# لیست

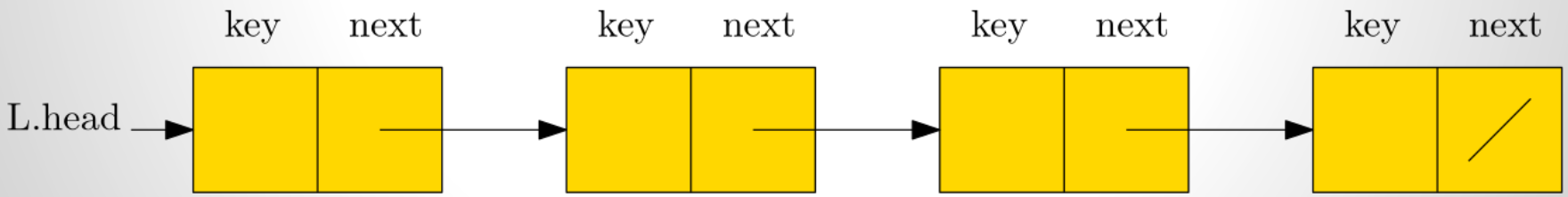
# لیست (شهود)

| شماره درس | گروه | واحد | نام درس | پیشنیاز و همنیاز | ظرفیت | تعداد ثبت نامی | نام استاد |
|---|---|---|---|---|---|---|---|
| 22015 | 5 | 4 | ریاضی عمومی ۱ | | 20 | 12 | سحر قاجار |
| 22015 | 1 | 4 | ریاضی عمومی ۱ | | 30 | 30 | سیدرضا مقدسی |
| 22015 | 4 | 4 | ریاضی عمومی ۱ | | 29 | 29 | علیرضا رنجبرمطلق |
| 22015 | 3 | 4 | ریاضی عمومی ۱ | | 16 | 16 | محمدرضا رزوان |
| 22015 | 2 | 4 | ریاضی عمومی ۱ | | 34 | 34 | سیدرضا مقدسی |
| 22016 | 1 | 4 | ریاضی عمومی ۲ | پیش‌نیاز: 22015 | 301 | 301 | محمدهادی مستفید |
| 22016 | 2 | 4 | ریاضی عمومی ۲ | پیش‌نیاز: 22015 | 325 | 325 | محسن جمالی |
| 22034 | 3 | 3 | معادلات دیفرانسیل | همنیاز: 22016 | 250 | 246 | حمیدرضا فنائی |
| 22034 | 2 | 3 | معادلات دیفرانسیل | همنیاز: 22016 | 250 | 219 | حمیدرضا فنائی |
| 22034 | 1 | 3 | معادلات دیفرانسیل | همنیاز: 22016 | 265 | 265 | محمدرضا پورنکی |
| 22035 | 1 | 3 | ریاضی مهندسی | پیش‌نیاز: 22034 | 143 | 143 | فرشته گازری |
| 22035 | 2 | 3 | ریاضی مهندسی | پیش‌نیاز: 22034 | 140 | 130 | حمیدرضا فرهادی |
| 22064 | 1 | 4 | آمار و کاربرد آن | پیش‌نیاز: 22089 | 71 | 71 | میر امید حاجی میر صادقی |
| 22080 | 1 | 0 | پروژه کارشناسی | | | 15 | |
| 22089 | 1 | 4 | احتمال و کاربرد آن | پیش‌نیاز: 22016 | 40 | 32 | سحر قاجار |

# تعریف (لیست)

ساختمان داده‌ای است که در آن داده‌ها با **نظم خطی** کنار هم قرار دارند.

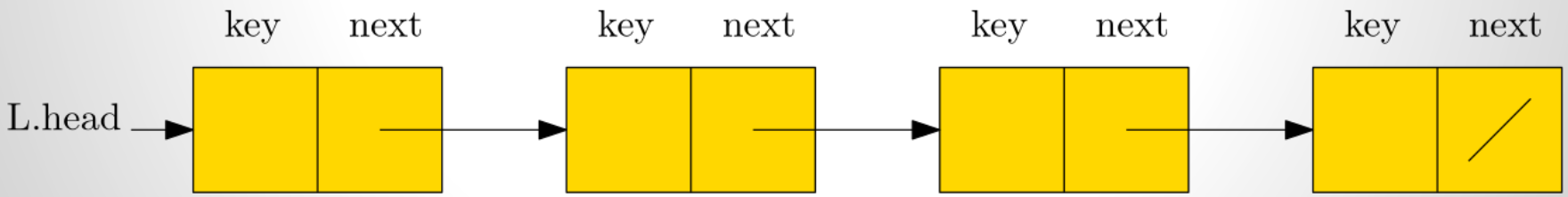
از آنجایی که ترتیب لیست‌ها توسط اشاره‌گرها مشخص می‌شود، به آن **لیست پیوندی** نیز می‌گویند.



# تعریف (لیست)

هر عنصر از لیست دو مولفه دارد

1. کلید: که مقدار مورد نظر مارا در خود نگه می‌دارد
2. بعدی: که اشاره‌گری به عنصر بعدی لیست است



# انواع لیست

بر حسب **مدل** و **عملیات** مجاز بر روی لیست‌ها، داده‌ساختارهای مختلفی بدست می‌آید:

- لیست‌های یکسویه، دوسویه و حلقوی
- پشته
- صف

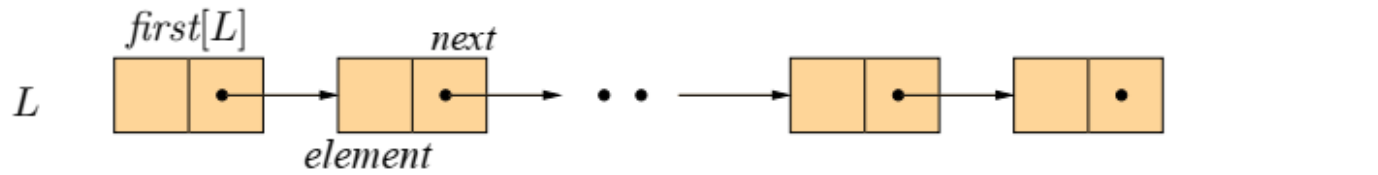
# اعمالی که بر روی لیست‌ها انجام می‌دهیم

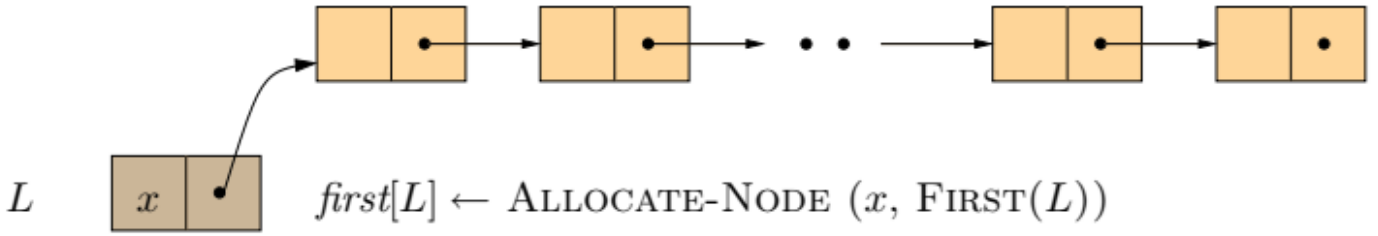
- $Create - List(L)$ایجاد یک لیست تهی :
- $Size(L)$را برمی‌گرداند $L$ تعداد عناصر لیست :
- $First(L)$را برمی‌گرداند $L$ عنصر اول :
- $isEmpty(L)$مشخص می‌کند که آیا لیست خالی است یا خیر :
- $Insert - First(L, x)$در ابتدای لیست $x$ درج عنصری با کلید :
- $Insert - After(L, x, n)$در لیست $n$ پس از عنصر $x$ درج عنصر : $L$
- $Delete - First(L)$را حذف می‌کند $L$ عنصر اول لیست :
- $Delete - After(L, n)$حذف می‌کند $L$ را از لیست $n$ عنصر پس از عنصر :

# پیاده‌سازی

```
CREATE(L)
    1  size[L] ← 0

SIZE(L)
    1  return size[L]

FIRST(L)
    1  if SIZE(L) ≠ 0
    2    then return first[L]
    3    else error list is empty

ISEMPTY(L)
    1  return SIZE(L) = 0
```

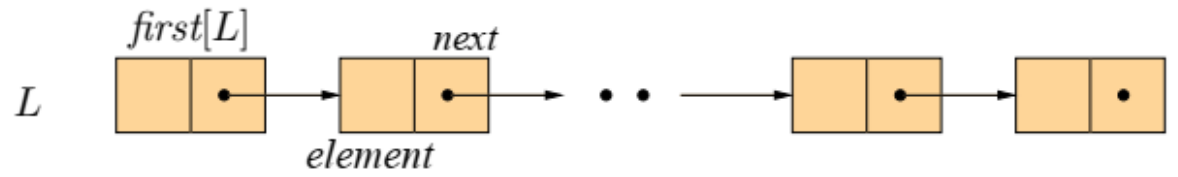
```
INSERT-FIRST(L, x)
    1  first[L] ← Allocate-Node (x, First(L))
    2  size[L] ← size[L] + 1
```

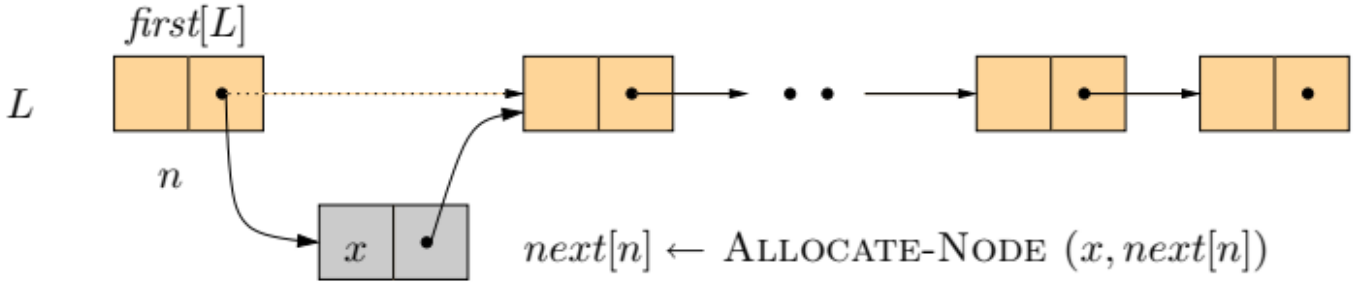



INSERT-FIRST$(L, x)$

```
INSERT-AFTER (L, x, n)
1  if n = null
2    then error element is empty
3  next[n] ← ALLOCATE-NODE (x, next[n])
4  size[L] ← size[L] + 1
```

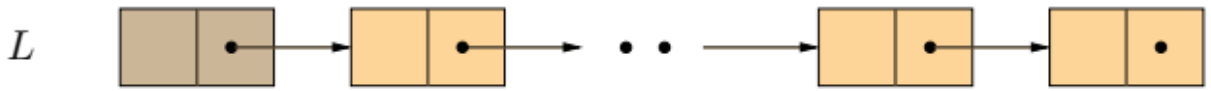



$next[n] \leftarrow$ ALLOCATE-NODE $(x, next[n])$

INSERT-AFTER$(L, x, n)$

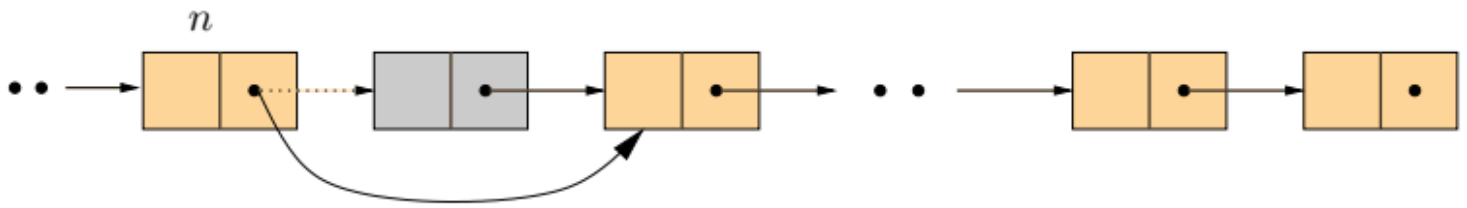
```
DELETE-FIRST(L)
 1  if ISEMPTY (L)
 2    then error list is empty
 3  n ← FIRST(L)
 4  first[L] ← next[n]
 5  FREE-NODE(n)
 6  size[L] ← size[L] − 1
```




$first[L] \leftarrow next[\ first[L]]$

DELETE-FIRST($L$)

```
DELETE-AFTER(L, n)
 1  if ISEMPTY (L)  or n = null  or next[n] = null
 2    then error element does not exist
 3  r ← next[n]
 4  next[n] ← next[r]
 5  FREE-NODE(r)
 6  size[L] ← size[L] − 1
```




$next[n] \leftarrow next[next[n]]$

DELETE-AFTER$(L, n)$

# پیچیدگی زمانی

- پیچیدگی زمانی دستورات فوق چقدر است؟(آیا به اندازه لیست ربطی دارد)

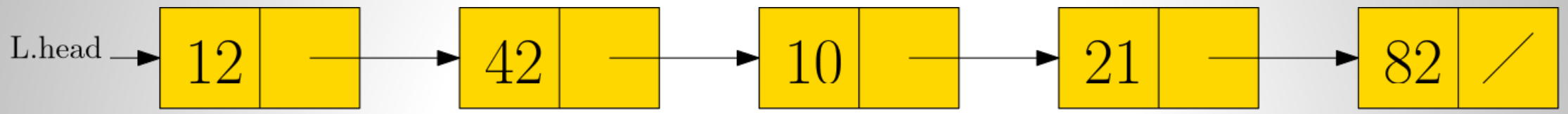
# پیچیدگی زمانی

- روشن است که هر یک از دستورات فوق مستقل از اندازه لیست، در زمان ثابتی انجام می‌شود.
- در نتیجه پیچیدگی زمانی آن $O(1)$ است.

