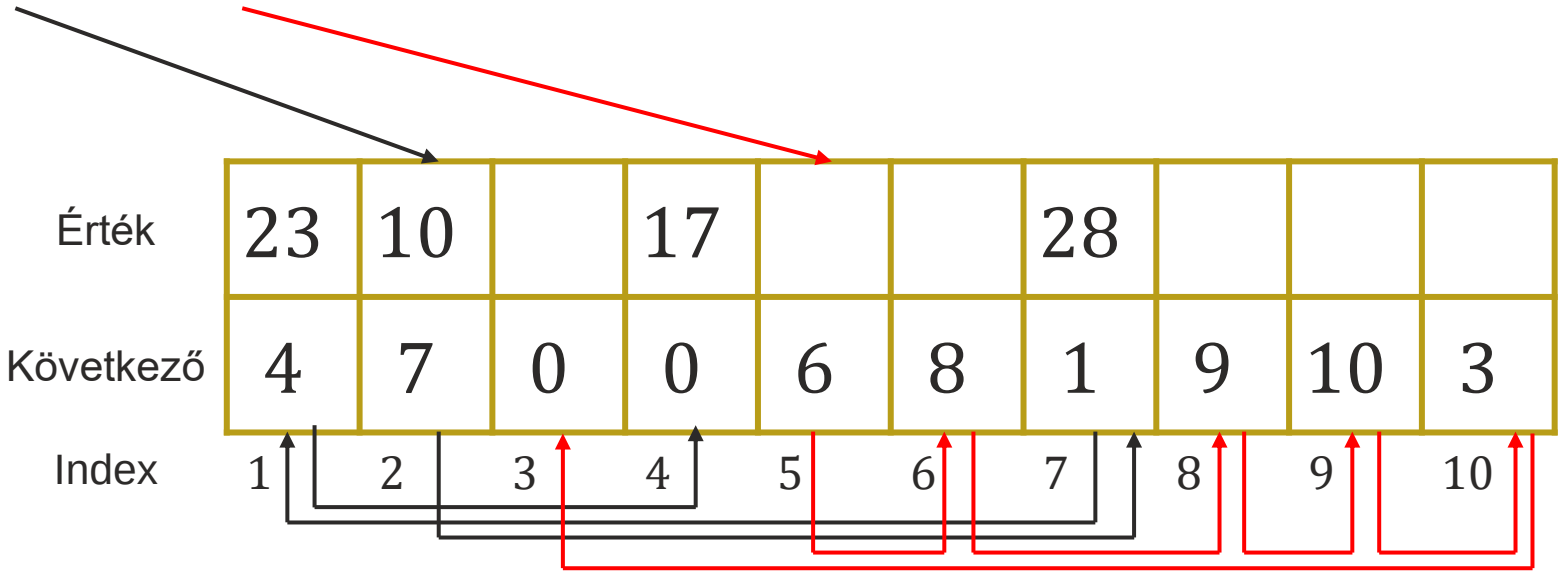
- Szekvenciális adatszerkezetben az egyes adatelemek egymás után helyezkednek el
  - Van egy logikai sorrendjük
- Az adatok között egy-egy jellegű a kapcsolat
  - Minden adatelem csak egy helyről érhető el és az adott elemtől csak egy másik látható
- Két kitüntetett elem
  - az első
  - az utolsó

# Szekvenciális adatszerkezetek

- Ez egy homogén adatszerkezet, azaz azonos típusú véges adatelemek sorozata
  - Jelölése : $L = (a_1, a_2, \ldots\ a_n\ )$
  - Ha $n = 0$, akkor $L = ()$ az üres lista.
- A láncolt lista olyan adatszerkezet, amelynek minden eleme tartalmaz egy (vagy több) mutatót (hivatkozást) egy másik, ugyanolyan típusú adatelemre
  - Ez a „következő” elem logikailag
- A lánc első elemének a címét a lista feje tartalmazza
  A listafej nem tartalmaz információs részt
- A lánc végét az jelzi, hogy az utolsó elemben a rákövetkező elem mutatója üres

# Reprezentációs szint

- Fix kapacitású ábrázolás
  - tömbben, a logikai sorrendet indexek mutatják, a szabad helyek is listában:
- L: 2 SZH: 5



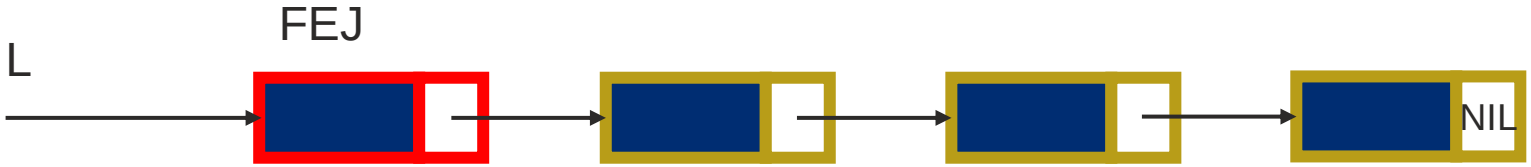
# Dinamikus, láncolt ábrázolás

- Az adatok száma nem ismert előre
  - Nem tudunk, vagy nem akarunk feleslegesen helyet foglalni az adatoknak
  - A feladat dinamikusan változik

# Láncolt ábrázolás

- Egyirányú láncolt lista
  - Fejelem nélkül

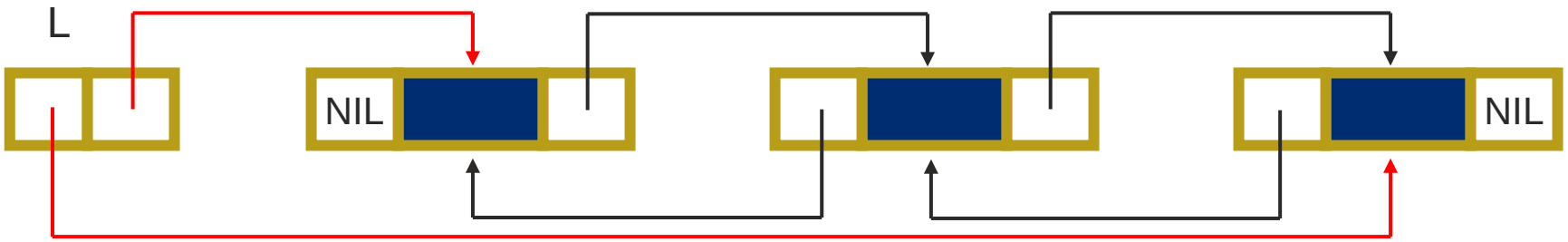

  - Fejelemmel: fejelem mindig létezik, ha üres a lista, akkor is




- Mindig van egy aktuális elemre mutató is, ez része a megvalósításnak

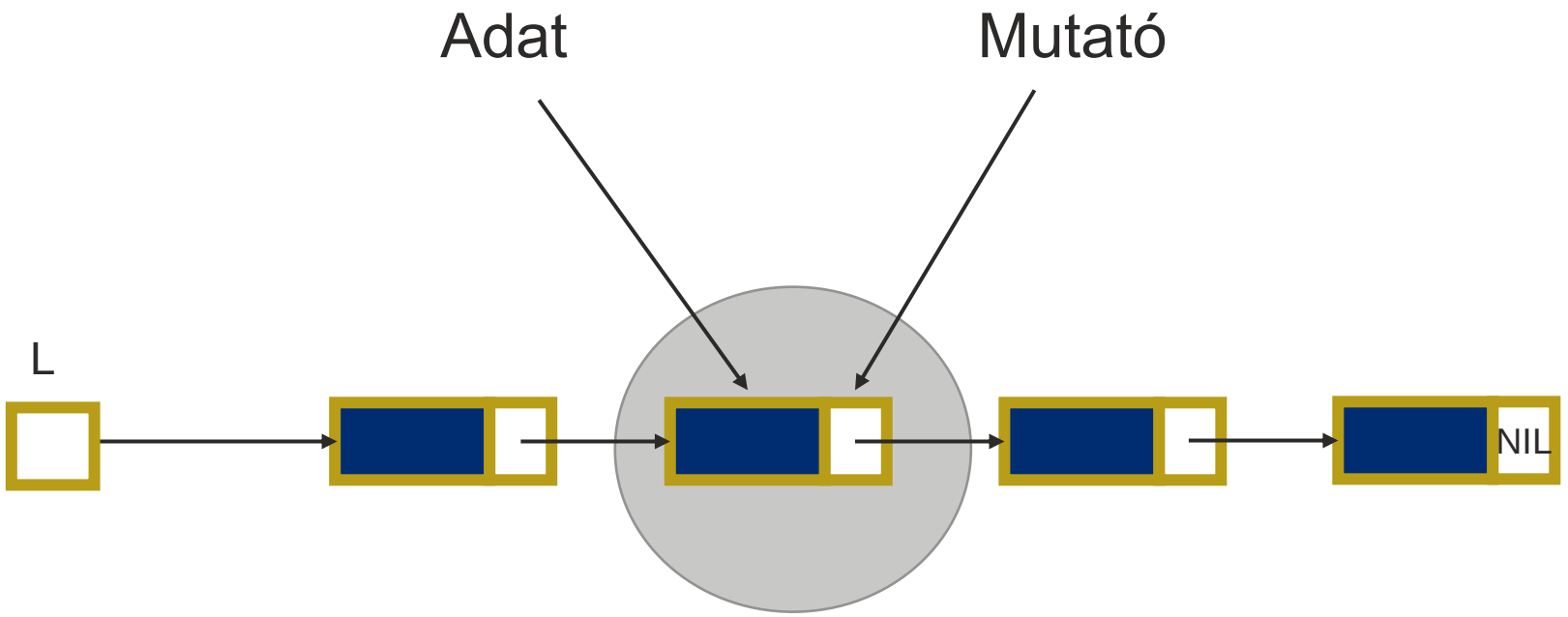
# Láncolt ábrázolás

- Kétirányú láncolt lista



# Láncolt ábrázolás

- A lista eleme

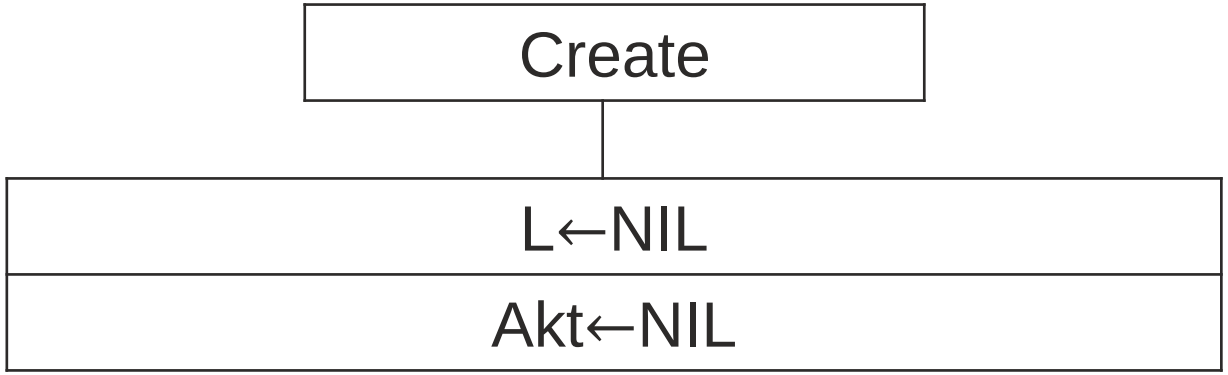


# Egyszerű lista – műveletek

- A Lista típus komponensei:
  - L
    - A lista első elemének mutatója,
  - Akt
    - A lista aktuális elemének mutatója

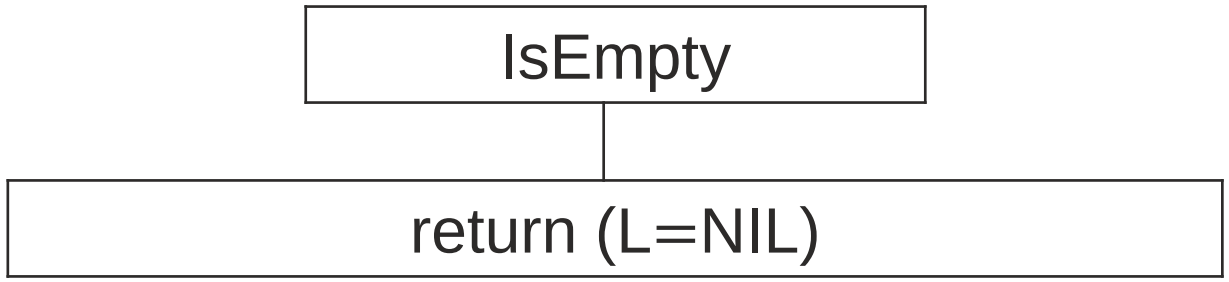
# Létrehozás

- Üres listát ad vissza

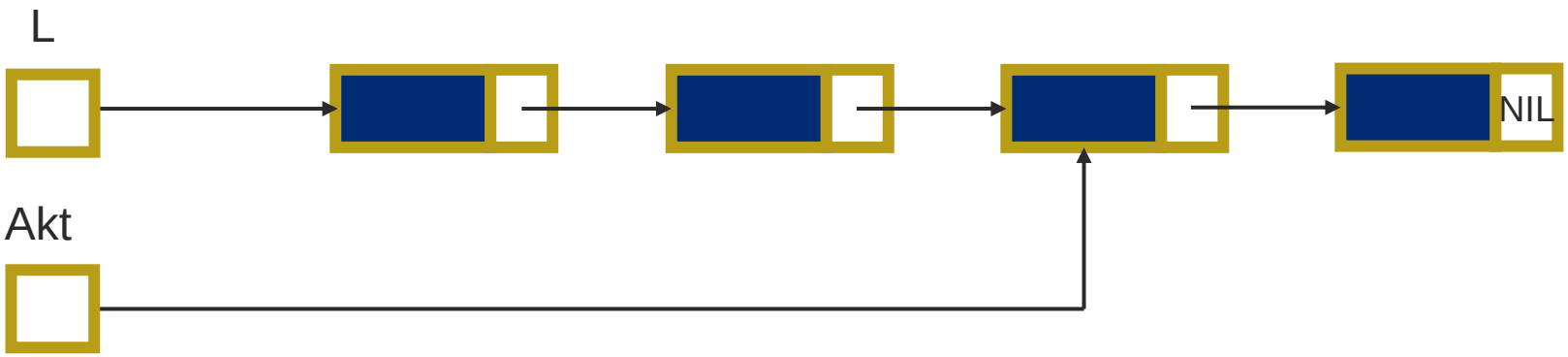


# Üres lista lekérdezése

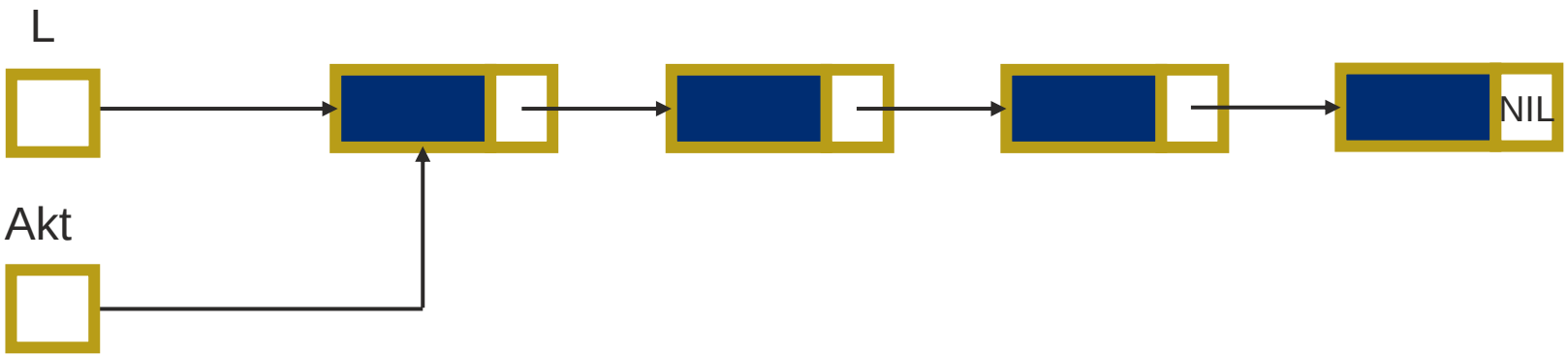
- Logikai értéket ad vissza



# Első elemre áll

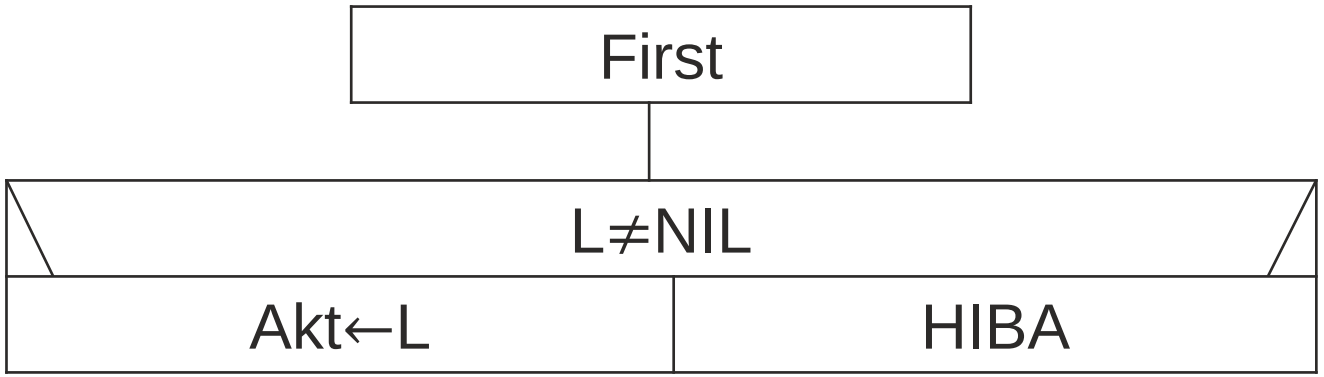


# Első elemre áll

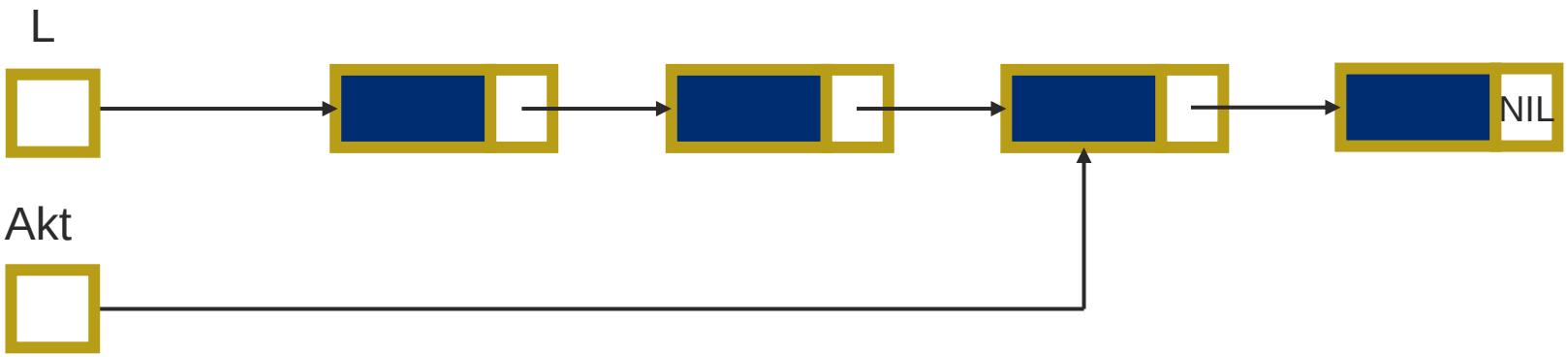


# Üres lista lekérdezése

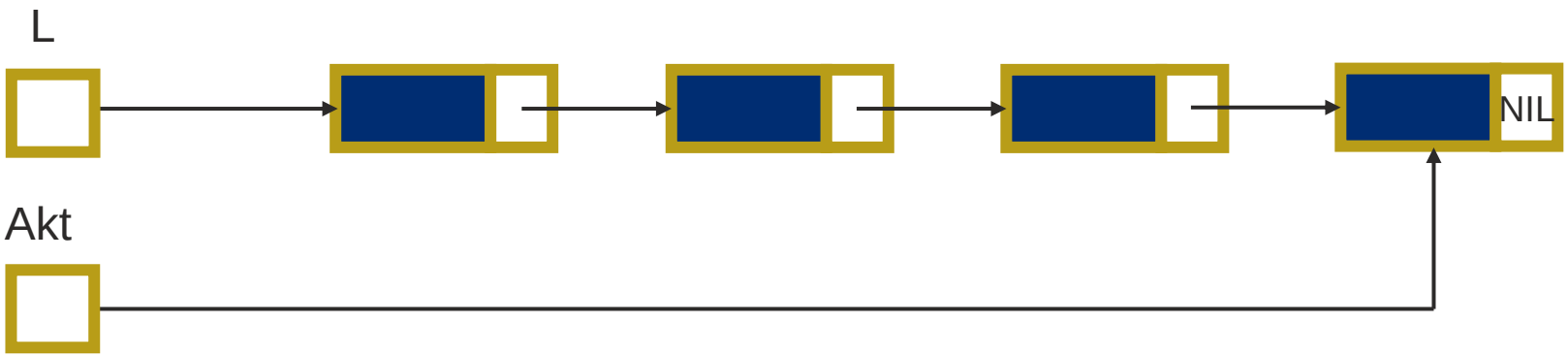
- Üres lista esetén hiba



# Következő elemre áll

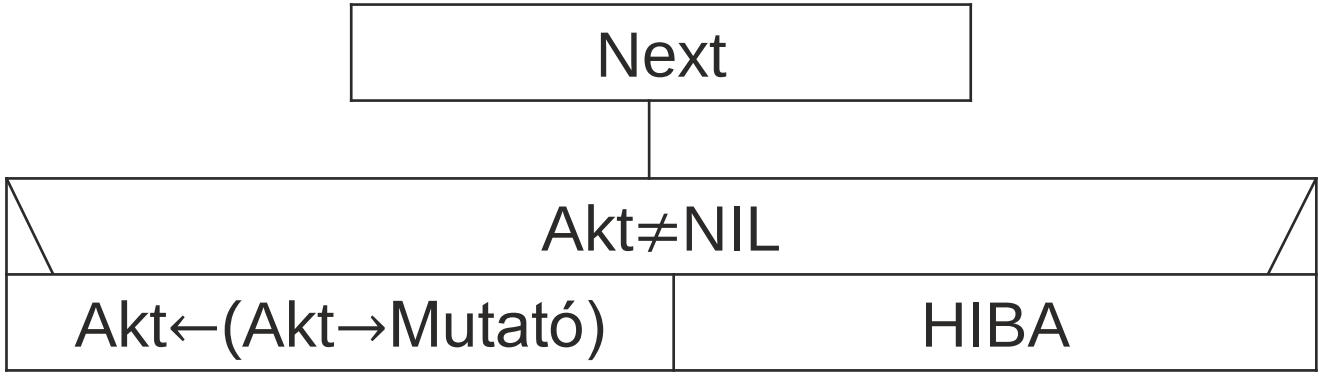


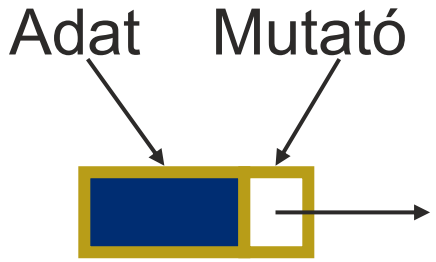
# Következő elemre áll



# Következő elemre áll

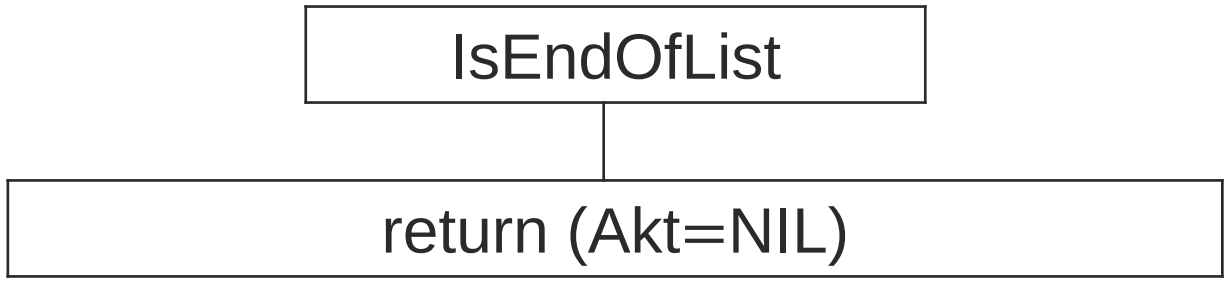
- A lista utolsó elemére kiadott Next hatása Akt=NIL lesz



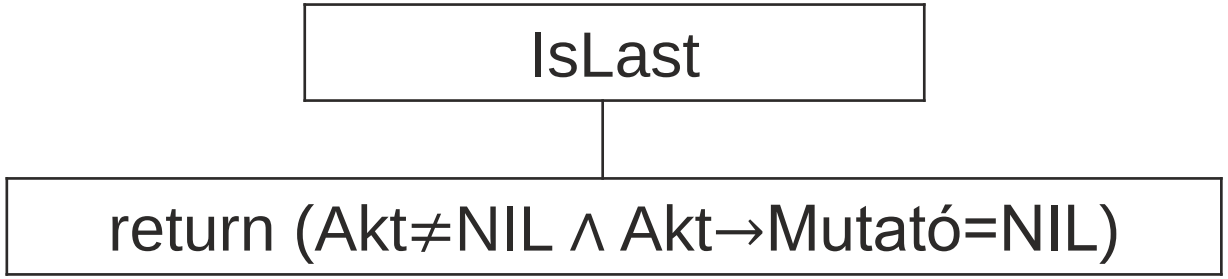
# Lista végének lekérdezése

- Logikai értéket ad vissza



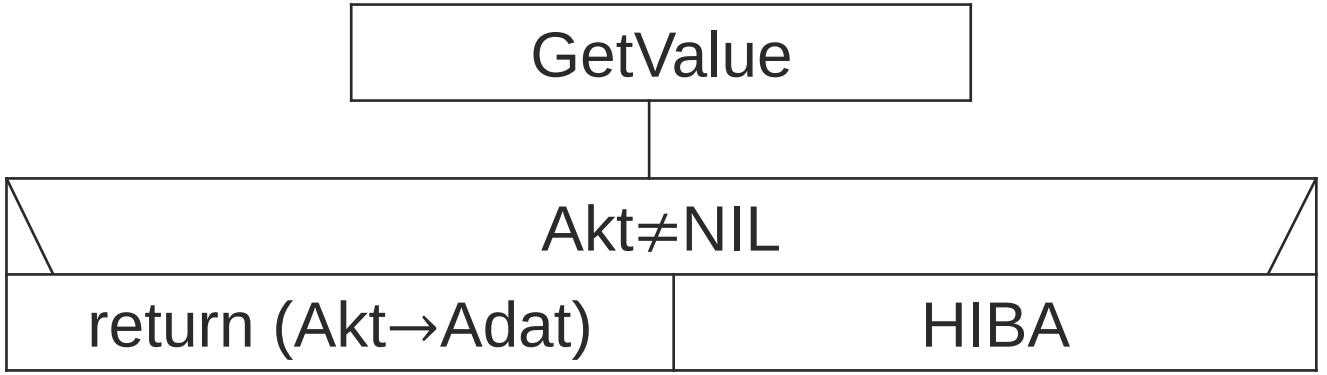
# Az aktuális az utolsó elem-e

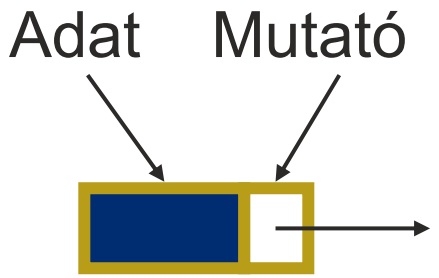
- Logikai értéket ad vissza



# Aktuális elem értéke

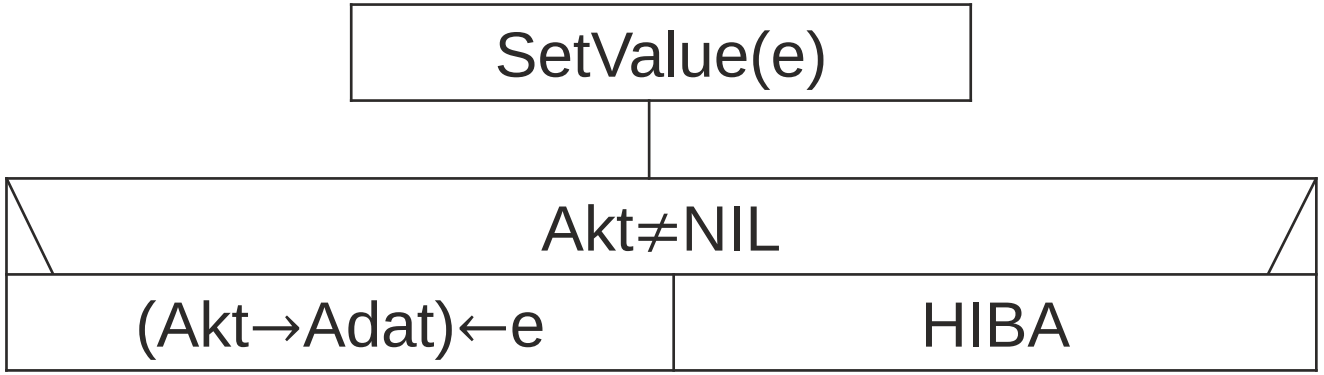
- Az aktuális elem értékével tér vissza



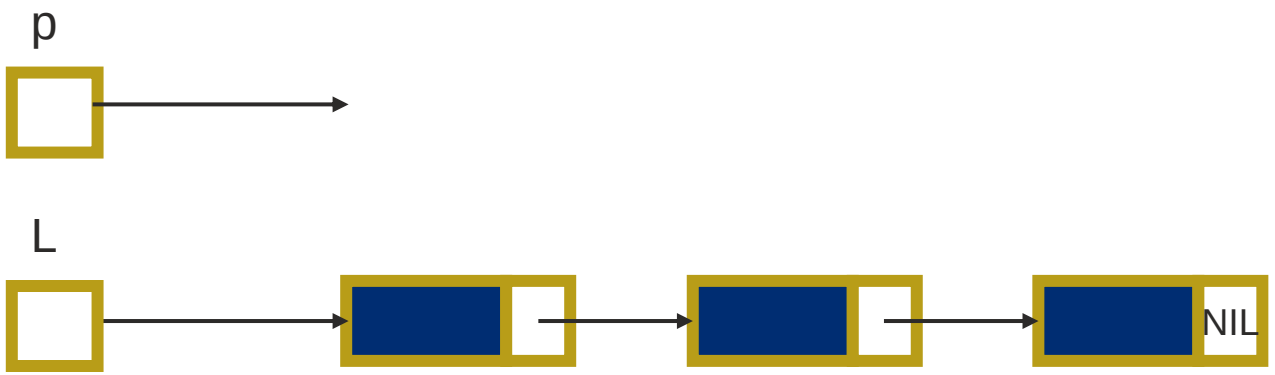
# Aktuális elem módosítása

- Az aktuális elem megváltoztatása e -re



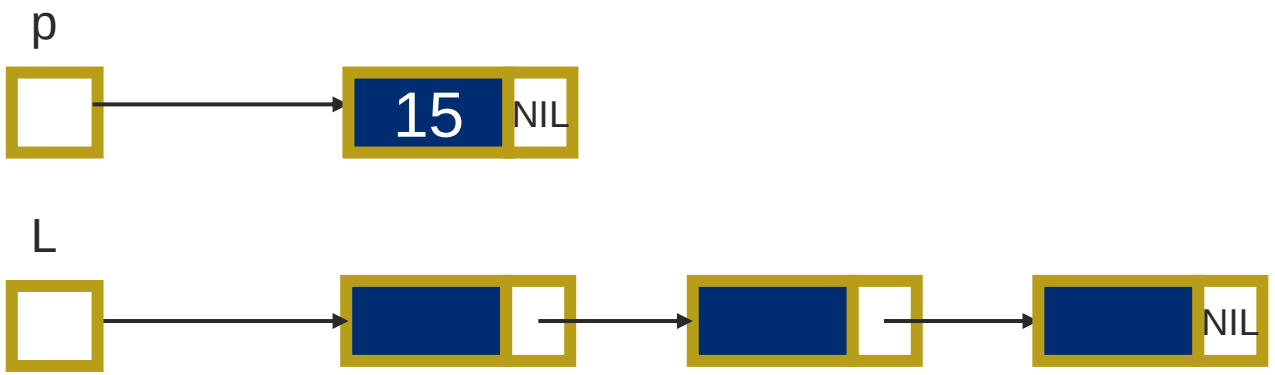
# Listaelem beszúrása

- Új listaelem létrehozás és beállítása
  - Deklarálás – p változó, Node típussal



