Verem, Sor

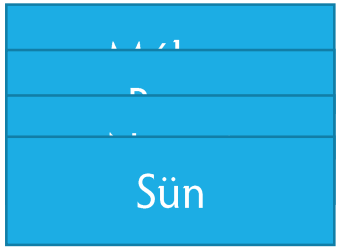
3. előadás

Verem

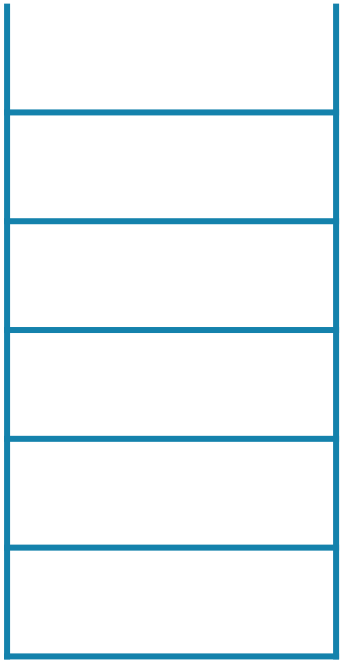
Stack, LIFO

Példa: VEREM

- LIFO: last-in, first-out
- Köznapi fogalma



- Kimászás sorrendje



Példa: A verem ADT axiomatikus leírása

E alaptípus feletti V verem típus jellemzése

- Műveletek
  - empty $\rightarrow V$
  - isempty $V \rightarrow L$
  - push $V \times E \rightarrow V$
  - pop $V \rightarrow V \times E$
  - top $V \rightarrow E$
- Megszorítások:
  - pop és top értelmezési tartománya
    - $V \backslash \{\text{empty}\}$

- Műveletek jelentése
  - Üres verem létrehozása
  - Üres a verem?
  - Elem betétele a verembe
  - Elem kivétele a veremből
  - Felső elem lekérdezése
- Figyelem!
  - A műveletek között nem szerepel „isfull" művelet!

A verem ADT axiomatikus leírása

- Axiómák
  - isempty(empty)
    - Vagy: $v = \text{empty} \rightarrow \text{isempty}(v)$
  - $\text{isempty}(v) \rightarrow v = \text{empty}$
  - $\neg\text{isempty}(push(v, e))$
  - $\text{pop}(\text{push}(v, e)) = (v, e)$
  - $\text{push}(\text{pop}(v)) = v$
  - $\text{top}(\text{push}(v, e)) = e$

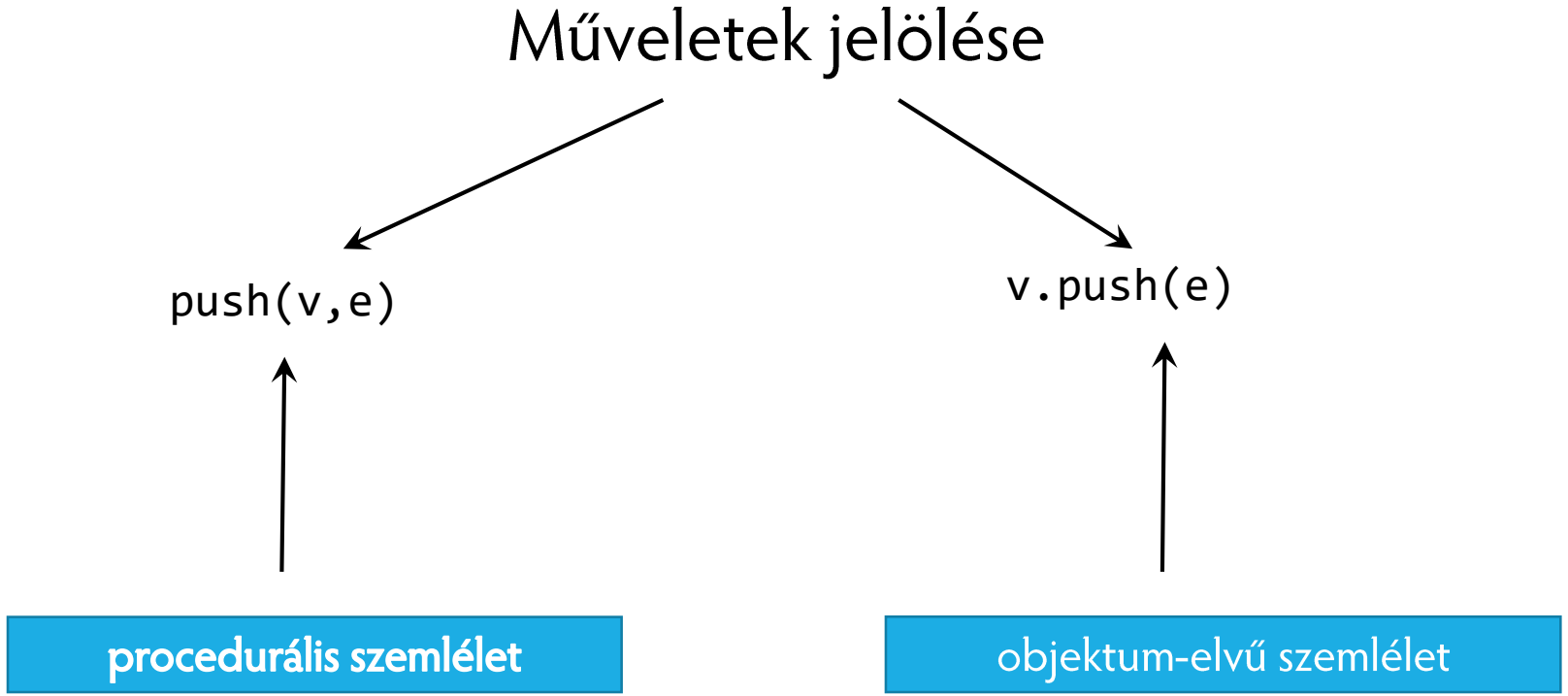
A verem ADT funkcionális leírása

- Matematikai reprezentáció
  - a verem rendezett párok halmaza
    $v = \{(e_1, t_1), \ldots (e_i, ti) |\ a\ t_j$ komponensek különbözőek$\}$
  1. komponens: a veremben elhelyezett (push) érték
  2. komponens: a verembe helyezés (push) időpontja
- Megszorítás (invariáns)
  - az idő értékek különbözők
- A valóságban nem így implementáljuk!

A verem ADT funkcionális leírása

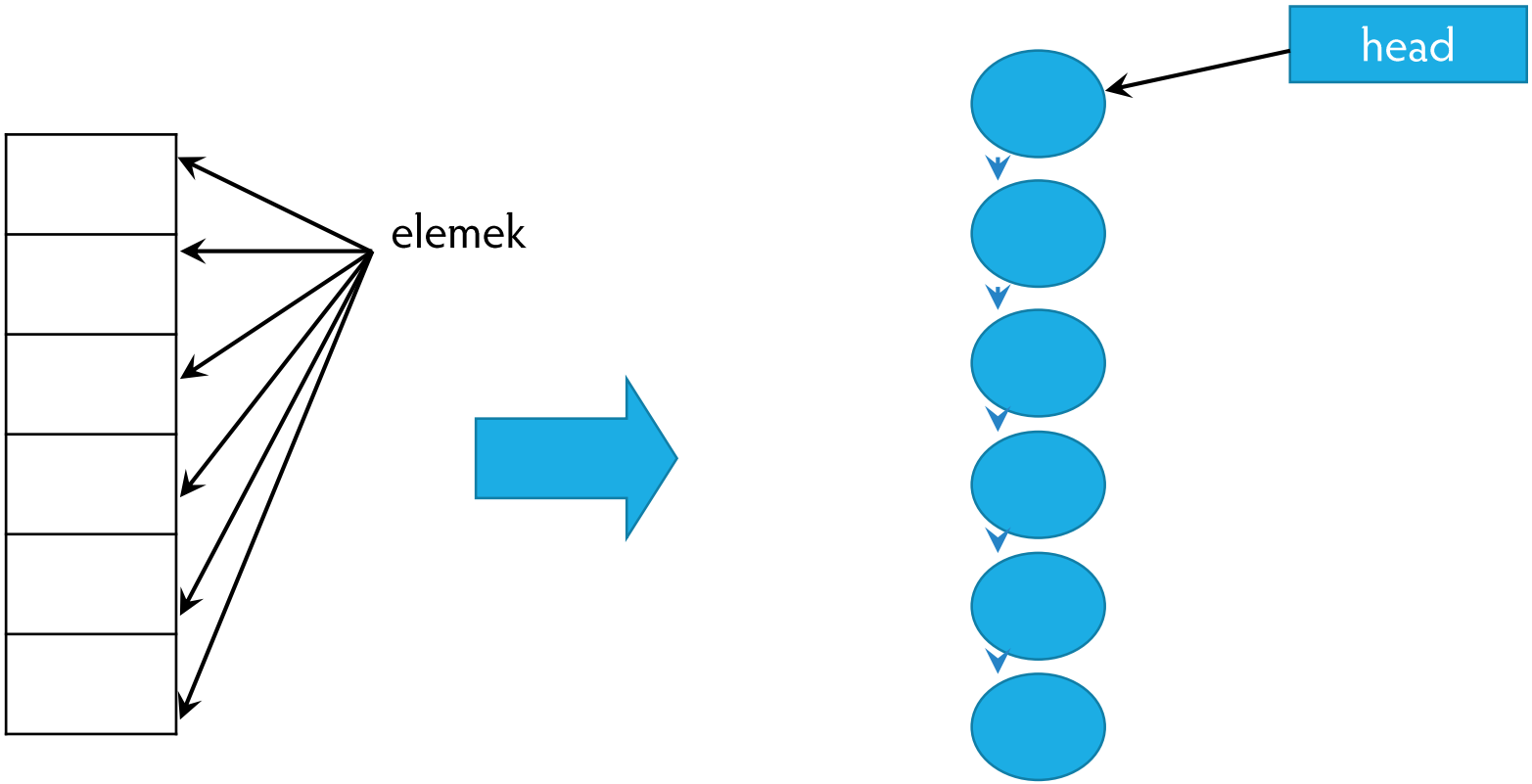
- A „pop" specifikációja:
  - Állapottér
    - $V \times E$ (Állapottér típusai: $V = \{(e_i, t_i), \dots\}$
    - $v \quad e$ (Állapottér változói)
  - Paramétertér
    - V és v'
  - Előfeltétel és utófeltétel:
  - $\mathrm{Ef} = (v = v' \wedge v' \neq \emptyset)$
  - $\mathrm{Uf} = \Big(\big(v = v' \backslash \{(e_j, t_j)\}\big) \wedge (e = e_j) \wedge \big((e_j, t_j) \in v'\big) \wedge \big(\forall i \big((e_i, t_i) \in v' \wedge i \neq j\big) : t_j > t_i\big)\Big)$

Verem


Műveletek jelölése
push(v,e)
v.push(e)
procedurális szemlélet
objektum-elvű szemlélet

Verem – ADS

- Lineáris adatszerkezet

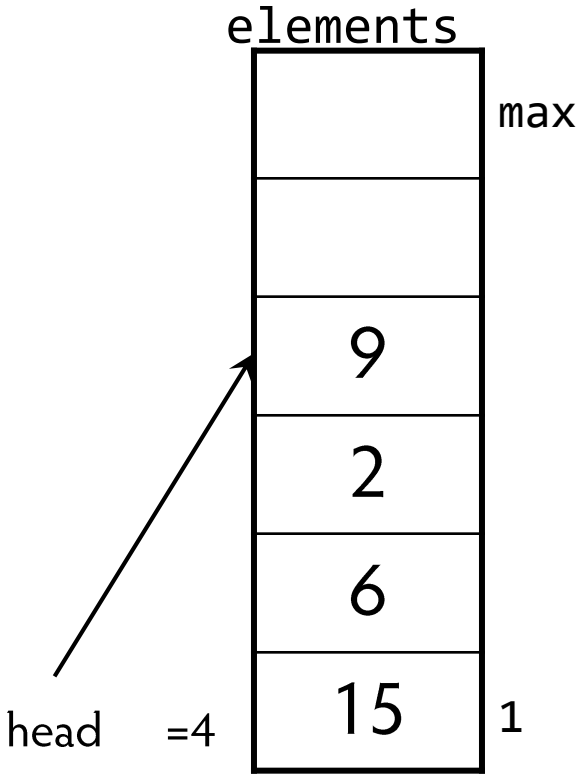


Elemek száma

- Statikus vs. Dinamikus
  - Az adatszerkezet elemeinek száma a feldolgozás során rögzített vagy változtatható
  - A rögzített nem jelenti, hogy a tárolt adatok nem megváltoztathatók
- Kapacitás: fix vs. változó
  - Fix: A tárolható adatelemek számának felső korlátja a létrehozáskor (esetleg fordítási időben) rögzített.
  - Változó: A memória mérete (illetve kapcsolódó technikai korlátok) szab határt az adatelemek számának

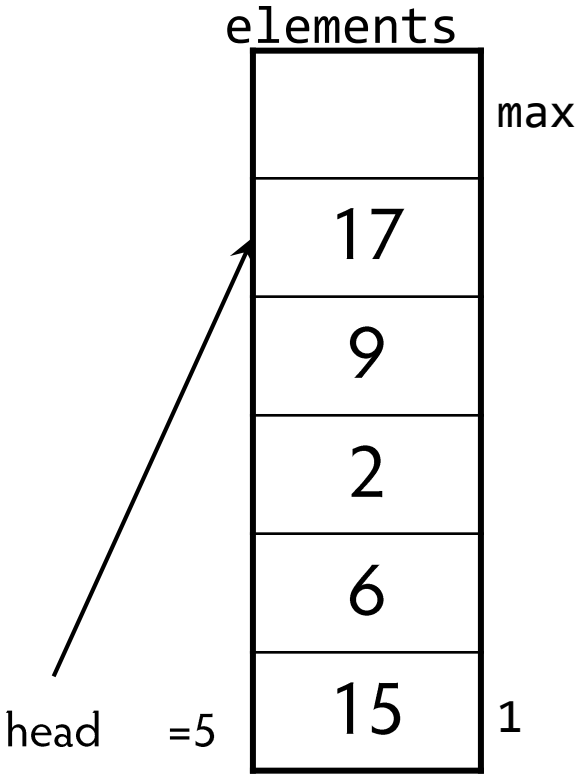
Reprezentáció

- Aritmetikai ábrázolás:
  - egy max hosszú vektor (ez az elemek tömbje) elements[1..max]
  - a verem tetejének mutatója head $\in$ [0, max] head=0 $\Leftrightarrow$ üres a verem
  - Választási lehetőség, hogy hova mutat a head
    - Az első szabad helyre
    - Az utolsó elfoglalt helyre



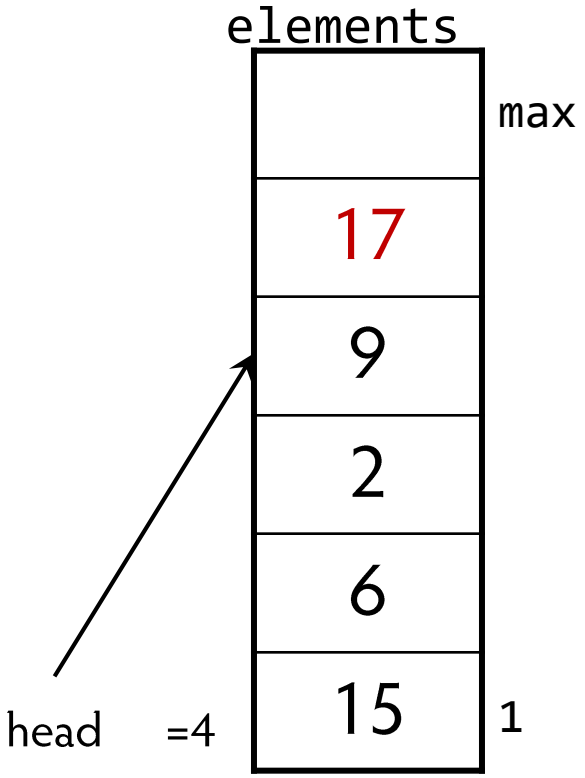
Reprezentáció

v.push(17) után



v.pop() után

Egyenlő a két verem?



Implementáció

- Műveletek pszeudokódja/struktogramja:
  - `v.empty`

    ```
    -- üresre állítja a vermet
    v.head ← 0
    ```

  - `v.isempty`

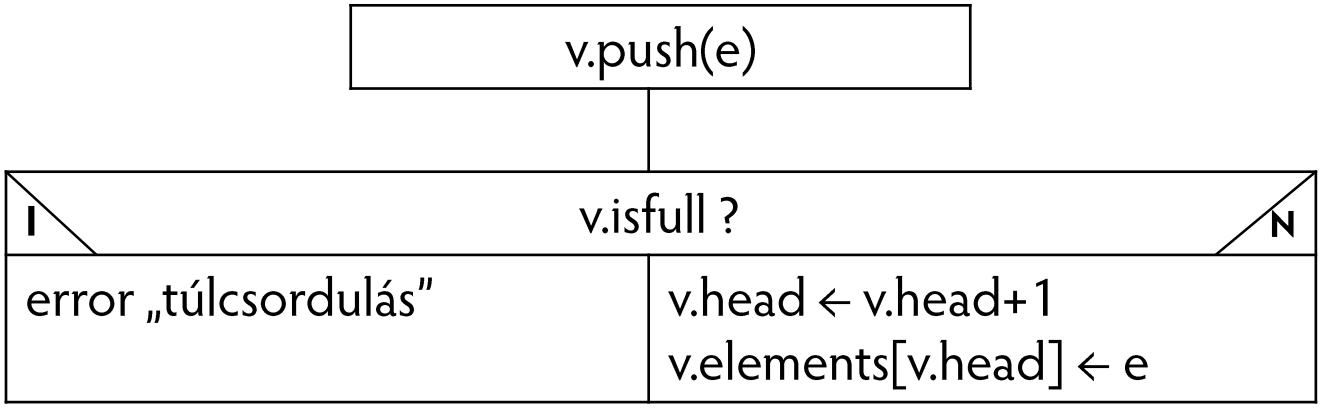
    ```
    -- Üres a verem? – Logikai értéket ad vissza
    return (v.head=0)
    ```

  - `v.isfull`

    ```
    -- Tele van a verem? – Logikai értéket ad vissza
    return (v.head=max)
    ```

Implementáció

- v.push(e)
  -- e-t beteszi a v verem tetejére
  if v.isfull
    then error „túlcsordulás”
    else v.head ← v.head +1
    v.elements[v.head] ← e
  end if

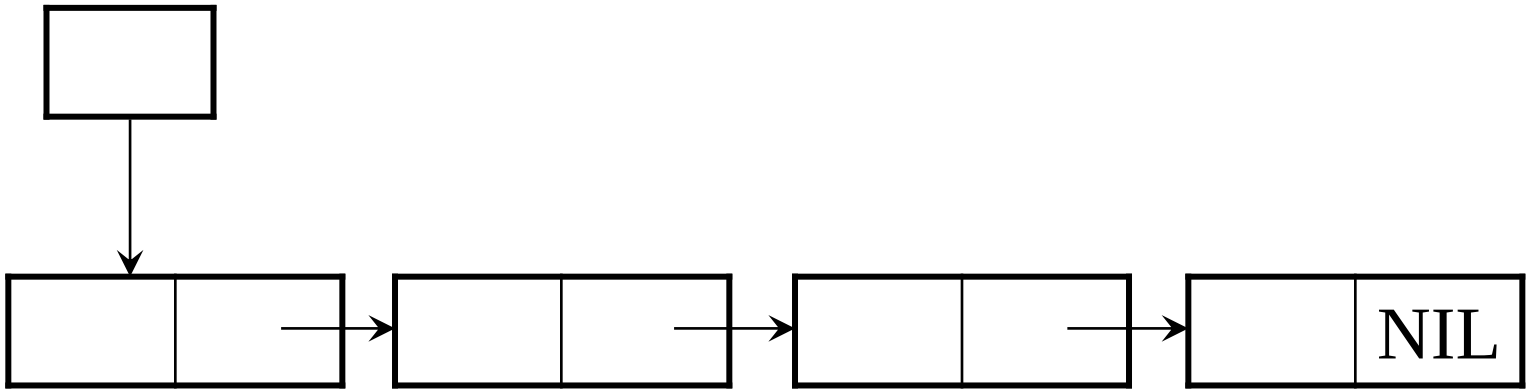


Implementáció

- ```
  v.pop
  -- kiveszi a legfelső elemet és visszaadja
  if v.isempty
    then error „alulcsordulás”
    else v.head ←v.head -1
      return v.elements[v.head+1]
  end if
  ```
  ```

Implementáció

- ```
  v.top
  -- lekérdezi a legfelső elemet
  if v.isempty
    then error „alulcsordulás”
    else return v.elements[v.head]
  end if
  ```
  ```

Reprezentáció

- Lesz még láncolt ábrázolás is! (Gyakorlaton)

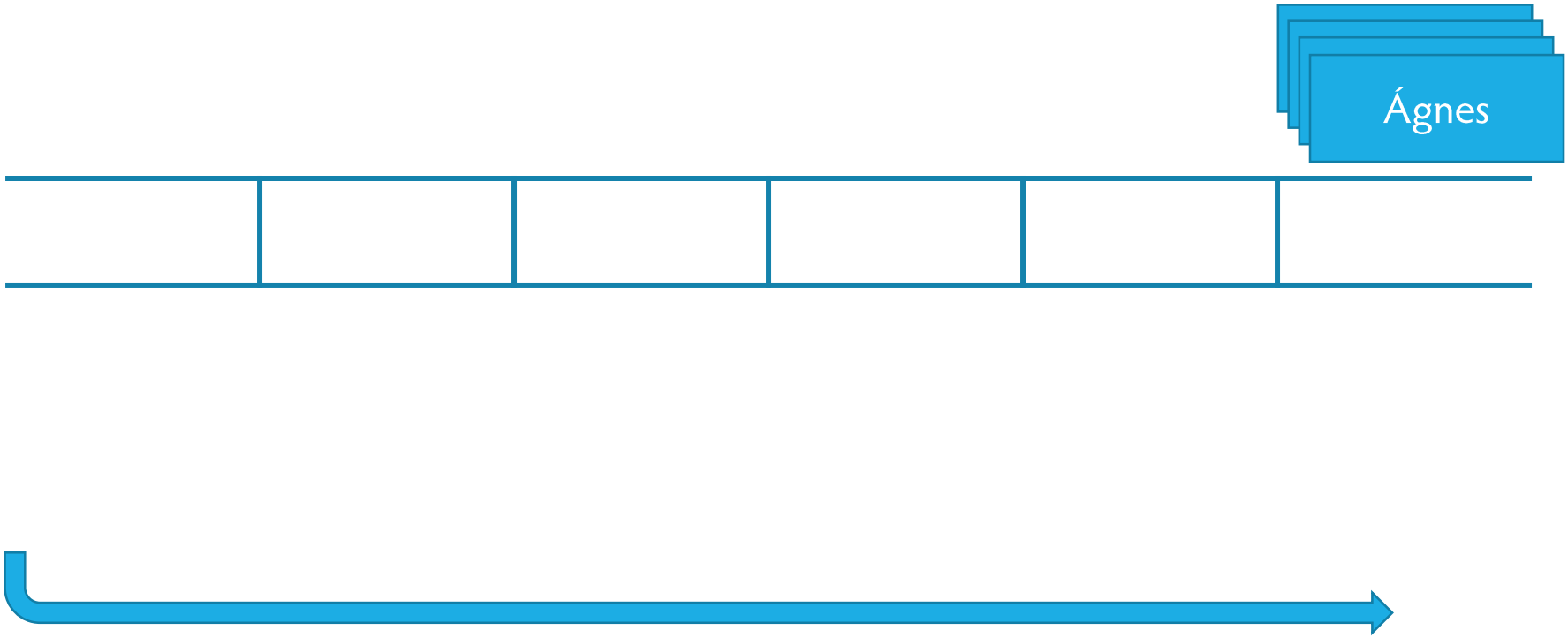


Sor

Queue, FIFO

Sor (Queue)

- FIFO: First-in, First-out
- Köznapi fogalma



A sor ADT axiomatikus leírása

E alaptípus feletti S sor típus jellemzése

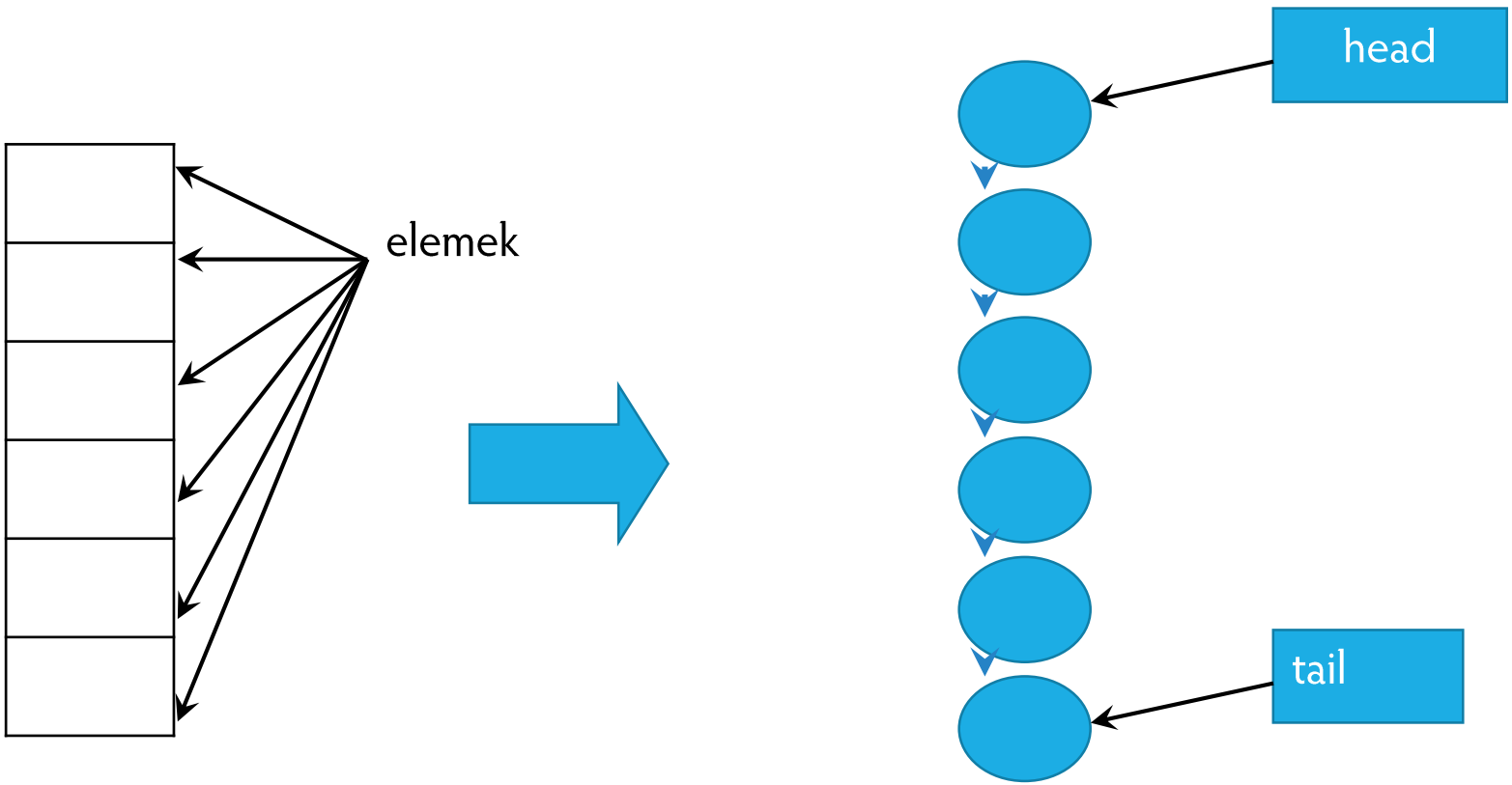
- Műveletek
  - empty $\rightarrow S$
  - isempty $S \rightarrow L$
  - in $S \times E \rightarrow S$
  - out $S \rightarrow S \times E$
  - first $S \rightarrow E$
- Megszorítások:
  - out és first értelmezési tartománya
    - $S \backslash \{\text{empty}\}$
- Műveletek jelentése
  - Üres sor létrehozása
  - Üres a sor?
  - Elem betétele a sorba
  - Elem kivétele a sorból
  - Első elem lekérdezése
- Figyelem!
  - A műveletek között nem szerepel „isfull" művelet!

