ساختمان داده‌ها و الگوریتم

رشته علوم کامپیوتر

ناصر آیت

در مورد ساختمان داده

- ساختمان داده روشی است برای معرفی و دستکاری داده
- و کلیه برنامه های معرفی داده
- برای معرفی داده نیازمند یک الگوریتم میباشد.

در مورد ساختمان داده

- روش های طراحی الگوریتم نیازمند پیشرفت برنامه هایی است که برای نگهداری داده است.
- در علوم کامپیوتر مطالعه ساختمان داده ها مهم وضروری میبا شد.

Perequisites

- C++
- پیچیدگی
- Big oh , theta and omega notation

Sorting

ترتیب زیر را در نظر بگیرید:

a[0],a[1],…, a[n-1]

پس از مرتب سازی صعودی داریم:

a[0] <=a[1] <= ….<=a[n-1]

example:8,6,9,4,3 => 3,4,6,8,9

Sort metods

Insertion sort

Bubble sort

Selection sort

Count sort

Shaker sort

Shell sort

Heap sort

Merge sort

Quick sort

insert an elementاضافه کردن یک

لیست ترتیبی زیر را در نظر بگیرید:

input: 3, 6, 9, 14

عنصر ۵ را به لیست فوق اضافه کنید.

output: 3, 5, 6, 9, 14

Insert An Element

3, 6, 9, 14 insert 5

عدد ۵ را با آخرین عنصر لیست مقایسه کنید .

Shift 14 right to get 3, 6, 9, , 14

Shift 9 right to get 3, 6, , 9, 14

Shift 6 right to get 3, , 6, 9, 14

با اضافه کردن ۵ خروجی:

Output: 3, 5, 6, 9, 14

Insert An Element

```
// insert into a[0:i-1]
Int j;
For (j=i-1 ; j>=0 && t <a[ j] ;j--)
A[ j+1] = a[ j]
A[ j+1] = t ;
```

Insertion sort

۱. لیستی با سایز ۱ در نظر بگیرید.”اولین عنصر را داخل لیست قرار دهید.”

۲. عمل insertionرا تکرار کنید بطوریکه ترتیب داده ها حفظ شود

Insertion sort

Sort 7, 3, 5, 6, 1

Start with 7 and insert 3=> 3,7

Insert 5=>3, 5, 7

Insert 6=>3, 5, 6, 7

Insert 1=>1, 3, 5, 6, 7

Insertion sort

```
For (i=1 ; i<a.length ; i++)
{// insert a[i] into a[0:i-1]
     //code to insert comes here
}
```

Insertion sort

```
For (i=1 ; i<a.length ; i++)
{// insert a[i] into a[0:i-1]
     //code to insert comes here
int t =a[ j]
int j;
For ( j=i-1 ; j>=0 && t <a[ j] ; j--)
A[ j+1] = a[ j]
A[ j+1] = t ;
}
```

Complexity یا پیچیدگی

- پیچیدگی مکانی /حافظه ای
- پیچیدگی زمانی

۱. شمارش یک عملگر خاص
۲. شمارش تعداد مراحل
۳. پیچیدگی Asymptotic

Compration count
شمارش مقایسه ای

```
For (i=1 ; i<a.length ; i++)
{// insert a[i] into a[0:i-1]
     //code to insert comes here
int t =a[ j]
int j;
For ( j=i-1 ; j>=0 && t <a[ j] ; j--)
A[ j+1] = a[ j]
A[ j+1] = t ;
}
```

Compration count

- یک نمونه کاراکتری از n را در نظر بگیرید، که در آن n طول لیستی باشد که می خواهیم روی آن Insertion sortرا انجام دهیم.
- و تعداد توابع این نمونه کاراکتری را بشمارید.؟؟؟؟

Determine count as a function of this instance characteristic.???

Compration count

```
For (j=i-1 ; j>=0 && t <a[ j] ;j--)
A[ j+1] = a[ j];
```

- چند مقایسه انجام شده است ؟

Compration count

```
For (j=i-1 ; j>=0 && t <a[ j] ;j--)
A[ j+1] = a[ j];
```

- شمارش تعداد مقایسات وابسته به a[] و t با توجه به i

Compration count

Worst-case count=maximum count

Best –case count=minimum count

Avarage count

Worst-case Compration count

For (j=i-1 ; j>=0 && t <a[j] ;j--)

A[j+1] = a[j];

A=[1,2,3,4] and t=0 =>4 compares

A=[1,2,3,..,i] and t=0 =>i compares

Worst-case Compration count

```
for(int i=1; i< n ; i++)
For (j=i-1 ; j>=0 && t <a[ j] ;j--)
A[ j+1] = a[ j];
```

تعداد کل مقایسات :

Total comprase =1+2+3+…+(n-1)

=(n-1)n/2

Step count

- یک مرحله از محاسبات وابسته است به مقادیر n
- برای مثال :

10 add , 100 subtracts,1000 multiplies

فقط یک step محسوب میشود .

- وبه این مفهوم نمی باشد که با افزایش nیک مرحله نیز به تعداد step اضافه شود.

Step count

	s/e
For (i=1 ; i<a.length ; i++)	1
{// insert a[i] into a[0:i-1]	0
//code to insert comes here	1
int t =a[j]	0
int j;	1
For (j=i-1 ; j>=0 && t <a[j] ; j--)	1
A[j+1] = a[j]	1
A[j+1] = t ;	0
}	

Step count

- s/e همیشه ۰ یا یک نمی باشد
- X=mymath.sum(a,n)

Where n is the instance characteristic has a s/e count of n

Step count

	s/e	step
For (i=1 ; i<a.length ; i++)	1	
{// insert a[i] into a[0:i-1]	0	
//code to insert comes here	1	
int t =a[j]	0	
int j;	1	i+1
For (j=i-1 ; j>=0 && t <a[j] ; j--)	1	i
A[j+1] = a[j]	1	
A[j+1] = t ;	0	
}		

Step count

For (int i=1;; i<a.length; i++)
{2i+3}
Step count for
For (int i=1; i<a.length;; i++)
Is n
Step count for body of for loop is
2(1+2+3+…+n-1)+3(n-1)
=(n-1)n +3(n-1)
=(n-1)(n+3)

محاسبه پیچیدگی در مرتب سازی درجی

$O(n^2)$

به چه معنی می باشد؟

محاسبه پیچیدگی در مرتب سازی درجی

- بدترین حالت:$\Theta(n^2)$
- بهترین حالت: $\Theta(n)$
- بنابراین انتظار میرود بدترین حالت زمانی است که n دوبل میشود(تکرار میشود).

محاسبه پیچیدگی در مرتب سازی درجی

- آیا **$O(n^2)$** خیلی زیاد است؟
- الگوریتم تجربی (practical) چه میباشد؟

Faster Computer Vs Better Algorithm

- پیشرفت الگوریتم ها مفید تر از پیشرفت سخت افزار است.

Data Structure ساختمان داده

- انواع داده :
- داده هایی که در یک مجموعه قرار می گیرند(در ارتباط).

Ex: integer= {0, +1,-1,+2,-2,….}

days of week = {S,M,T,W,TH,SA}

Data object

- داده هایی که نامربوط با هم هستند.

Example:

MyDataObject={apple, chair, 2,5,red,green ,jack}

Data Structure

Data oject +

روابطی که وجود دارد برای مقایسه میان عناصر

Ex:369<370

280+4=284

Data Structure

- میان عناصری که مقایسه میشوند در یک مثال:

369

3 is more signification than 6

3 is immediately to the left of 6

9 is immediately to the right of 6

Data Structure

- روابط خاص معمولا توسط عملگر های خاص روی چندین نمونه داده ایجاد می شود عبارتند از:

Add,subtract, predecessor,multiply

جمع ، تفریق ، ضرب

Linear (or Ordered) lists

Instannces are of the form

(e0,e1,e2,….,en-1)

Where ei denodes a list element n>=0 is finite

List size is n

Linear (or Ordered) lists

L= (e0,e1,e2,e3,….,en)

روابط زیر بر قرار است:

e0 : عنصر جلوی لیست میباشد. ”zeroth element“

en-1: عنصر آخرلیست میباشد.”last elements“

ei+1 : دقیقا بعدازei قرار می گیرد.

مثالهایی از لیست های خطی:

Student in COP3530=(jack,jill, Abe ,Henry,Mary ,…,judy)

Exams in COP3530=(exam1, exam 2, exam3)

Days of Week=(S,M,T,W,TH,SA)

Months=(Jan,Feb ,Mar ,Apr,…,Nov,Dec)

Linear list Oprations-size()

- اندازه گیری سایز لیست

Example:

L=(a,b,c,d,e)

Size=5

Linear list Oprations-get(the index)

- یک عنصر را میگیرد و اندیس آن را بعنوان خروجی می دهد.

Example:

L=(a,b,c,d,e)

Get(0)=a

Get(2)=c

Get(4)=e

Get(-1)=error

Get(9)=error

Linear list Oprations-indexof (the element)

اندیس هر عنصر را برمی گرداند.

L= (a,b,d,b,a)

Index of(d)=2

Index of(a)=0

Index of(z)=-1

Linear list Oprations-remove(the index)

حذ ف را انجام داده و محتوای اندیس مورد نظر را برمی گرداند.

L=(a,b,c,d,e,f,g)

Remove(2) returns c

and L become (a,b,d,e,f,g)

index of d,e,f and g decrease by 1

Linear list Oprations-remove(the index)

حذ ف را انجام داده و محتوای اندیس مورد نظر را برمی گرداند.

L=(a,b,c,d,e,f)

Remove(-1)=>error

Remove(20)=>error

Data structure specification

Language independet

Abstract Data Type

Java

Abstaract class

Liner List Abstaract Data Type

AbstractData Type Linear List

{

- instances

Ordered finit collection of zero or more elements

- عملگر ها

Empty( ):

نتیجه درست را بر می گرداند اگر و فقط اگر لیست خالی باشد ،در غیرنتیجه false : میبا شد.

Size( ):

اندازه لیست را بر می گرداند.بعبارتی تعداد عناصر داخل لیست را بر می گرداند.

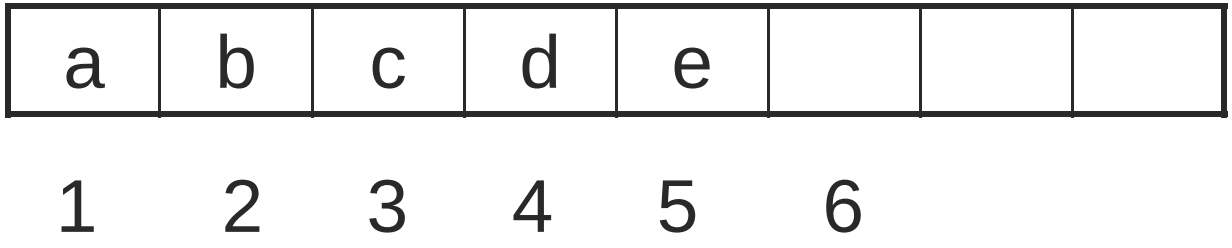
Data Representation Methods

Array ---chapter 5

Linked---chapter 6

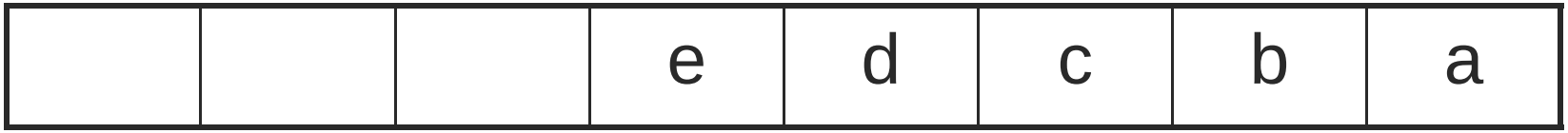
Simulated

Linear List Array Representation

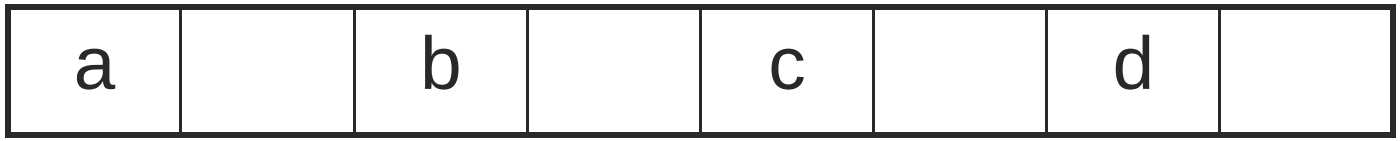


L=(a,b,c,d,e)

Right to Left Mapping

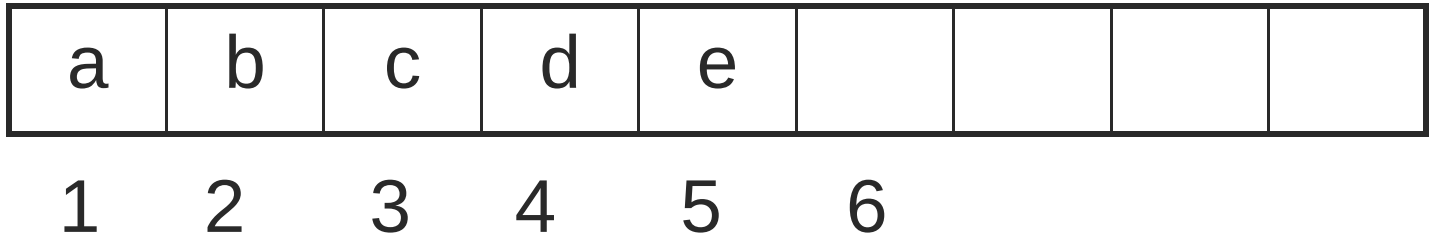


Mapping That Skip Every Other position



Wrap Around Mapping

Represention Used In Text



Add/Remove An Element _1

Size=5

Add/Remove An Element_2

Add (l, g)

size-=6

a	g	b	c	d	e			

Length of Array element[]

چون نمی دانیم چه تعداد عنصر در لیست وجود خواهد داشت ،بنابراین باید یک مقدار اولیه برای آن فرض کرد.و سپس بر حسب نیاز آن را افزایش داد.

Liner List Abstaract Data Type

Get (index):

خروجی آن اندیس عنصر میباشد بر طبق جایگاه آن ودرصورتی مقدار(۱-) را برمی گرداند که عنصر مورد نظر در لیست نباشد.

Remove (index):

عنصر را حذف کرده و محتوای عنصر را برمی گرداند.

Add (index, x):

عنصر x را در indexداده شده اضافه کرده و پس از آن شماره اندیس ما بقی عناصر از موقعیت جاری یک واحد افزایش میابد.

Output( ):

خروجی لیست است که از چپ به راست مرتب می شود.

Linear List As Java abstract Class

```
Public abstractclass Linear ListAsAbstractClass
{
Public abstarct Boolean isEmpty();
Public abstarct int size();
Public abstarct object get(int index);
Public abstarct int indexOf(Object the element);
Public abstarct object remove(int index);
Public abstarct void add(int index, Object the element);
Public abstarct string to sorting();
}
```

Linked Representation

عناصر لیست در حافظه با ترتیبی دلخواه نگهداری می شوند.

- explicit information (called a link)

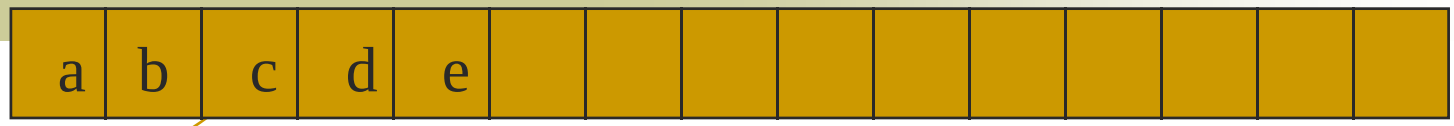
اطلاعات صریح که لینک نامیده می شوند

برای رفتن از یک عنصر به عنصر دیگر استفاده میشوند

Memory Layout

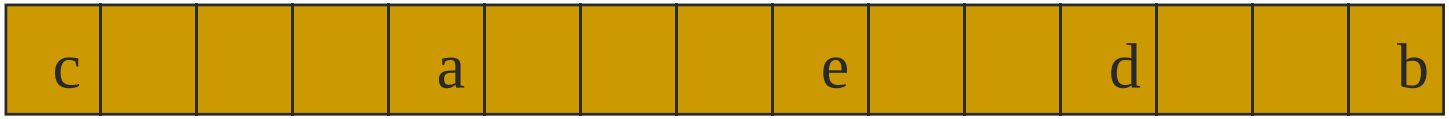
برای نمایش لیست ها از آرایه ها استفاده می شود.

L = (a,b,c,d,e)

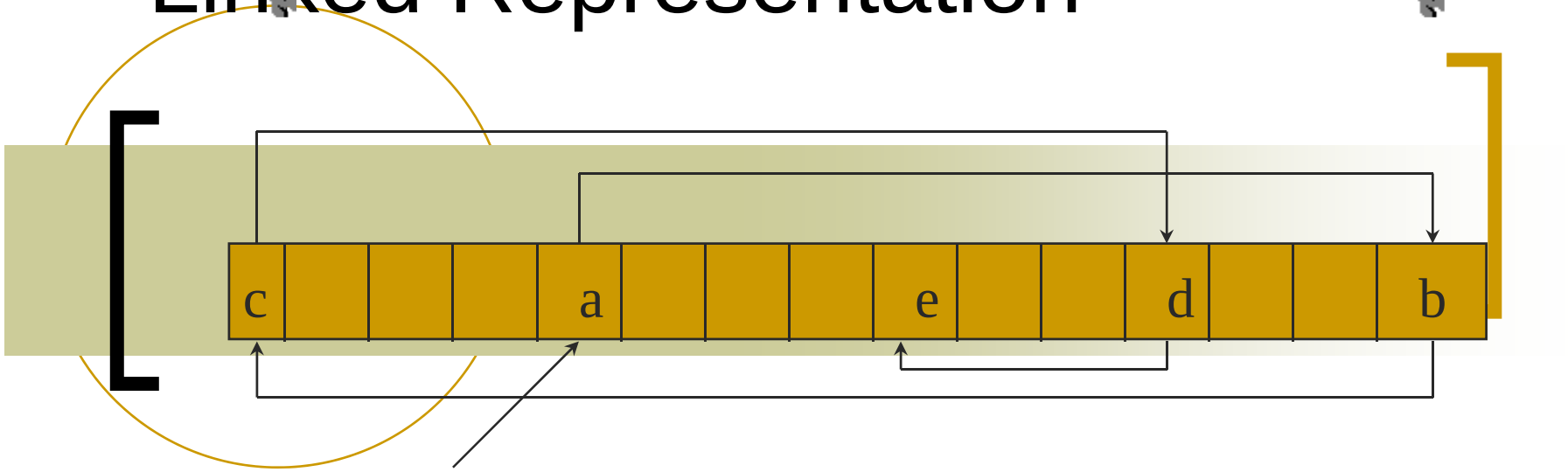


هرلینک از یک جدول دلخواه استفاده می کند.

A linked representation uses an arbitrary layout.



Linked Representation

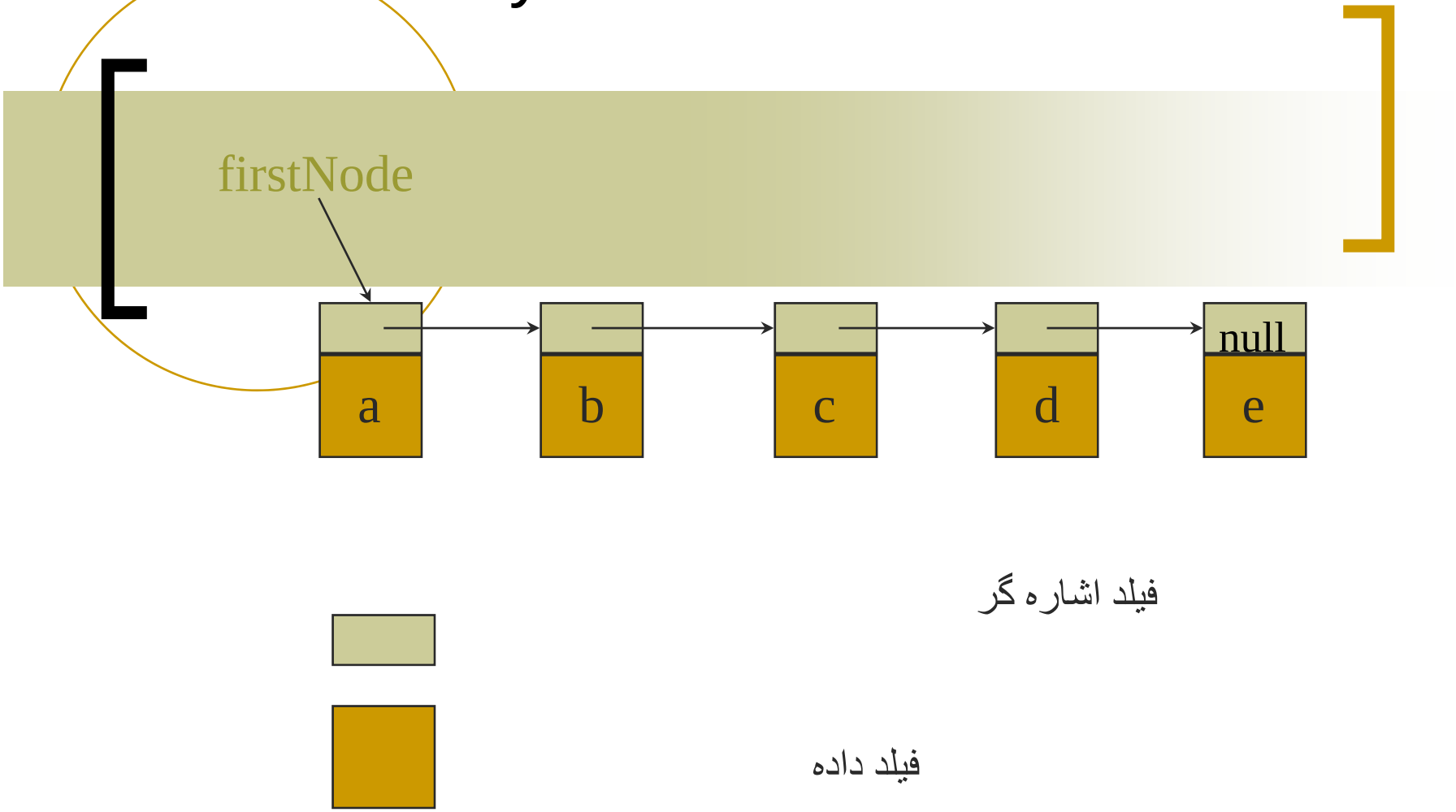


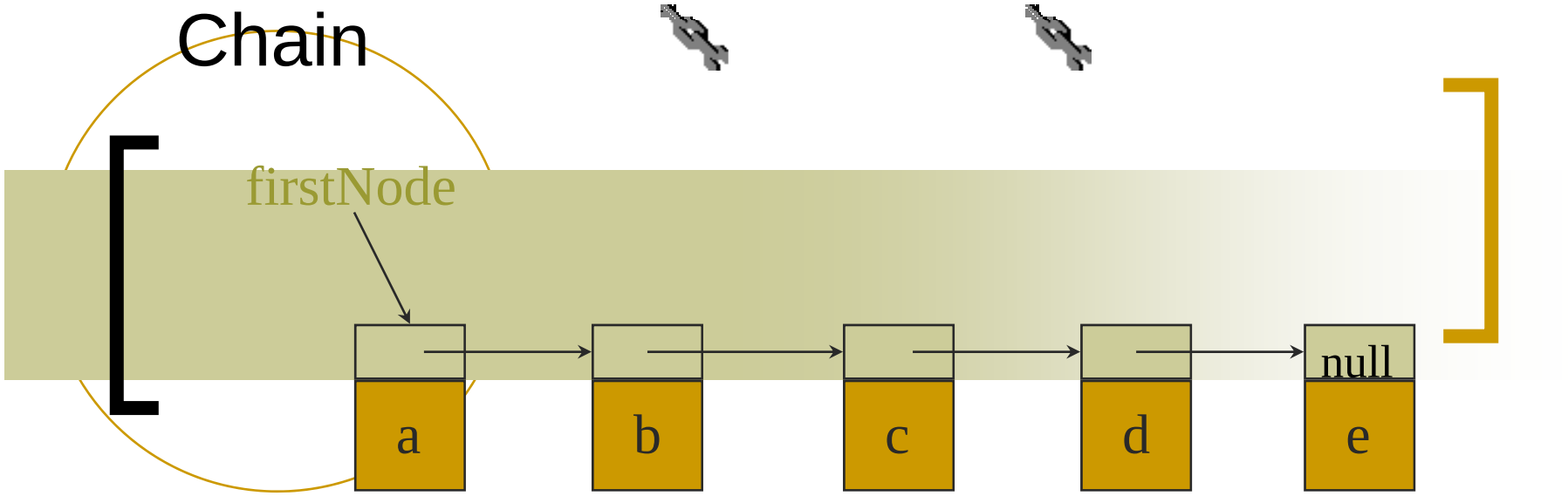
تهی می باشد. (e) اشاره گر عنصر

firstNode:

برای اشاره به عنصر شروع استفاده میشود.

Normal Way To Draw A Linked List





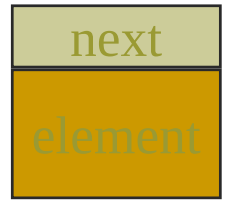
دریک زنجیر یا در یک لیست از اشاره گر هاهر نود نشان دهنده یک عنصر می باشد.

از یک نود به نود دیگر یک اشاره گر وجود دارد.

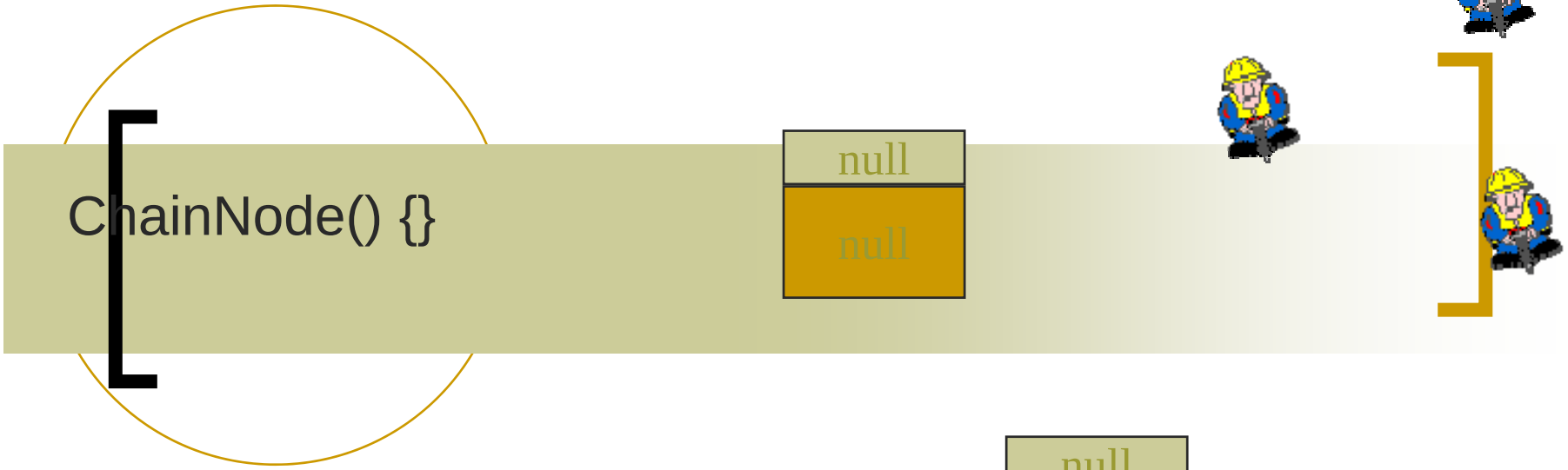
اشاره گر آخرین لیست تهی میباشد.

Node Representation

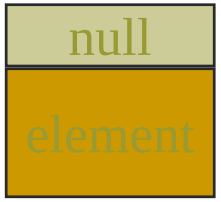
```
package dataStructures;


class ChainNode
{

   Object element;
   ChainNode next;


}
```



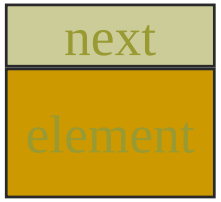
Constructors Of ChainNode

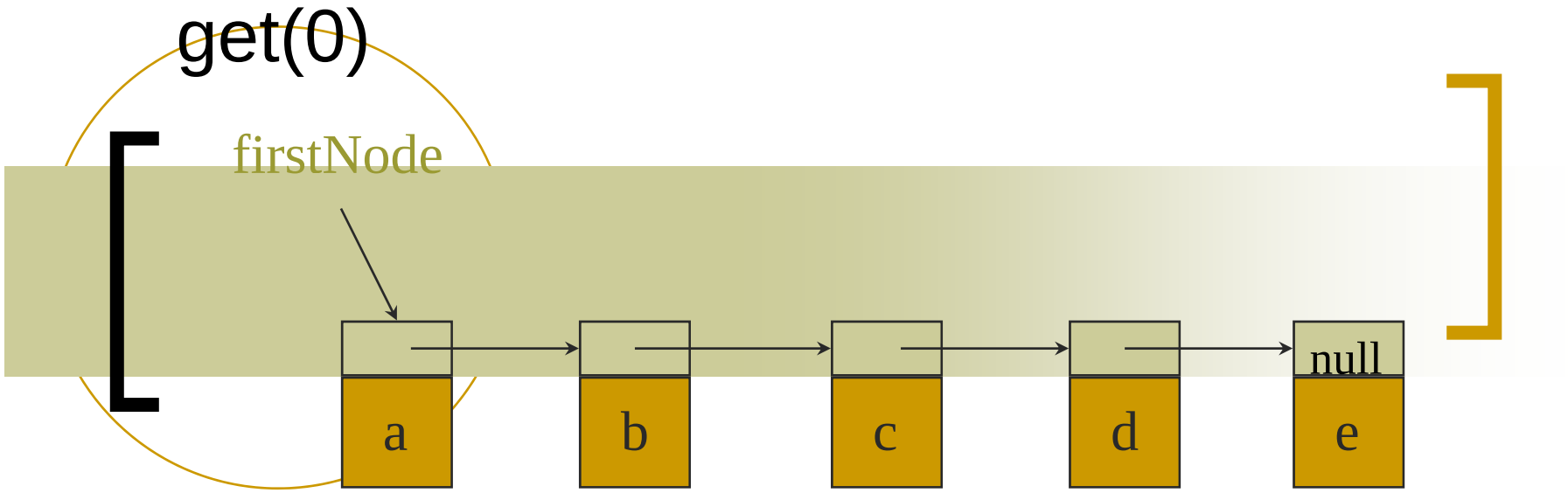


ChainNode(Object element)
{this.element = element;}

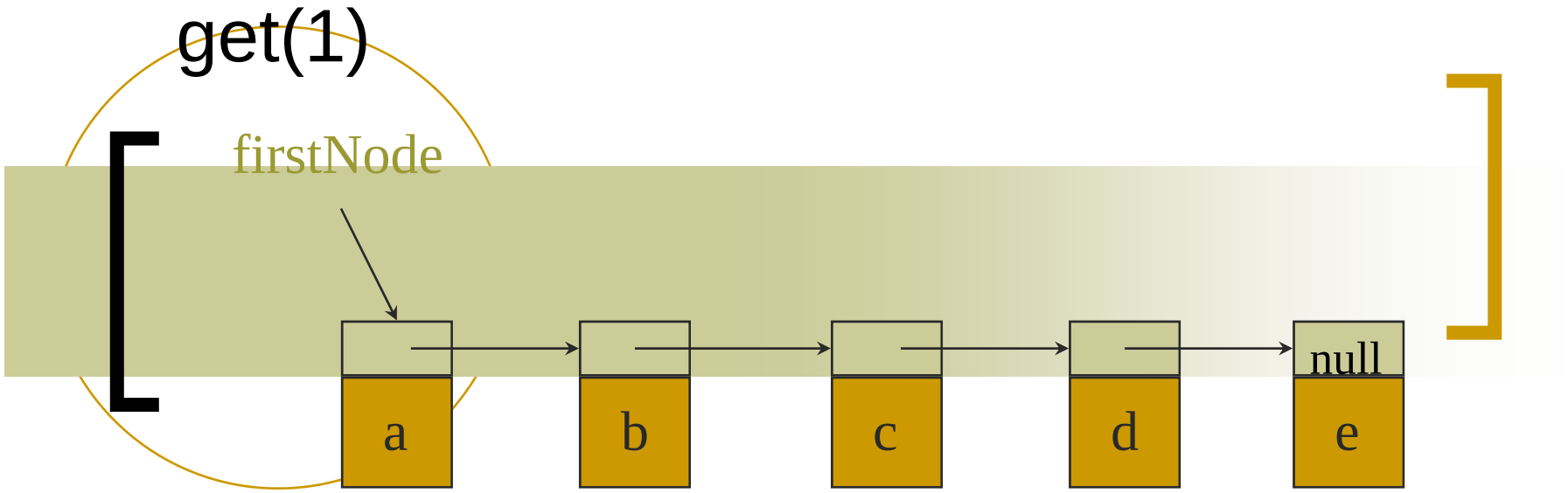


ChainNode(Object element, ChainNode next)
{this.element = element;
this.next = next;}