# جست و جو در لیست

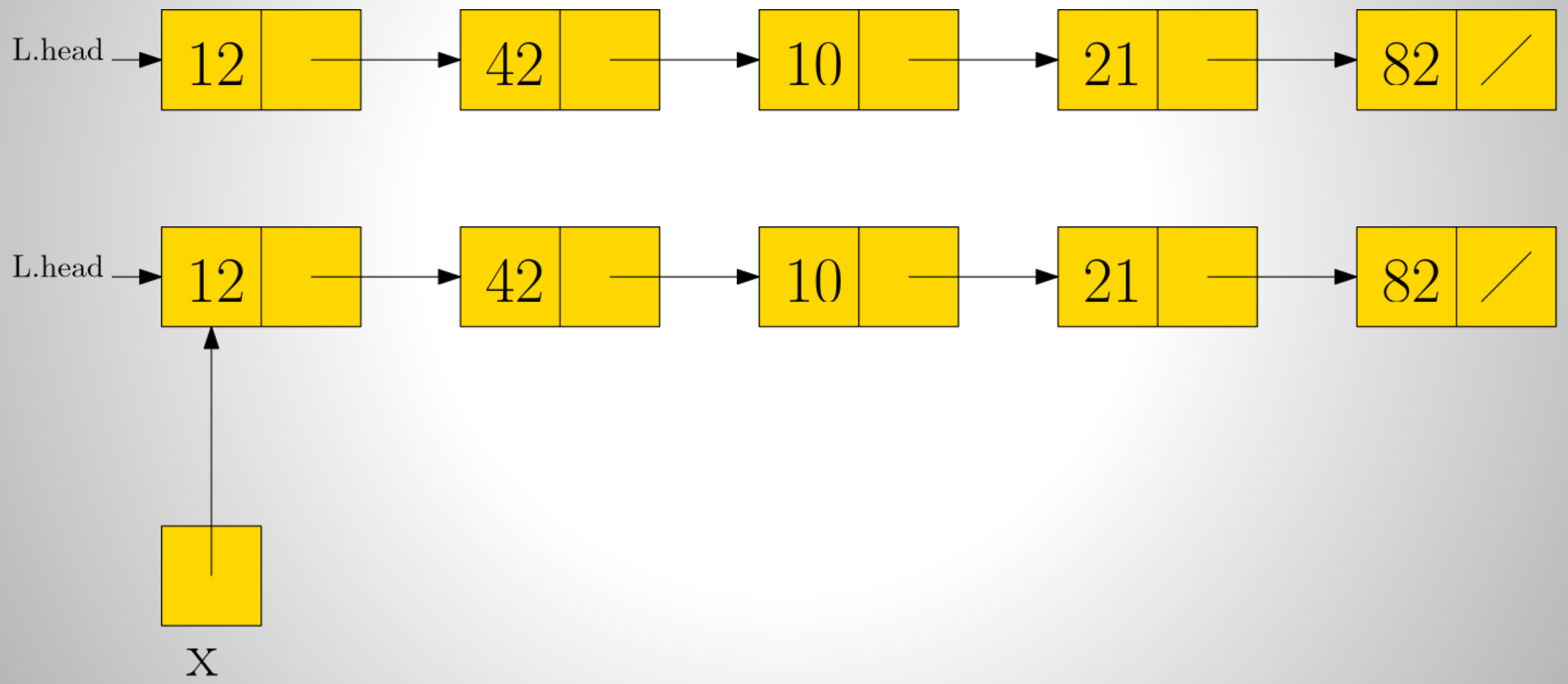
```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3        x = x.next
4  return x
```

# جست و جو در لیست مثال

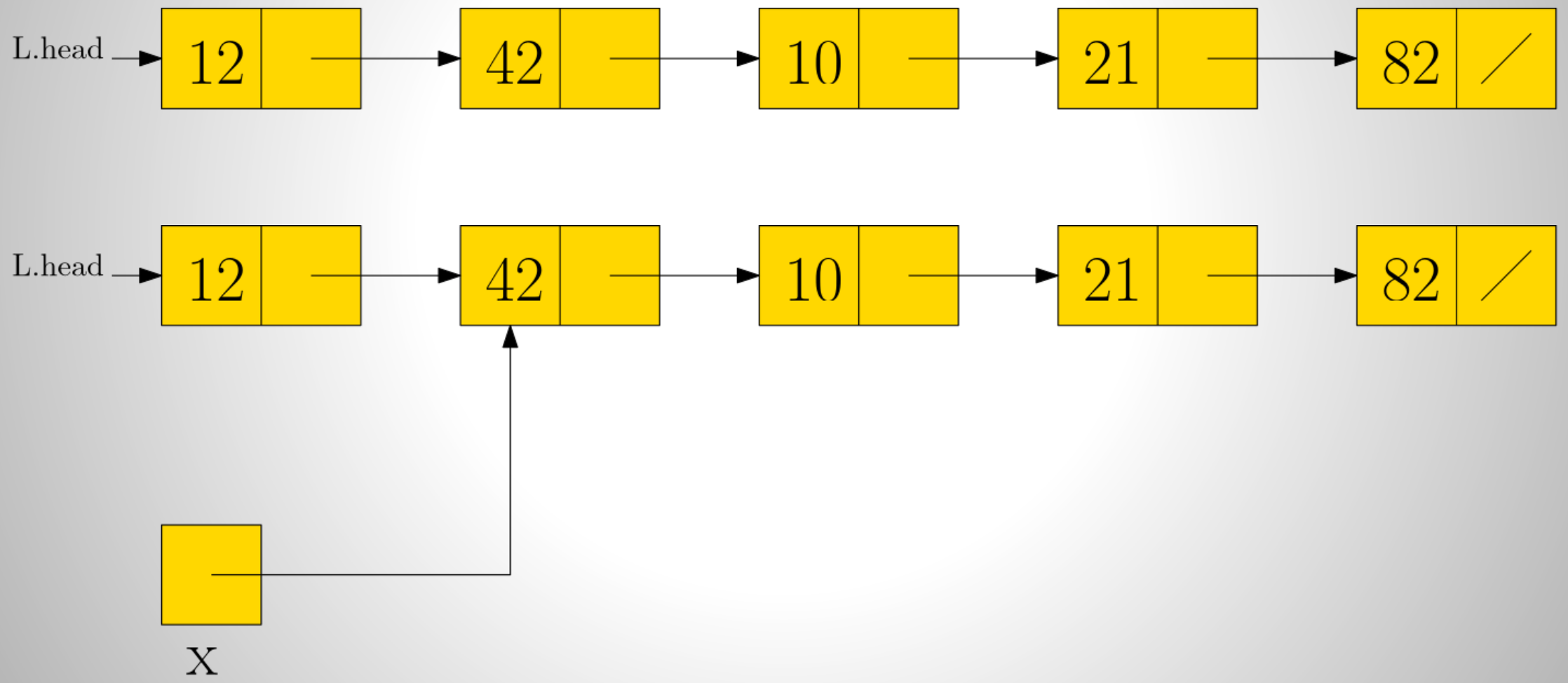



عنصر ۱۰ را بیاب.

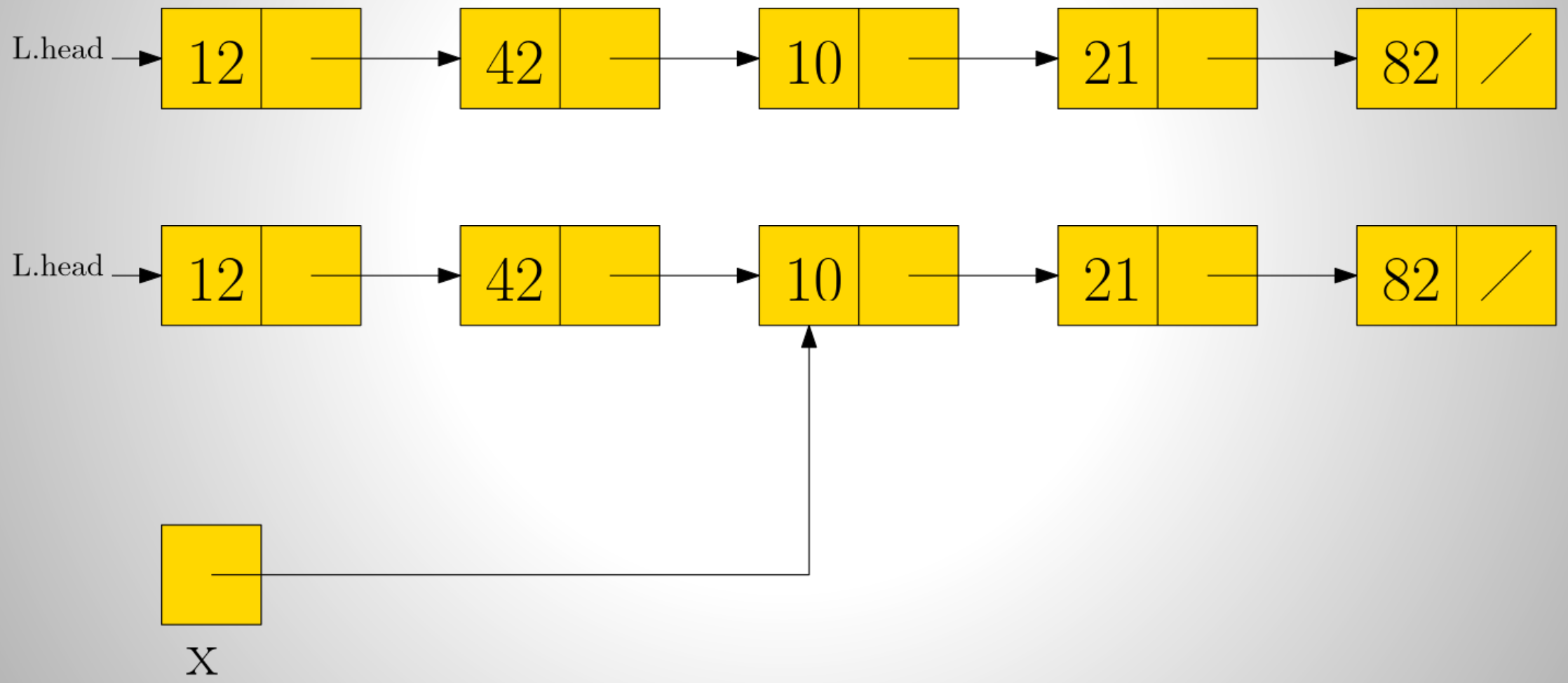
# جست و جو در لیست مثال



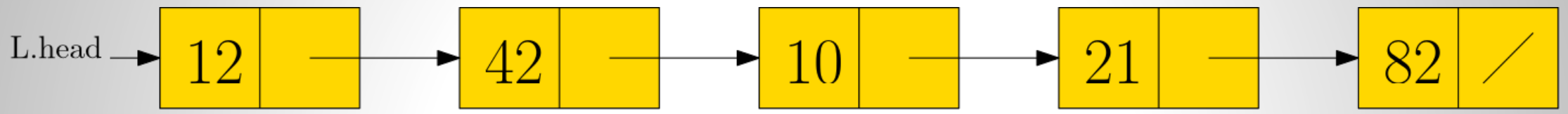
# جست و جو در لیست مثال



# جست و جو در لیست مثال



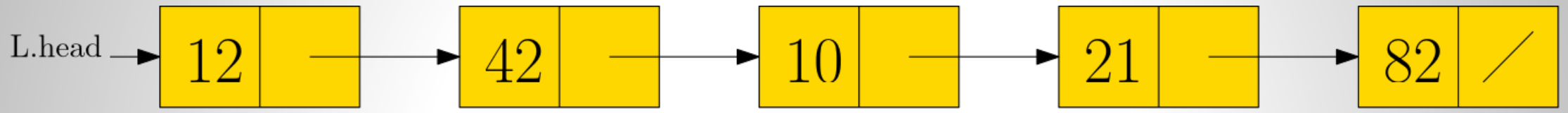
# جست و جو در لیست مثال




عنصر ۱۱ را بیاب.

پیچیدگی زمانی جست و جو چقدر است؟

# جست و جو در لیست مثال




عنصر ۱۱ را بیاب.

پیچیدگی زمانی جست و جو: $O(n)$

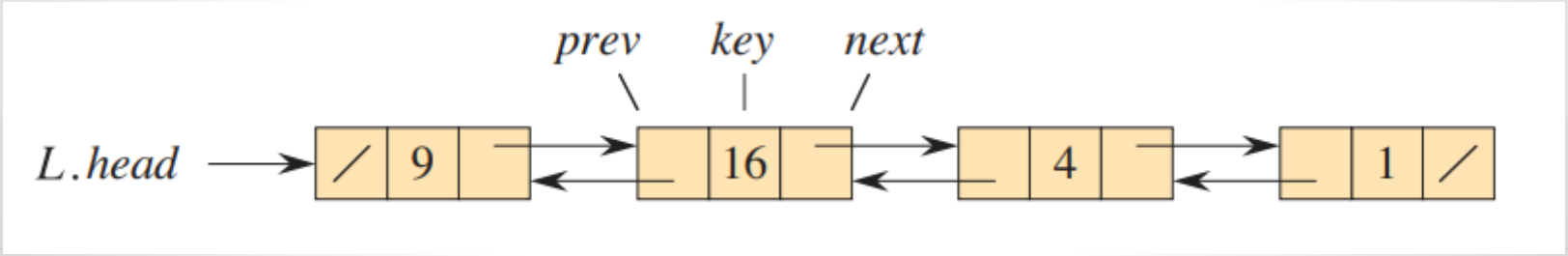
# لیست دوسویه

لیستی است که هر عنصر علاوه بر عنصر بعد عنصر قبل خود را نیز نگه می‌دارد.