Sor – ADT

- Axiómák:
  - $s = \text{empty} \rightarrow \text{isempty}(s)$
  - $\text{isempty}(s) \rightarrow s = \text{empty}$
  - $\neg\text{isempty}(\text{in}(s, e))$
  - $\neg\text{isempty}(s) \rightarrow \text{out}(\text{in}(s, e))_2 = \text{out}(s)_2$
  - $\neg\ \text{isempty}(s) \rightarrow \text{in}(\text{out}(s)_1, e) = \text{out}(\text{in}(s, e))_1$
  - $\text{isempty}(s) \rightarrow \text{out}\big(\text{in}(s, e)\big)_2 = e$
  - $\text{first}(s) = \text{out}(s)_2$
  - Ahol: $(s,e)_1 = s$, $(s,e)_2 = e$
- Végig kell gondolni, önállóan – vizsgakérdés!

Sor – ADS

- Lineáris adatszerkezet

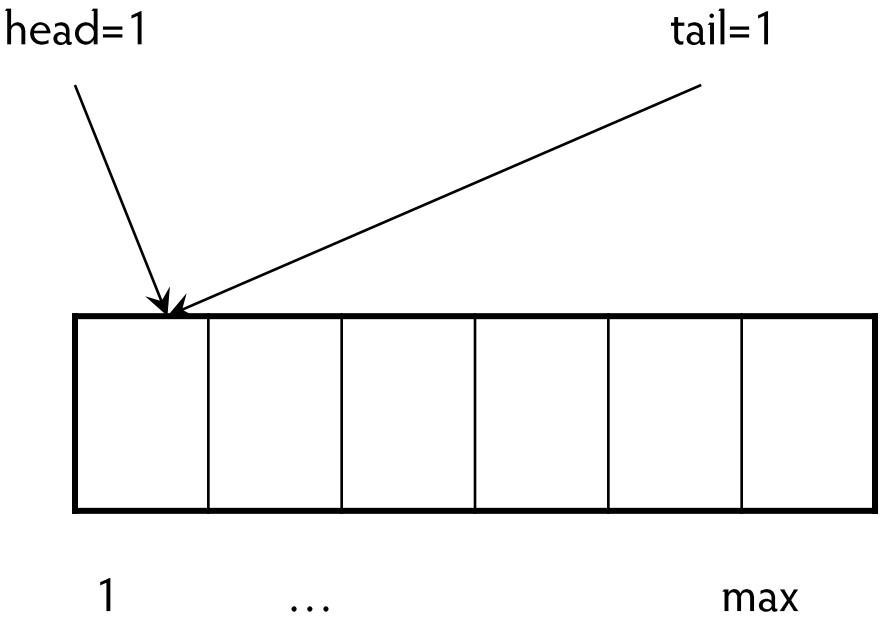


Reprezentáció

- Aritmetikai ábrázolás:
  - egy max hosszú vektor
    - ez az elemek tömbje
    - $\mathrm{elements}[1..\mathrm{max}]$
  - és a sor első elemének mutatója
    - $\mathrm{head} \in [1, \mathrm{max}]$
  - és a sor első üres (utolsó) helyének mutatója
    - $\mathrm{tail} \in [1, \mathrm{max}]$
- Vegyük észre, hogy az aritmetikai ábrázolás három részből áll!

Reprezentáció

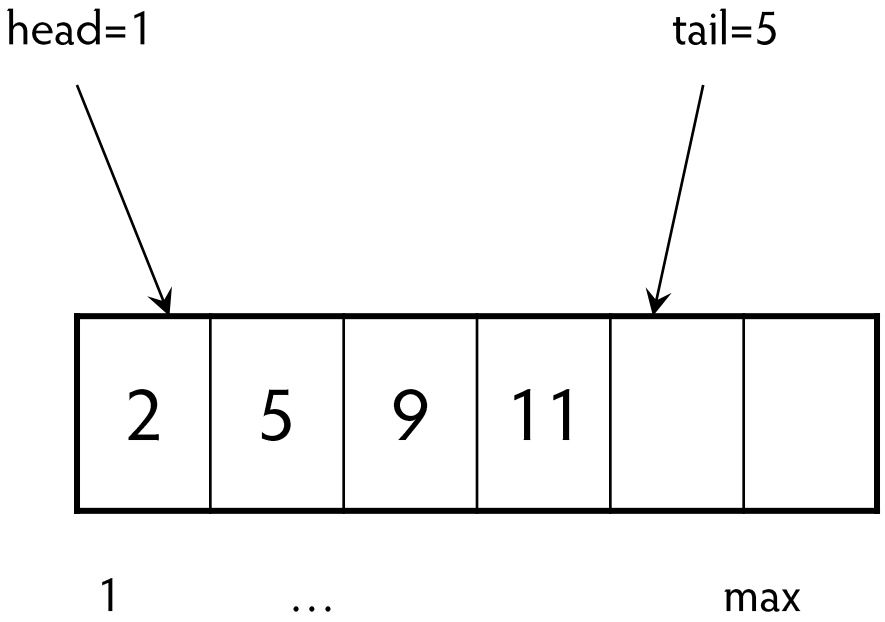
- Ciklikus ábrázolással
  - Kezdetben



Üres a sor

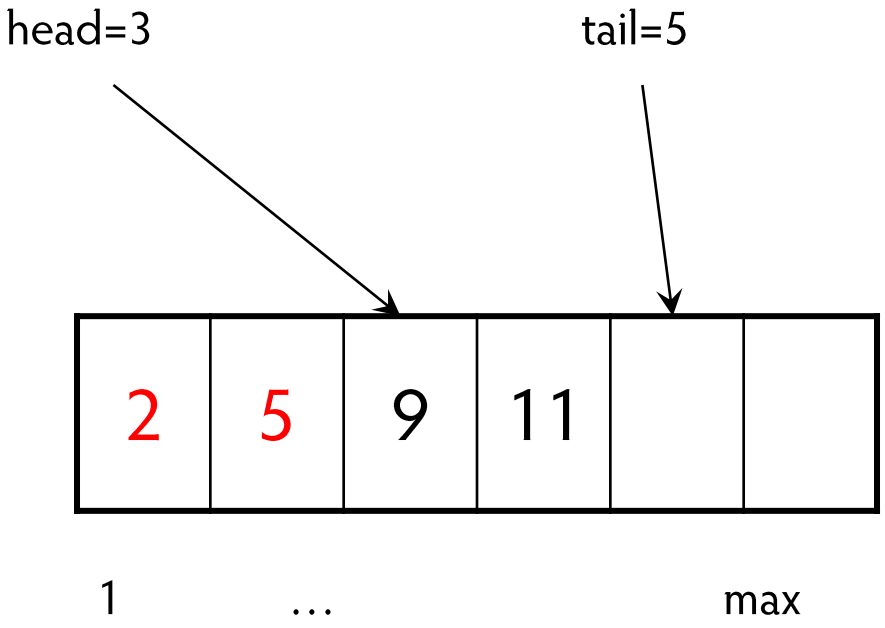
Reprezentáció

- 2, 5, 9, 11 betétele után:



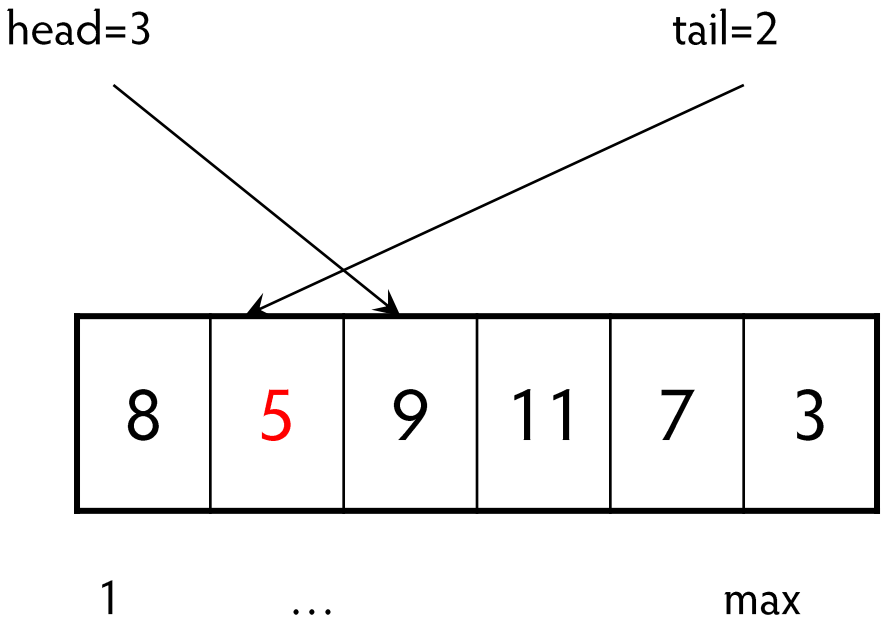
Reprezentáció

- 2, 5 kivétele után:



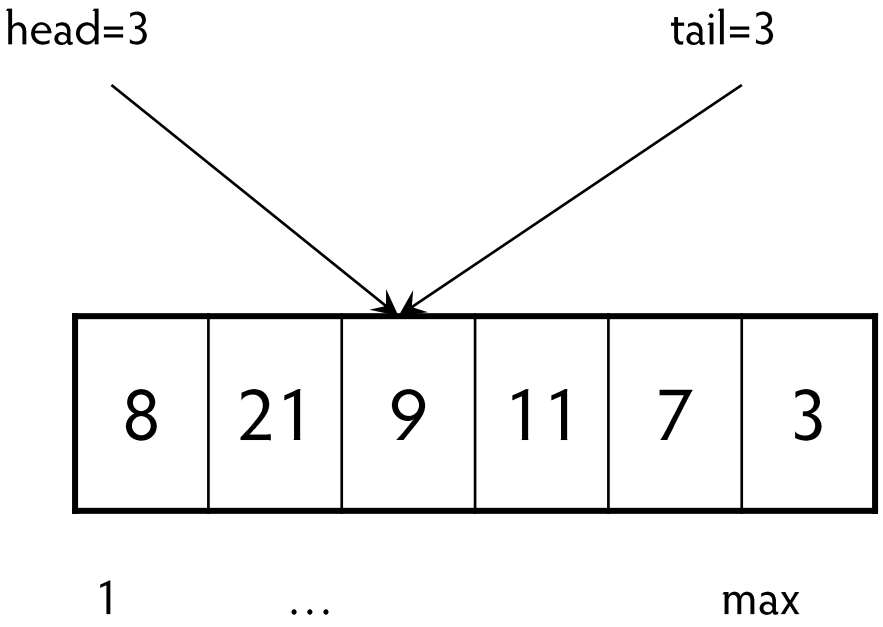
Reprezentáció

- 7, 3, 8 betétele után:



Reprezentáció

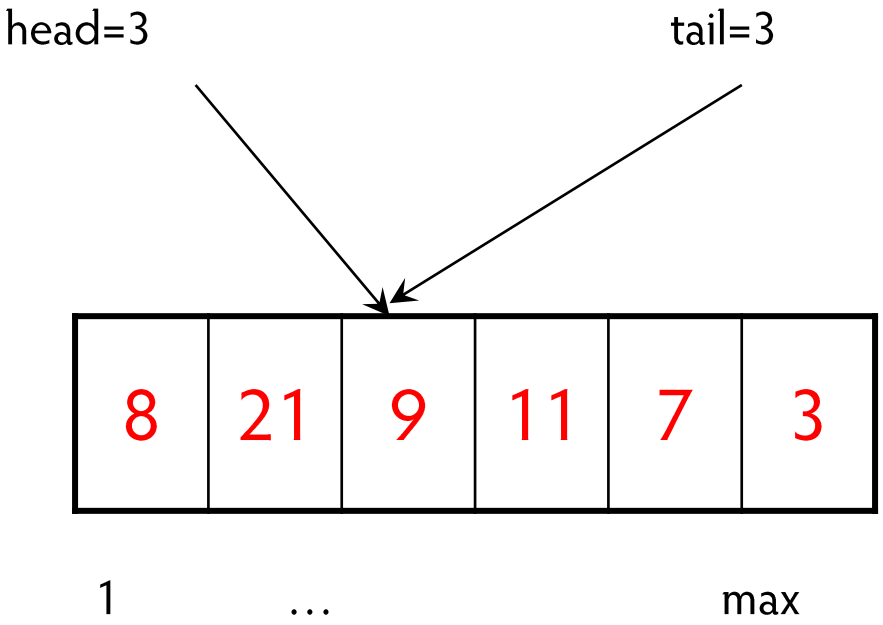
- 21 betétele után:



Tele a sor

Reprezentáció

- 9, 11, 7, 3, 8, 21 kivétele után:



Üres a sor

Reprezentáció

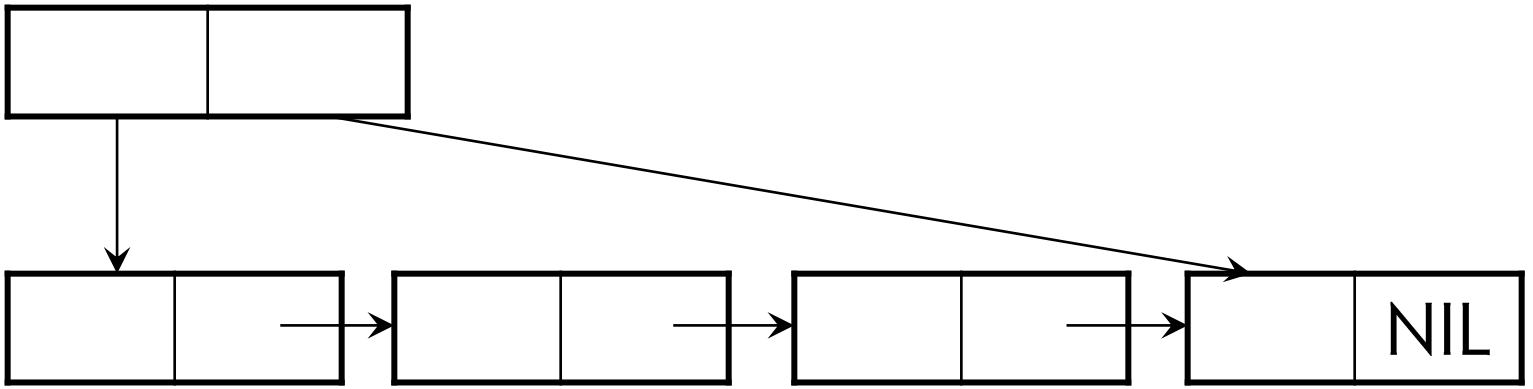
- Mit tegyünk? – Milyen lehetőségek vannak:
  - Vezessünk be még egy jelzőt a reprezentációba, ami mutatja, hogy a sor üres-e
    - empt – kezdetben igaz, később vizsgáljuk, és megfelelően állítjuk
  - Vezessünk be még egy attribútumot a reprezentációba, ami mutatja, hogy hány elem van a sorban - count

Implementáció

- Műveletek pszeudokódja:
  - ```
    s.empty
    -- üresre állítja a sort
    s.head ← 1; s.tail ← 1; s.empt ←true
    ```
  - ```
    s.isempty
    -- üres a sor? - logikai értéket ad vissza
    return s.empt
    ```
  - ```
    s.isfull
    -- tele van a sor?
    return ((not s.empt) and (s.head = s.tail))
    ```
    ```

Implementáció

- s.In(e)

```
-- e-t beteszi az s sor végére
-- s.tail-t ciklikusan növeli
if s.IsFull
  then error „túlcsordulás”
  else s.empt ← false
    s. elements[s.tail] ← e
    if s.tail=max
      then s.tail ← 1
      else s.tail ← s.tail+1
    end if
end if
```

Implementáció

- s.Out

```
-- kiveszi és visszaadja az s sor első elemét
-- s.head-et ciklikusan növeli
-- figyeli, hogy nem üres-e a sor
if s.empt
  then error „alulcsordulás”;
  else e ← s.elements[s.head]
       if s.head=max
         then s.head ← 1
         else s.head ← s.head+1
       end if
       if s.head=s.tail then s.empt ← true end if
       return e
end if
```

Implementáció

- ```
  s.First
  -- visszaadja az s sor első elemét,
  -- figyeli, hogy nem üres-e a sor
  if s.empt
    then error „alulcsordulás”
    else return s. elements[s.head]
  end if
  ```

- Lehetne az is, hogy a darabszámot tároljuk
  - Házi feladat: átgondolni
  ```

Reprezentáció

- Lesz még láncolt ábrázolás is!



Implementáció

- Vigyázni kell, amikor a programokat a választott programnyelven megvalósítjuk!
  - Például
    - C++ nyelv esetén a vektorok indexelése nullával kezdődik!
  - Értékadás jele
  - Egyenlőség vizsgálat jele

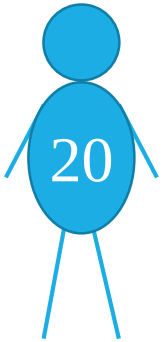
Prioritásos sor

Egyszerű sor → prioritásos sor

- Egyszerű sor:
  - FIFO szemantika
  - Elem hozzáadása, törlése konstans ($\mathcal{O}(1)$) igényű
- Mit tegyünk, ha a sorban lévő elemeknek valamifajta rendezése is van?
  - Ezt általában prioritásnak nevezzük.
  - Úgy kell rendezzük az elemeket, hogy a legnagyobb (legkisebb) prioritású elemet töröljük először.

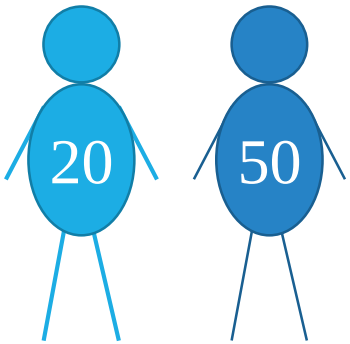
Elsőbbségi (prioritásos) sor

- Példa
  - sürgősségi osztály
    - különböző súlyosságú esetek



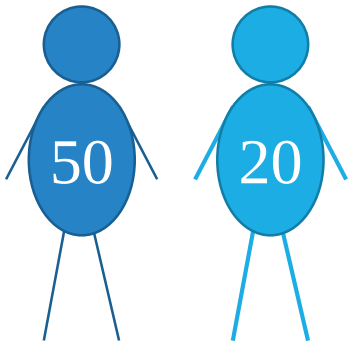
Elsőbbségi (prioritásos) sor

- Példa
  - sürgősségi osztály
    - különböző súlyosságú esetek



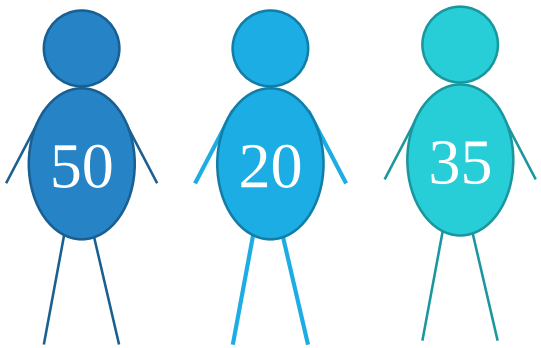
Elsőbbségi (prioritásos) sor

- Példa
  - sürgősségi osztály
    - különböző súlyosságú esetek



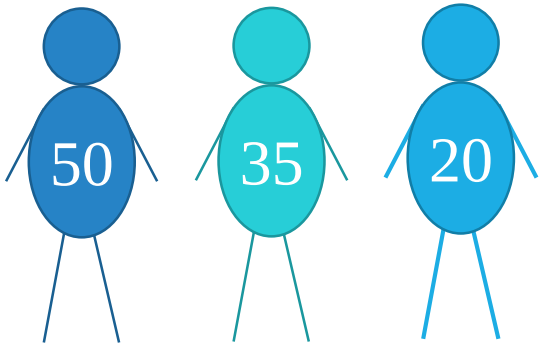
Elsőbbségi (prioritásos) sor

- Példa
  - sürgősségi osztály
    - különböző súlyosságú esetek



Elsőbbségi (prioritásos) sor

- Példa
  - sürgősségi osztály
    - különböző súlyosságú esetek



Elsőbbségi (prioritásos) sor

- Mire lehet használni?
  - Tennivalók, melyiket kell először elvégezni
  - Operációs rendszer, mely prioritásos feladatokat (jobokat) dolgoz fel
    - Itt különböző további algoritmusok, amelyek a prioritást meghatározzák egy-egy folyamat (job) számára
  - Telekommunikációban a csomagok továbbításánál is

Elsőbbségi (prioritásos) sor

- Az elsőbbségi sor ADT axiomatikus leírása
- E alaptípus feletti P elsőbbségi sor típus jellemzése:
  - Egyszerűsítés: csak prioritásokat teszünk bele (N)
  - Műveletek „full" nem szerepel
    - empty $\rightarrow P$ az üres prior. sor konstruktor – létrehozás
    - isempty $P \rightarrow L$ üres a prioritásos sor?
    - insert: $P \times N \rightarrow P$ elem betétele a prioritásos sorba
    - delmax: $P \rightarrow P \times N$ maximális elem kivétele a pr. sorból
    - max: $P \rightarrow N$ maximális elem lekérdezése
  - Megszorítások:
    - delMax és max értelmezési tartománya $P \setminus \{empty\}$

Elsőbbségi (prioritásos) sor

- Axiómák:
  1. $\text{isempty}(\text{empty})$ vagy $p = \text{empty} \rightarrow \text{isempty}(p)$
  2. $\text{isempty}(p) \rightarrow p = \text{empty}$
  3. $\neg\text{isempty}(\text{insert}(p, n))$
  4. $\text{insert}(\text{delmax}(p)) = p$
  5. $\max(p) = \text{delmax}(p)_2$
  6. $\text{delmax}(p)_1 \neq \text{empty} \rightarrow \max(p) \geq \max(\text{delmax}(p)_1)$
  7. $n \geq \max(p) \rightarrow \text{delmax}(\text{insert}(p, n))_1 = p \wedge \max(\text{insert}(p, n)) = n$
  8. $n < \max(p) \rightarrow \max(\text{insert}(p, n)) = \max(p)$
  9. $\text{delmax}(\text{insert}(\text{empty}, n)) = (\text{empty}, n)$
  10. $\max(\text{insert}(\text{empty}, n)) = n$
  - (7. és 8.-nál feltettük, hogy nem üres a prioritásos sor – ez az értelmezési tartomány megszorítása!)

Elsőbbségi (prioritásos) sor

- Kérdés: hogyan ábrázoljuk, mivel reprezentáljuk?
  - Rendezetlen tömbbel, a beérkezési idő szerint
    - max műveletigénye mindig egy maxker, vagyis $\Theta(n)$
  - Rendezett tömbbel
    - insert műveletigénye:
      - A hely megkeresése → logaritmikus keresés $\Theta(\log_2 n)$
      - Tőle jobbra léptetés $\Theta(n)$
      - Összesen $\Theta(n)$
  - Heap (kupac) adatszerkezettel
    - később

Lengyel forma

Az előzőek alkalmazása

Kifejezések

- Infix
  - Amikor az operátor az operandusok között van
  - $a + b$
- Postfix
  - Amikor az operátor az operandusok mögött van
  - $a\ b\ +$
- Prefix
  - Amikor az operátor az operandusok előtt van
  - $+a\ b$

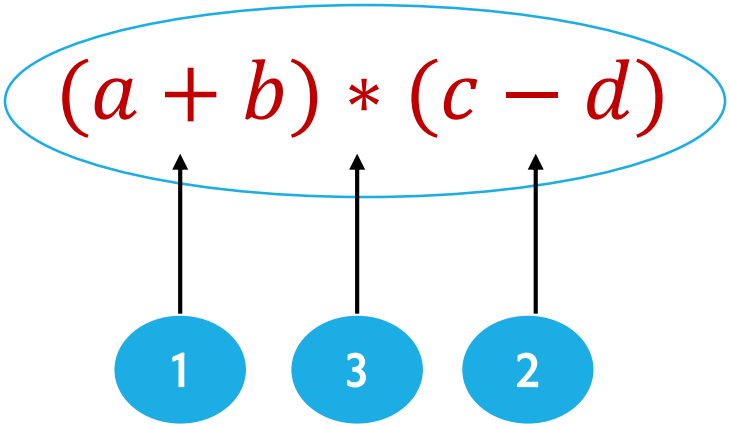
Lengyel forma

- J. Lukasewitz lengyel matematikus használta először
  - Prefix forma – lengyel forma
  - Postfix forma – fordított lengyel forma
- Átalakítás
  - Infix kifejezés
    - $(a + b) * (c - d)$
  - Postfix kifejezés
    - $a\ b + c\ d\ -\ *$
- Az Infix kifejezés átalakítható Postfix kifejezéssé
  - A kifejezések feldolgozásánál két fontos előnye van

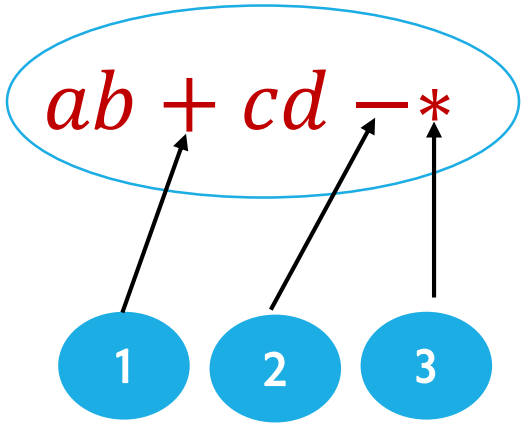
Fordított lengyel forma – előnyök

- A műveleti jelek olyan sorrendben követik egymást, amilyen sorrendben végre kell hajtani azokat.