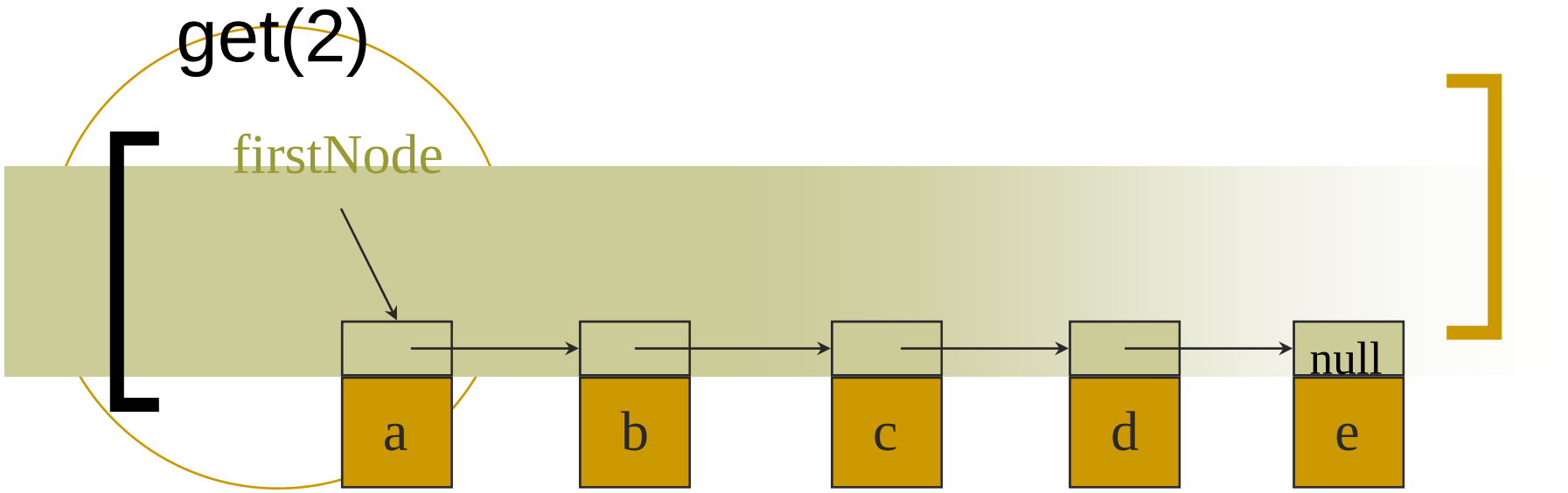




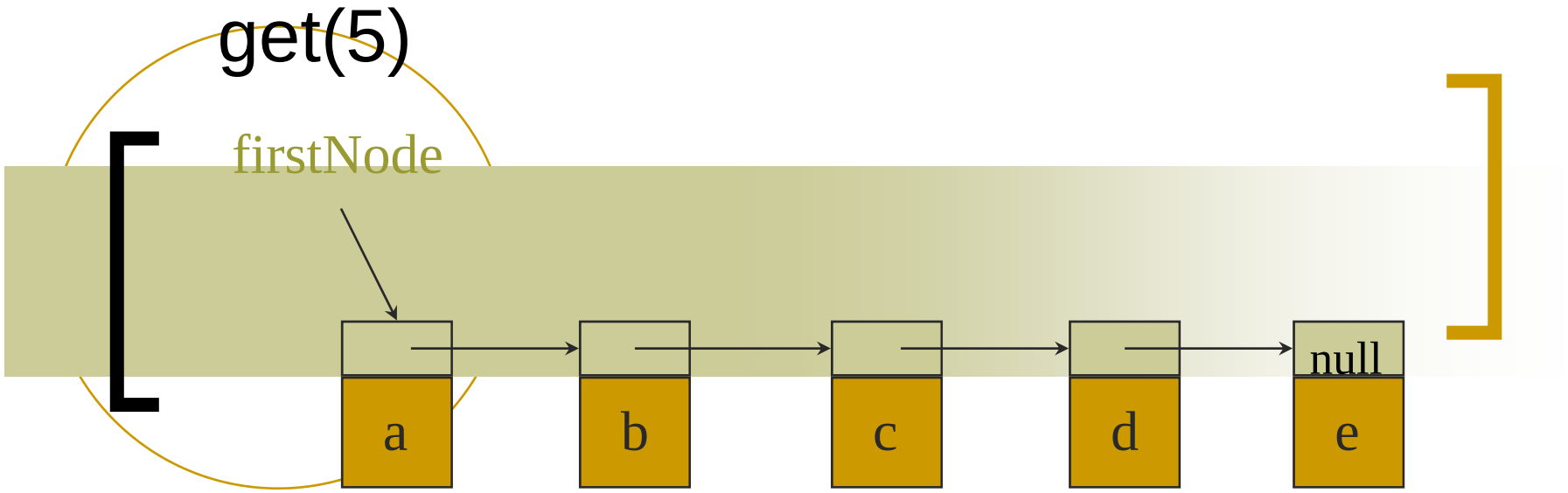

```
checkIndex(0);
desiredNode = firstNode;
return desiredNode.element;
```




```
checkIndex(1);
desiredNode = firstNode.next;
return desiredNode.element;
```



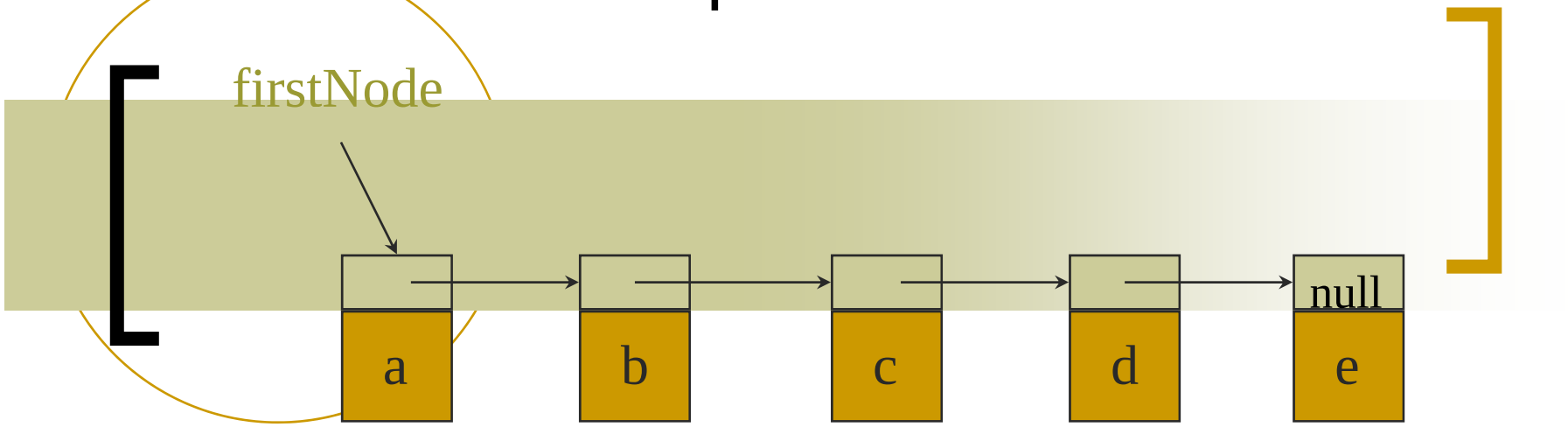

```
checkIndex(2);
desiredNode = firstNode.next.next;
return desiredNode.element;
```



checkIndex(5); desiredNode =
firstNode.next.next.next.next.next;

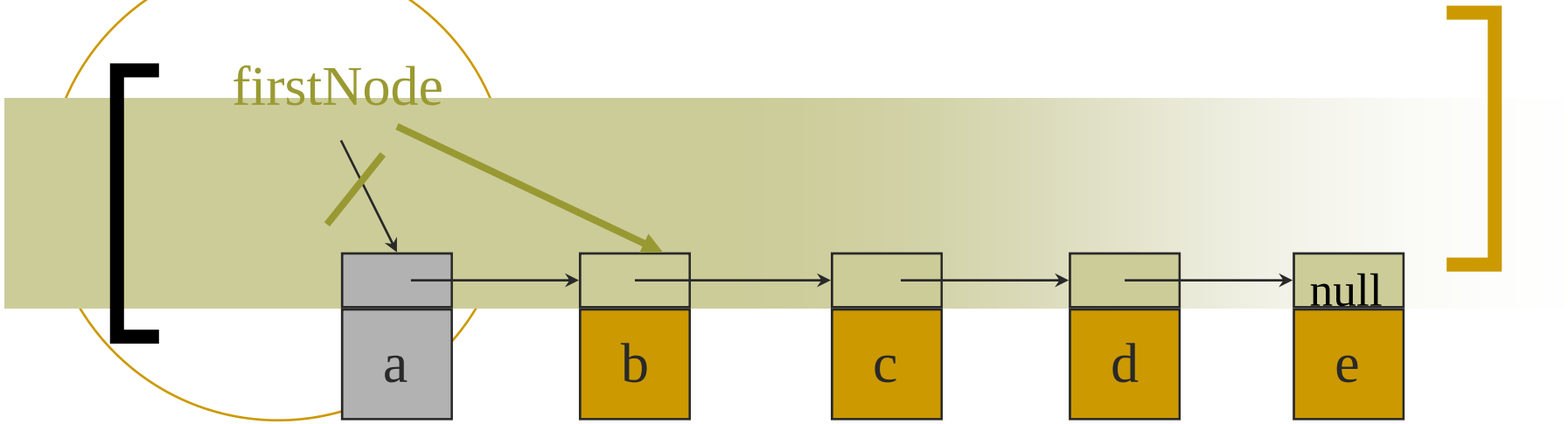
return desiredNode.element;

NullPointerException



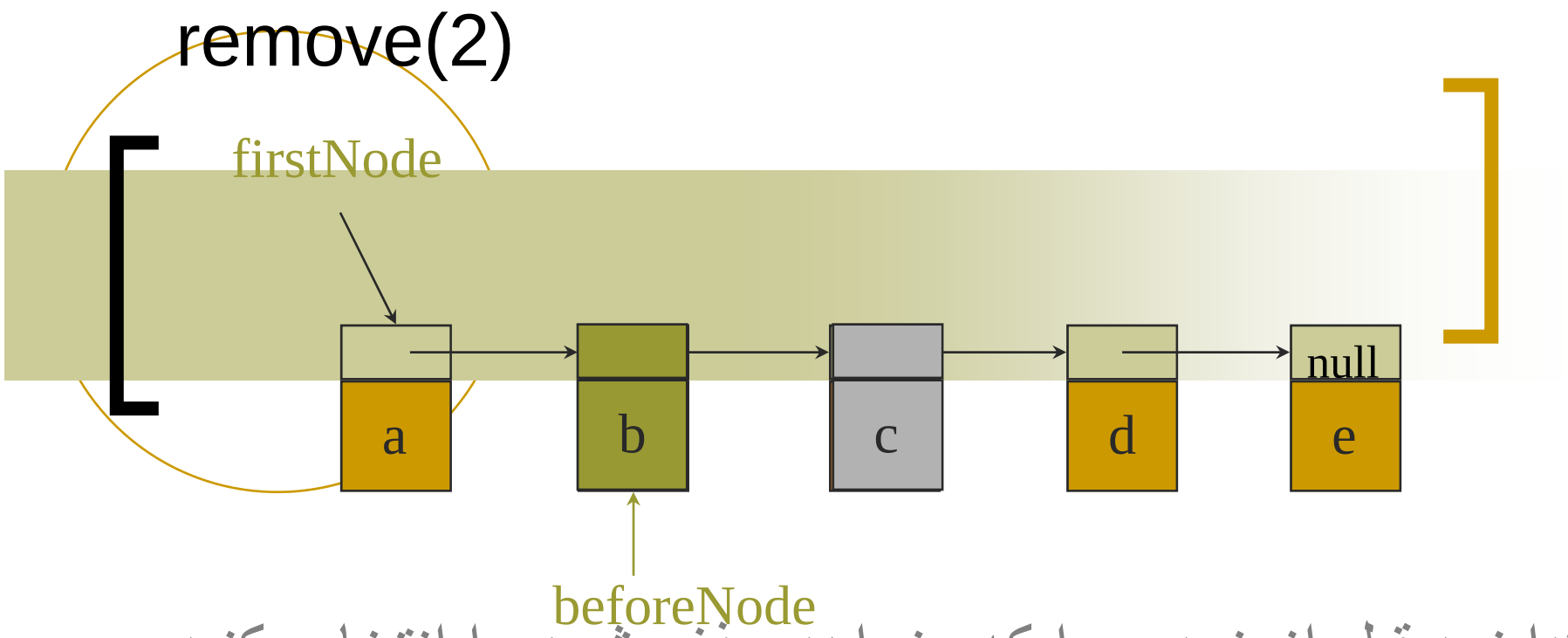
desiredNode =
firstNode.next.next.next.next.ne
xt.next;

Remove An Element



remove(0)

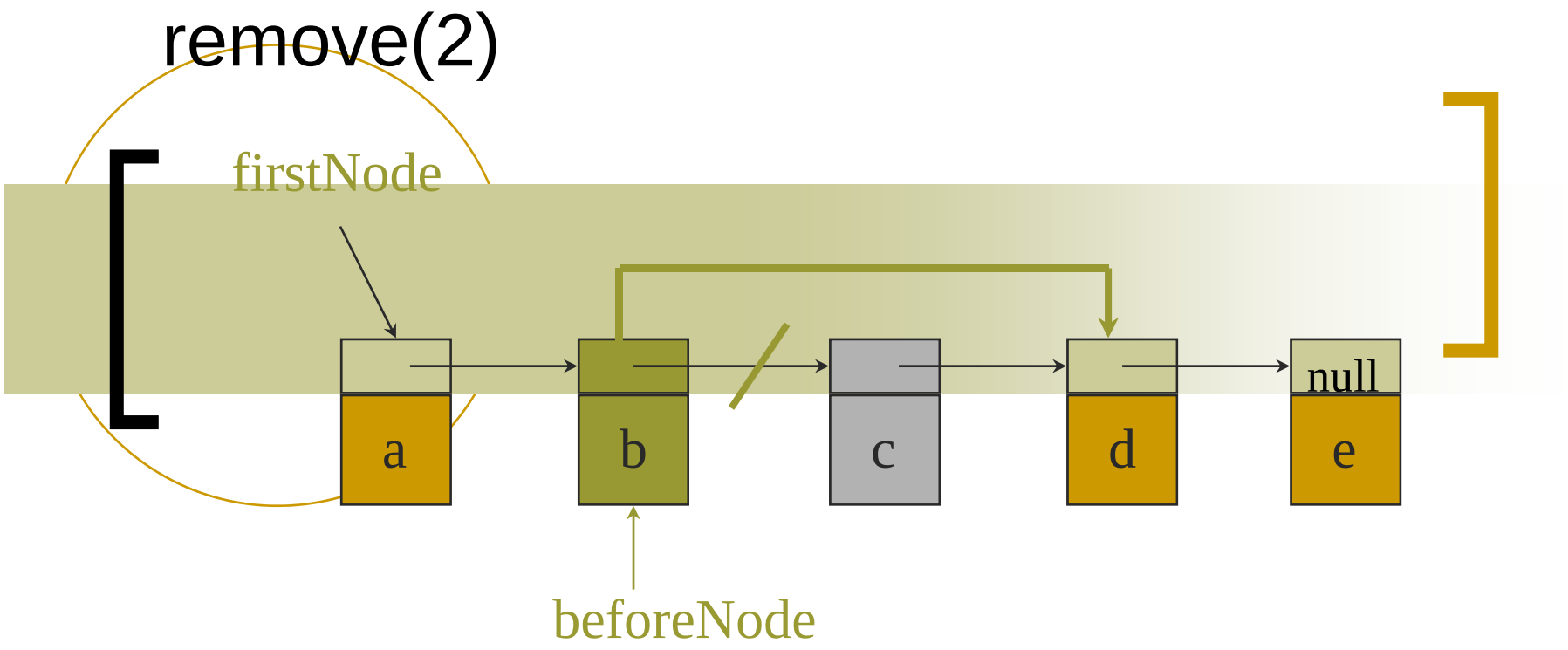
firstNode = firstNode.next;



ابتدا نود قبل از نودی را که بخواهد حذف شود را انتخاب کنید

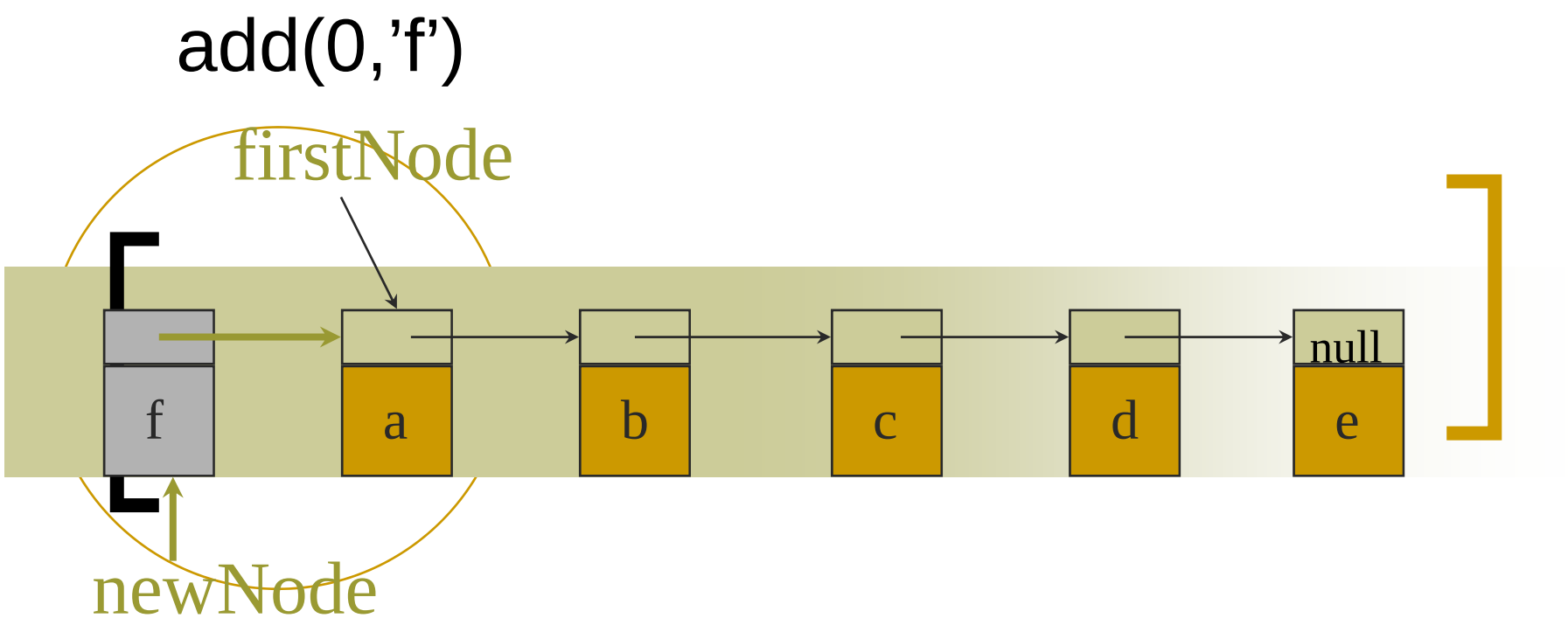
یعنی"first node "

```
beforeNode = firstNode.next;
```



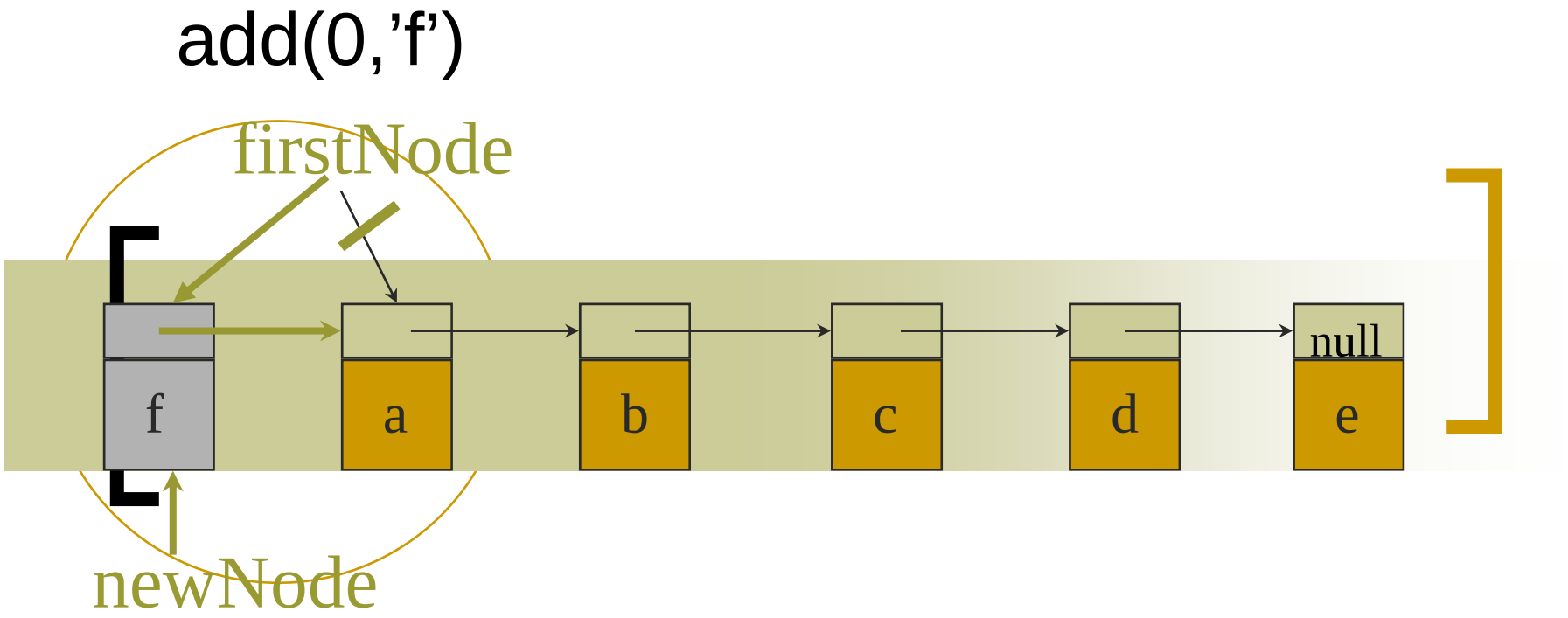
اکنون اشاره گرآن را تغییر دهید.beforeNode در

```
beforeNode.next = beforeNode.next.next;
```



گام اول: نود جدید را انتخاب کنید که حتما دارای فیلد داده واشاره گرباشد .

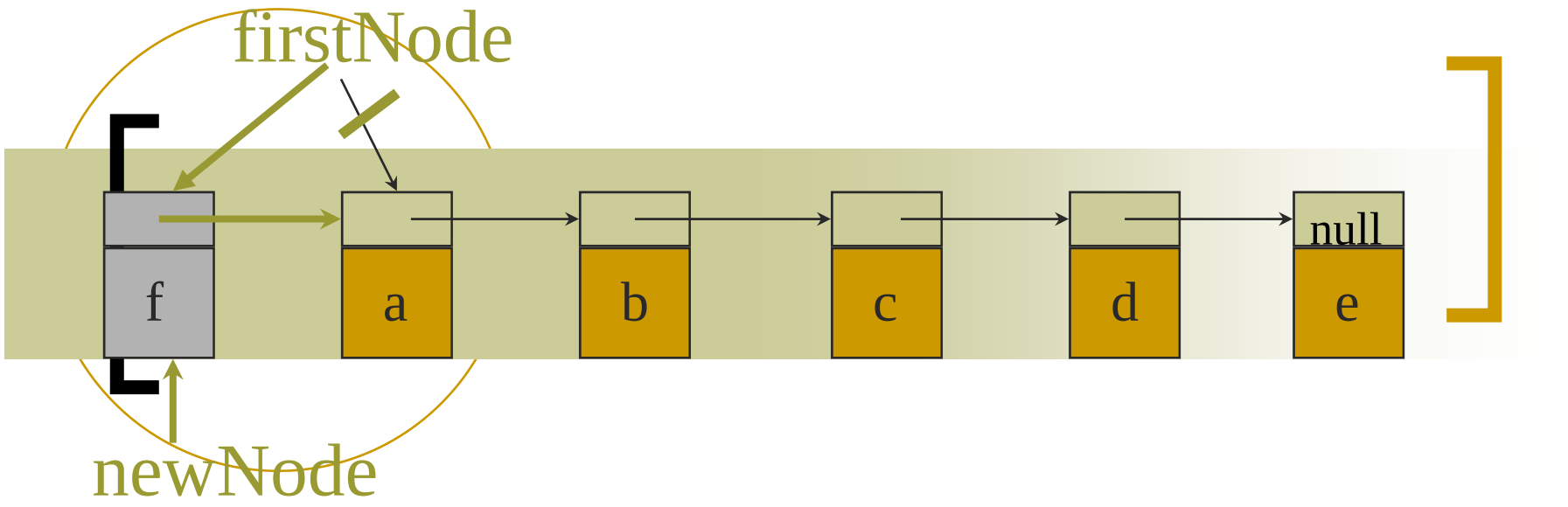
```
ChainNode newNode =
    new ChainNode(new Character(‘f’), firstNode);
```



گام اول: آن را بعنوان نود آغاز انتخاب کنید.

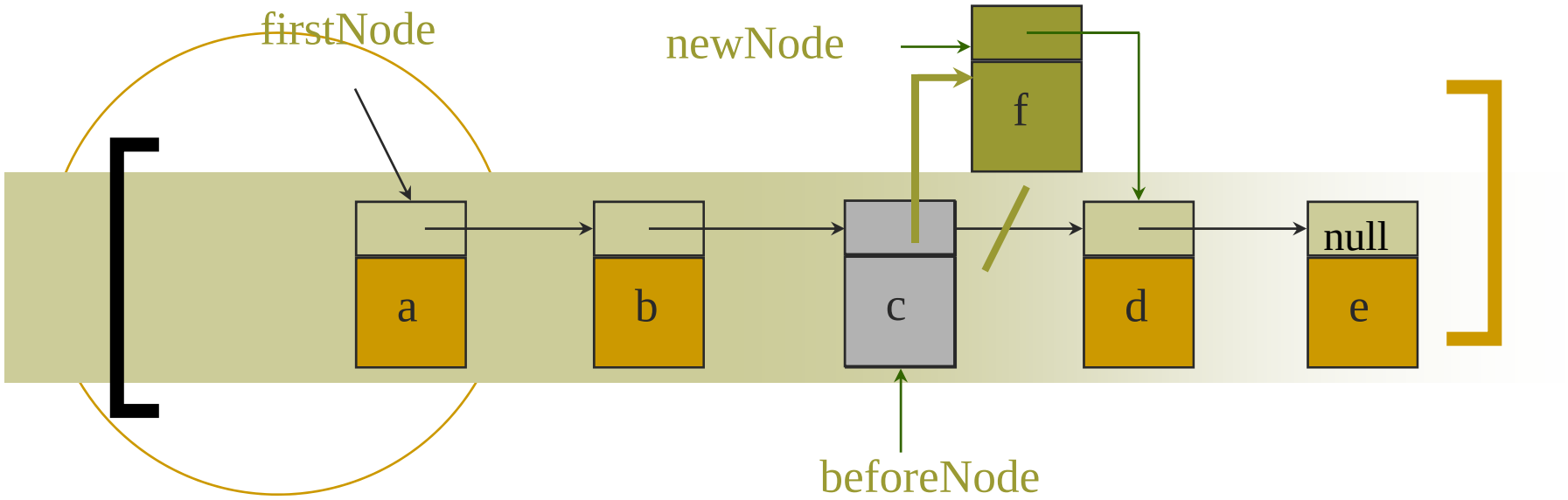
```
firstNode = newNode;
```

One-Step add(0,'f')




```
firstNode = new ChainNode(
new Character('f'),
firstNode);
```

add(3,’f’)



گام اول: ابتدا نود با اندیس ۲را انتخاب کنید.

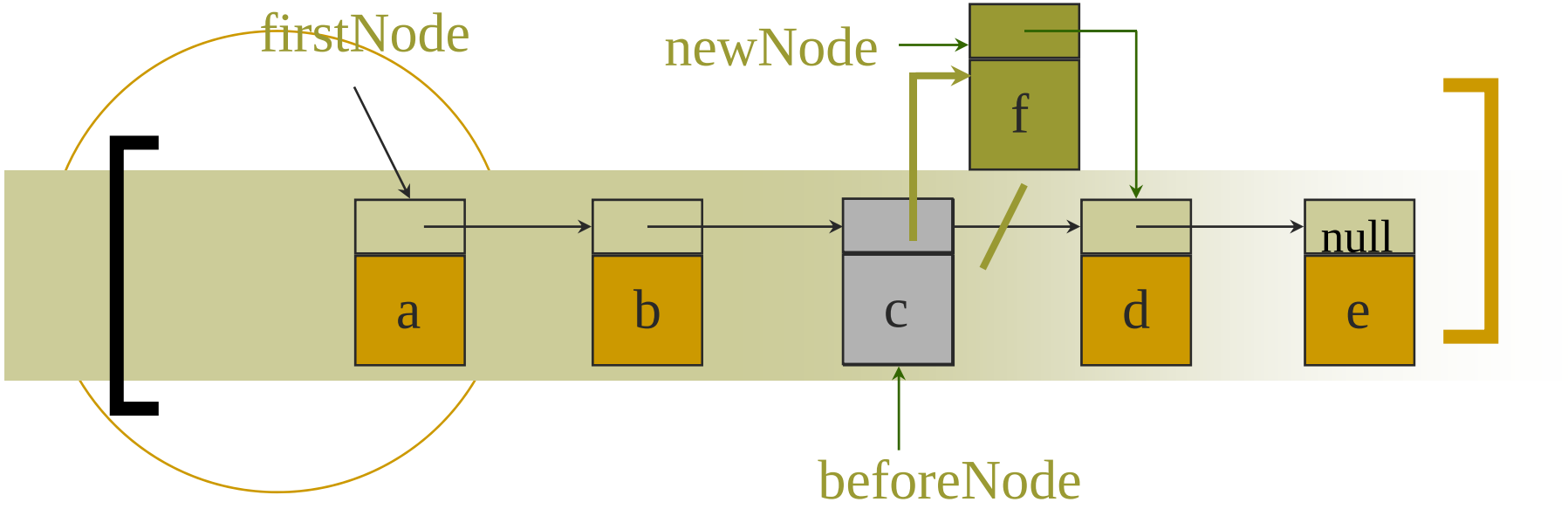
گام دوم: حال نود جدید را ایجاد کرده بطوریکه دارای فیلد داده و فیلد اشاره گر داشته باشد.

```
ChainNode newNode = new ChainNode(new Character(‘f’),
                                  beforeNode.next);
```

- finally link beforeNode to newNode

```
beforeNode.next = newNode;
```

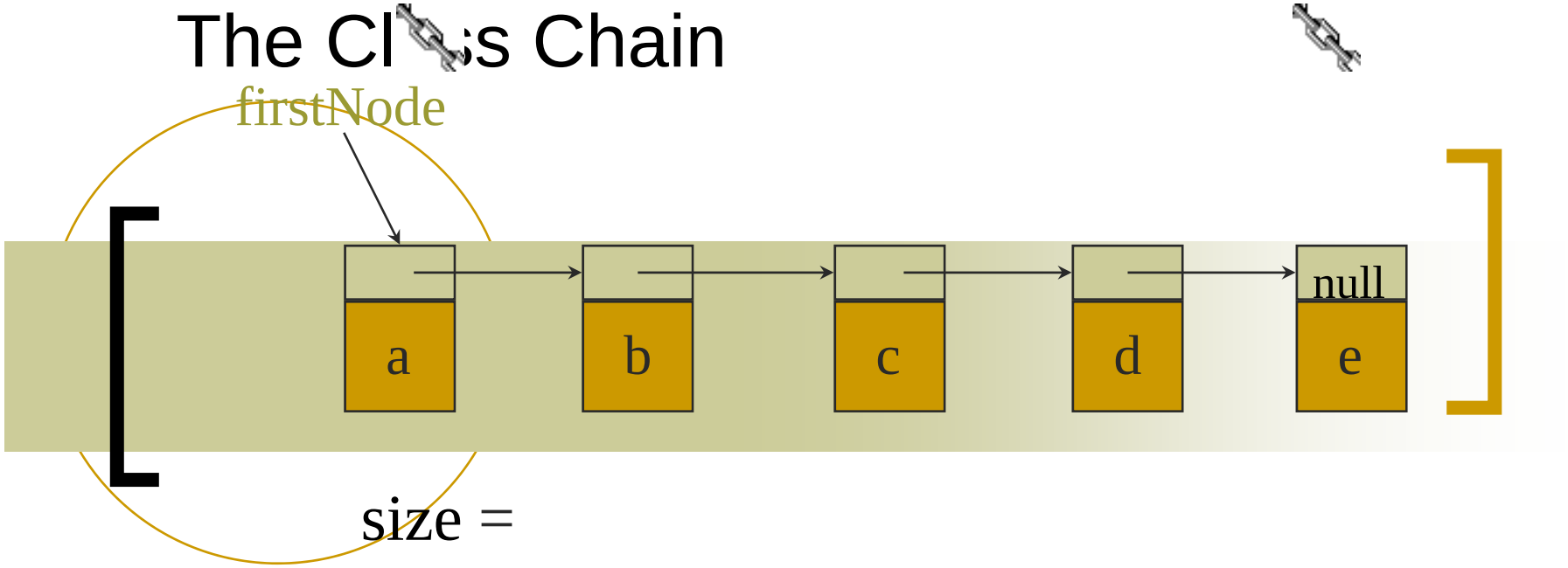
Two-Step add(3,'f')




```
beforeNode = firstNode.next.next;
beforeNode.next = new ChainNode(new Character('f'),
                                beforeNode.next);
```

The Class Chain

The Class Chain



شمارش تعدادعناصر لیست میباشد.