# Listaelem beszúrása

- Új listaelem létrehozás és beállítása
  - Deklarálás – p változó, Node típussal
  - Létrehozás – new(p)
  - Értékének megadása

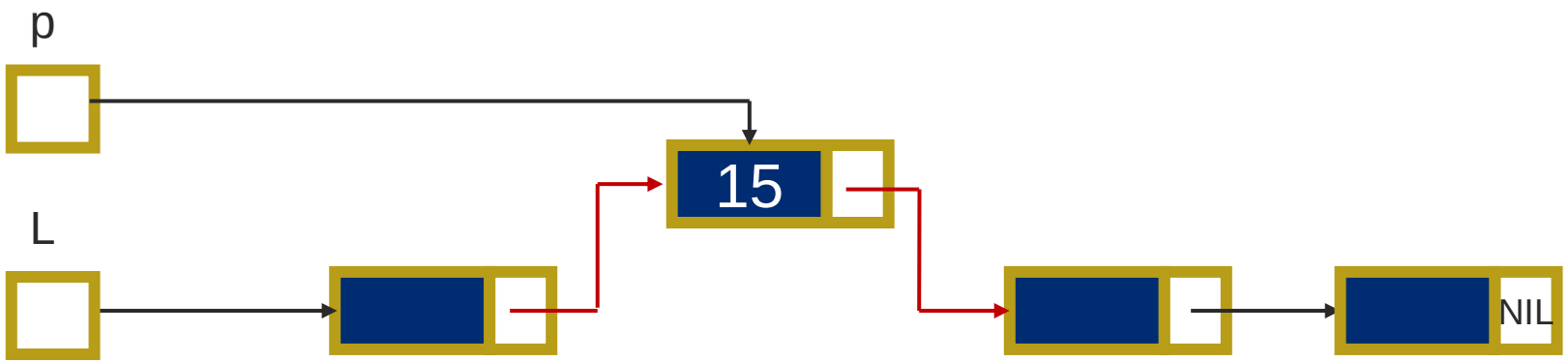


# Listaelem beszúrása

- Új listaelem létrehozás és beállítása
  - Deklarálás – p változó, Node típussal
  - Létrehozás – new(p)
  - Értékének megadása
  - Új elem befűzése a láncolatba

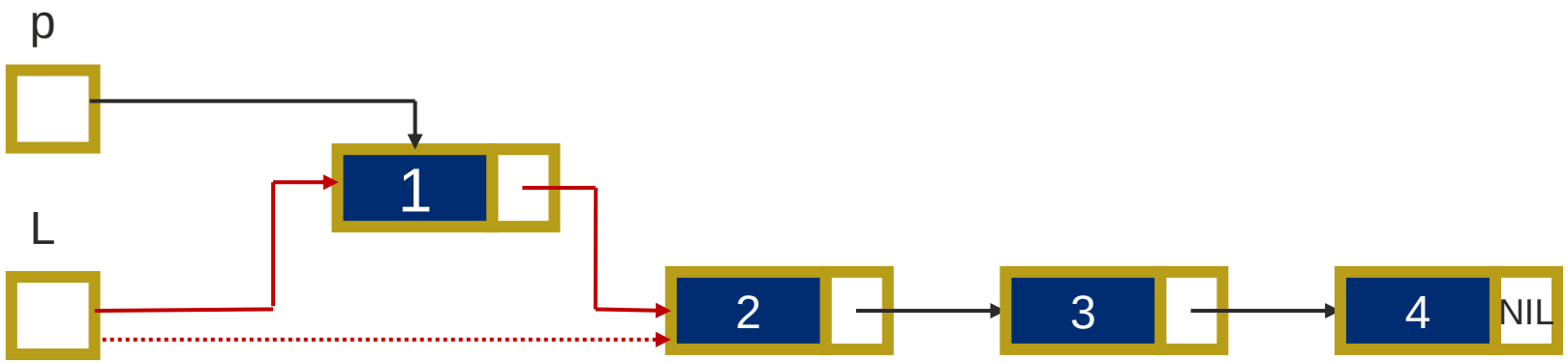
# Listaelem beszúrása

- Új listaelem létrehozás és beállítása
  - Deklarálás – p változó, Node típussal
  - Létrehozás – new(p)
  - Értékének megadása
  - Új elem befűzése a láncolatba



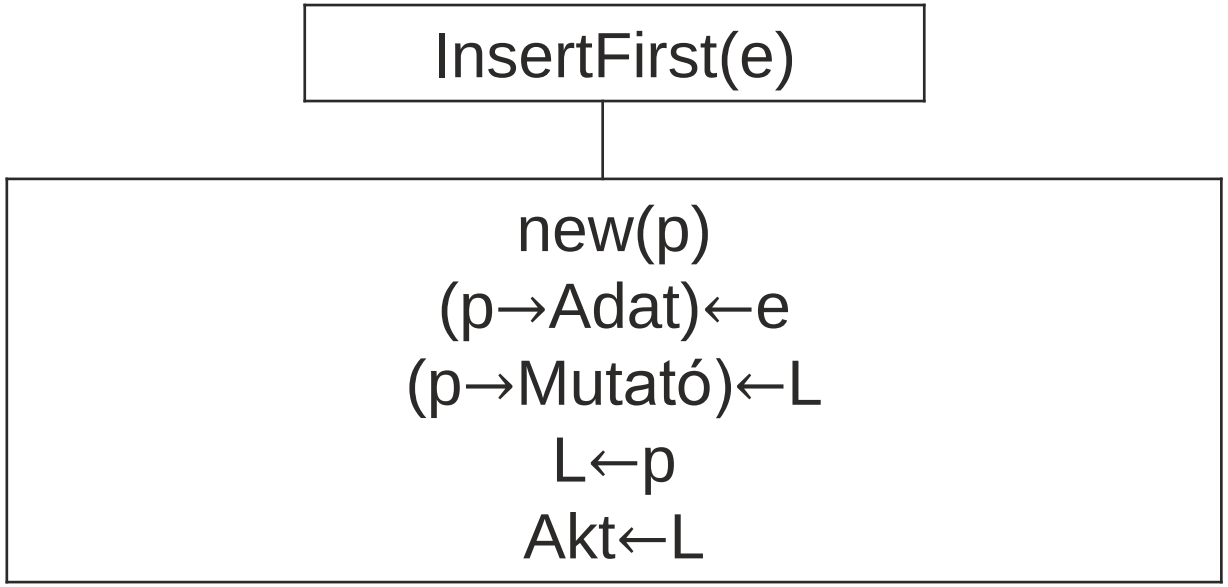
# Listaelem beszúrása

- Beszúrás első elemként
  - Üres és nem üres listára is működik
  - Az Akt mutatót (aktuális elem) az újonnan beszúrtra állítja, ami az első



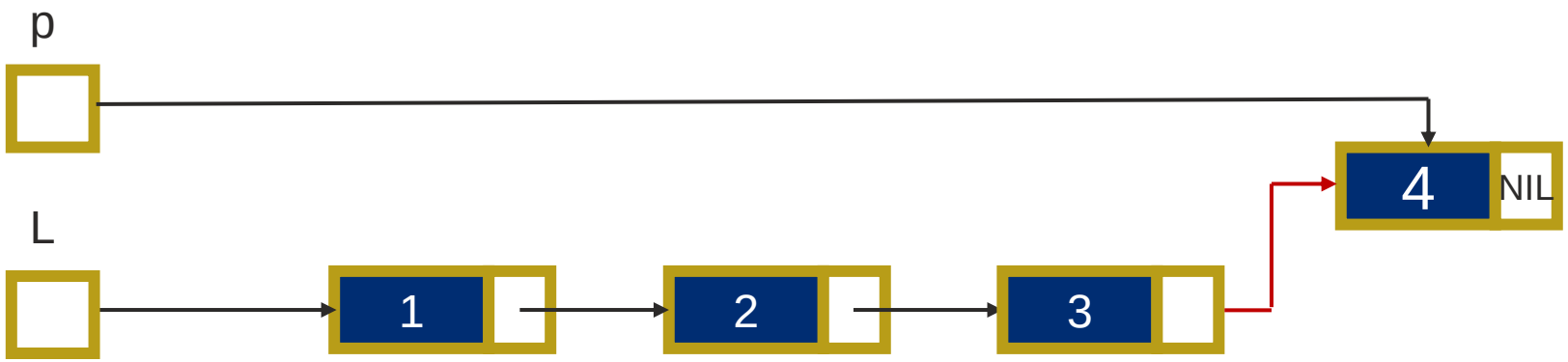
# Listaelem beszúrása

- „e” adatelem beszúrása első elemként
  - Üres és nem üres listára is működik
  - Az Akt mutatót (aktuális elem) az újonnan beszúrtra állítja, ami az első



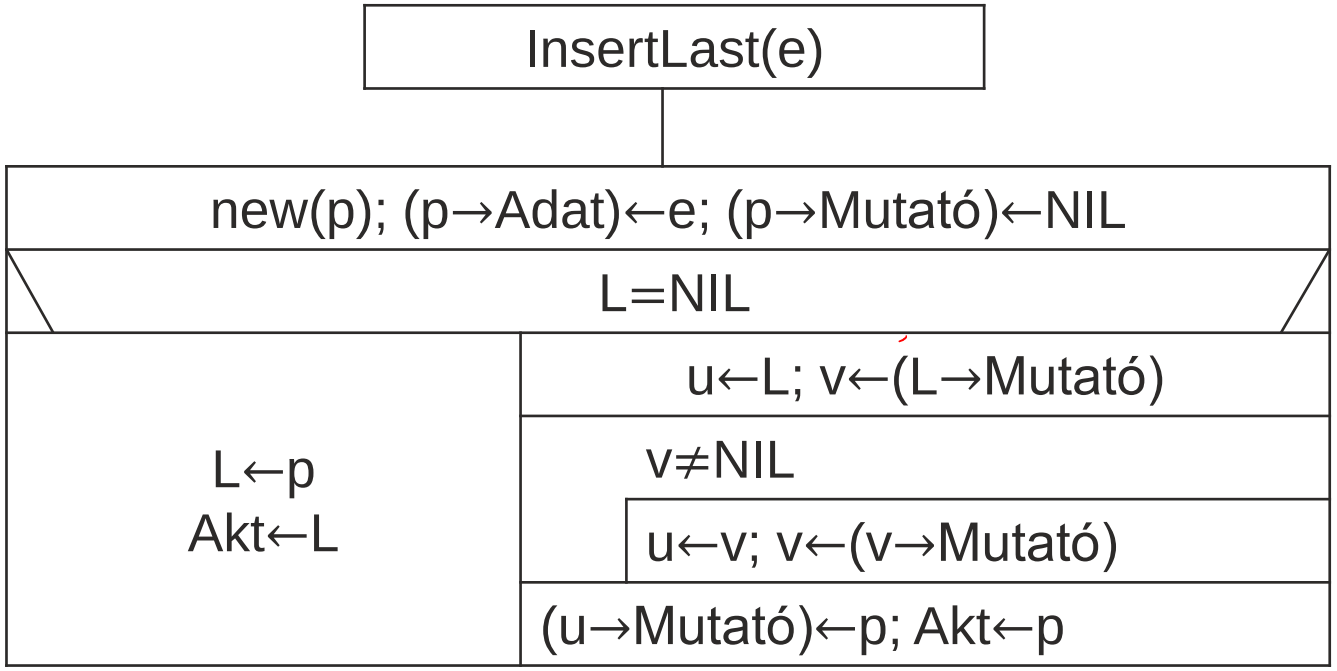
# Listaelem beszúrása

- Beszúrás utolsó elemként
  - Üres és nem üres listára is működik
  - Az Akt mutatót (aktuális elem) az újonnan beszúrtra állítja, ami az utolsó



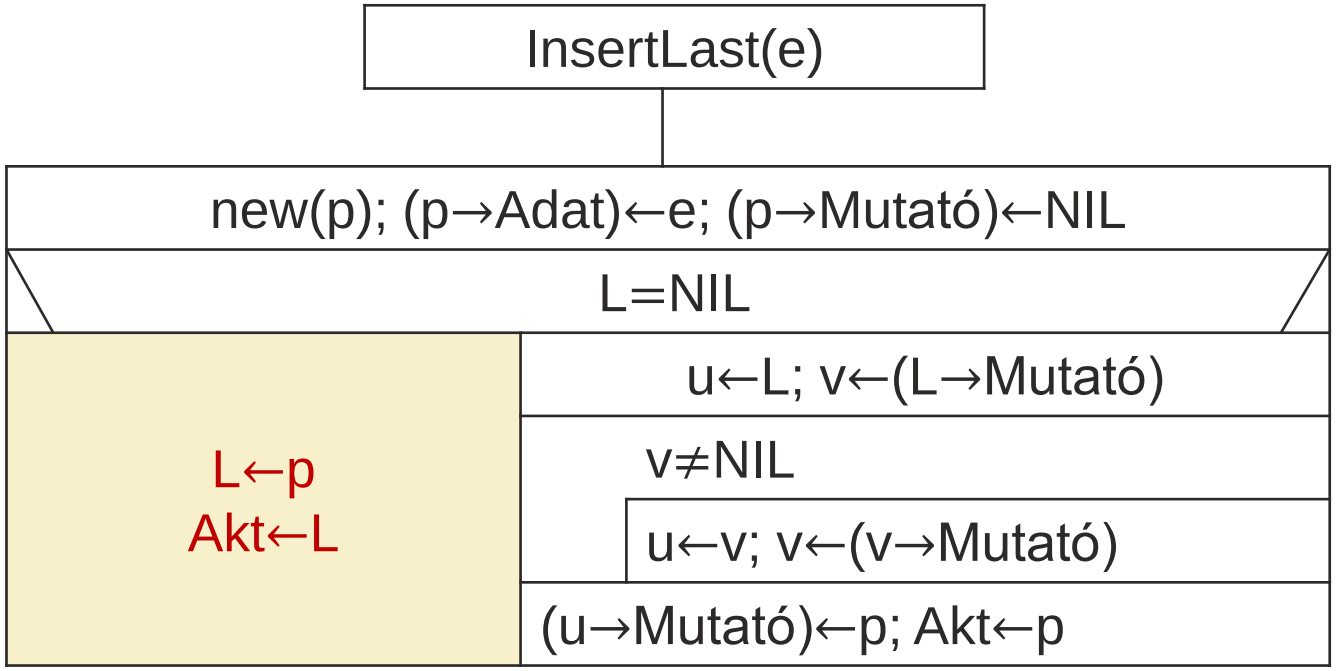
# Listaelem beszúrása

- „e” adatelem beszúrása utolsó elemként
  - Üres és nem üres listára is működik
  - Az Akt mutatót (aktuális elem) az újonnan beszúrtra állítja, ami az utolsó



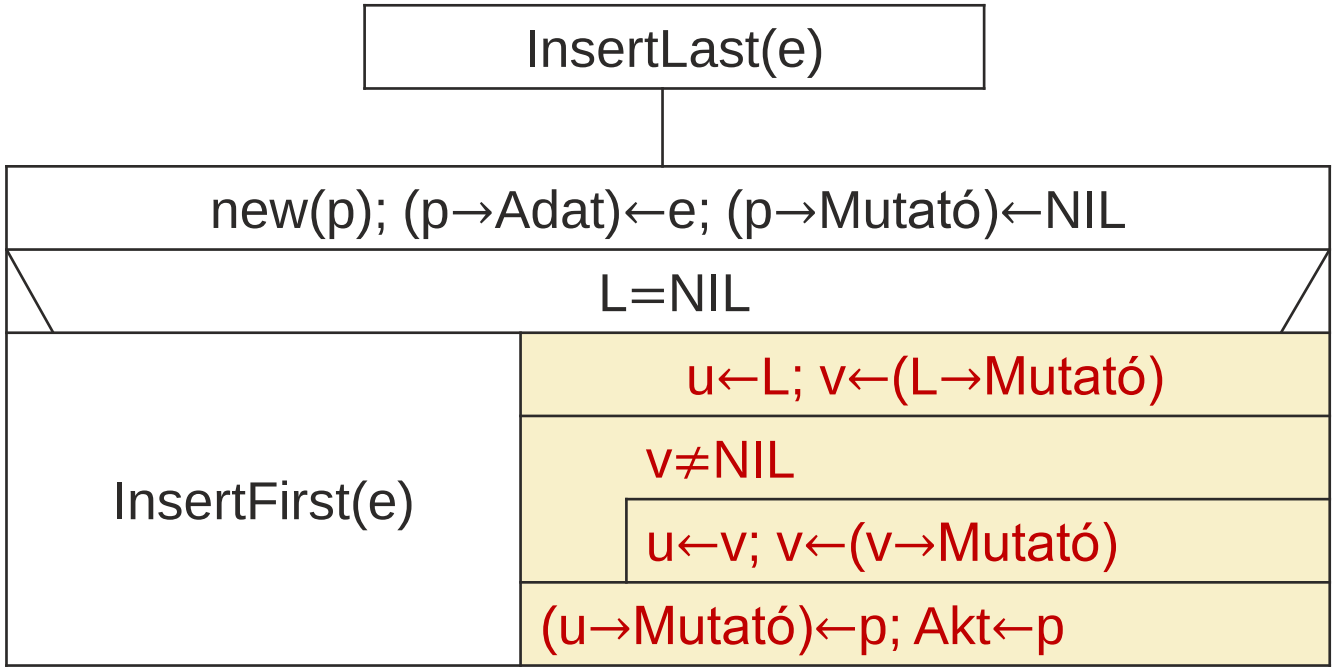
# Listaelem beszúrása

- Ha a lista üres
  - Ami egyezik az InsertFirst(e) algoritmusával
    - Azonban itt nem lehet behelyettesíteni, mert az új csomópont már létrejött előbb



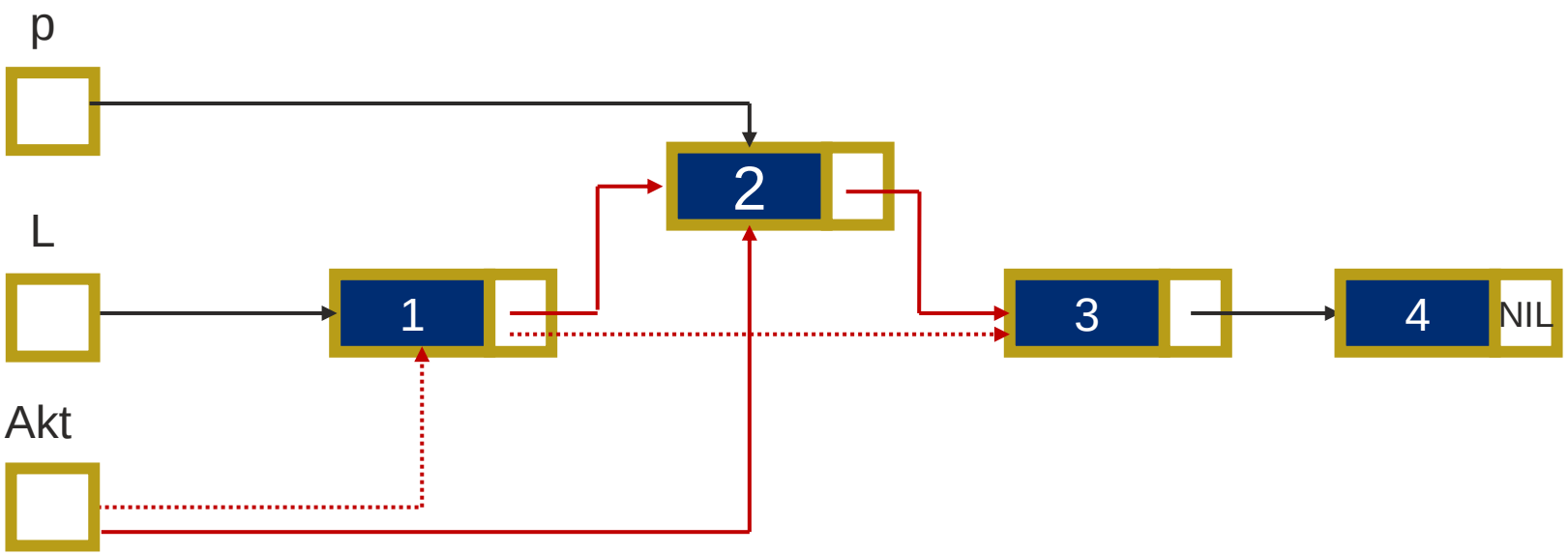
# Listaelem beszúrása

- Ha a lista nem üres
  - Meg kell keresni az eredeti lista utolsó elemét
  - Amögé kell beszúrni az új elemet



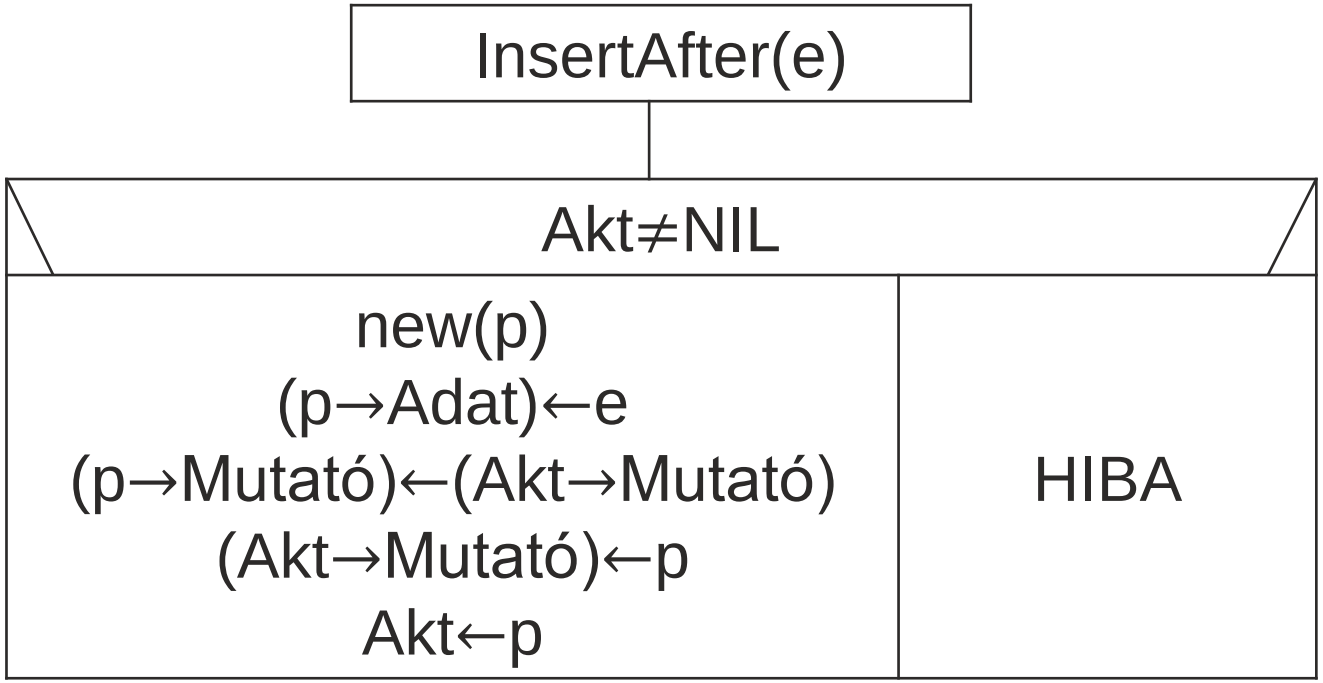
# Listaelem beszúrása

- Beszúrás az aktuális elem után



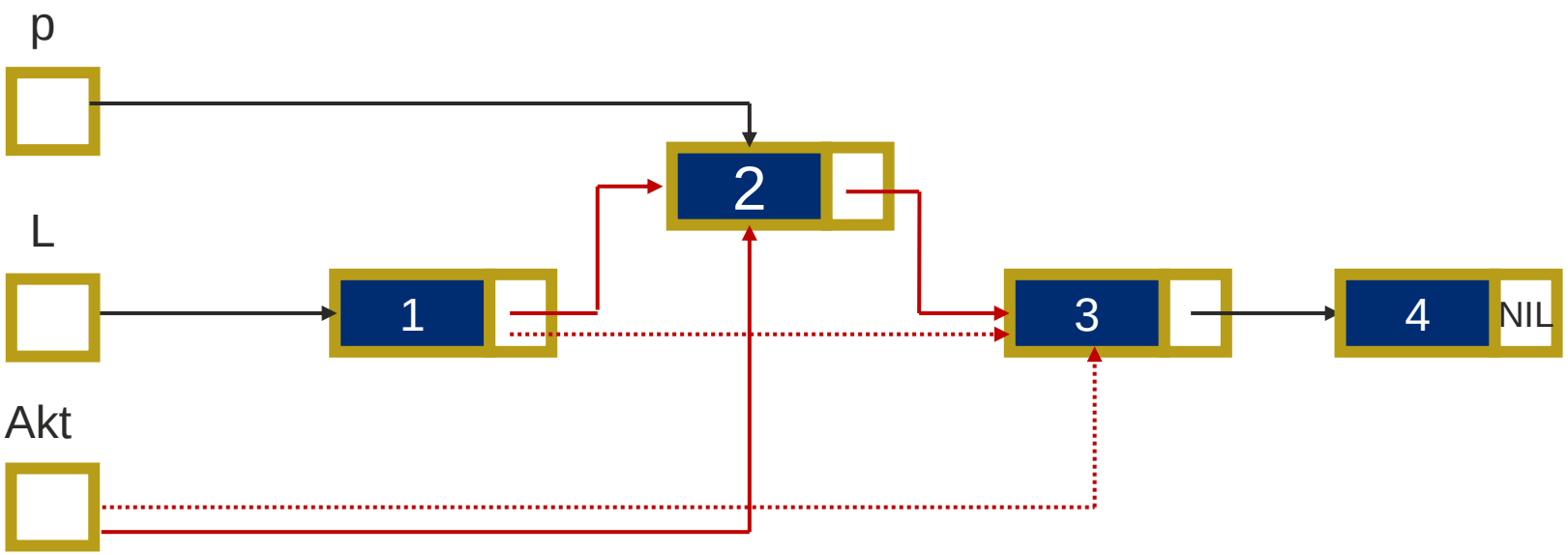
# Listaelem beszúrása

- Beszúrás az aktuális elem után
  - Ha nincsen aktuális elem, az hiba
  - Az újonnan beszúrt lesz az aktuális elem



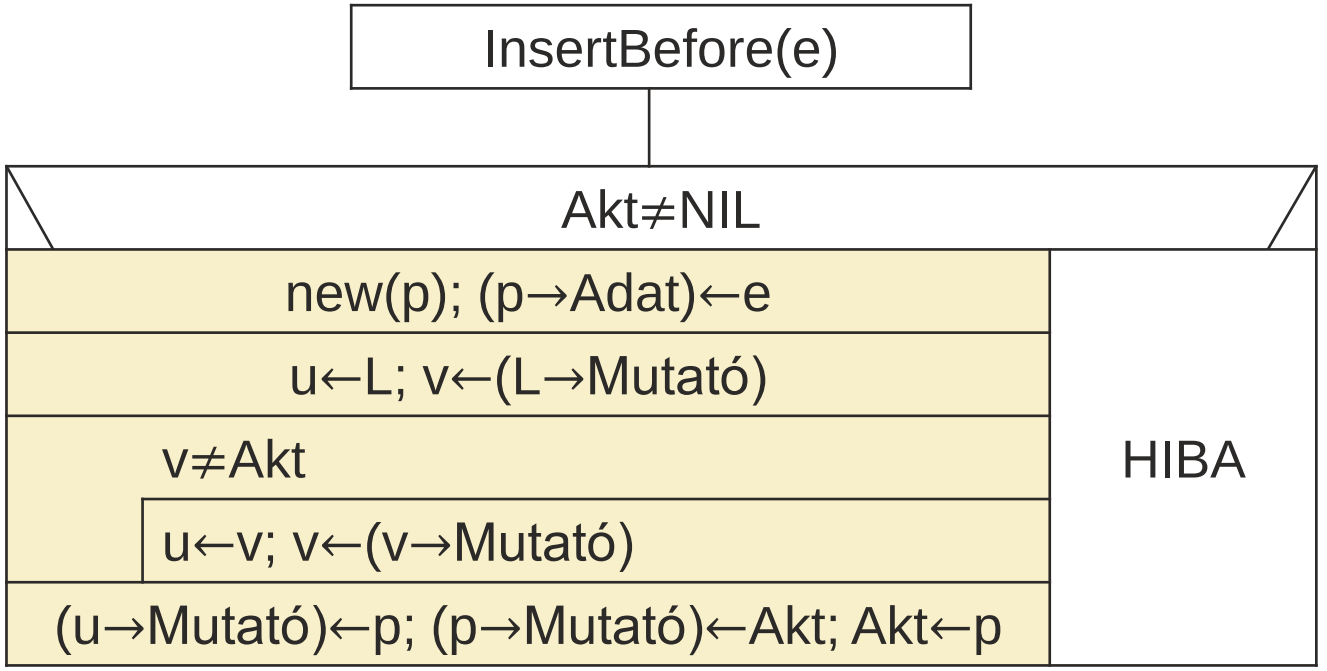
# Listaelem beszúrása

- Beszúrás az aktuális elem elé



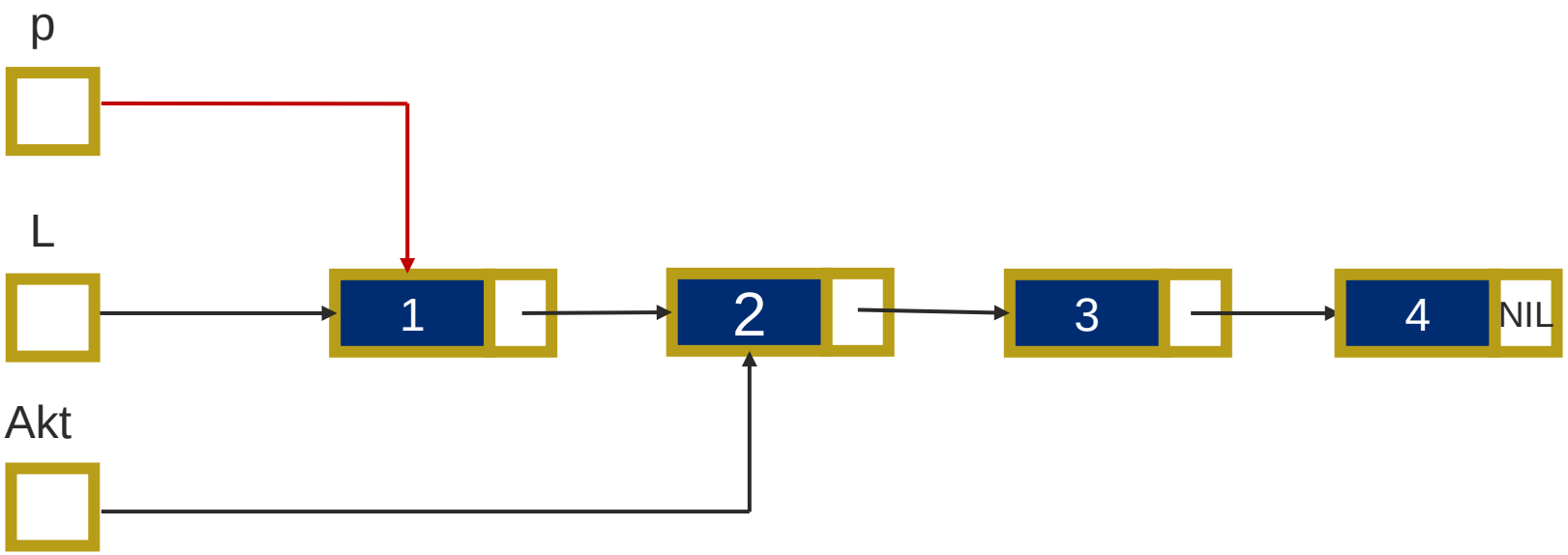
# Listaelem beszúrása

- Beszúrás az aktuális elem elé
  - Ha nincsen aktuális elem, az hiba
  - Az újonnan beszúrt lesz az aktuális elem