Infix forma

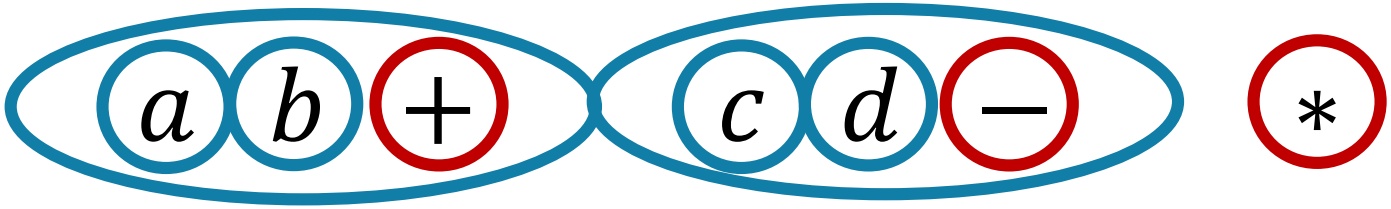


Lengyelforma



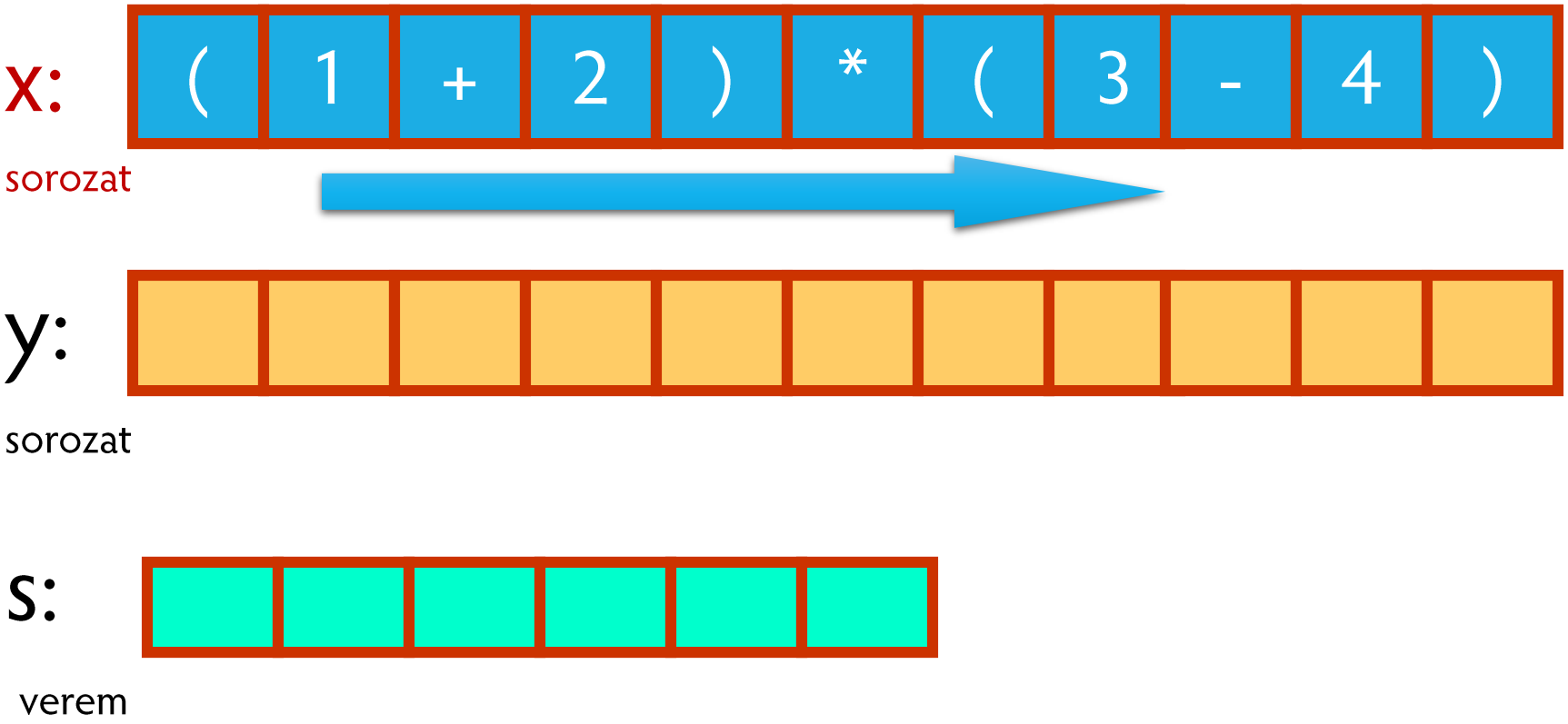
Fordított lengyel forma – előnyök

- A műveleti jel (operátor) közvetlenül az operandusai után áll
  - Infix forma: $(a + b) * (c - d)$
  - Postfix forma
    - Operátor
      Operandus

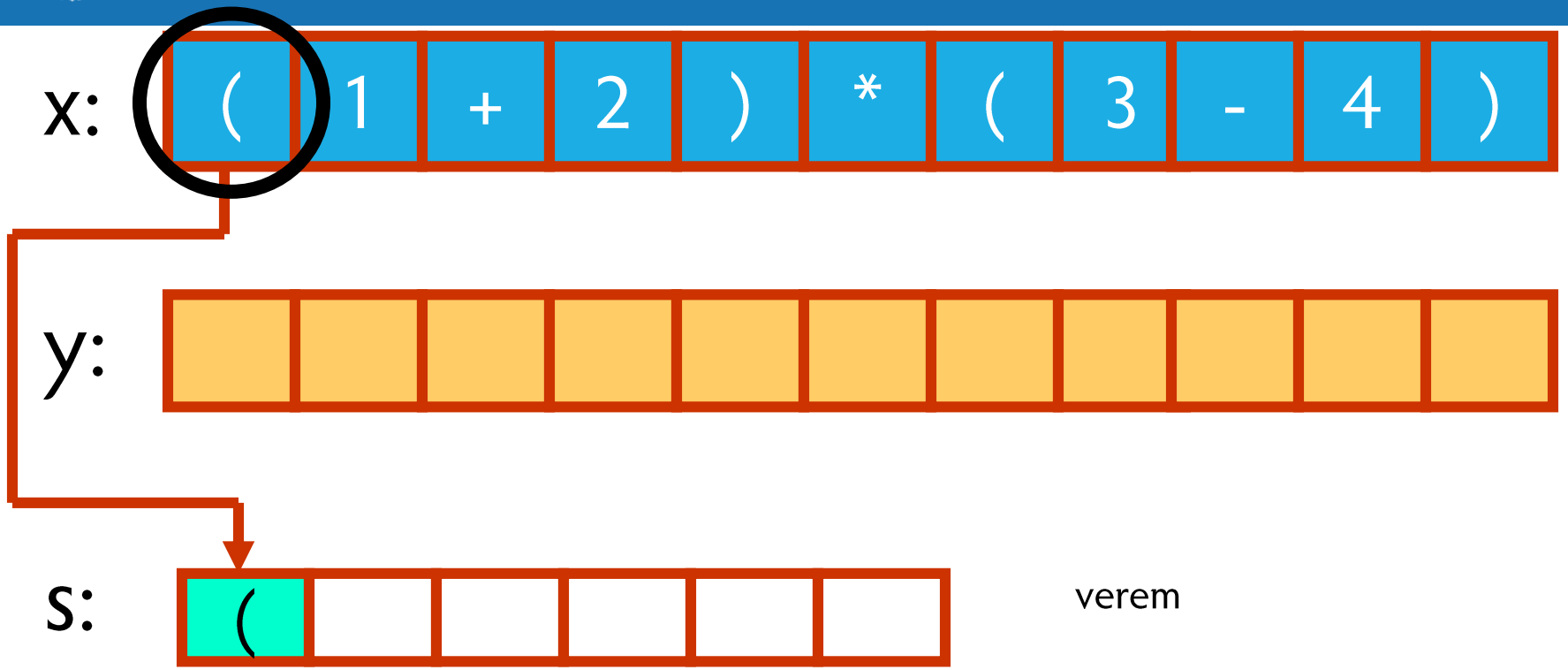


További példák

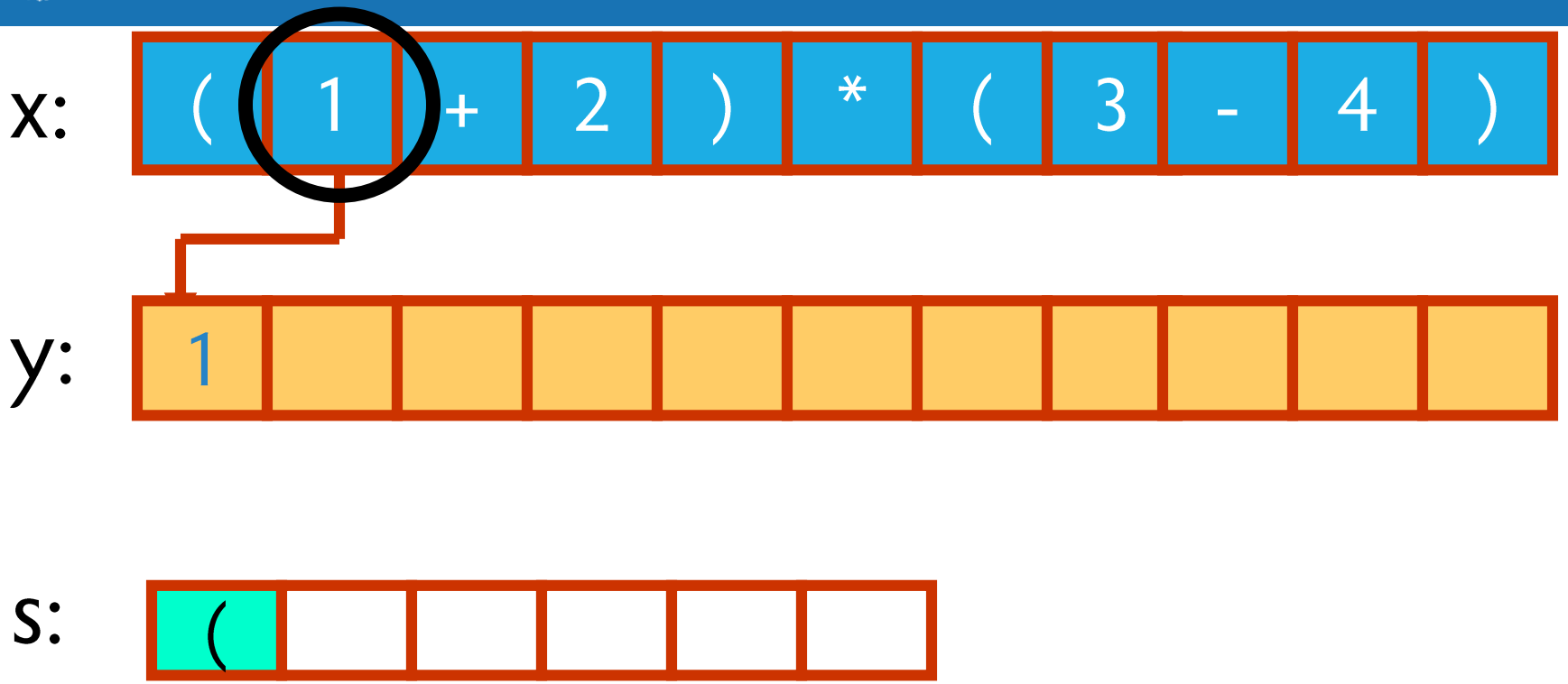
- Kifejezés lengyel formára hozása
- Szokásos forma
  - $(1 + 2) * (3 - 4)$
  - $(a + b * c) * (d * 3 - 4)$
- Fordított lengyel forma
  - $1\ 2\ +\ 3\ 4\ -\ *$
  - $abc * +d\ 3 * 4\ - *$



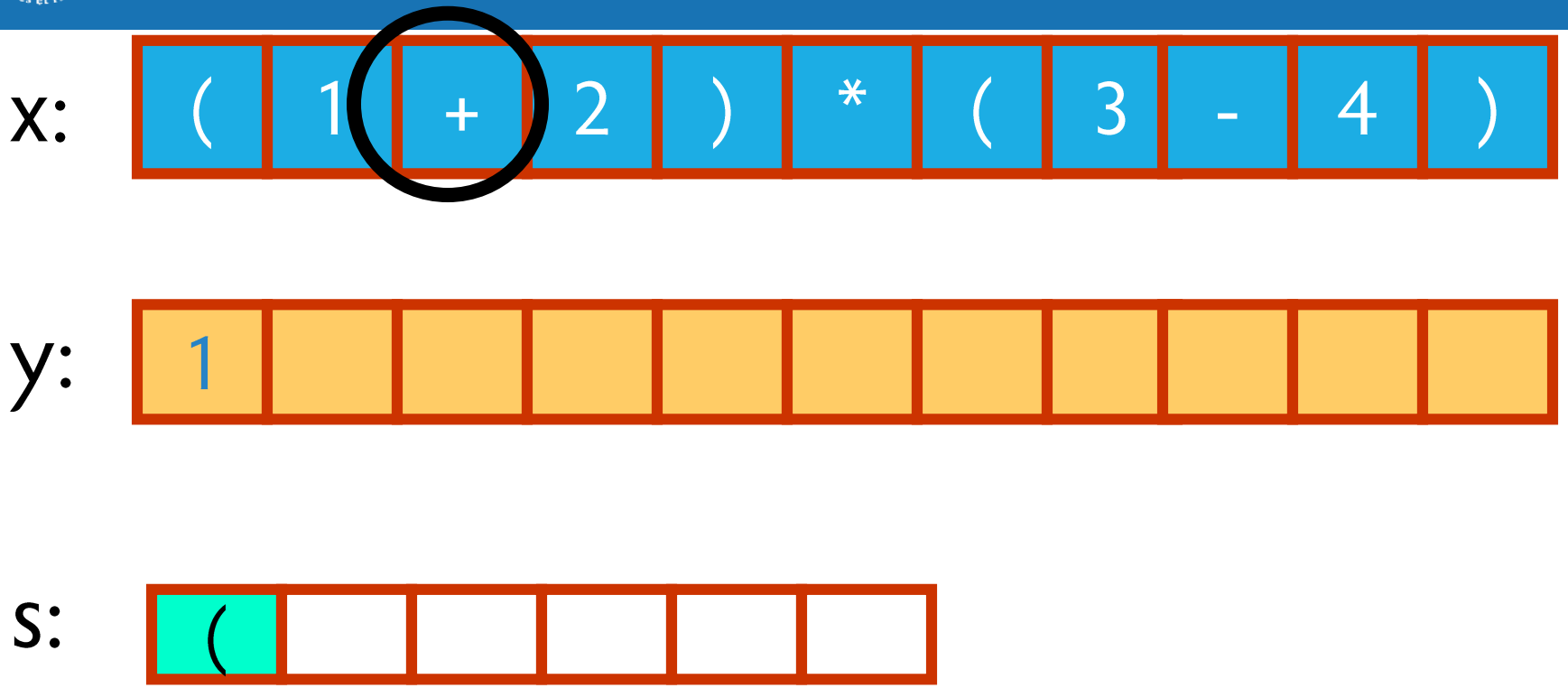
Az x sorozatot balról jobbra haladva dolgozzuk fel.
A sorozat végét ; jelzi.



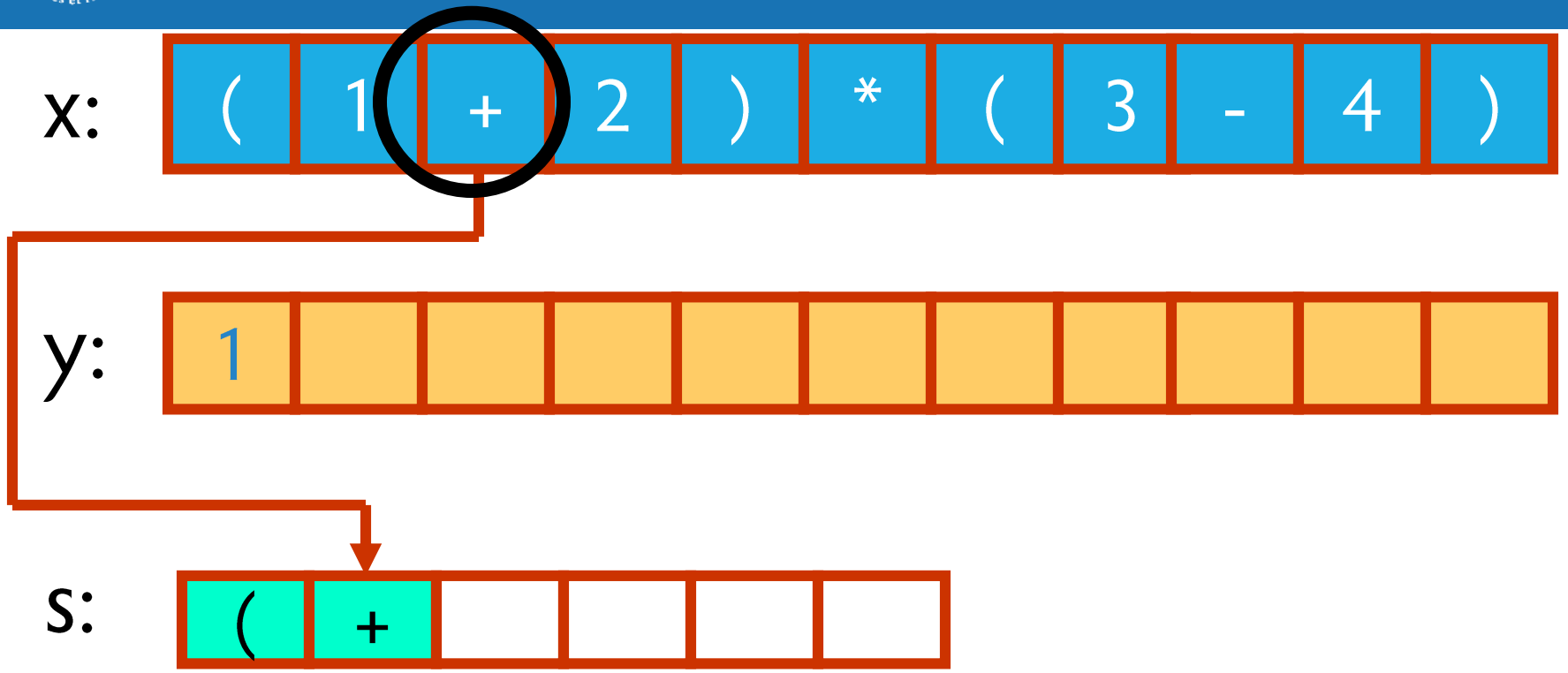
A következő szimbólum nyitózárójel: tegyük a verembe.



A következő szimbólum operandus: írjuk ki.

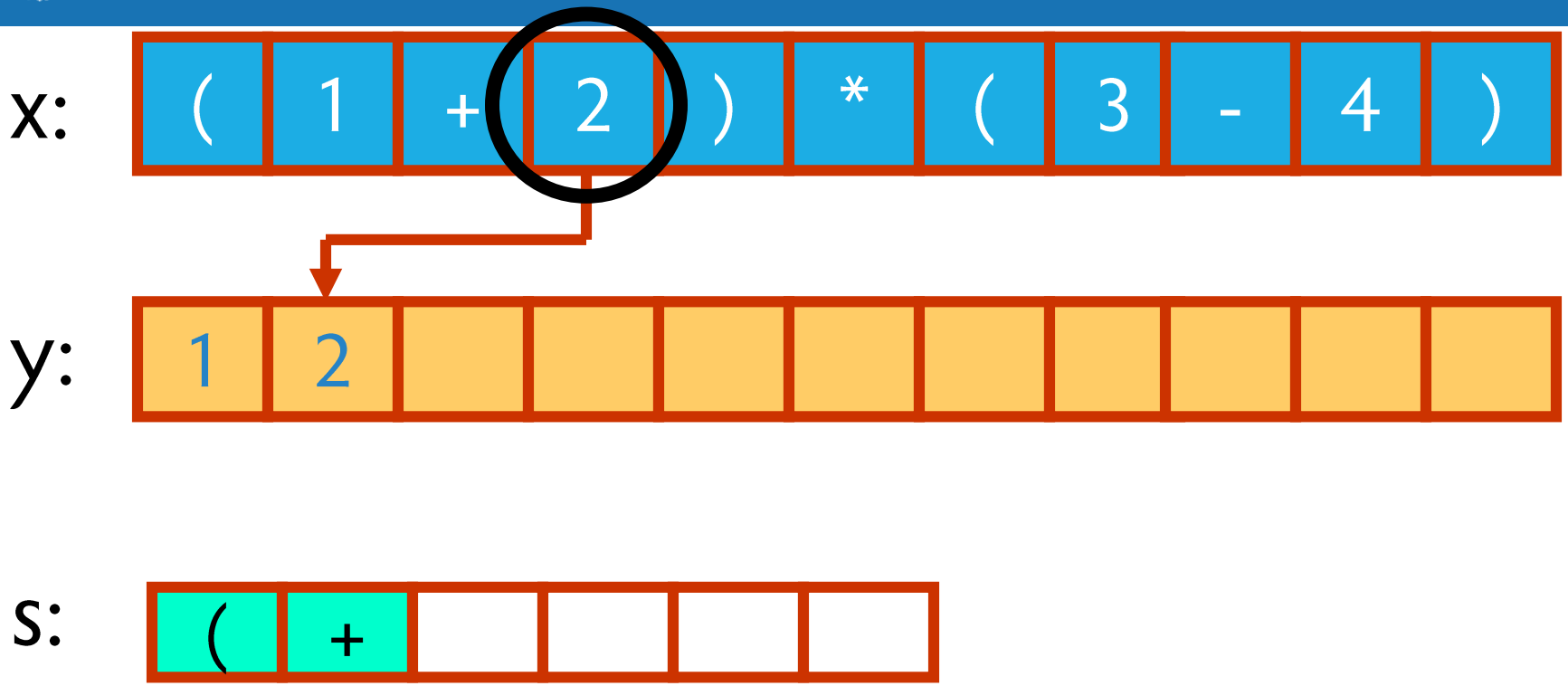


A következő szimbólum operátor:
1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,

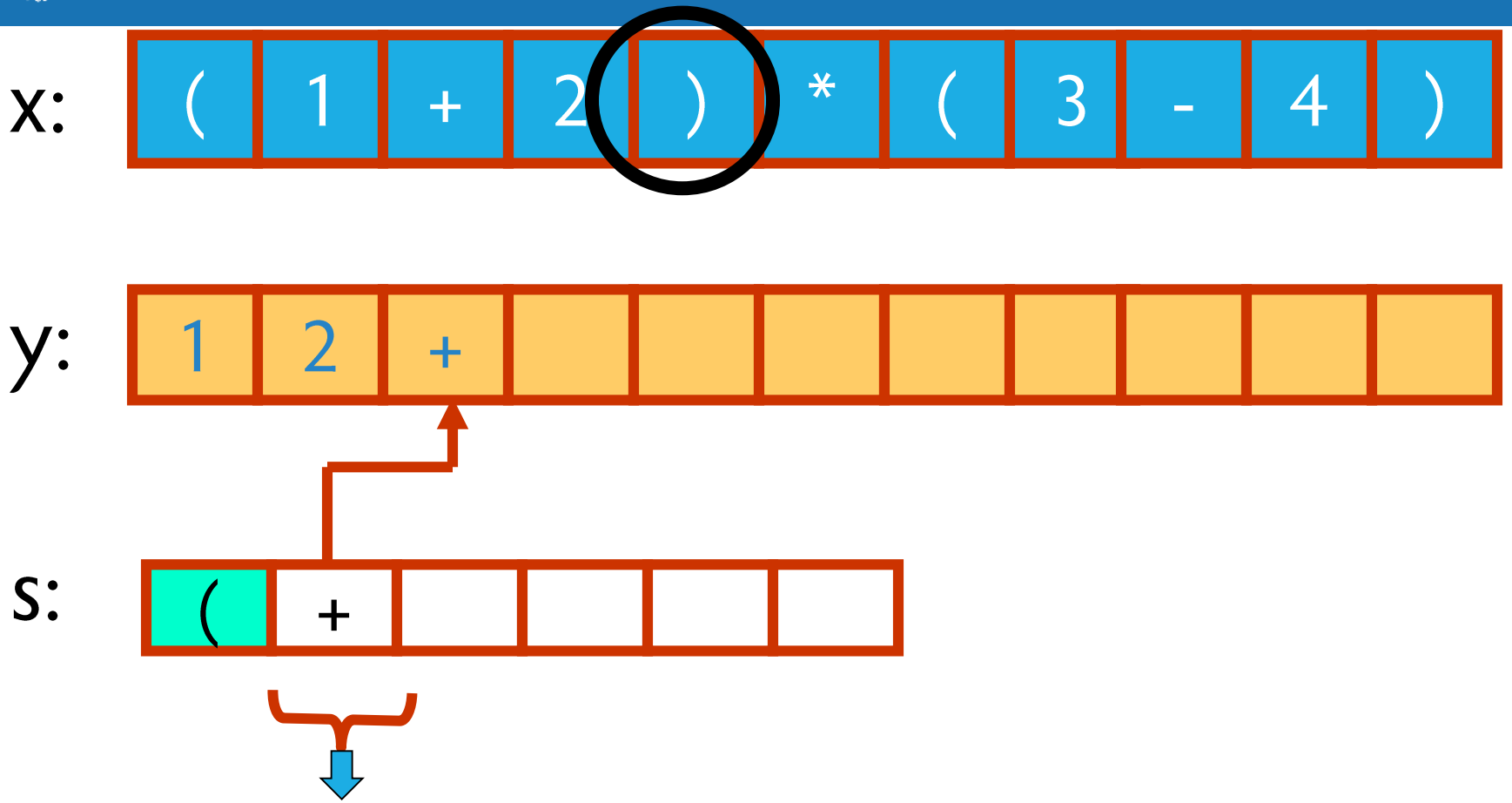


A következő szimbólum operátor:

1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,
2. Ezt az operátort betesszük a verembe.

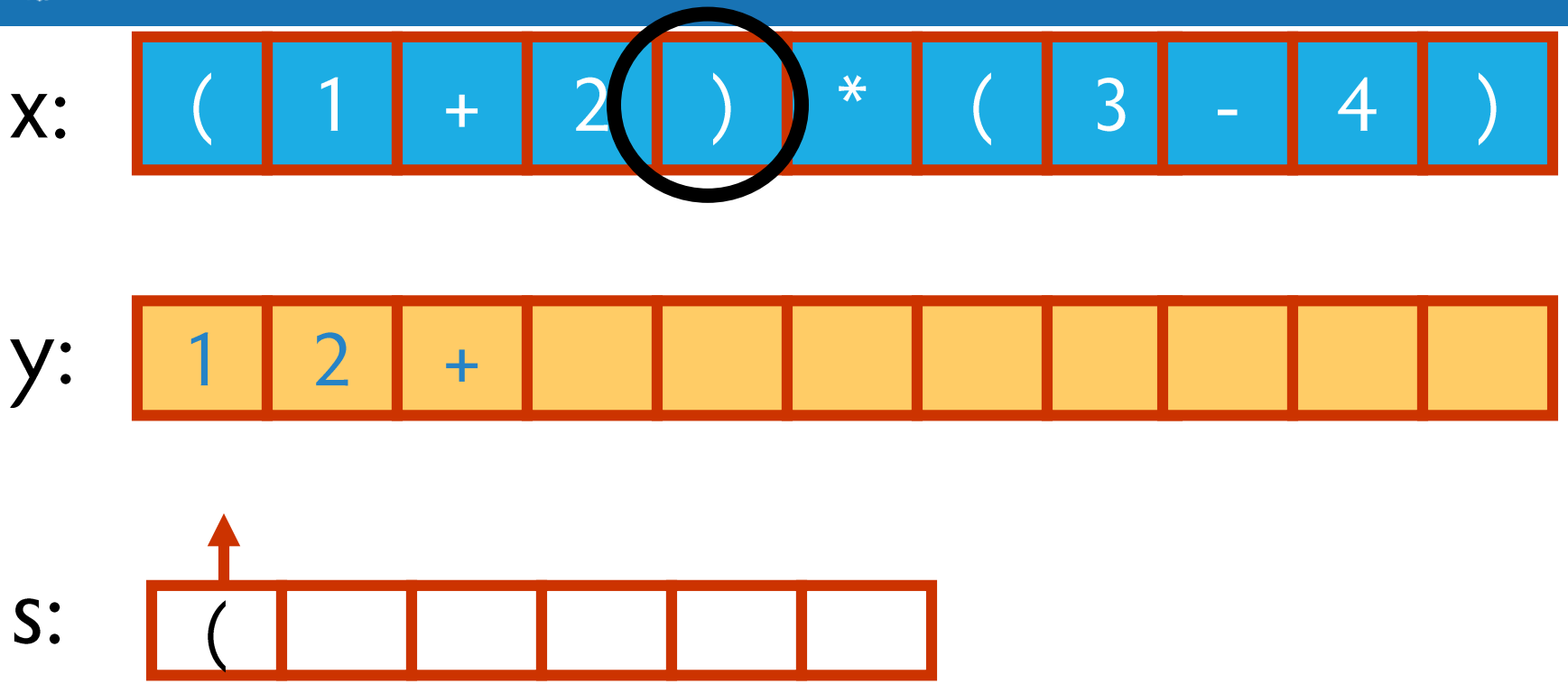


A következő szimbólum operandus: írjuk ki.