کاربرد ChainNode

next (datatype ChainNode)

element (datatype Object)

The Class Chain

```
/** linked implementation of LinearList */
package dataStructures;
Import c++;  // has Iterator
public class Chain implements LinearList
{
   // data members
   protected ChainNode firstNode;
   protected int size;

   // methods of Chain come here
}
```

Constructors

```
/** ایجاد یک لیست خالی */
  public Chain(int initialCapacity)
  {
firstNode and size: مقادیر اولیه //
        // null and 0
  }

  public Chain()
    {this(0);}
```

The Method isEmpty

```
/**     مقدار درست را برمی گرداندا‌گر وفقط اگر لیست خالی باشد*/
public boolean isEmpty()
  {return size == 0;}
```

The Method size()

```
/**  خروجی آن شمارش تعداد عناصر داخل لیست میباشد*/
  public int size()
    {return size;}
```

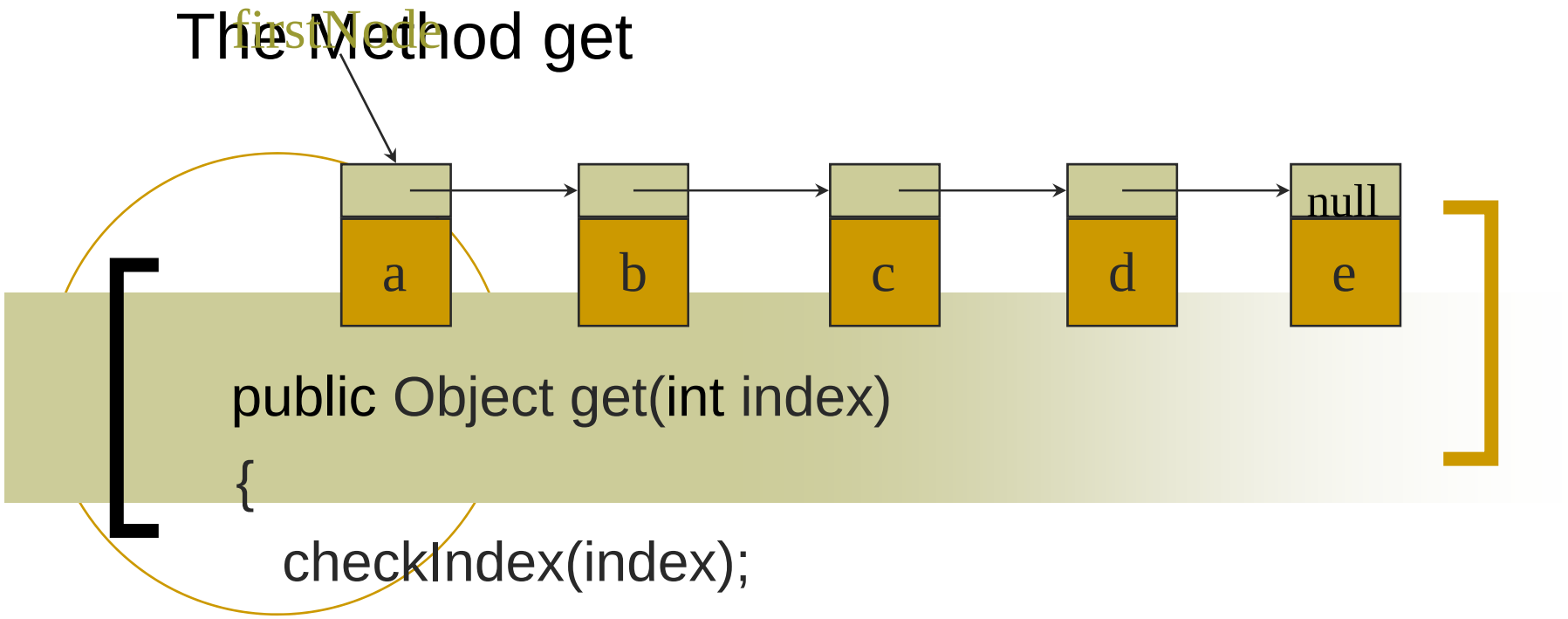
The Method checkIndex

```
/** @throws IndexOutOfBoundsException when
  * index is not between 0 and size - 1 */
```

/**اندیس مربوط به هر عنصر را بر می گرداند و زمانی که ۰ یا ۱- را برگرداند به این مفهوم است که آن عنصر در لیست وجود ندارد.*/

```
void checkIndex(int index)
{
   if (index < 0 || index >= size)
      throw new IndexOutOfBoundsException
            ("index = " + index + "  size = " + size);
}
```

The Method get




```
public Object get(int index)
{
   checkIndex(index);

// نود دلخواه راتغییر دهید
   ChainNode currentNode = firstNode;
   for (int i = 0; i < index; i++)
      currentNode = currentNode.next;

   return currentNode.element;
}
```

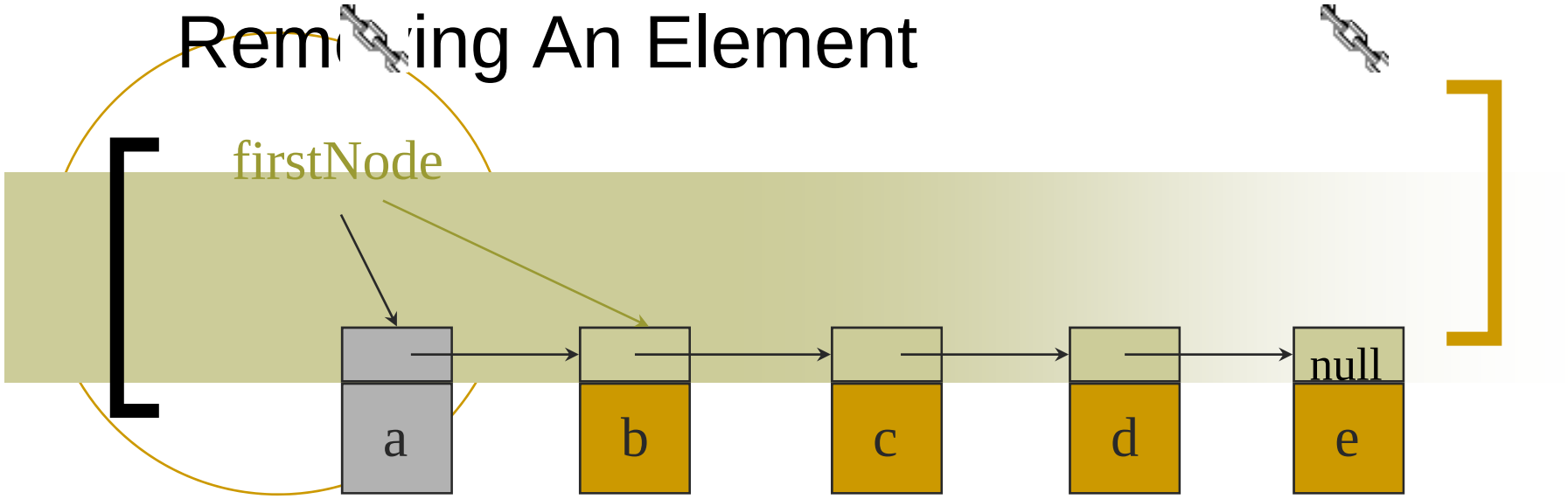
The Method indexOf

```
public int indexOf(Object theElement)
 {
// یک زنجیر از نودها را جستجو کنید
    ChainNode currentNode = firstNode;
// اندیس نود جاری
    int index = 0;
    while (currentNode != null &&

!currentNode.element.equals(theElement))
    {
// به نود بعدی تغییر دهید
      currentNode = currentNode.next;
      index++;
    }
```

The Method indexOf

```
        اطمینان از اینکه عنصر مربوطه را یافته ایم        //
        if (currentNode == null)
            return -1;
        else
            return index;
    }
```

Removing An Element

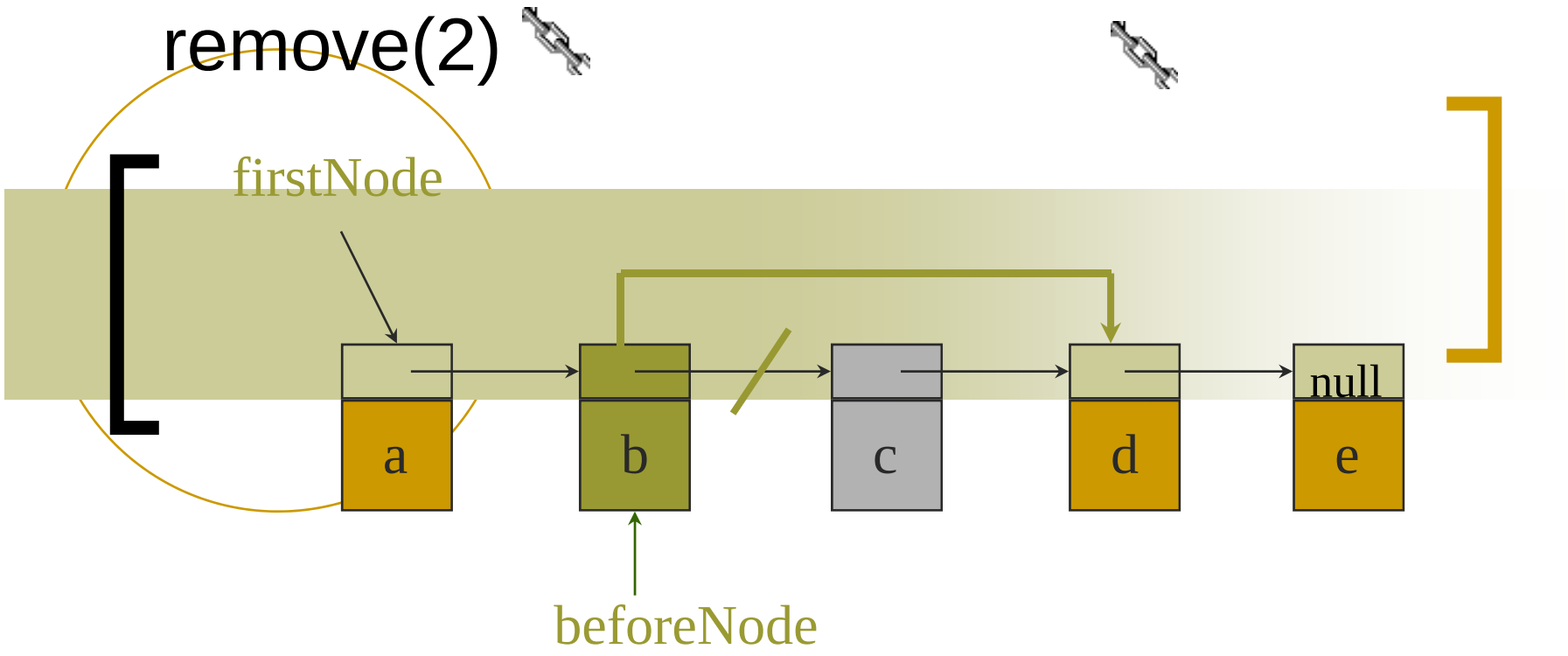


remove(0)

firstNode = firstNode.next;

Remove an Element

```
public Object remove(int index)
{
   checkIndex(index);

   Object removedElement;
   if (index == 0) //  نود شروع را حذف کنید
   {
      removedElement = firstNode.element;
      firstNode = firstNode.next;
   }
```



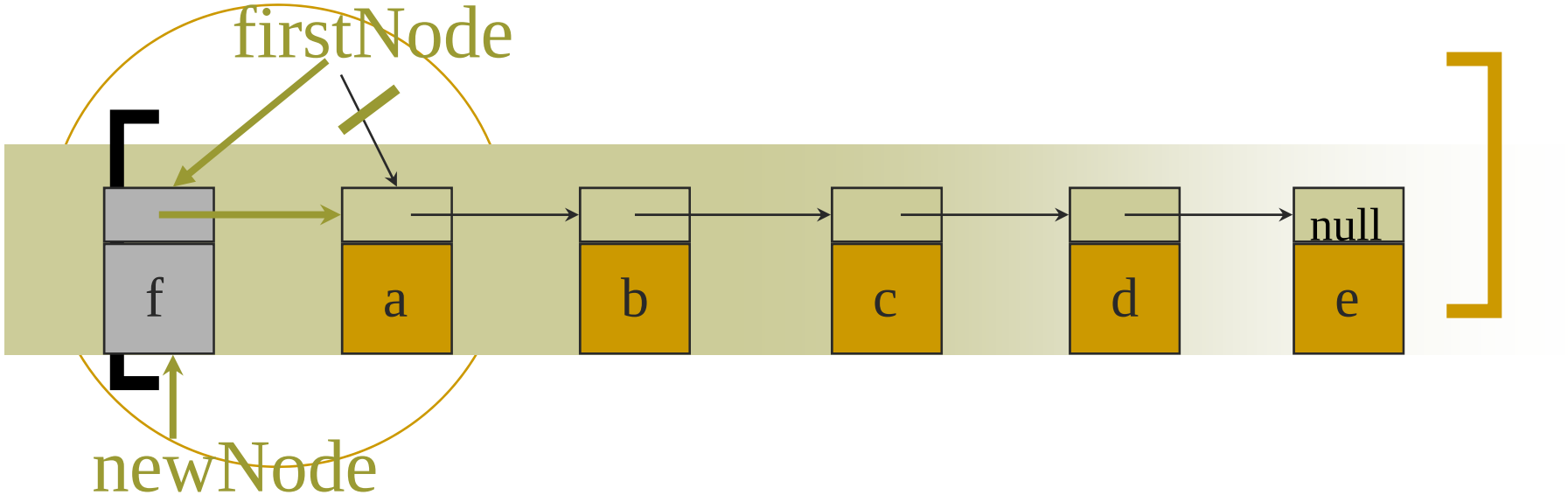
beforeNode را یافته و اشاره گر آن را تغییر دهید

beforeNode.next = beforeNode.next.next;

Remove an Element

```
        else
جایگاهی برای قرار دادن نود جاری میباشدq:    {  //
            ChainNode q = firstNode;
            for (int i = 0; i < index - 1; i++)
                q = q.next;

            removedElement = q.next.element;
نود دلخواه را حذف کنید          q.next = q.next.next; //
        }
        size--;
        return removedElement;
    }
```

One-St[illegible]o add(0,’f’)

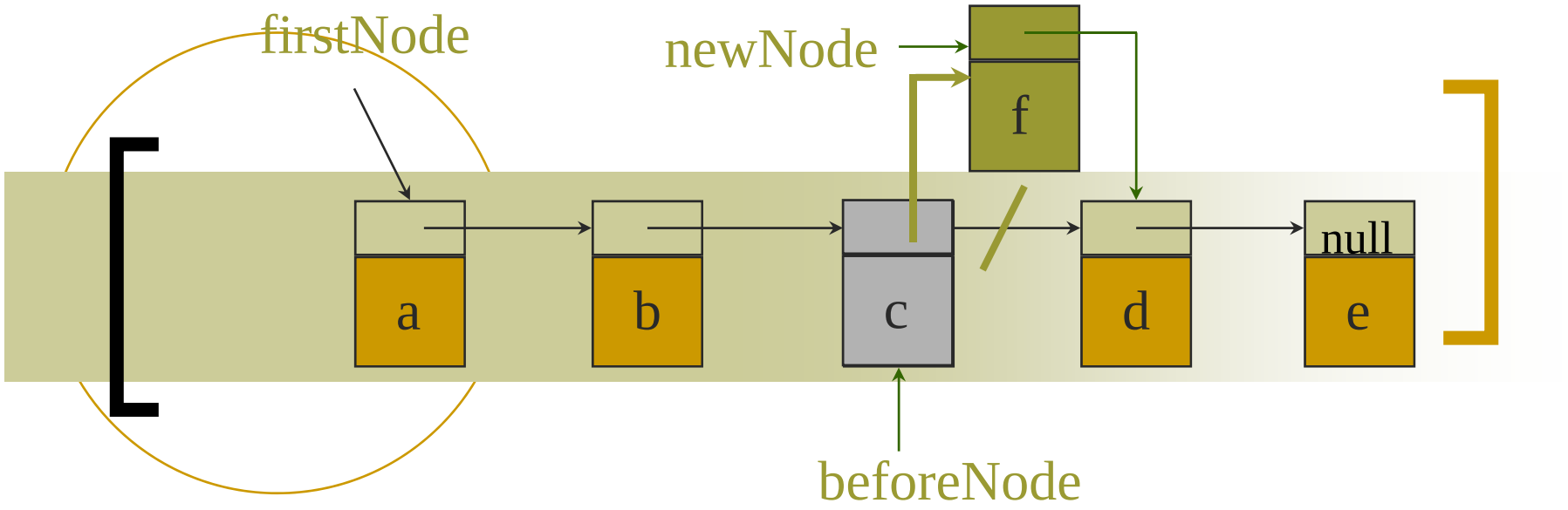



```
firstNode = new ChainNode(‘f’,
firstNode);
```

Add An E ment

```
public void add(int index, Object theElement)
{
   if (index < 0 || index > size)
موقعیت لیست نادرست است          //
      throw new IndexOutOfBoundsException
            ("index = " + index + "  size = " + size);

   if (index == 0)
به جلو اضافه کنید.          //
      firstNode = new ChainNode(theElement, firstNode);
```

Two-Step add(3,'f')

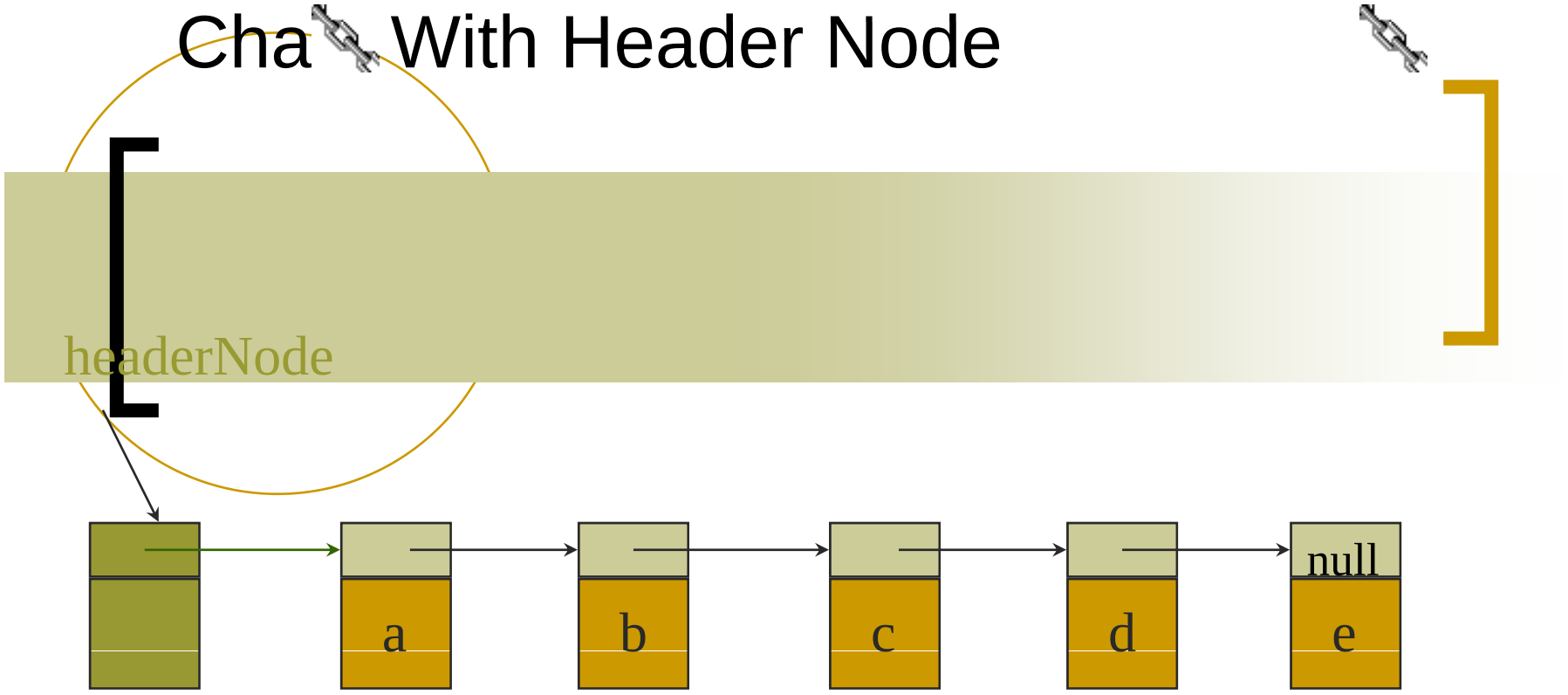



```
beforeNode = firstNode.next.next;
beforeNode.next = new ChainNode('f', beforeNode.next);
```

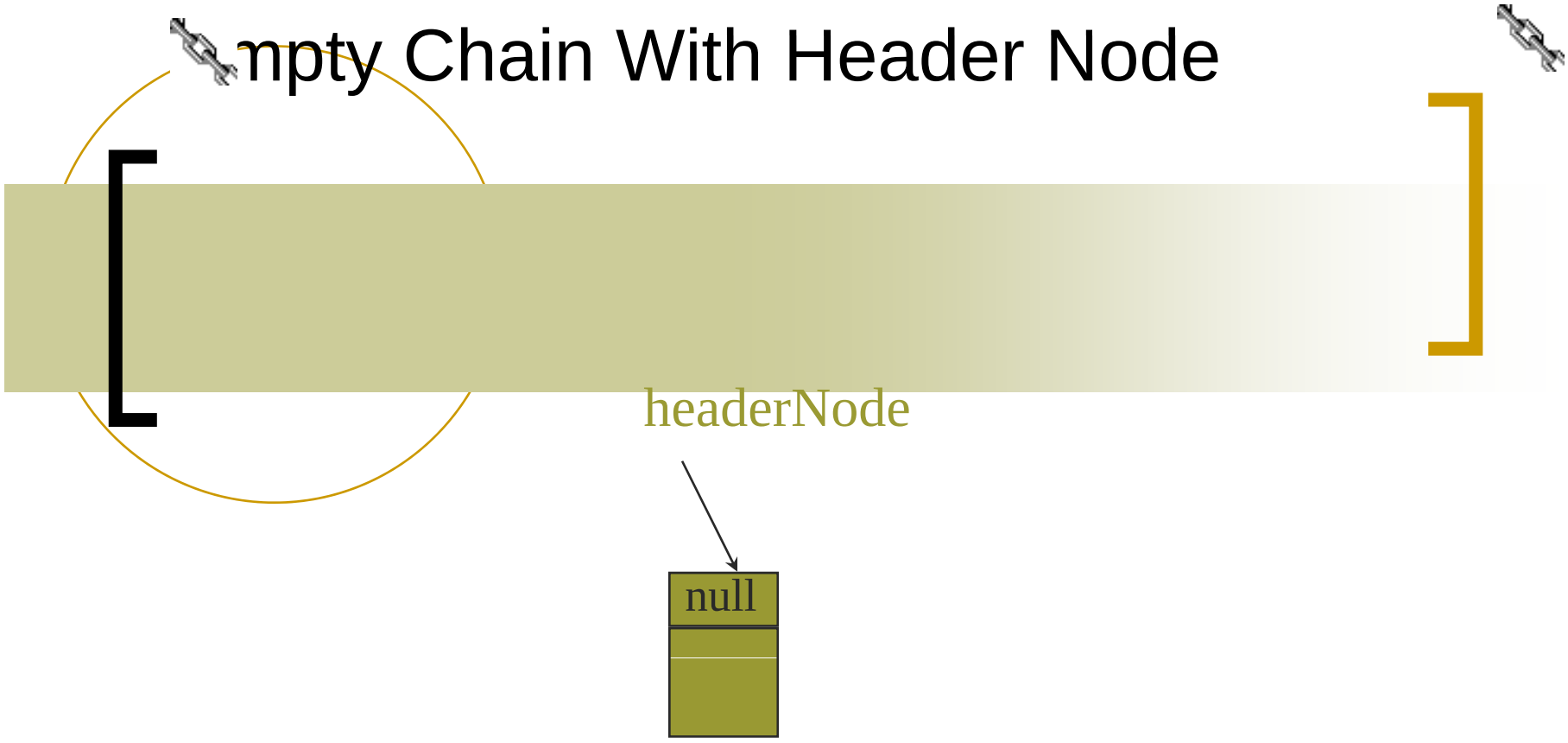
Adding An Element

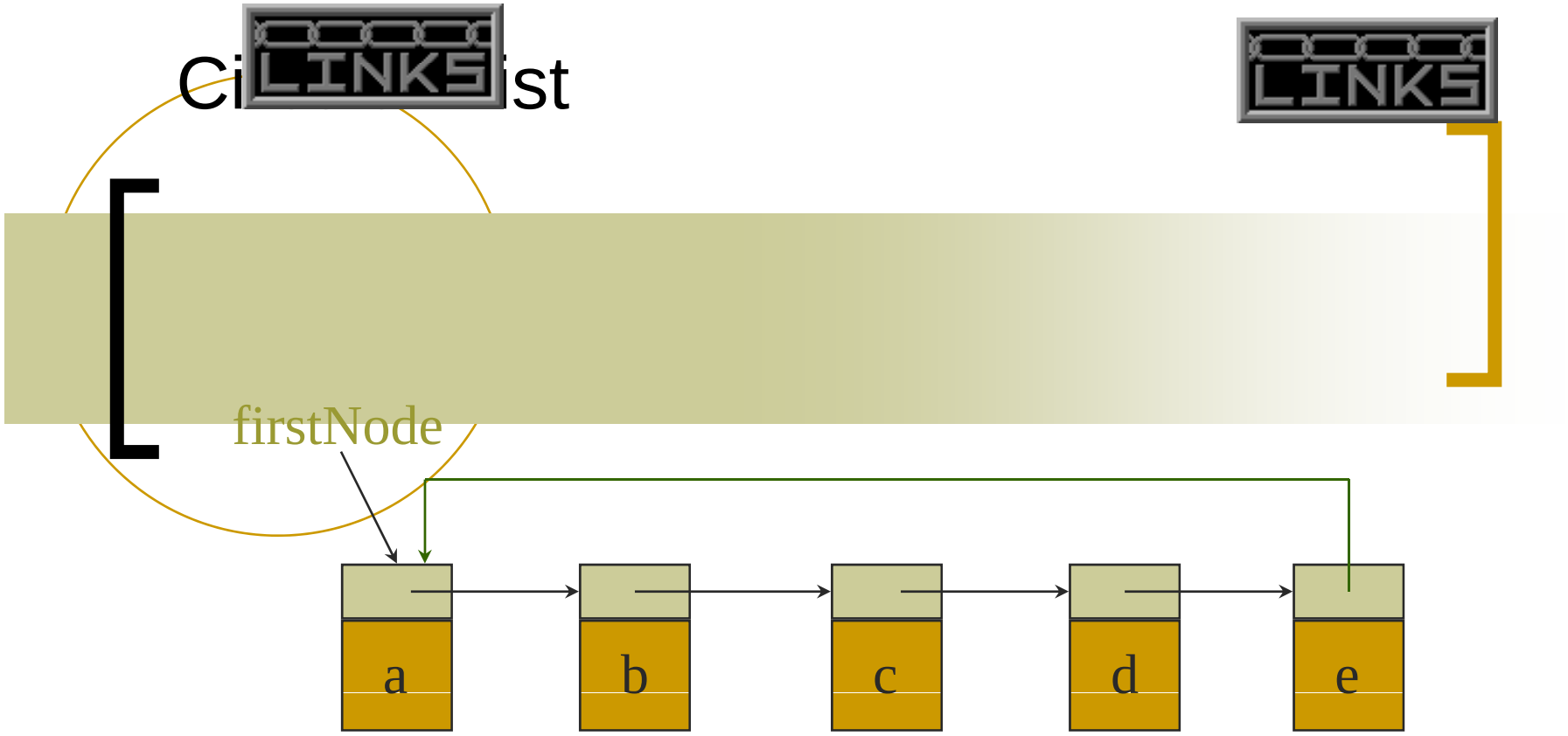
```
else
{ // عنصر جدید را بیابید
    ChainNode p = firstNode;
    for (int i = 0; i < index - 1; i++)
        p = p.next;

// p اضافه کنید بعد از
    p.next = new ChainNode(theElement, p.next);
  }
  size++;
}
```