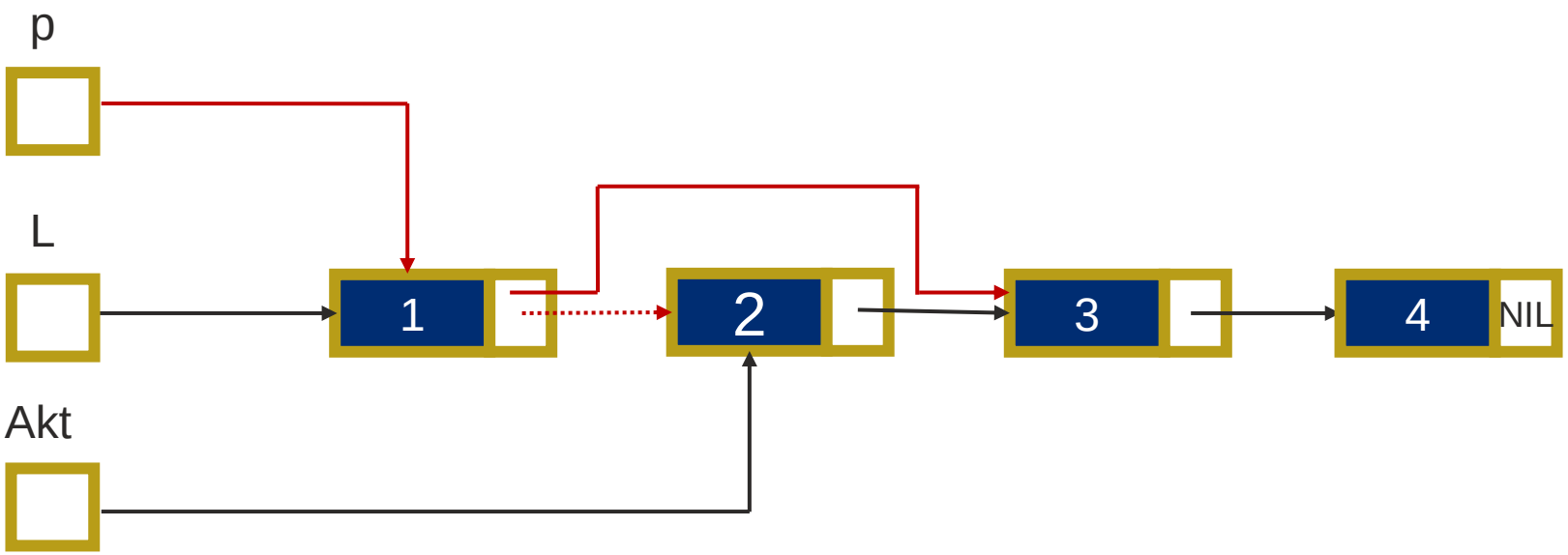
# Listaelem törlése

- Aktuális elem törlése
  - Törlendő elem megelőzőjének megkeresése
  - Láncolás megváltoztatása (átláncolás)



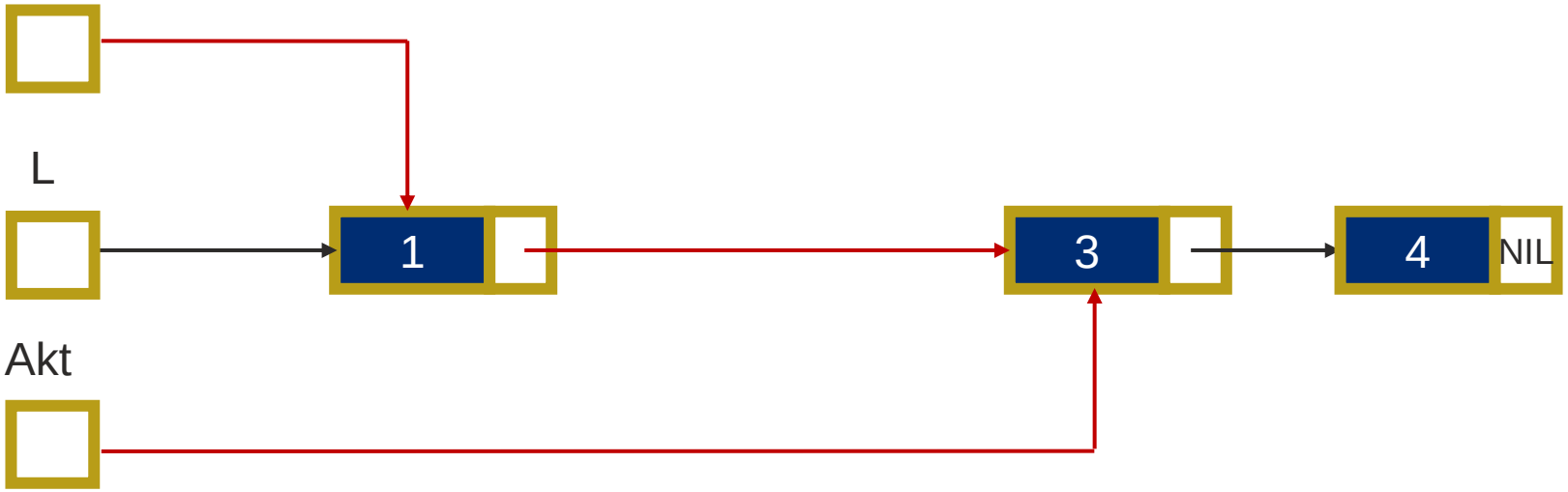
# Listaelem törlése

- Aktuális elem törlése
  - Törlendő elem megelőzőjének megkeresése
  - Láncolás megváltoztatása (átláncolás)



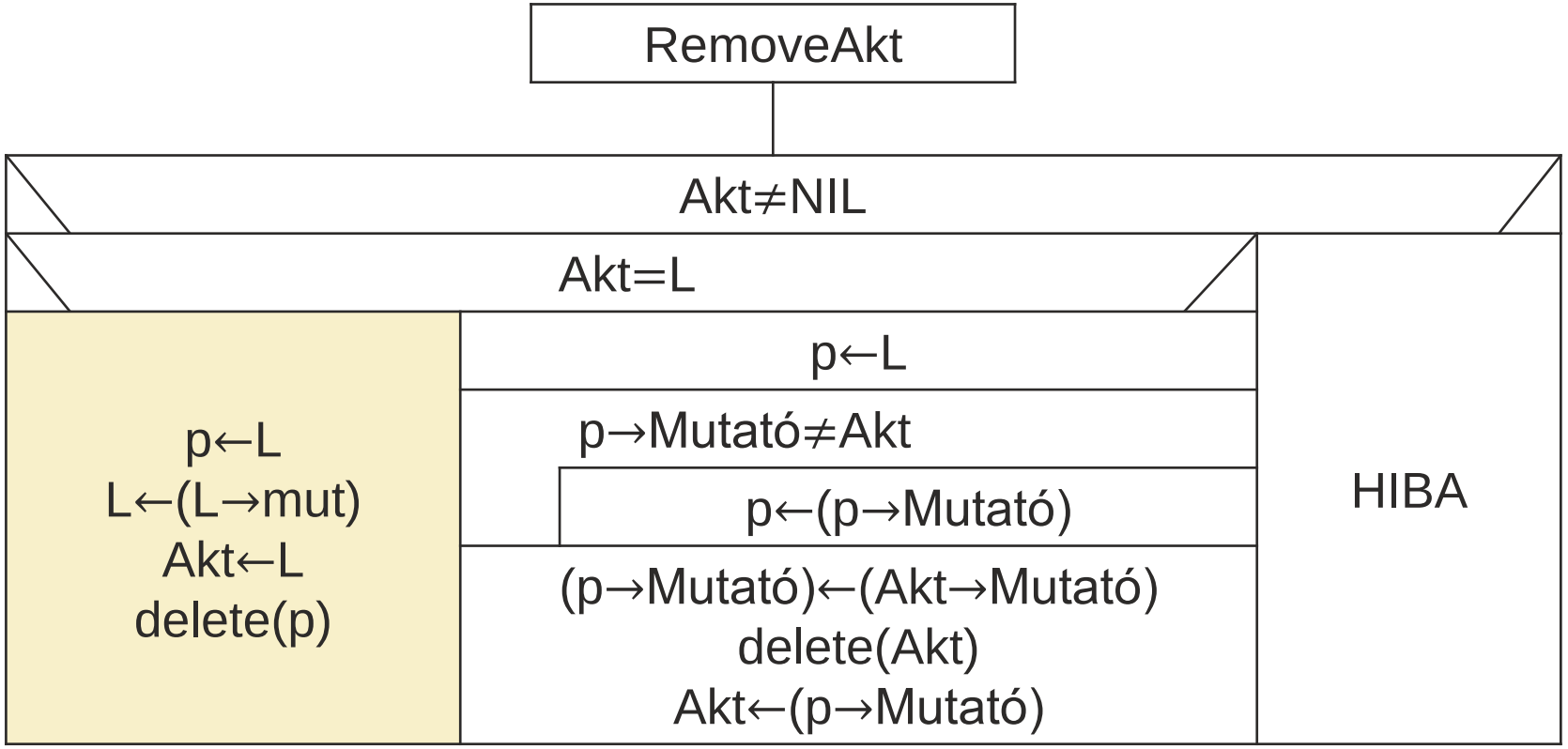
# Listaelem törlése

- Aktuális elem törlése
  - Törlendő elem megelőzőjének megkeresése
  - Láncolás megváltoztatása (átláncolás)
  - Memóriából eltávolítás (törlés)
  - Akt beállítása



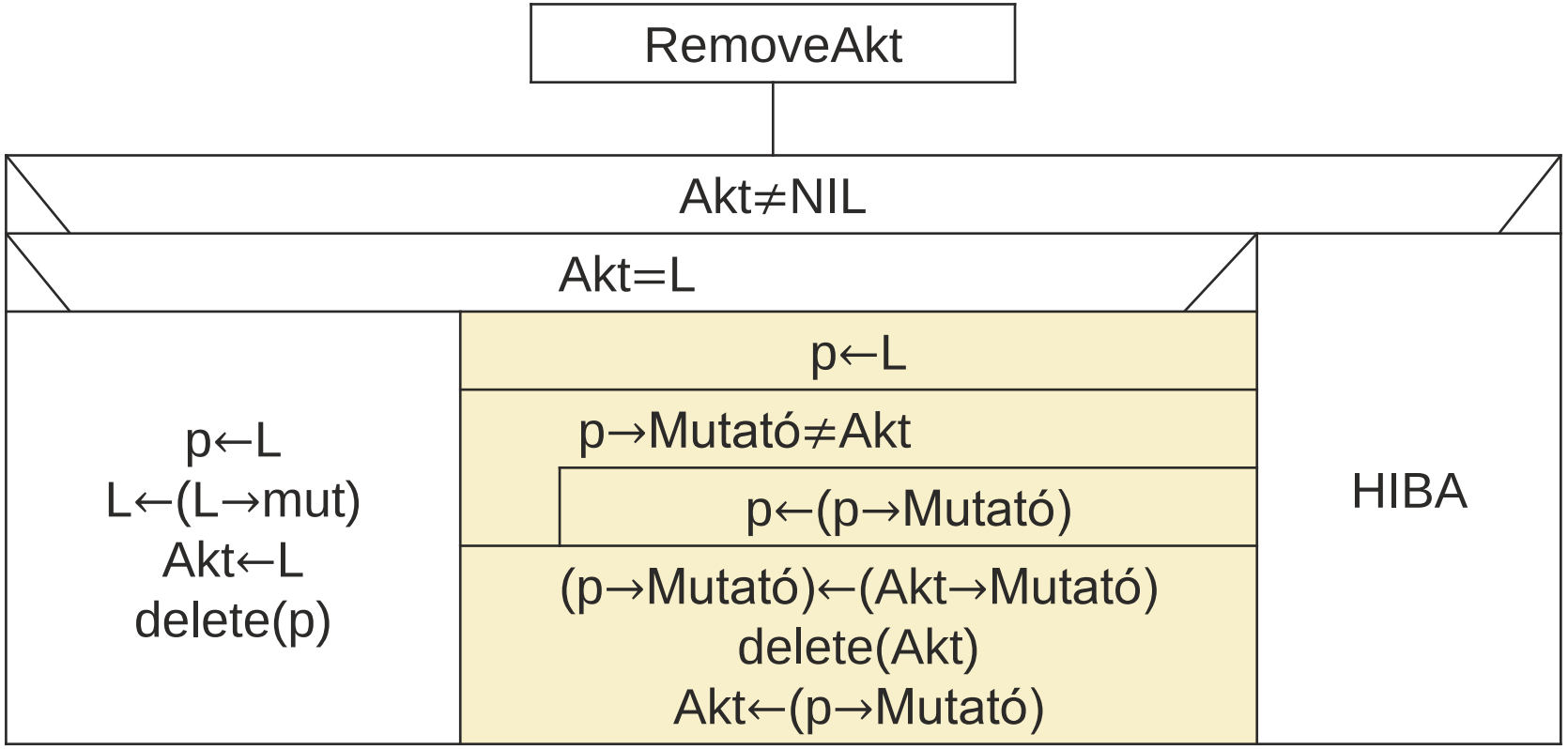
# Listaelem törlése

- Aktuális elem törlése
  - Ha az aktuális az első



# Listaelem törlése

- Aktuális elem törlése
  - Ha az aktuális nem az első



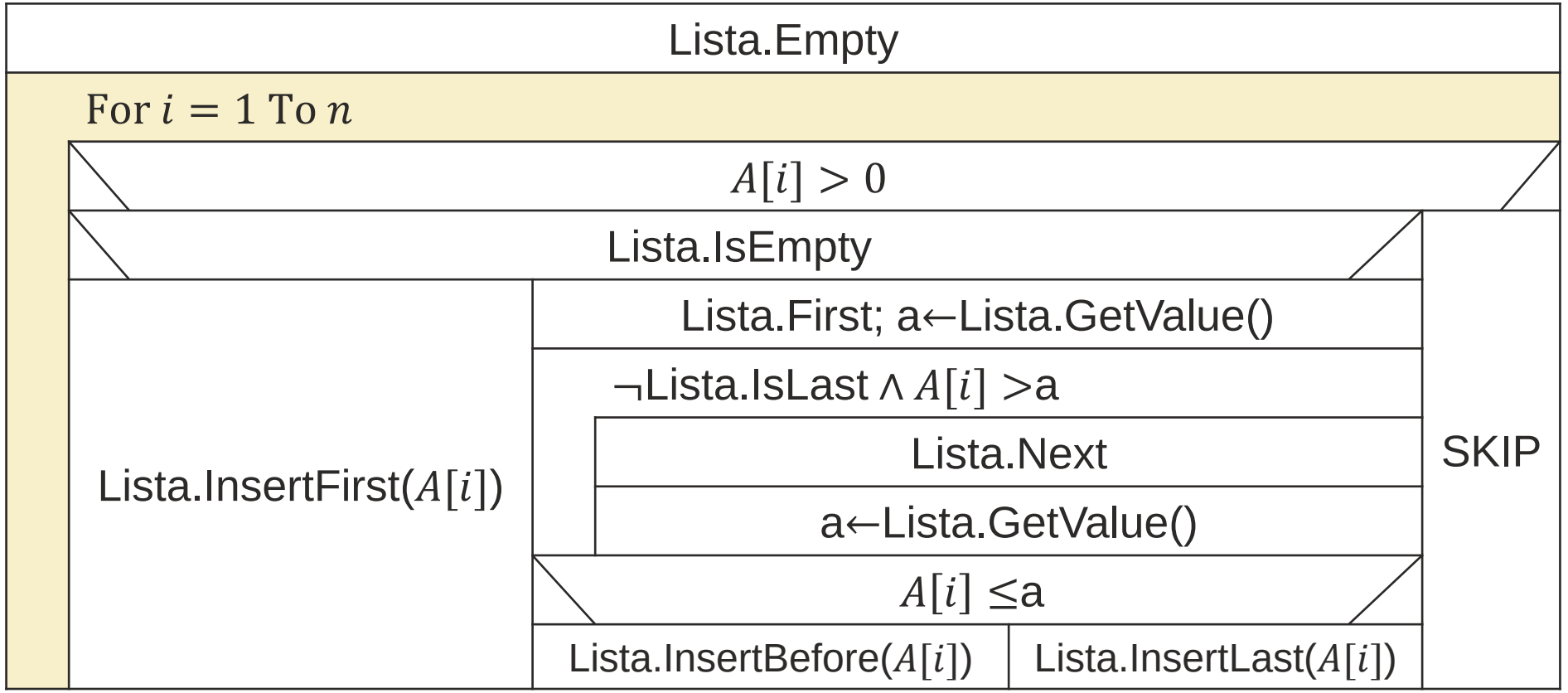
# Egyszerű lista – Műveletek

- Megjegyzések – lehetőségek
  - Az Akt nem változik a módosítás során
  - További műveletekre példa
    - Teljes lista törlése
    - Listák összefűzése,
    - Elemszám lekérdezése
  - Ebben az implementációban nem hatékony a megvalósítás
    - Last, Remove, InsertBefore, InsertLast
  - Hatékonnyá tehető
    - Kétirányú láncolással

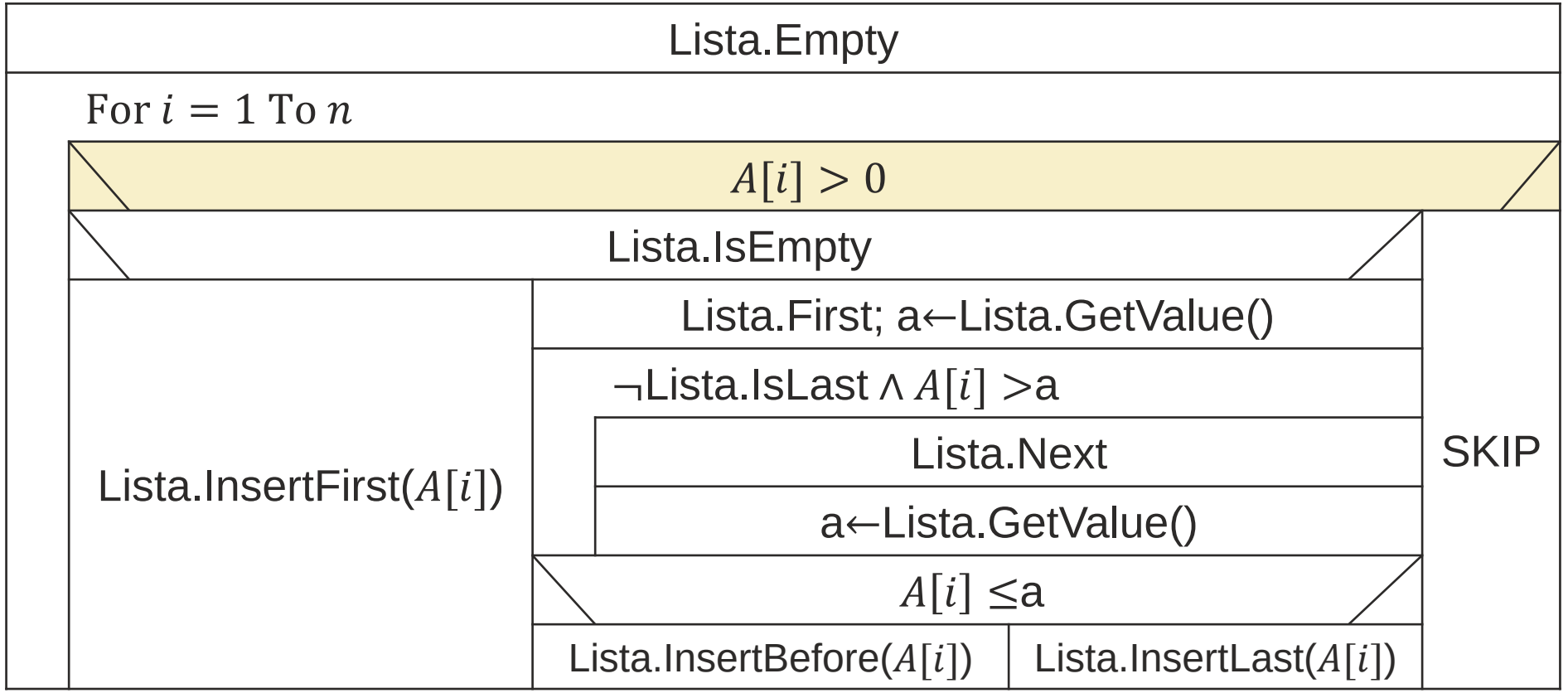
# Példa

- Elemek sorbarendezése lista használatával
  - Adott az $A[1..n]$ egészeket tartalmazó tömb
  - Helyezzük el a pozitív elemeit rendezett módon egy listába