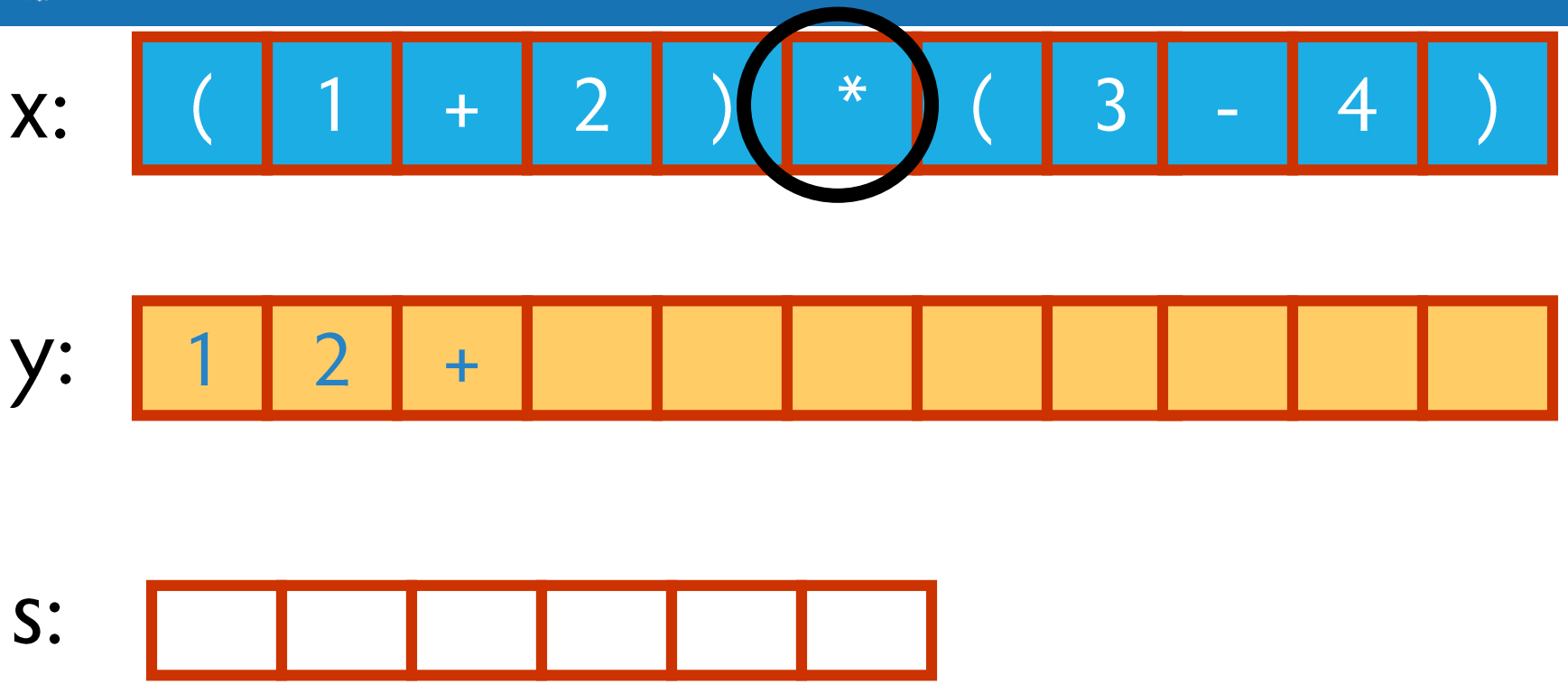
A következő szimbólum csukózárójel:

1. Írjuk ki a verem tetején lévő elemeket egészen a nyitózárójelig.

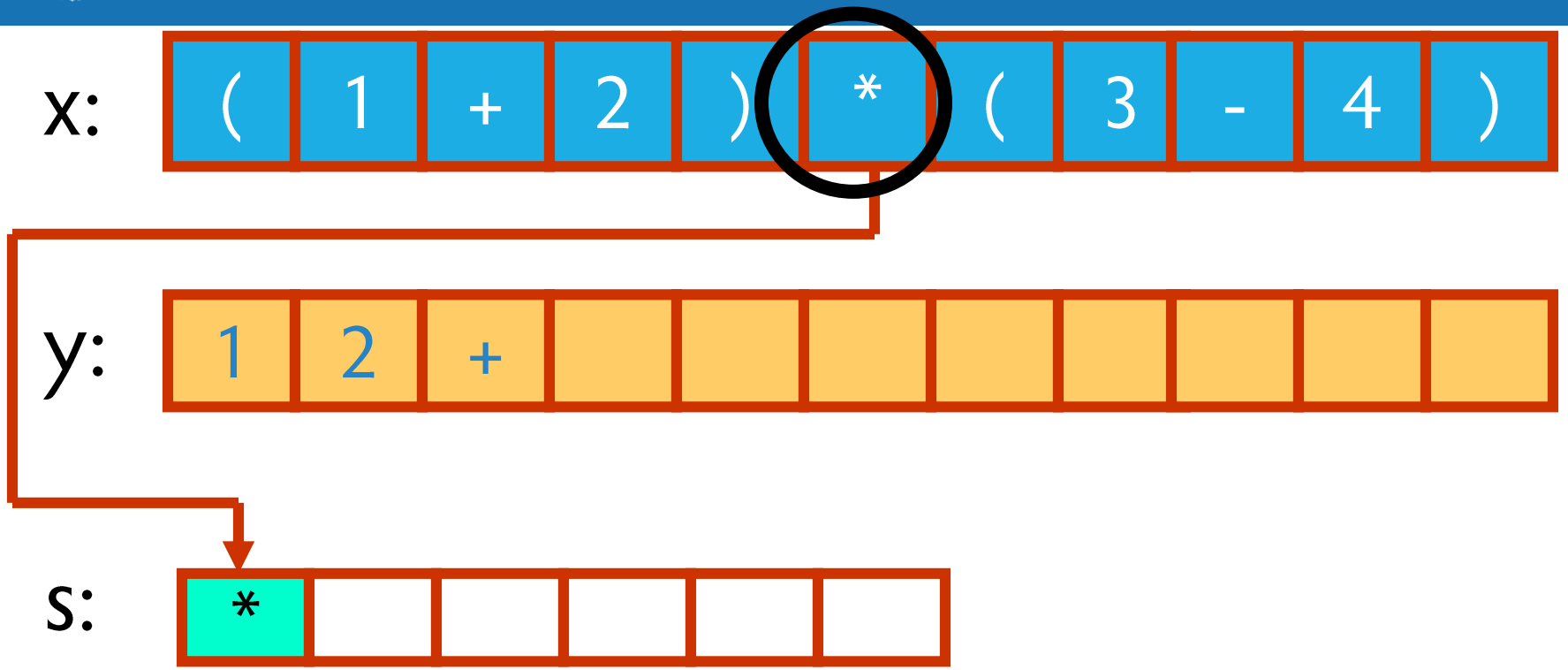


A következő szimbólum csukózárójel:

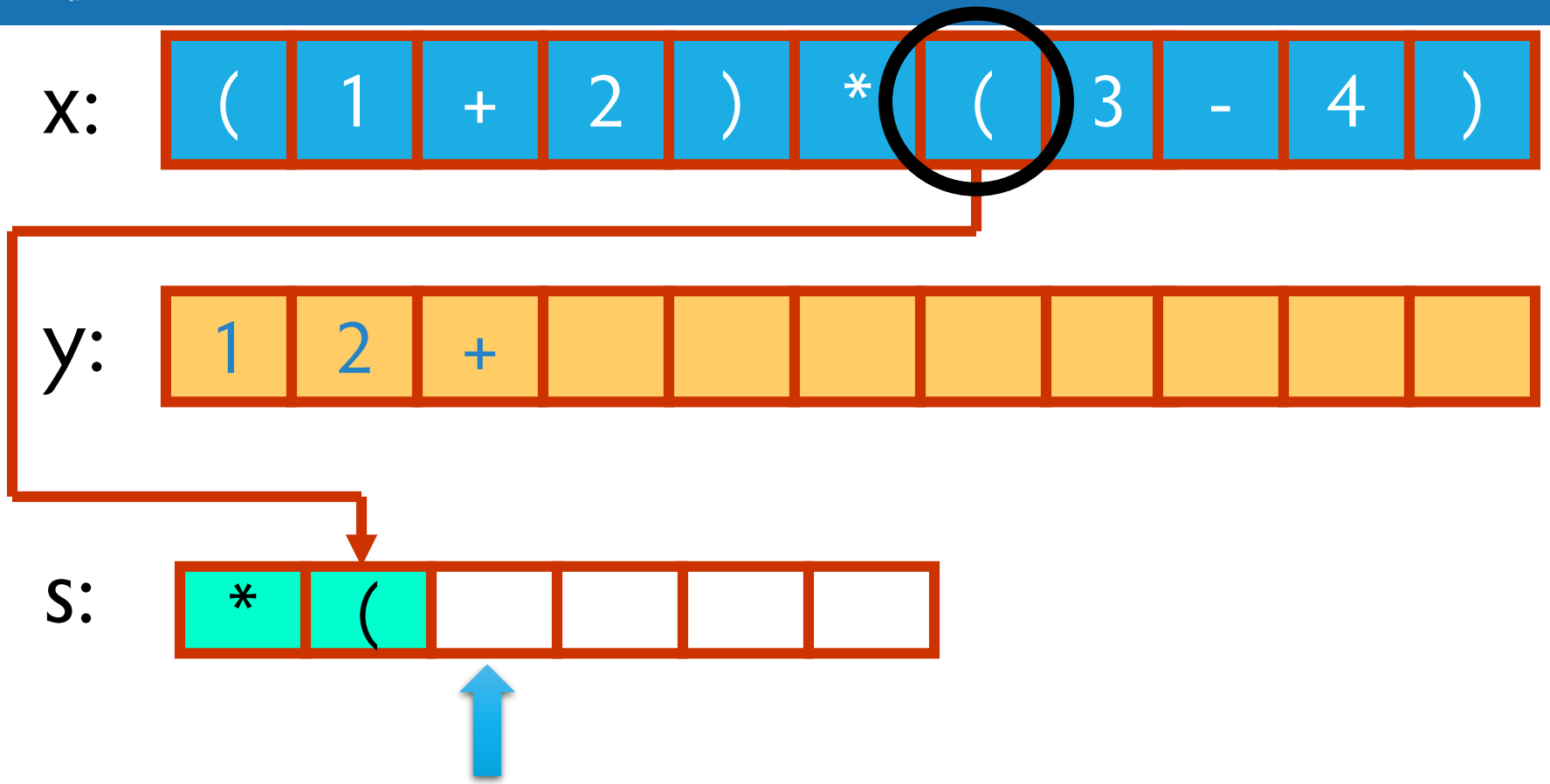
1. Írjuk ki a verem tetején lévő elemeket egészen a nyitózárójelig.
2. Vegyük ki a verem tetejéről a nyitózárójelet.



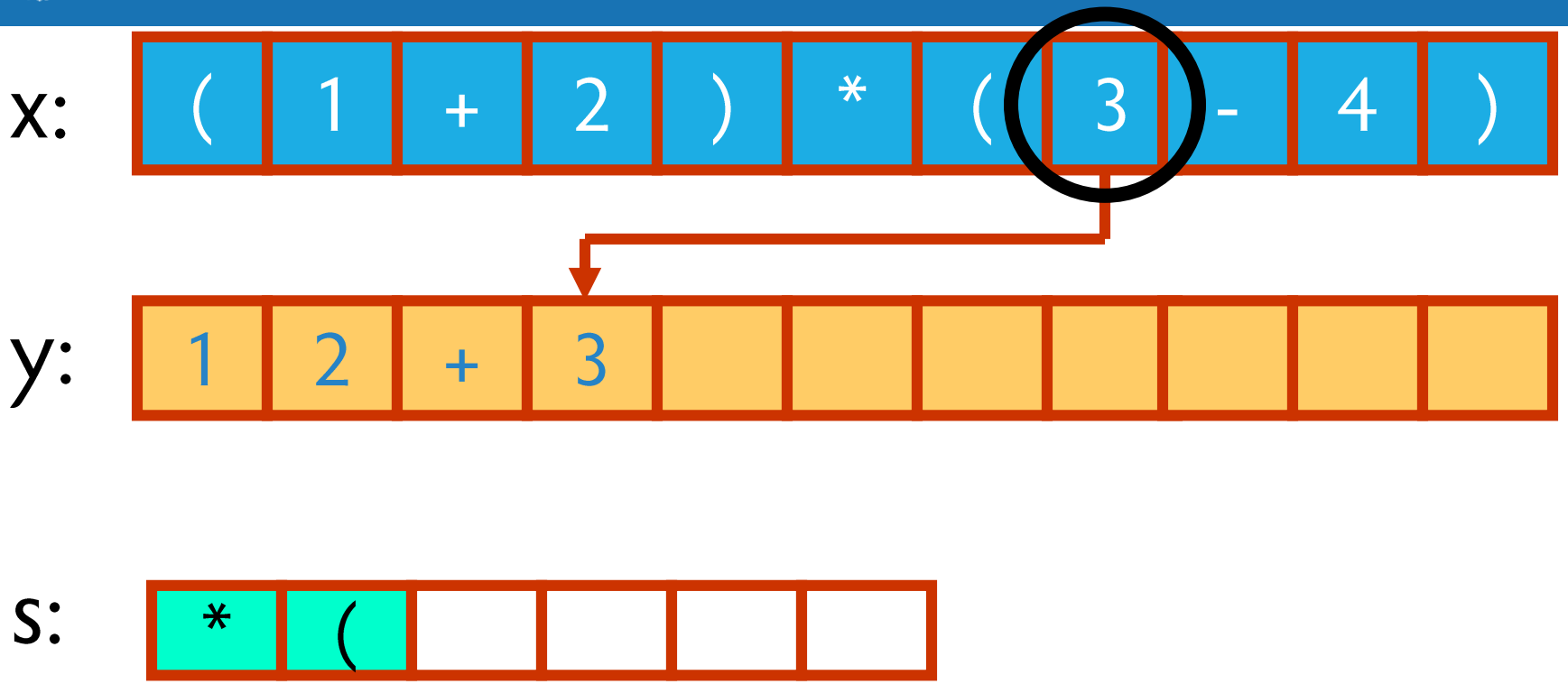
A következő szimbólum operátor:
1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,



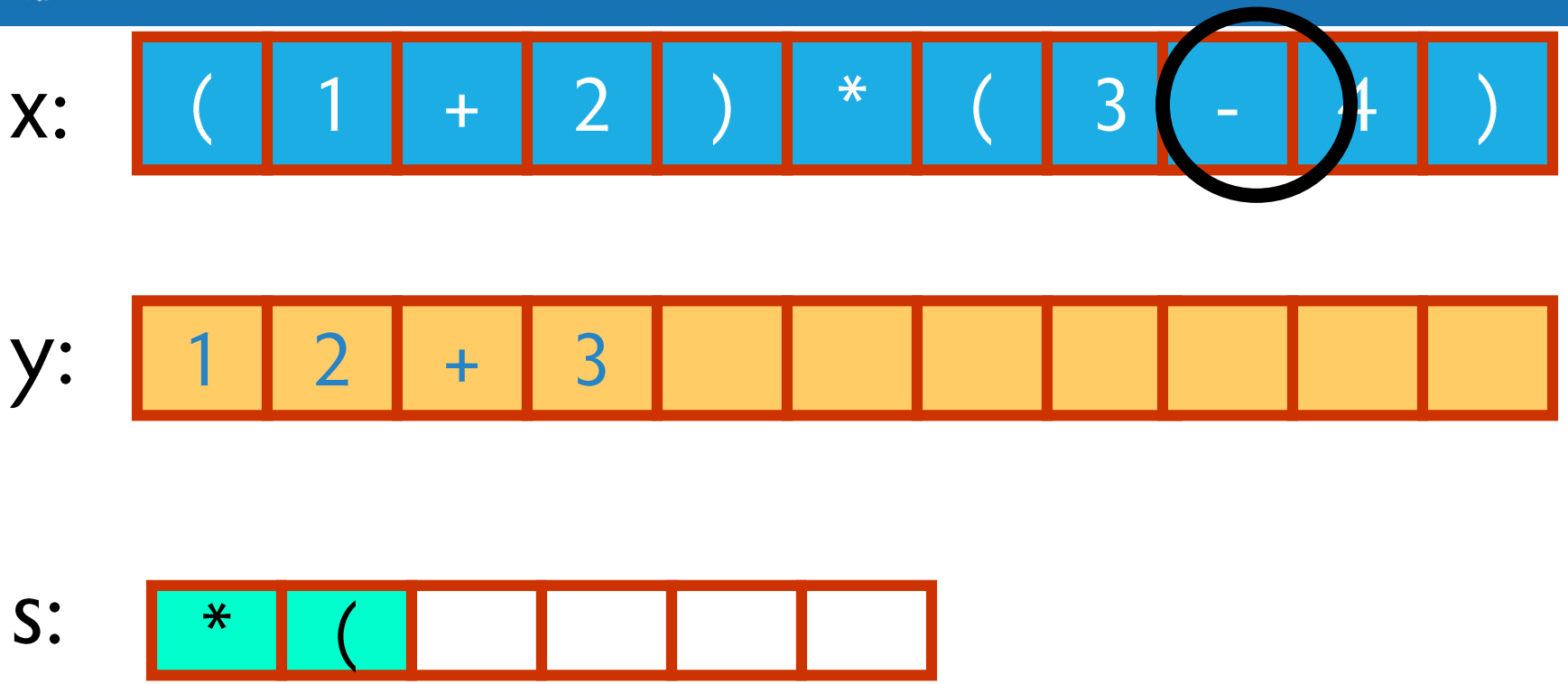
A következő szimbólum operátor:
1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,
2. Ezt az operátort betesszük a verembe.



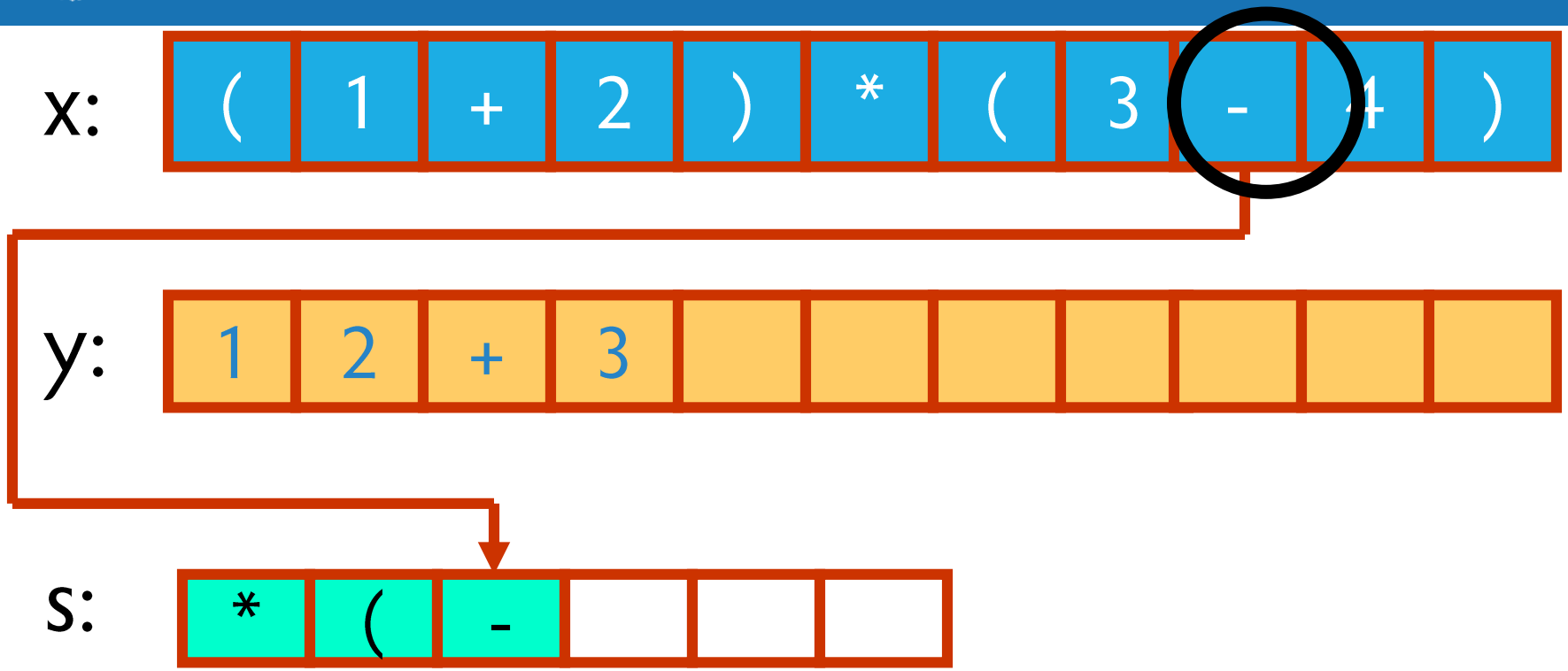
A következő szimbólum nyitózárójel: tegyük a verembe.



A következő szimbólum operandus: írjuk ki.

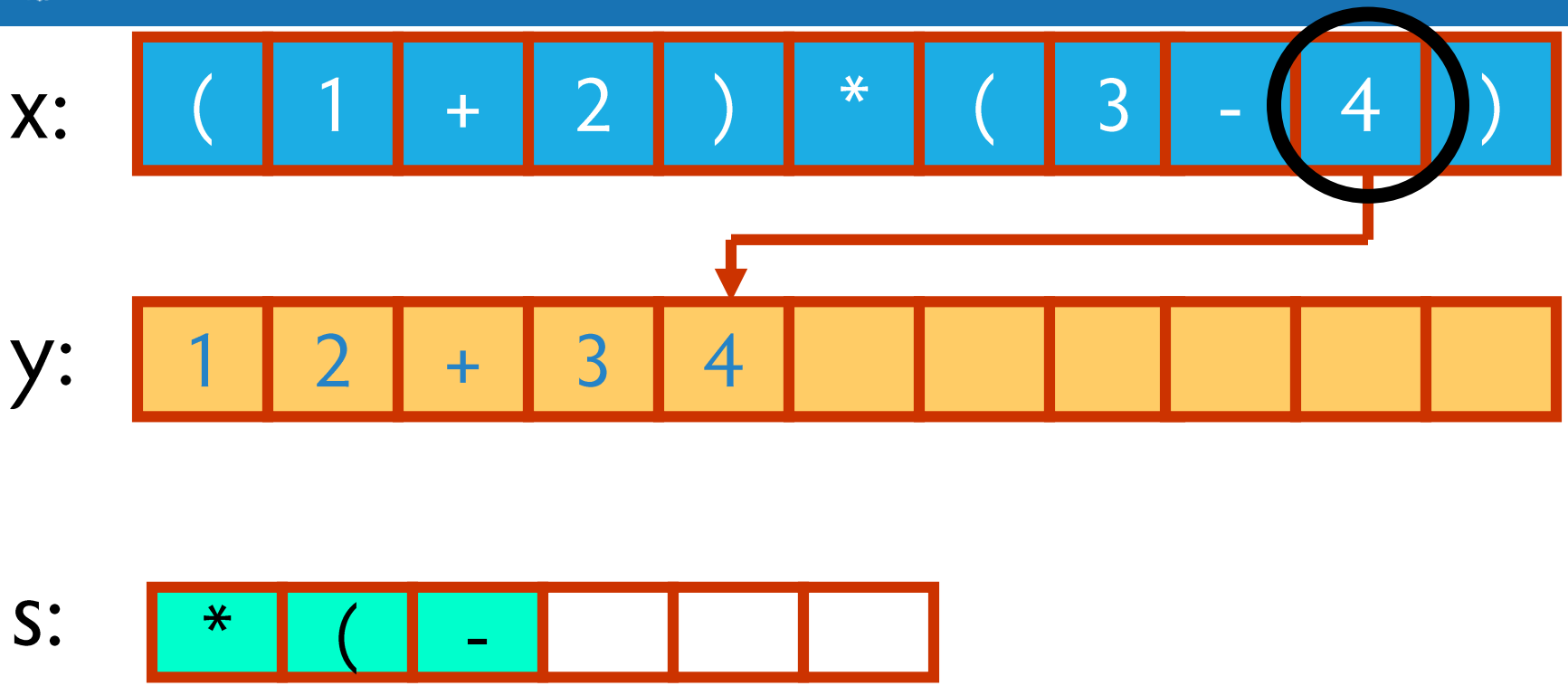


A következő szimbólum operátor:
1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,

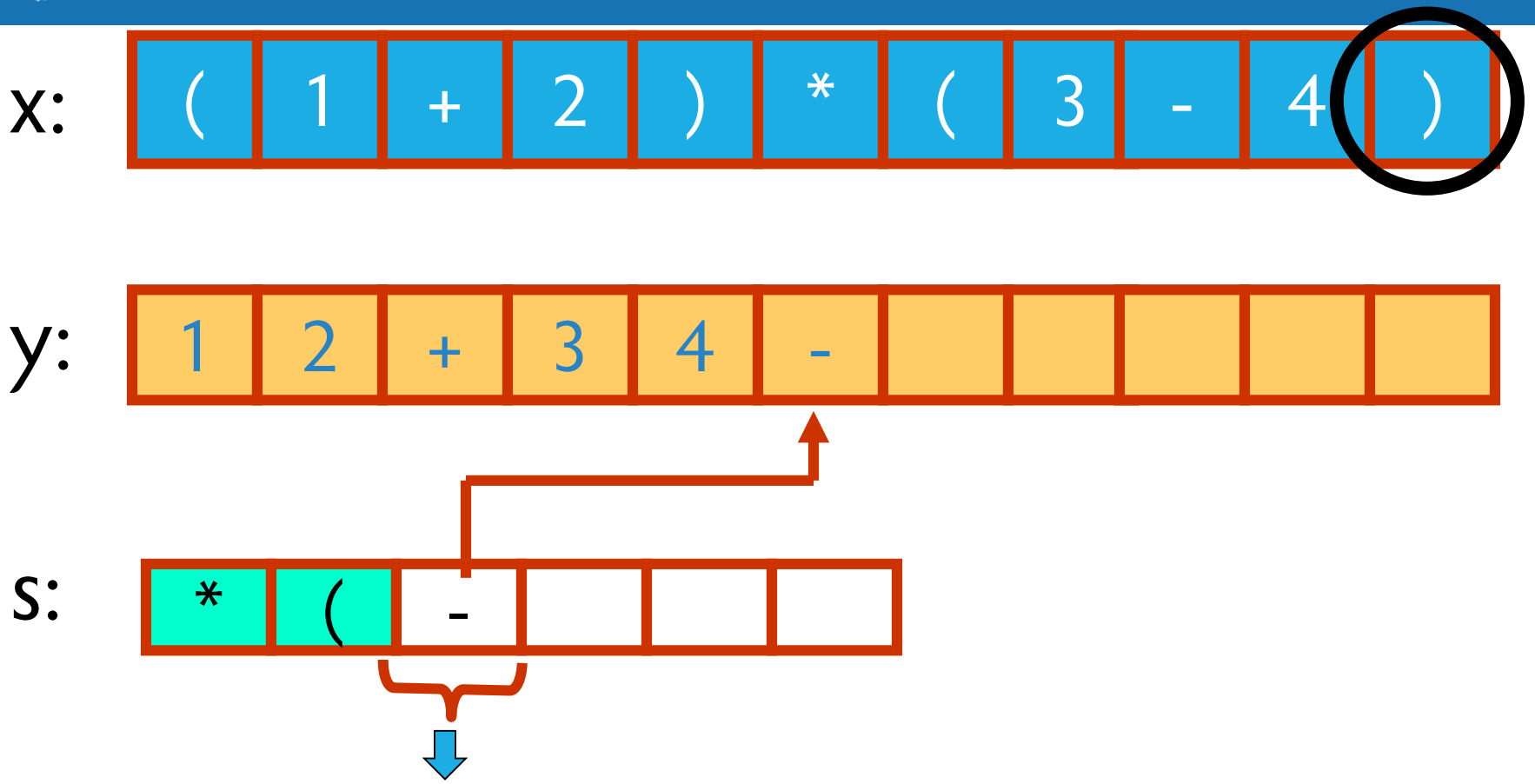


A következő szimbólum operátor:

1. legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat,
2. Ezt az operátort betesszük a verembe.