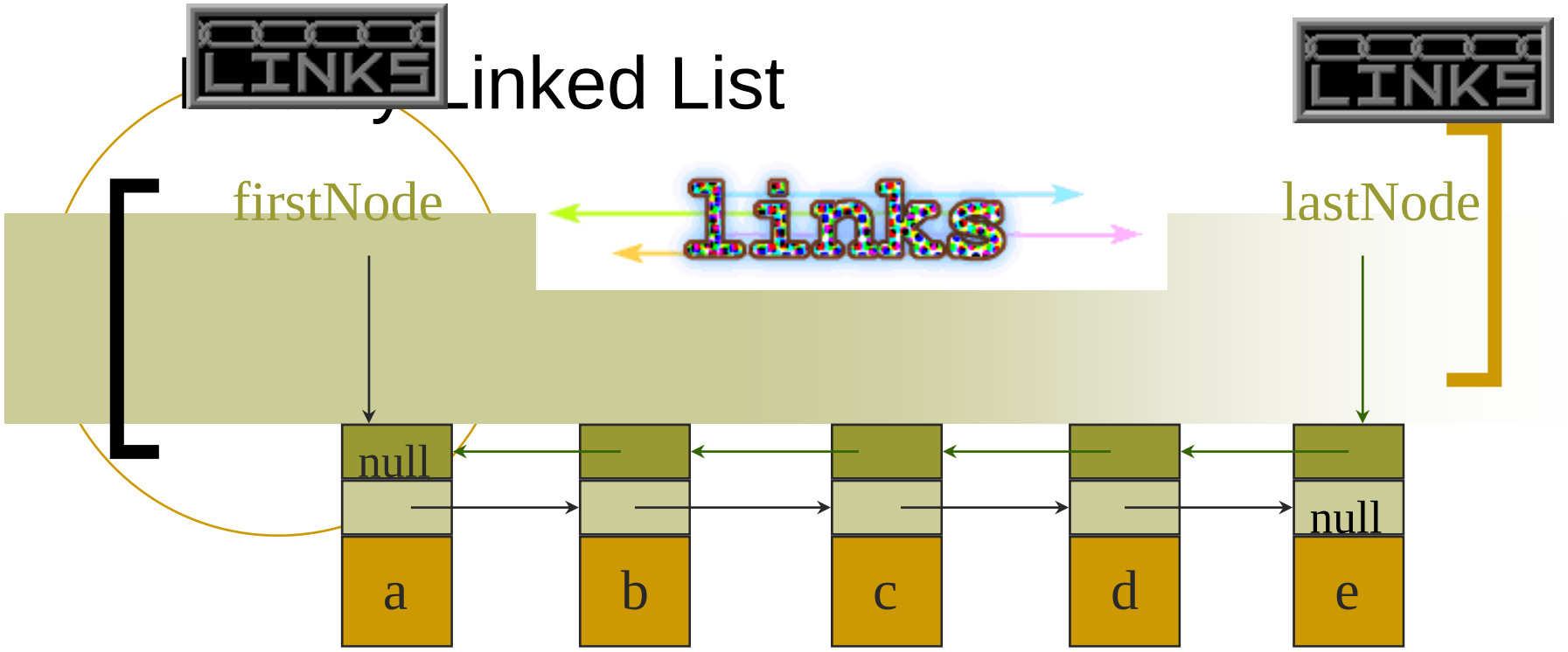
Cha With Header Node

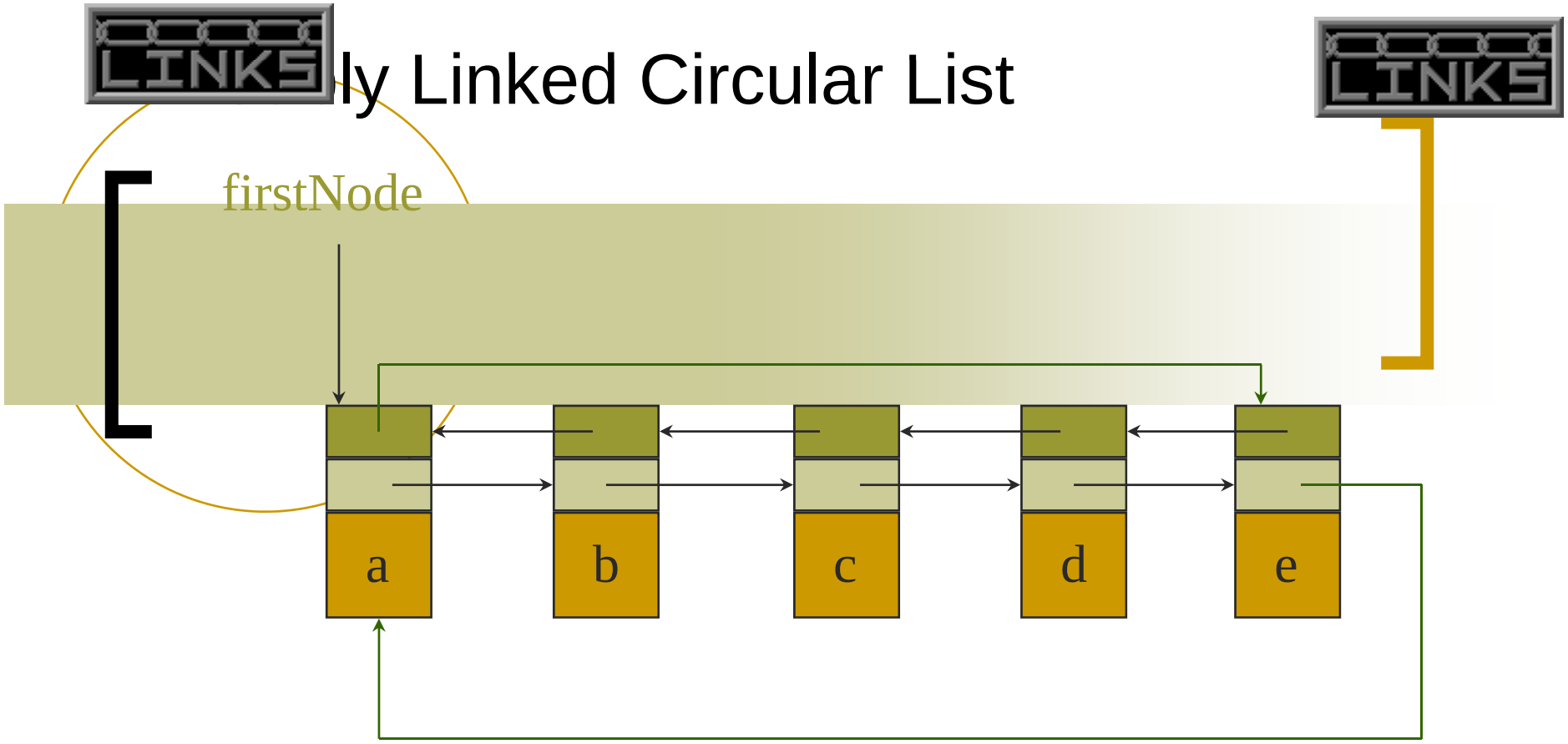


Empty Chain With Header Node

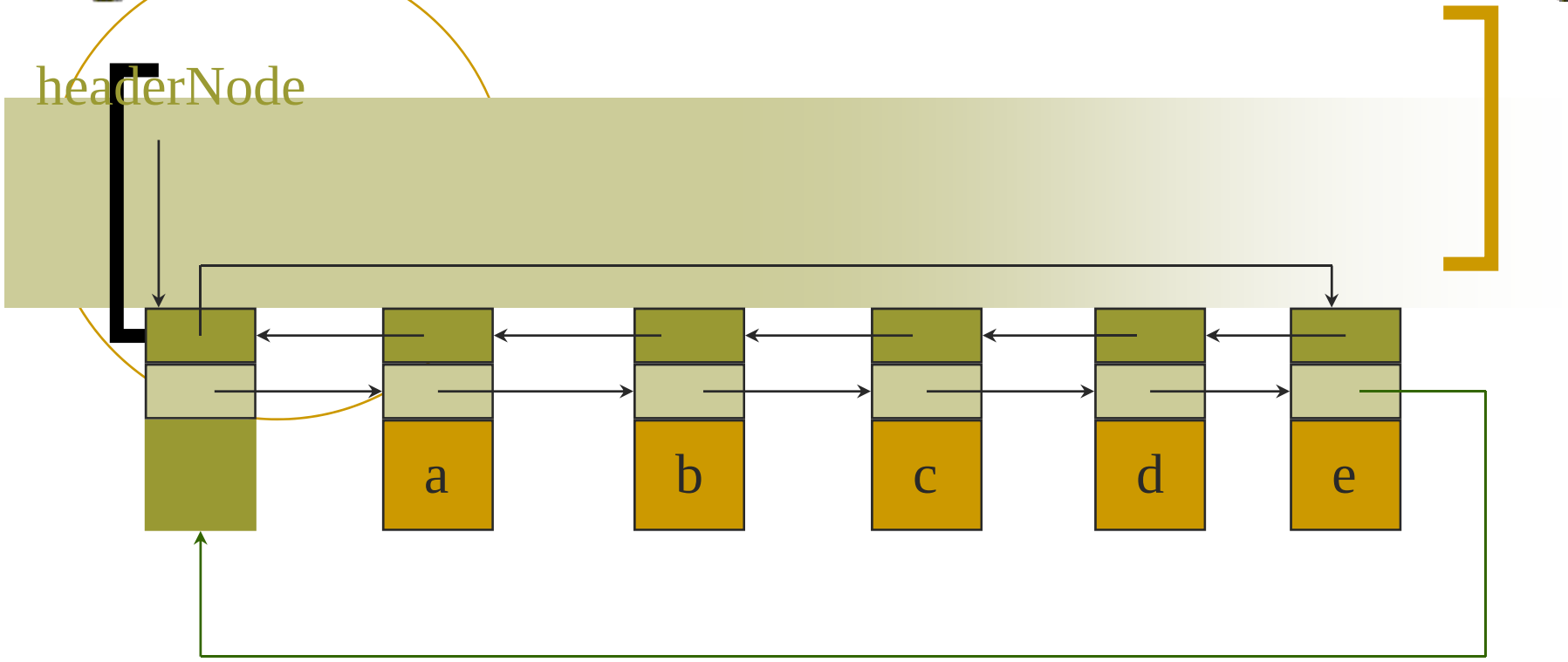



Ci
LINKS
ist
LINKS
firstNode
a
b
c
d
e


Linked List
firstNode
links
lastNode
null
null
a
b
c
d
e

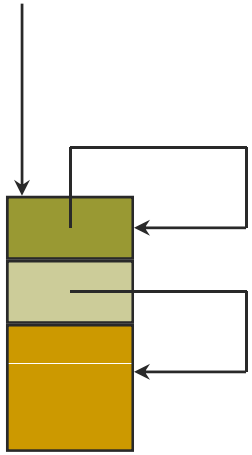

ly Linked Circular List
firstNode
a
b
c
d
e

Doubly Linked Circular List With Header Node

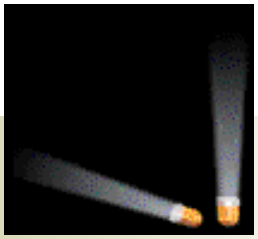


Empty Doubly Linked Circular List With Header Node

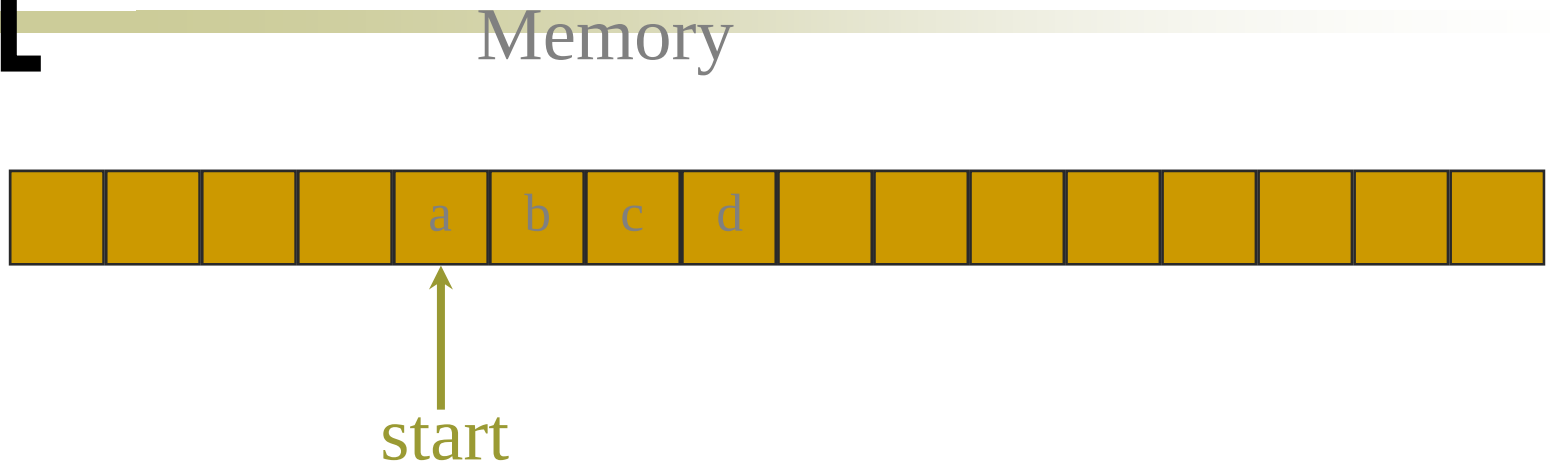
headerNode



Arrays



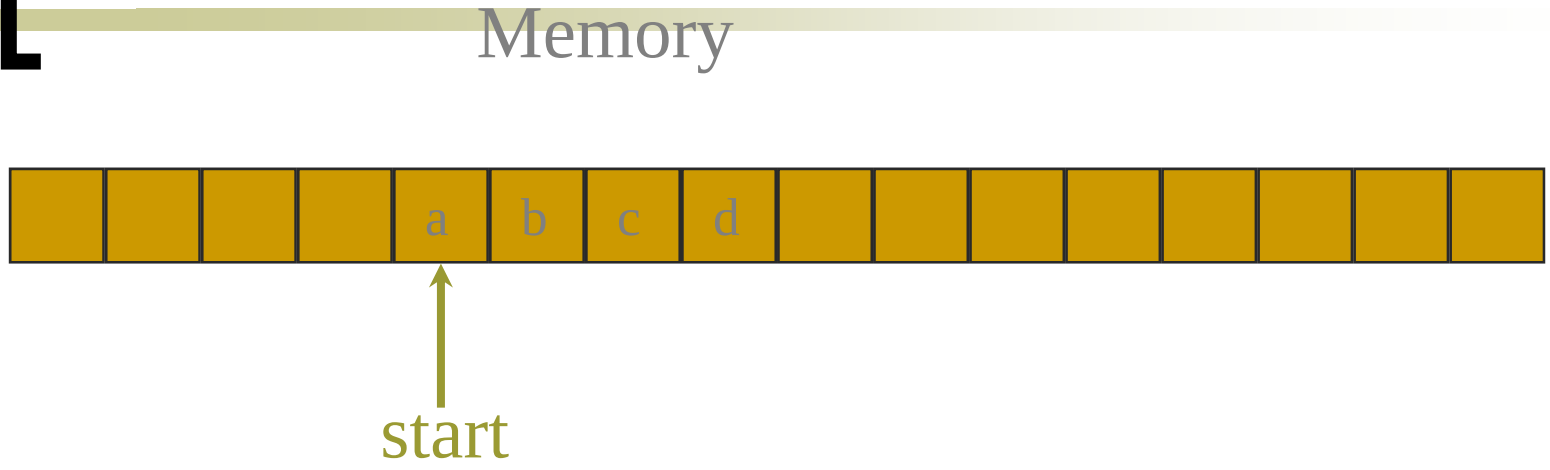
1D Array Representation In C, and C++



- یک نمونه آرایه یک بعدی x = [a, b, c, d]
- د ر خانه های مجاور یکدیگر قرار میگیرند

- location(x[i]) = start + i

Space Overhead



space overhead = 4 bytes for start

+ 4 bytes for x.length

= 8 bytes

)میزان فضایی که برای آرایه فوق نیاز داریم (

2D Arrays

آرایه دو بعدی a

declared as:

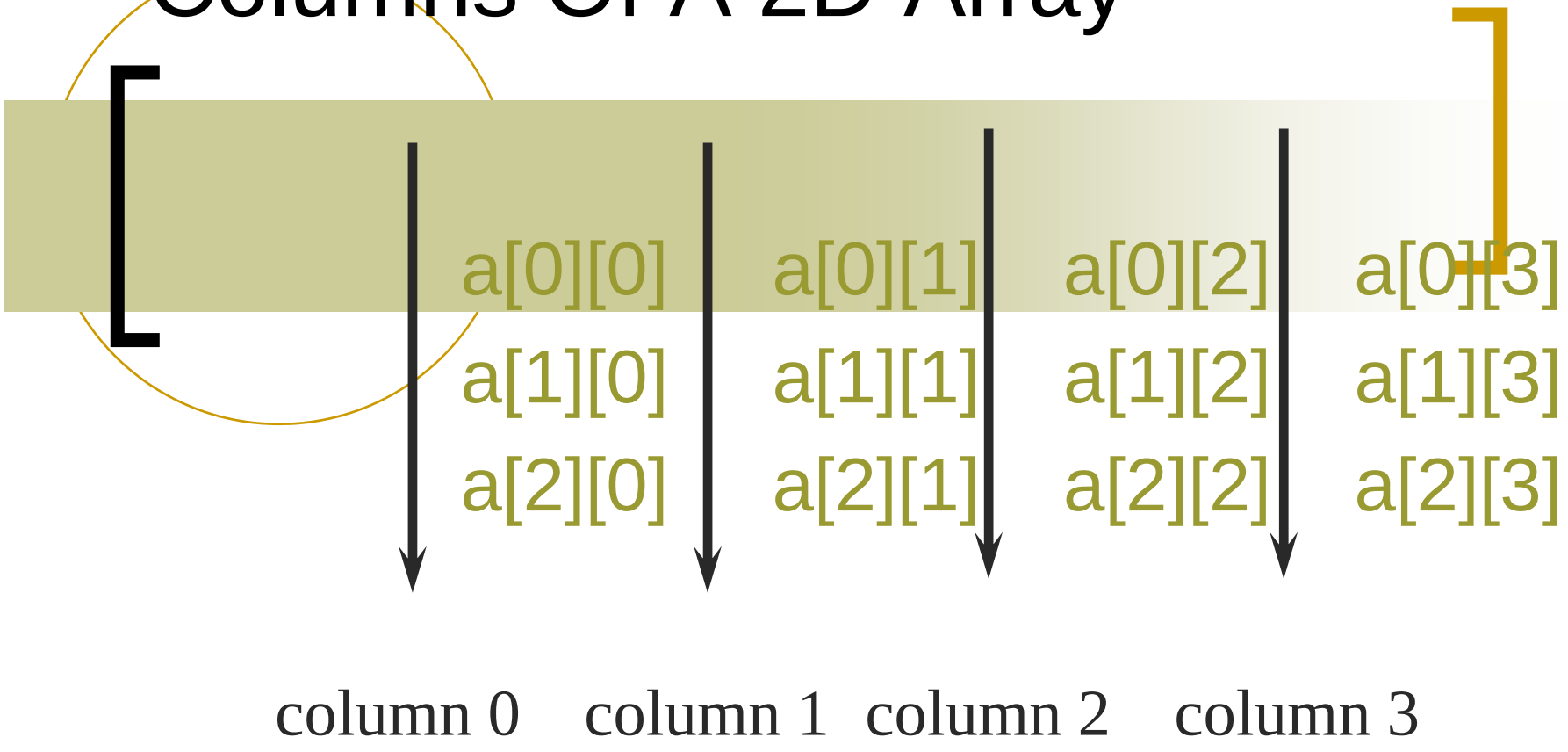
```
int [][]a = new int[3][4];
```

که در یک جدول نشان داده م شود.

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows Of A 2D Array

Columns Of A 2D Array



2D Array Representation In C and C++

آرایه دو بعدی x

a, b, c, d

e, f, g, h

i, j, k, l

سطر های آرایه دو بعدی ,آرایه های یک بعدی هستند.

نمایش آرایه های دوبعد با استفاده از آرایه یک بعدی

x = [row0, row1, row 2]

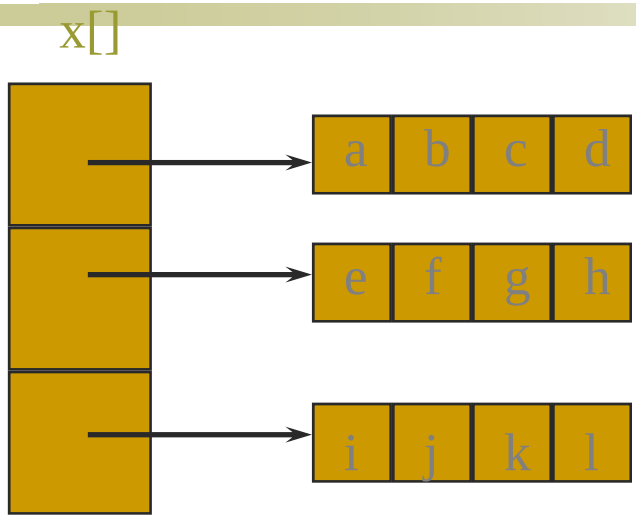
row 0 = [a,b, c, d]

row 1 = [e, f, g, h]

row 2 = [i, j, k, l]

and store as 4 1D arrays

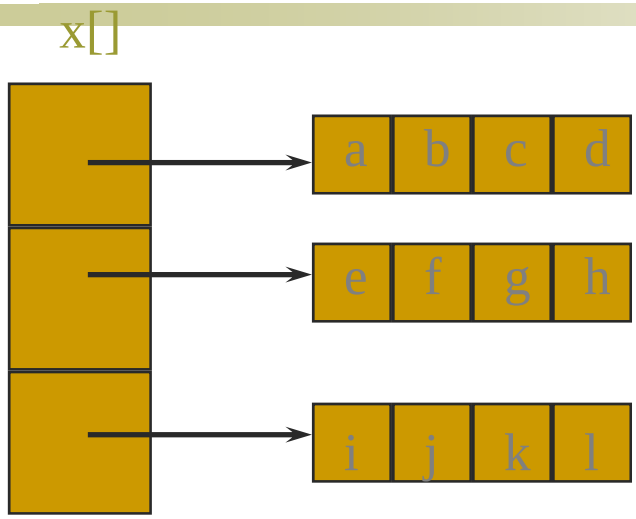
2D Array Representation In Java, C, and C++



x.length = 3

x[0].length = x[1].length = x[2].length = 4

Space Overhead



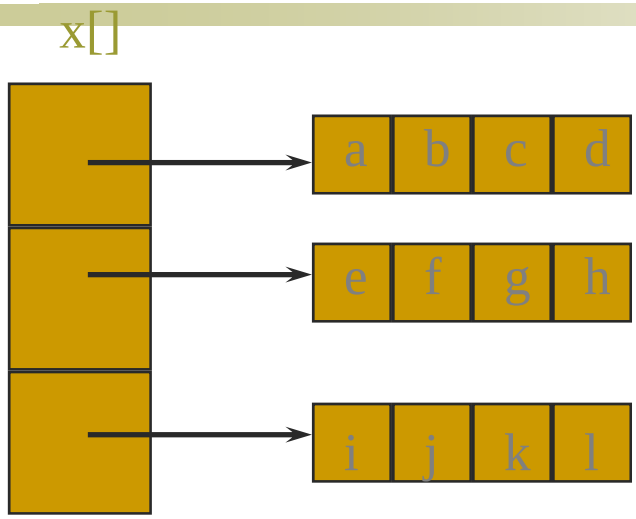
space overhead = overhead for 4 1D arrays

= 4 * 8 bytes

= 32 bytes

= (number of rows + 1) x 8 bytes

Array Representation In C and C++



این نمایش آرایه آرایه ها نامیده می شود.

با توجه به شکل فوق نیازبه حافظه ای با سایز های ۳و۴و۴و۴ برای ذخیره آرایه های یک بعدی دارد.

یک بلوک حافظه شامل سایزتعداد ستونها و سطرها میباشد

Row-Major Mapping

Example 3 x 4 array:

a b c d

e f g h

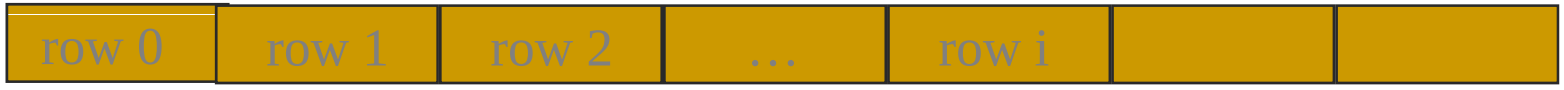
i j k l

آرایه فوق در یک آرایه یک بعدی قرار گرفته است بطوریکه هر عنصر آن یک سطر از آرایه فوق است و بعبارتی هر سطر آن یک آرایه یک بعدی می باشد.

عناصرداخل هر سطر از چپ به راست مرتب شده اند .

سطرها از بالا به پایین مرتب شده اند .

We get {a, b, c, d, e, f, g, h, i, j, k, l}



Column-Major Mapping

آرایه فوق در یک آرایه یک بعدی قرار گرفته است بطوریکه هر عنصر آن یک ستون از آرایه فوق است و بعبارتی هر ستون آن یک آرایه یک بعدی می باشد.

عناصرداخل هر ستون از بالا به پایین مرتب شده اند .

ستون هااز چپ به راست مرتب شده اند .

We get {a, e, i, b, f, j, c, g, k, d, h, l}

Sparse Matrices

- sparse ... تعدادبیشتر عناصر آن صفر می باشد
- dense ... تعدادکمتر عناصر آن صفر می باشد

Representation Of Unstructured Sparse Matrices

هر عنصر غیر صفری در ماتریس اسپارس را میتوان بصورت زیر نشان داد

(row, column, value)

Single Linear List Example

0 0 3 0 4

0 0 5 7 0

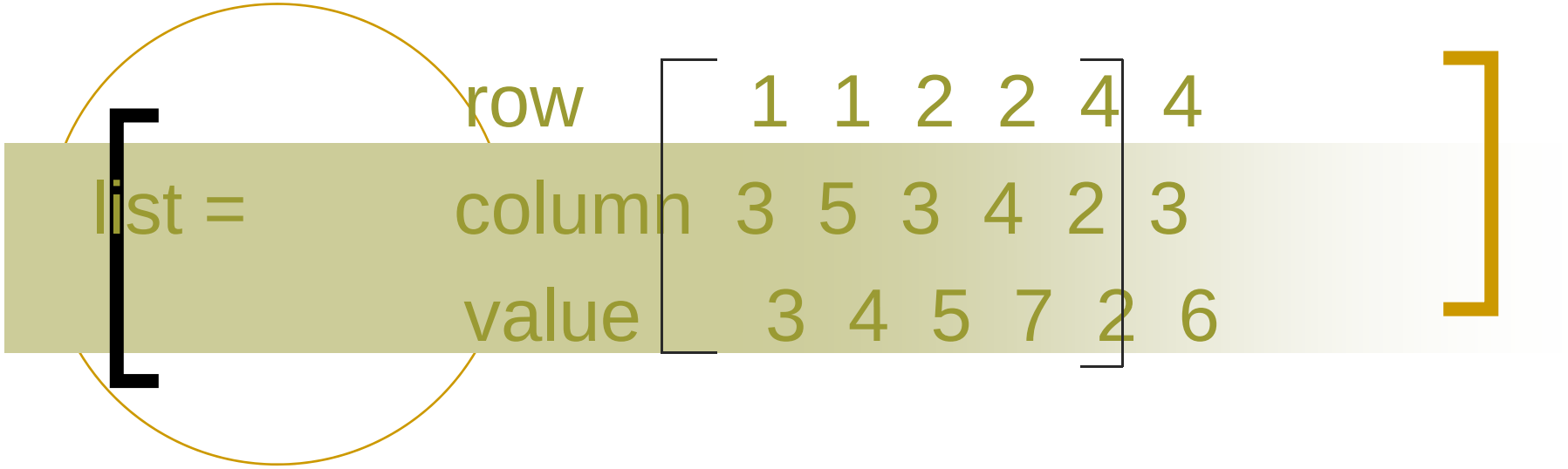
0 0 0 0 0

0 2 6 0 0

list =

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

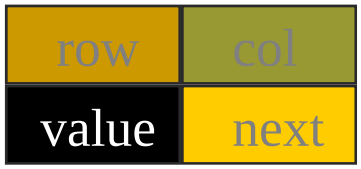
Array Linear List Representation



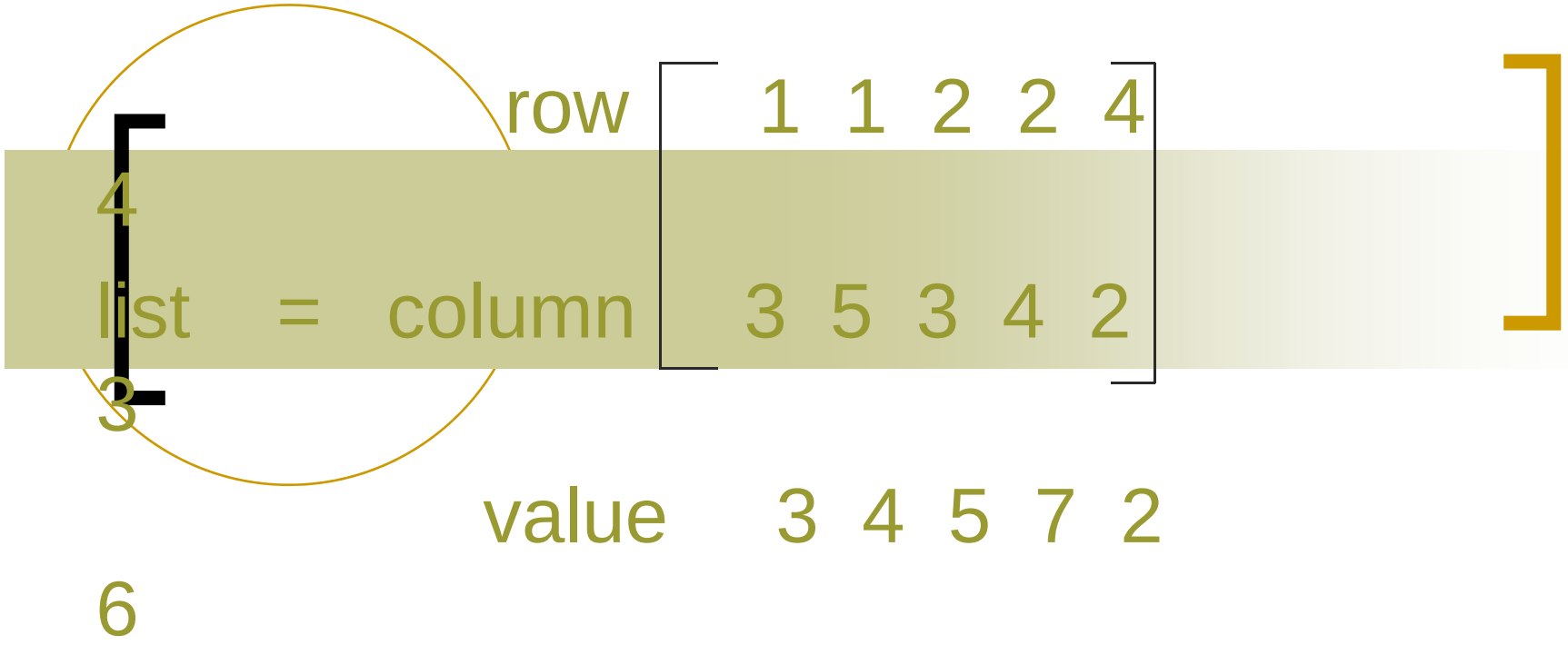
element	0	1	2	3	4	5
row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

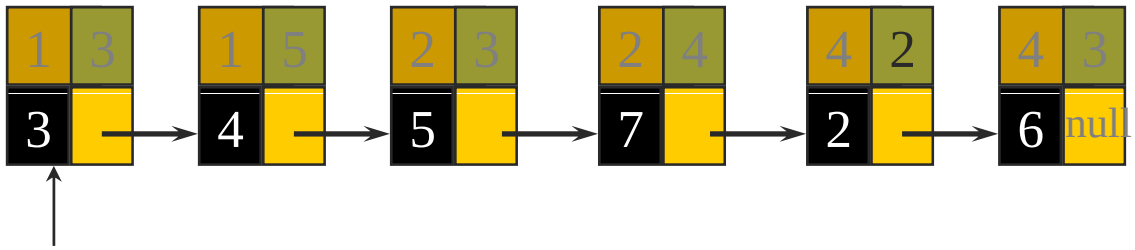
Chain Representation

ساختار نود



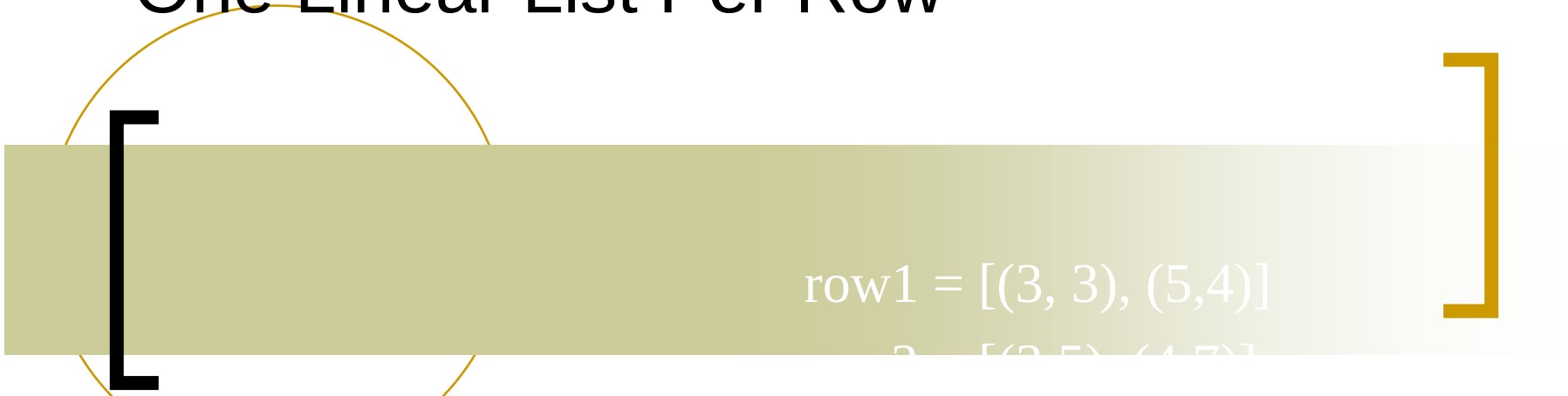
Single Chain





firstNode

One Linear List Per Row



0 0 3 0 4

0 0 5 7 0

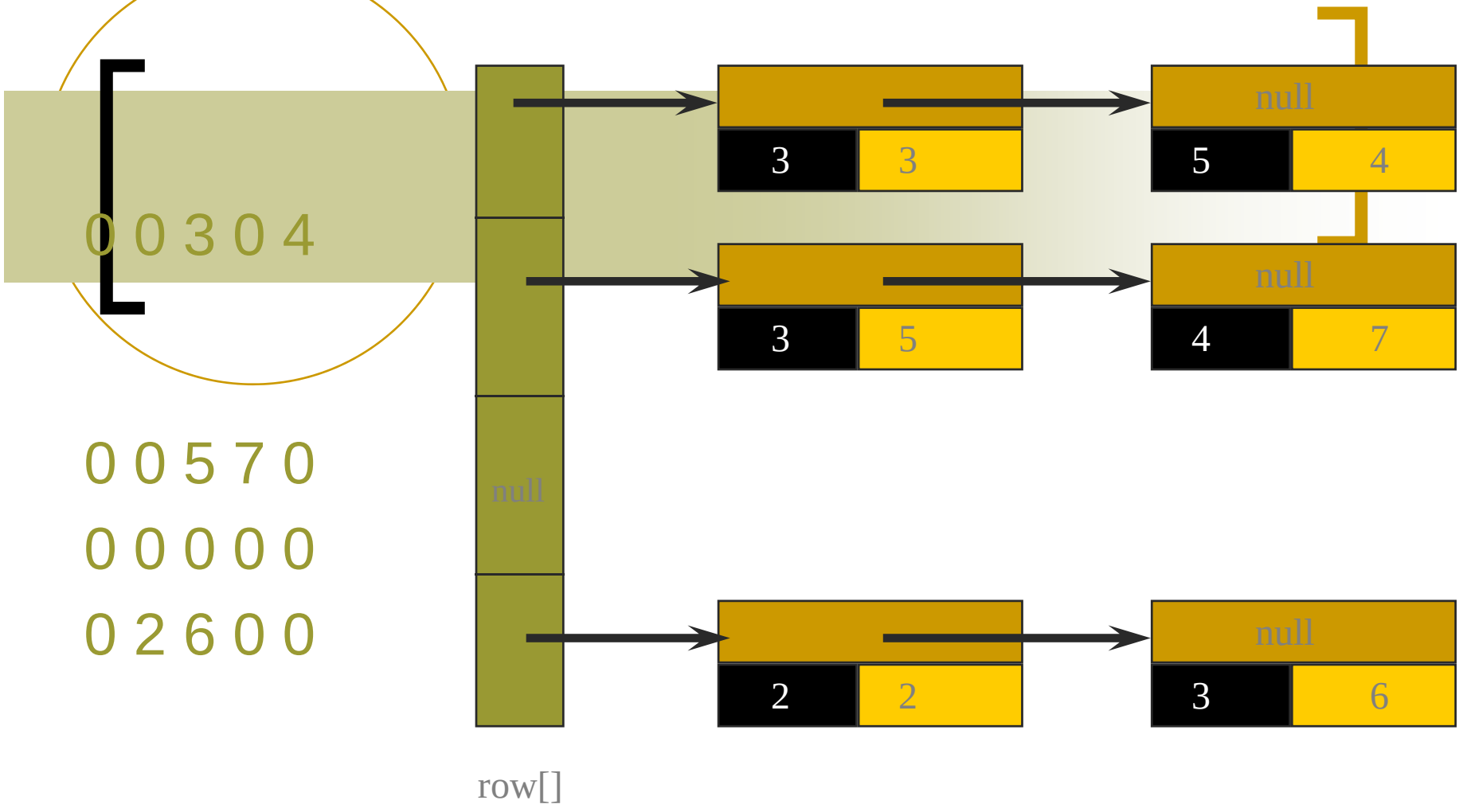
0 0 0 0 0

0 2 6 0 0

Array Of Row Chains

ساختار نود

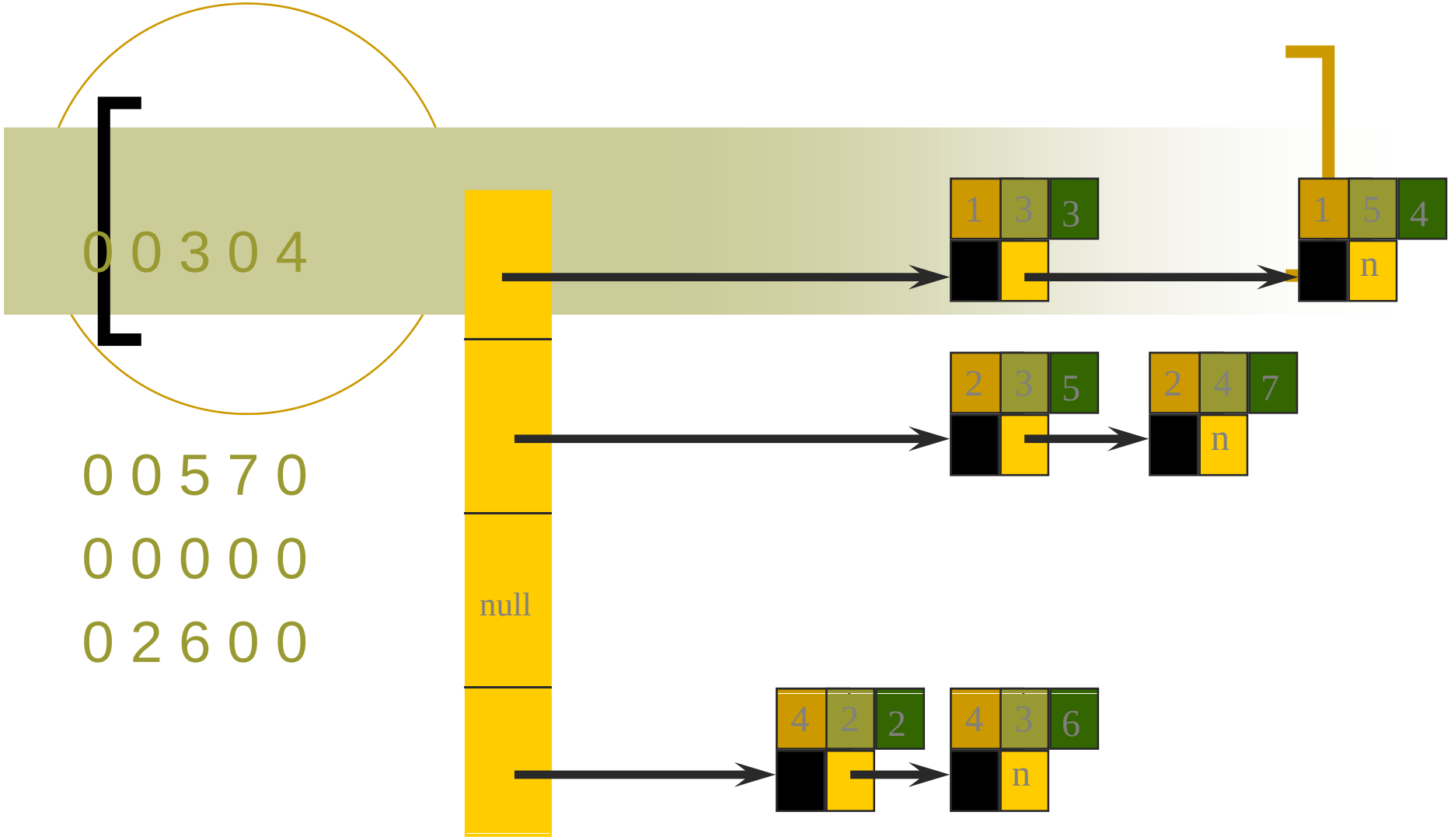
Array Of Row Chains



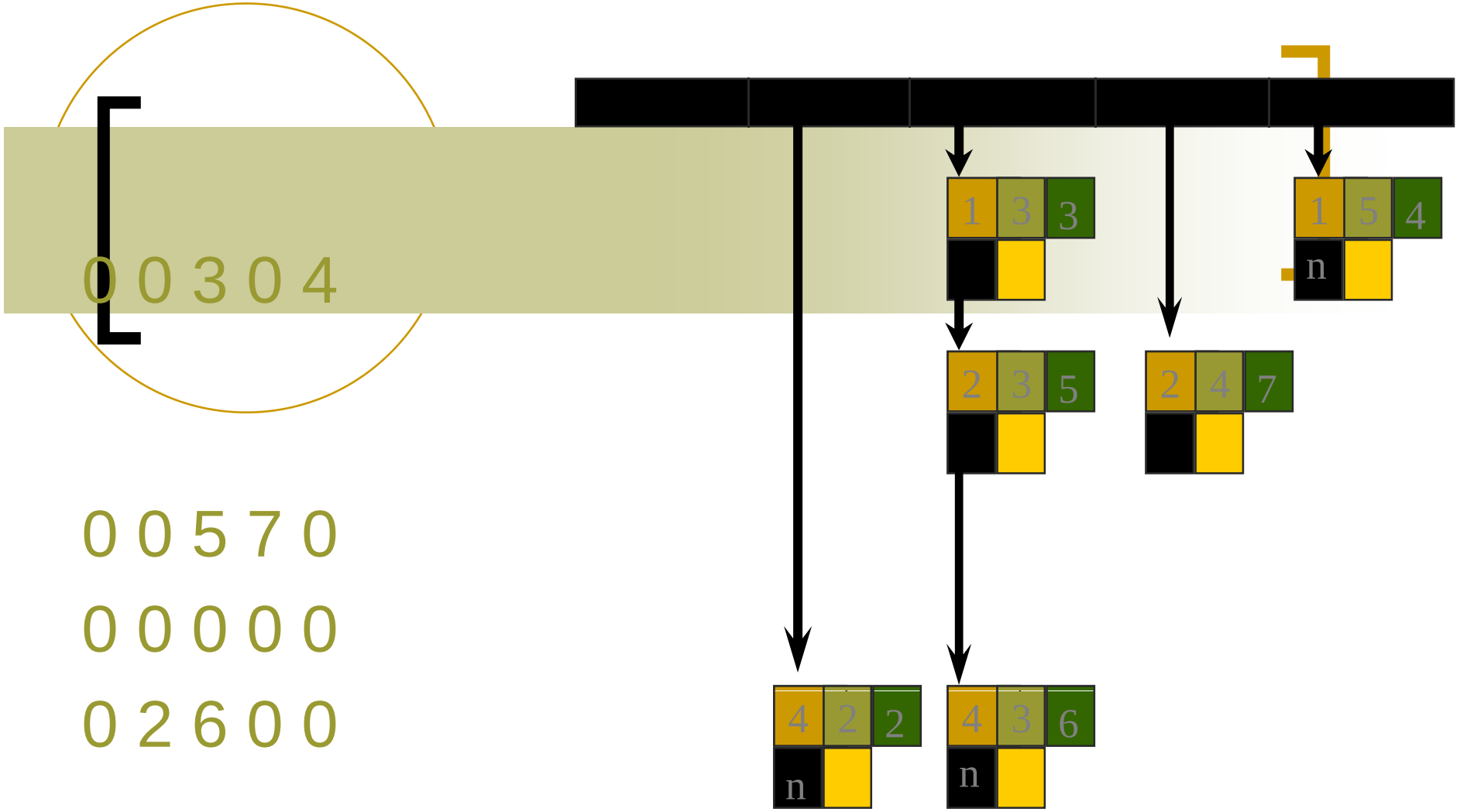
Orthogonal List Representation

Node structure.