# تعریف (لیست دوسویه)

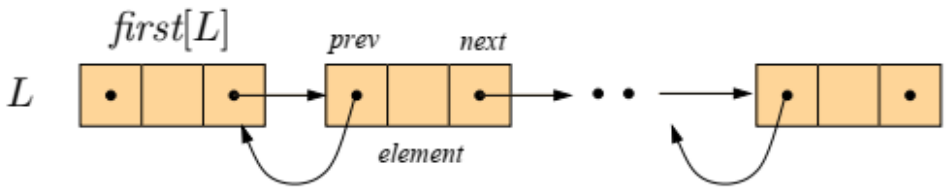
هر عنصر از لیست سه مولفه دارد

1. کلید: که مقدار مورد نظر مارا در خود نگه می‌دارد
2. بعدی: که اشاره‌گری به عنصر بعدی لیست است
3. قبلی: که اشاره‌گری به عنصر قبلی لیست است

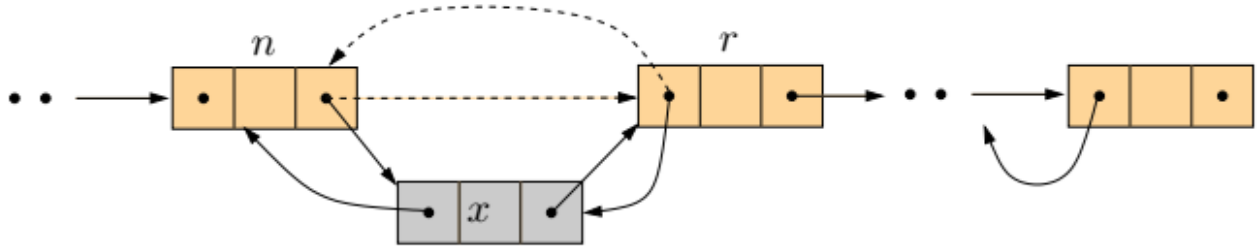


# درج و حذف در لیست دوسویه‌ی خطی

```
INSERT-AFTER(L, x, n)
    ▷ عنصری با محتوای x را پس از عنصر n در لیست دوسویه‌ی L درج می‌کند
 1  if ISEMPTY(L)  or n = null
 2    then  error  element n does not exist
 3  r ← next[n]
 4  next[n] ← ALLOCATE-NODE (x, n, r)
 5  prev[r] ← next[n]
 6  size[L] ← size[L] + 1
```

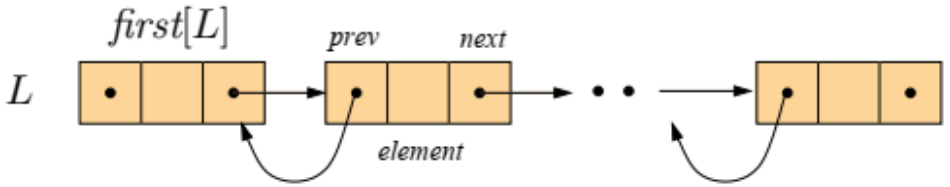



لیست خطی دوسویه‌ی

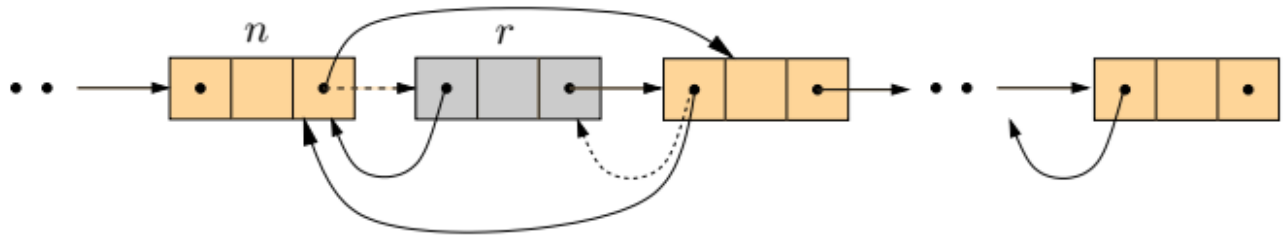



INSERT-AFTER$(L, x, n)$

```
DELETE-AFTER(L, n)
      ▷ عنصر بعدی n را در لیست دو سویه‌ی L حذف می‌کند
 1  if ISEMPTY(L)  or  n = null  or  next[n] = null
 2    then  error  element does not exist
 3  r ← next[n]
 4  if next[r] ≠ null
 5    then prev[next[r]] ← n
 6  next[n] ← next[r]
 7  FREE-NODE(r)
 8  size[L] ← size[L] − 1
```




لیست خطی دوسویه‌ی

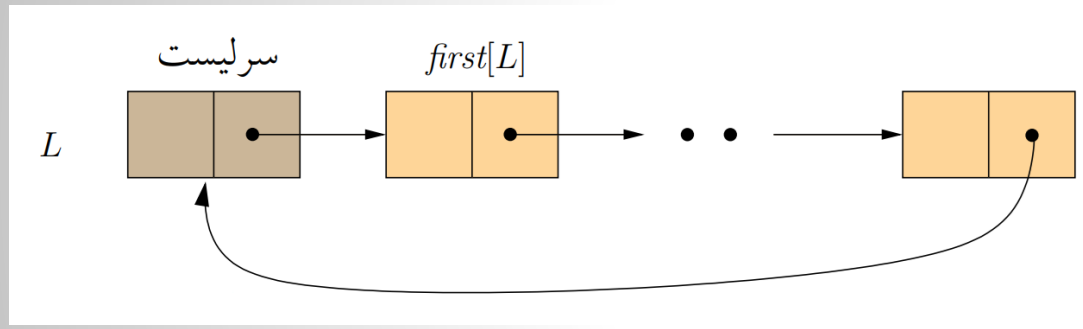



DELETE-AFTER$(L, n)$

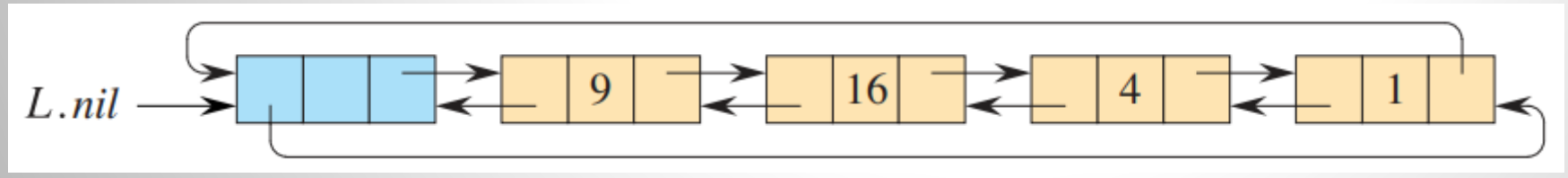
# لیست حلقوی

لیستی است که عنصر ابتدایی و عنصر انتهایی به هم متصل هستند

یکسویه حلقوی




- دوسویه حلقوی



# مسئله ۱

- یک لیست را به عنوان ورودی دریافت کرده، تمامی عناصر تکراری آن را حذف کرده، به طوری که هر عنصر فقط یک بار ظاهر شود.

- ورودی: یک لیست
- خروجی: لیست بدون عناصر تکراری

# مسئله ۱

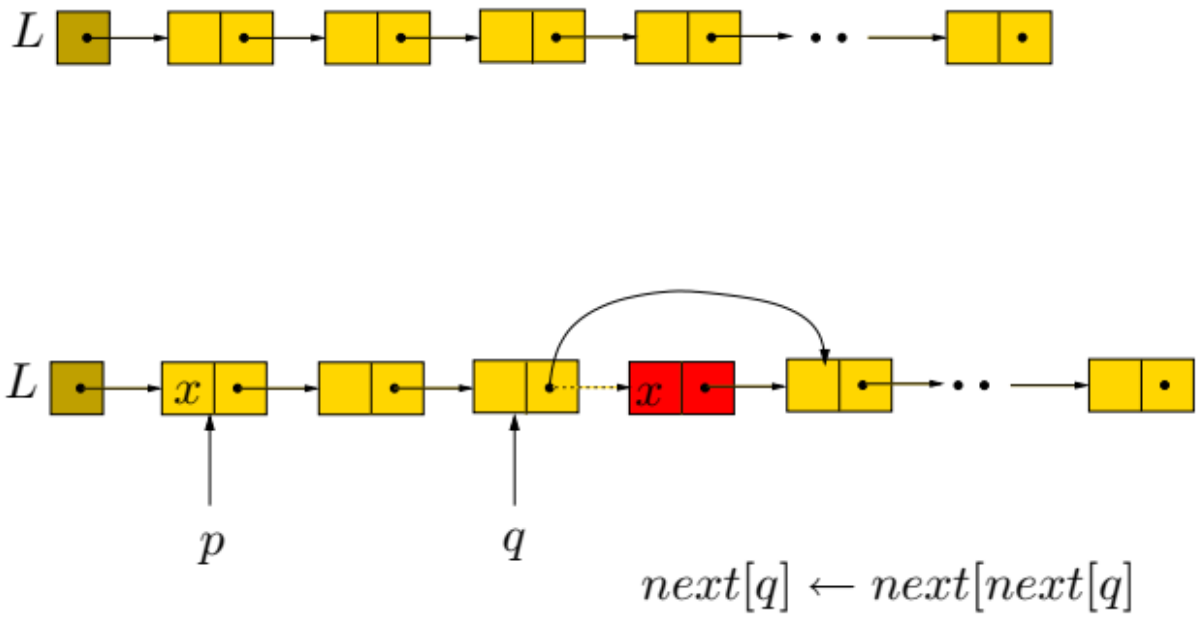
- یک لیست را به عنوان ورودی دریافت کرده، تمامی عناصر تکراری آن را حذف کرده، به طوری که هر عنصر فقط یک بار ظاهر شود.

- ورودی: یک لیست
- خروجی: لیست بدون عناصر تکراری

- با استفاده از: $Delete - After(L, n)$

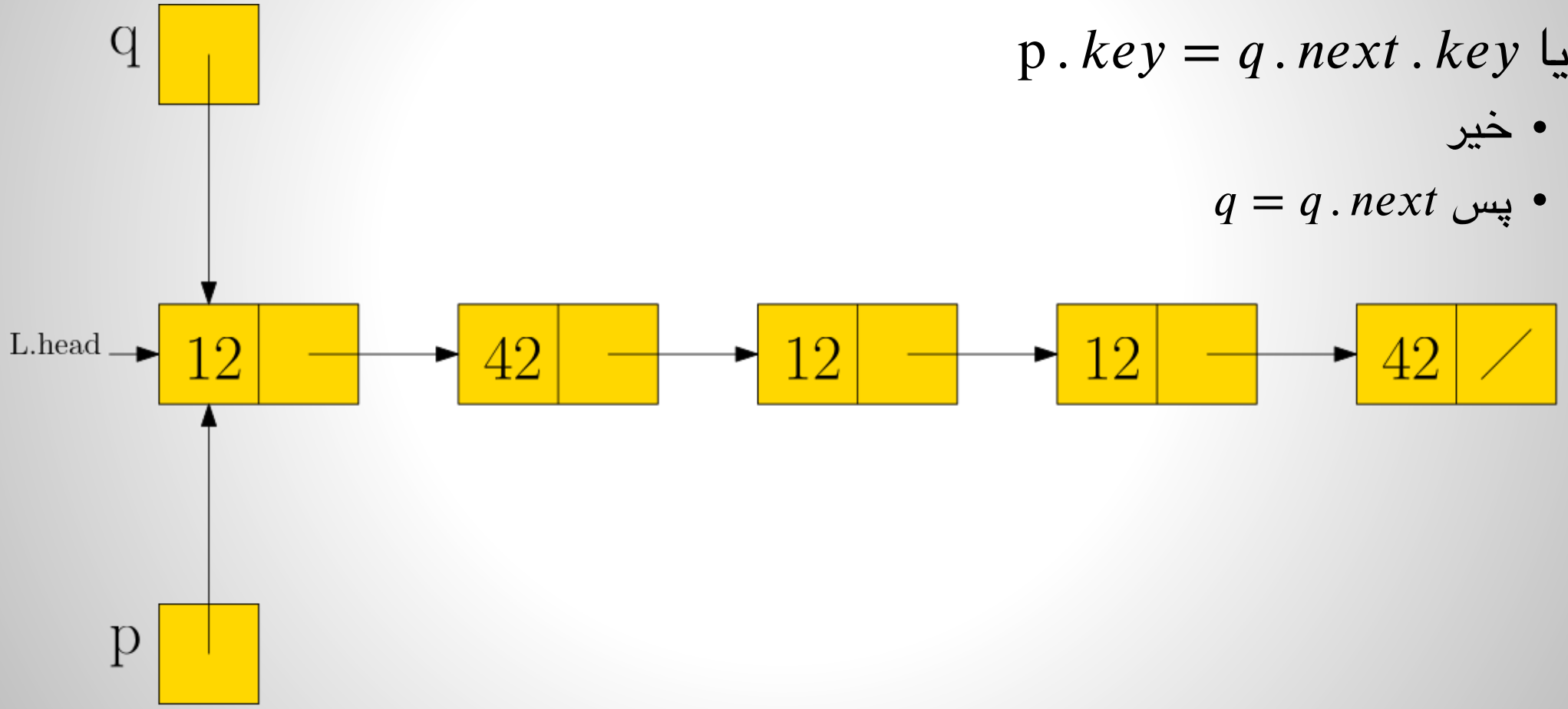
```
PURGELIST(L)
      ▷ همه‌ی عناصر یکسان را جز یکی حذف می‌کند
 1  p ← FIRST(L)
 2  while p <> null
 3        do q ← p
 4           while next[q] <> null
 5                 do if element[p] = element[next[q]]
 6                       then DELETE-AFTER(L, q)
 7                       else q ← next[q]
 8           p ← next[p]
```