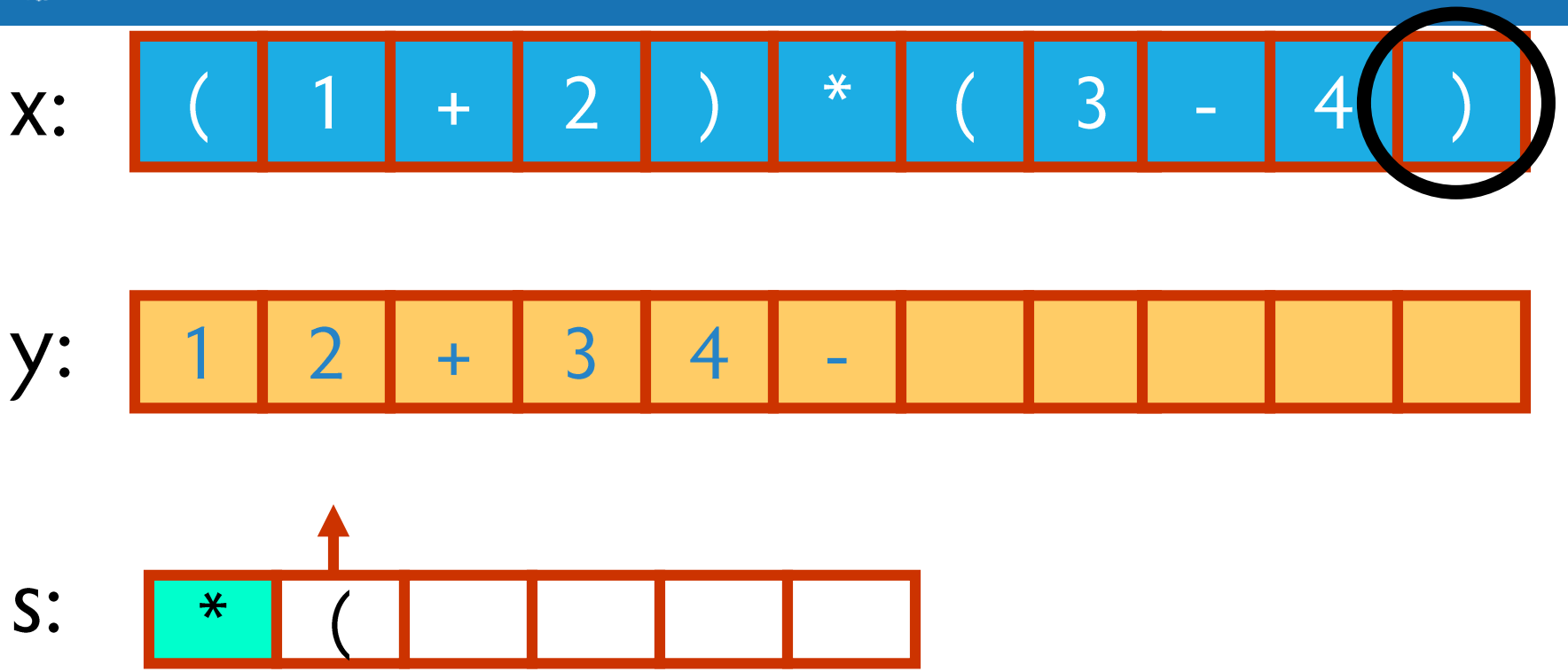


A következő szimbólum operandus: írjuk ki.



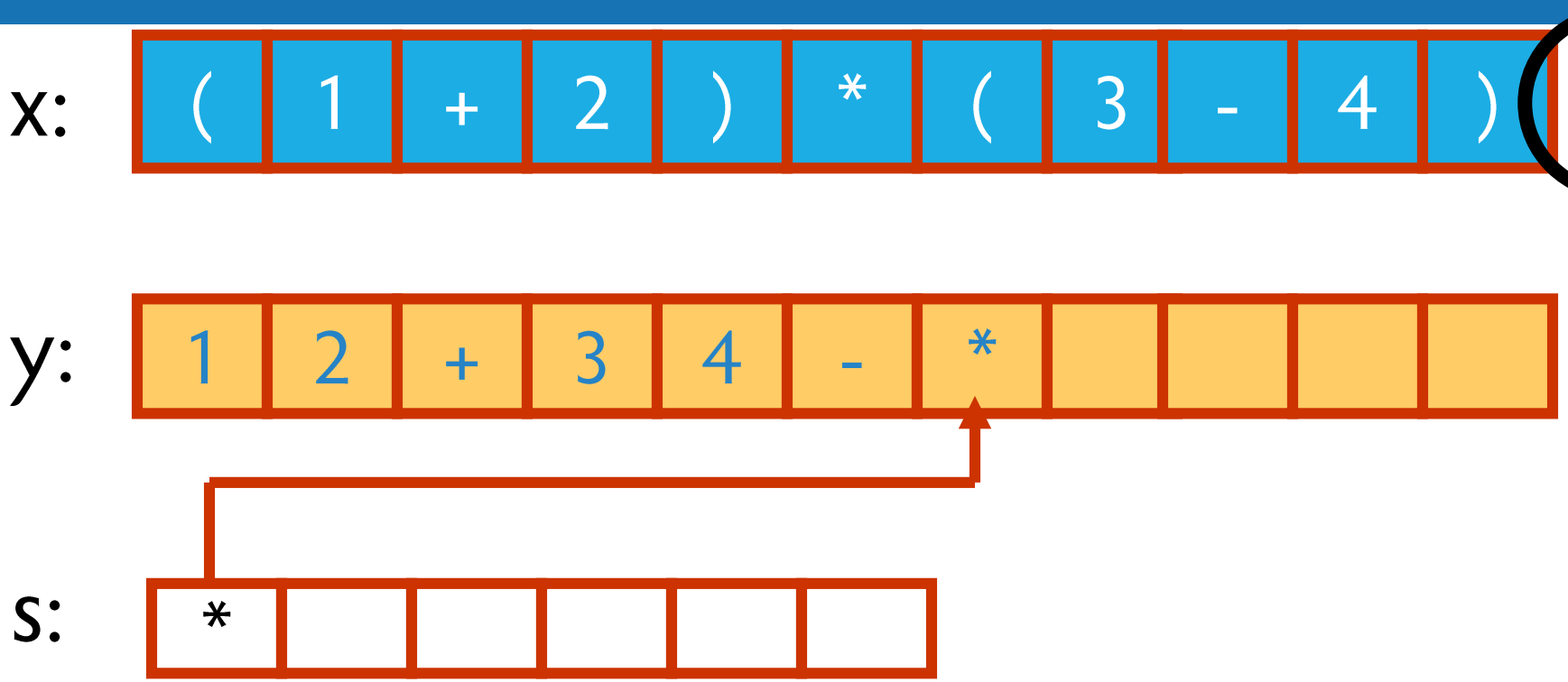
A következő szimbólum csukózárójel:

1. Írjuk ki a verem tetején lévő elemeket egészen a nyitózárójelig.



A következő szimbólum csukózárójel:

1. Irjuk ki a verem tetején lévő elemeket egészen a nyitózárójelig.
2. Vegyük ki a verem tetejéről a nyitózárójelet.

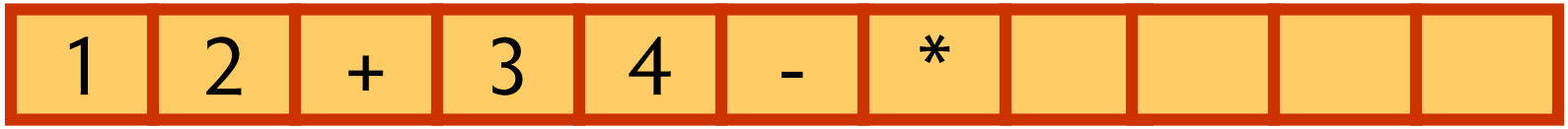


Elértük a kifejezés végét:
Írjuk ki a veremben lévő összes elemet.

Kiértékelés

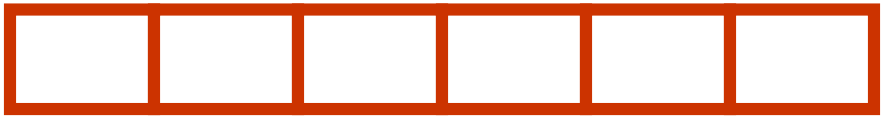
- 1 2 + 3 4 − *
  - Kiértékelés után
  - -3

y:

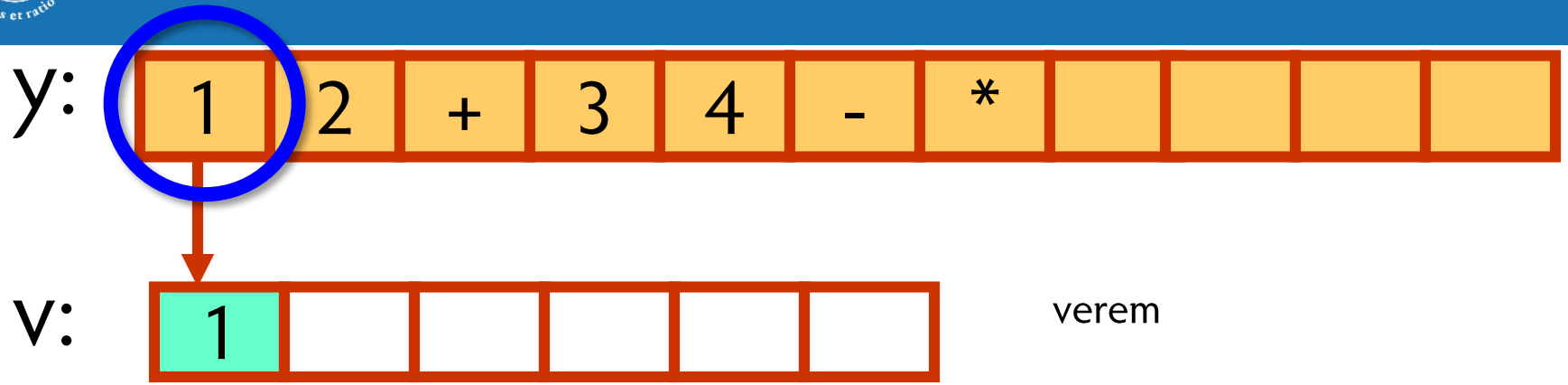


sorozat

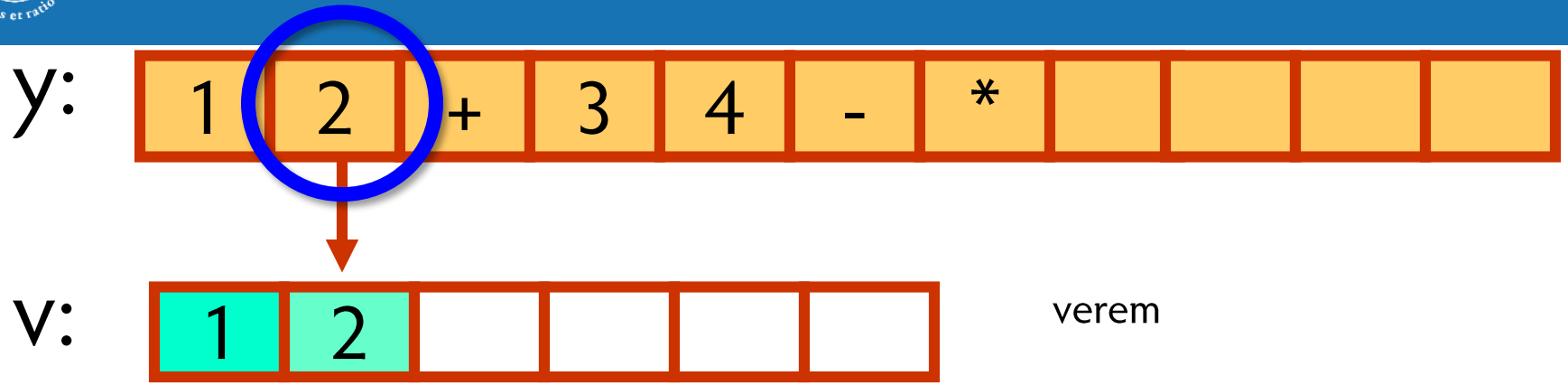
v:



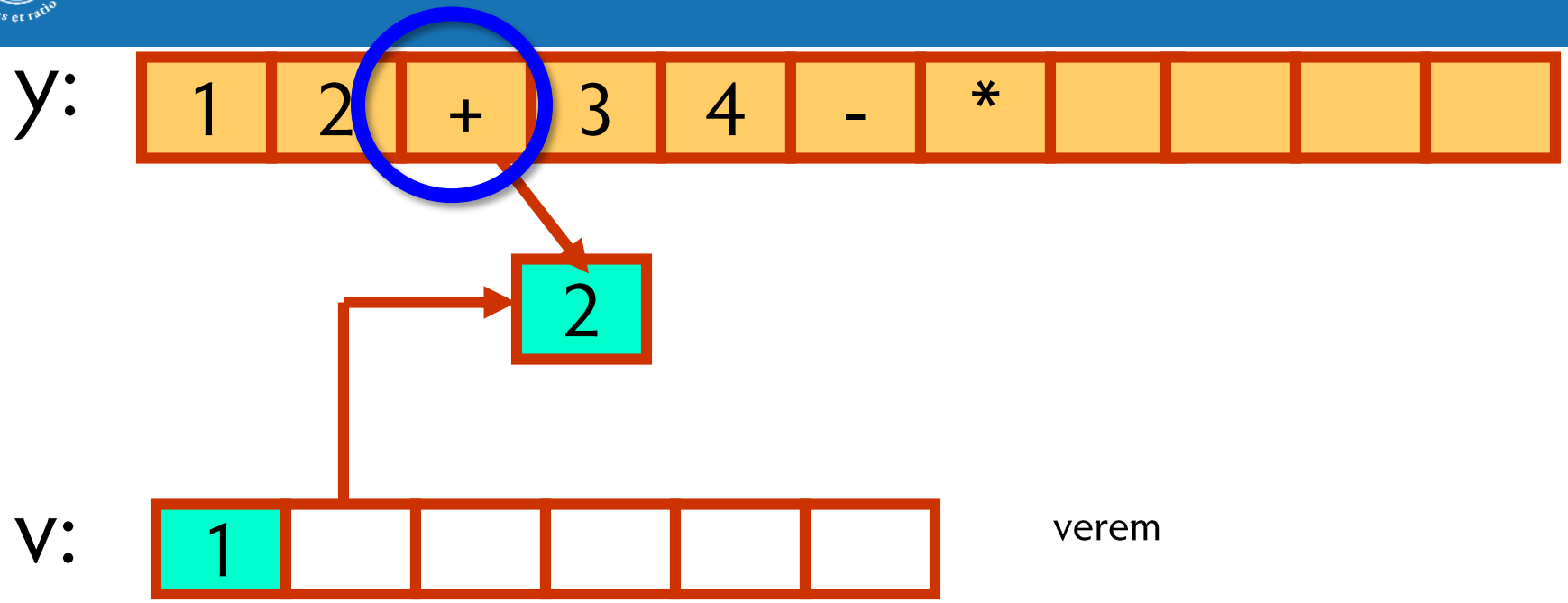
verem



A következő szimbólum operandus:
Tegyük a verembe.

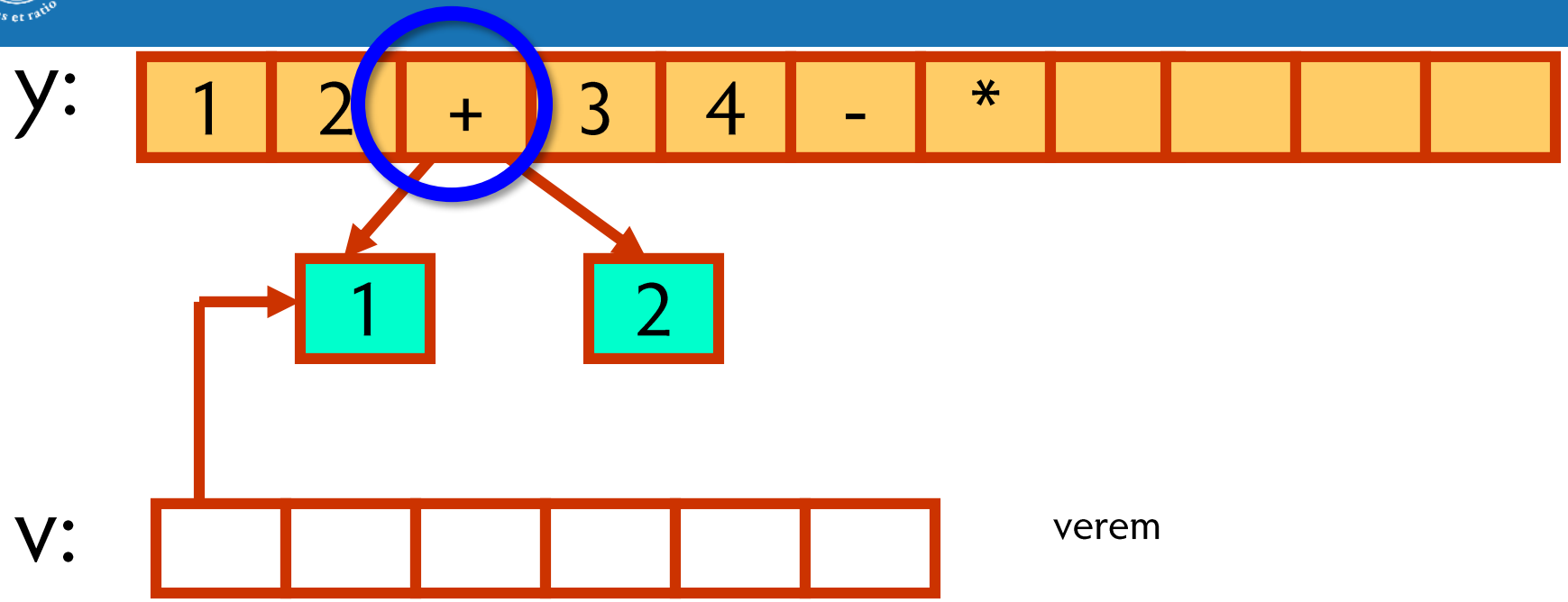


A következő szimbólum operandus:
Tegyük a verembe.

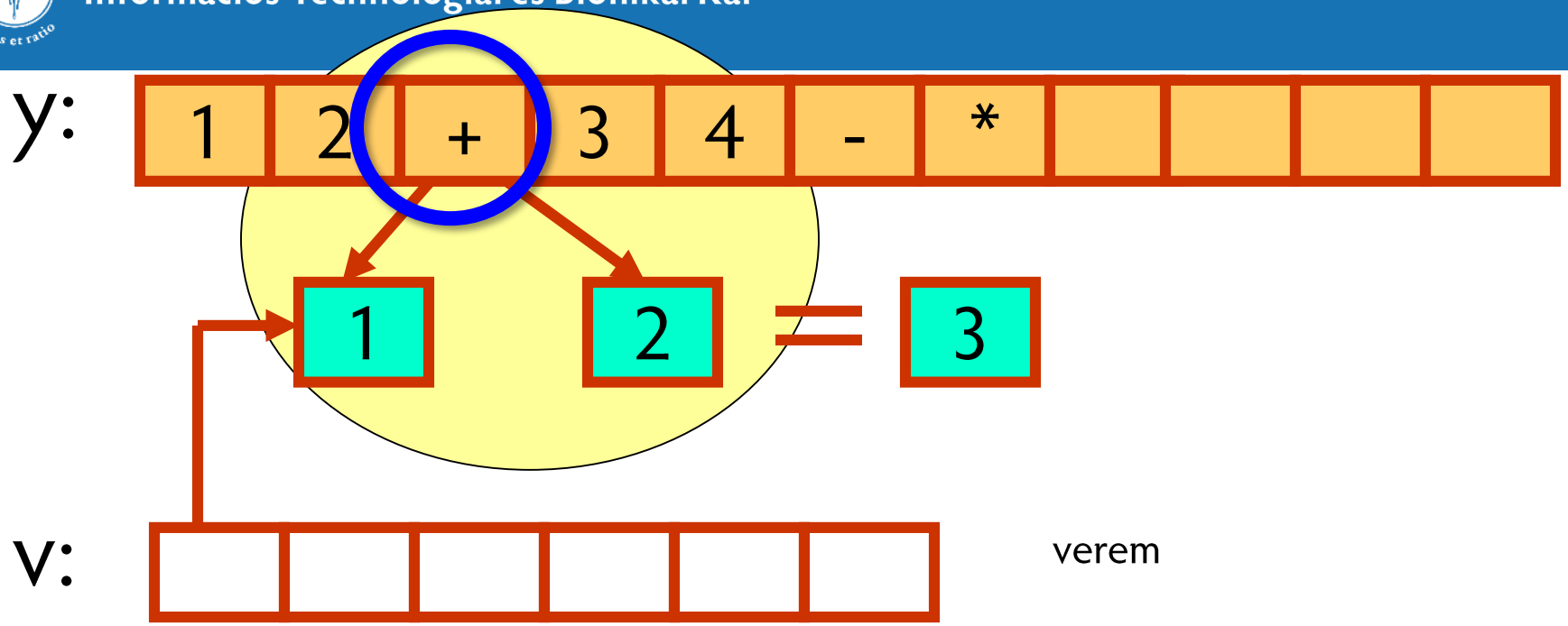


A következő szimbólum operátor:

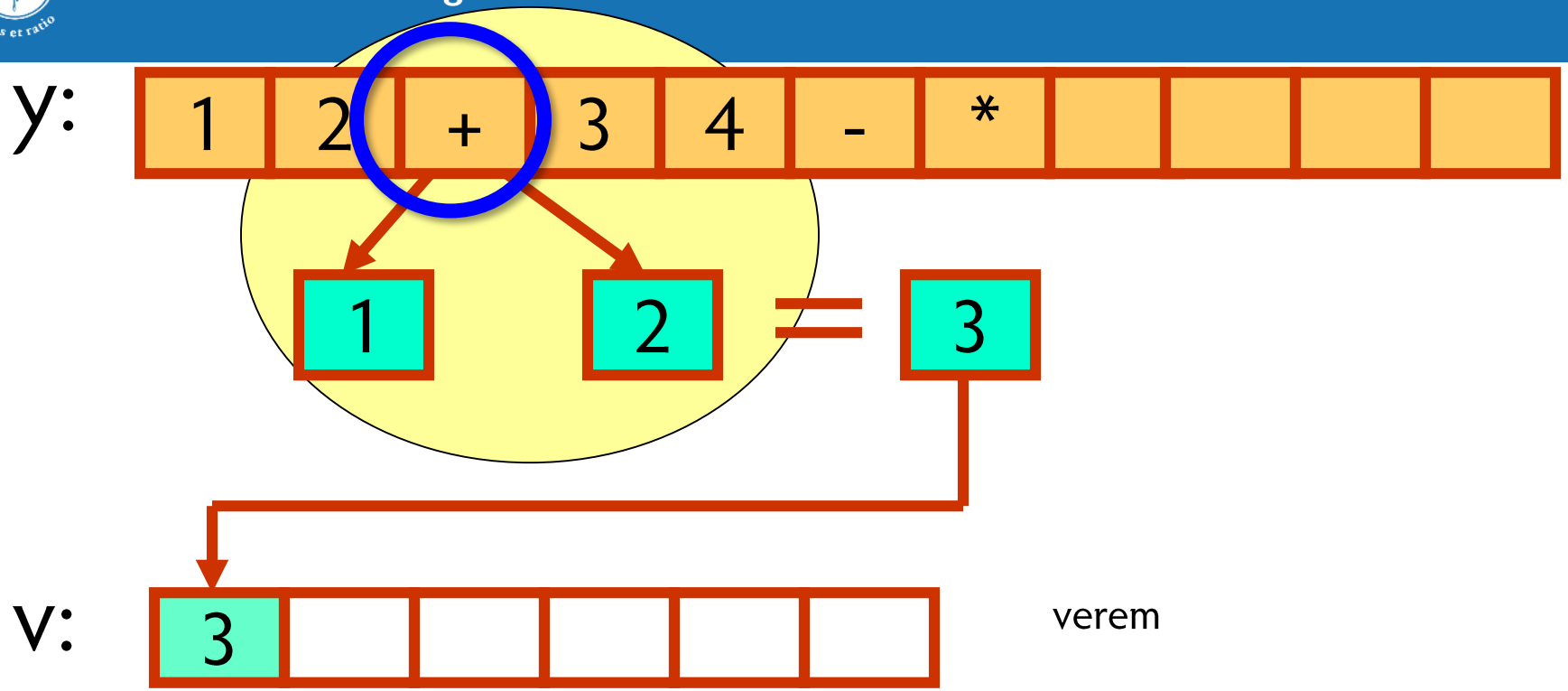
1. Vegyük ki a veremből a második operandust.



A következő szimbólum operátor:
1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.

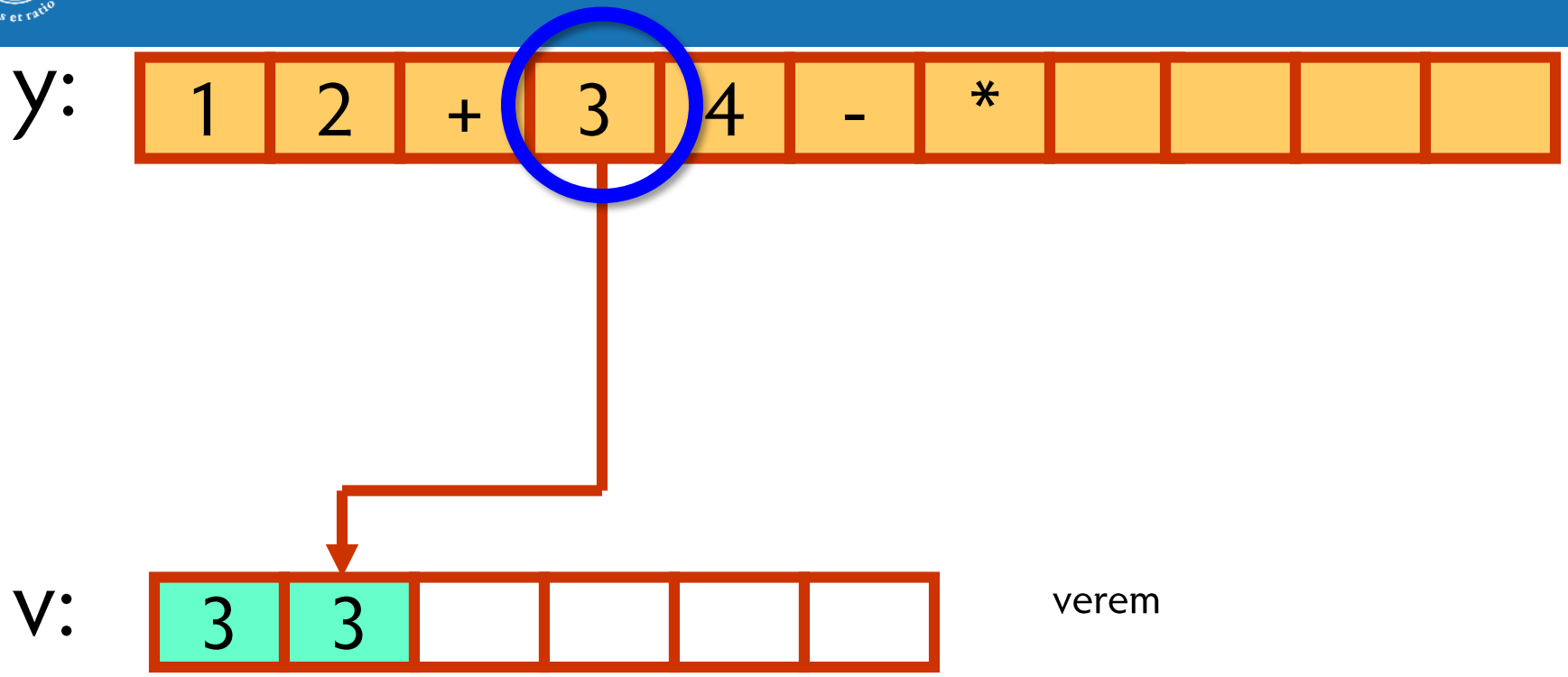


A következő szimbólum operátor:
1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.