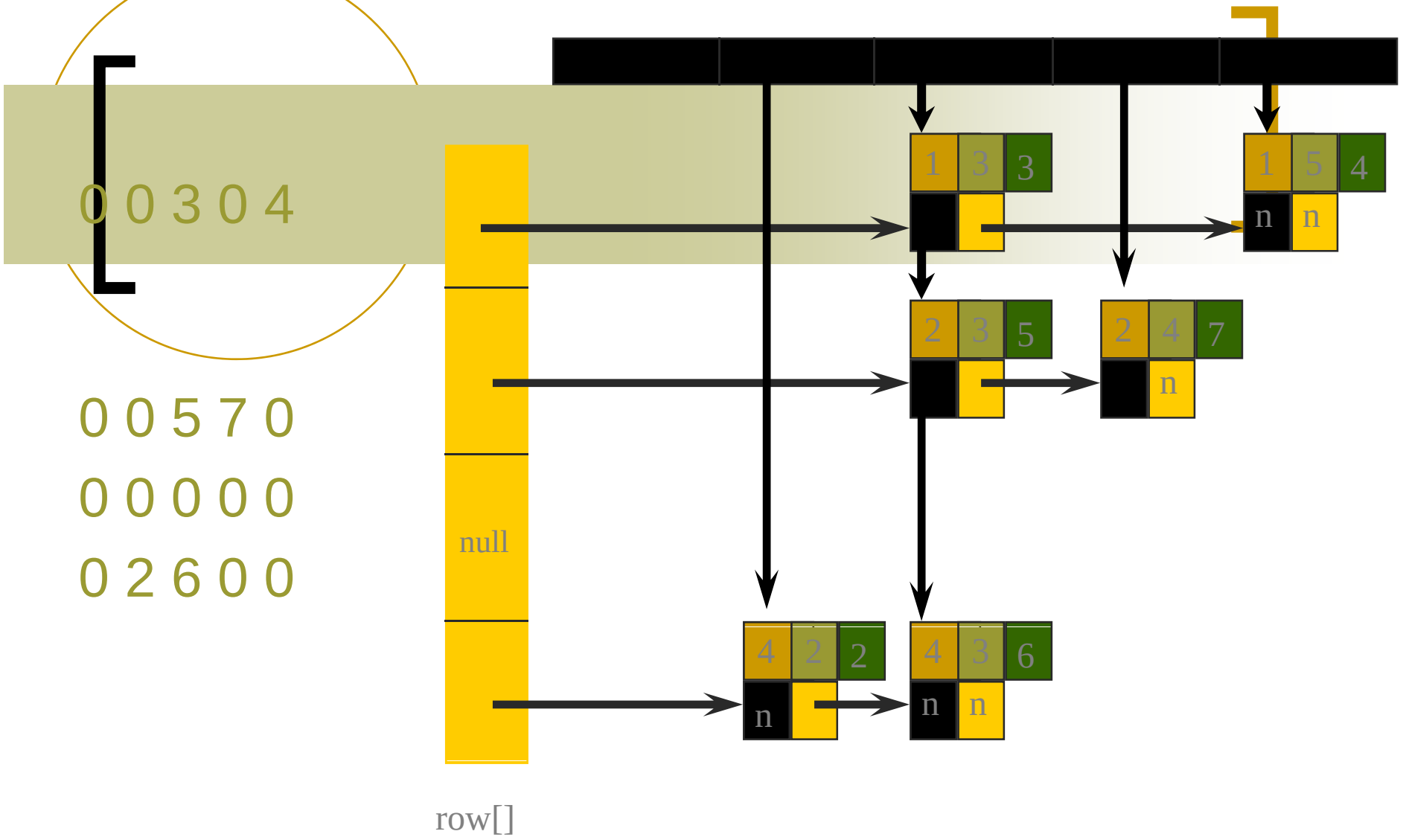
Row Lists



Column Lists

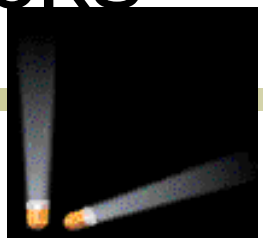


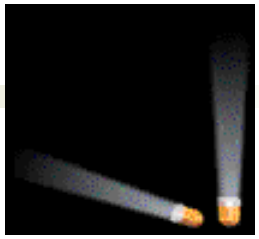
Orthogonal Lists



Variations

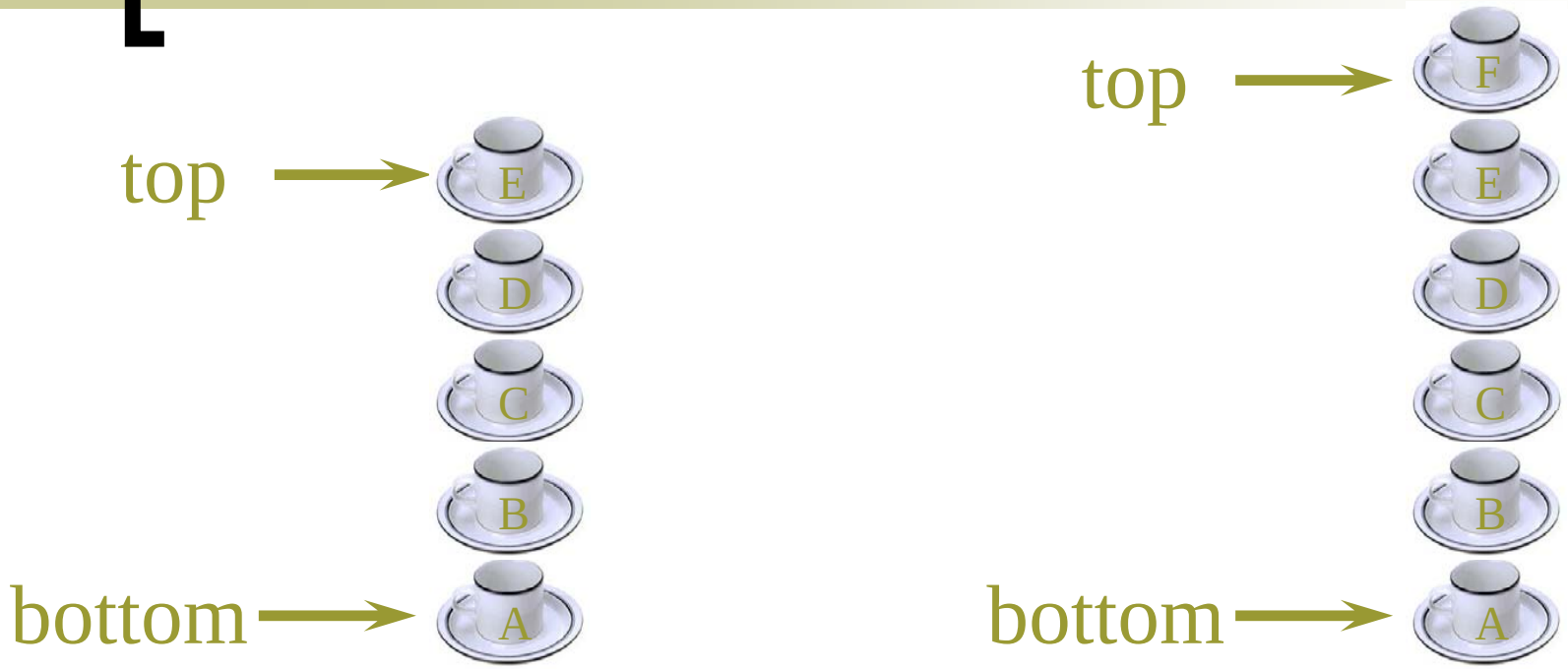
Stacks





- پشته نوعی لیست خطی میباشد.
- اولین عنصری که داخل پشته قرار میگیرد bottom نامیده می شود
- آخرین عنصری که داخل پشته قرار بگیرد top نامیده می شود
- عملیات حذف و درج فقط از top امکان پذیر می باشد

Stack Of Cups



- برای درج وحذف F باید از topعمل کرد.

- Stack=LIFO=LAST IN FIRST OUT

The Interface Stack

```
public interface Stack
                {
    public boolean empty();
    public Object peek();
public void push(Object theObject);
    public Object pop();
                }
```

Parentheses Matching

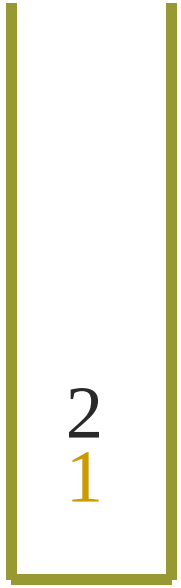
- ((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)
  - Output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at v.
    - (2,6) (1,13) (15,19) (21,25) (27,31) (34,38)
- a+b)*((c+d)
  -
  - right parenthesis at 5 has no matching left parenthesis
  - (8,12)
  - left parenthesis at 7 has no matching right parenthesis

Parentheses Matching

- عبارت را از چپ به راست بخوانید.
- وقتی به Left parenthesis رسید آن رادر پشته push کنید.
- وقتی به right parenthesis رسید آن رااز پشته pop کنید.

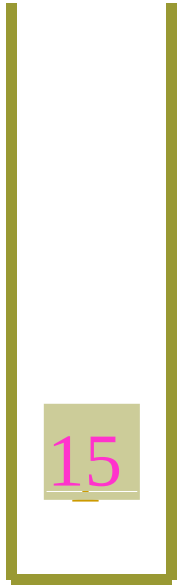
Example

((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)



Example

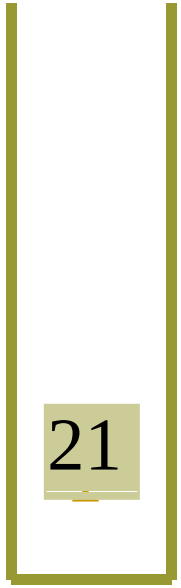
((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)



(2,6) (1,13)

Example

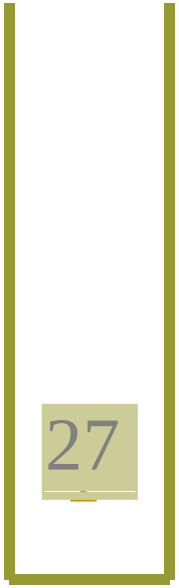
((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)



(2,6) (1,13) (15,19)

Example

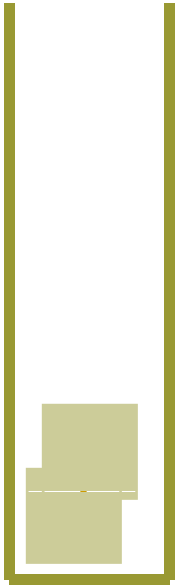
((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)



(2,6) (1,13) (15,19) (21,25)

Example

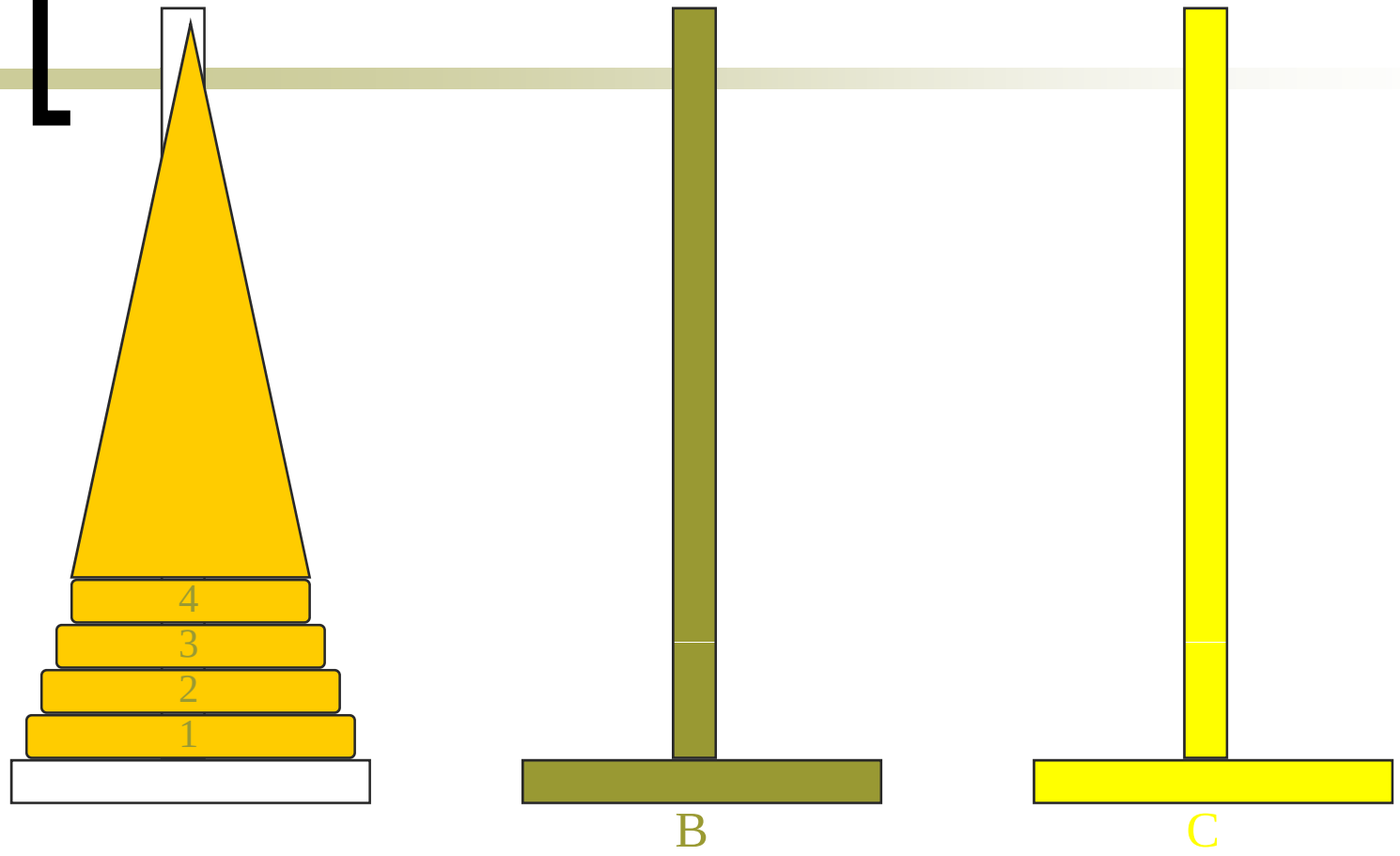
((a+b)*c+d-e)/(f+g)-(h+j)*(k-l) /(m-n)



(2,6) (1,13) (15,19) (21,25)(27,31)

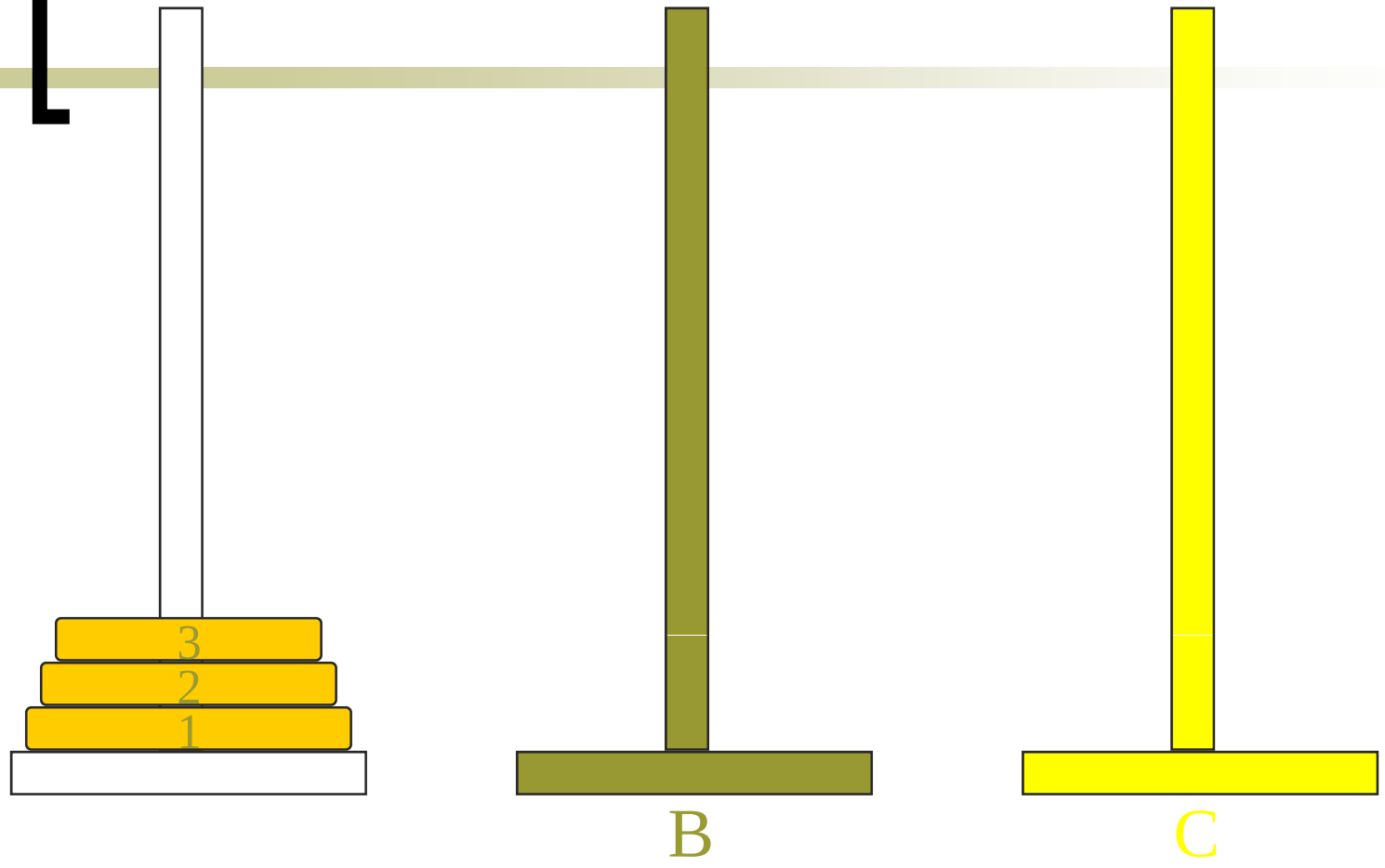
- and so on

Towers Of Hanoi/Brahma



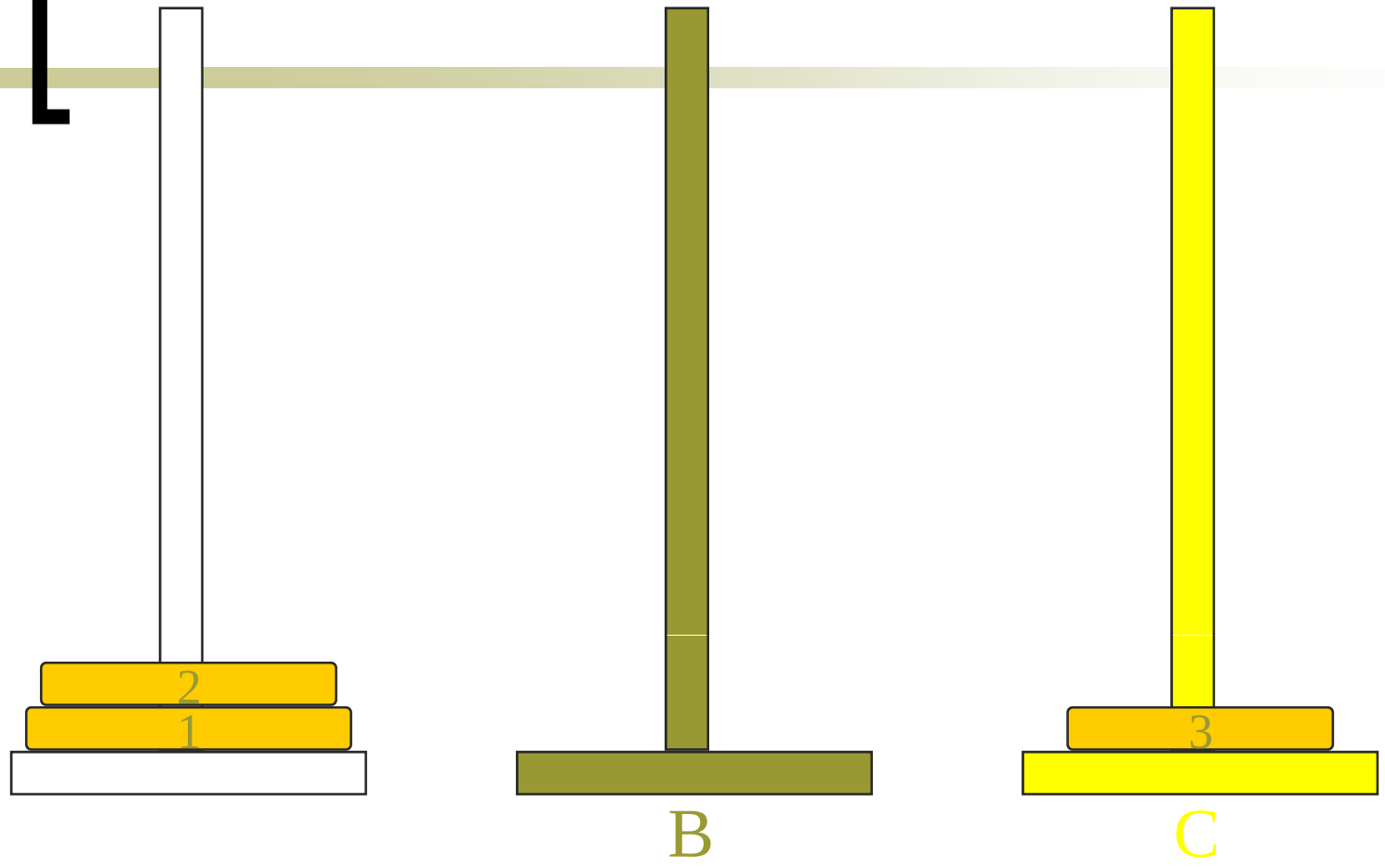
- ۶۴ حلقه طلا از ستون A به C انتقال داده می شوند.
- هر ستون در آن مانند یک پشته عمل می کند.
- و هیچ حلقه بزرگی روی کوچکتر از خودش قرار نمی گیرد.

Towers Of Hanoi/Brahma



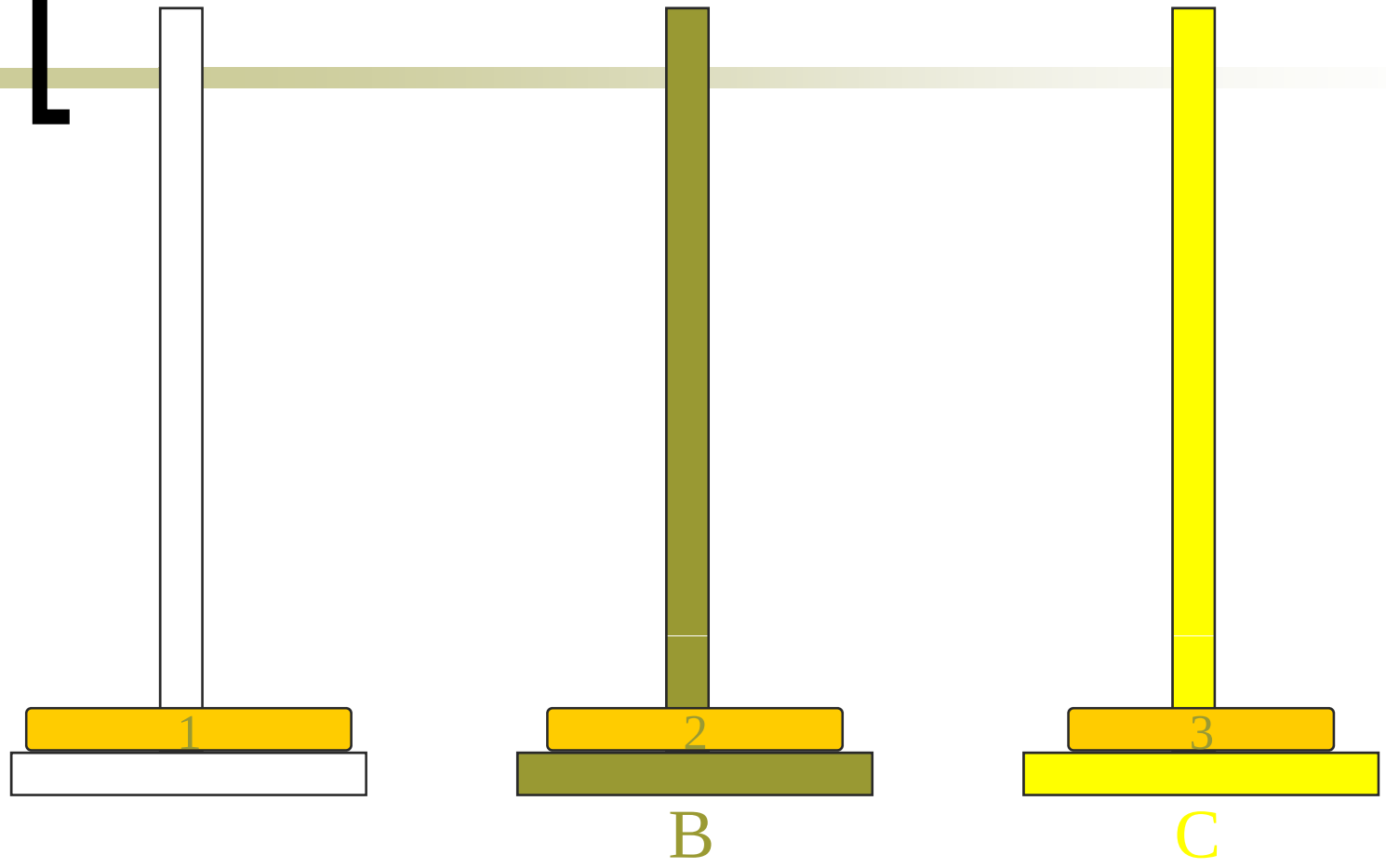
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



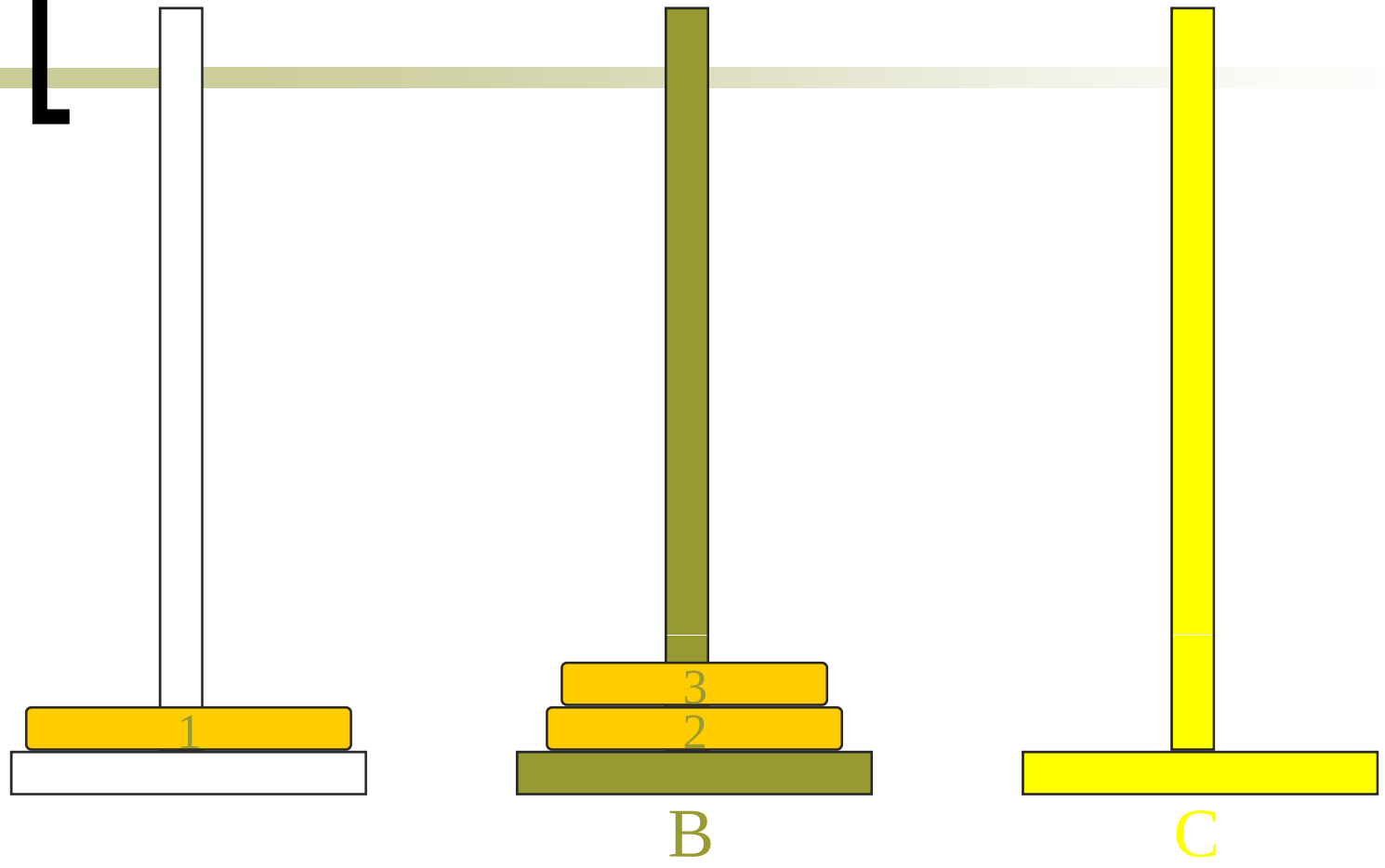
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



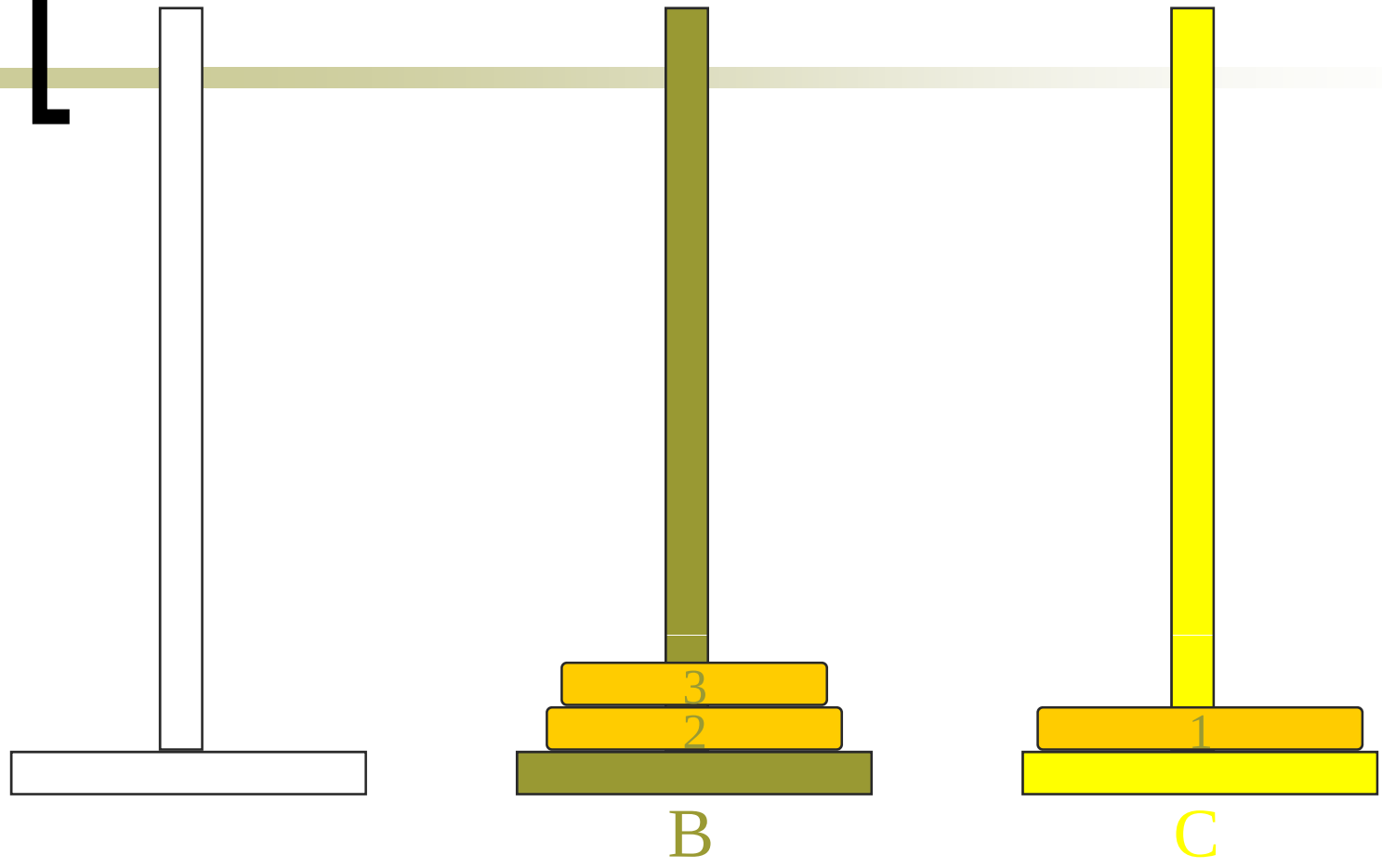
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