# عملیات دیگر بر روی لیست‌ها: حذف عناصر تکراری در یک لیست



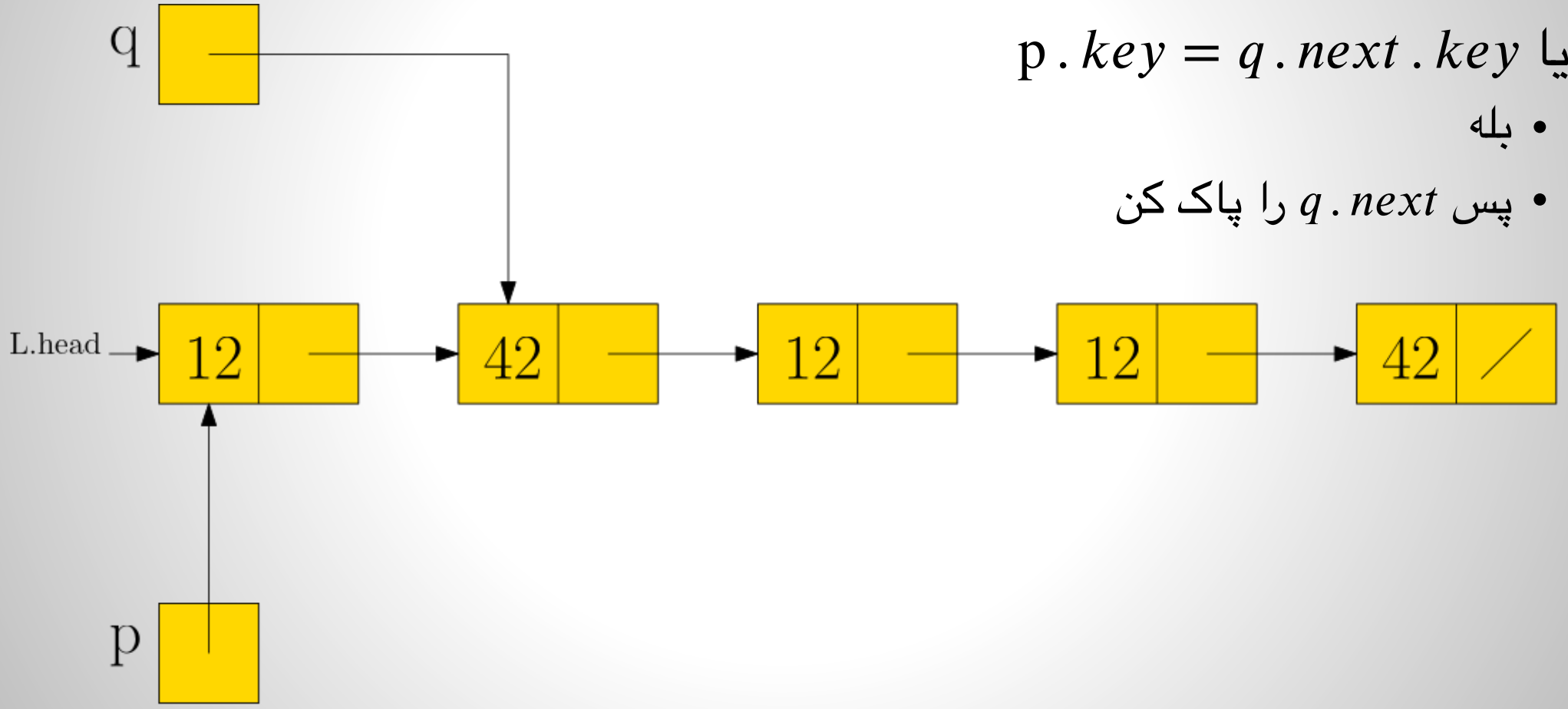
# مراحل

- آیا $p.key = q.next.key$
- خیر
- پس $q = q.next$



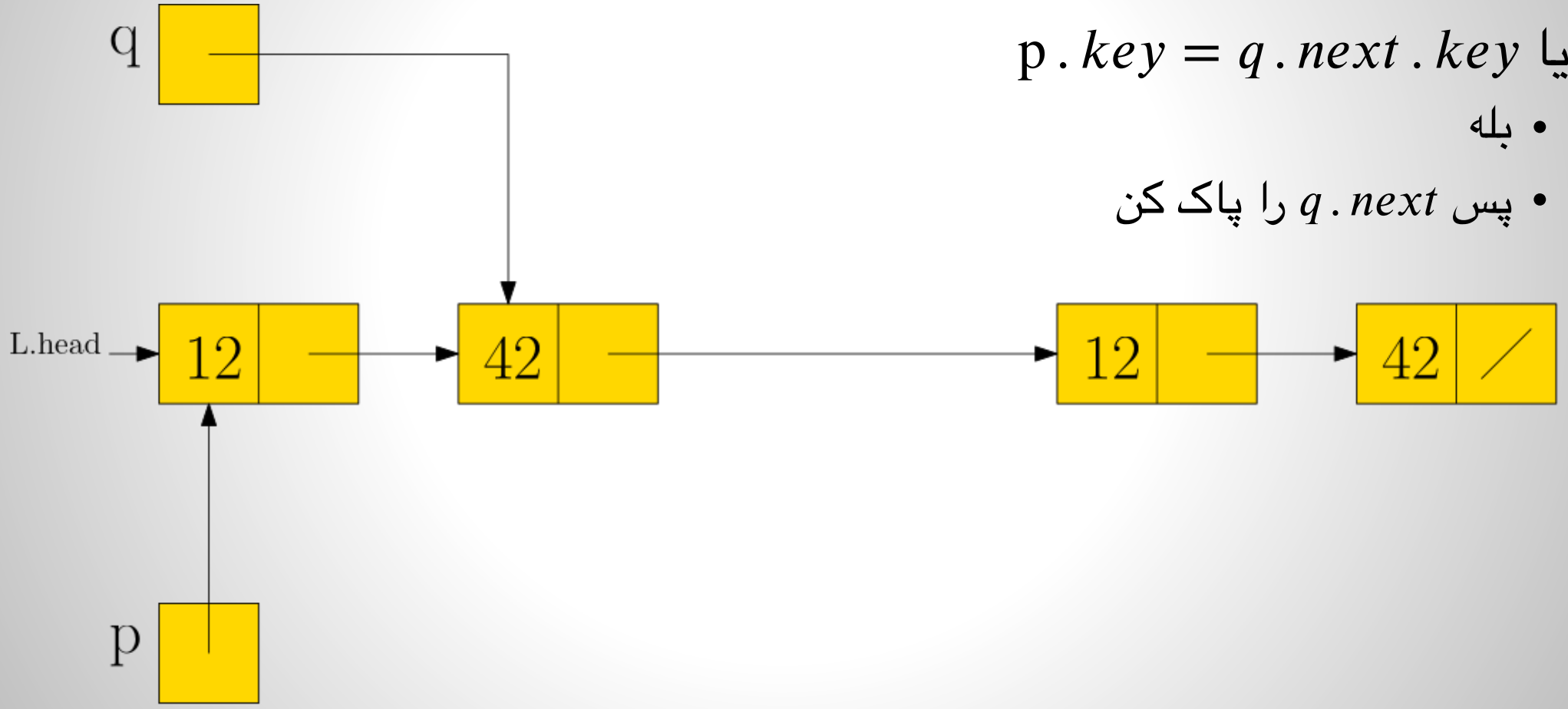
# مراحل

- آیا $p.key = q.next.key$
- بله
- پس $q.next$ را پاک کن



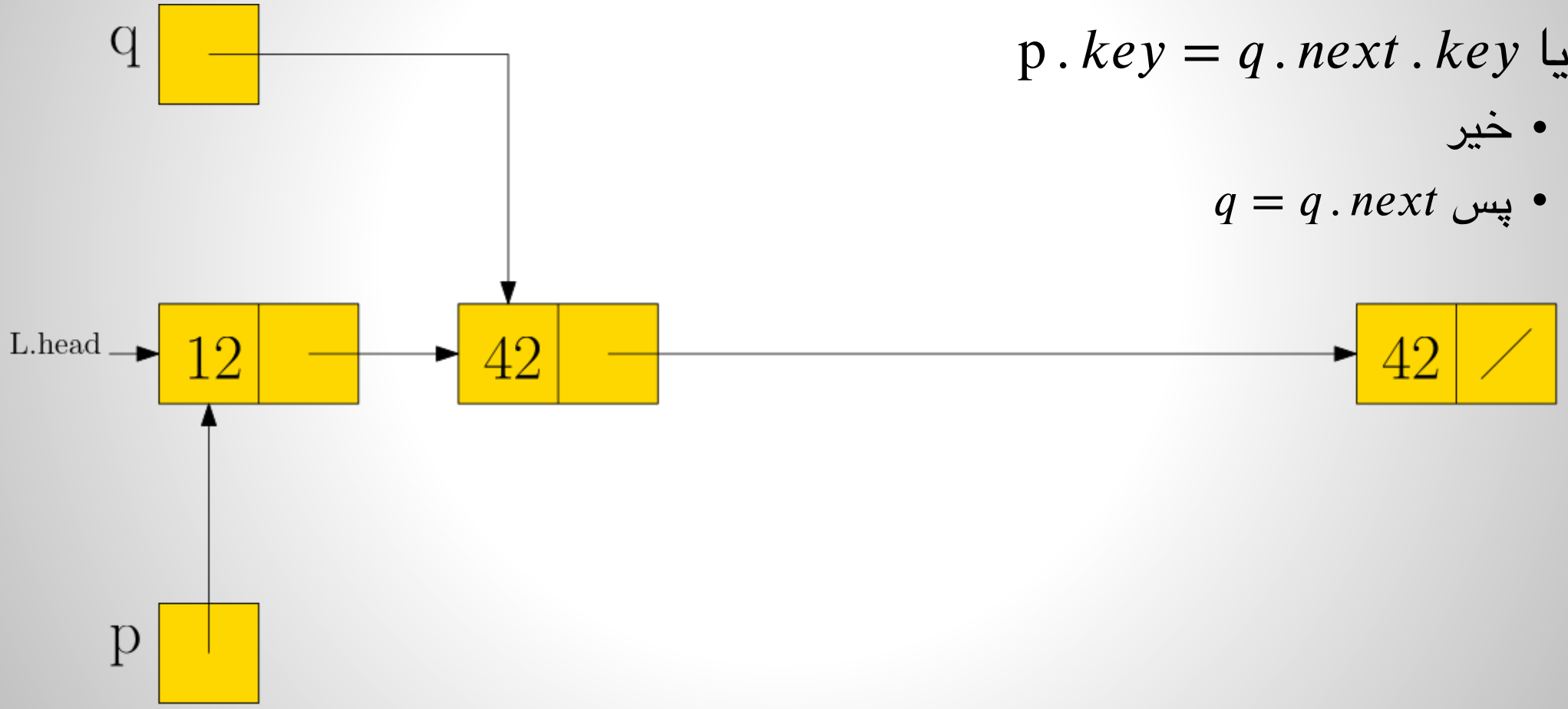
# مراحل

- آیا $p.key = q.next.key$
- بله
- پس $q.next$ را پاک کن



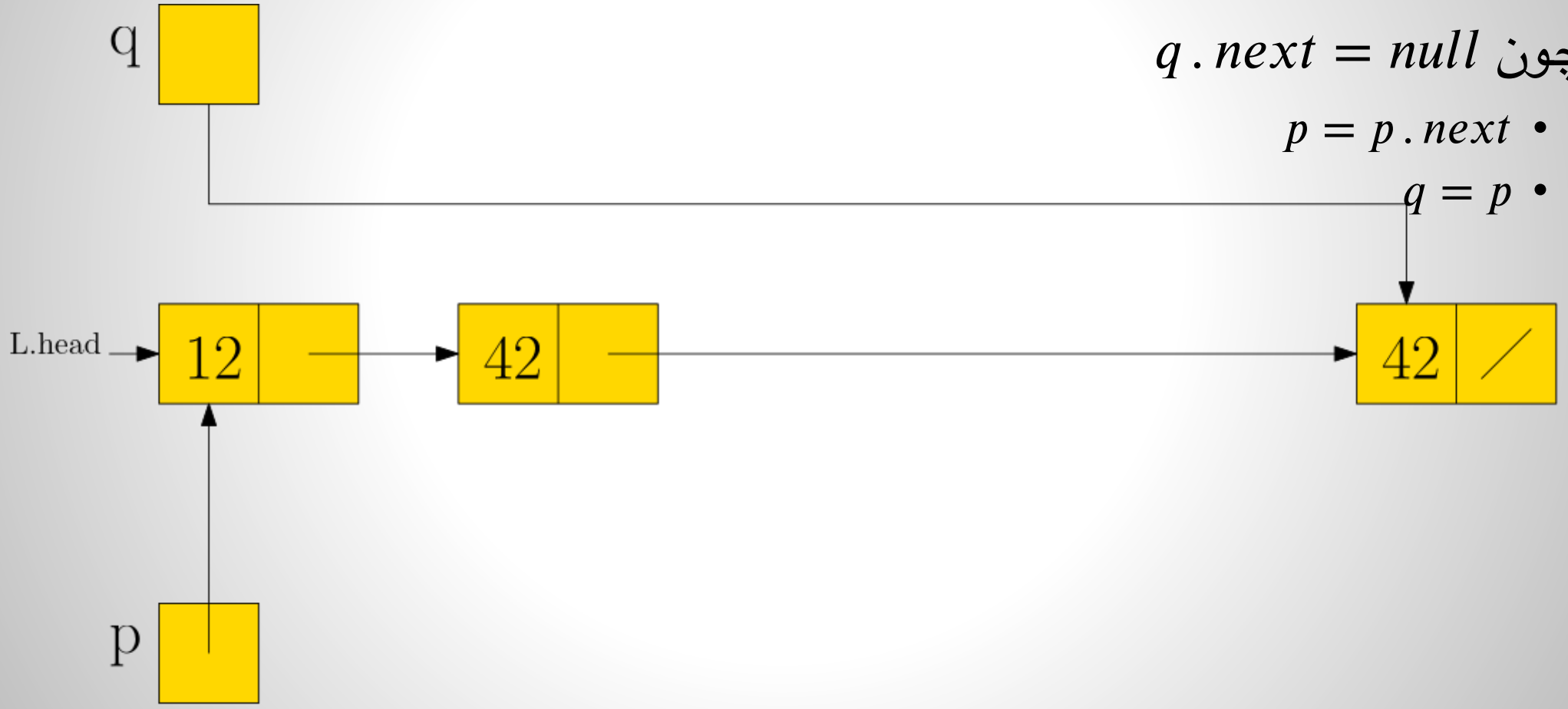
# مراحل

- آیا $p.key = q.next.key$
- خیر
- پس $q = q.next$



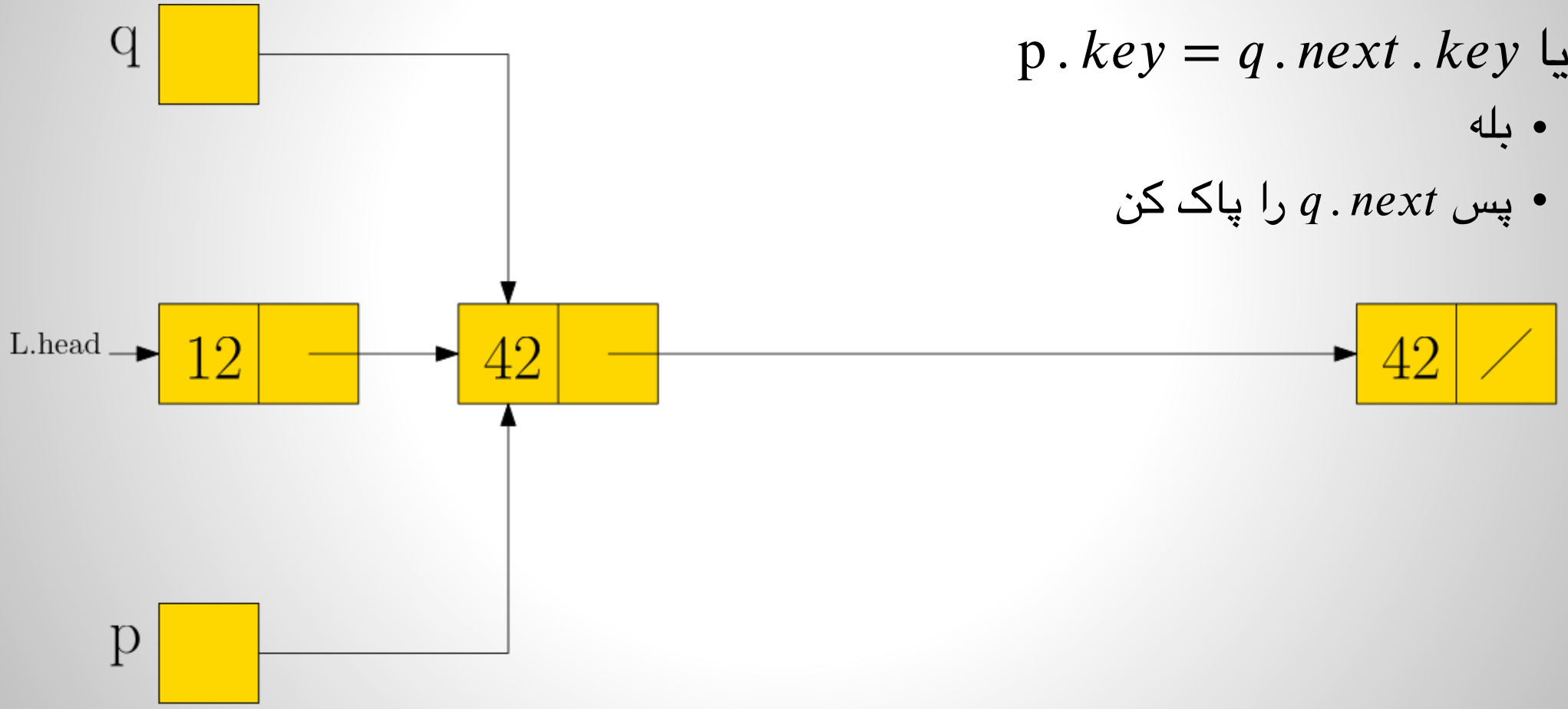
# مراحل

- چون $q.next = null$
- $p = p.next$
- $q = p$

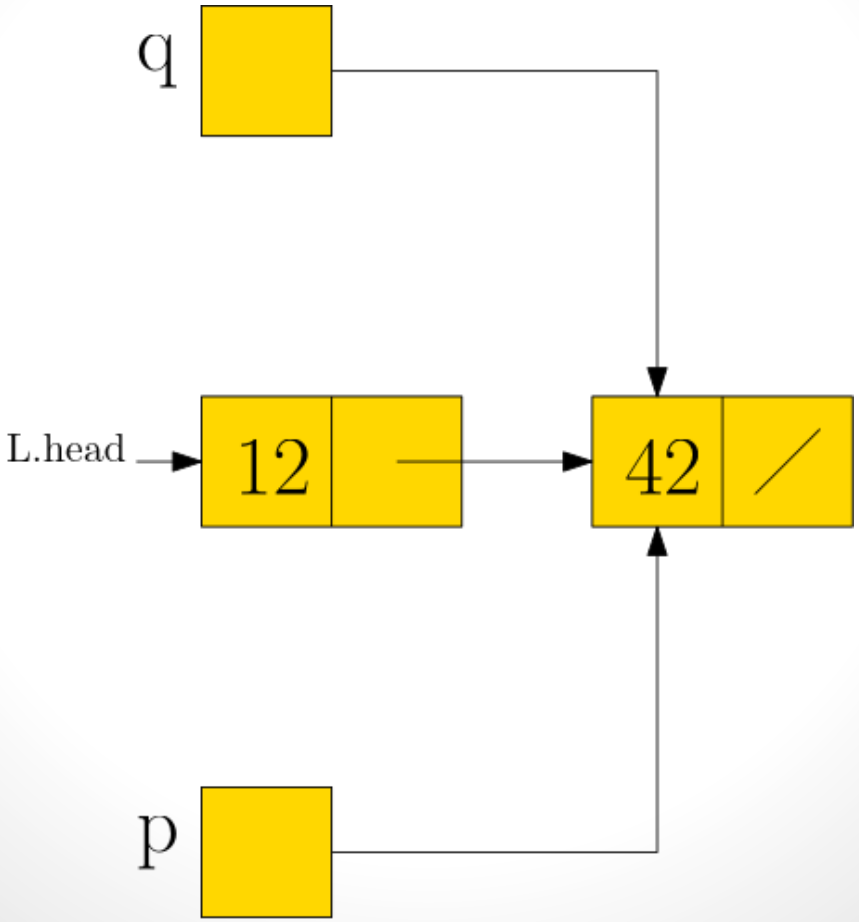


# مراحل

- آیا $p.key = q.next.key$
- بله
- پس $q.next$ را پاک کن



# مراحل



# مسئله ۲

- یک لیست را گرفته و ترتیب عناصر آن را وارونه کنید:

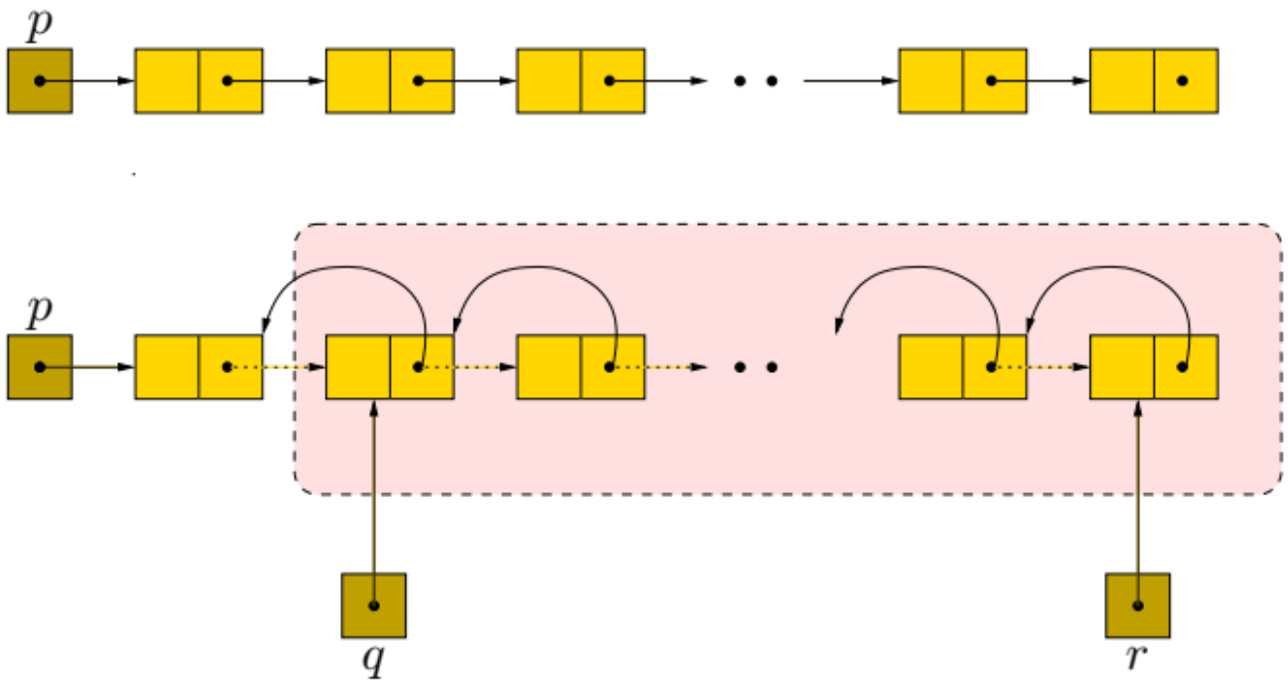
$$a \to b \to c$$
$$c \to b \to a$$

ورودی: یک لیست

خروجی: یک لیست با ترتیب وارونه

# روش اول به صورت بازگشتی

# وارون کردن یک لیست با تغییر اشاره‌گرها

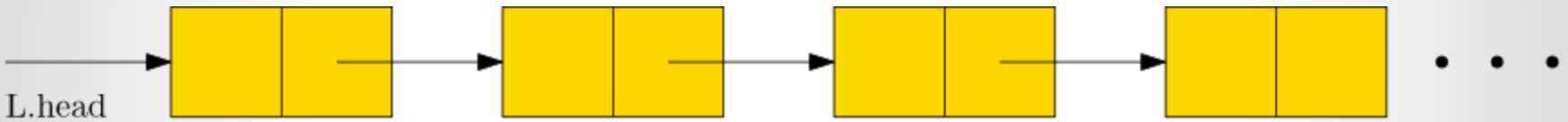


```
RECURSIVE-REVERSE(L, p)
    ▷ لیست L را از عنصر p به بعد وارون می‌کند و حاصل را برمی‌گرداند.
 1  if p = null  or  next[p] = null
 2    then return p
 3  q ← next[p]
 4  r ← RECURSIVE-REVERSE (L, q)
 5  next[q] ← p
 6  next[p] ← null
 7  return r
```

# روش دوم به صورت غیربازگشتی

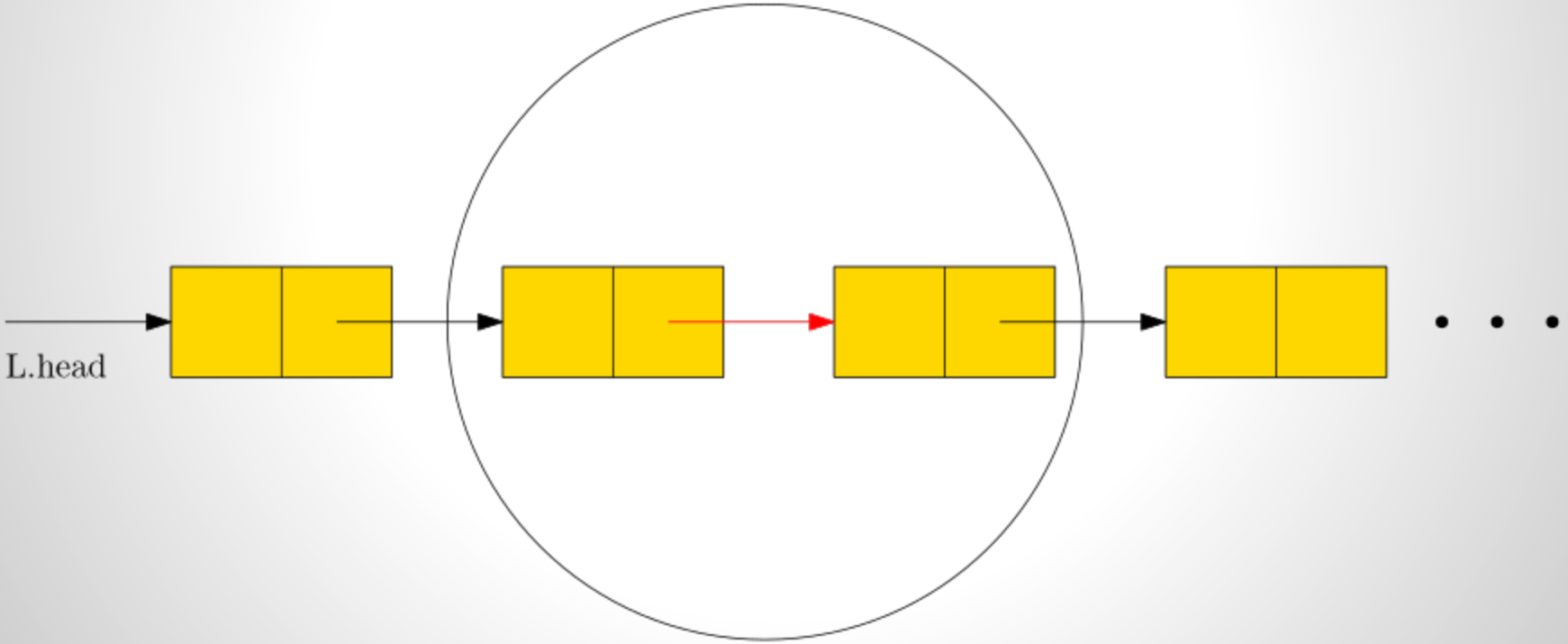
# ایده

- لیست را گرفته



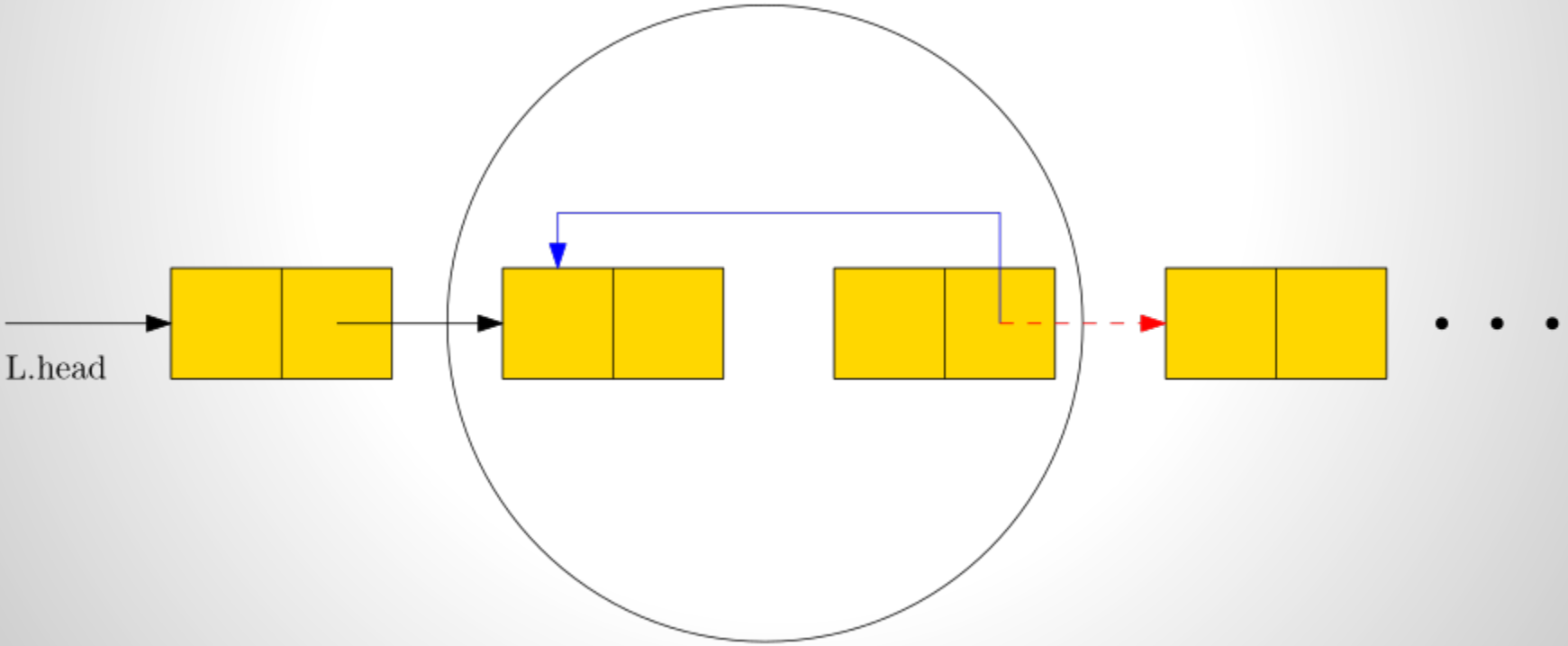
# ایده

- از ابتدا به ازای هر دو عنصر متوالی جهت لیست را جابه‌جا می‌کنیم.