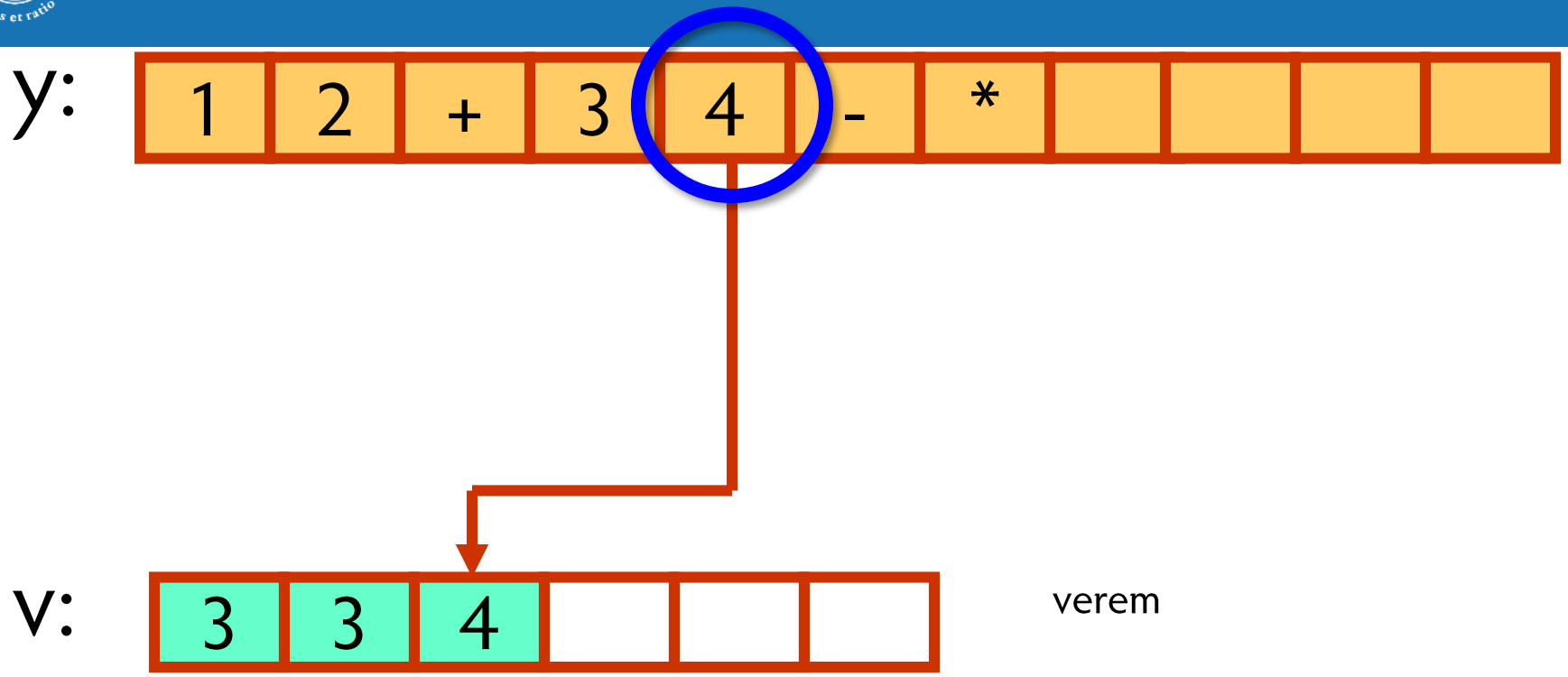


A következő szimbólum operátor:

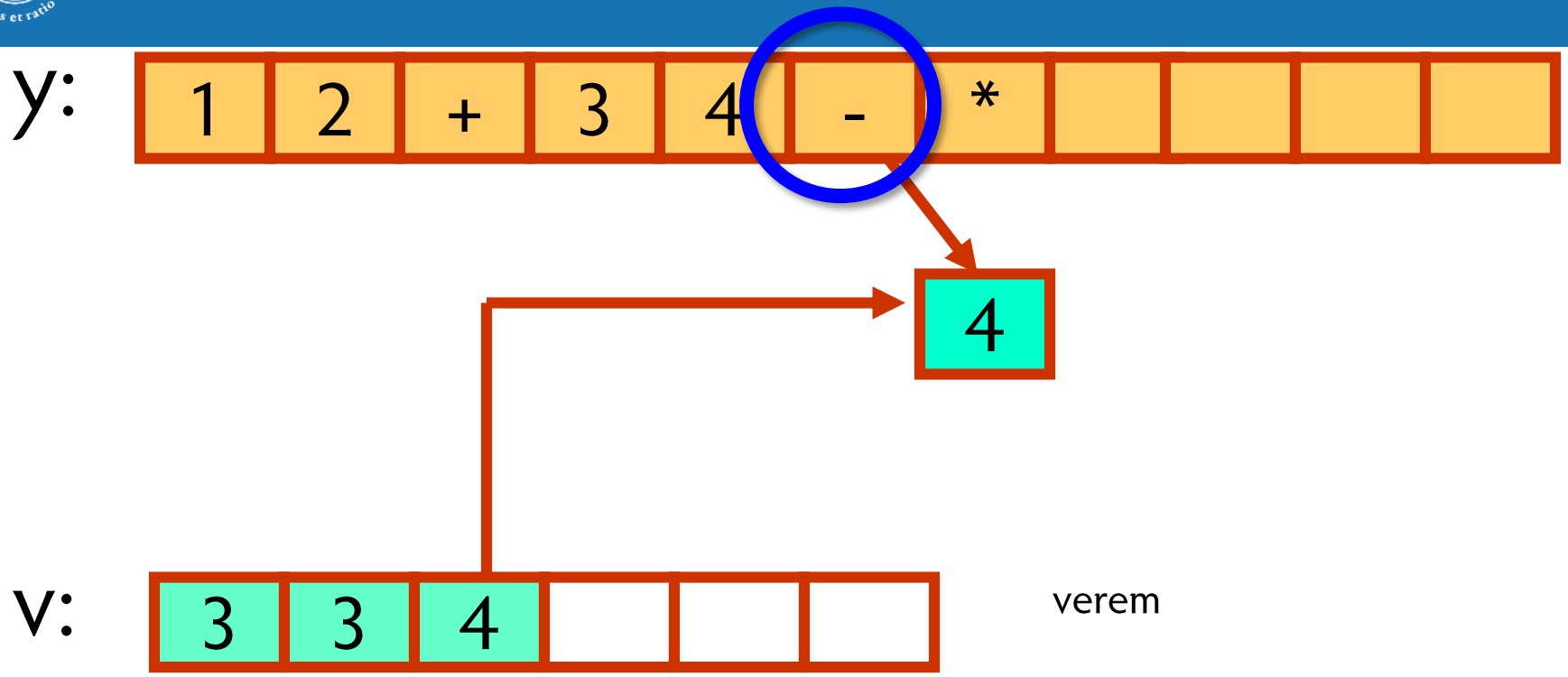
1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.
4. Az így kapott eredményt tegyük a verembe.



A következő szimbólum operandus:
Tegyük a verembe.

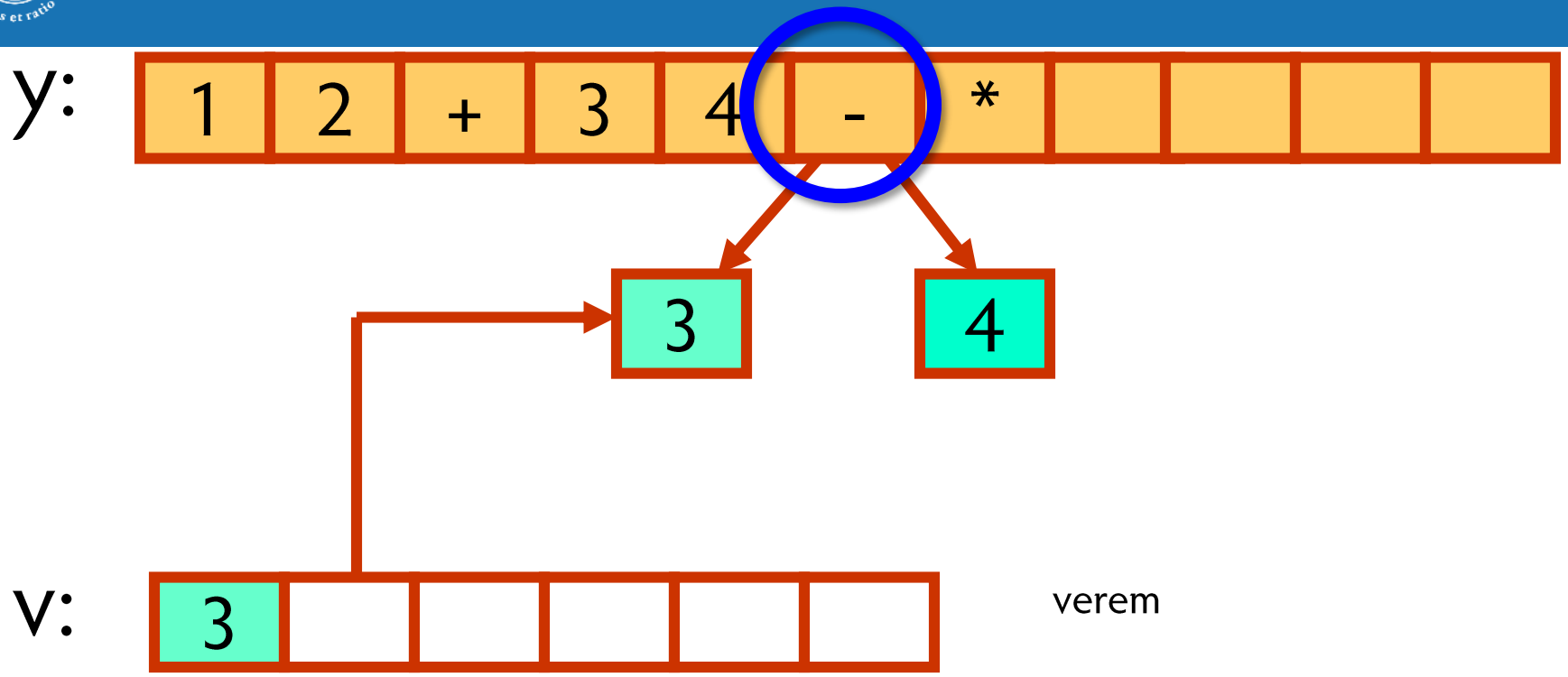


A következő szimbólum operandus:
Tegyük a verembe.

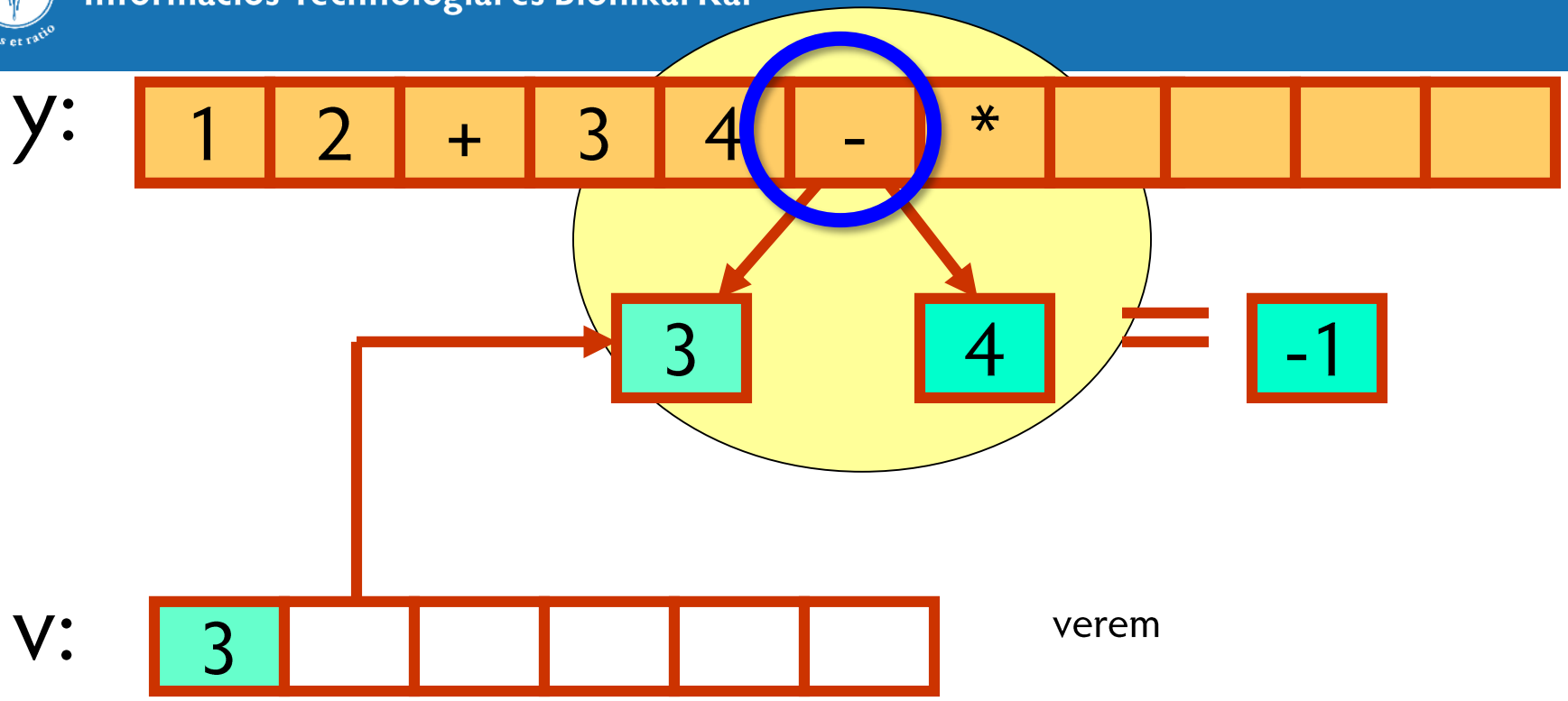


A következő szimbólum operátor:

1. Vegyük ki a veremből a második operandust.

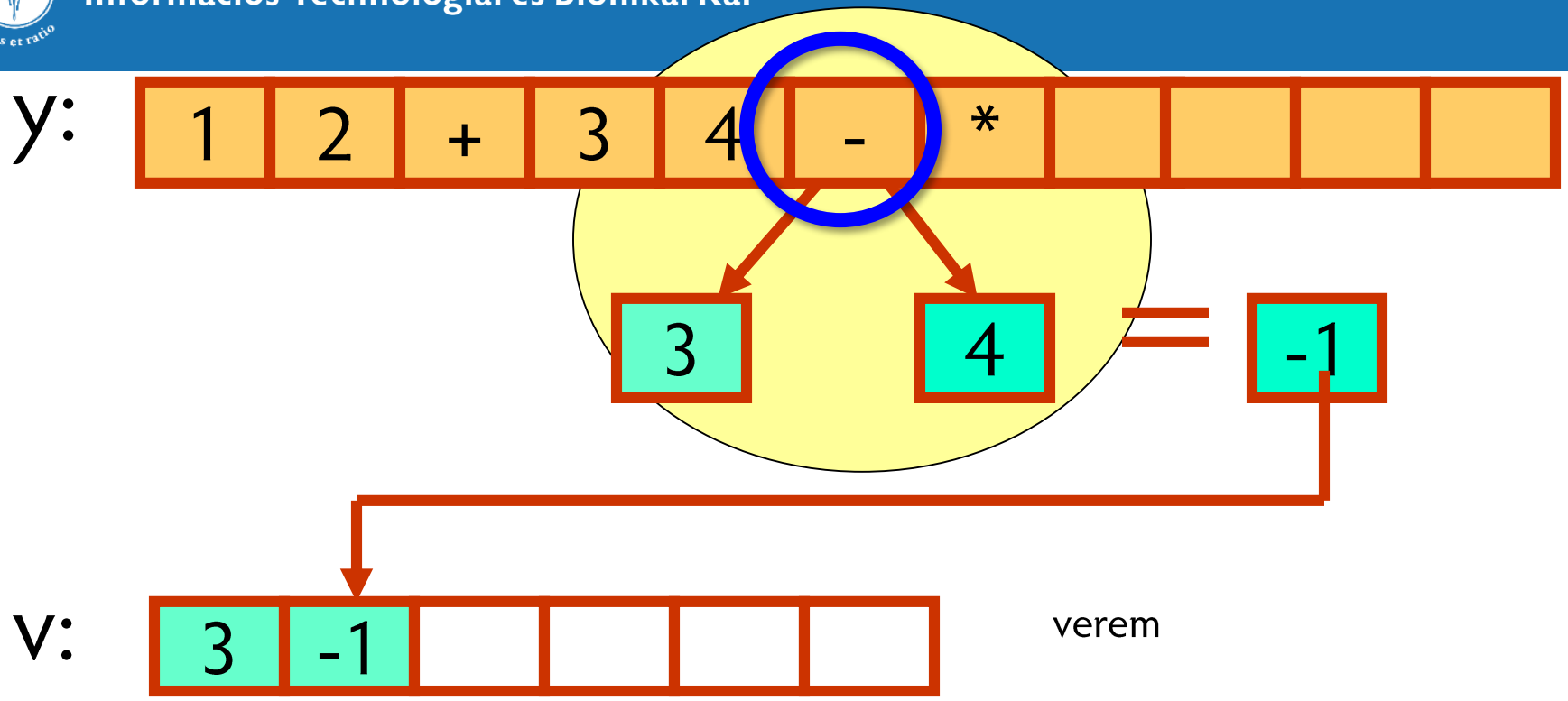


A következő szimbólum operátor:
1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.



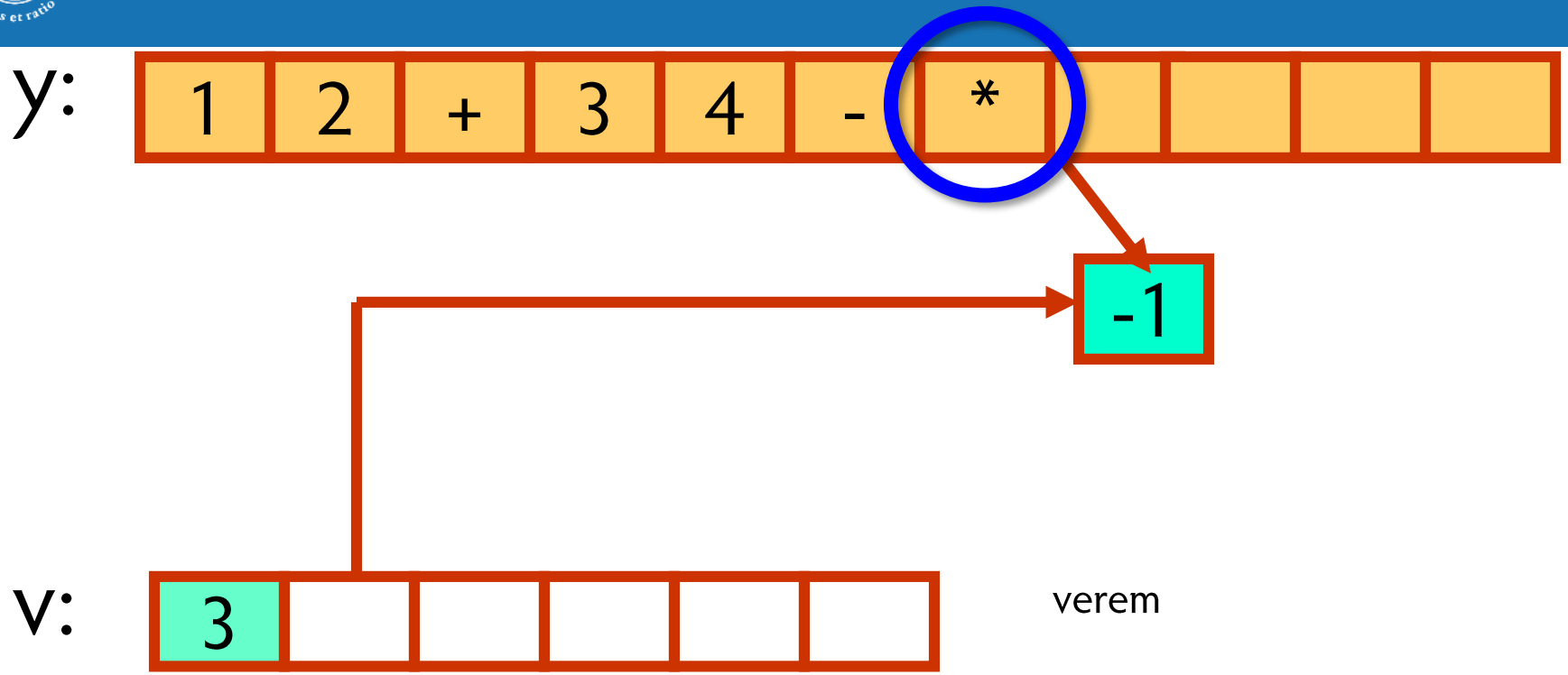
A következő szimbólum operátor:

1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.



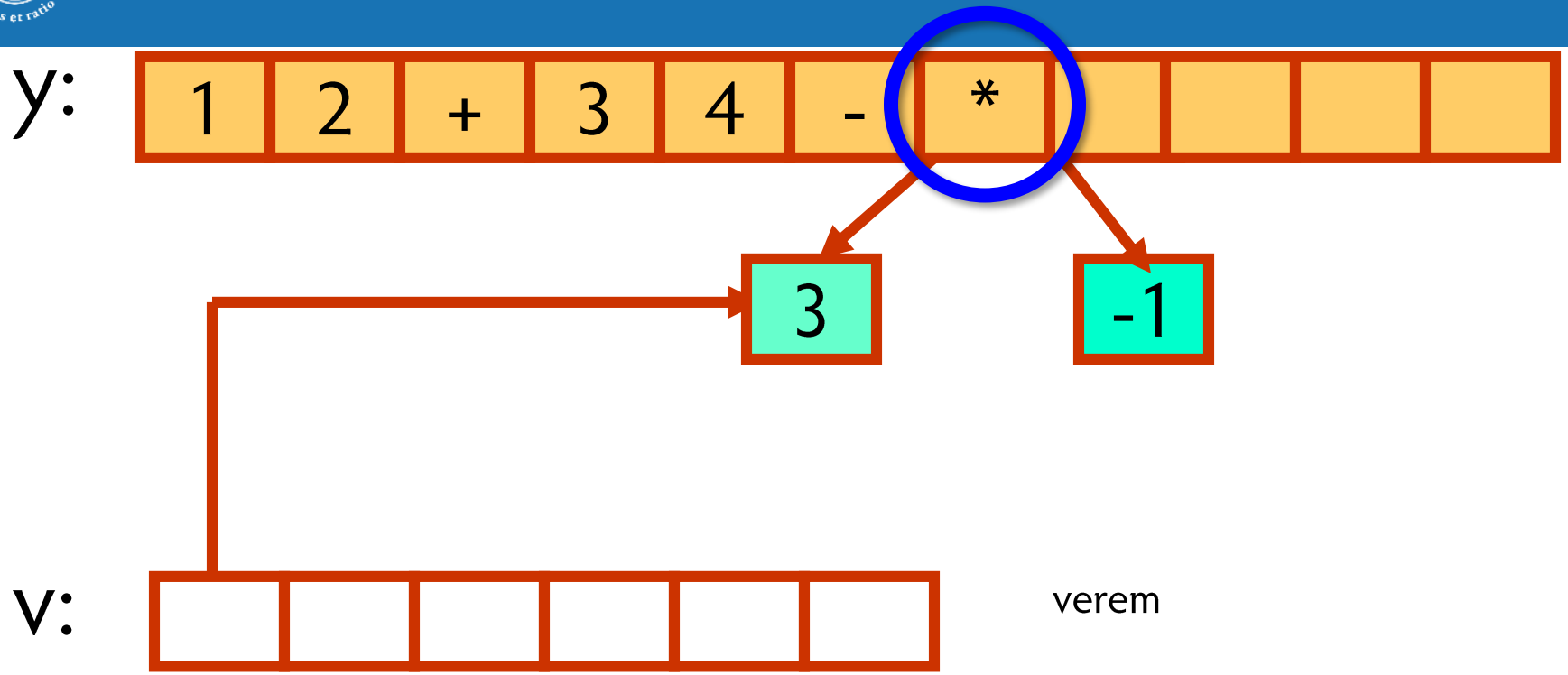
A következő szimbólum operátor:

1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.
4. Az így kapott eredményt tegyük a verembe.