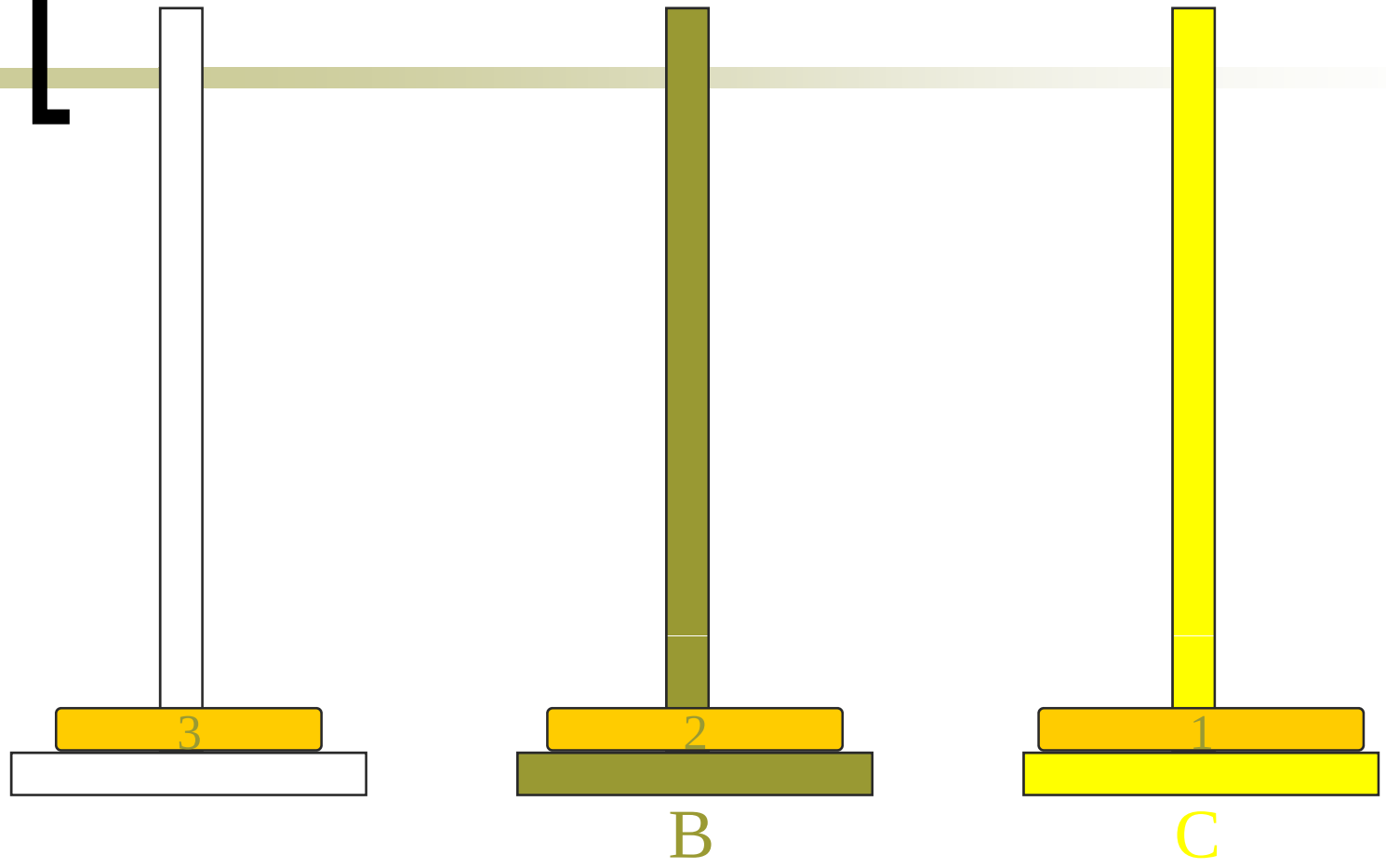
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



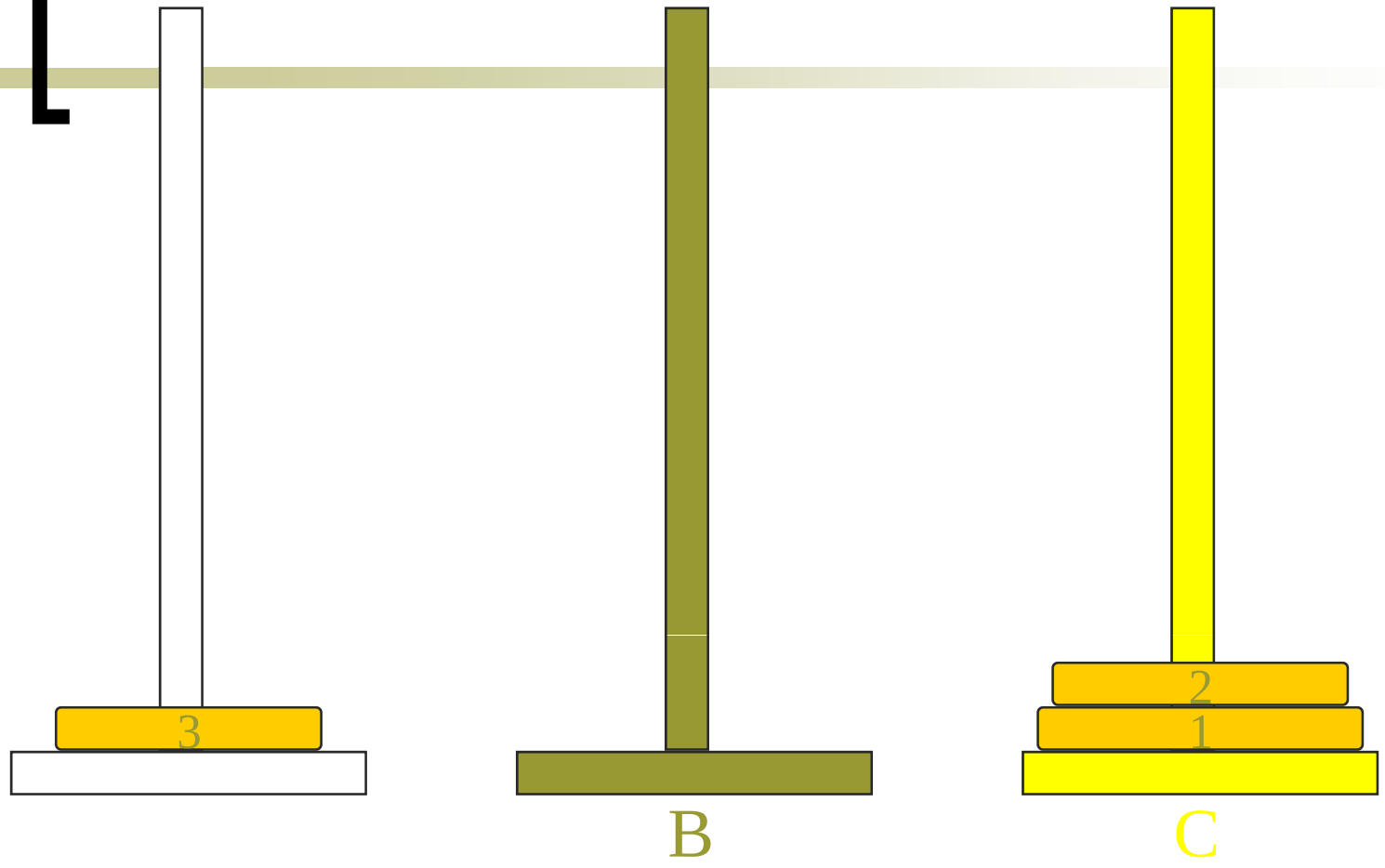
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



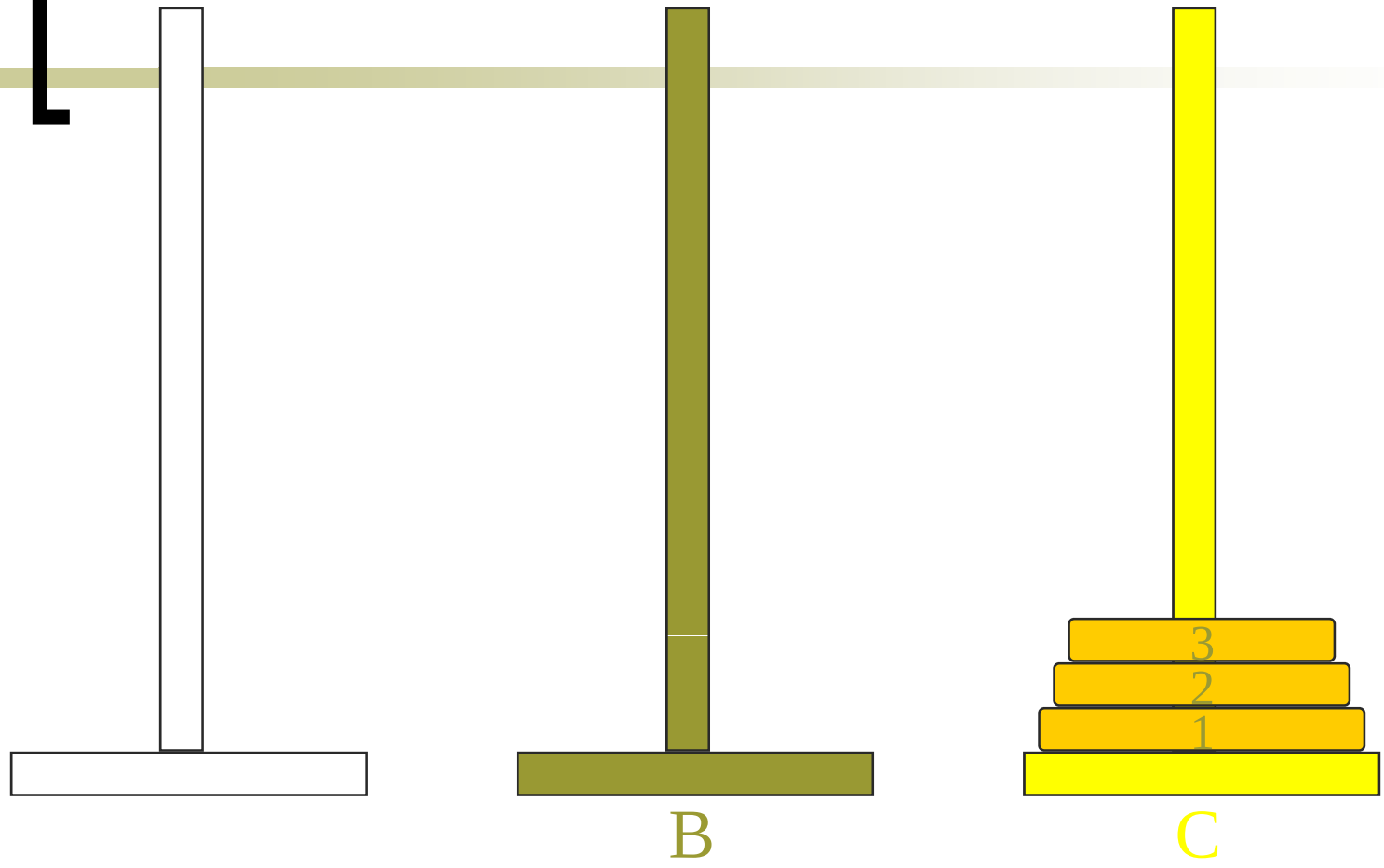
3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



3-disk Towers Of Hanoi/Brahma

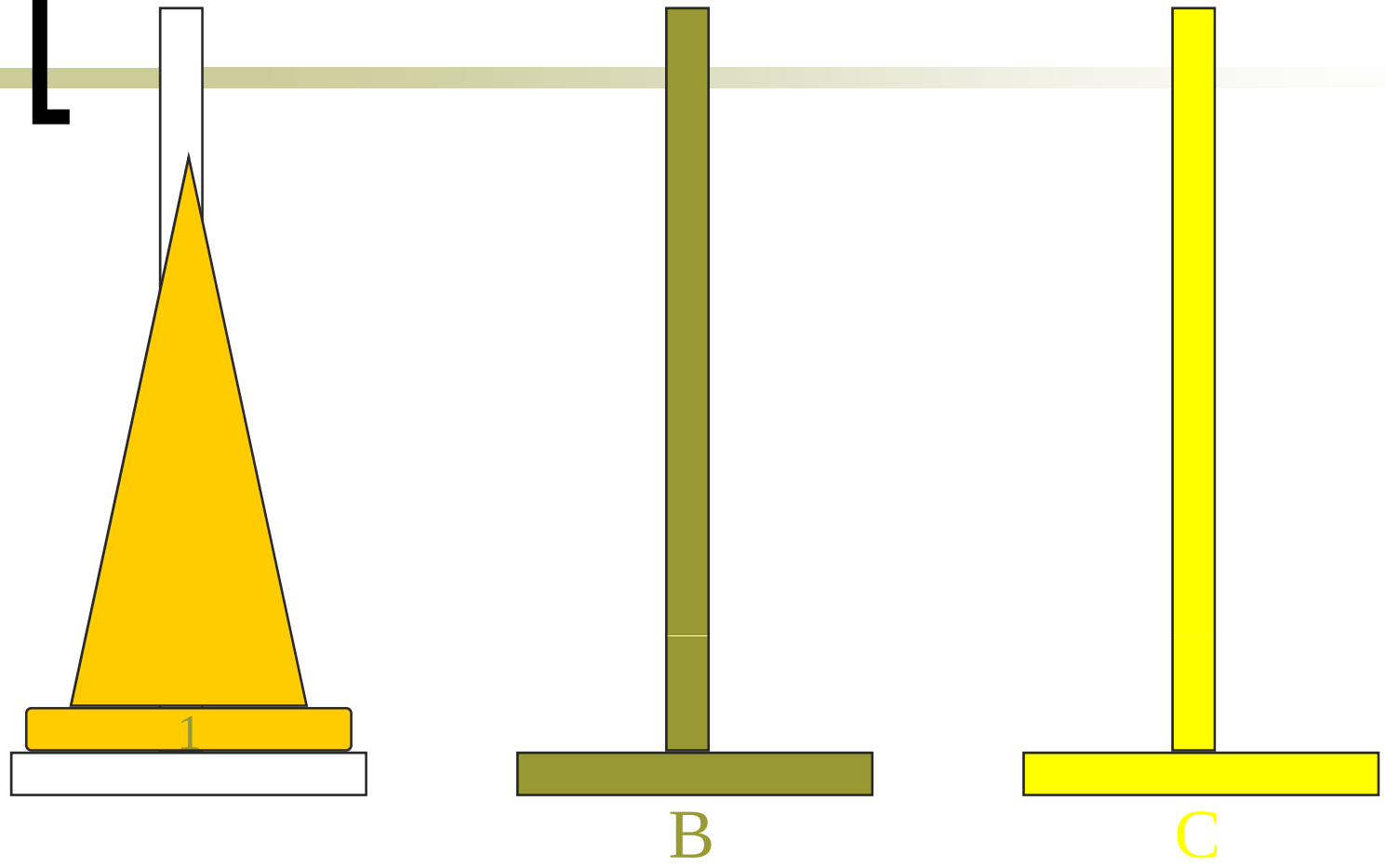
Towers Of Hanoi/Brahma



3-disk Towers Of Hanoi/Brahma

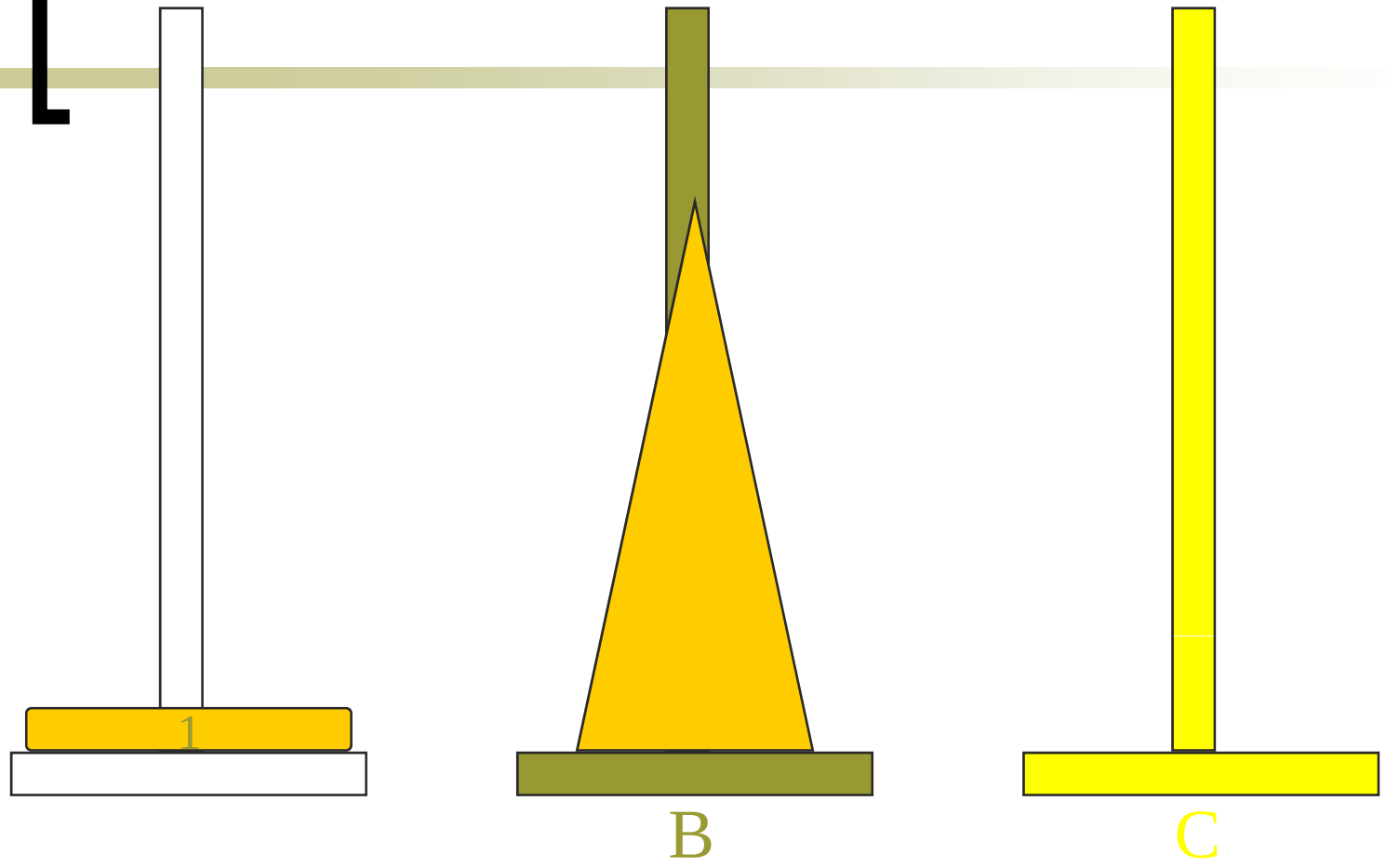
- 7 disk moves

Recursive Solution



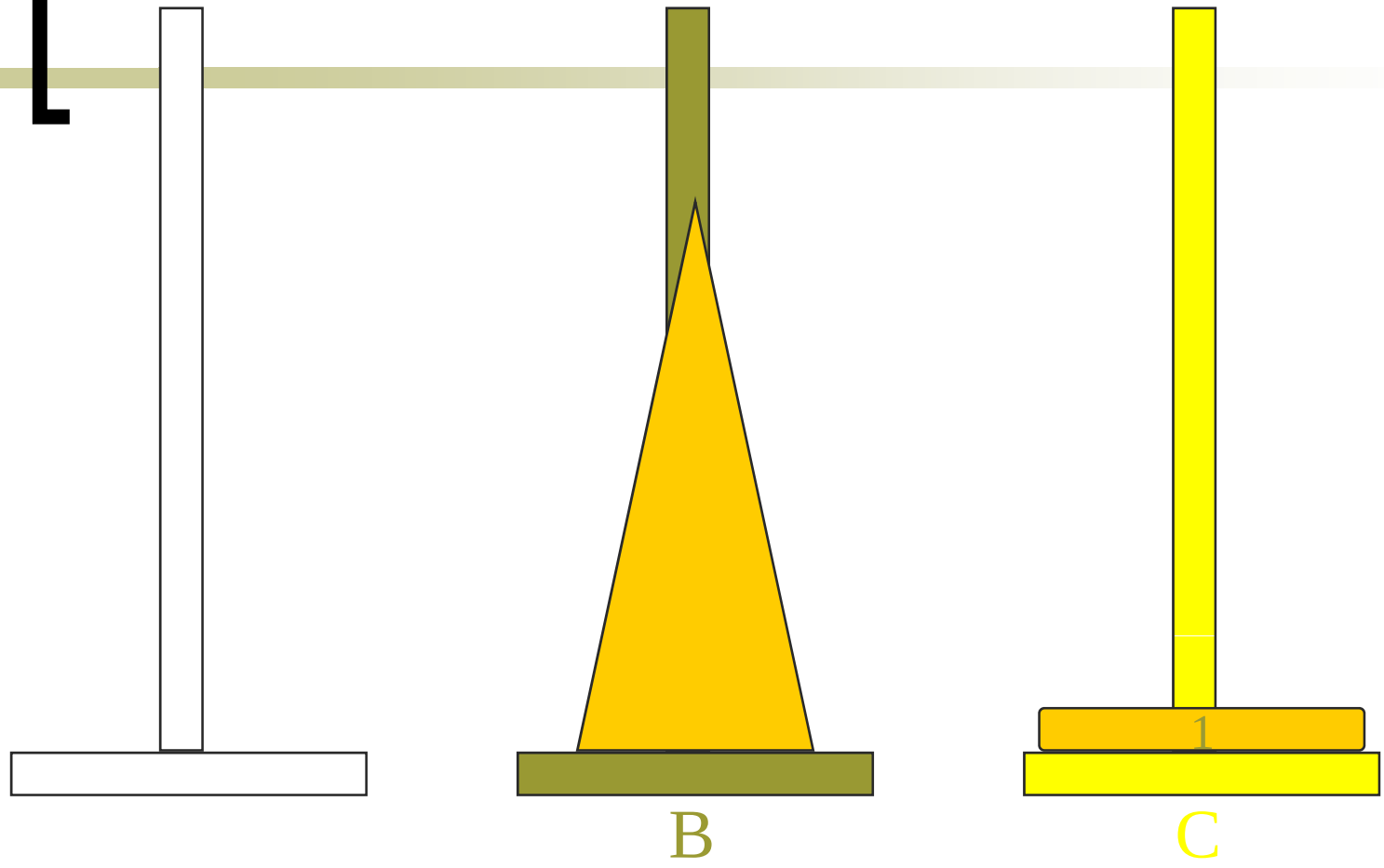
- با استفاده از مساله برج هانوی $n > 0$ حلقه را میتوان از A به C با استفاده از B منتقل کرد.
- با استفاده از C تعداد $n-1$ حلقه را از Aبه B انتقال داده.

Recursive Solution



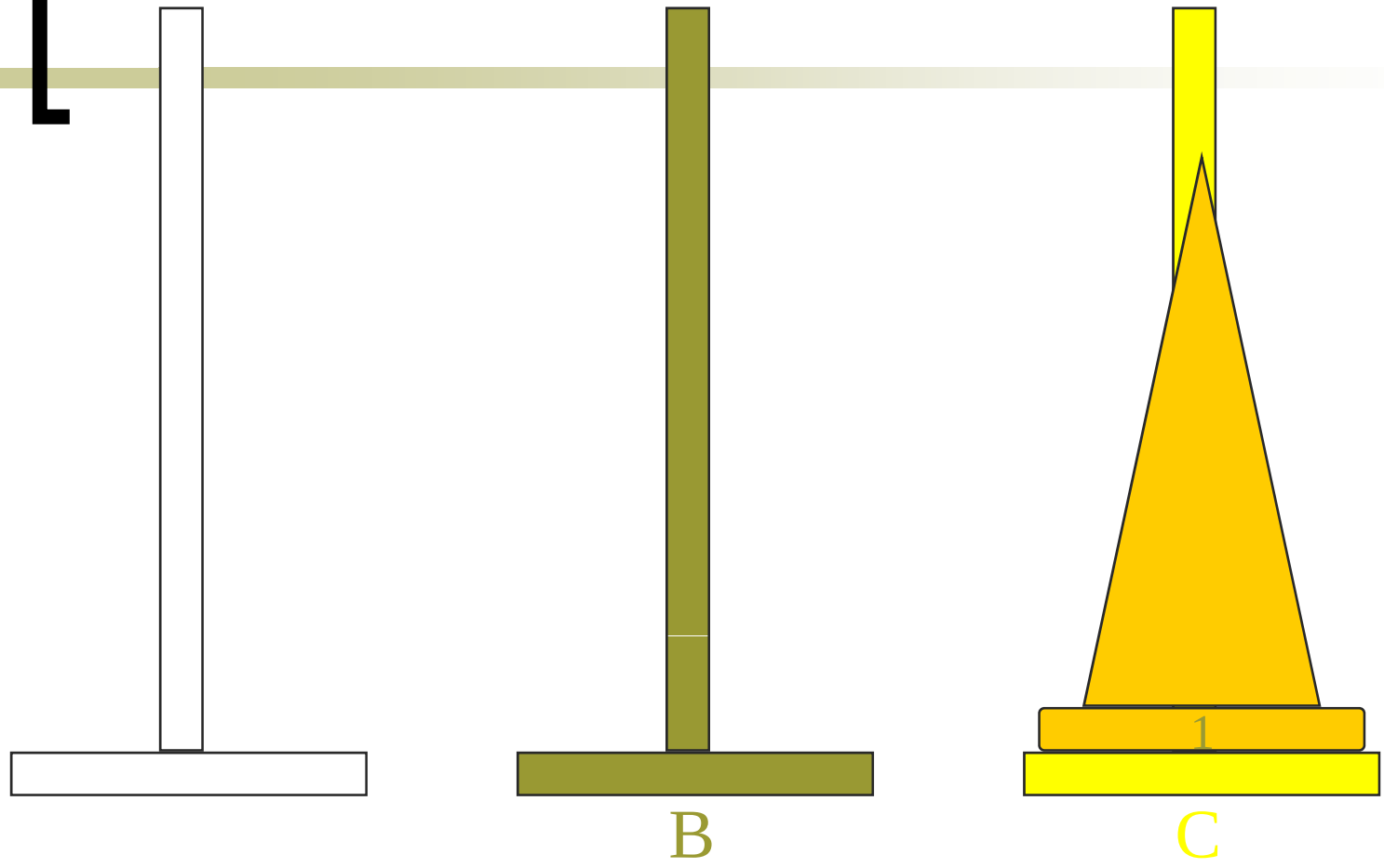
- حلقه اول ستون Aرا از Aبه Cانتقال دهید.

Recursive Solution



- n-1 حلقه را از B به C انتقال دهید.

Recursive Solution



- moves(n) = 0 when n = 0
- moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n >

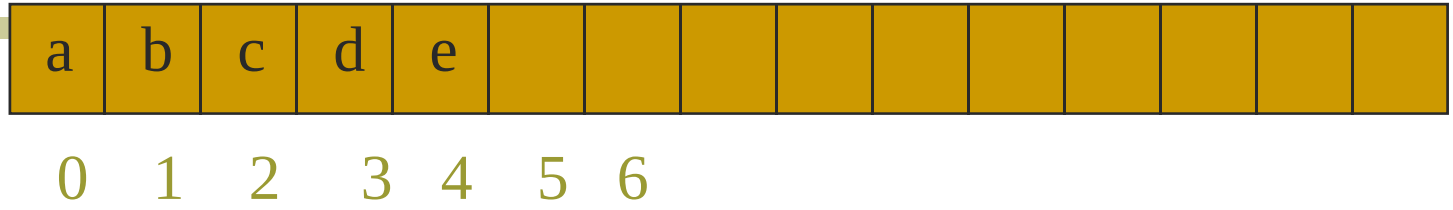
Stacks

```
public interface Stack
                {
    public boolean empty();
    public Object peek();
public void push(Object theObject);
    public Object pop();
                }
```

Derive From A Linear List Class

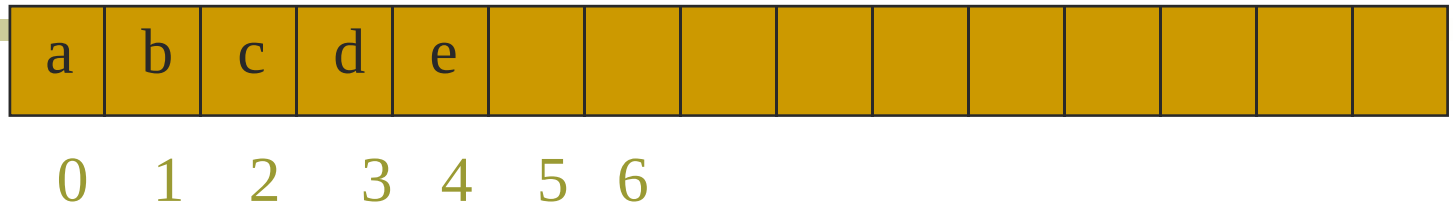
- ArrayLinearList
- Chain

Derive From ArrayLinearList



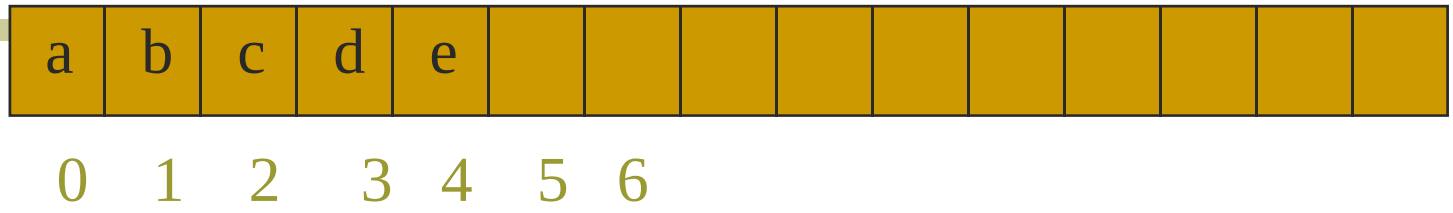
- عنصر بالای پشته معادل چپ ترین ویا راست ترین عنصر داخل آرایه می باشد.
- empty() => isEmpty()
  - O(1) time
- peek() => get(0) or get(size() - 1)
  - O(1) time

Derive From ArrayLinearList



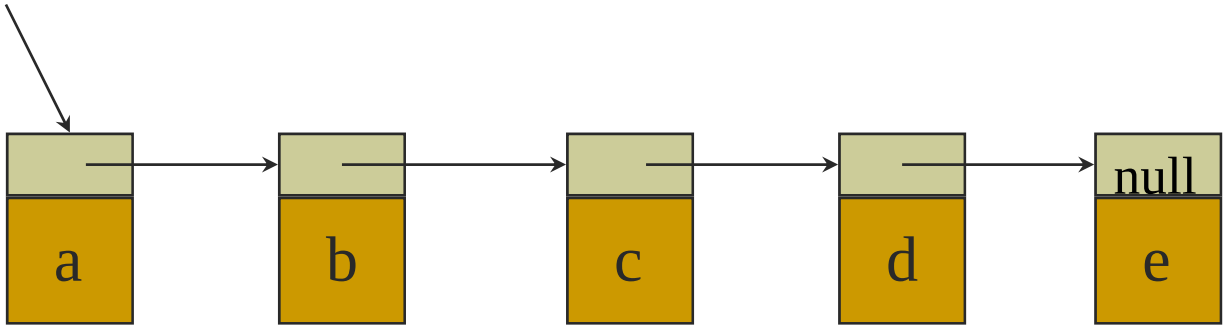
- وقتی که top در موقعیت چپ ترین عنصر آرایه قراردارد
  - push(theObject) => add(0, theObject)
    - O(size) time
  - pop() => remove(0)
    - O(size) time

Derive From ArrayLinearList



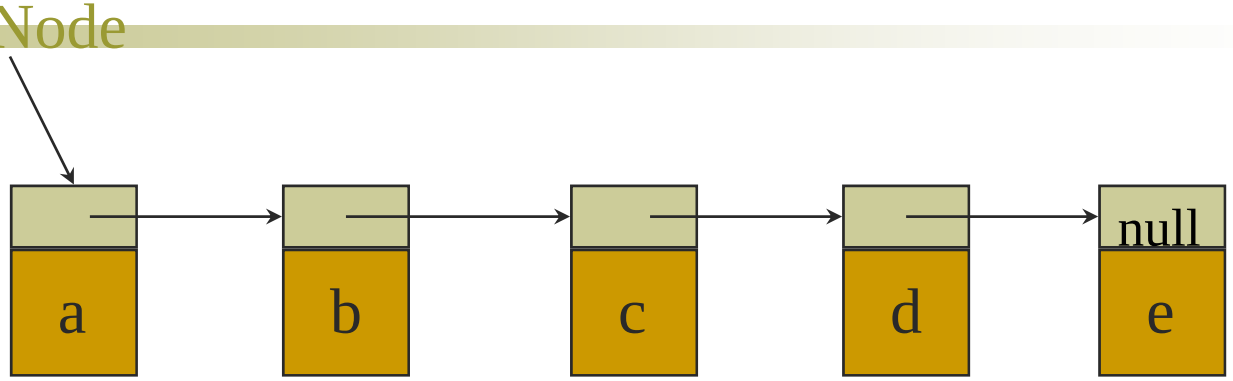
- وقتی که top در موقعیت راست ترین عنصر آرایه قراردارد
  - push(theObject) => add(size(), theObject)
    - O(1) time
  - pop() => remove(size()-1)
    - O(1) time
- use right end of list as top of stack