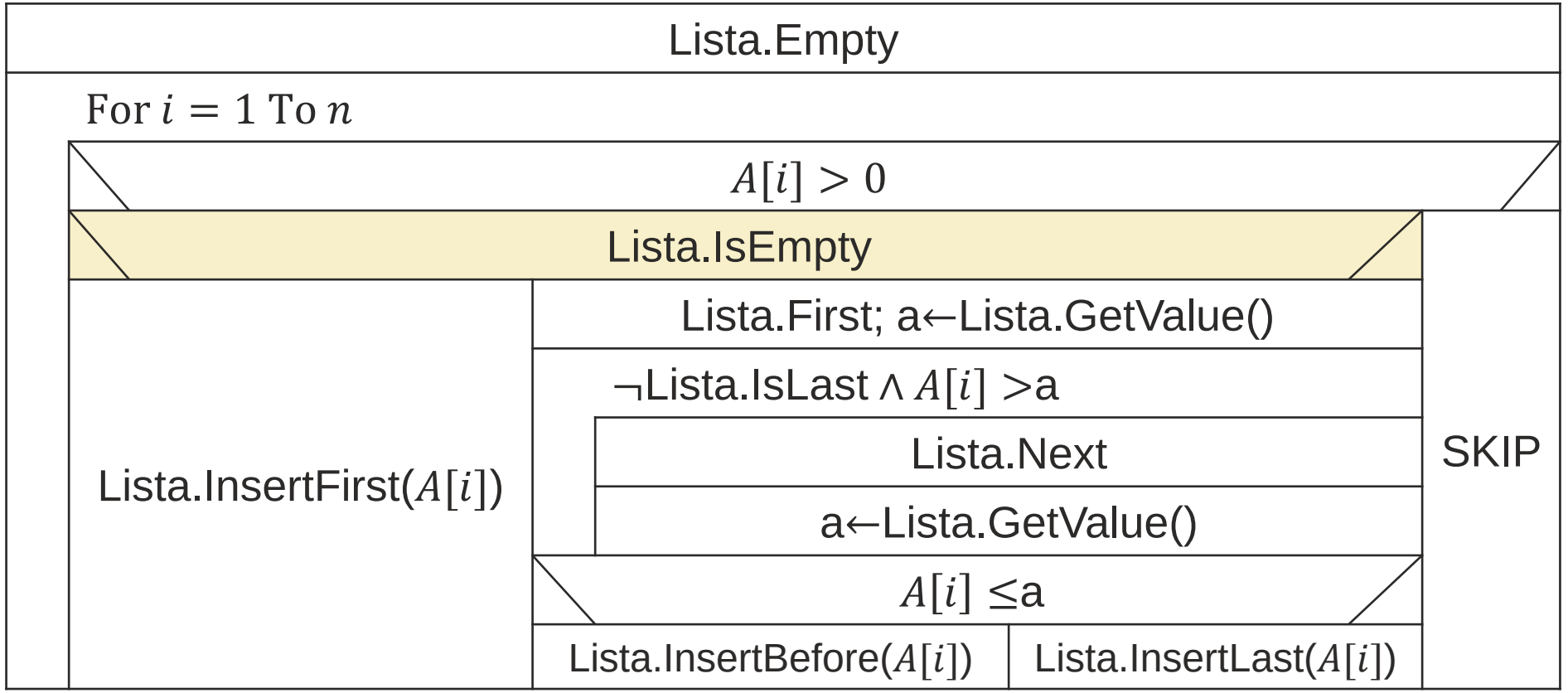
# Az algoritmus



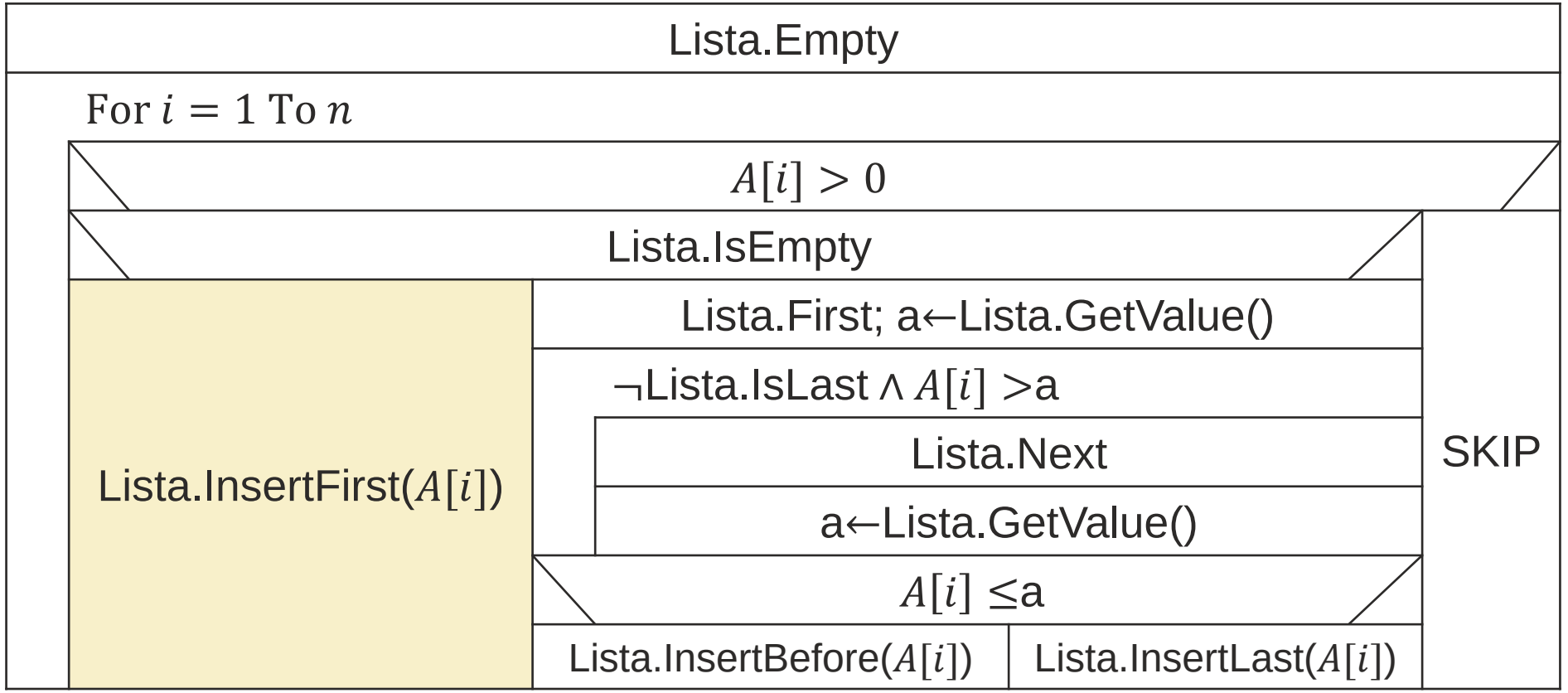
# Az algoritmus



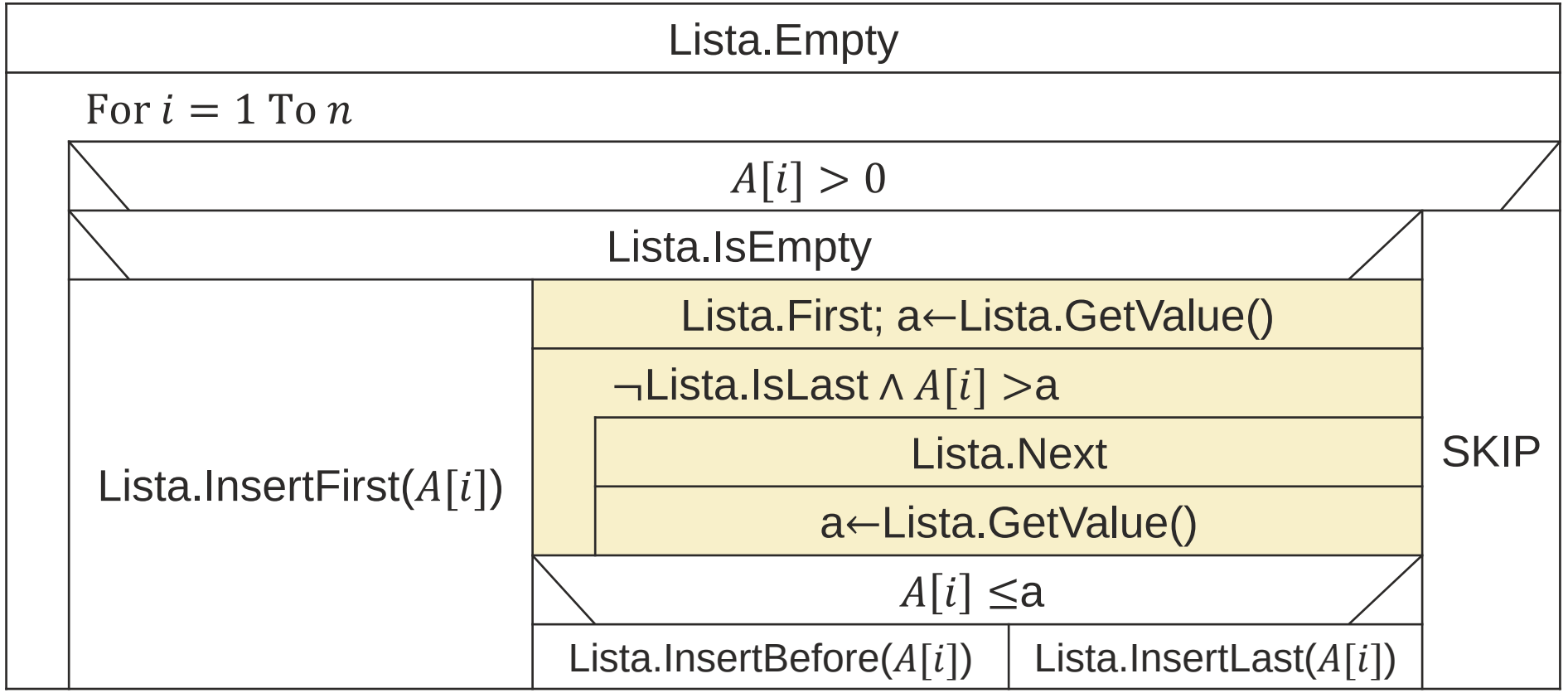
# Az algoritmus



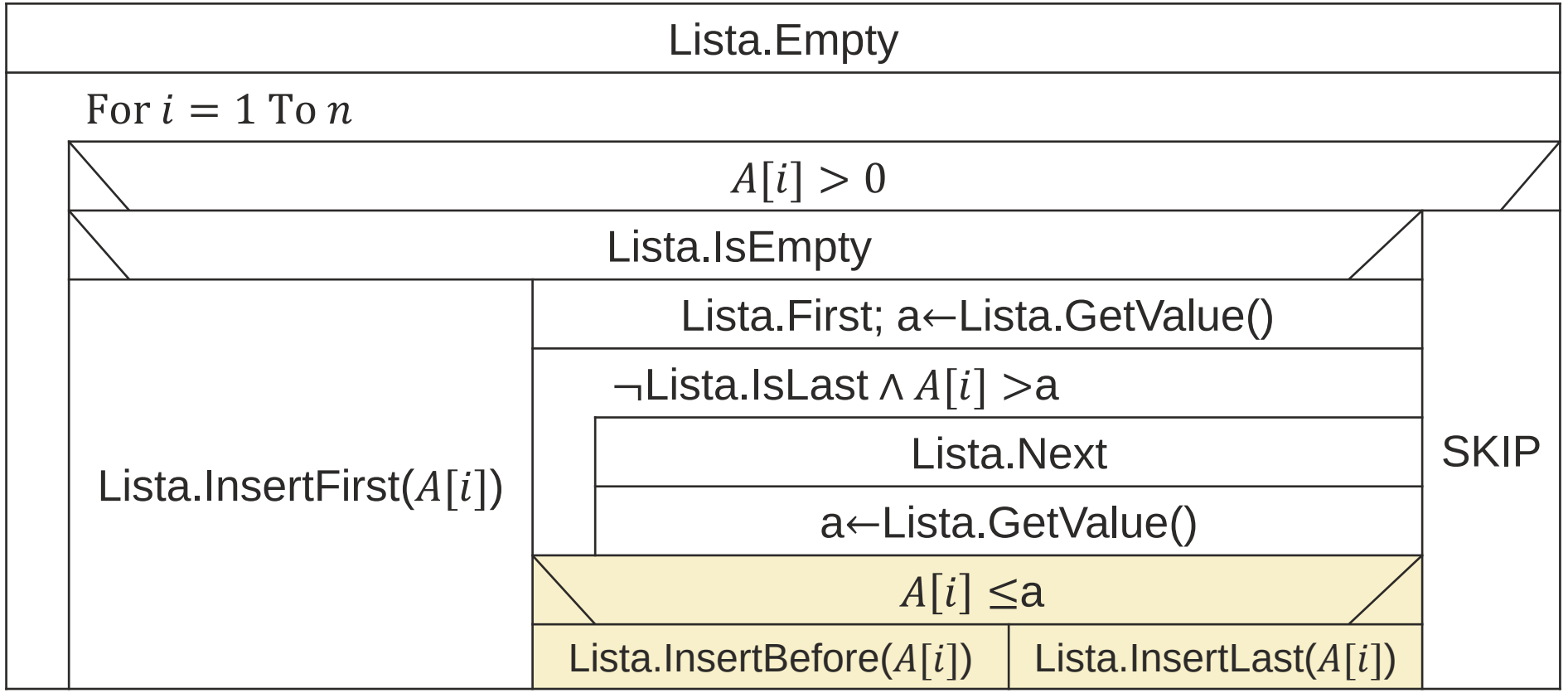
# Az algoritmus



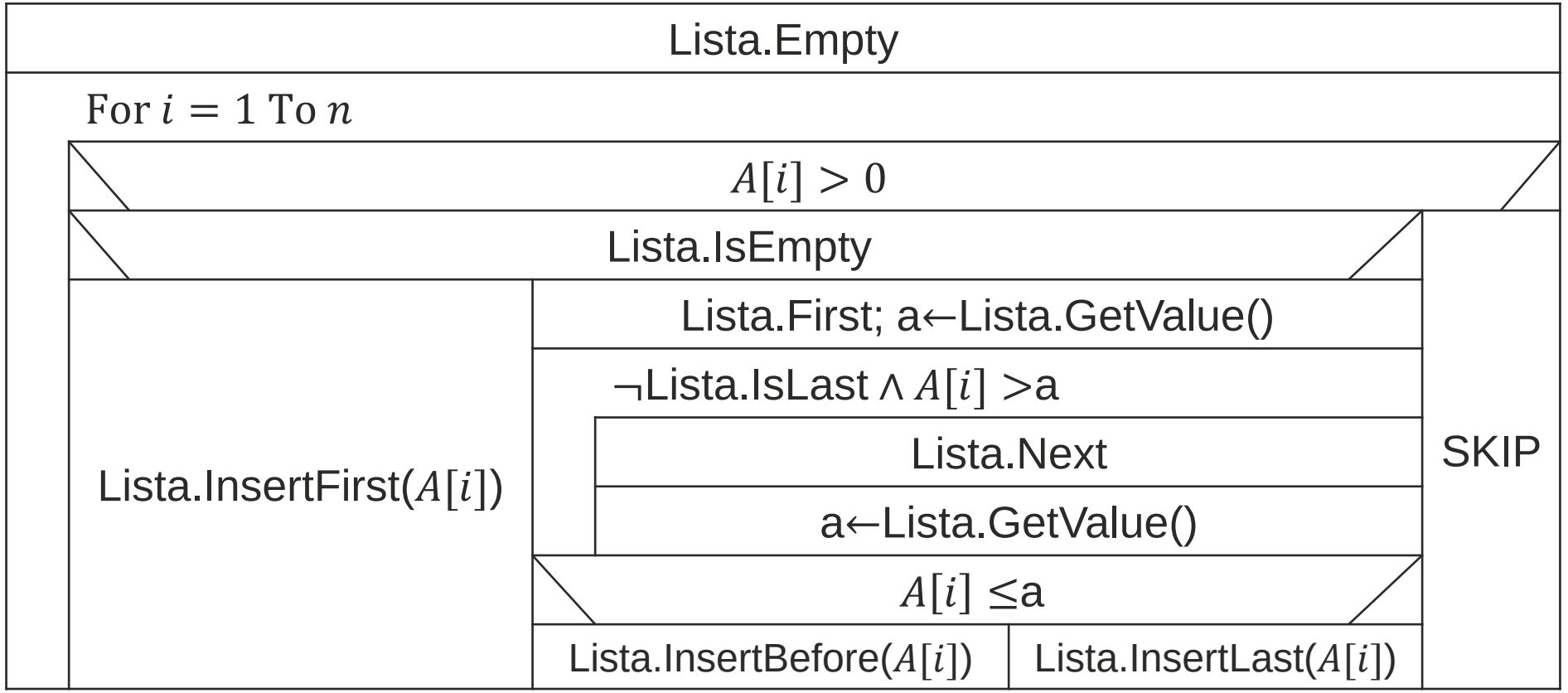
# Az algoritmus



# Az algoritmus



# Az algoritmus



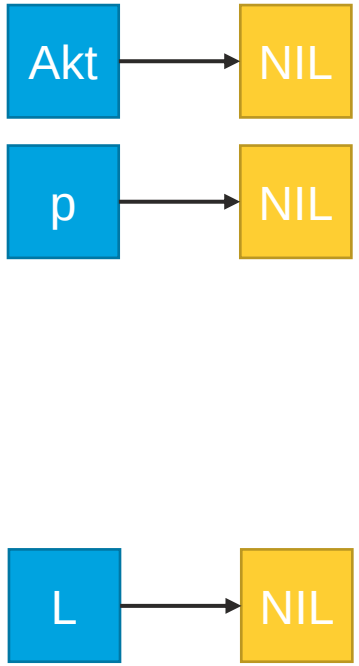
# Hierarchikus adatszerkezetek

Következő téma
De előtte még egy animáció az egyszeresen láncolt listához

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

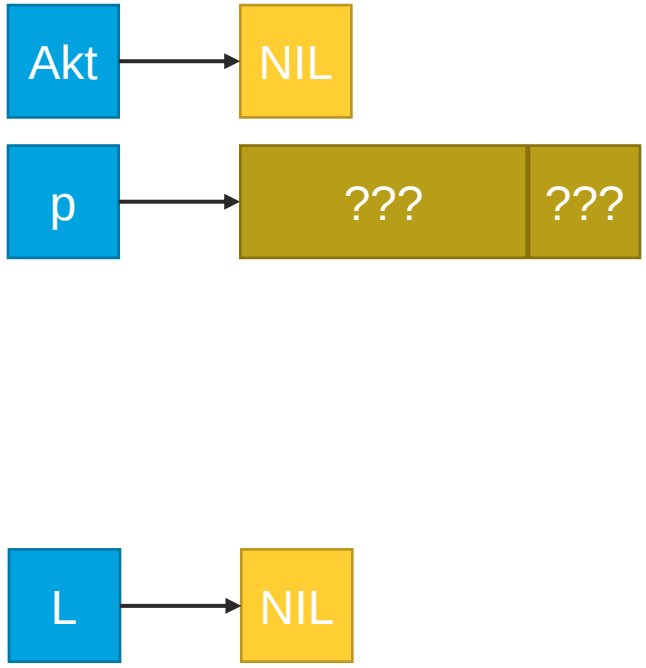



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

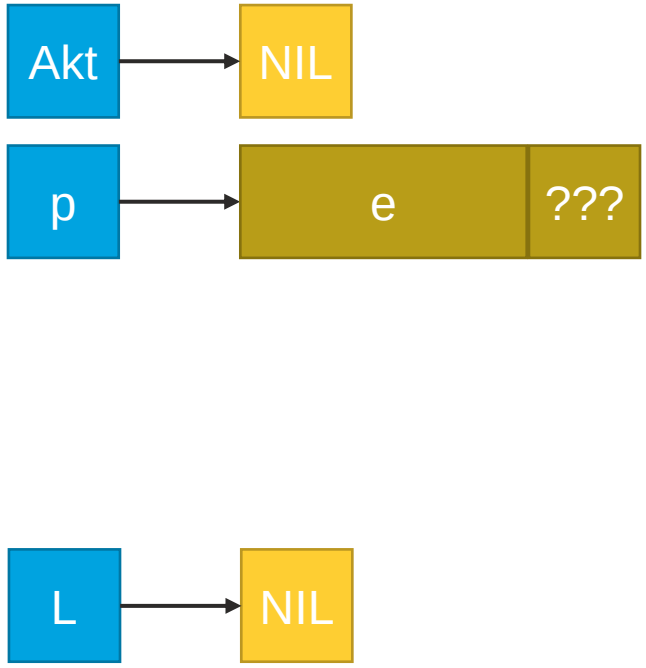



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

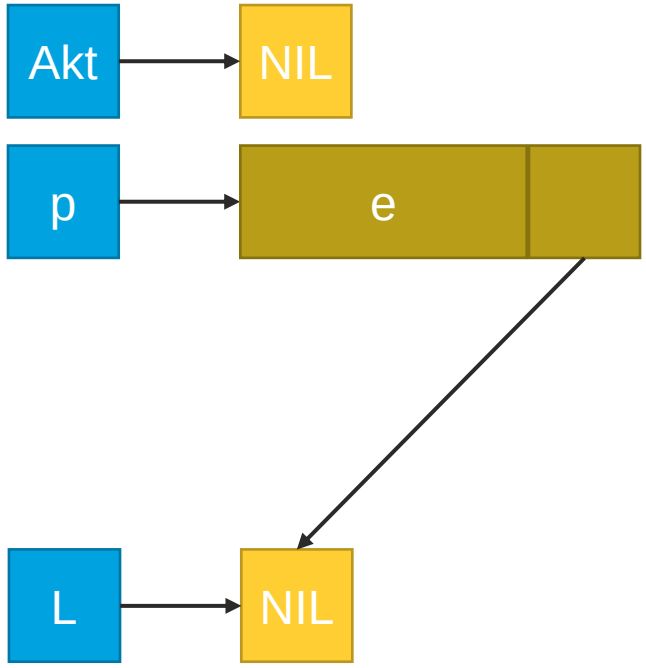



| new(p) |
|---|
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

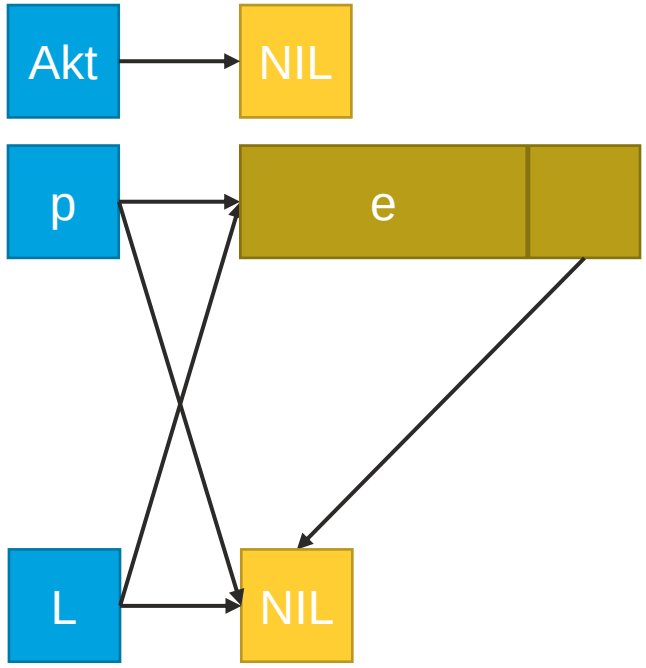



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

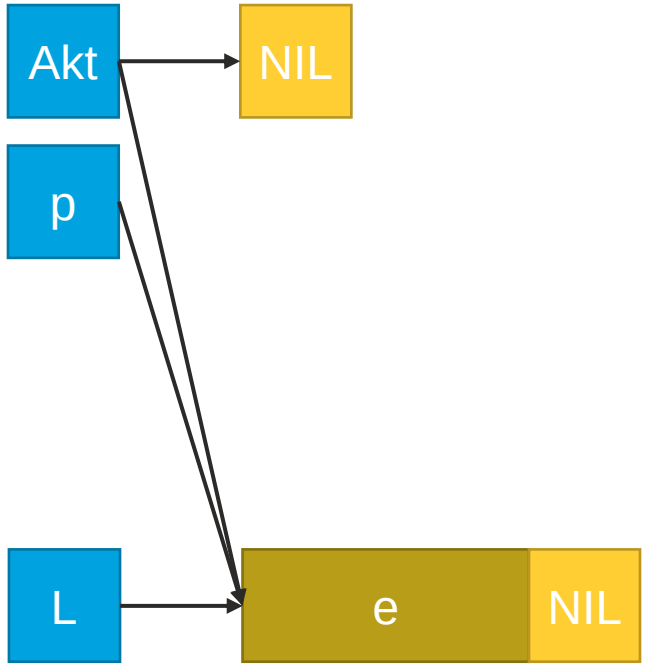



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- L jelenti a lista első elemére a mutatót
- Az iniciális állapotban a lista üres

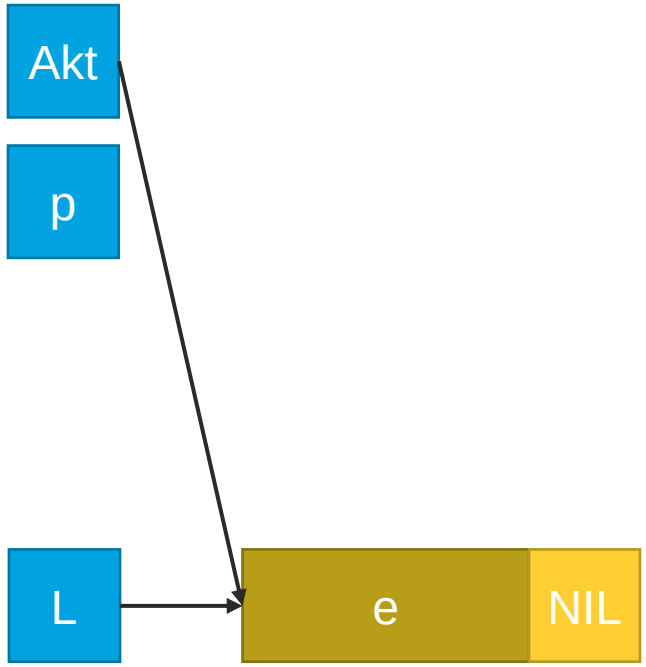



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

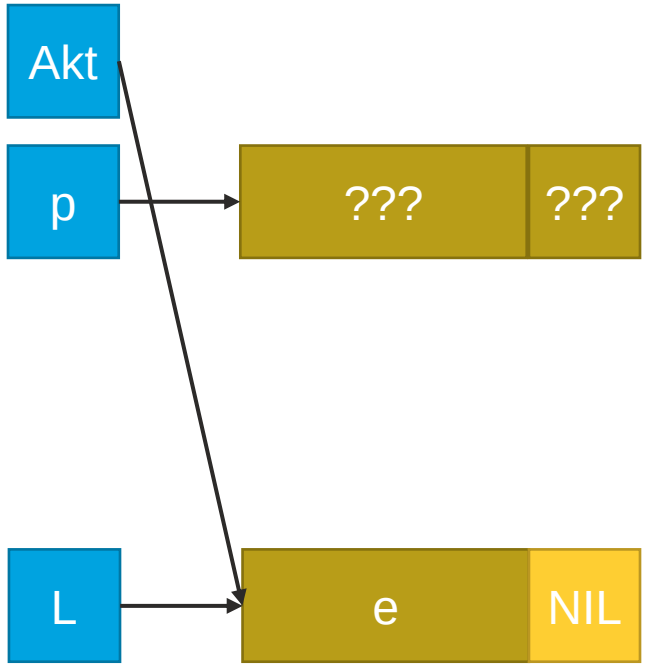



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

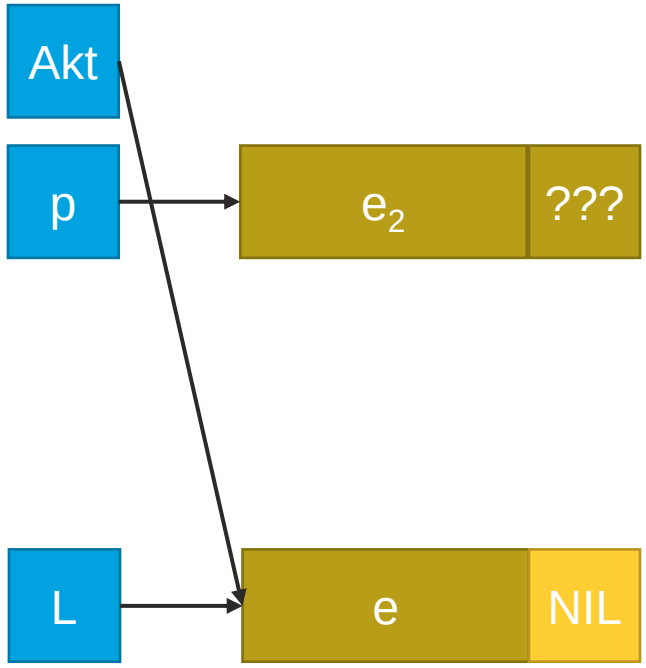



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

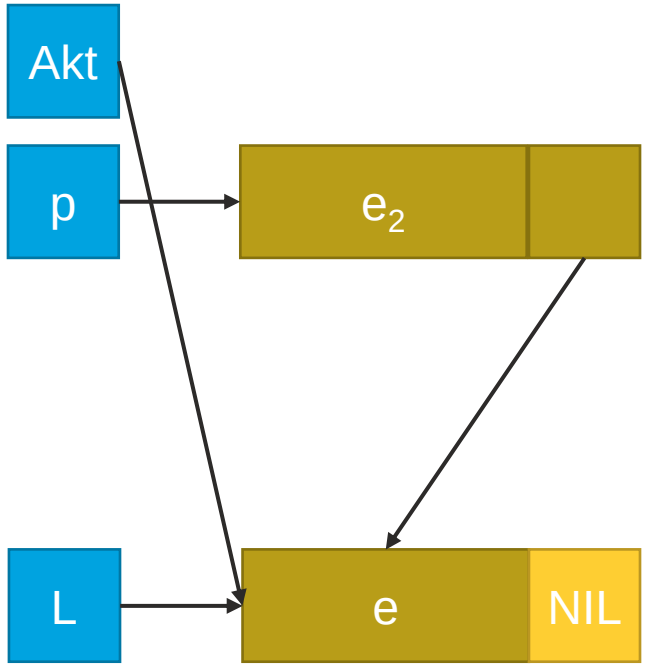



| new(p) |
|---|
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

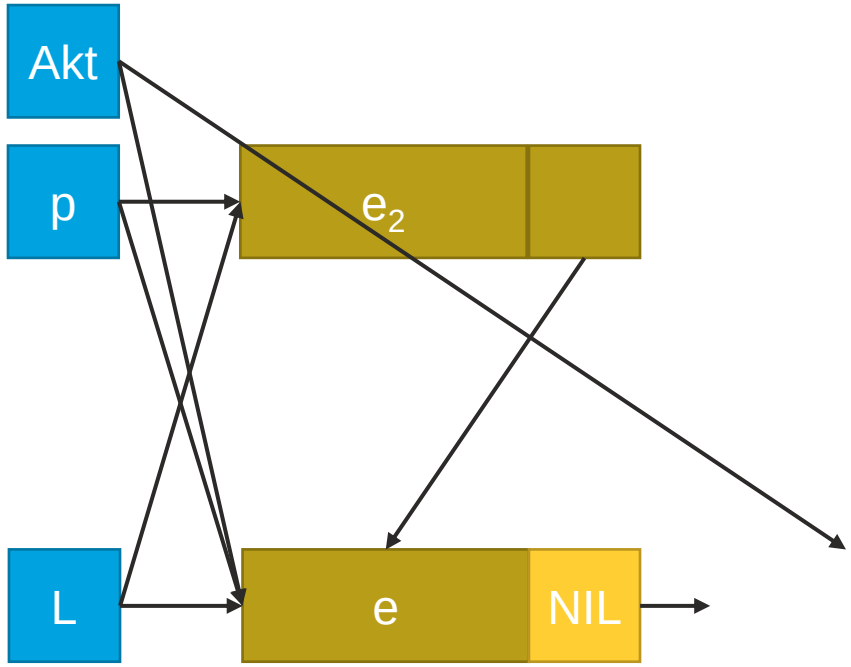



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

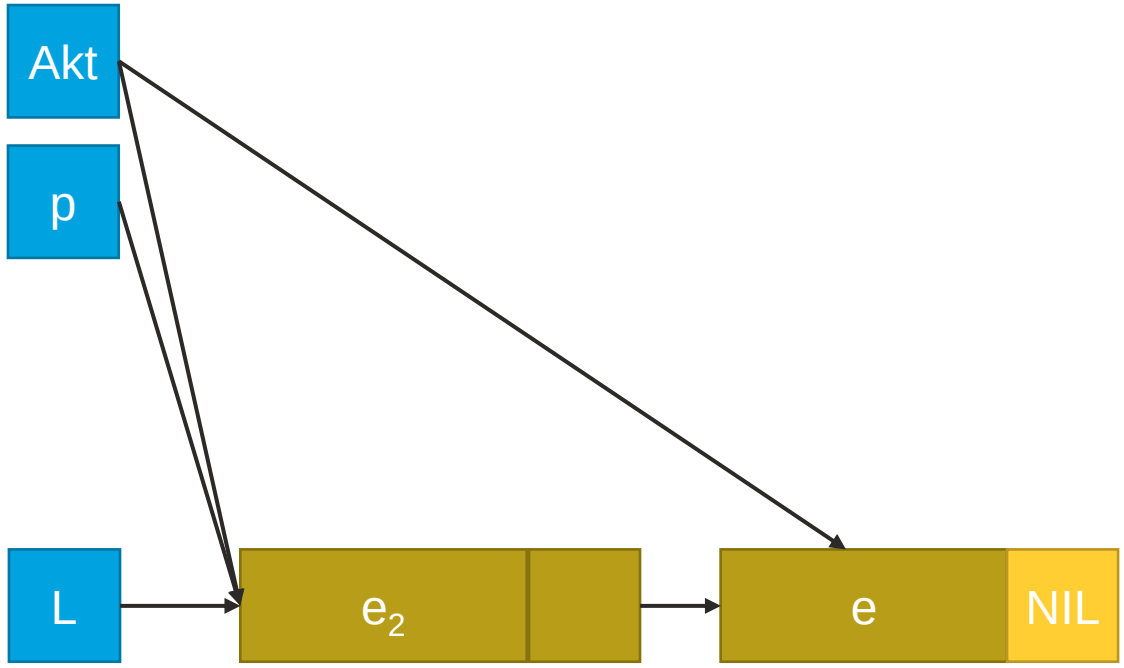



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertFirst(e)

Elem beszúrása az első helyre

- Vegyük észre, hogy a beszúrás algoritmusa üres és nem üres listára is azonos
- Szúrjunk be még egy elemet, az első helyre

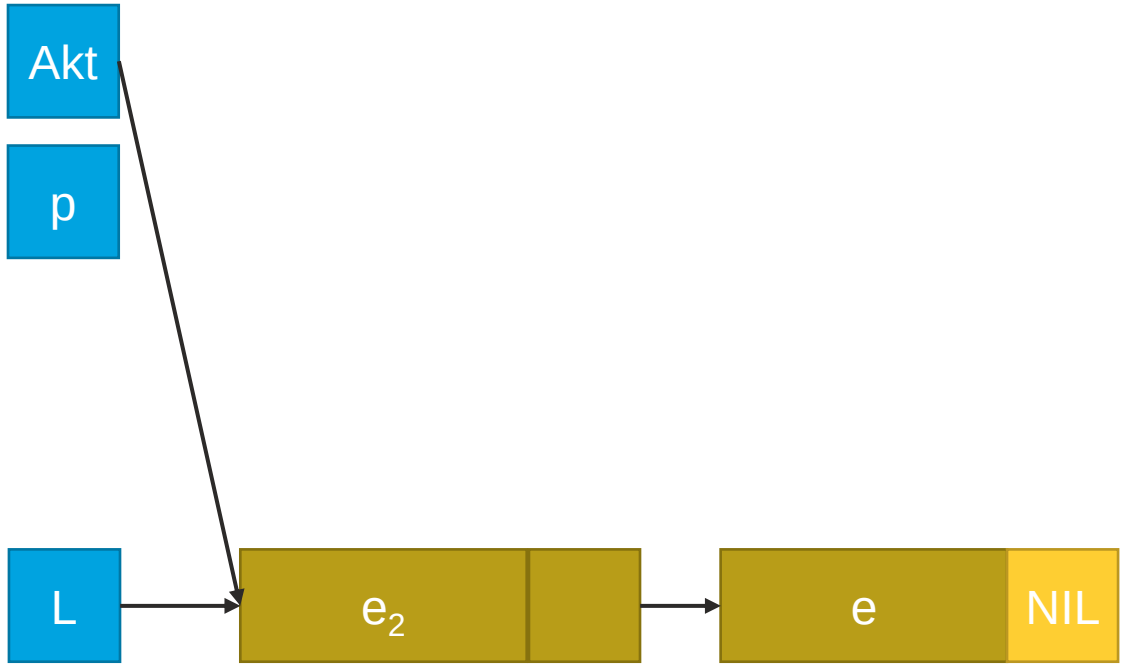



| new(p) |
| --- |
| (p→Adat)←e |
| (p→Mutató)←L |
| L←p |
| Akt←L |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