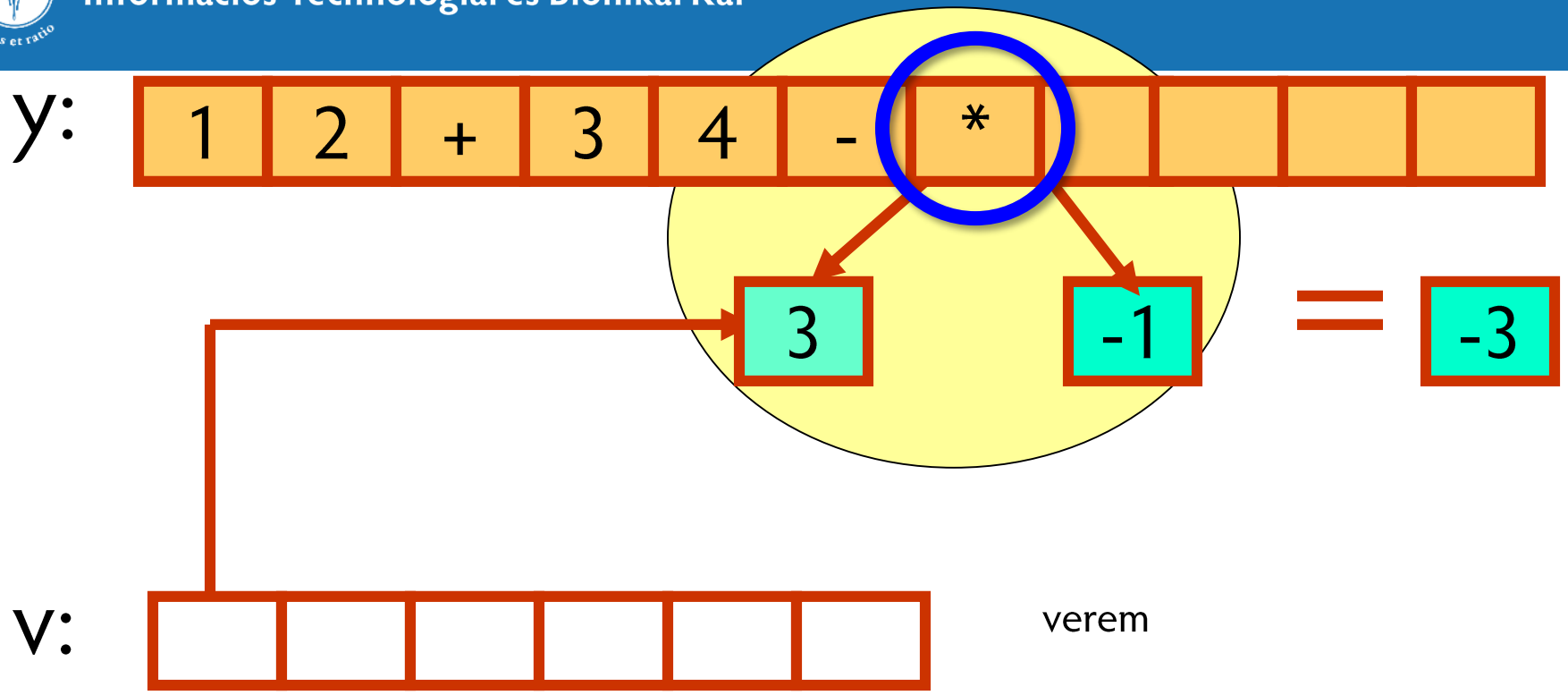


A következő szimbólum operátor:

1. Vegyük ki a veremből a második operandust.

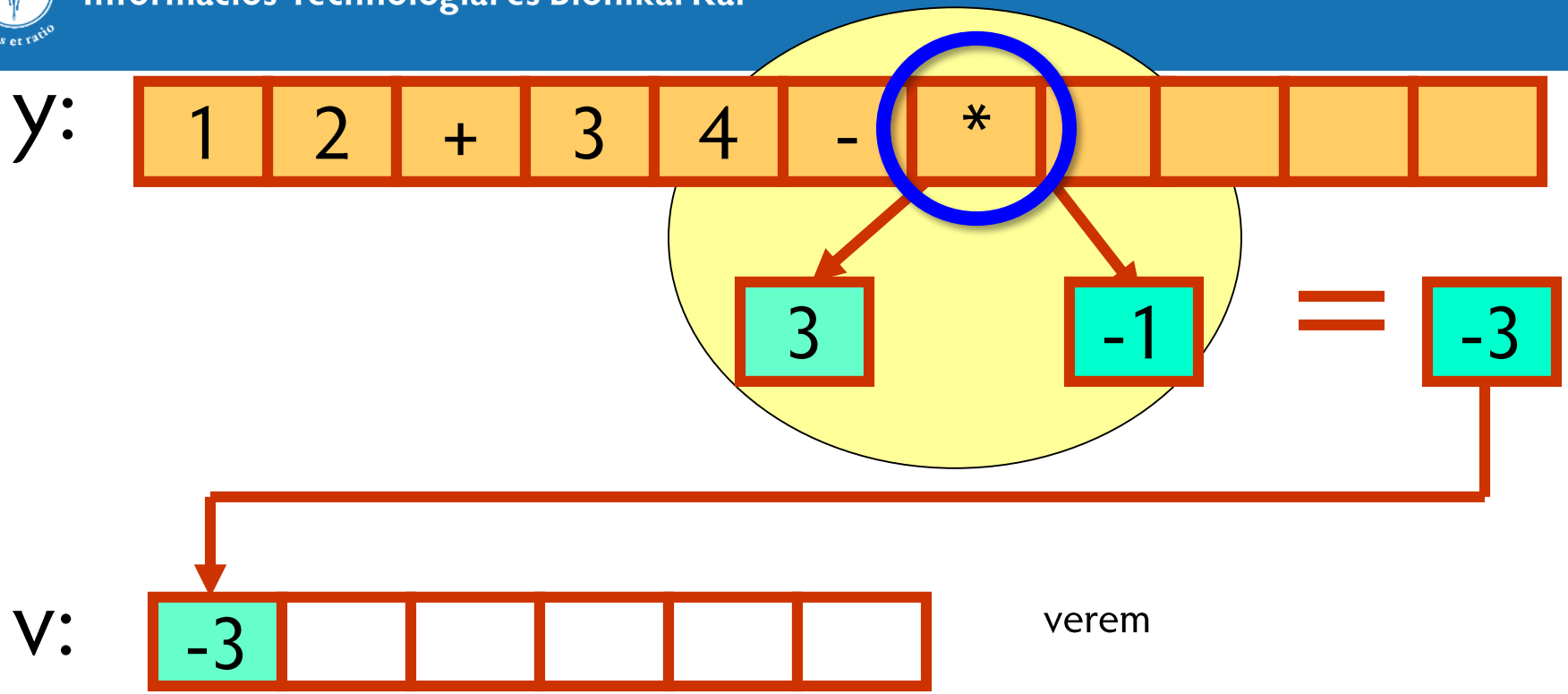


A következő szimbólum operátor:
1. Vegyük ki a veremből a második operandust
2.Vegyük ki a veremből az első operandust.

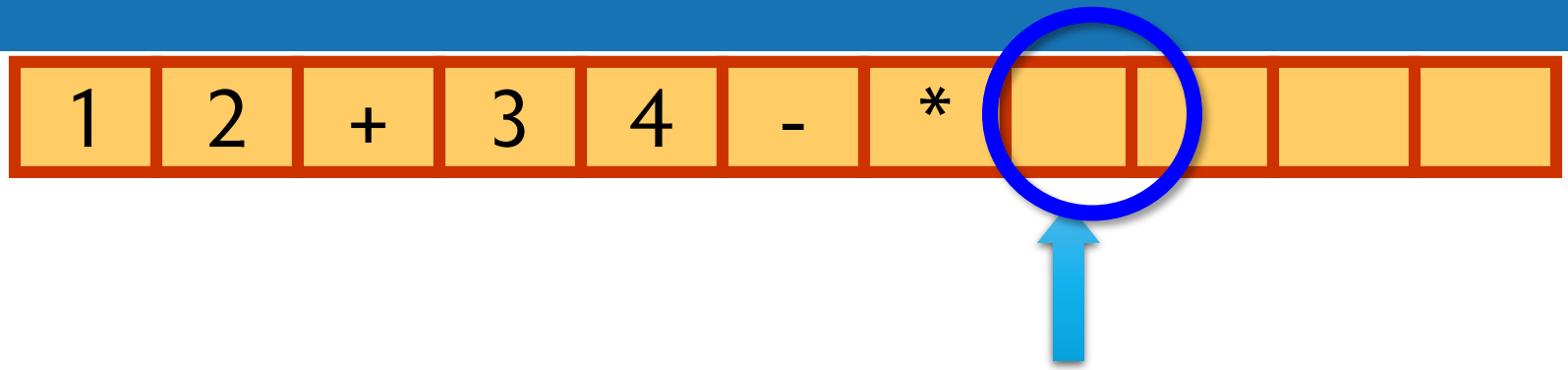


A következő szimbólum operátor:

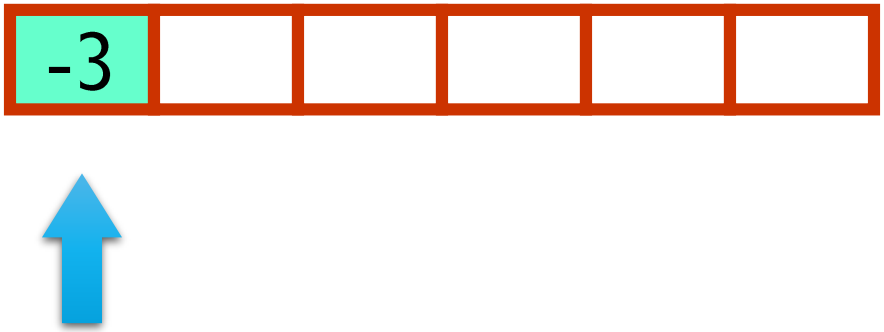
1. Vegyük ki a veremből a második operandust
2. Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.



A következő szimbólum operátor:
1. Vegyük ki a veremből a második operandust
2.Vegyük ki a veremből az első operandust.
3. Számítsuk ki az adott műveleti jellel a kifejezés értékét.
4. Az így kapott eredményt tegyük a verembe.



Elértük a kifejezés végét



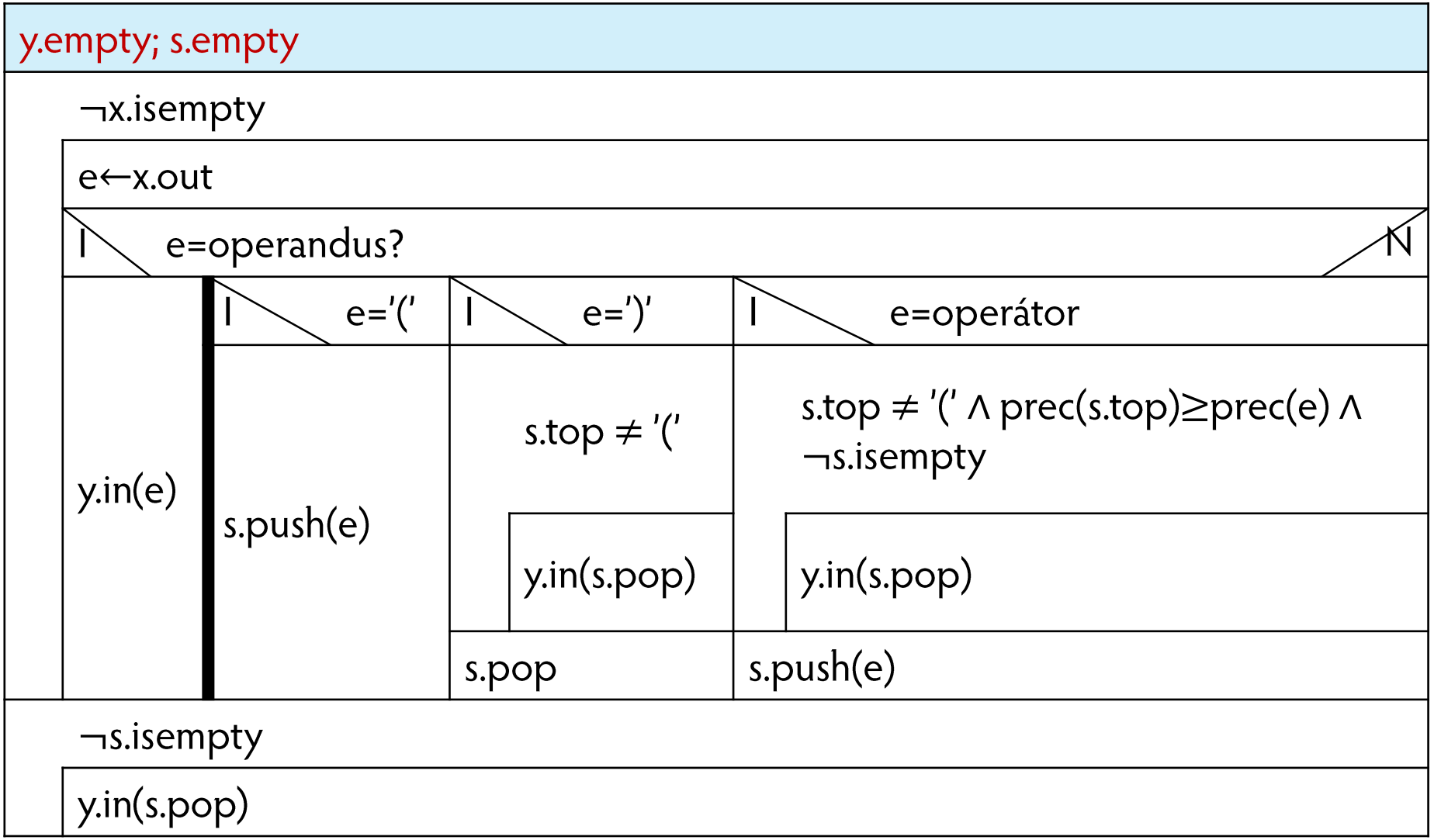
verem

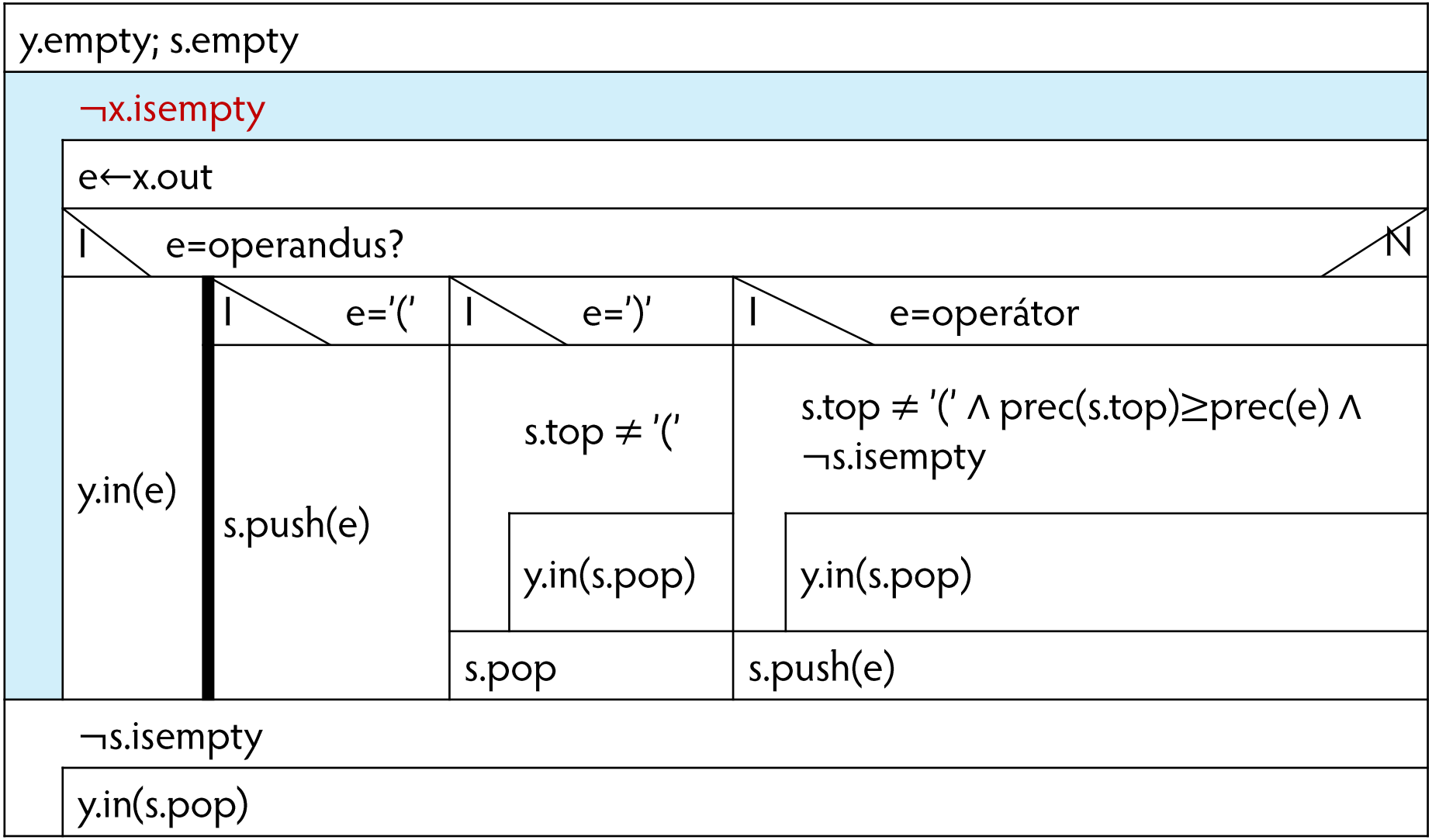
Az eredmény a verem tetején van.

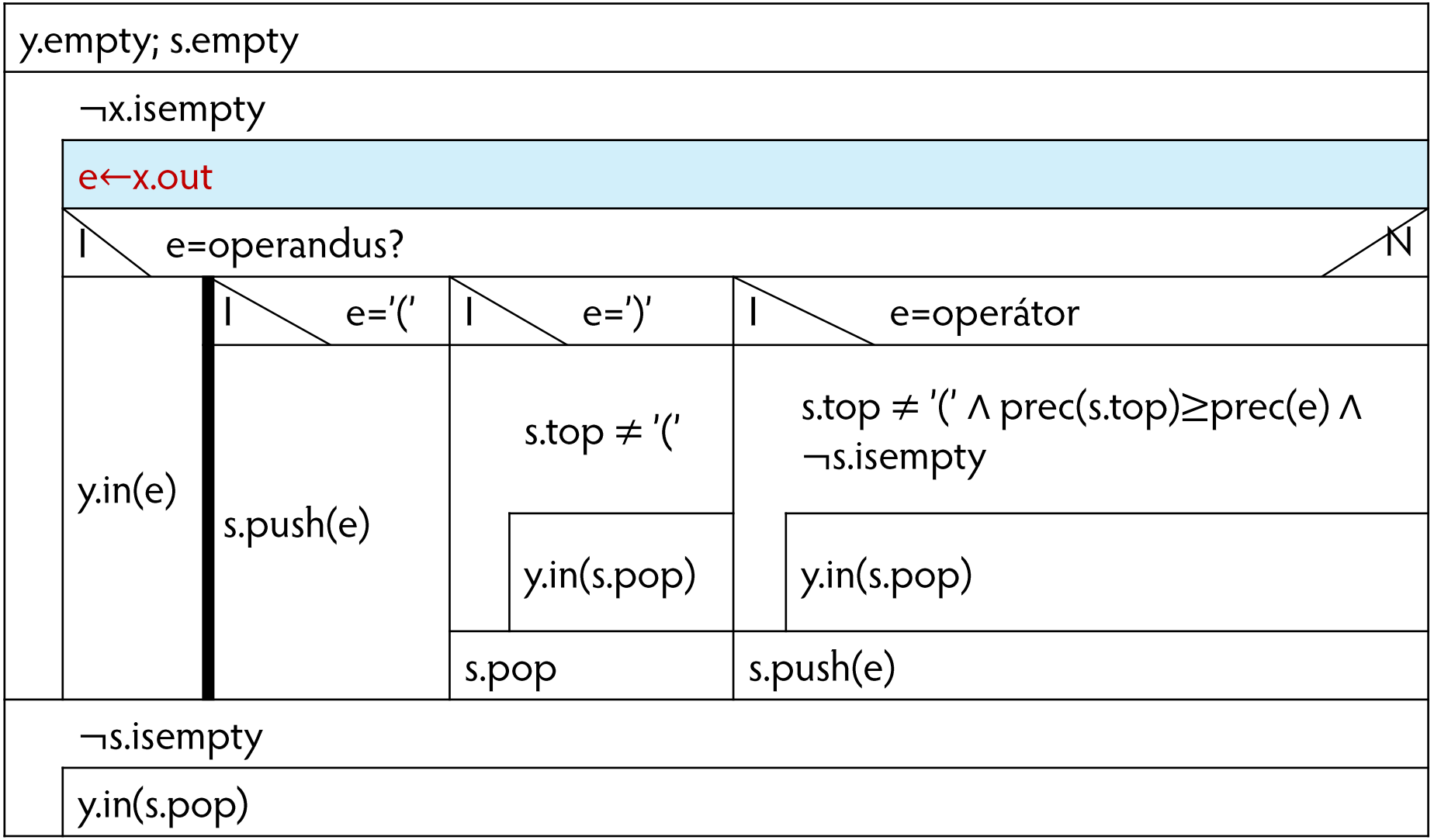
Lengyel formára alakítás

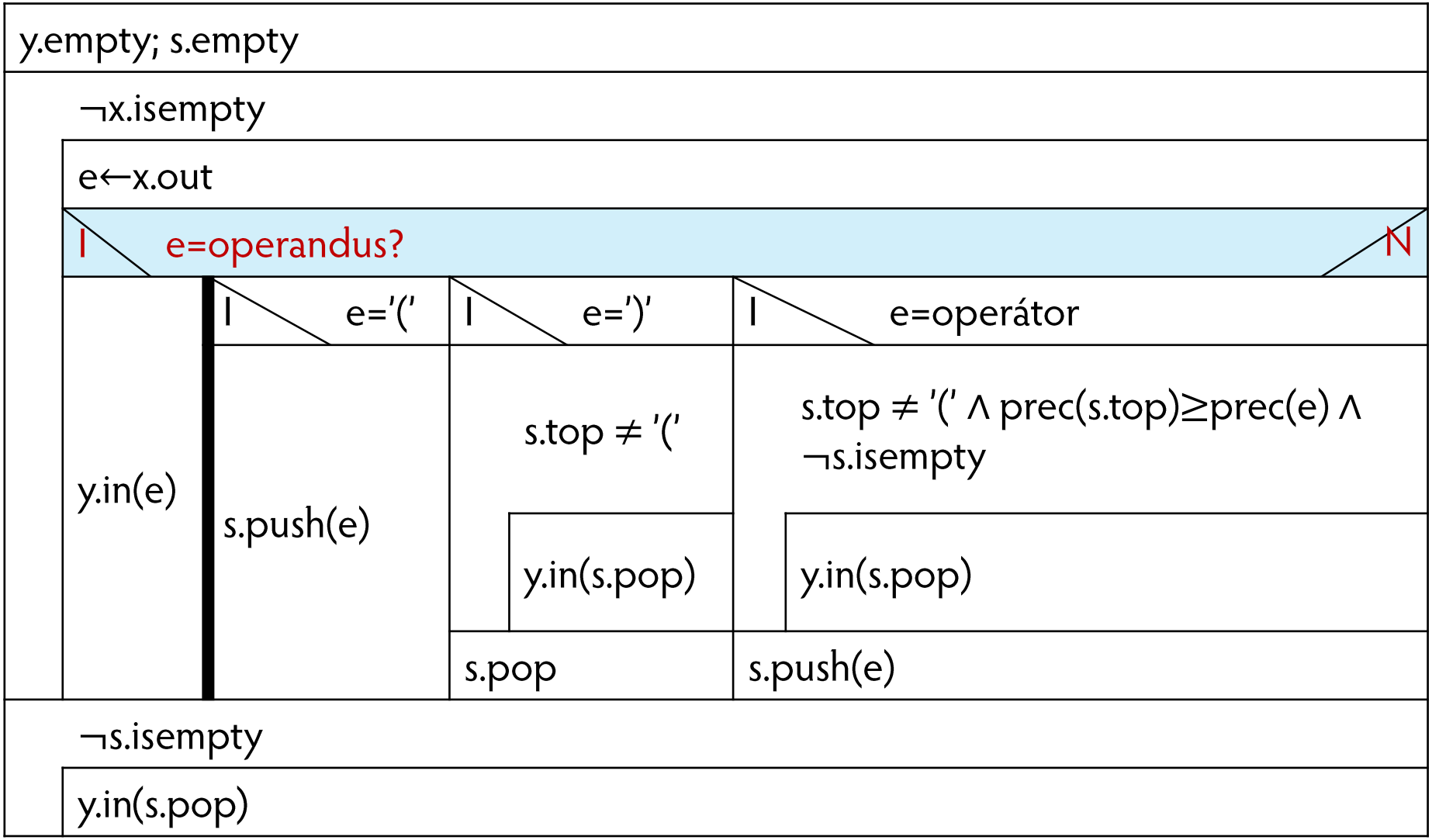
- Tegyük fel, hogy az x „token"-ekből álló sor tartalmazza a szintaktikailag helyes kifejezést,
- Egy token lehet:
  - Operandus, vagy
  - Operátor (bináris), vagy
  - „(" vagy „)"
  - az y sorba hozzuk létre a postfix formájú kifejezést,
- közben felhasználva az s vermet, mely operátorokat és nyitózárójelt tartalmazhat