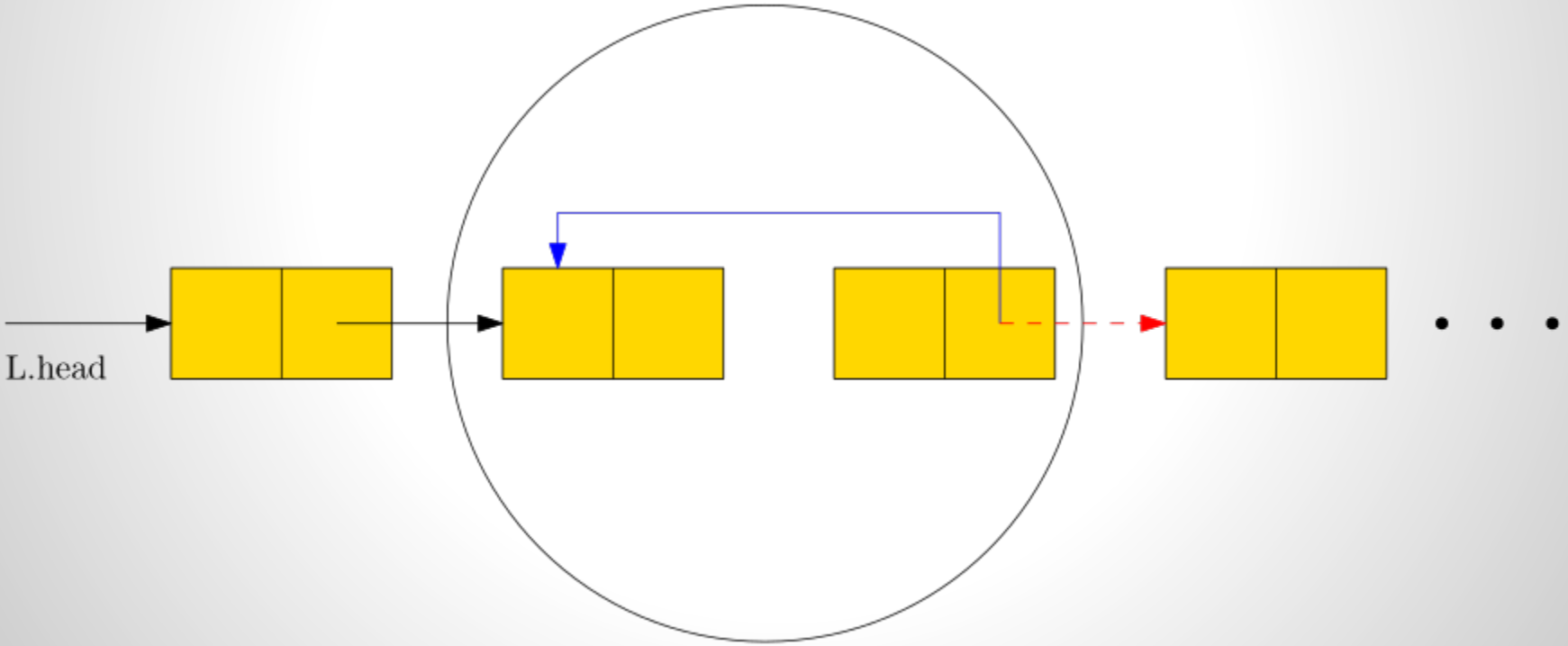
# ایده

- از ابتدا به ازای هر دو عنصر متوالی جهت لیست را جابه‌جا می‌کنیم.

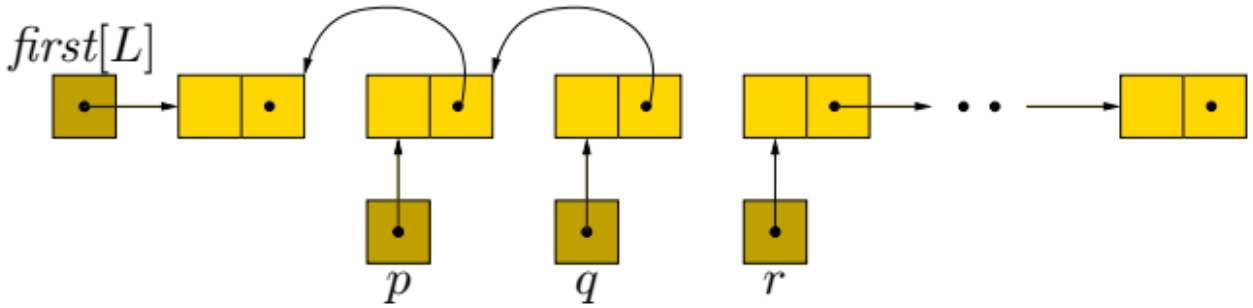


# ایده

- همیشه به حالات مرزی توجه کنید



# وارون کردن یک لیست به‌صورت غیربازگشتی

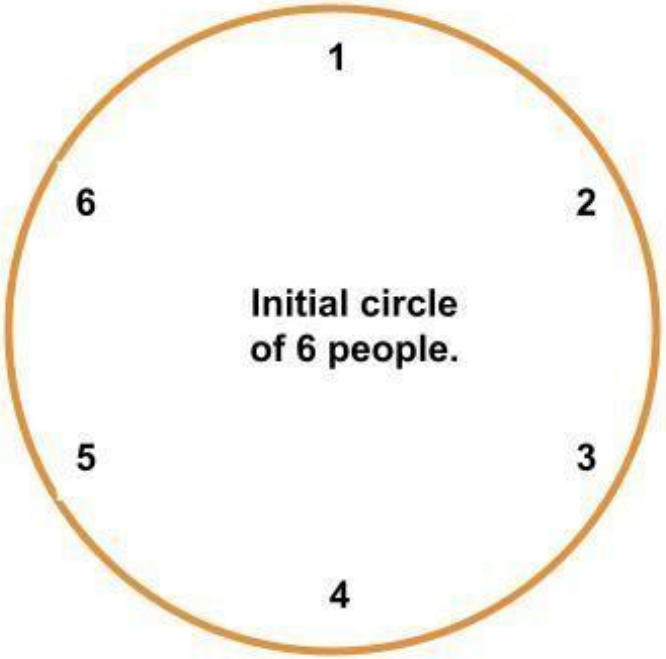


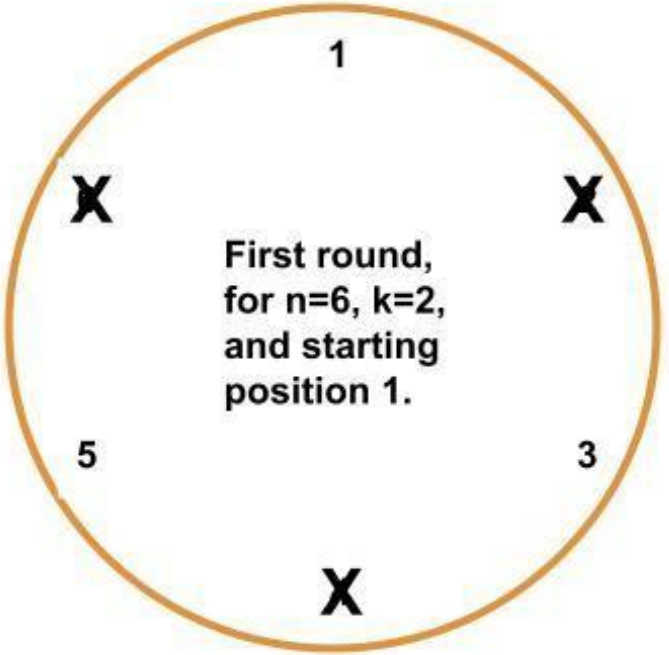
```
NR-Reverse(L)
 1  if Size(L) ≤ 1
 2    then return first[L]
 3  p ← null
 4  q ← First(L)
 5  r ← next[q]
 6  while r ≠ null
 7        do next[q] ← p
 8           p ← q
 9           q ← r
10           r ← next[r]
11  next[q] ← p
12  return q
```

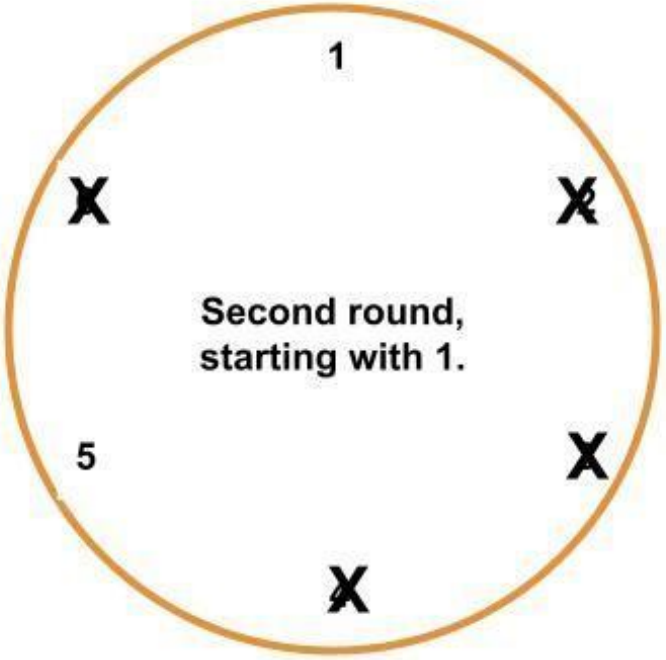
# مسئله ژوزفوس (Josephus)

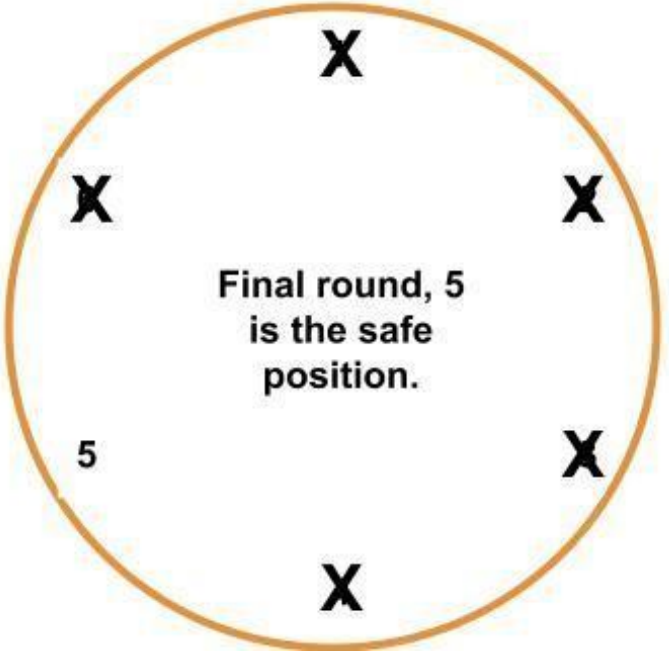
- تعداد $n$ نفر دور یه میز نشسته‌اند. از نفر اول به صورت ساعت‌گرد شروع می‌کنیم.
- در هر مرحله شخصی که نوبت اوست، نفر بعدی را از بازی حذف می‌کند، سپس نوبت را به نفر بعدی می‌دهد.

- اگر این بازی تا جایی ادامه پیدا کند که فقط یک نفر بماند، شماره او چند است.


1
2
3
4
5
6
Initial circle
of 6 people.


1
X
X
X
5
3
First round,
for n=6, k=2,
and starting
position 1.


1
X
X
X
X
5
Second round,
starting with 1.


X
X
X
X
X
5
Final round, 5
is the safe
position.

# حل مسئله ژوزفوس

- روش اول: به صورت ترکیبیاتی (که به آن نمی‌پردازیم)

جواب این مسئله $J(n)$ به‌صورت ریاضی قابل محاسبه است و می‌توان جواب را از رابطه‌ی بازگشتی زیر به‌دست آورد.

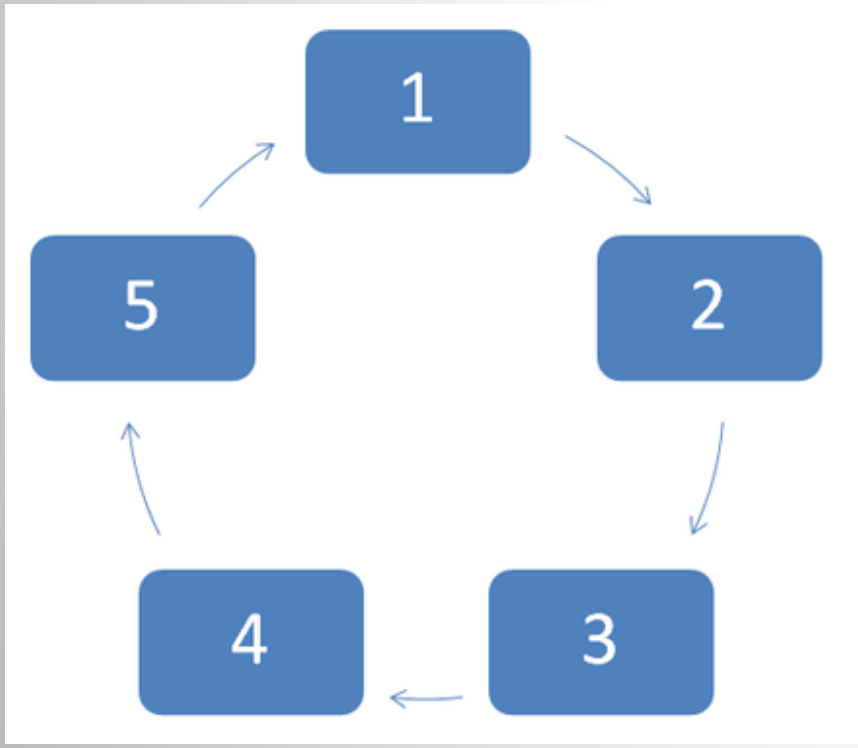
$$\begin{aligned} J(۱) &= ۱ \\ J(۲n) &= ۲J(n) - ۱, \text{ for } n \geq ۱, \\ J(۲n+۱) &= ۲J(n) + ۱ \quad \text{for } n \geq ۱. \end{aligned}$$

اگر $n$ را به‌صورت عدد دودویی بنویسیم و آن‌را یک بیت شیفت چپ دورانی دهیم $J(n)$ به‌دست می‌آید.