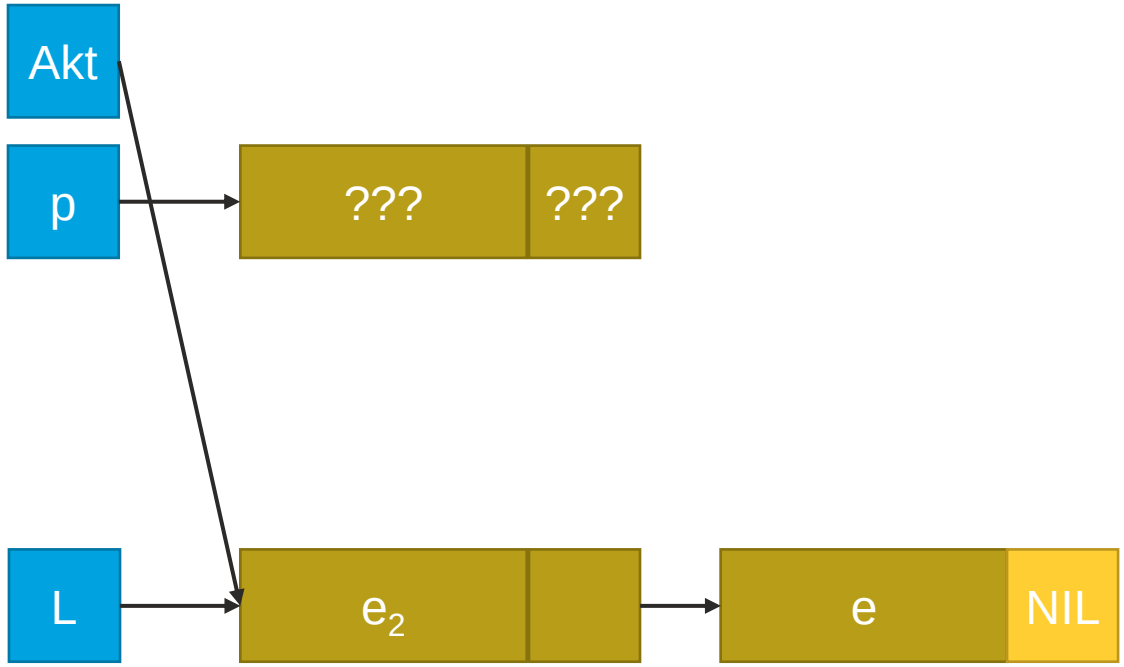



| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

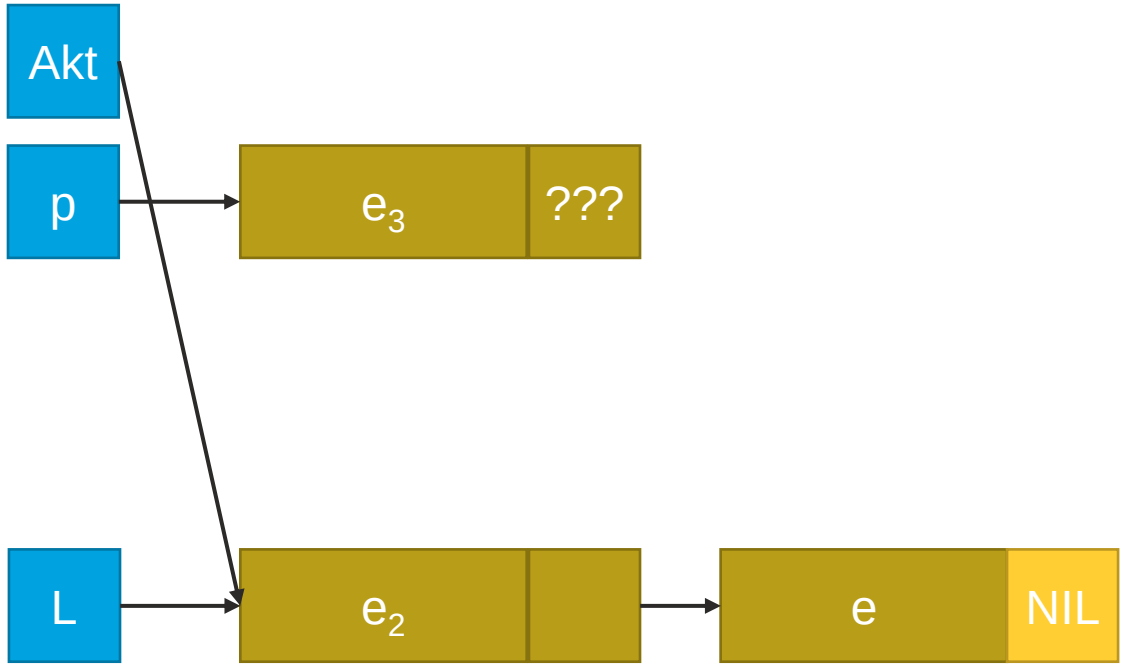



| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

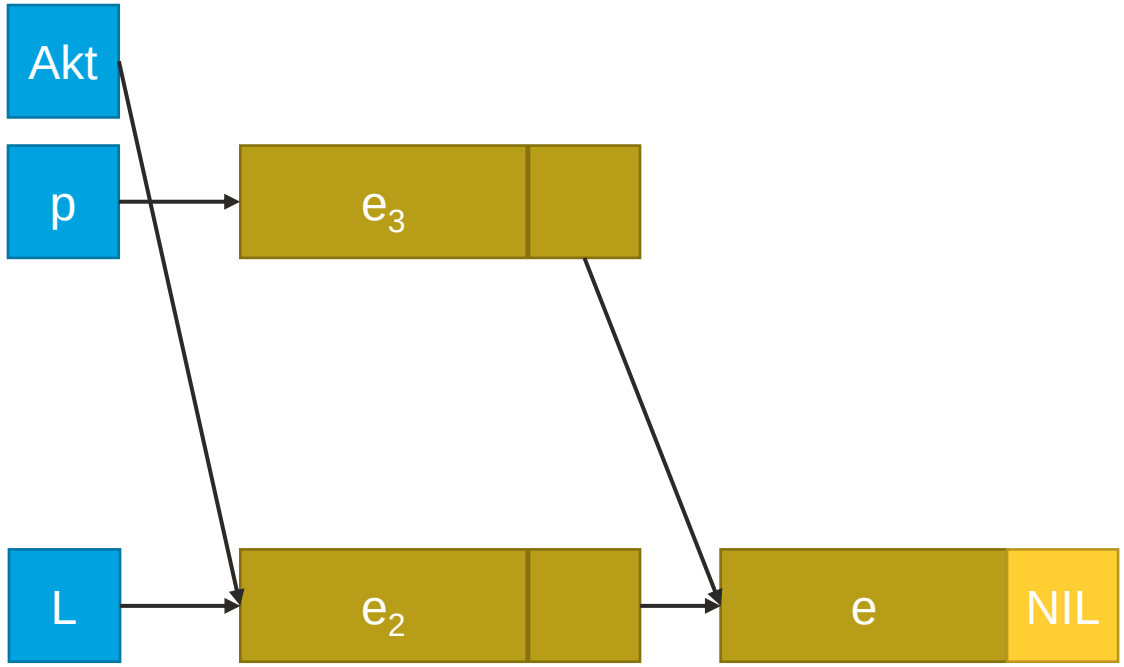



| Akt≠NIL | |
|---|---|
| new(p) | HIB A |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

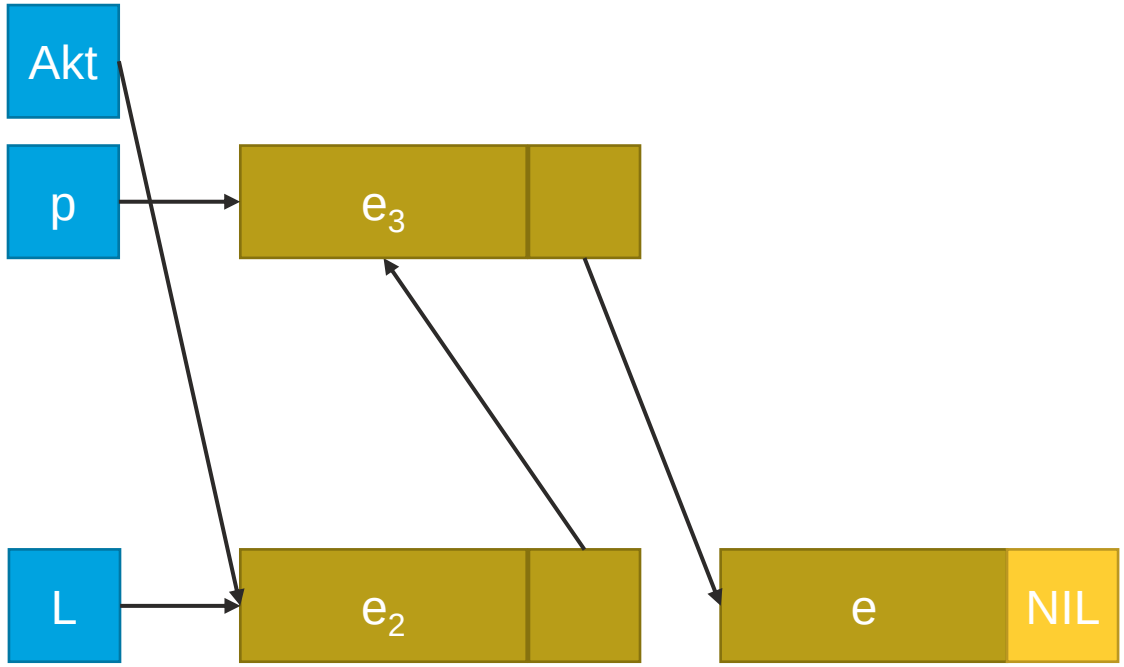



| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

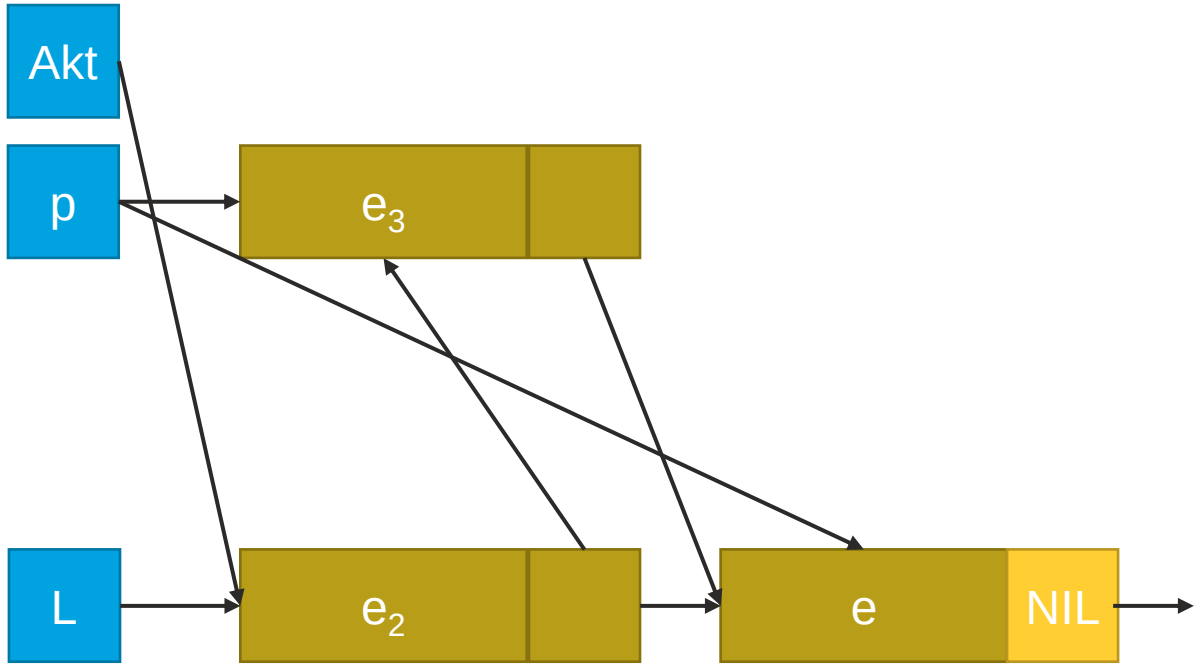



| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi



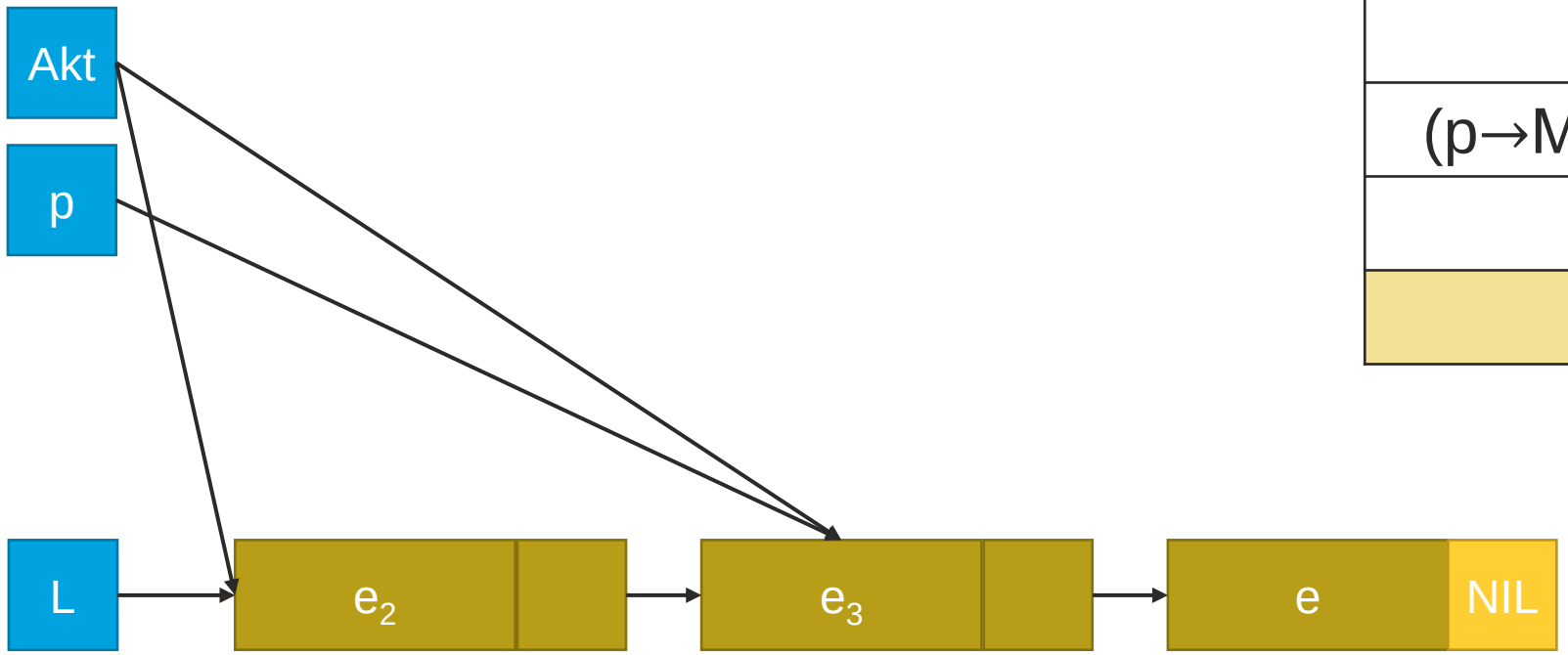

| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |

# Egyszeresen láncolt lista – InsertAfter(e)

Elem beszúrása az aktuális után

- Ha az Akt nem érvényes, akkor nem történik semmi

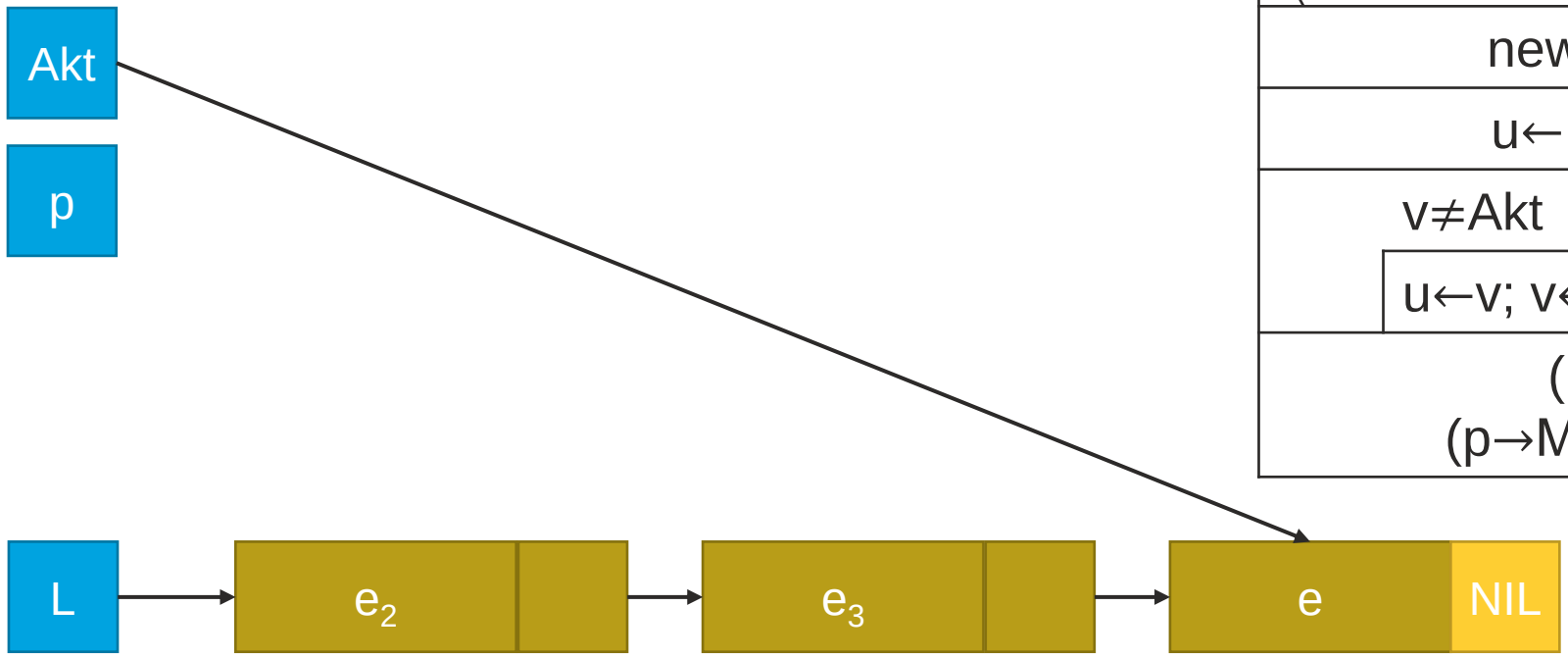
| Akt≠NIL | |
|---|---|
| new(p) | HIBA |
| (p→Adat)←e | |
| (p→Mutató)←(Akt→Mutató) | |
| (Akt→Mutató)←p | |
| Akt←p | |



# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is

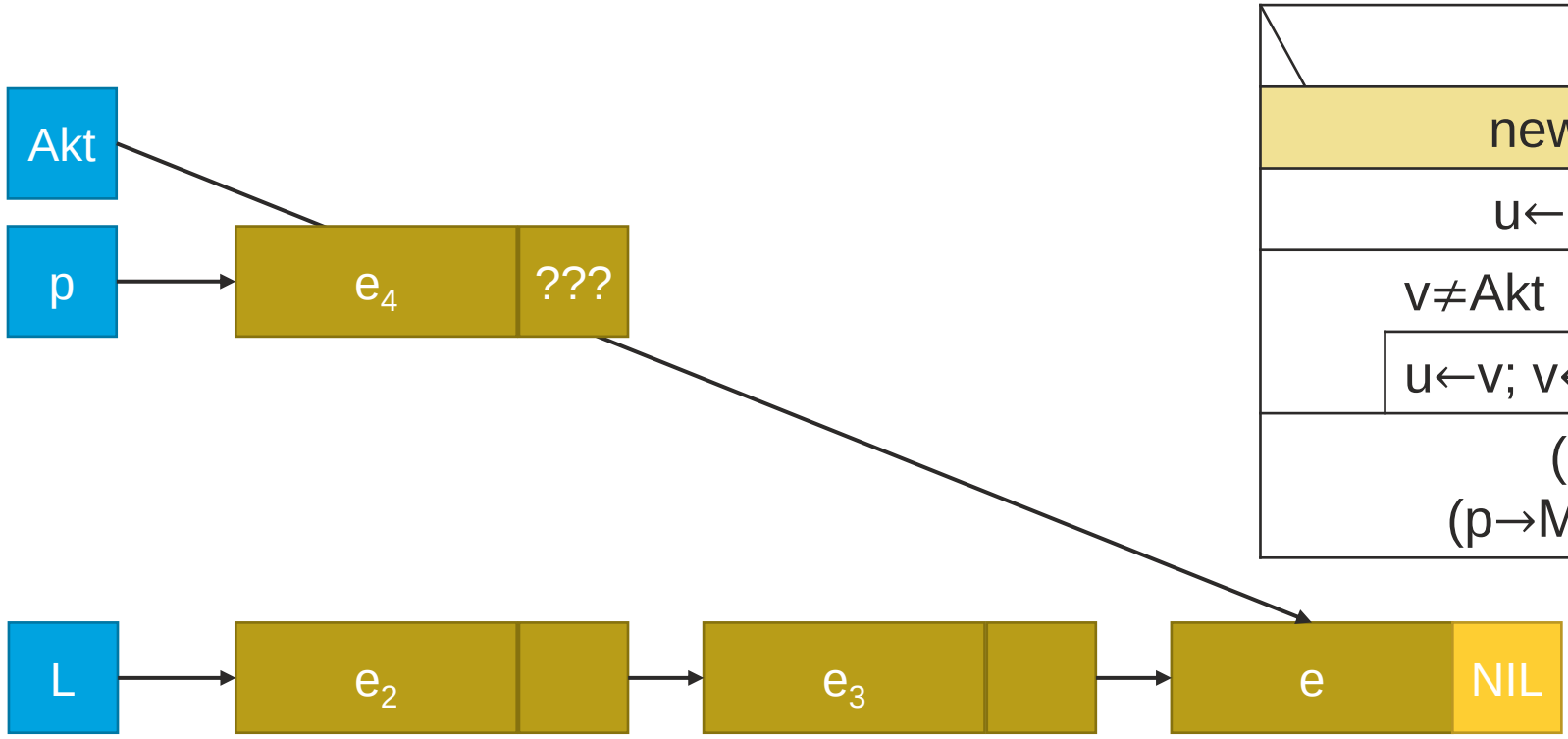



| Akt≠NIL | |
|---|---|
| new(p); (p→Adat)←e | HIBA |
| u←L; v←(L→Mutató) | |
| v≠Akt | |
| u←v; v←(v→Mutató) | |
| (u→Mutató)←p; (p→Mutató)←Akt; Akt←p | |

# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is




Akt≠NIL

new(p); (p→Adat)←e

u←L; v←(L→Mutató)

v≠Akt

u←v; v←(v→Mutató)

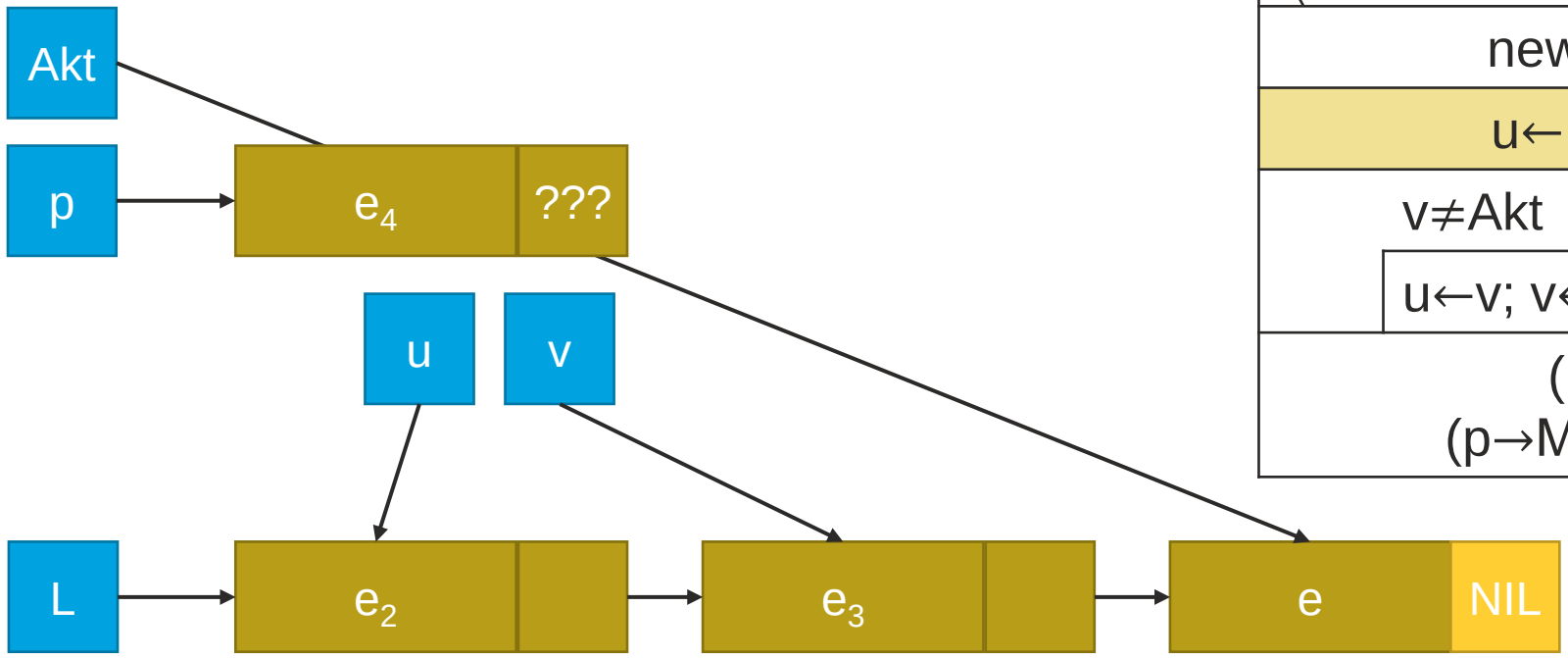
(u→Mutató)←p;
(p→Mutató)←Akt; Akt←p

HIBA

# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is




Akt≠NIL

new(p); (p→Adat)←e

u←L; v←(L→Mutató)

v≠Akt

u←v; v←(v→Mutató)

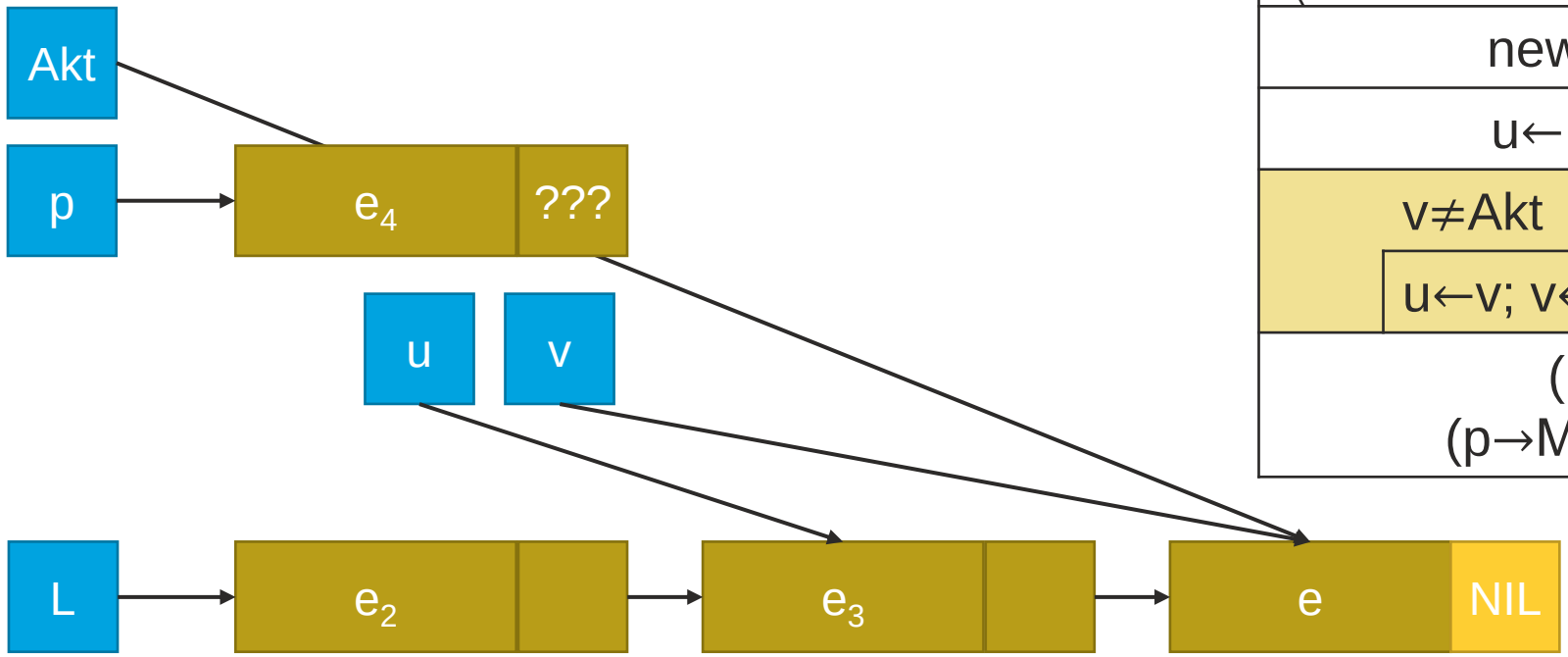
(u→Mutató)←p;
(p→Mutató)←Akt; Akt←p

HIBA

# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is

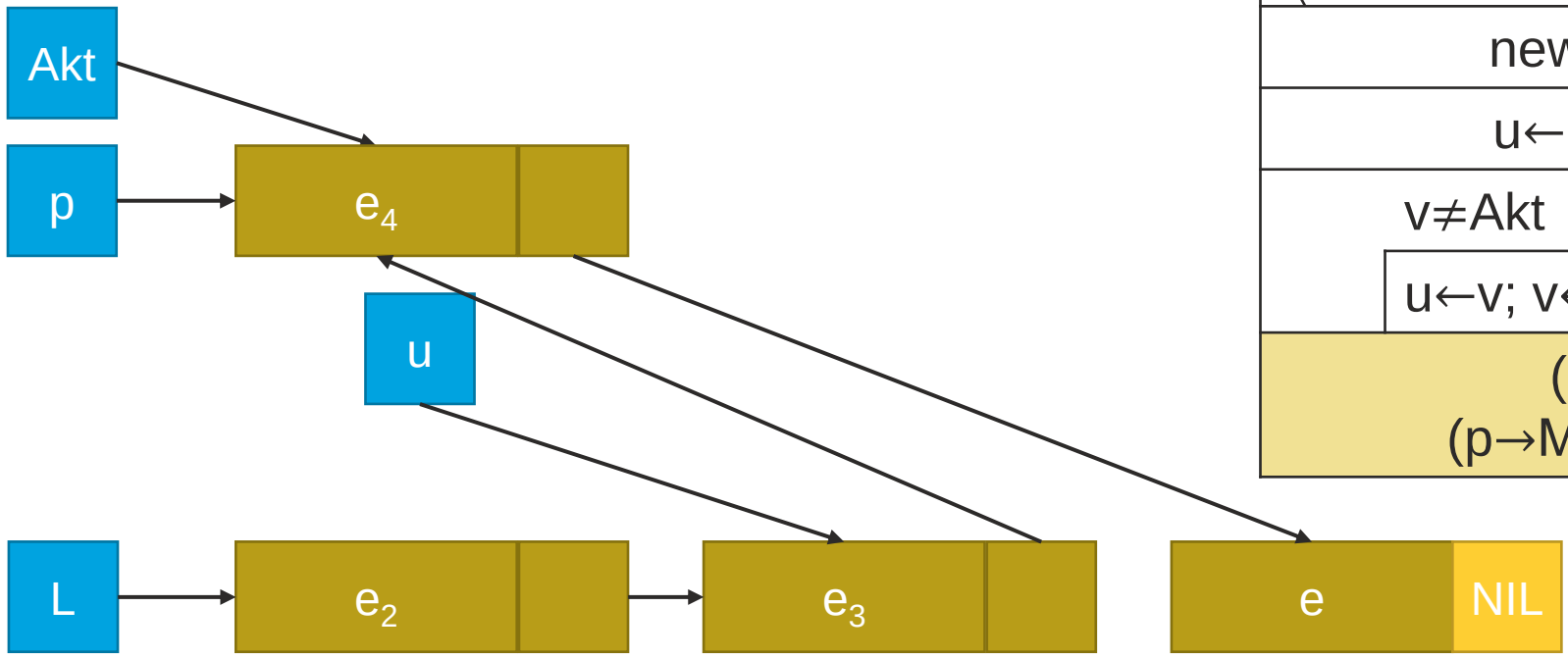



| Akt≠NIL | |
|---|---|
| new(p); (p→Adat)←e | HIBA |
| u←L; v←(L→Mutató) | |
| v≠Akt | |
| u←v; v←(v→Mutató) | |
| (u→Mutató)←p;<br>(p→Mutató)←Akt; Akt←p | |

# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is




Akt≠NIL

new(p); (p→Adat)←e

u←L; v←(L→Mutató)

v≠Akt

u←v; v←(v→Mutató)