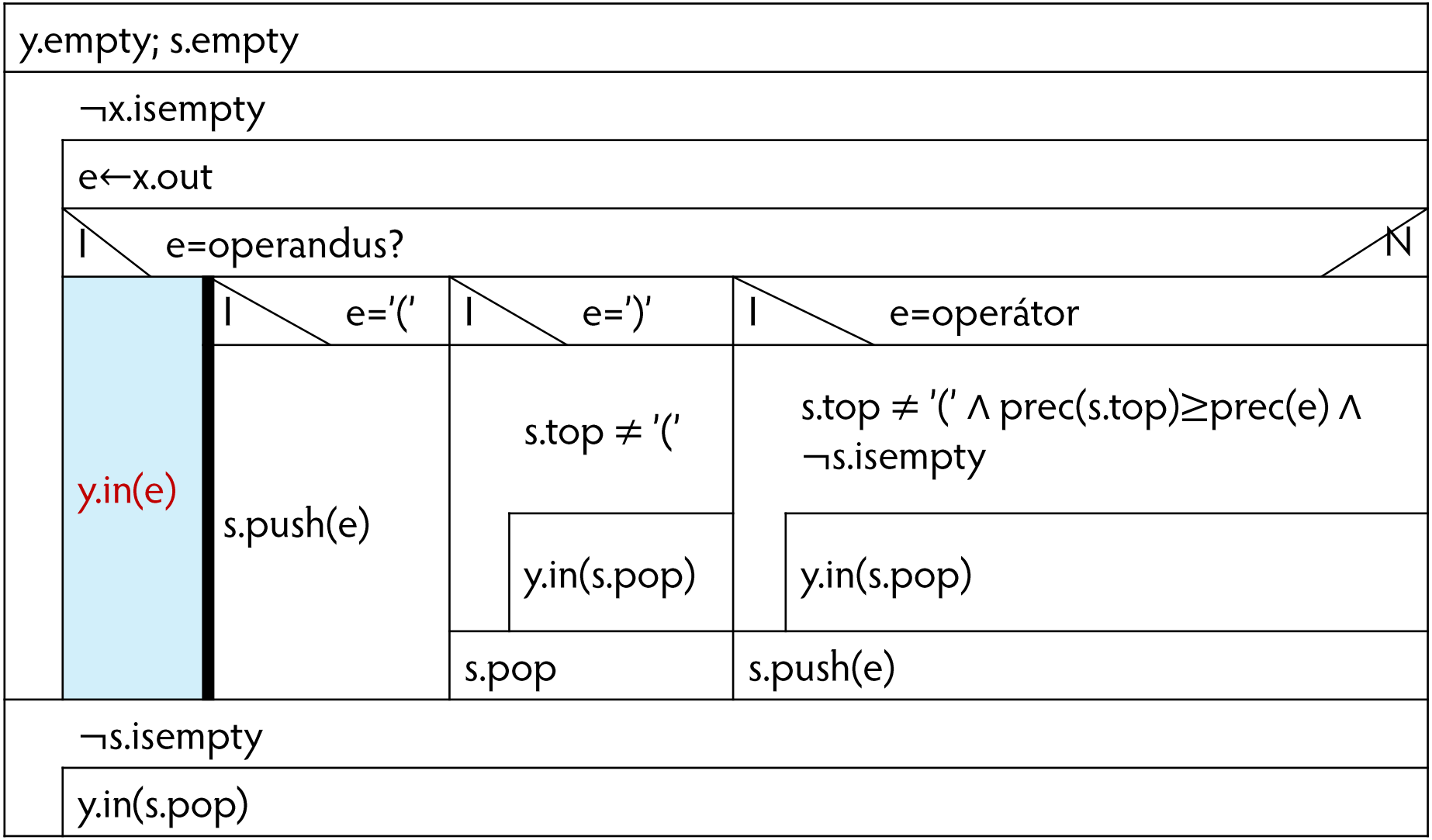
y.empty; s.empty

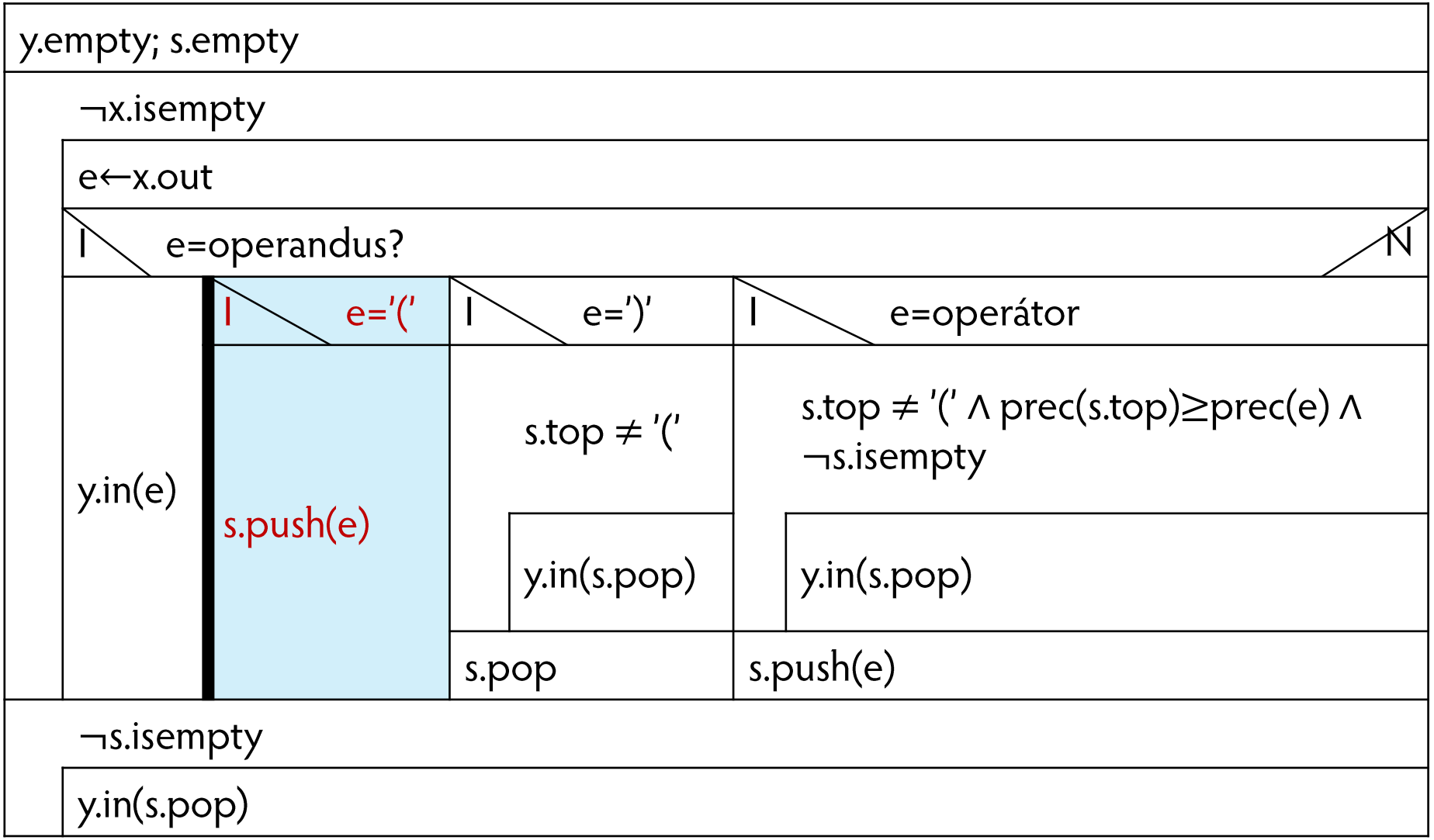


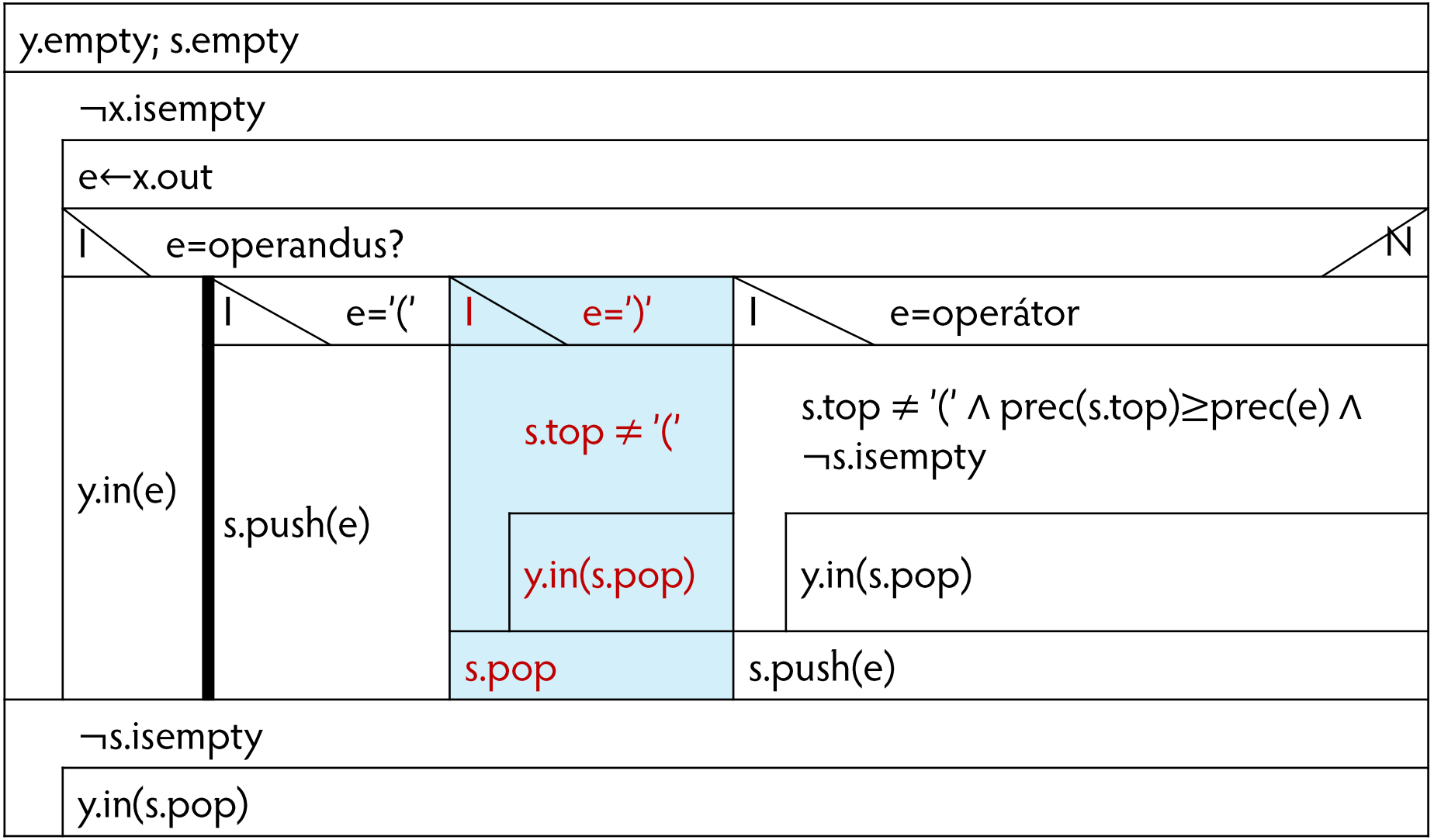

y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)

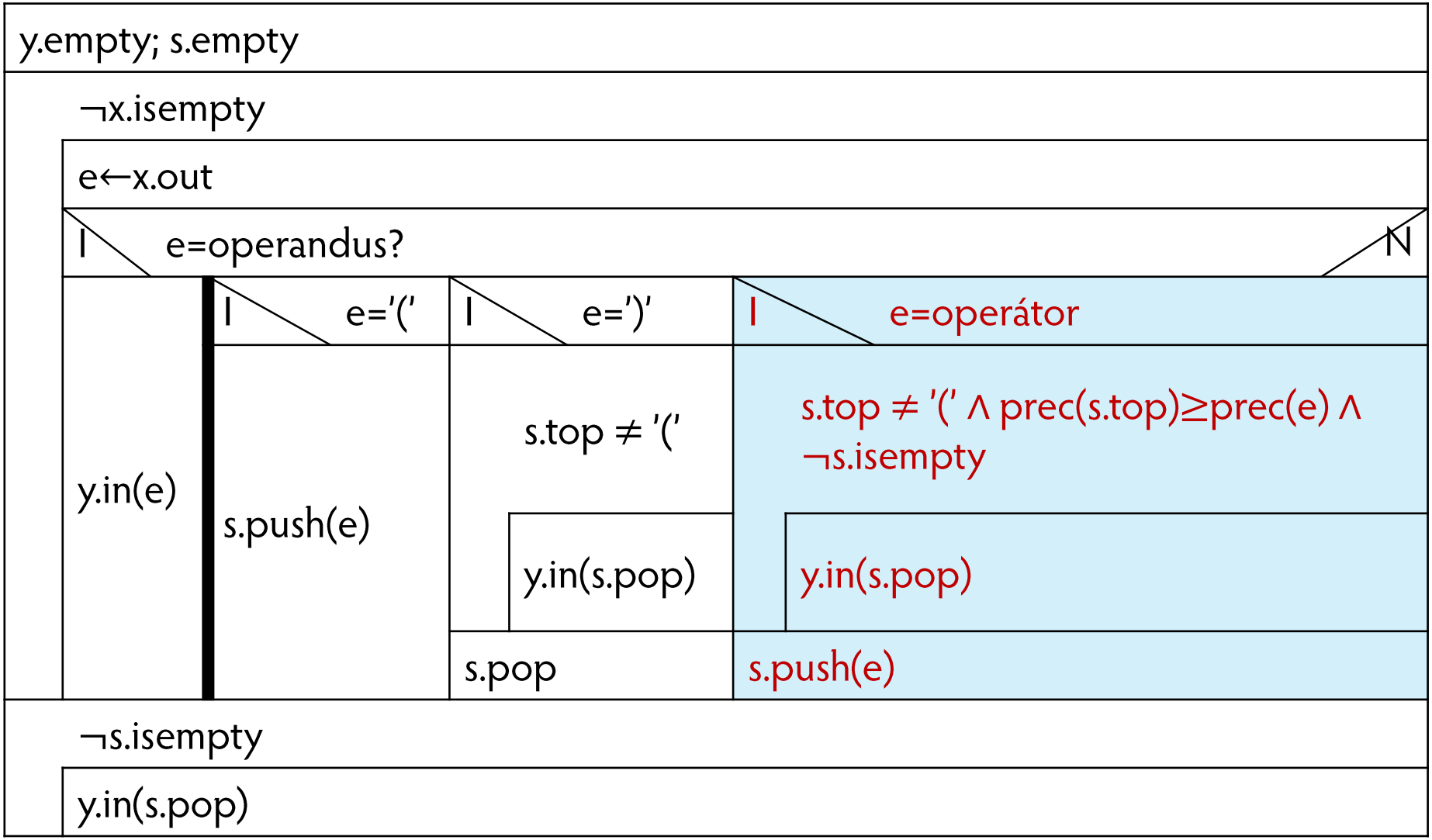

y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)

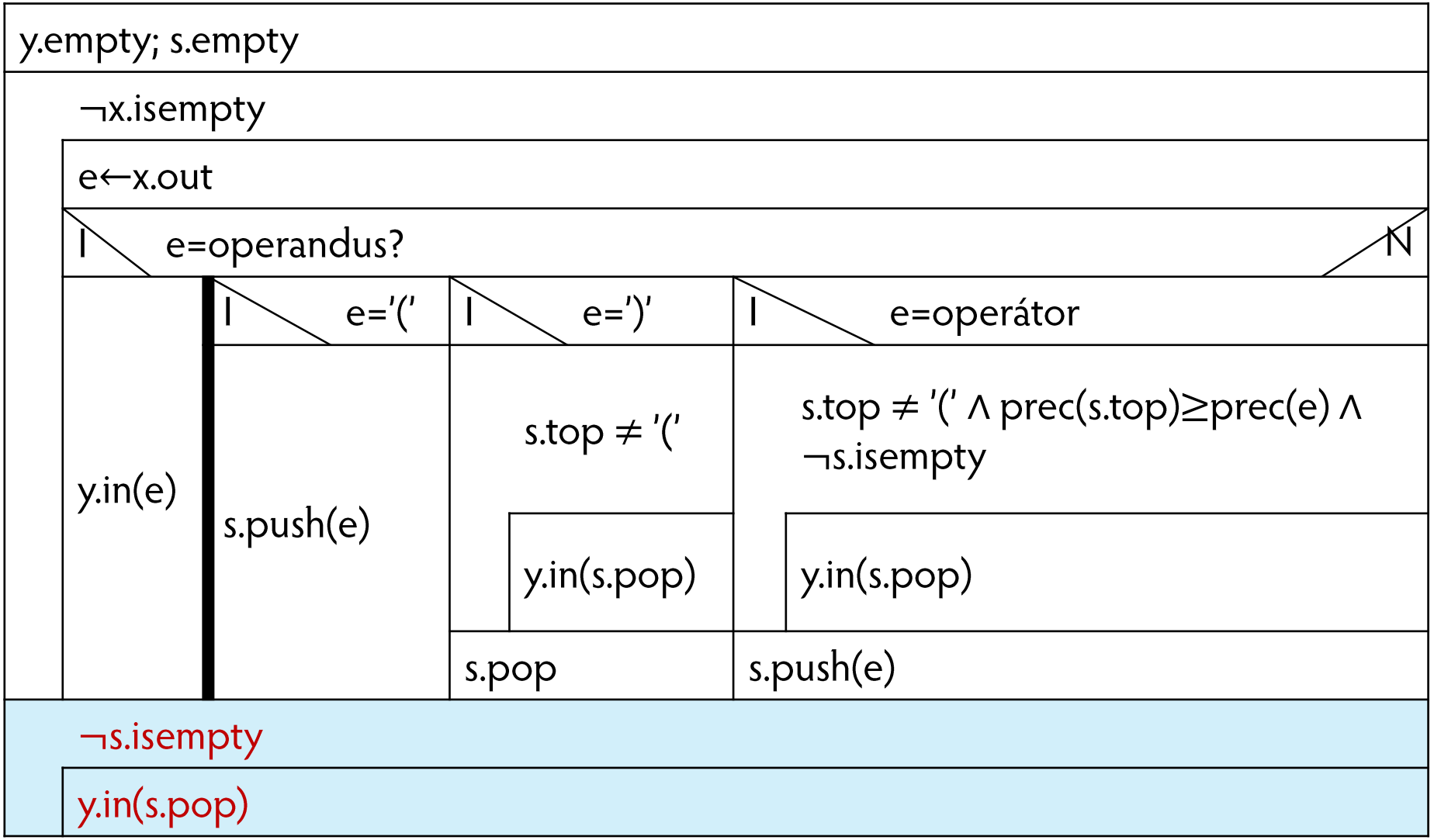

y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)

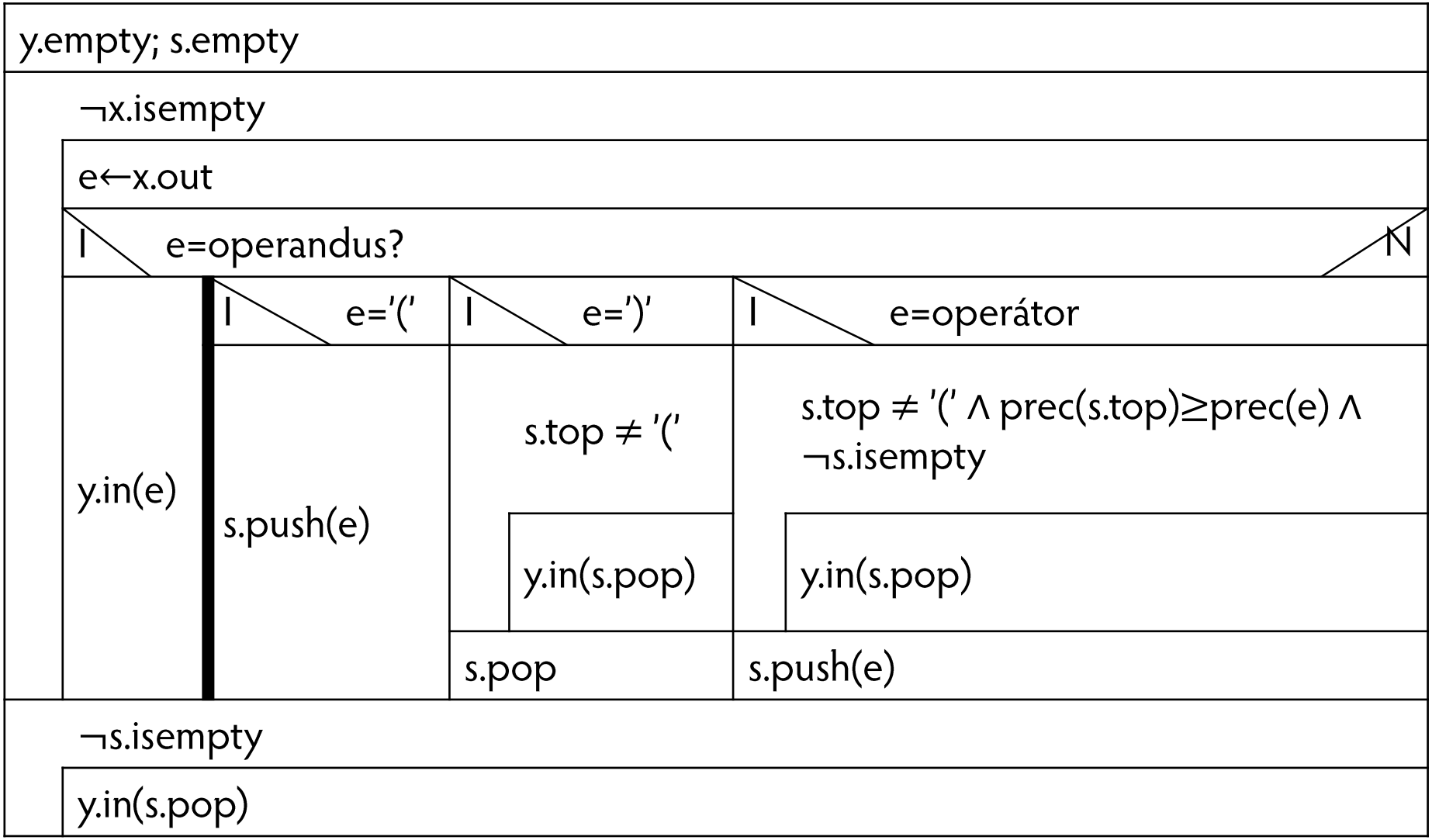

y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)


y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
I
e='('
I
e=')'
I
e=operátor
y.in(e)
s.push(e)
s.top ≠ '('
y.in(s.pop)
s.pop
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)


y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
I
e='('
I
e=')'
I
e=operátor
y.in(e)
s.push(e)
s.top ≠ '('
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
y.in(s.pop)
s.pop
s.push(e)
¬s.isempty
y.in(s.pop)


y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)


y.empty; s.empty
¬x.isempty
e←x.out
e=operandus?
I
N
e='('
e=')'
e=operátor
y.in(e)
s.push(e)
s.top ≠ '('
y.in(s.pop)
s.pop
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)


y.empty; s.empty
¬x.isempty
e←x.out
I
e=operandus?
N
y.in(e)
I
e='('
s.push(e)
I
e=')'
s.top ≠ '('
y.in(s.pop)
s.pop
I
e=operátor
s.top ≠ '(' ∧ prec(s.top)≥prec(e) ∧ ¬s.isempty
y.in(s.pop)
s.push(e)
¬s.isempty
y.in(s.pop)

Lengyel forma kiértékelése

- Tegyük fel, hogy az y sor tartalmazza a postfix alakban lévő kifejezést,
- Értékeljük ki, felhasználva a v vermet, mely operandusokat tartalmazhat
- Az eredményt tároljuk a z változóban

v.empty; z←0

¬y.isempty

e←y.out

I	e=operandus?	N
v.push(e)		op2←v.pop op1←v.pop r←Művelet végrehajtása v.push(r)

z←v.pop

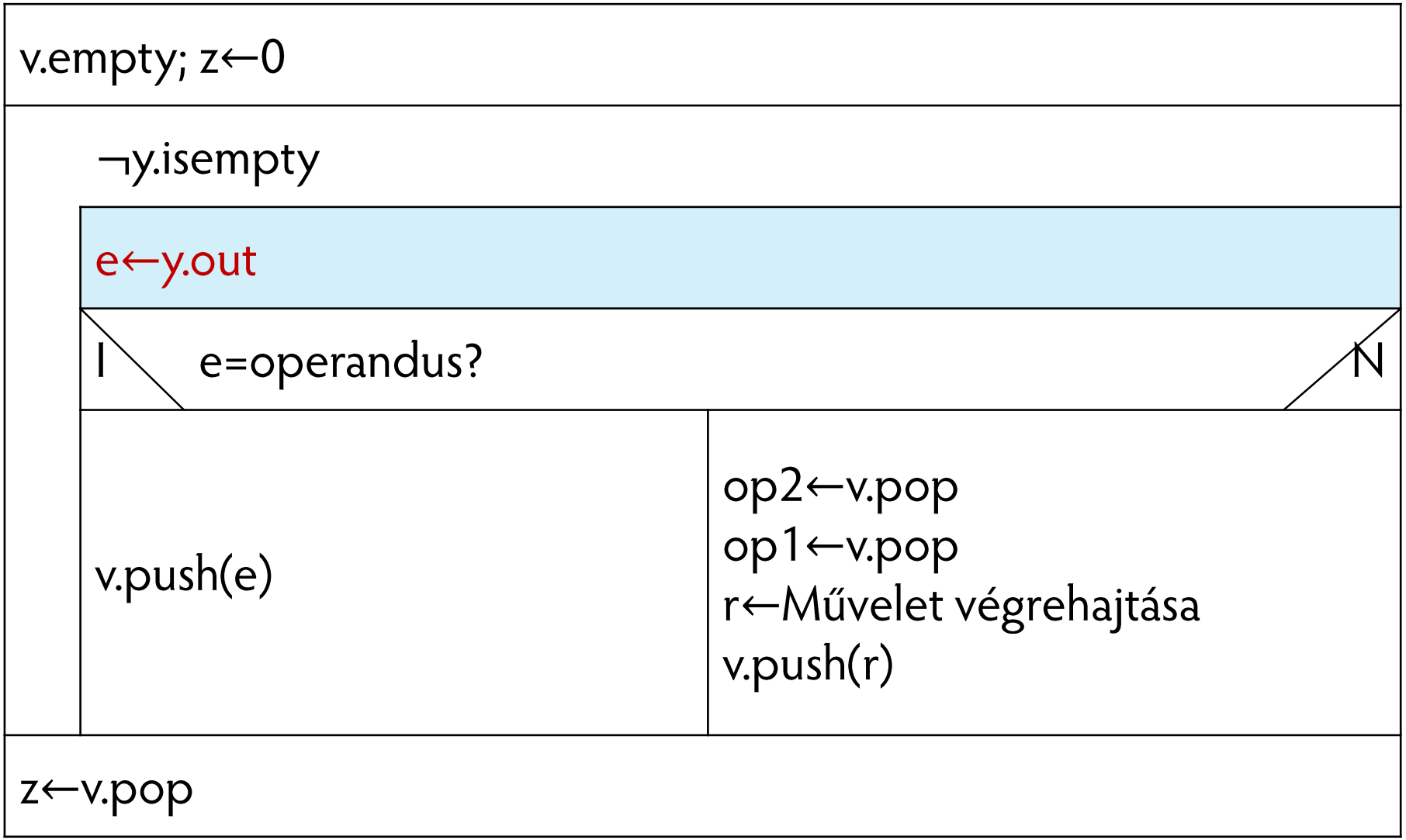
v.empty; z←0

¬y.isempty

e←y.out

I	e=operandus?	N
v.push(e)		op2←v.pop op1←v.pop r←Művelet végrehajtása v.push(r)

z←v.pop


v.empty; z←0
¬y.isempty
e←y.out
I
e=operandus?
N
v.push(e)
op2←v.pop
op1←v.pop
r←Művelet végrehajtása
v.push(r)
z←v.pop

v.empty; z←0

¬y.isempty

e←y.out

I e=operandus? N

v.push(e)	op2←v.pop op1←v.pop r←Művelet végrehajtása v.push(r)

z←v.pop

v.empty; z←0

¬y.isempty

e←y.out

I e=operandus? N

v.push(e)

op2←v.pop
op1←v.pop
r←Művelet végrehajtása
v.push(r)

z←v.pop

v.empty; z←0

¬y.isempty

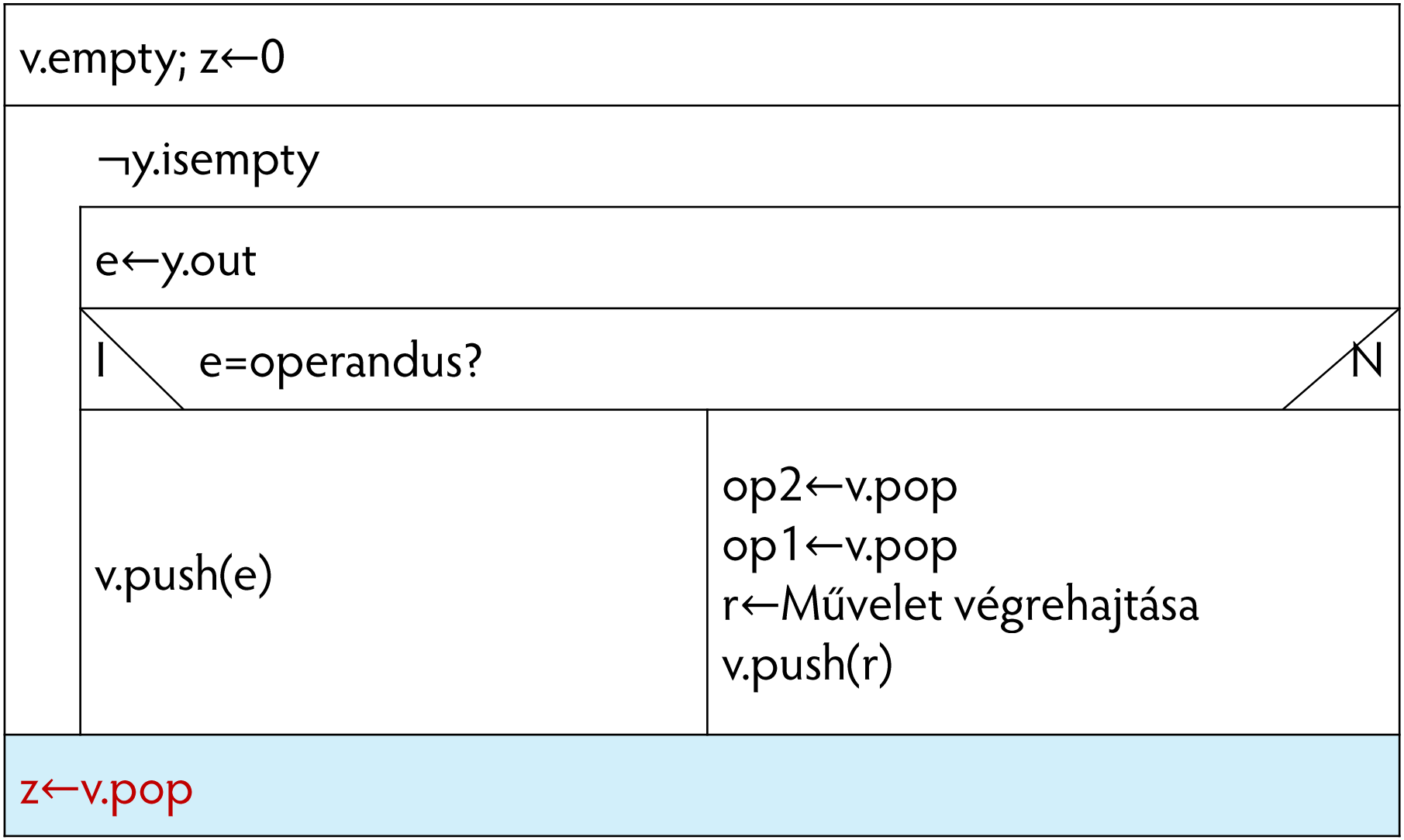
e←y.out

I e=operandus? N

v.push(e)

op2←v.pop
op1←v.pop
r←Művelet végrehajtása
v.push(r)

z←v.pop


v.empty; z←0
¬y.isempty
e←y.out
I
e=operandus?
N
v.push(e)
op2←v.pop
op1←v.pop
r←Művelet végrehajtása
v.push(r)
z←v.pop

v.empty; z←0

¬y.isempty

e←y.out

I e=operandus? N

v.push(e)

op2←v.pop
op1←v.pop
r←Művelet végrehajtása
v.push(r)

z←v.pop

Tömb, Lista, Fa, …

Következő alkalommal