Derive From Chain



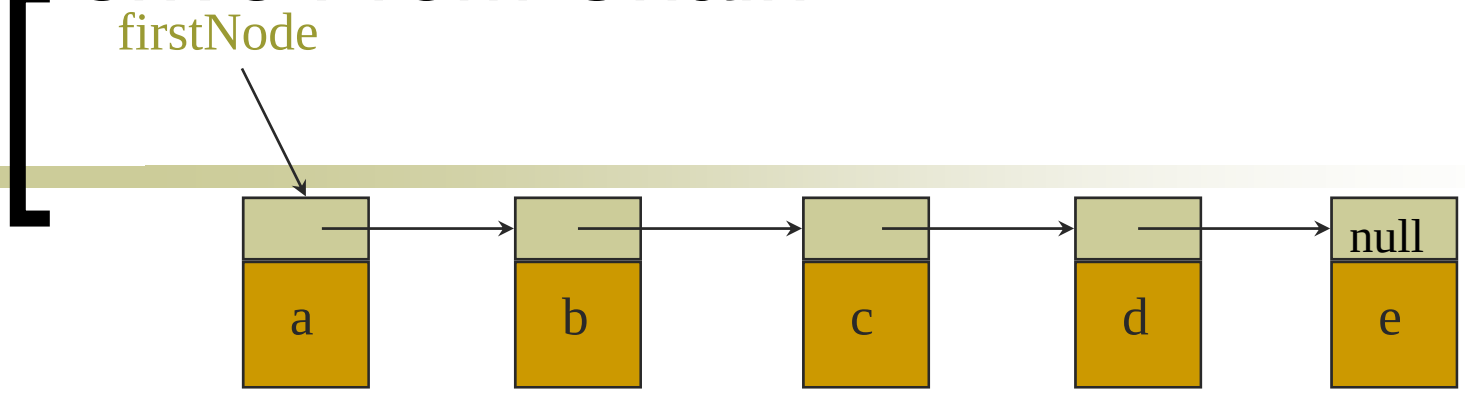
- عنصر بالای پشته معادل چپ ترین ویا راست ترین عنصر داخل آرایه می باشد.
- empty() => isEmpty()
  - O(1) time

Derive From Chain



- وقتی که top در موقعیت چپ ترین عنصر آرایه قراردارد
  - peek() => get(0)
  - O(1) time
  - push(theObject) => add(0, theObject)
  - O(1) time
  - pop() => remove(0)
  - O(1) time

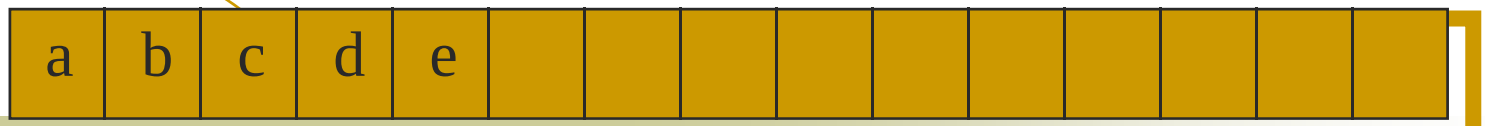
Derive From Chain



- وقتی که top در موقعیت راست ترین عنصر آرایه قراردارد
  - peek() => get(size() - 1)
  - O(size) time
  - push(theObject) => add(size(), theObject)
  - O(size) time
  - pop() => remove(size()-1)
  - O(size) time

– use left end of list as top of stack

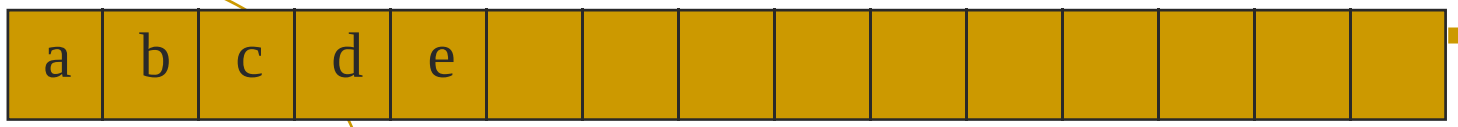
empty() And peek()




```
public boolean empty()
{return isEmpty();}
```

```
public Object peek()
{
  if (empty())
     throw new EmptyStackException();
  return get(size() - 1)
}
```

push(theObject) And pop()




```
public void push(Object theElement)
  {add(size(), theElement);}

public Object pop()
{
  if (empty())
     throw new EmptyStackException();
  return remove(size() - 1);
}
```

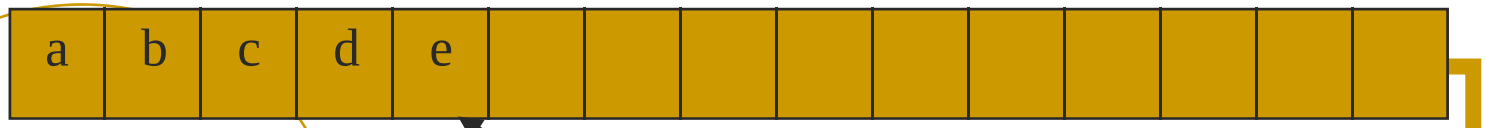
A Faster pop()

```
if (empty())
    throw new EmptyStackException();
return remove(size() - 1);
```

vs.

```
try {return remove(size() - 1);}
catch(IndexOutOfBoundsException e)
   {throw new EmptyStackException();}
```

push(...)




```
public void push(Object theElement)
{
  // increase array size if necessary
   if  (top == stack.length - 1)
       stack = ChangeArrayLength.changeLength1D
               (stack, 2  * stack.length);
  // put theElement at the top of the stack
  stack[++top] = theElement;
}
```

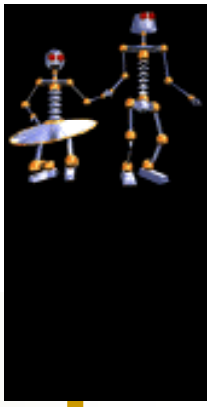
pop()


```
public Object pop()
{
  if (empty())
     throw new EmptyStackException();
  Object topElement = stack[top];
  stack[top--] = null;   // enable garbage collection
  return topElement;
}
```

Linked Stack From Scratch

- See text.

Performance



500,000 pop, push, and peek operations

Class	initial capacity 10	500,000
ArrayStack	0.44s	0.22s
DerivedArrayStack	0.60s	0.38s
DerivedArrayStackWithCatch	0.55s	0.33s
java.util.Stack	1.15s	-
DerivedLinkedStack	3.20s	3.20s
LinkedStack	2.96s	2.96s

Queues

- یک لیست خطی می باشد.
- عنصر جلوی صف front نامیده میشود.
- عنصر انتهای لیست rear نامیده میشود.
- افزودن عنصر به لیست فقط ازانتهای صف" rear " امکان پذیر است.
- حذف کردن عناصراز لیست از جلوی صف "front"امکان پذیر است.

Bus Stop Queue

front rear

Bus Stop Queue

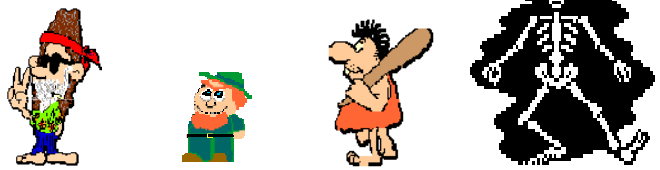
front

rear

Bus Stop Queue

front rear

Bus Stop Queue

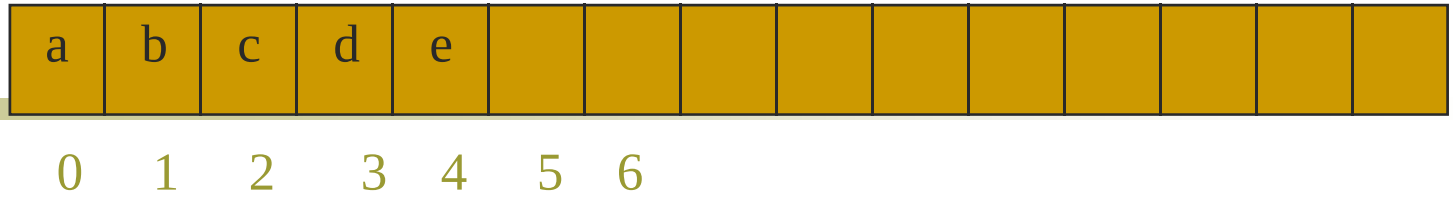


front rear

The Interface Queue

```
public interface Queue
                {
    public boolean isEmpty();
public Object getFrontEelement();
public Object getRearEelement();
public void put(Object theObject);
    public Object remove();
                }
```

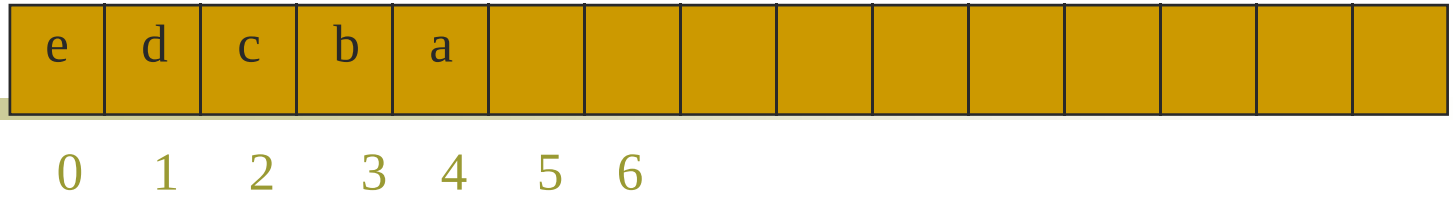
Derive From ArrayLinearList



زمانی که frontچپ لیست و rear در راست لیست باشد.

- Queue.isEmpty() => super.isEmpty()
  - O(1) time
- getFrontElement() => get(0)
  - O(1) time
- getRearElement() => get(size() - 1)
  - O(1) time
- put(theObject) => add(size(), theObject)
  - O(1) time
- remove() => remove(0)
  - O(size) time

Derive From ArrayLinearList

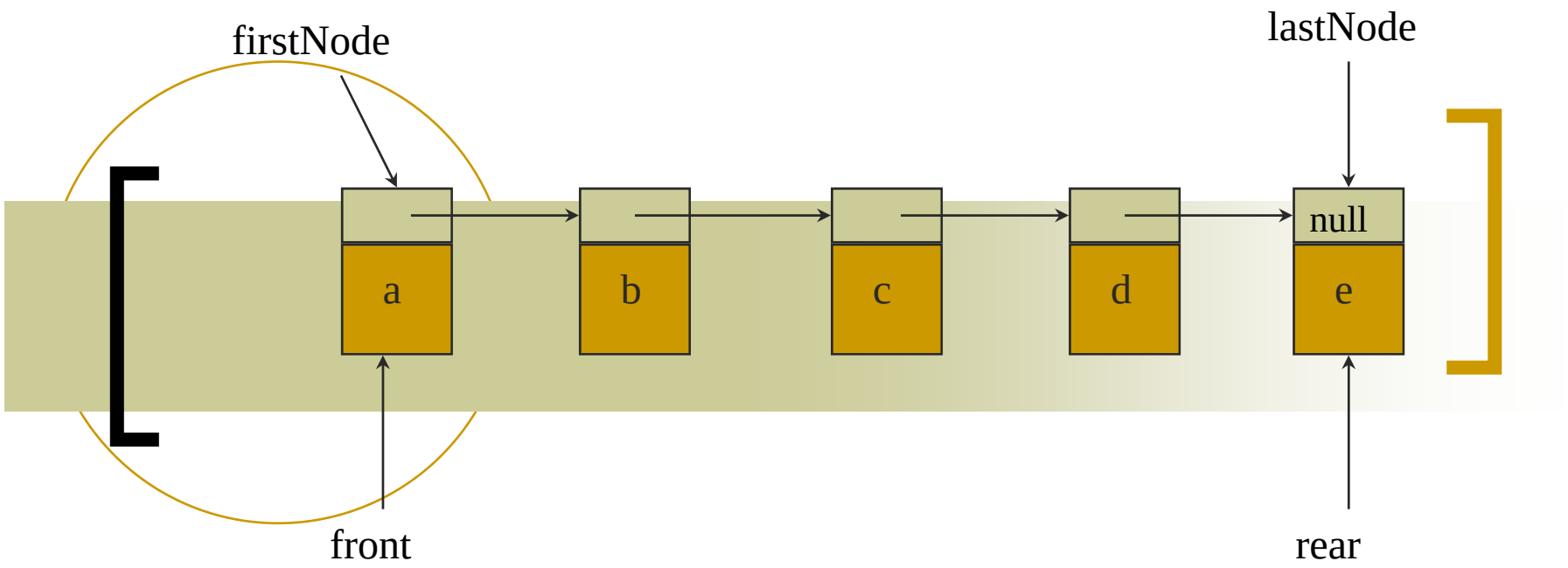


- زمانی که frontراست لیست و rear درچپ لیست باشد.
  - Queue.isEmpty() => super.isEmpty()
    - O(1) time
  - getFrontElement() => get(size() - 1)
    - O(1) time
  - getRearElement() => get(0)
    - O(1) time
  - put(theObject) => add(0, theObject)
    - O(size) time
  - remove() => remove(size() - 1)
    - O(1) time

Derive From ArrayLinearList

- عملیات درج وحذف در صف با O(۱) صورت می گیرد.

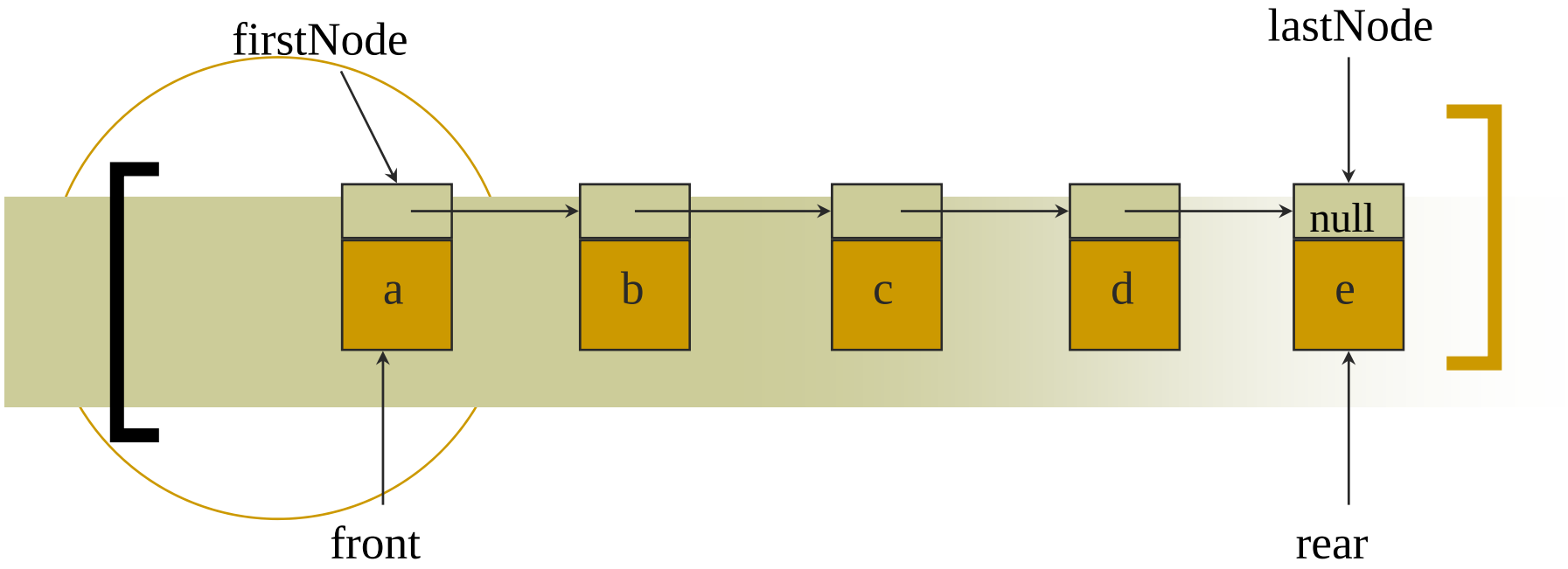
Derive From ExtendedChain



وقتی که frontچپ لیست و rearراست لیست قرار دارد

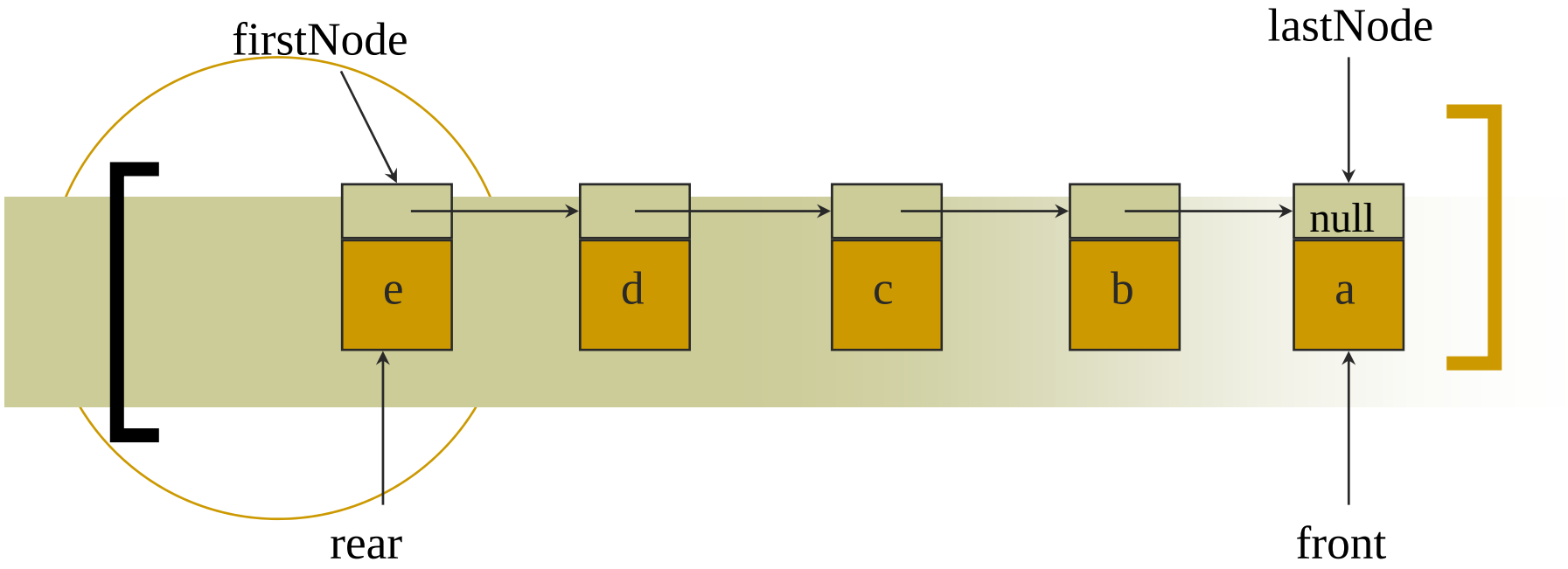
- Queue.isEmpty() => super.isEmpty()
  - O(1) time
- getFrontElement() => get(0)
  - O(1) time

Derive From ExtendedChain



- getRearElement() => getLast() … new method
  - O(1) time
- put(theObject) => append(theObject)
  - O(1) time
- remove() => remove(0)
  - O(1) time

Derive From ExtendedChain



وقتی که frontراست لیست و rearچپ لیست قرار دارد

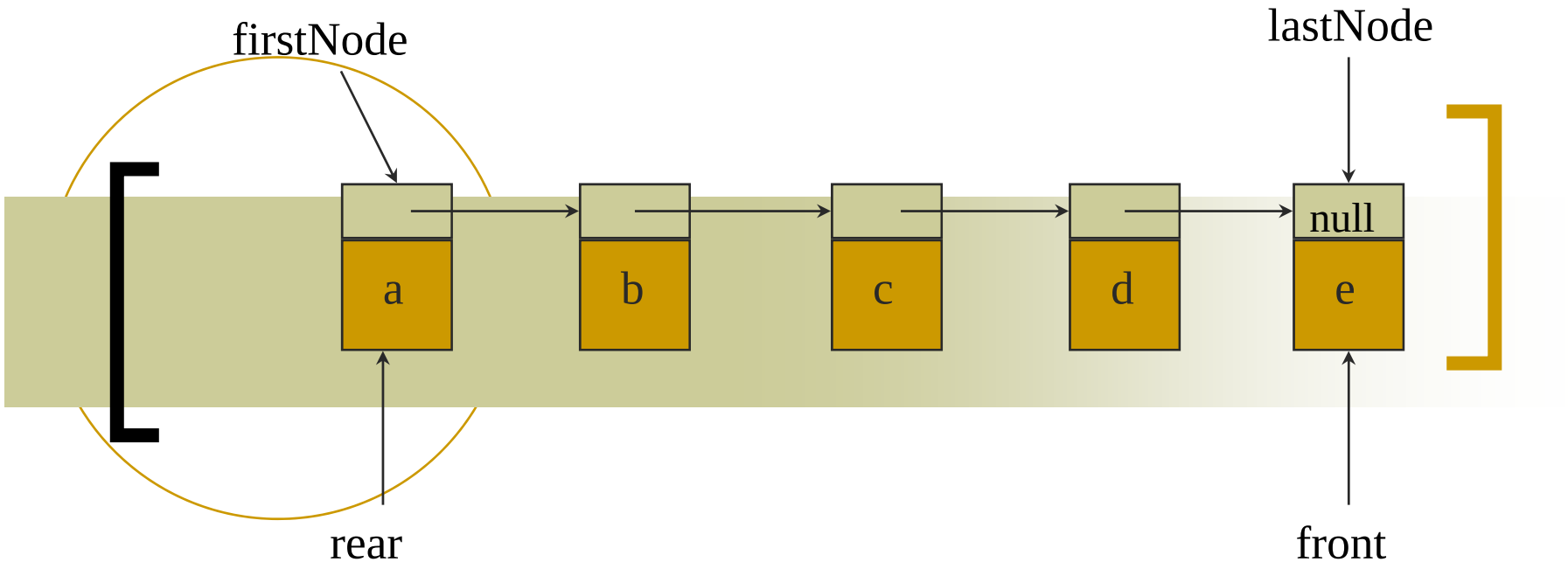
Queue.isEmpty() => super.isEmpty()

– O(1) time

- getFrontElement() => getLast()

– O(1) time

Derive From ExtendedChain



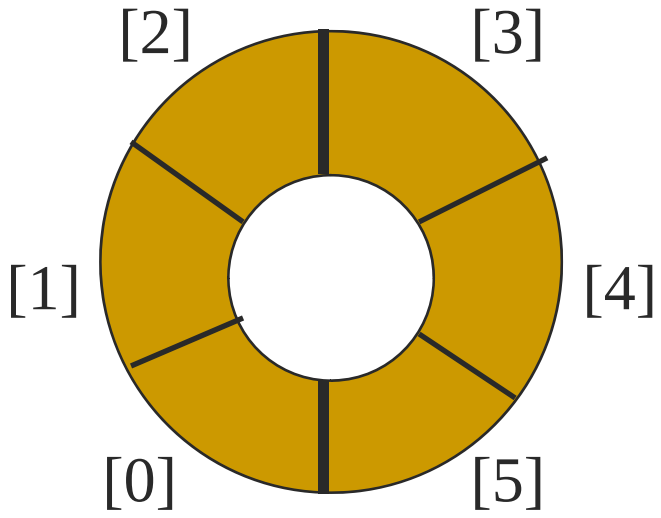
- getRearElement() => get(0)
  - O(1) time
- put(theObject) => add(0, theObject)
  - O(1) time
- remove() => remove(size-1)
  - O(size) time

Custom Array Queue

- کاربرد یک آرایه یک بعدی بعنوان صف

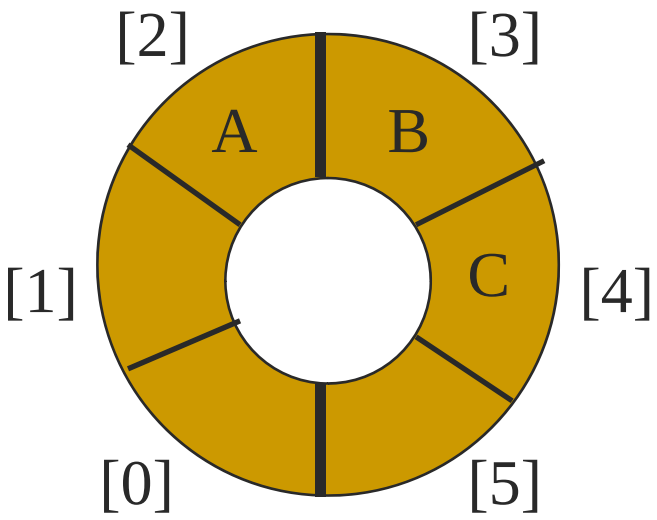
queue[]

- کاربرد یک آرایه یک بعدی بعنوان صف حلقوی



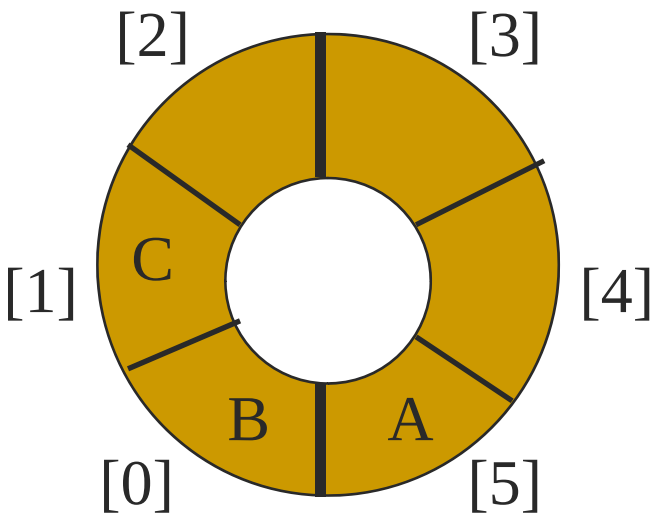
Custom Array Queue

- صف حلقوی با ۳ عنصر پر شده است.



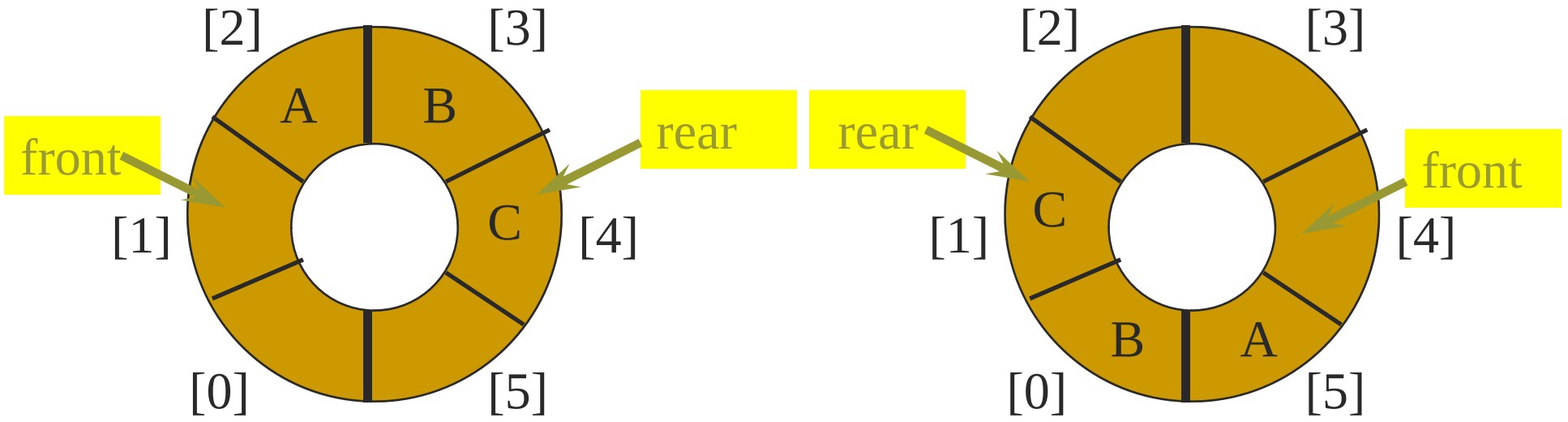
Custom Array Queue

- حالت دیگری از قرار گیری ۳ عنصر در لیست **حلقوی**



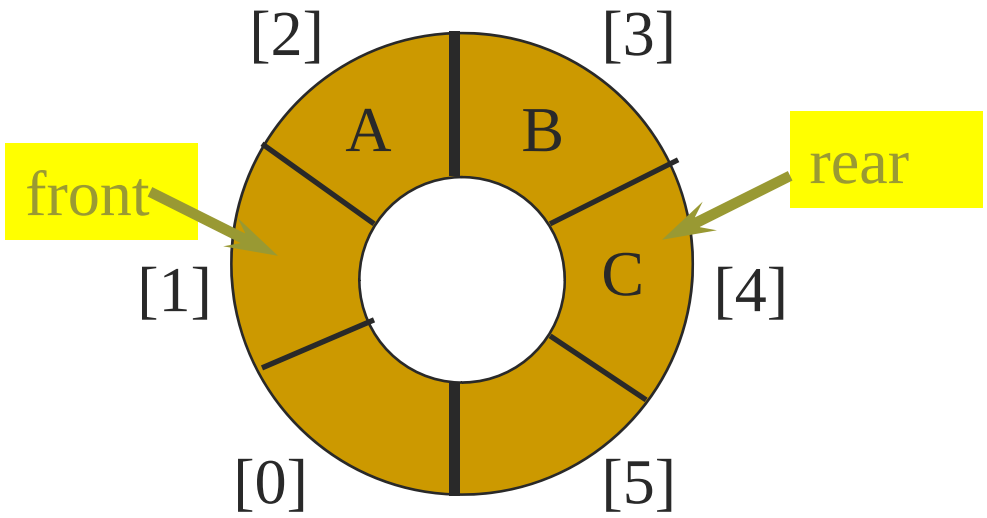
Custom Array Queue

- کاربرد front and rear در لیست حلقوی
  - frontمکانی است که ب ابتدای لیست اشاره می کند.
  - Rear – مکانی است که به انتهای لیست اشاره می کند.



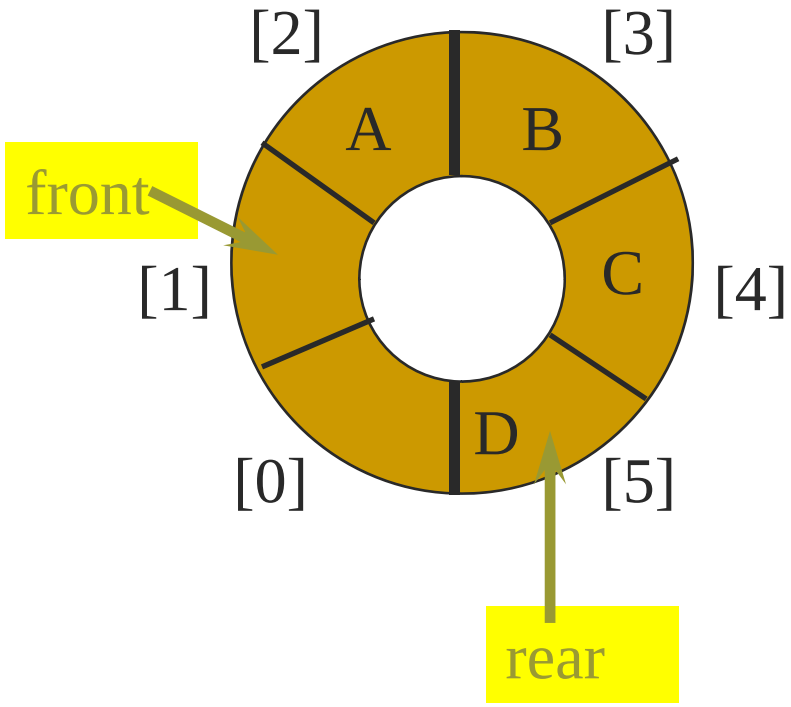
Add An Element

- rear درجهت عقربه های ساعت حرکت می کند.



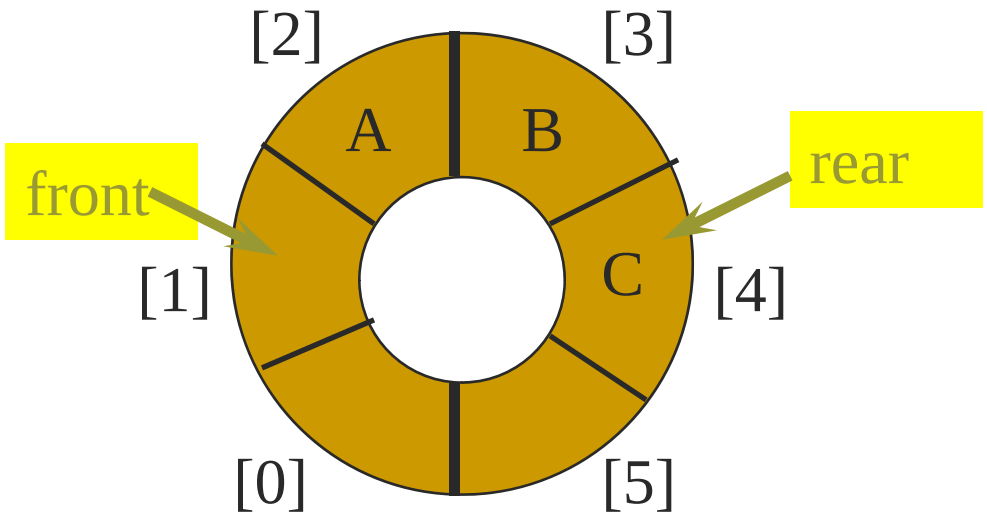
Add An Element

- rear درجهت عقربه های ساعت حرکت دهید.
- در مکانی که rear به آن اشاره می کند queue[rear] را اضافه کنید.



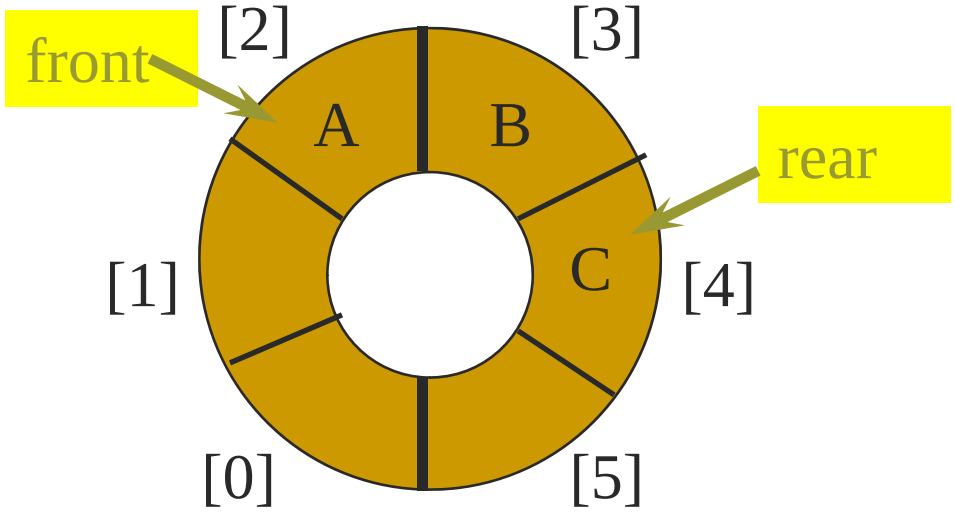
Remove An Element

- Front را یک واحد جابه جا کنید.