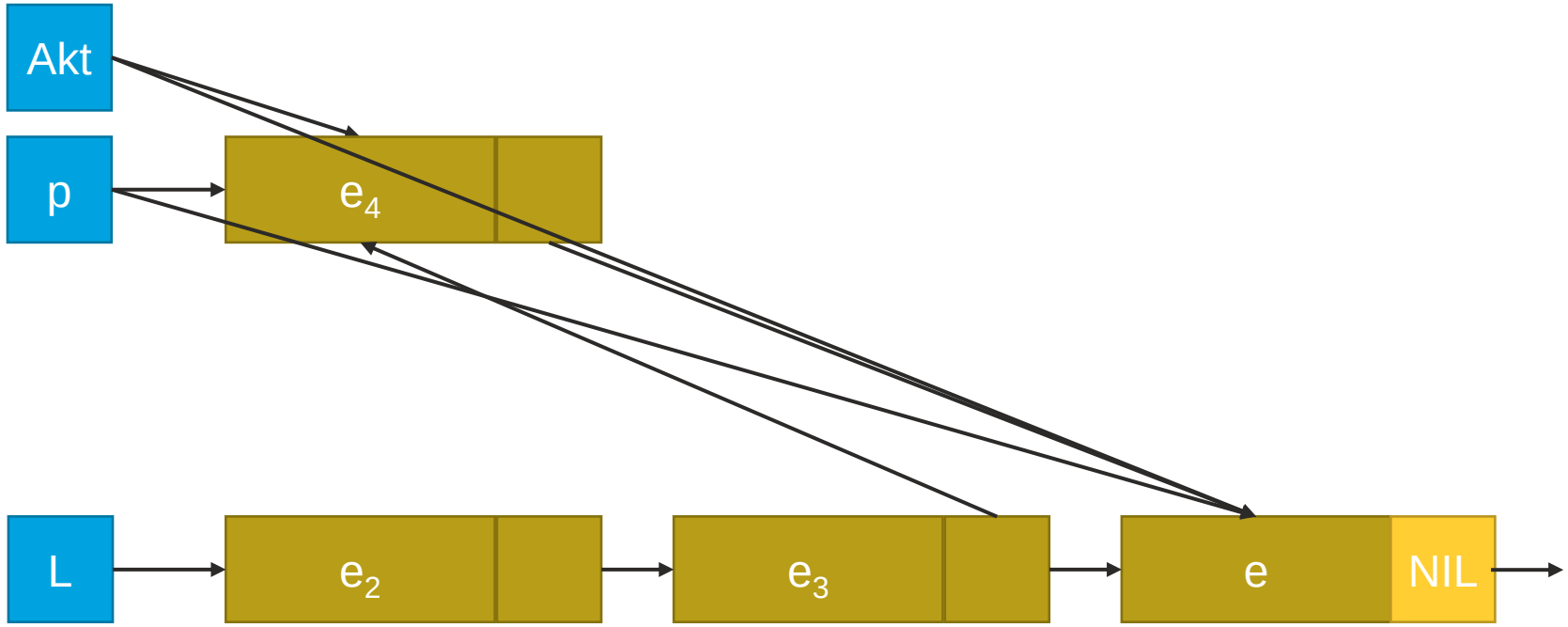
(u→Mutató)←p;
(p→Mutató)←Akt; Akt←p

HIBA

# Egyszeresen láncolt lista – InsertBefore(e)

Elem beszúrása az aktuális elé

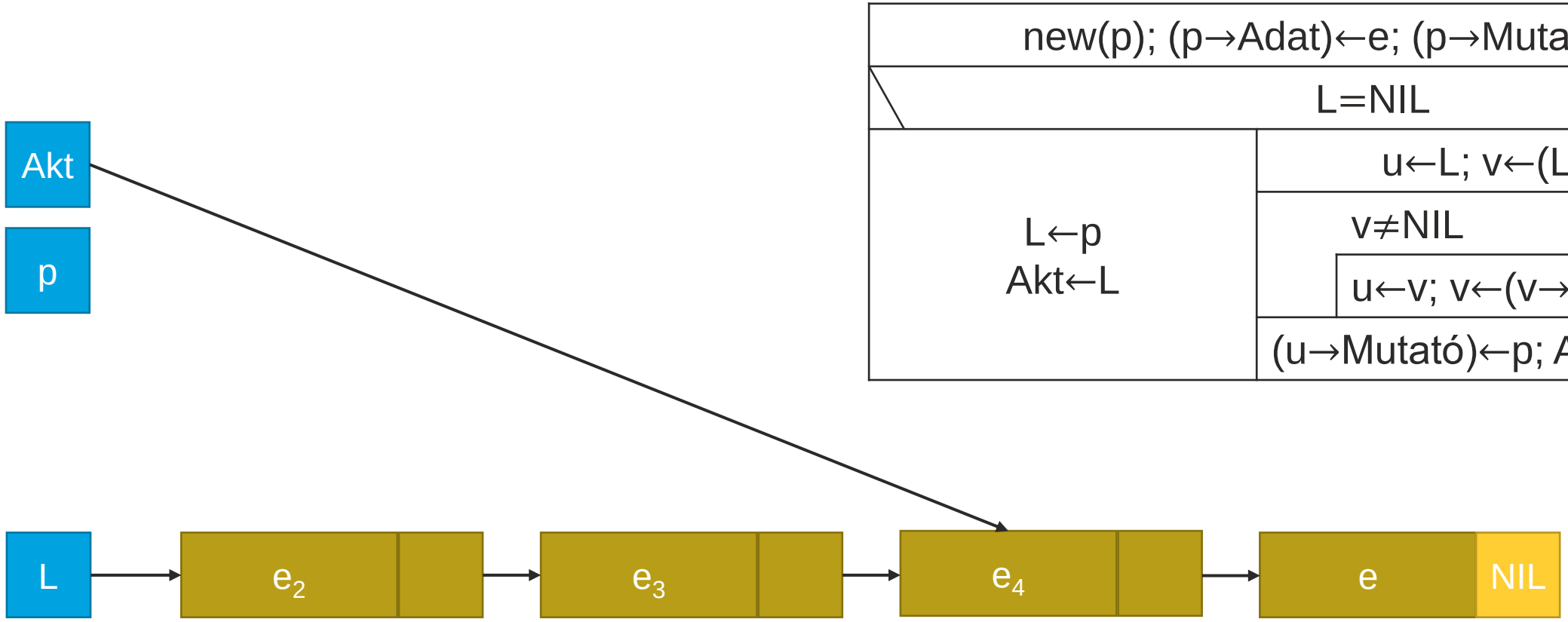
- Ha az Akt nem érvényes, akkor nem történik semmi
- A nehézséget az jelenti, hogy a megelőző elemet meg kell találni először
- A példában az Akt most az utolsó elemre mutat, hogy legyen előtte két elem is



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

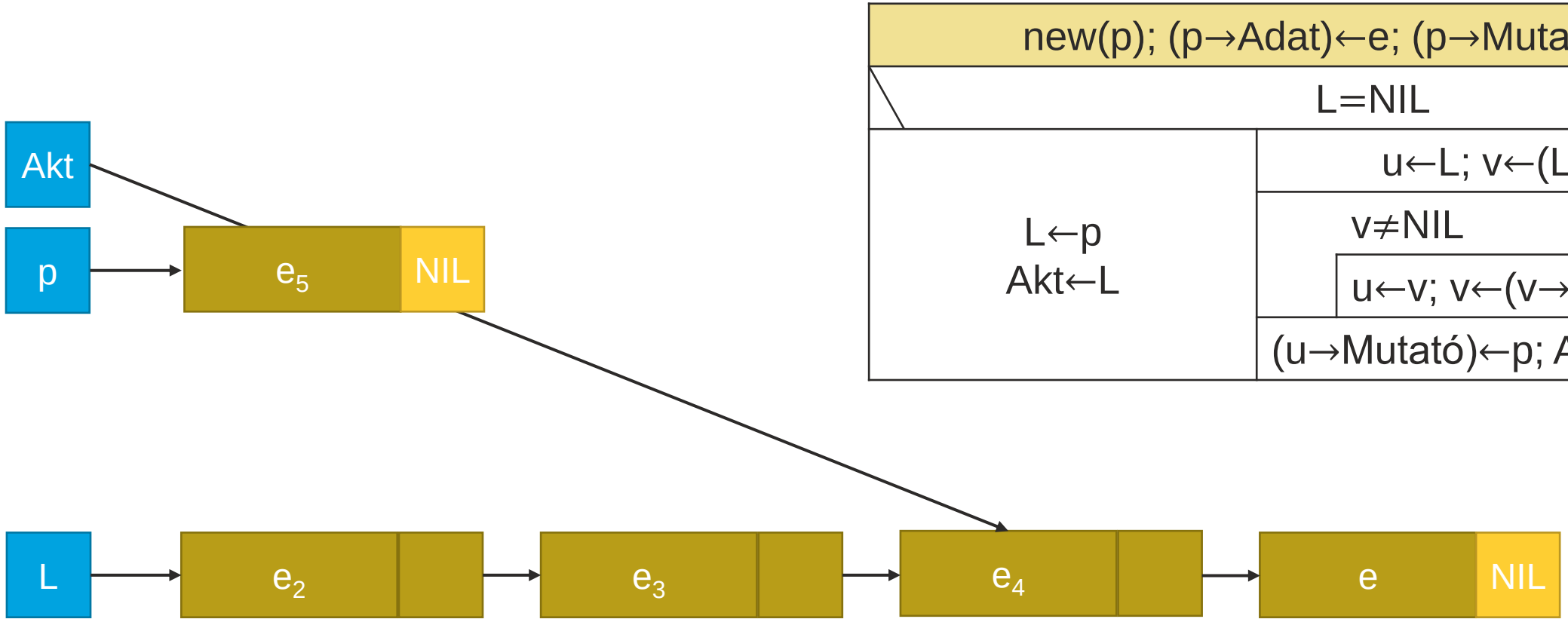
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

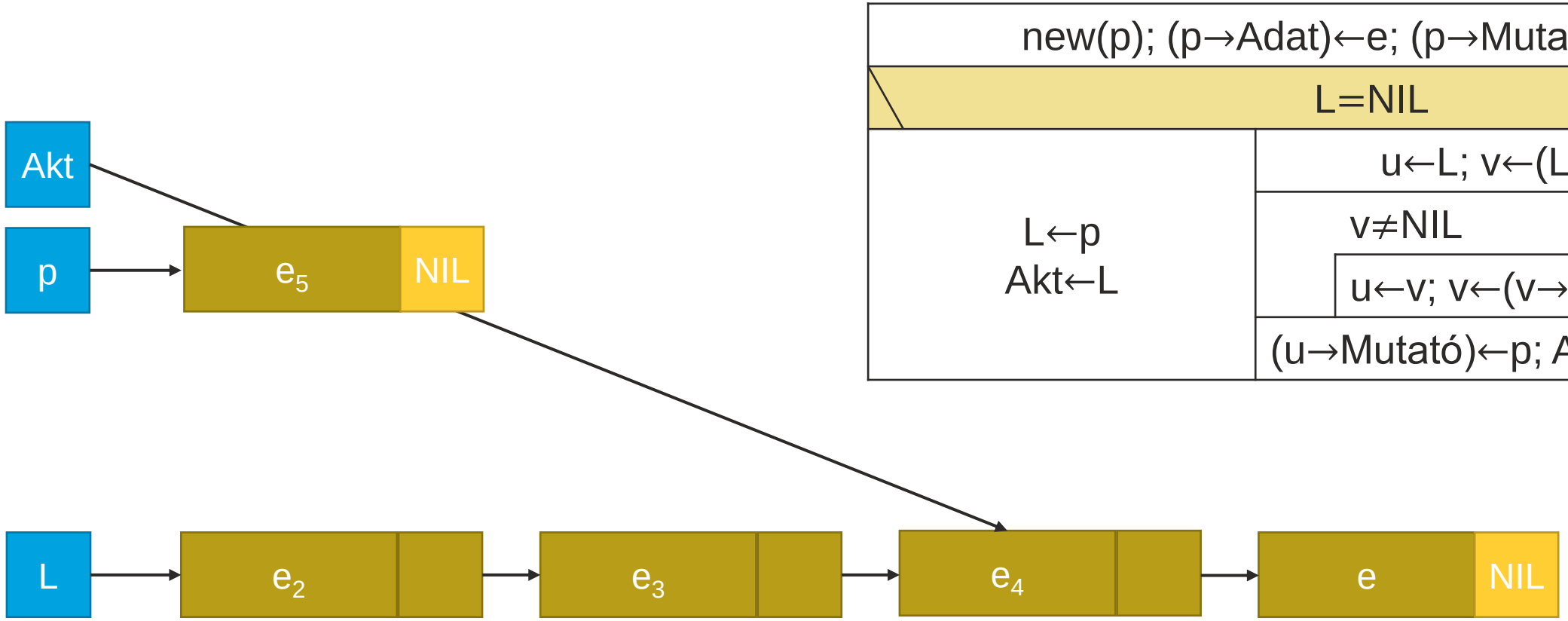
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

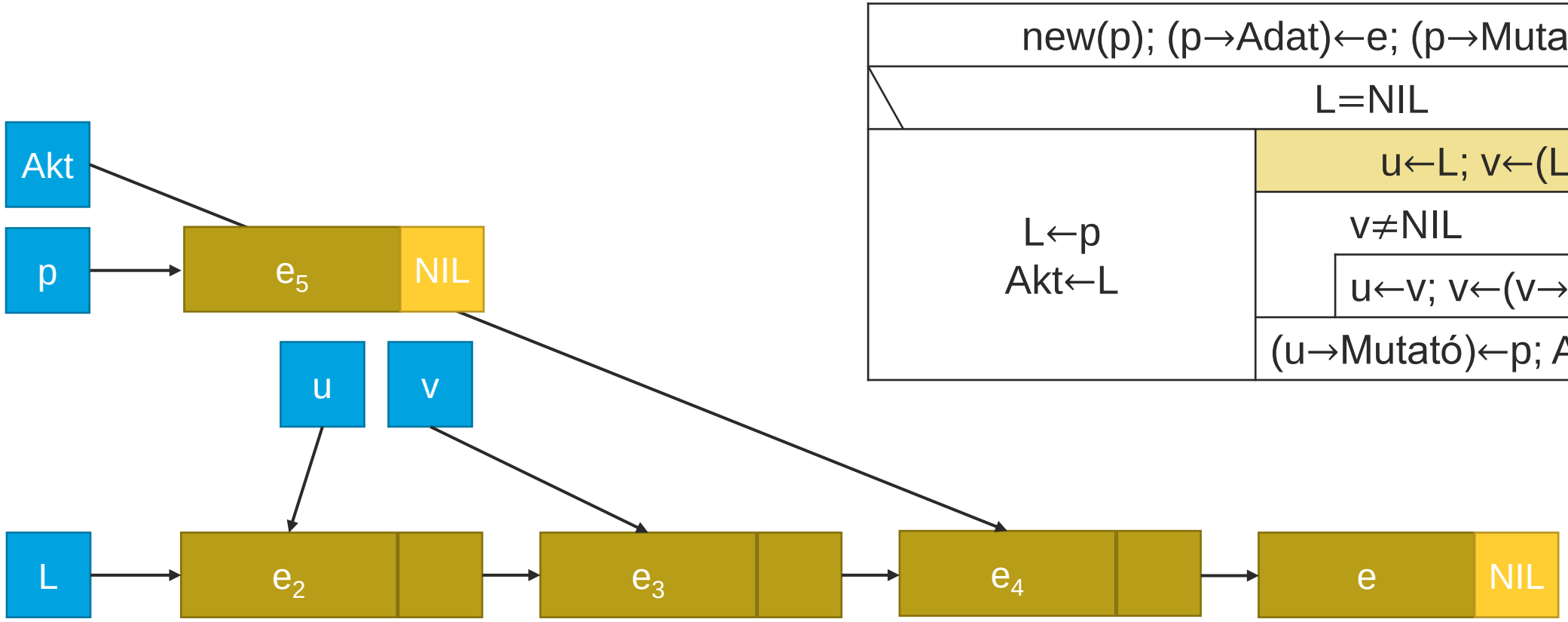
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha üres a lista, akkor az első helyre kell betenni, ami egyszerű



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el




new(p); (p→Adat)←e; (p→Mutató)←NIL

L=NIL

L←p
Akt←L

u←L; v←(L→Mutató)

v≠NIL

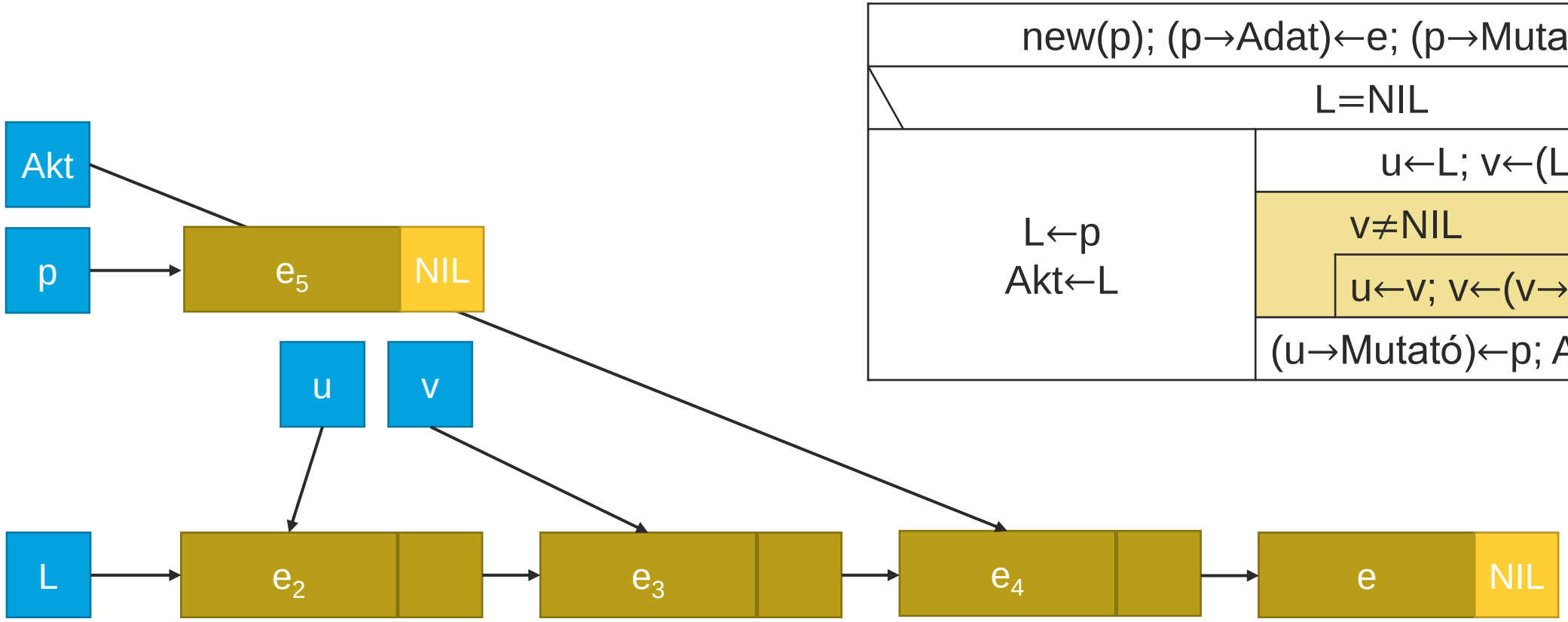
u←v; v←(v→Mutató)

(u→Mutató)←p; Akt←p

# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el




new(p); (p→Adat)←e; (p→Mutató)←NIL

L=NIL

L←p
Akt←L

u←L; v←(L→Mutató)

v≠NIL

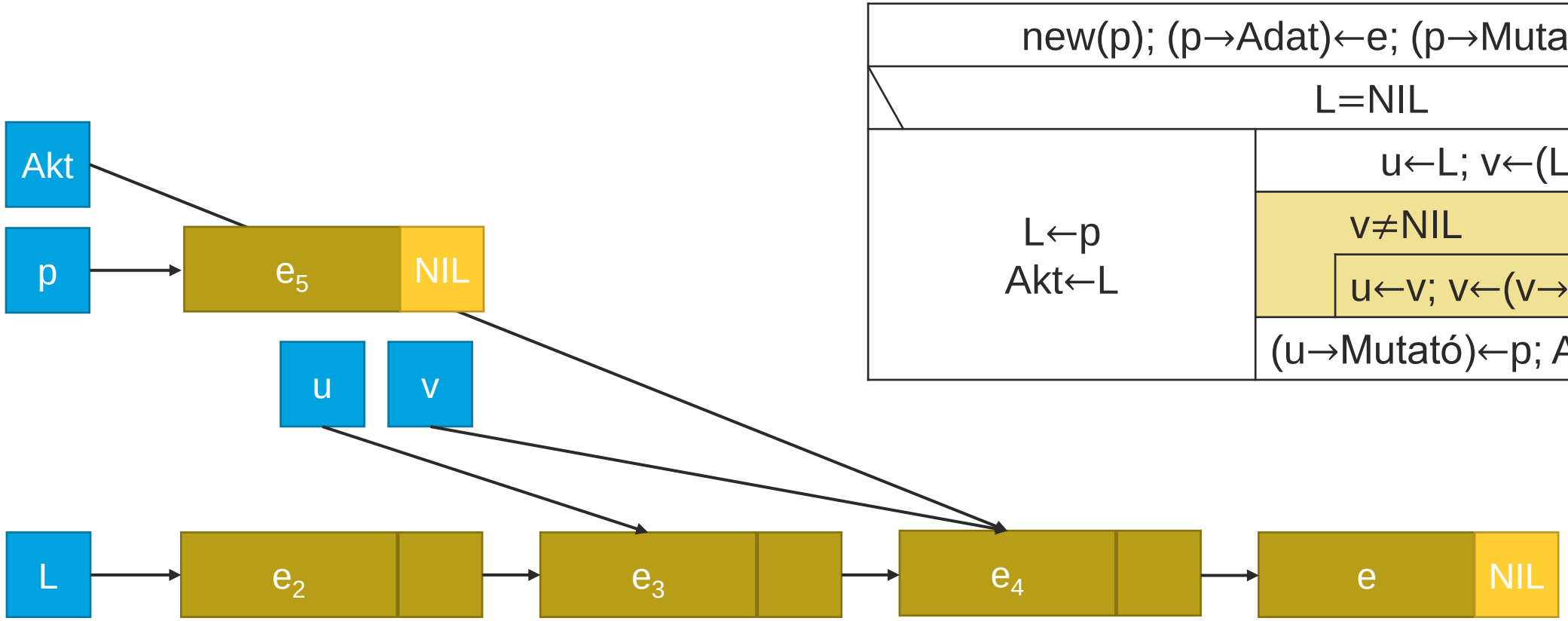
u←v; v←(v→Mutató)

(u→Mutató)←p; Akt←p

# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el




new(p); (p→Adat)←e; (p→Mutató)←NIL

L=NIL

L←p
Akt←L

u←L; v←(L→Mutató)

v≠NIL

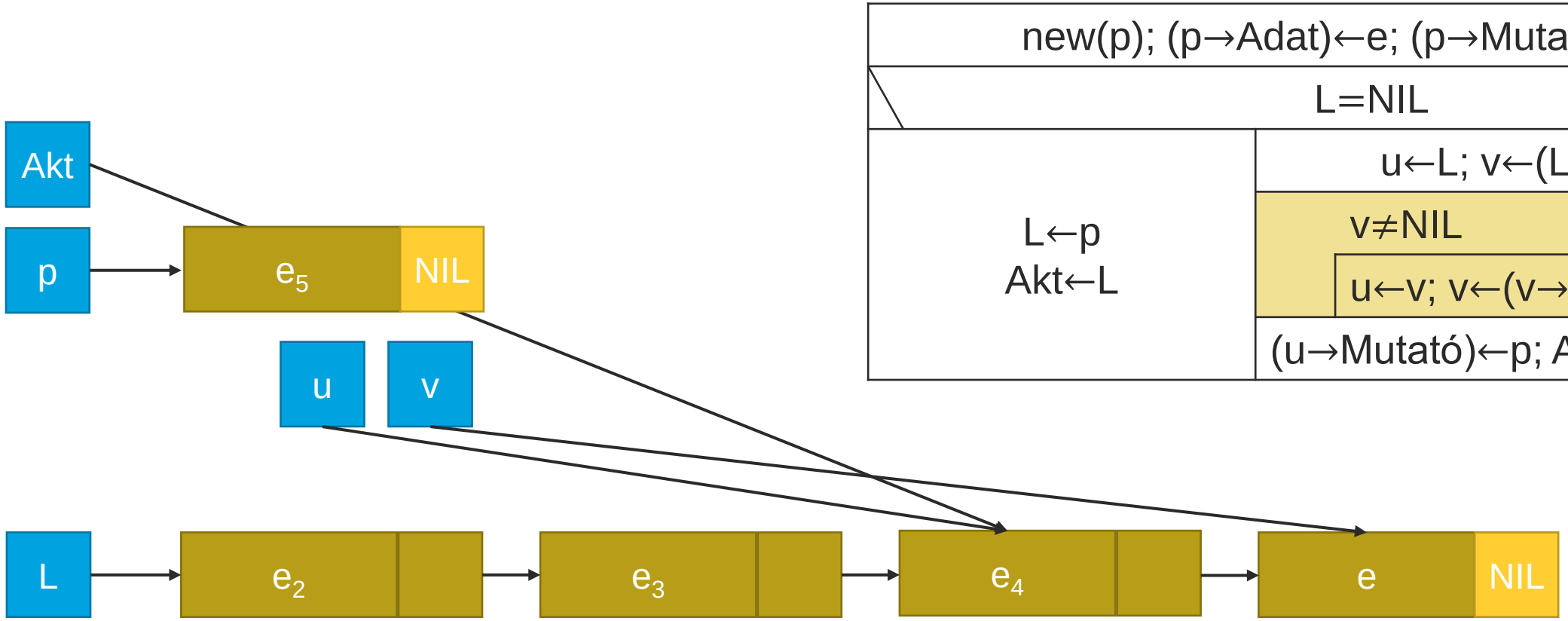
u←v; v←(v→Mutató)

(u→Mutató)←p; Akt←p

# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el




new(p); (p→Adat)←e; (p→Mutató)←NIL

L=NIL

L←p
Akt←L

u←L; v←(L→Mutató)

v≠NIL

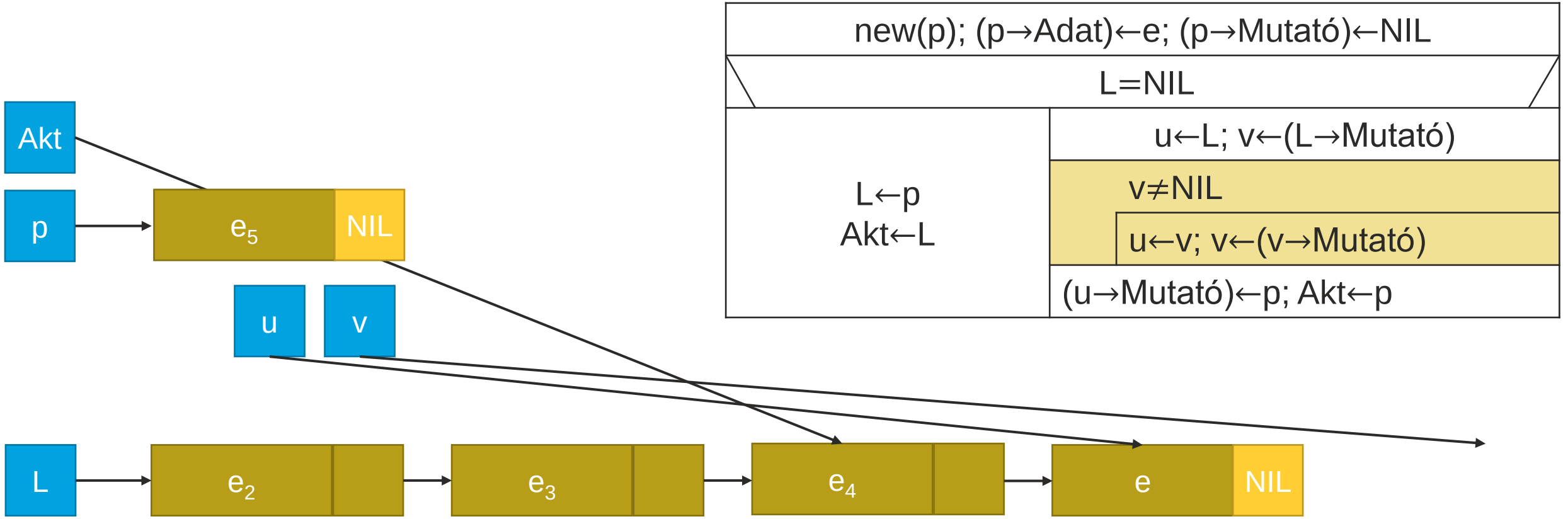
u←v; v←(v→Mutató)

(u→Mutató)←p; Akt←p

# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

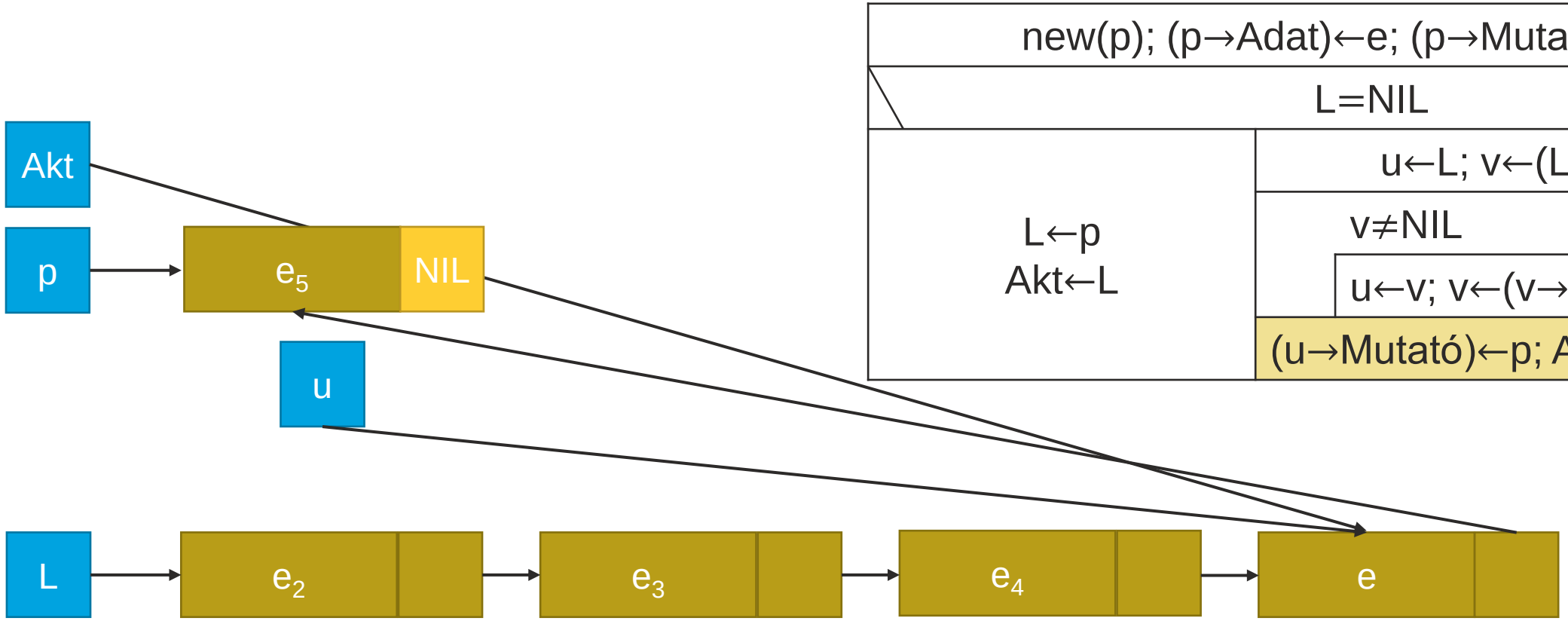
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