مثلاً برای $n = ۱۰۰ = (۱۱۰۰۱۰)_۲$، جواب $J(n) = (۱۰۰۱۰۱)_۲ = ۷۳$ است.

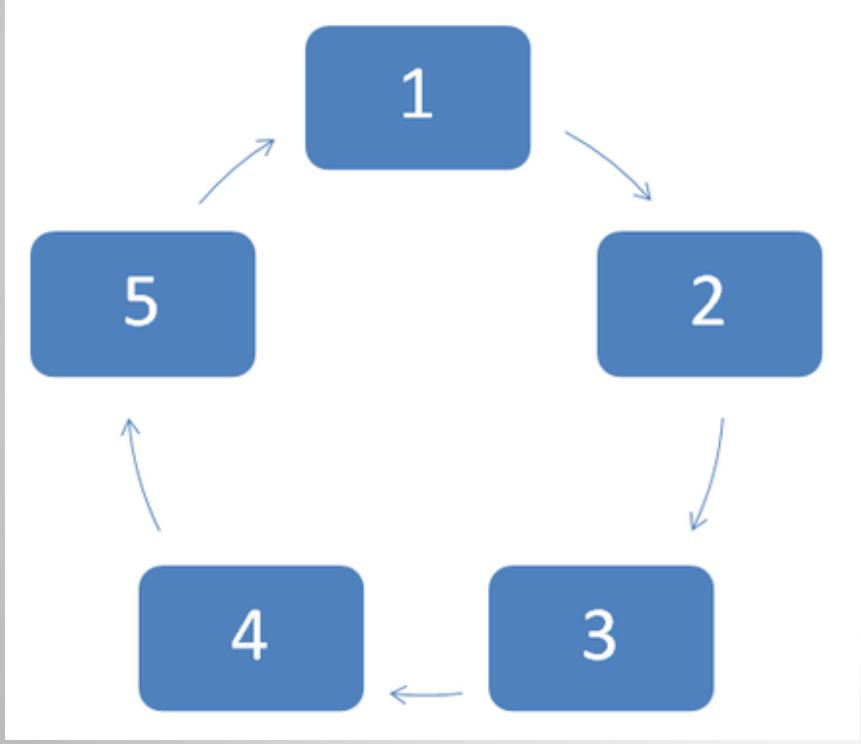
# حل مسئله ژوزفوس

- روش اول: به صورت ترکیبیاتی (که به آن نمی‌پردازیم)
- روش دوم: پیاده سازی توسط لیست حلقوی
- پیچیدگی زمانی: ؟



# حل مسئله ژوزفوس

- روش اول: به صورت ترکیبیاتی (که به آن نمی‌پردازیم)
- روش دوم: پیاده سازی توسط لیست حلقوی
- پیچیدگی زمانی: $O(n)$
  - چون در هر مرحله از محاسبه یک نفر حذف می‌شود



```
JOESEPHOUS(n)
      ▷ در ابتدا یک لیست پیوندی حلقوی با n گره ایجاد می‌کند
 1  CREATE(L)
 2  first[L] ← q ← ALLOCATE-NODE(1, null )
 3  p ← q
 4  for i ← 2 to n
 5     do next[p] ← ALLOCATE-NODE(i, next[p])
 6        p ← next[p]
 7  next[p] ← q;   size[L] ← n
      ▷ و حالا راه‌حل گُند مسئله‌ی ژوزفوس
 8  p ← FIRST(L)
 9  while next[p] ≠ p
10        do DELETE-AFTER(L, p)
11           p ← next[p]
12  PRINT element[p]
```

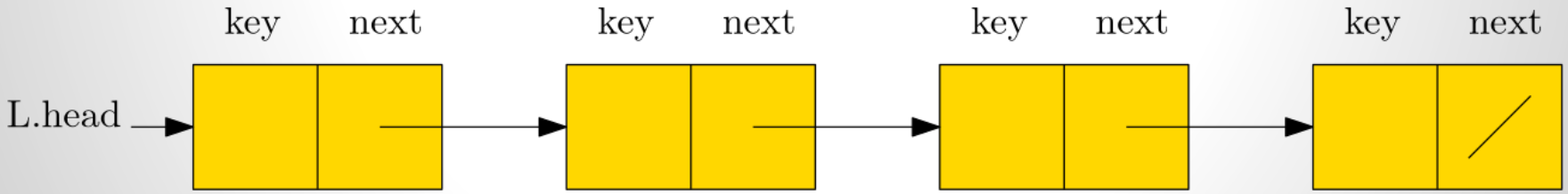
# پیاده سازی لیست توسط آرایه

- تعریف لیست

# تعریف (لیست)

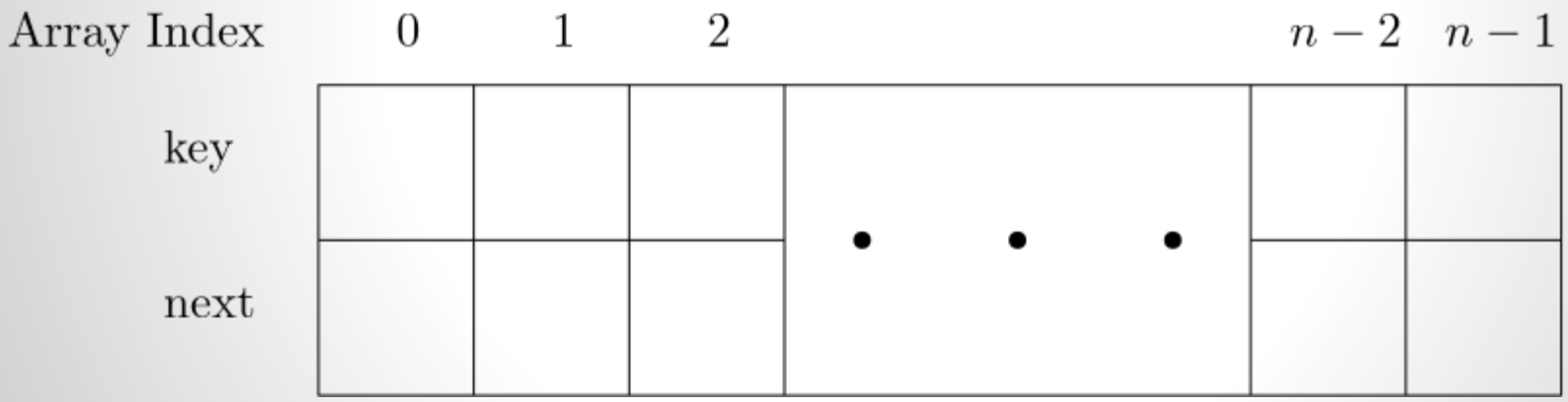
هر عنصر از لیست دو مولفه دارد

1. کلید: که مقدار مورد نظر مارا در خود نگه می‌دارد
2. بعدی: که اشاره‌گری به عنصر بعدی لیست است



# پیاده سازی لیست توسط آرایه

- تعریف لیست با یک آرایه $2 \times n$



# پیاده سازی لیست توسط آرایه

- تعریف لیست با یک آرایه $2 \times n$

- اعمال بر روی لیست

# اعمالی که بر روی لیست‌ها انجام می‌دهیم

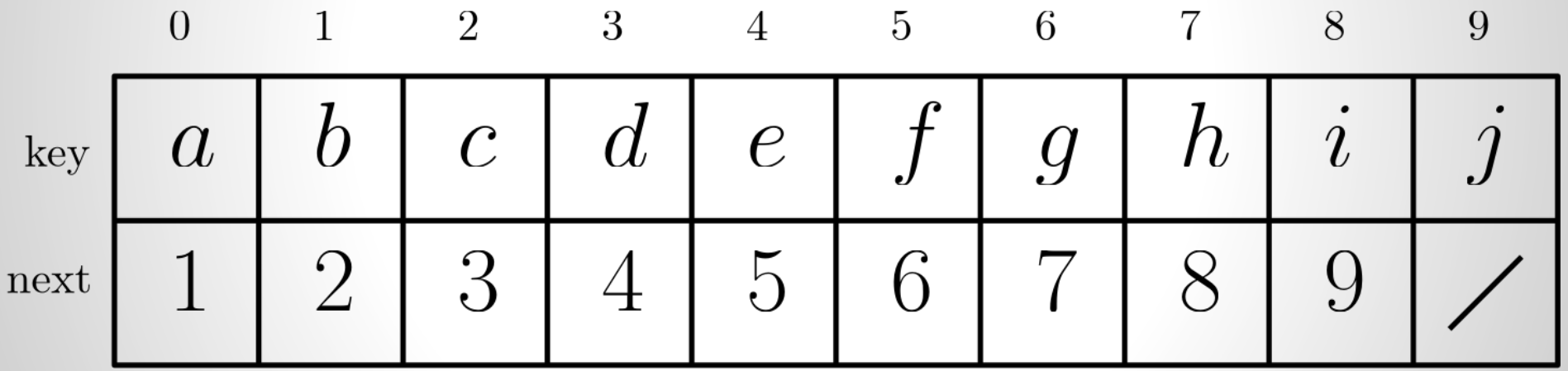
- $Create - List(L)$ایجاد یک لیست تهی :
- $Size(L)$را برمی‌گرداند $L$ تعداد عناصر لیست :
- $First(L)$را برمی‌گرداند $L$ عنصر اول :
- $isEmpty(L)$مشخص می‌کند که آیا لیست خالی است یا خیر :
- $Insert - First(L, x)$در ابتدای لیست $x$ درج عنصری با کلید :
- $Insert - After(L, x, n)$در لیست $n$ پس از عنصر $x$ درج عنصر : $L$
- $Delete - First(L)$را حذف می‌کند $L$ عنصر اول لیست :
- $Delete - After(L, n)$حذف می‌کند $L$ را از لیست $n$ عنصر پس از عنصر :

# اضافه کردن در لیست

- برای اضافه کردن یک عنصر جدید به آرایه به کجای لیست اضافه کنیم؟

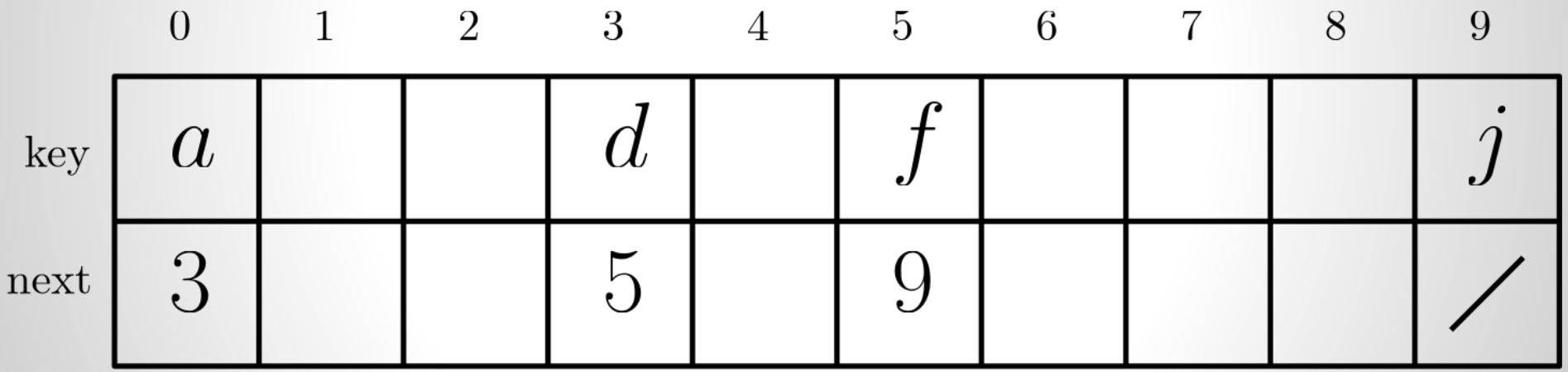
# اضافه کردن در لیست

- اگر همیشه به انتهای آرایه اضافه کنیم؟



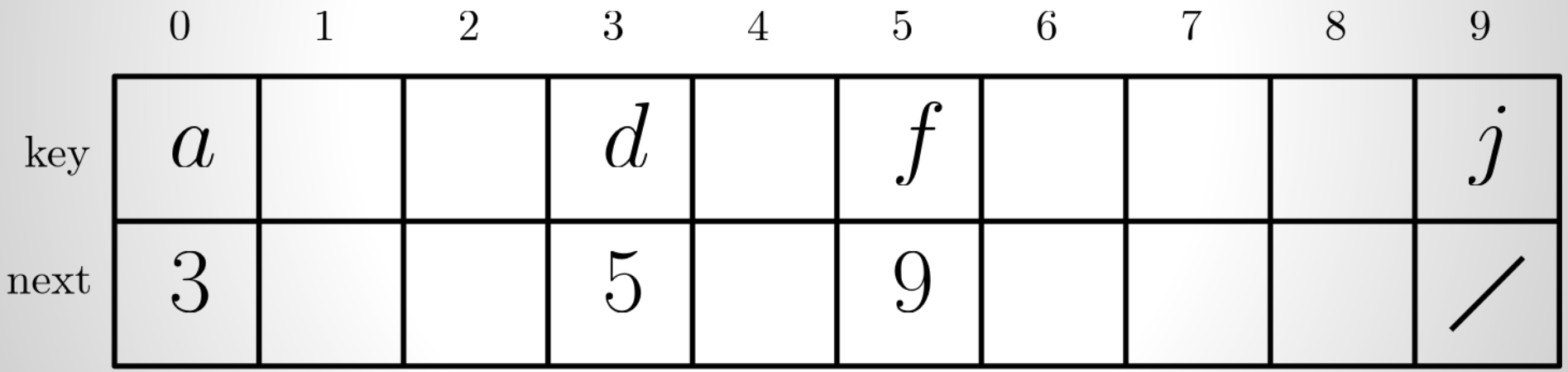
# اضافه کردن در لیست

- اگر چندین بار عمل درج و حذف رخ دهد.