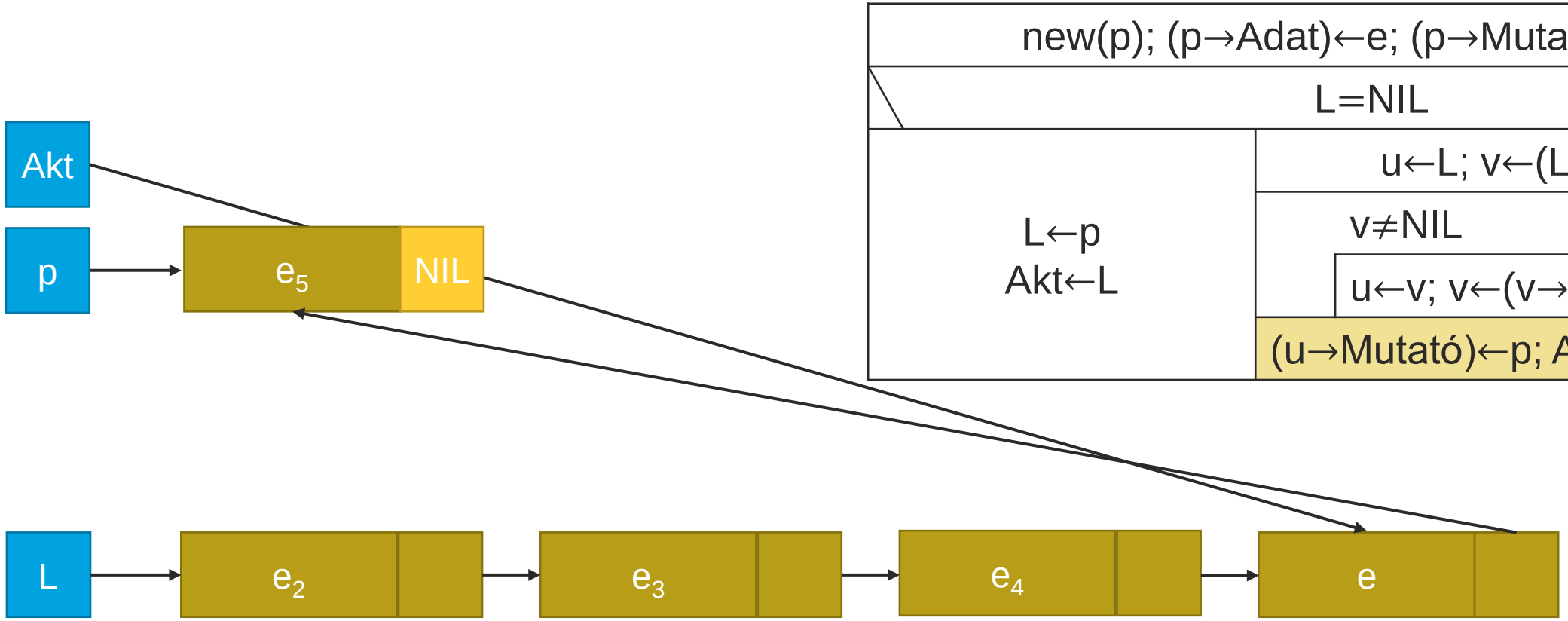
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

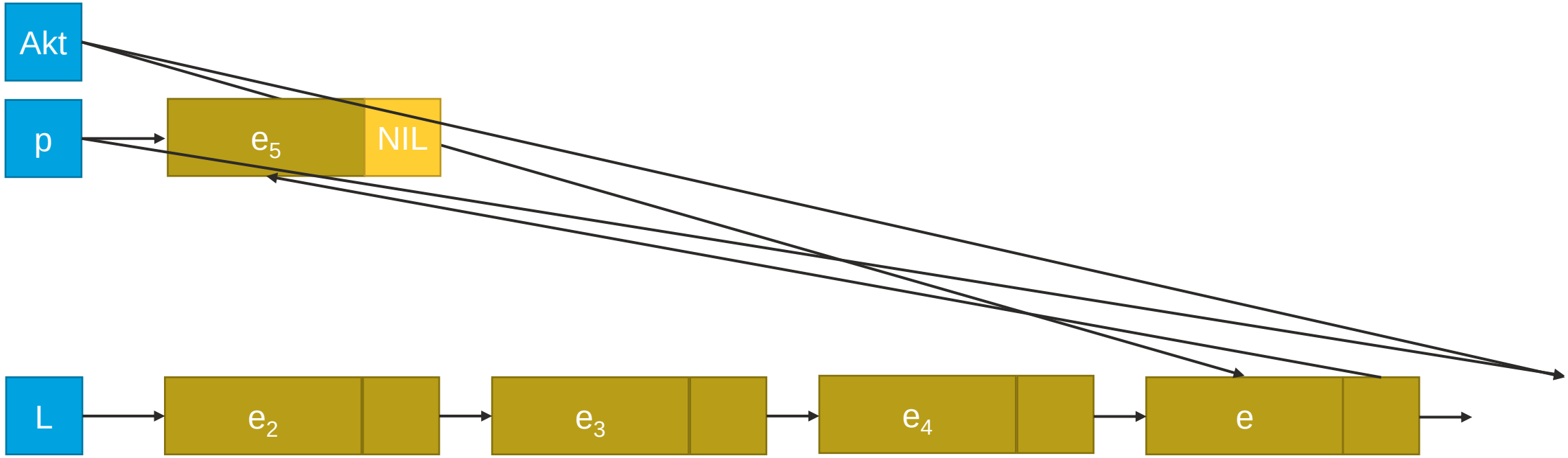
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el



# Egyszeresen láncolt lista – InsertLast()

Elem beszúrása utolsónak

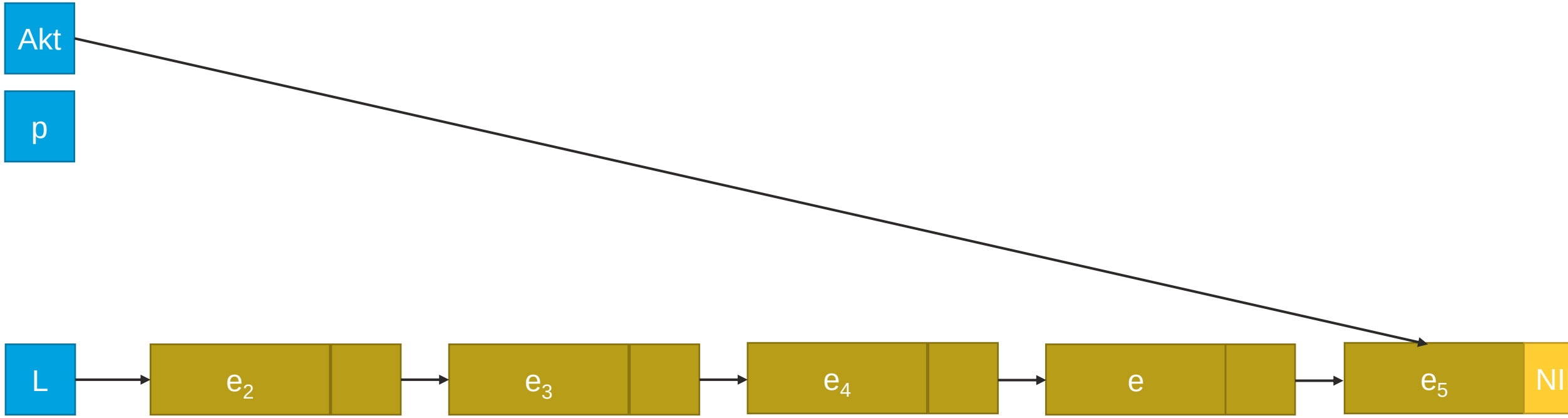
- A nehézséget az jelenti, hogy az eddigi utolsó elemet meg kell találni
- Ha nem üres, akkor az algoritmus szerint járunk el



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

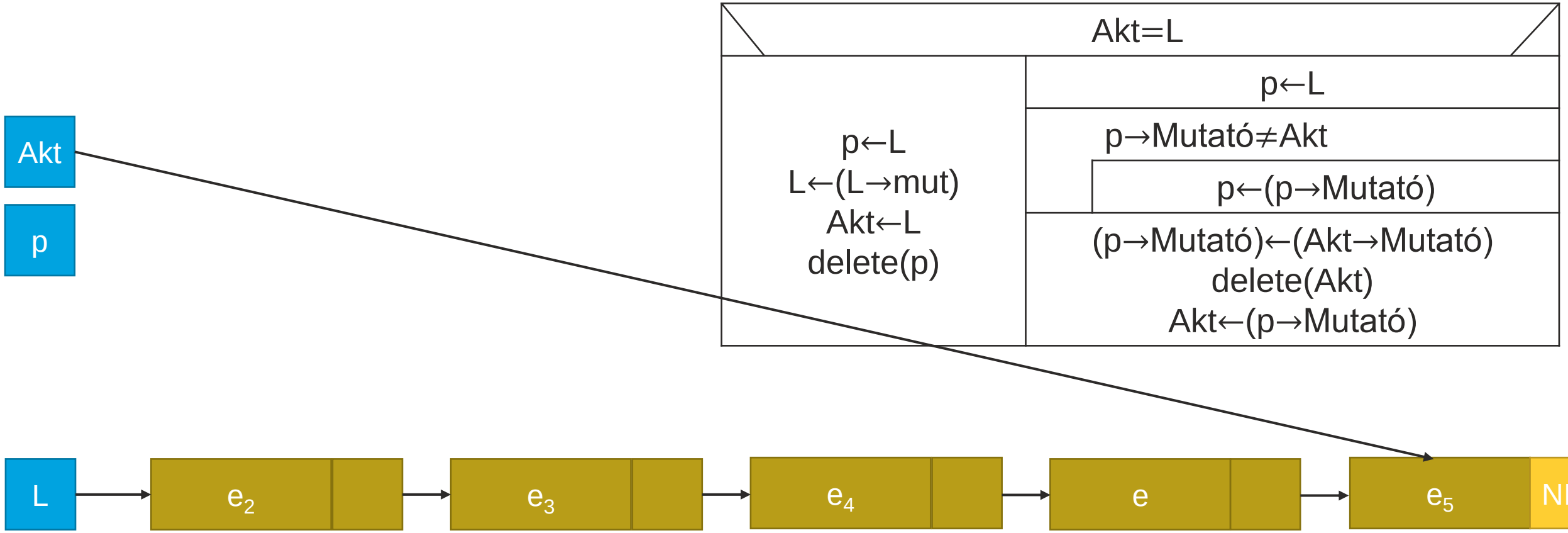
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

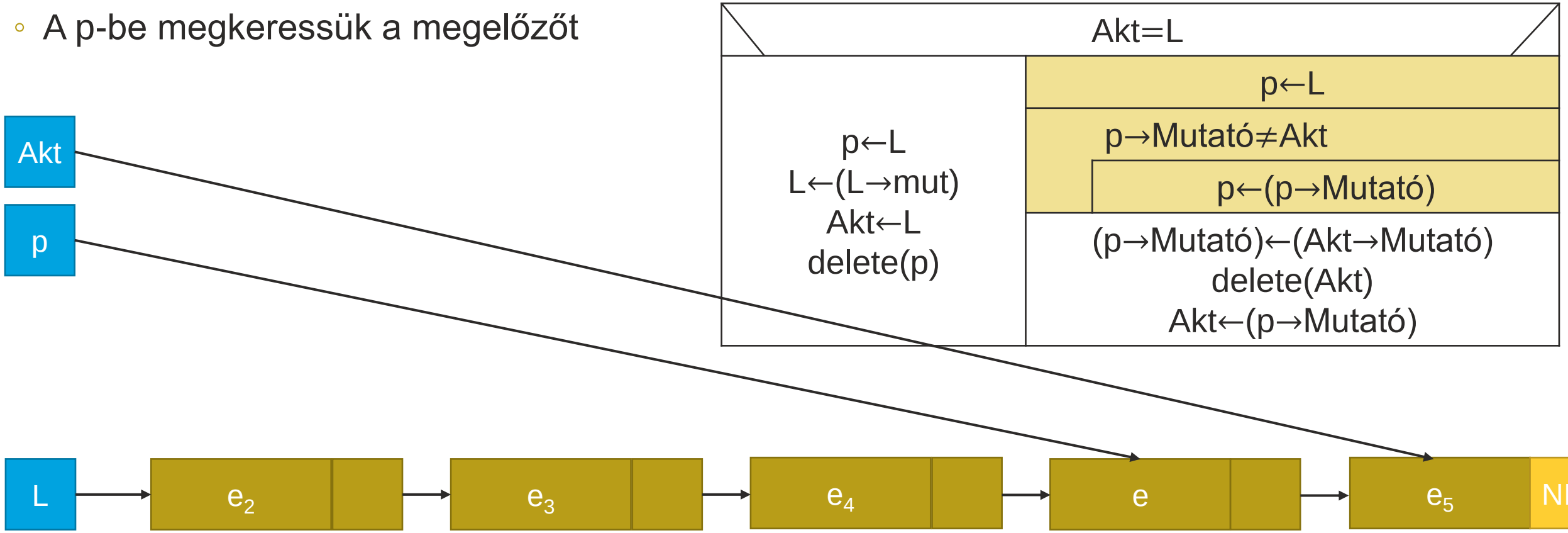
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

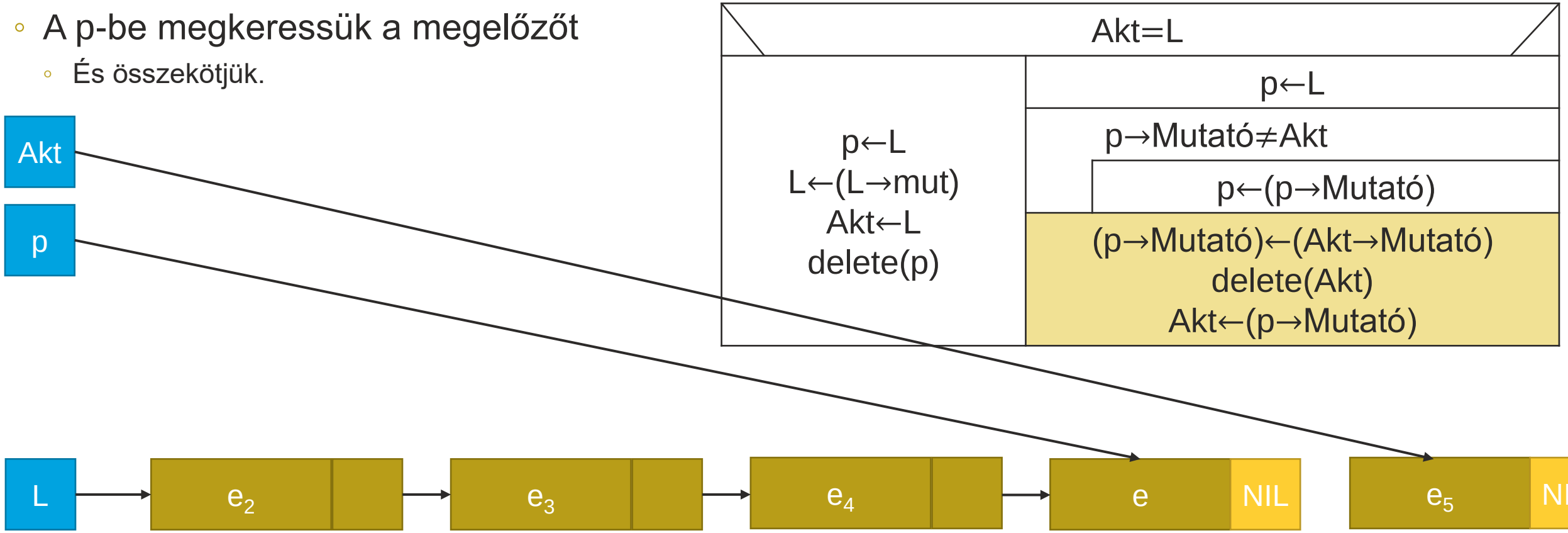
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- A p-be megkeressük a megelőzőt



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

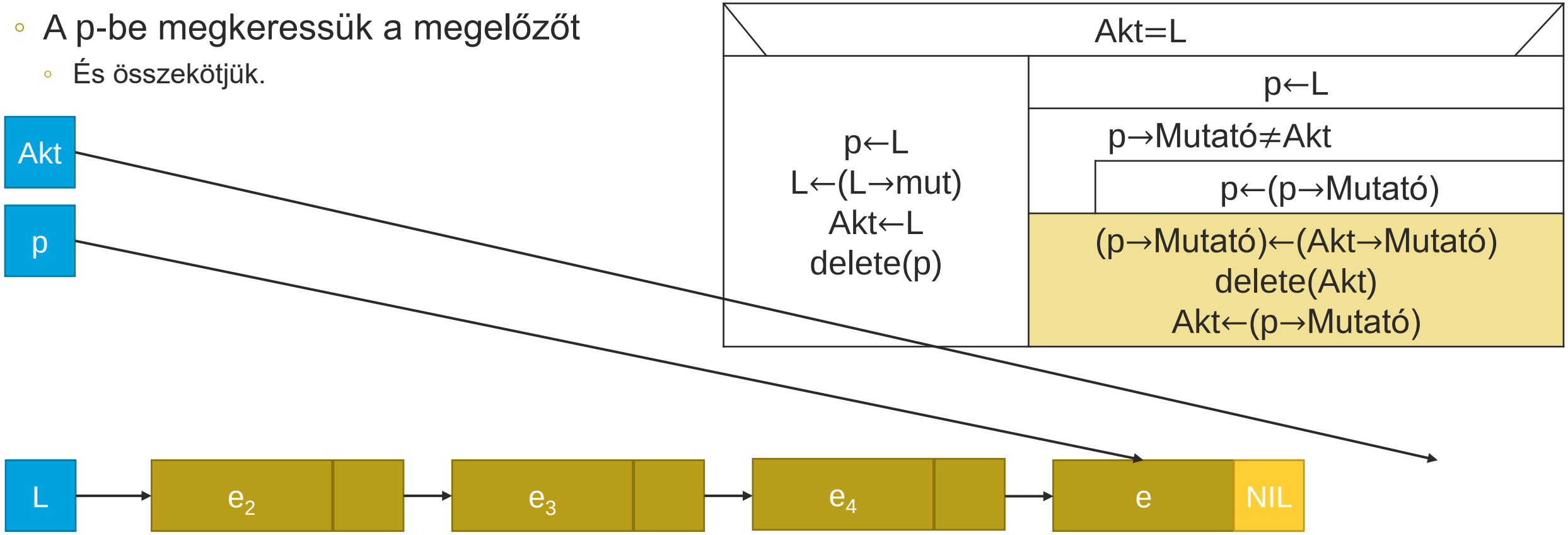
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- A p-be megkeressük a megelőzőt
  - És összekötjük.



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

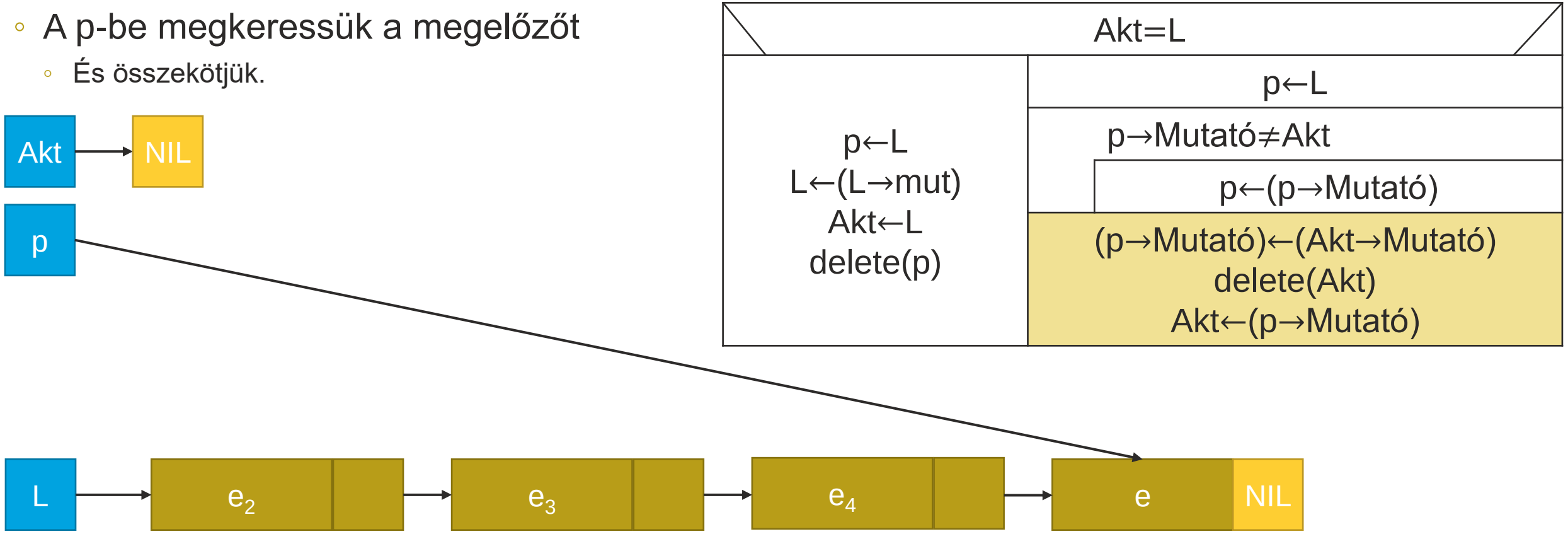
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- A p-be megkeressük a megelőzőt
  - És összekötjük.



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

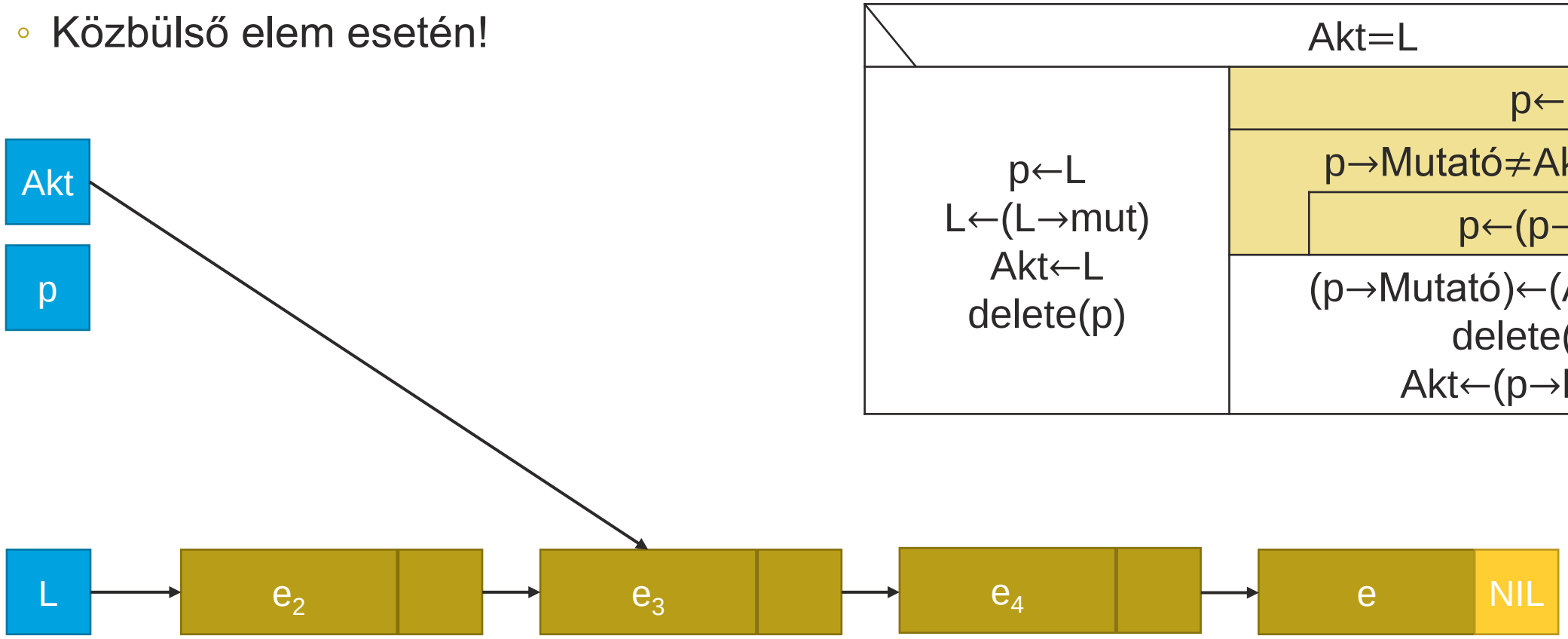
- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- A p-be megkeressük a megelőzőt
  - És összekötjük.



# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

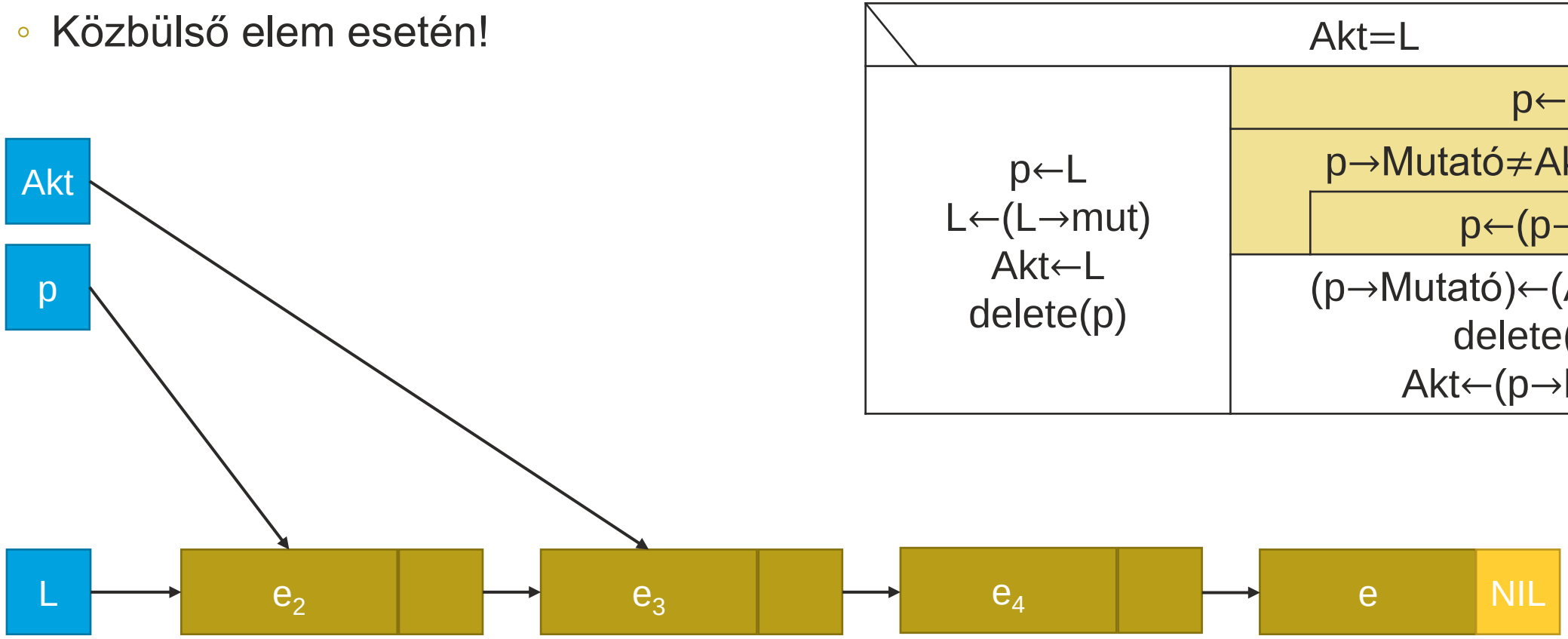



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L<br>p→Mutató≠Akt<br>p←(p→Mutató)<br>(p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

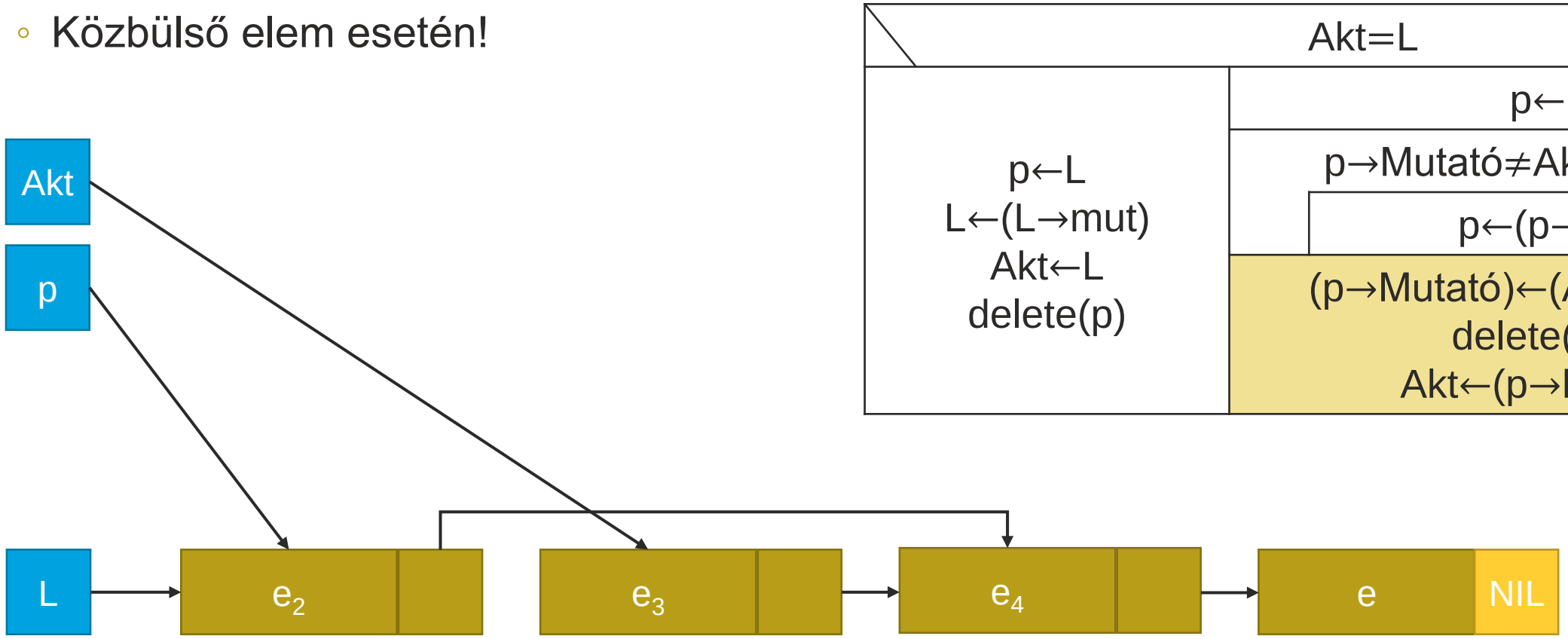



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L<br>p→Mutató≠Akt<br>p←(p→Mutató)<br>(p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

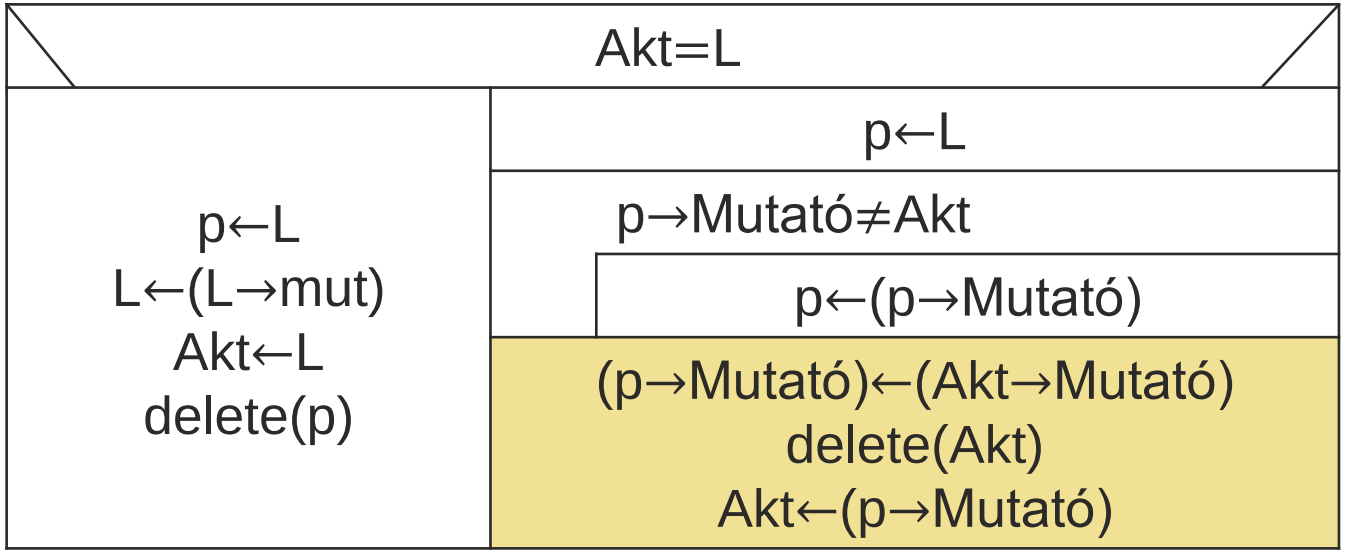


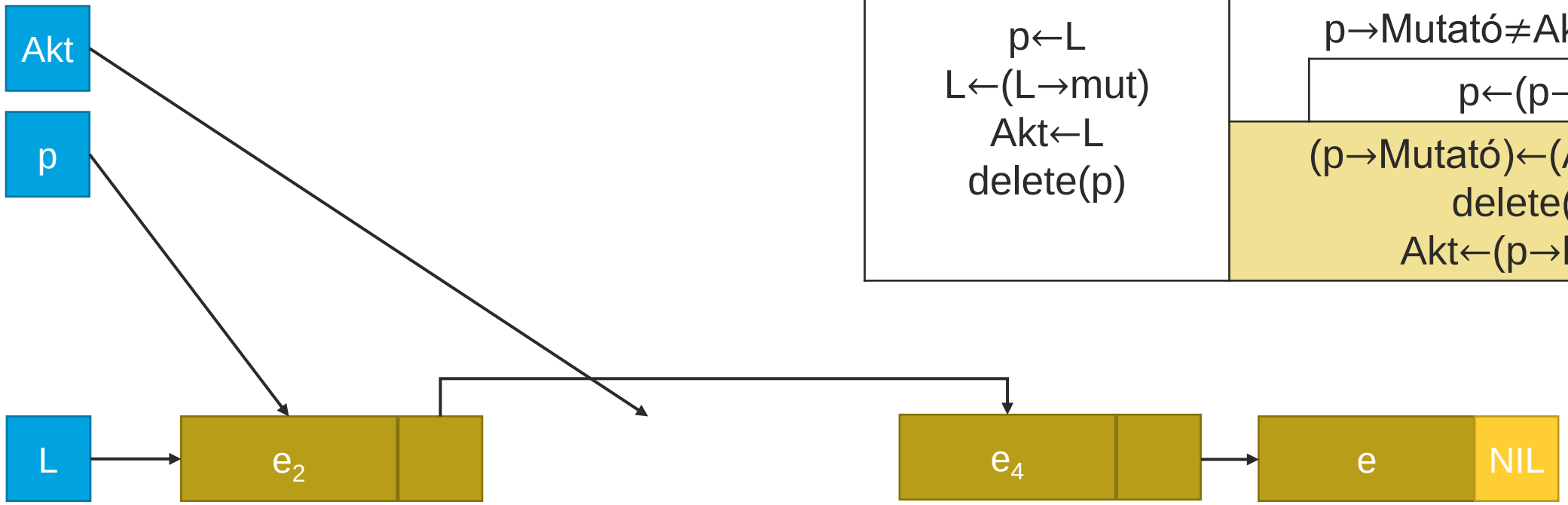

| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L<br>p→Mutató≠Akt<br>p←(p→Mutató)<br>(p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

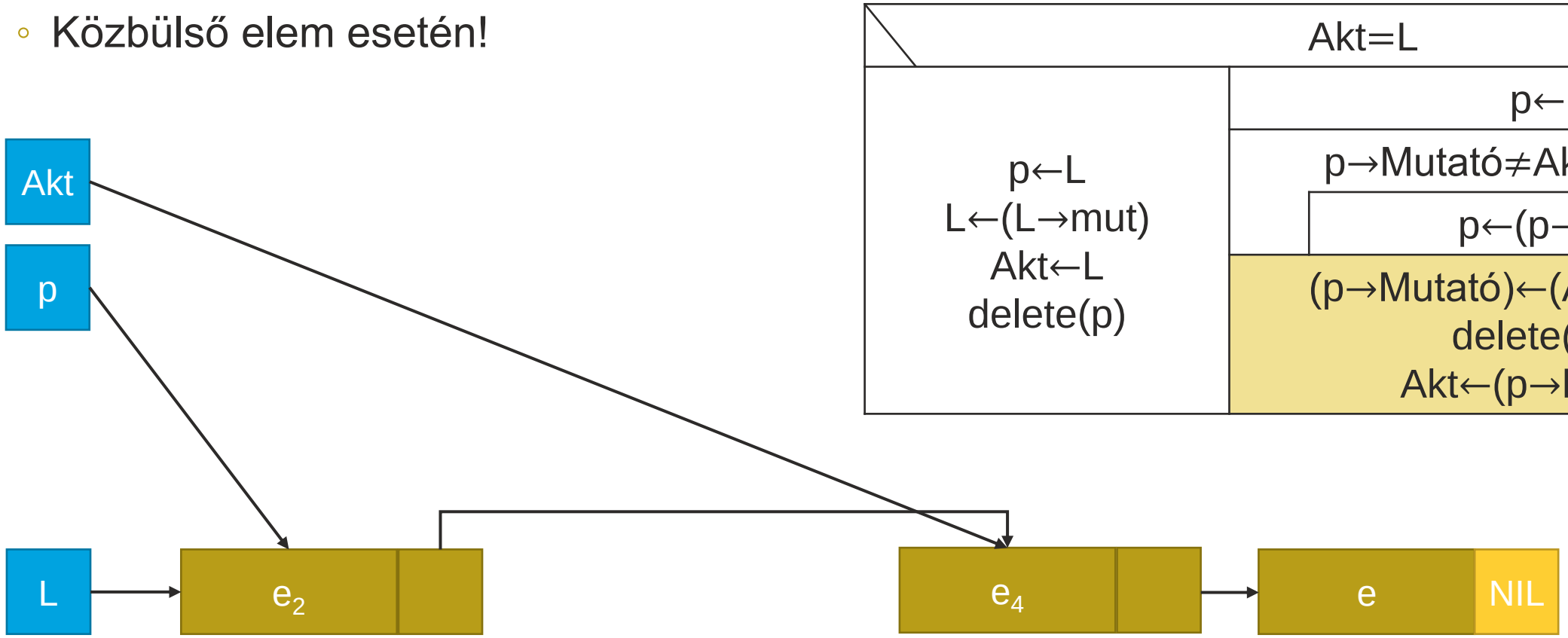


# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

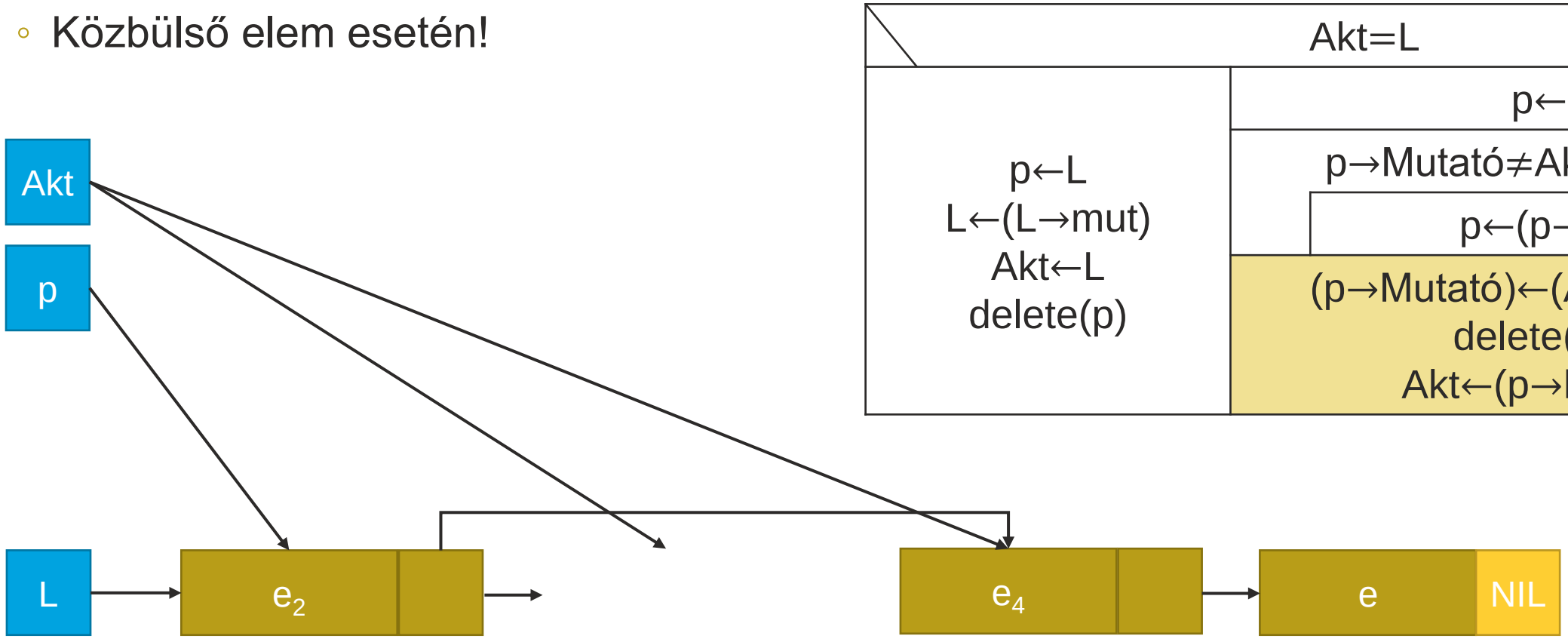



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L<br>p→Mutató≠Akt<br>p←(p→Mutató)<br>(p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- A törlés után az elemet megelőző és az elemet követő elemet kötjük össze
  - Vegyük észre, hogy csak az első elem törlését kell külön kezelni!
- Közbülső elem esetén!

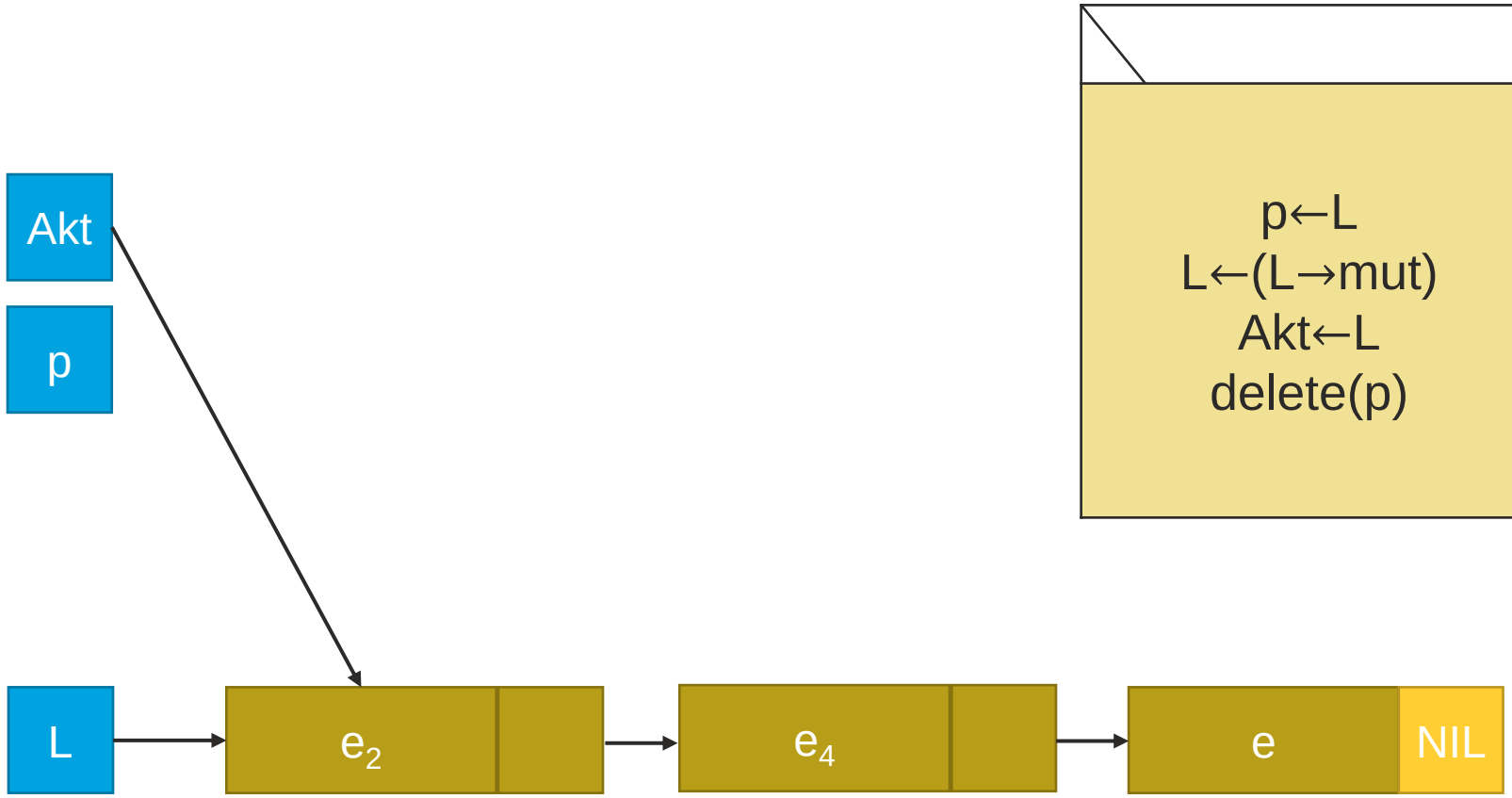



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L<br>p→Mutató≠Akt<br>p←(p→Mutató)<br>(p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- Első elem esetén.

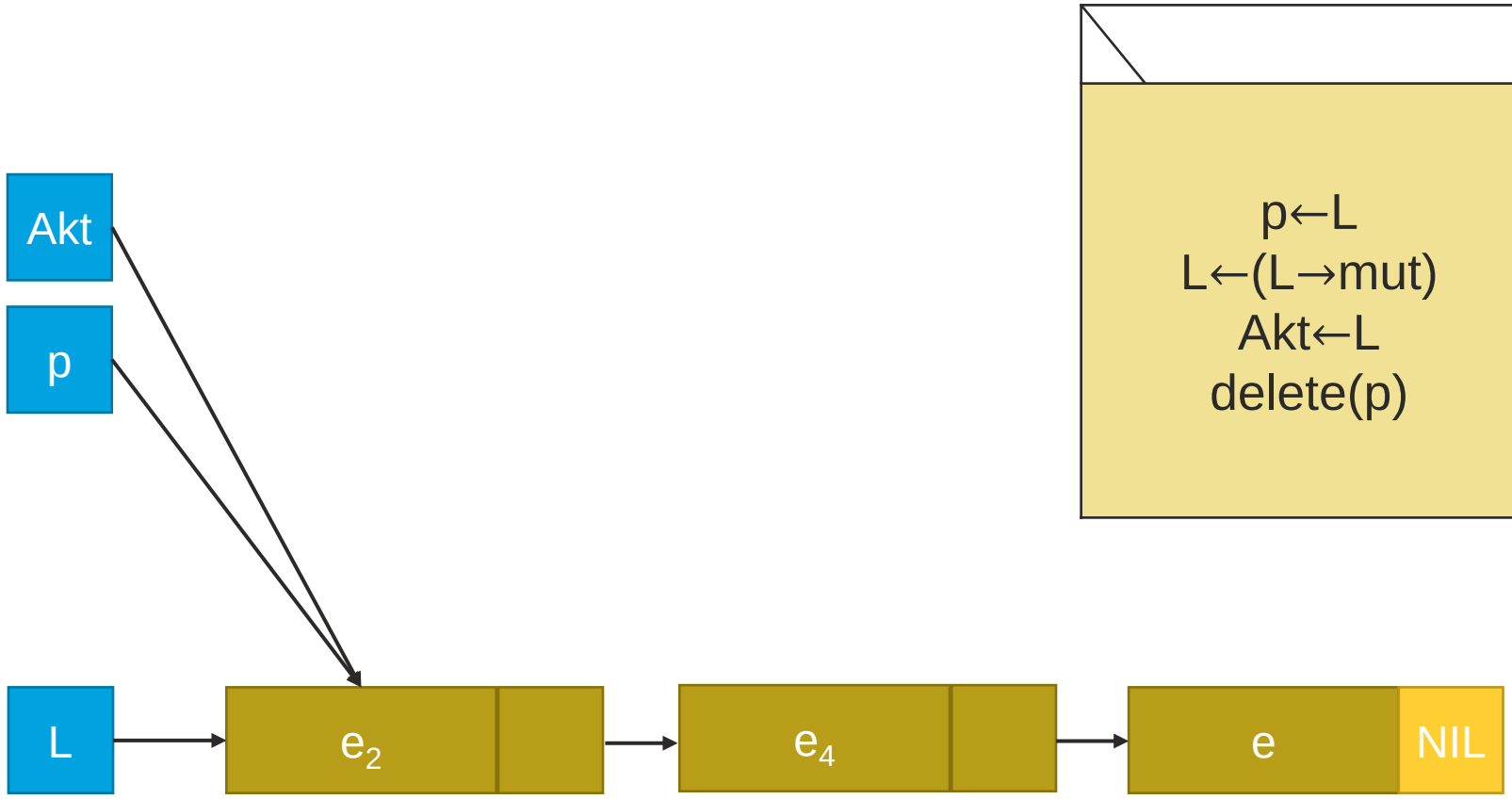



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L |
| | p→Mutató≠Akt |
| | p←(p→Mutató) |
| | (p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- Első elem esetén.

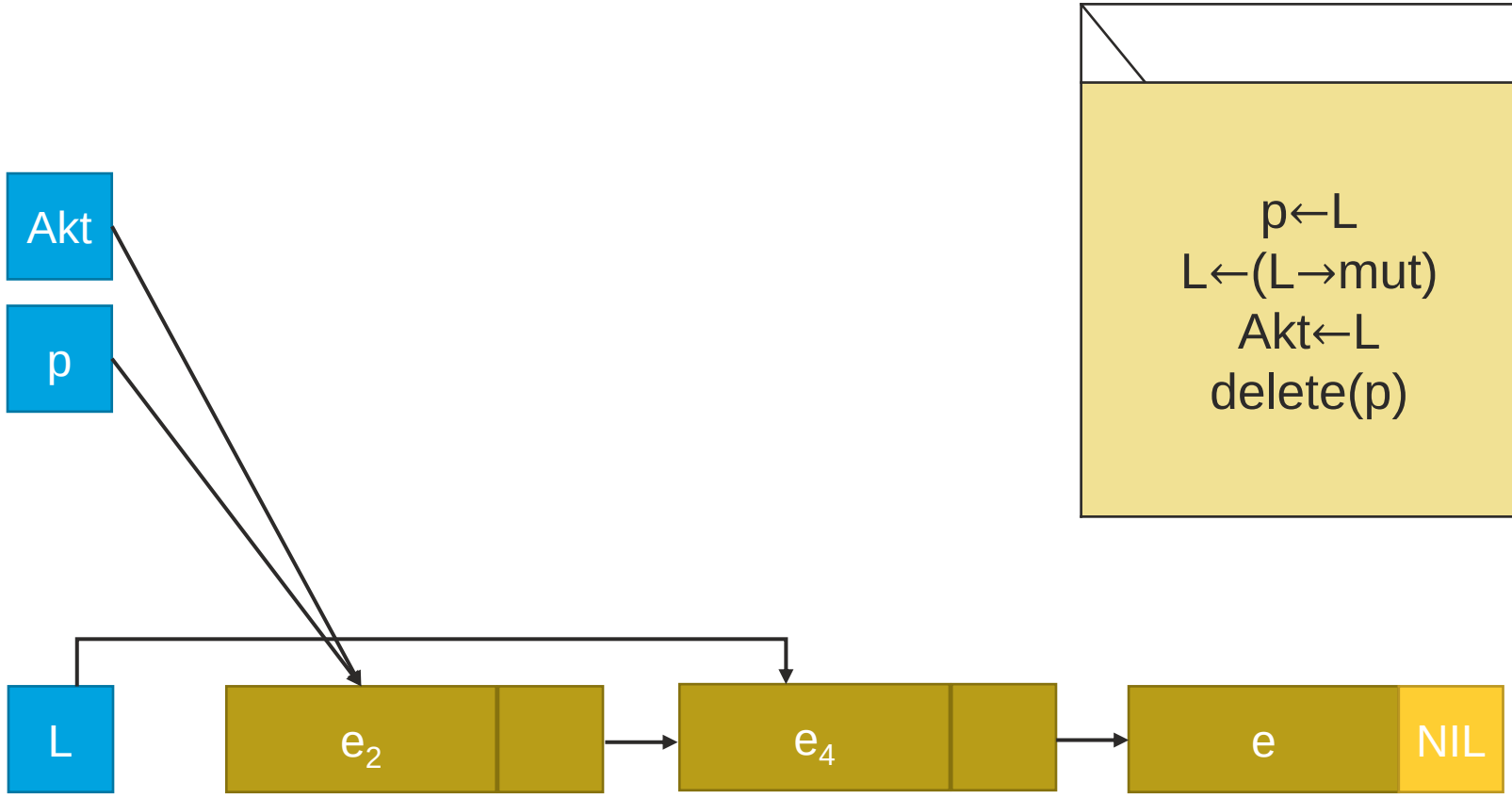



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L |
| | p→Mutató≠Akt |
| | p←(p→Mutató) |
| | (p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- Első elem esetén.

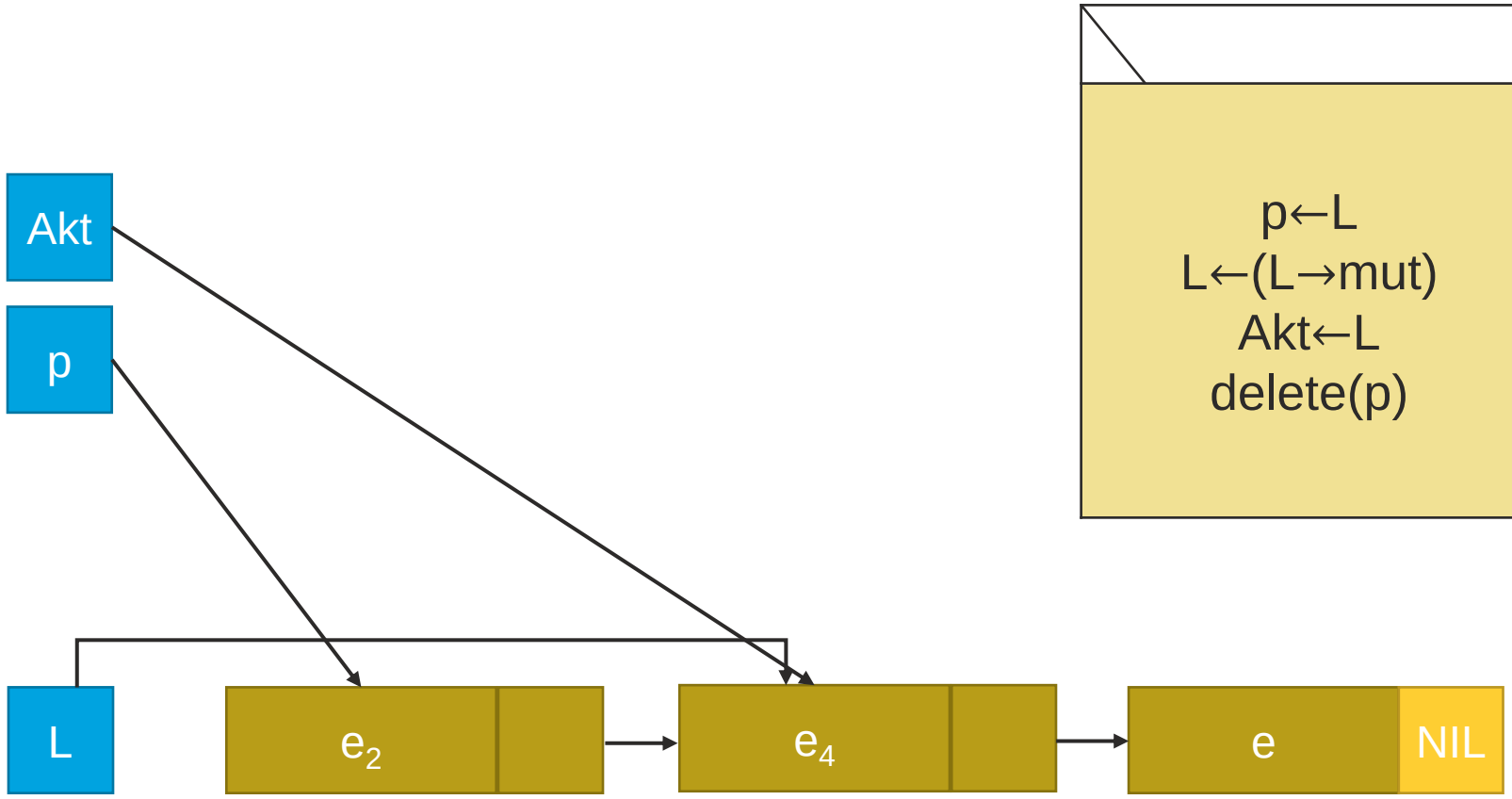



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L |
| | p→Mutató≠Akt |
| | p←(p→Mutató) |
| | (p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- Első elem esetén.

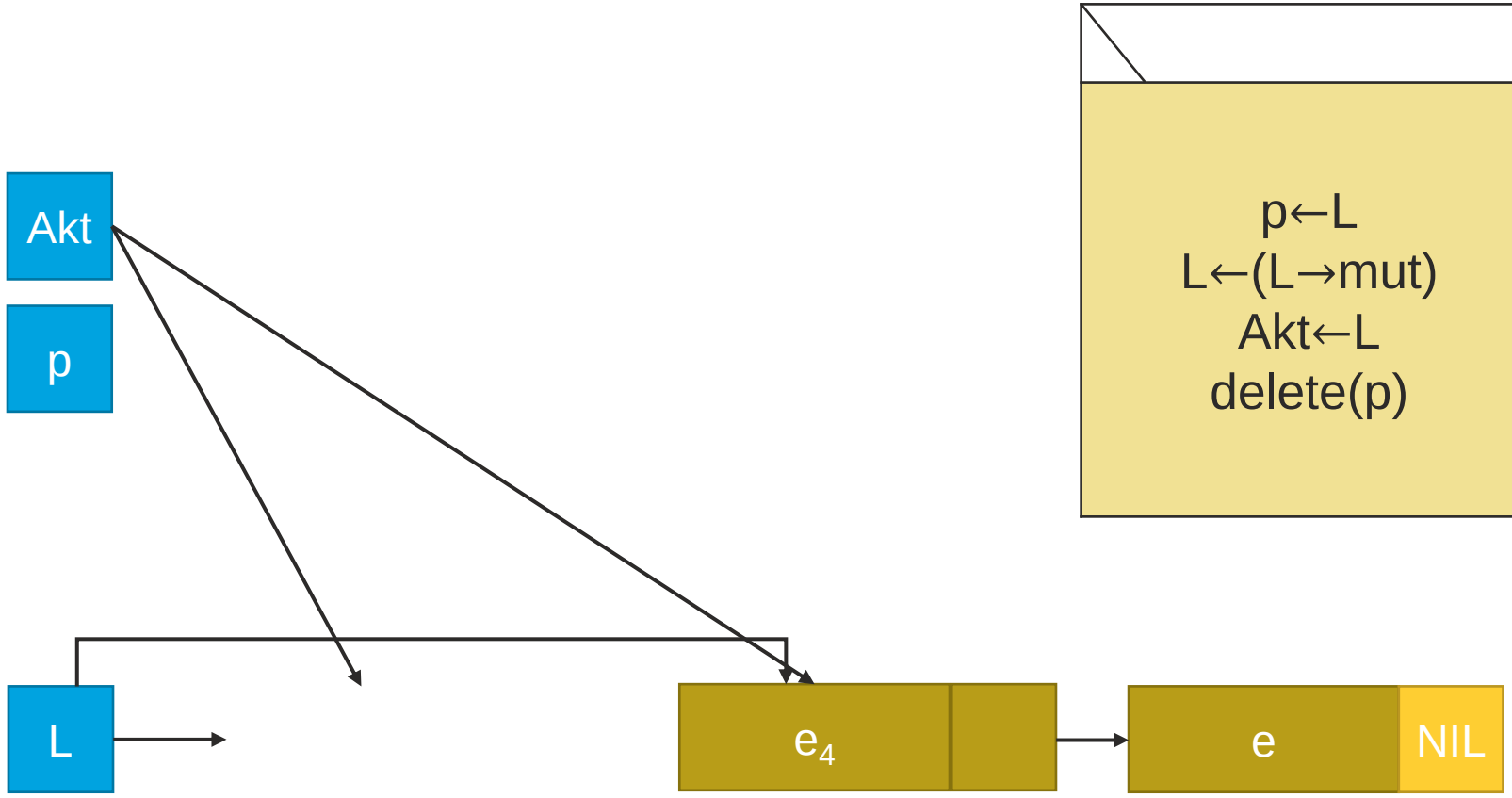



| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L |
| | p→Mutató≠Akt |
| | p←(p→Mutató) |
| | (p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Egyszeresen láncolt lista – RemoveAkt()

Aktuális elem törlése (feltételezzük, hogy létezik)

- Első elem esetén.




| Akt=L | |
|---|---|
| p←L<br>L←(L→mut)<br>Akt←L<br>delete(p) | p←L |
| | p→Mutató≠Akt |
| | p←(p→Mutató) |
| | (p→Mutató)←(Akt→Mutató)<br>delete(Akt)<br>Akt←(p→Mutató) |

# Hierarchikus adatszerkezetek

Következő téma