# اضافه کردن در لیست

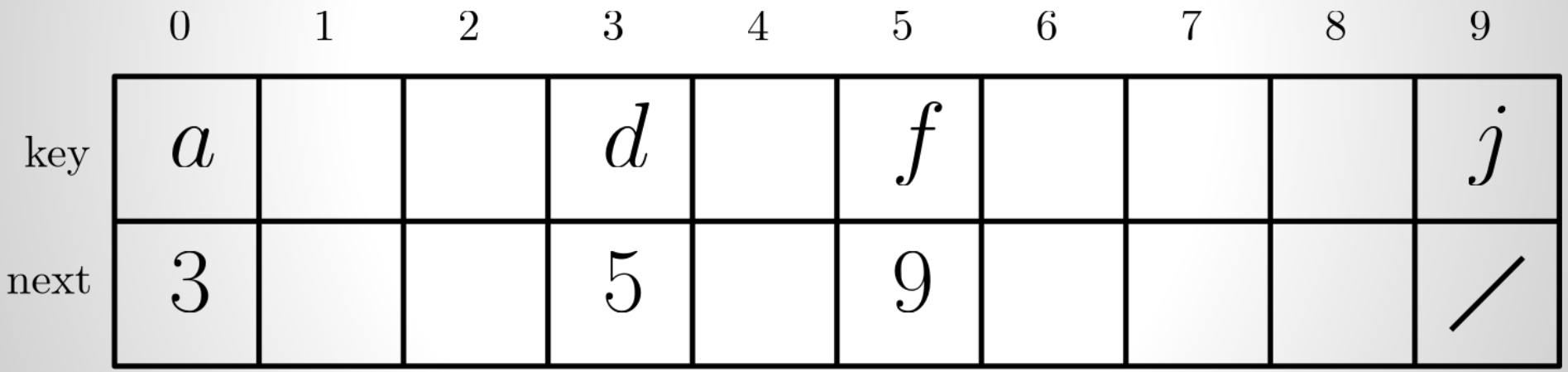
- حال برای اضافه کردن یک عنصر جدید به آرایه به کجای لیست اضافه کنیم؟




- هزینه ؟

# اضافه کردن در لیست

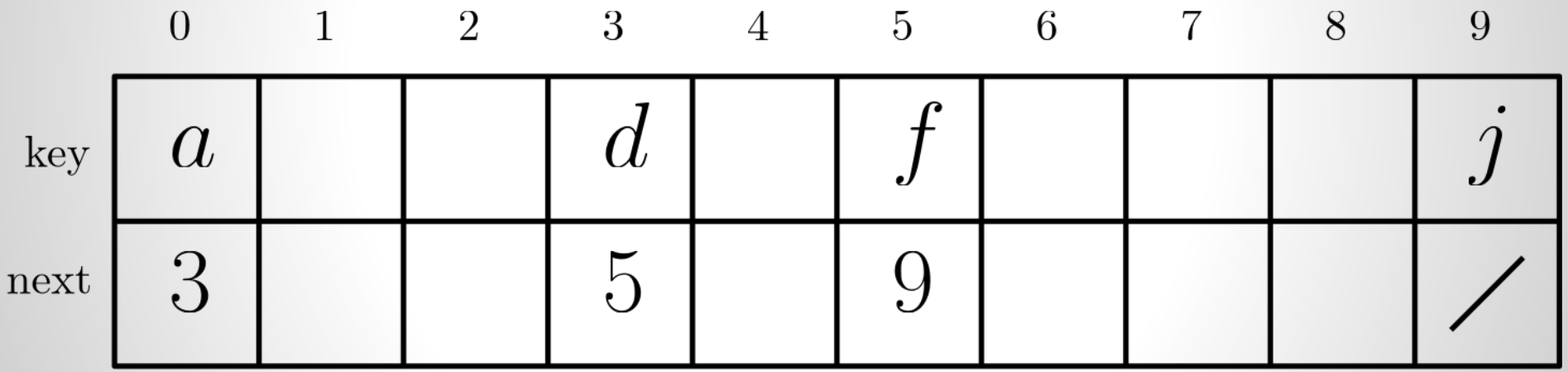
- حال برای اضافه کردن یک عنصر جدید به آرایه به کجای لیست اضافه کنیم؟




- هزینه $O(n)$

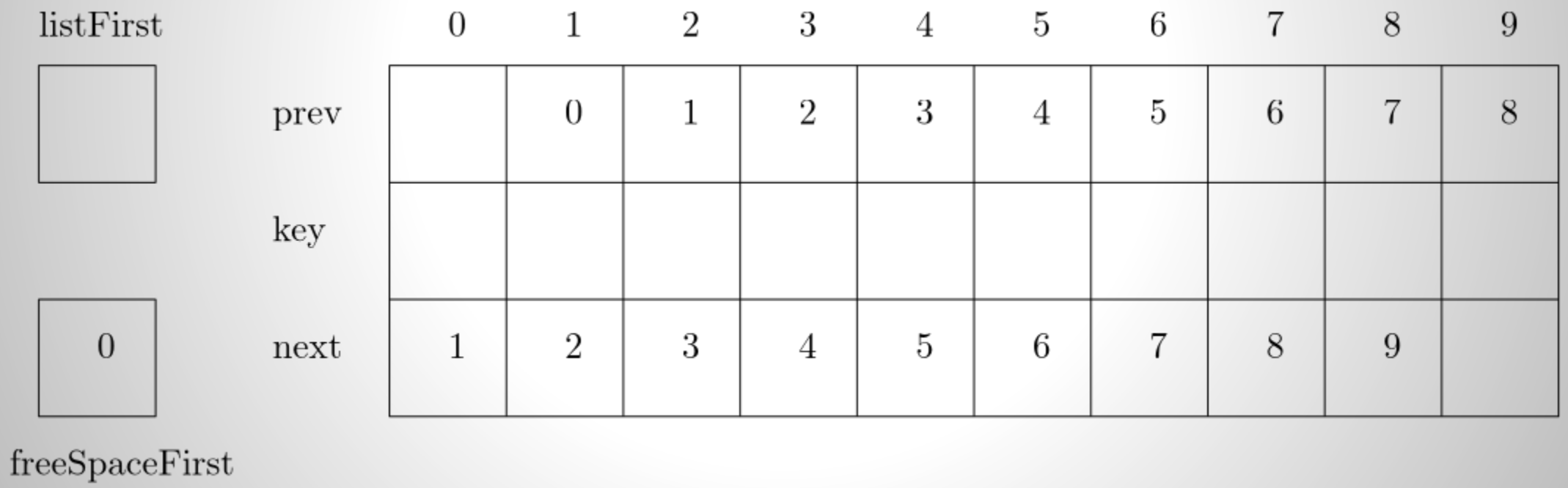
# اضافه کردن در لیست

- راه حل: نگه داشتن عناصر خالی در یک لیست



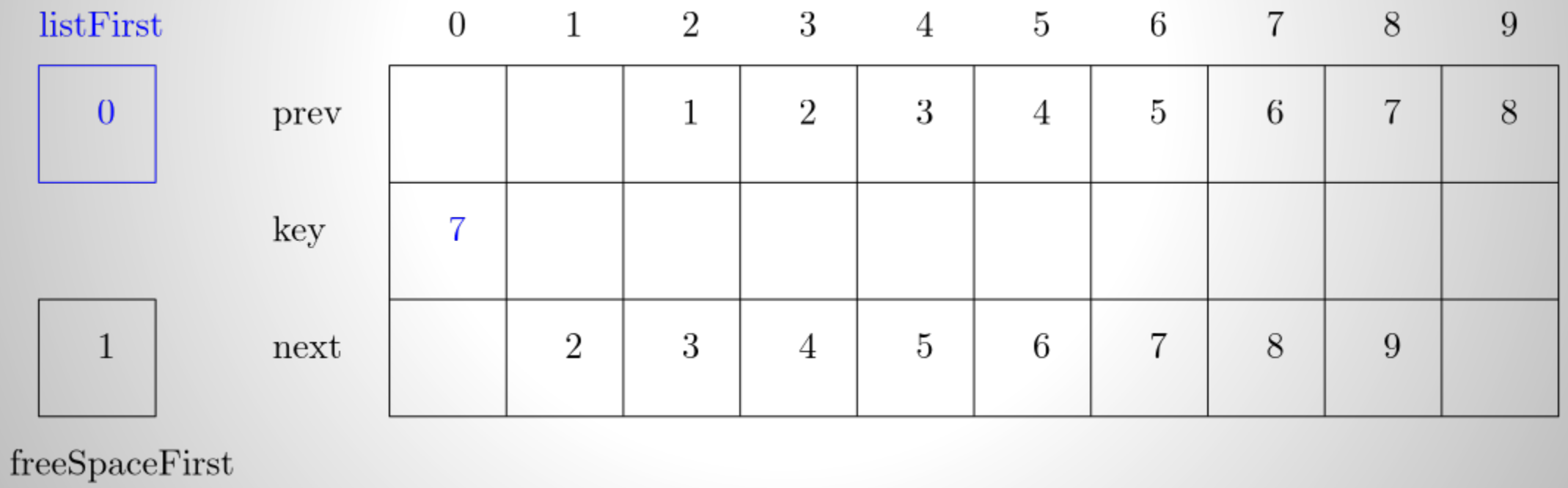
# اضافه کردن در لیست

- راه حل: نگه داشت خانه‌های خالی در یک لیست دو طرفه



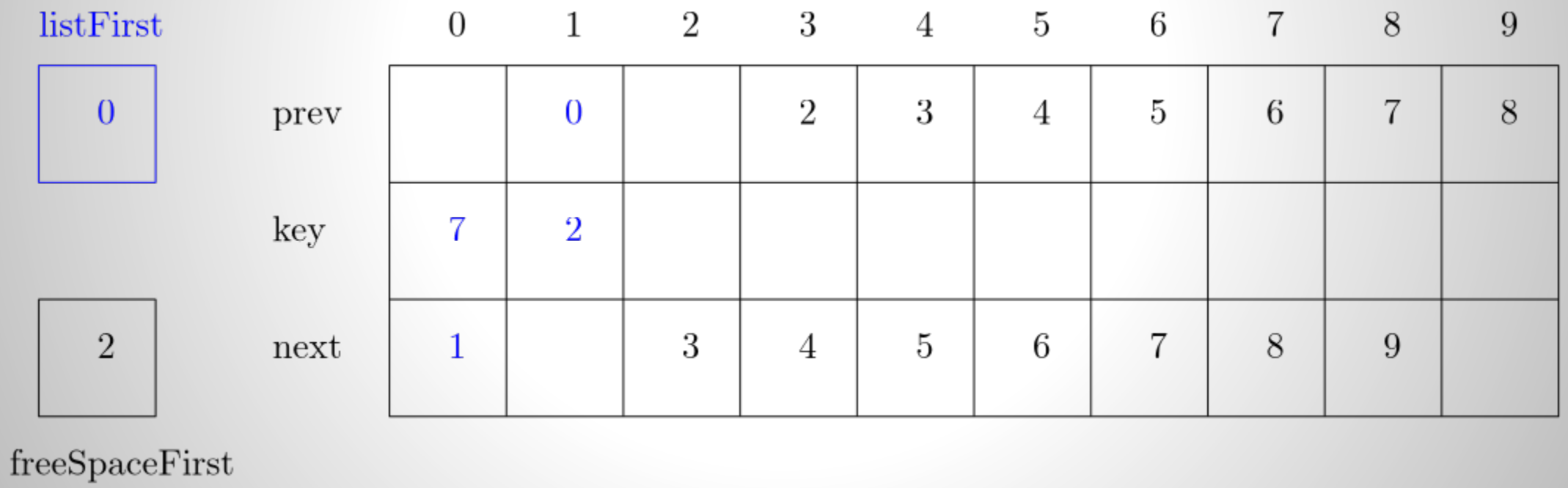
# اضافه کردن در لیست

- اضافه کردن عنصر ۷



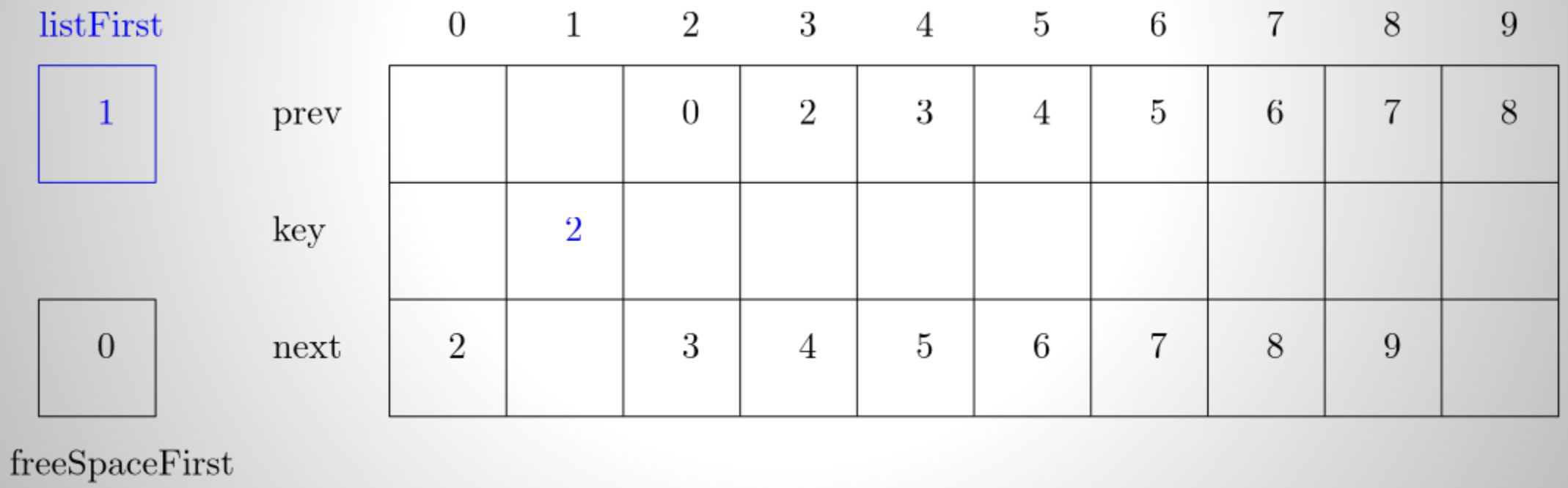
# اضافه کردن در لیست

- اضافه کردن عنصر ۲



# اضافه کردن در لیست

- حذف کردن عنصر ۷



# عمل درج در لیست

- هر بار عمل درج در لیست انجام می‌دهیم، باید عمل حذف از ابتدای لیست فضای خالی انجام دهیم.

- هر بار عمل حذف از لیست انجام می‌دهیم، باید آن را به ابتدای لیست فضای خالی اضافه کنیم.