Remove An Element

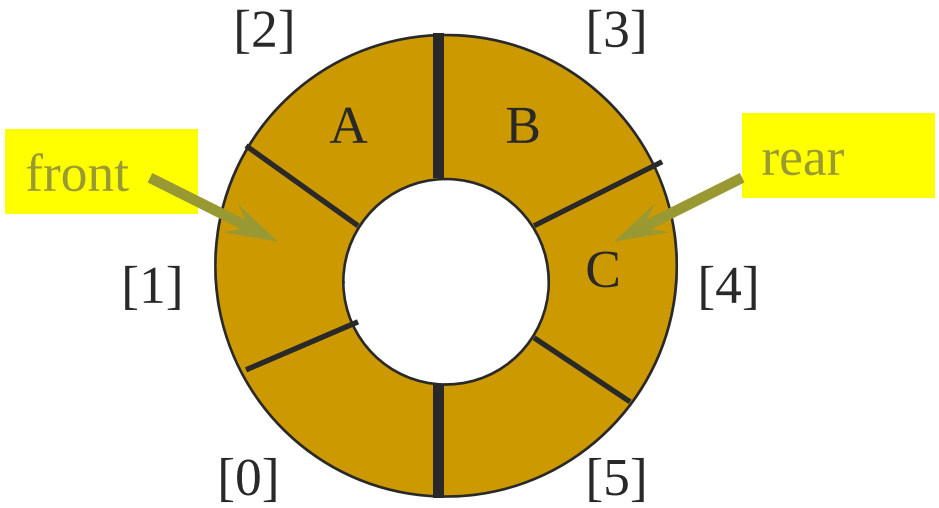
- Front را یک واحددر جهت عقربه های ساعت جابه جا کنید.
- و محتوای queue[front] را از صف حذف کنید.



Moving rear Clockwise

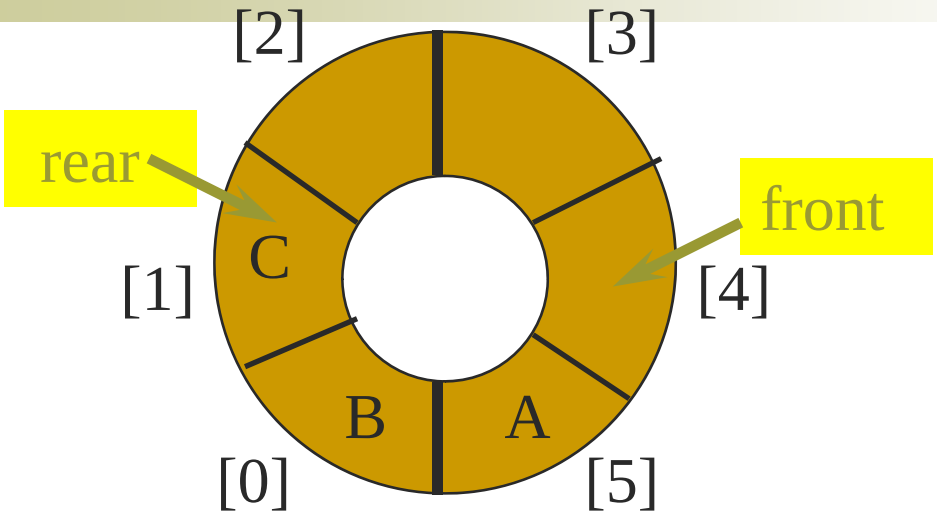
rear++;

if (rear = = queue.length) rear = 0;

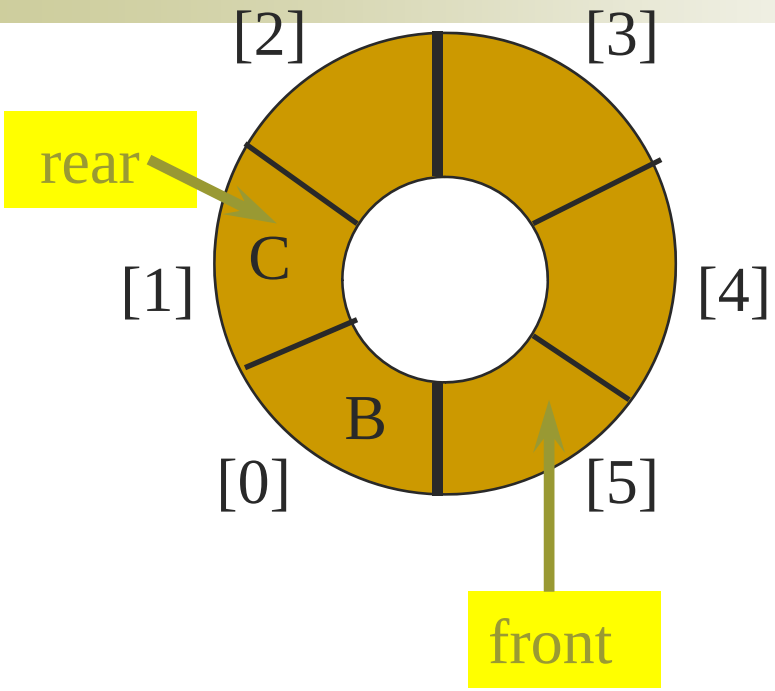


- rear = (rear + 1) % queue.length;

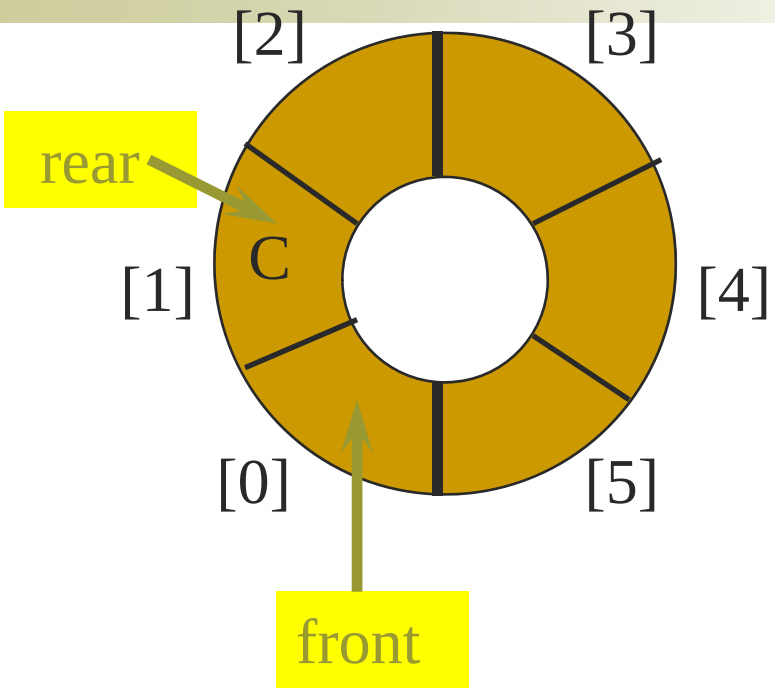
Empty That Queue



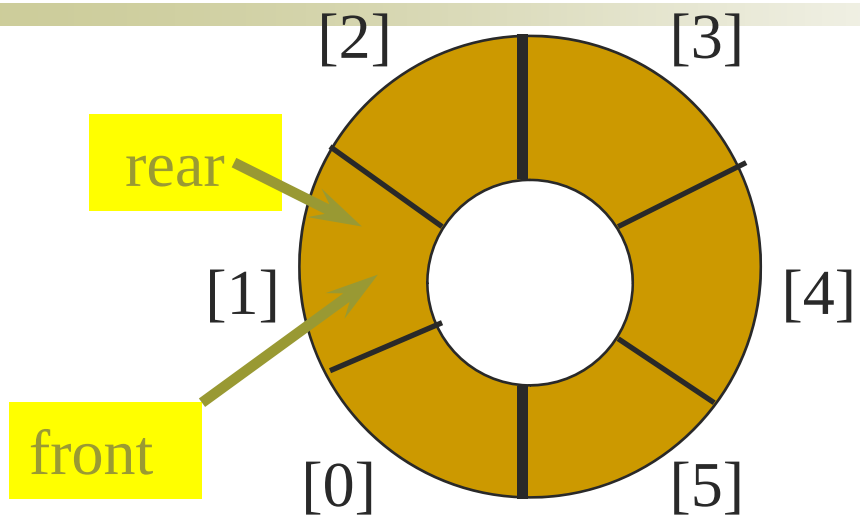
Empty That Queue



Empty That Queue

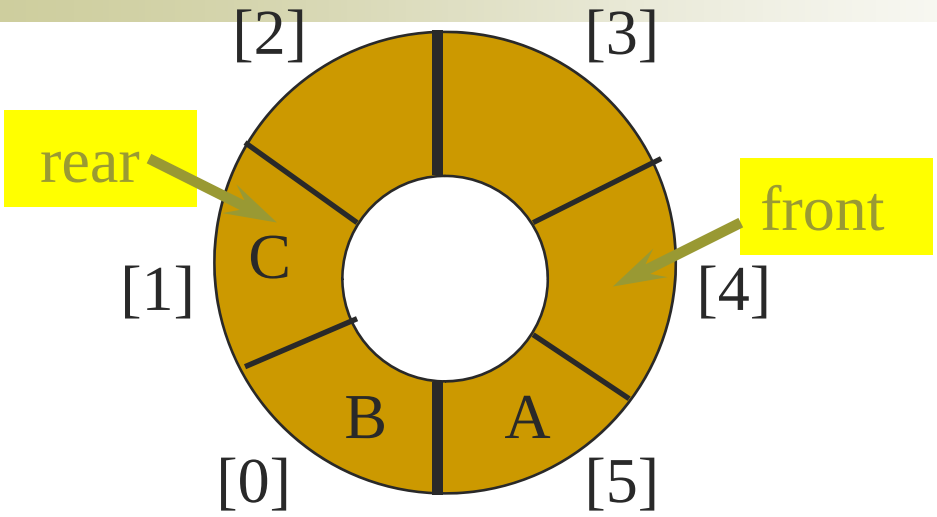


Empty That Queue

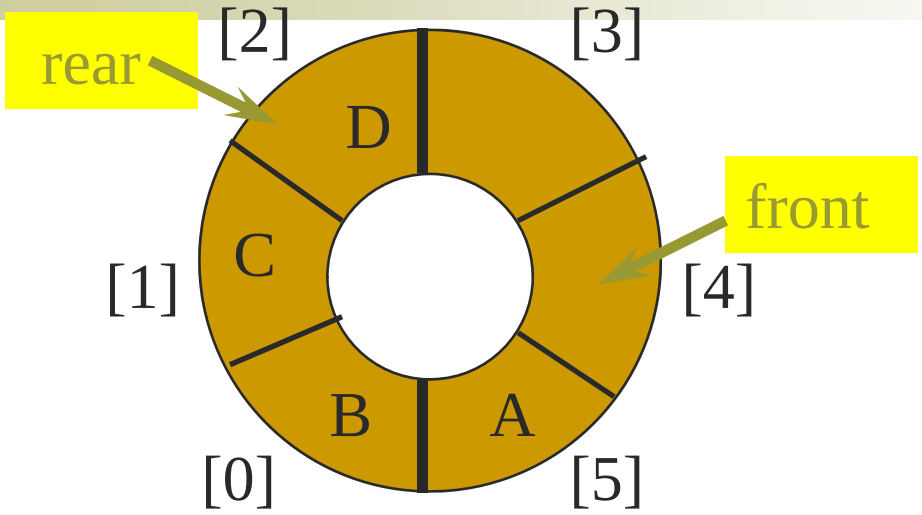


- زمانی تمام عناصر داخل صف حذف شود آنگاه: front = rear
- اگر صف خالی باشد front = rear = 0

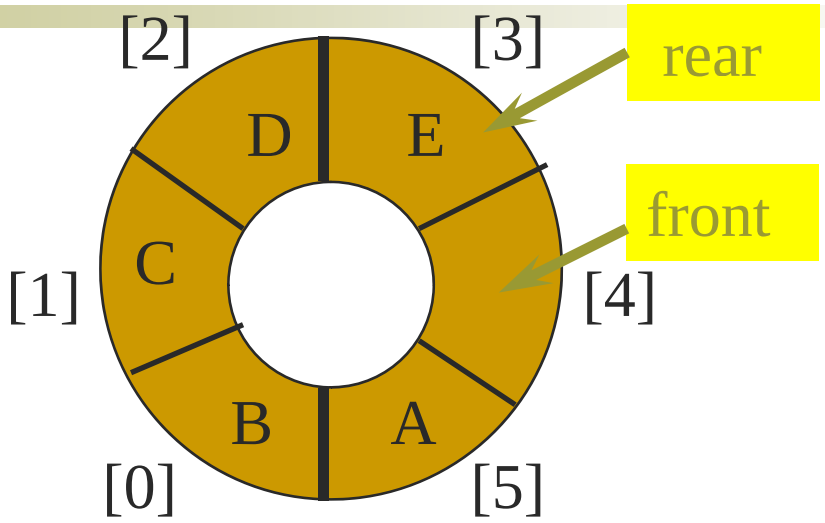
A Full Tank Please



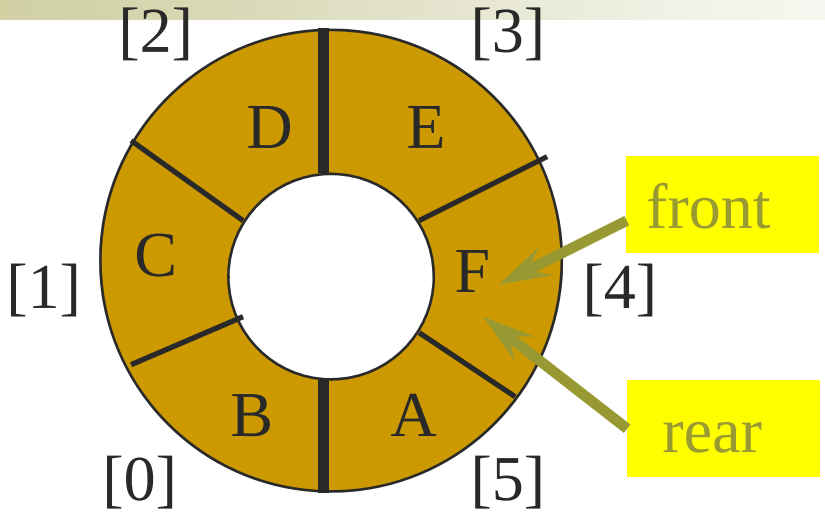
A Full Tank Please



A Full Tank Please



A Full Tank Please



- اگر به پر کردن صف ادامه دهیم در نهایت می رسیم به شرط: front = rear
- اگر دقت کنید همان شرط خالی بودن صف می باشد درصورتی که صف پر است!

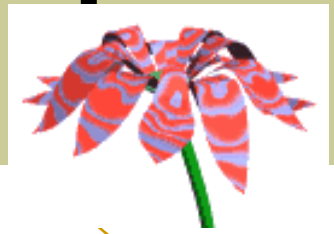
Ouch!!!!!

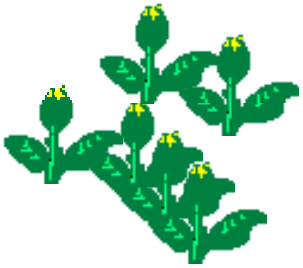
- شرط پر و خالی بودن در صف حلقوی
  - Queue is empty iff (front == rear) && !lastOperationIsPut
  - Queue is full iff (front == rear) && lastOperationIsPut

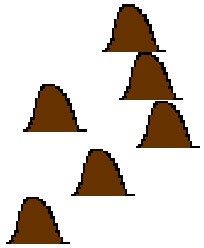
Ouch!!!!!

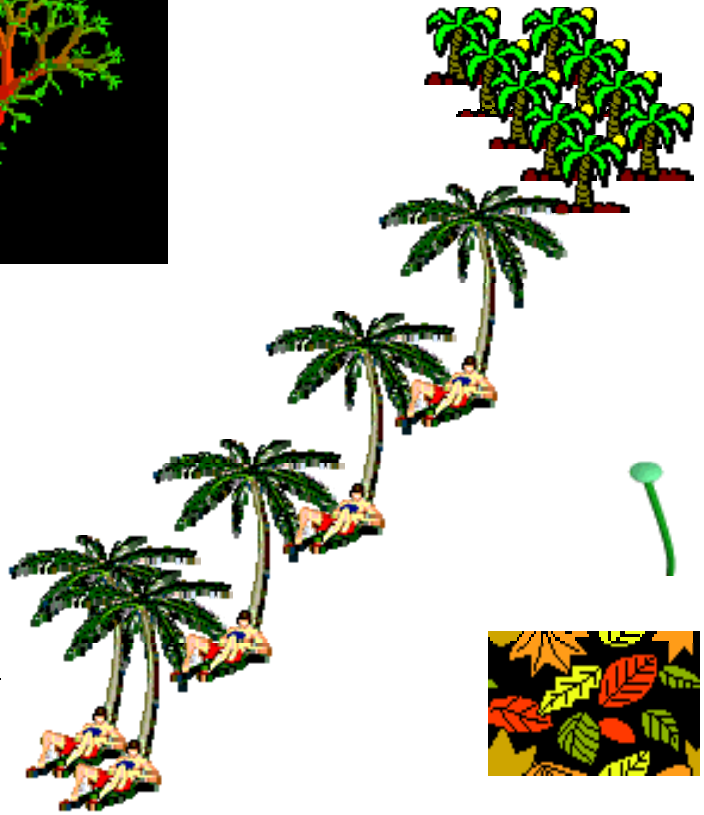
- شرط پر و خالی بودن در صف معمولی
  - Queue is empty iff (size == 0)
  - Queue is full iff (size == queue.length)

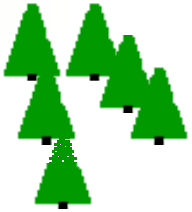
Trees





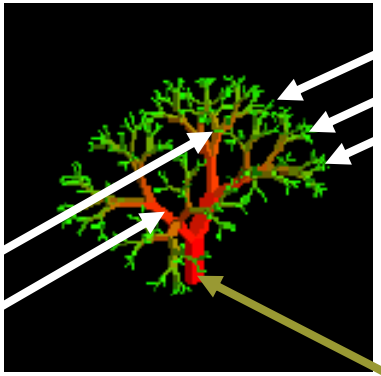






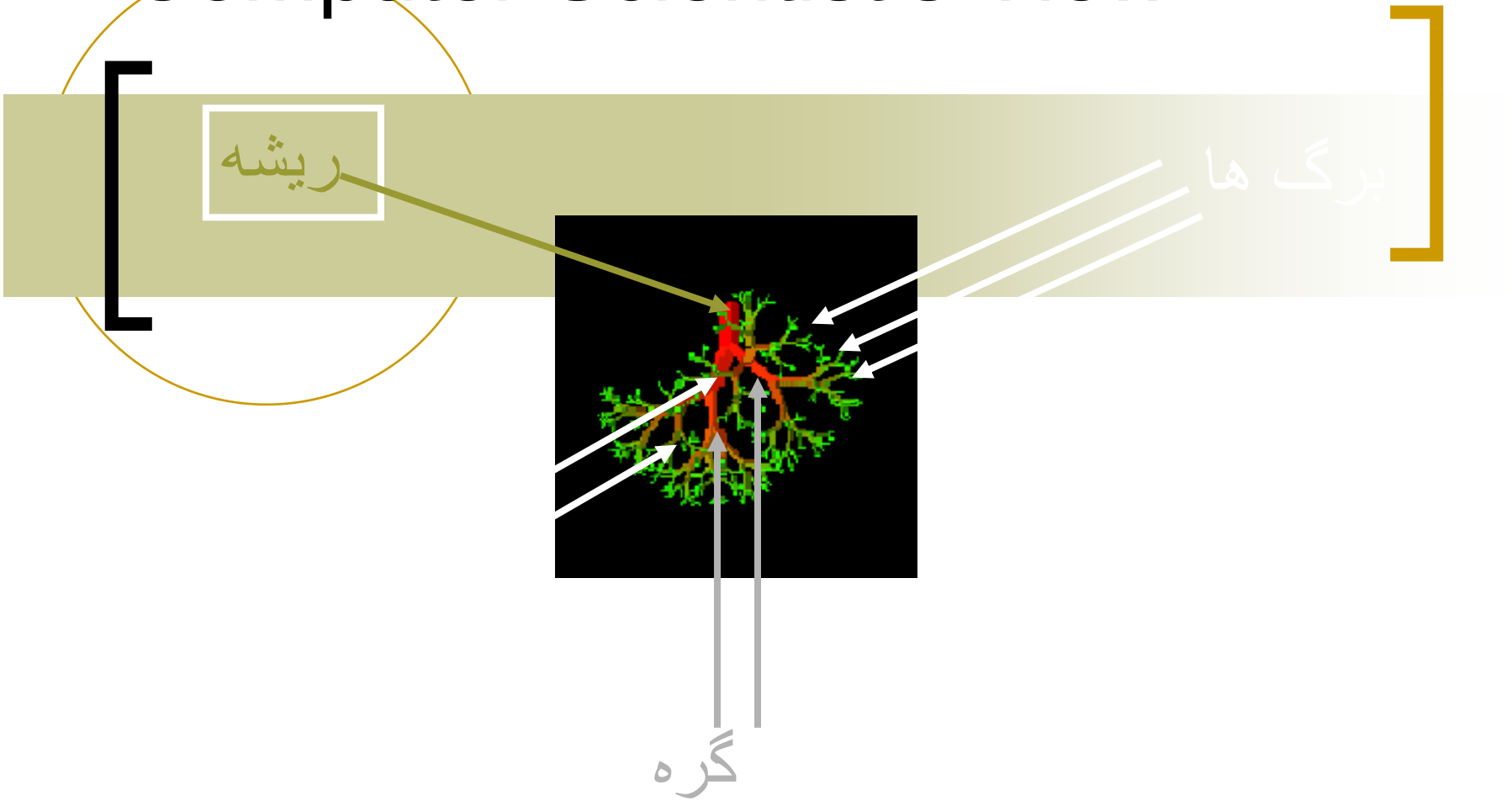
Nature Lover’s View Of A Tree

برگ ها



ریشه

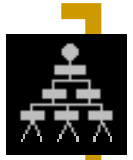
Computer Scientist's View




Linear Lists And Trees

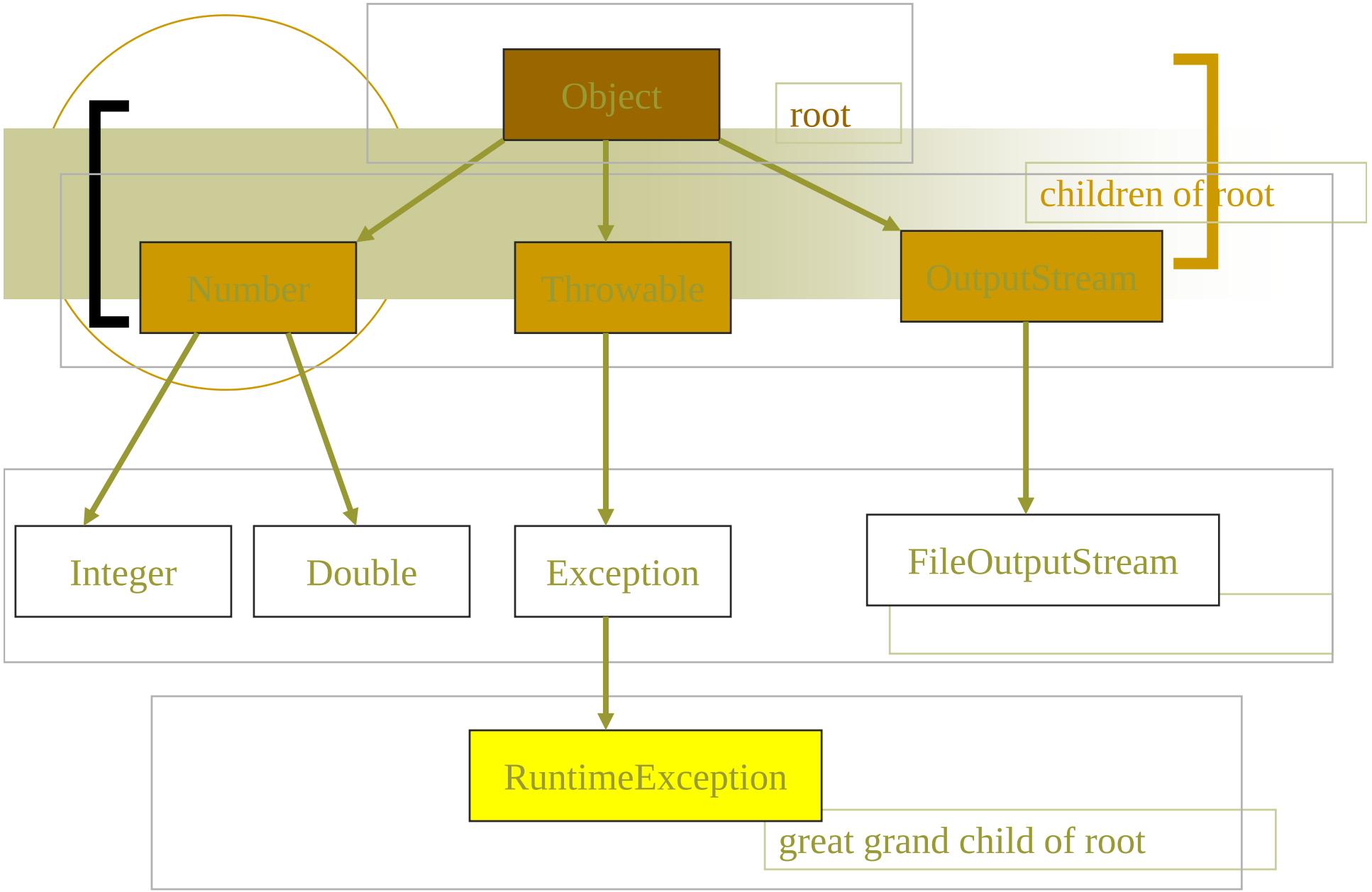
- کاربردهای لیست های خطی :برای داده های ترتیبی است
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - روزهای هفته
  - ماه های سال
  - دانشجویان یک کلاس
- کاربرد درخت:برای داده هایی با سلسله مراتب یکسان
  - مانند کارمندان یک شرکت

Hierarchical Data And Trees



- عنصری که در بالاترین مرتبه قرار دارد یا ریشه نامیده میشود.
- عناصری که در مراتب بعد از ریشه قرار می گیرند نامیده می شود.
- عناصری هستند که فاقد childrenهستند.

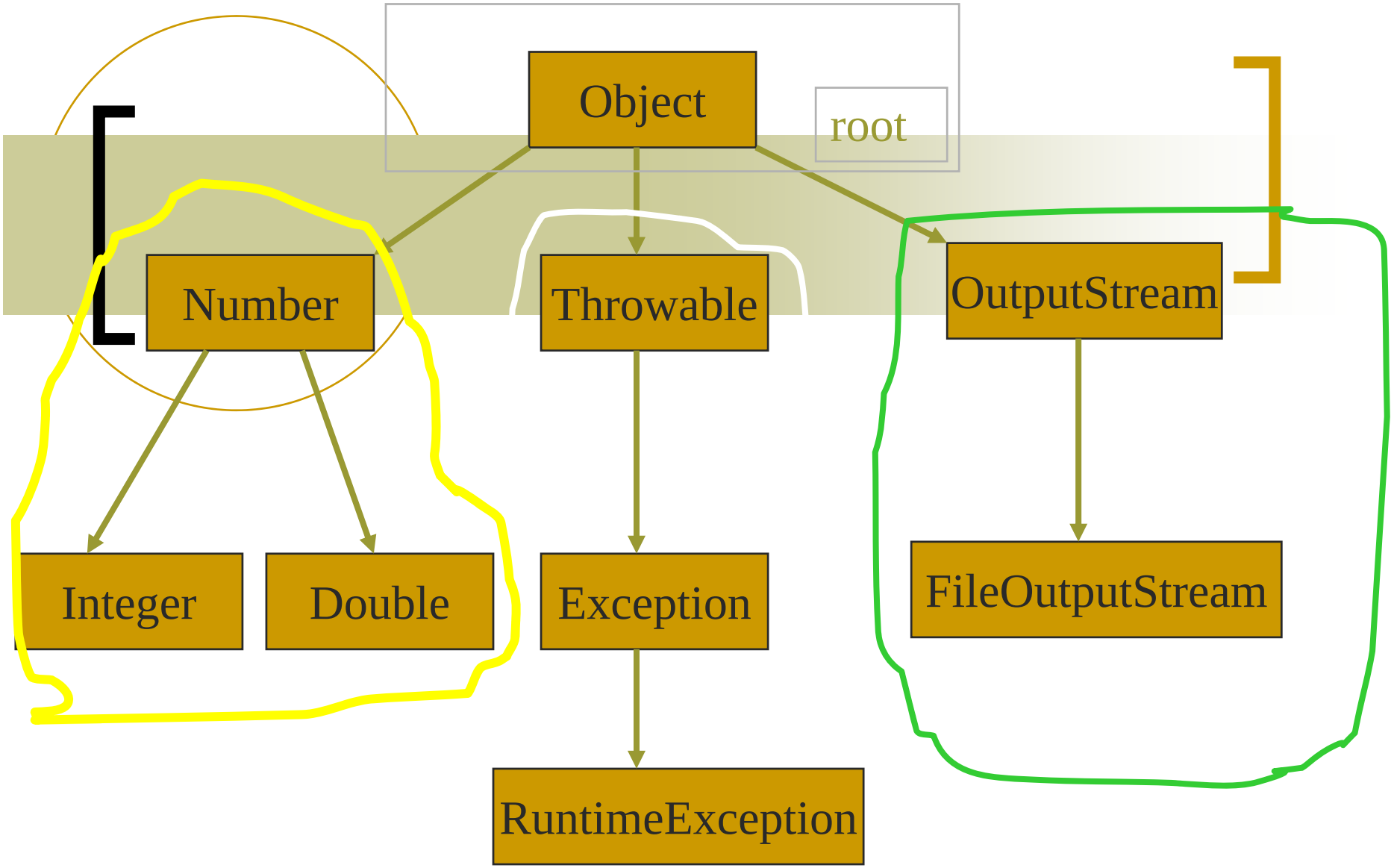
Classes (Part Of Figure 1.1)



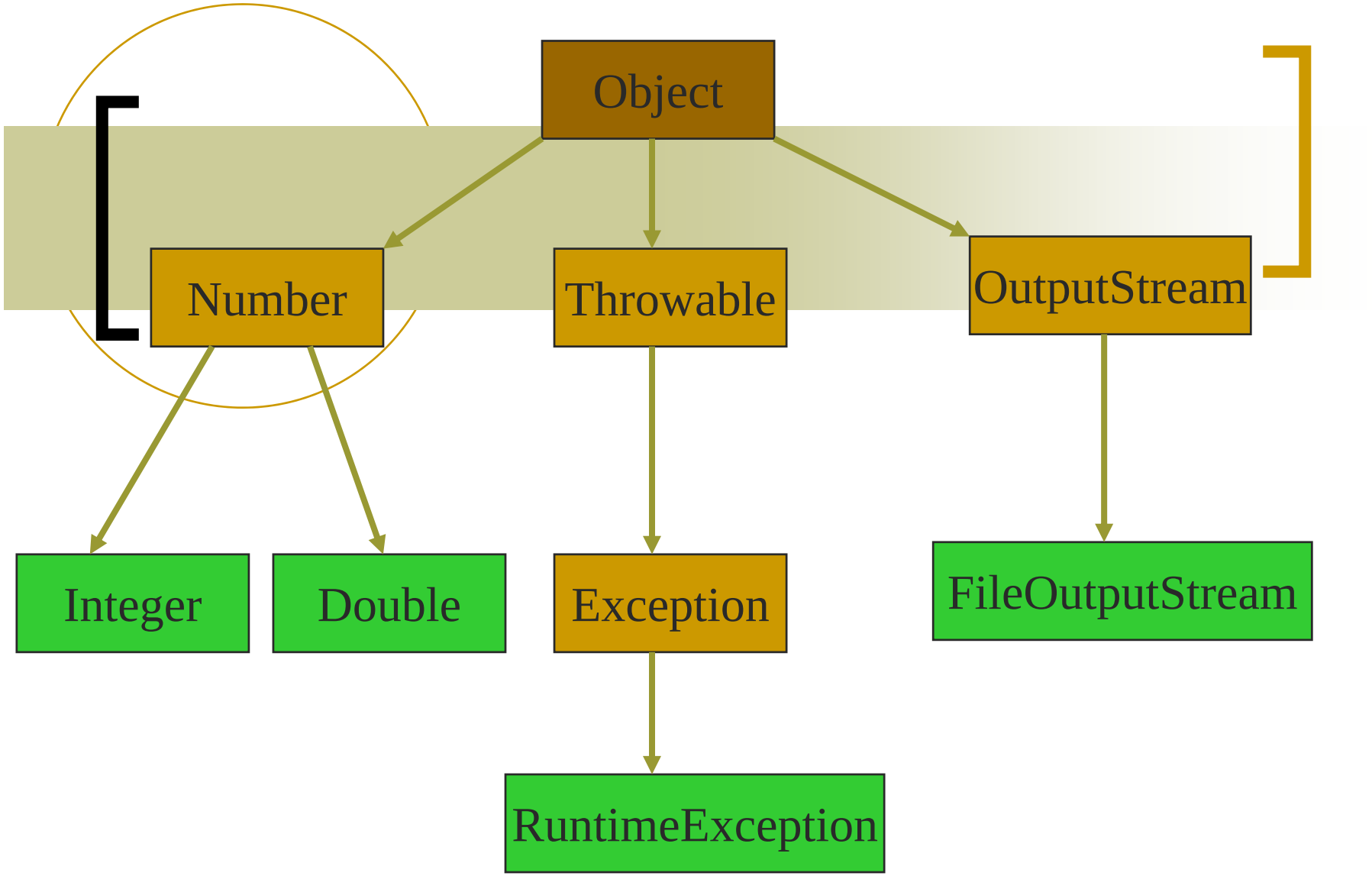
Definition

- یک درخت از یک مجموعه عناصر متناهی تشکیل شده است.
- اولین عنصر ریشه نامیده می شود.
- مابقی عناصرزیر درخت نامیده " subtrees of "می شود.

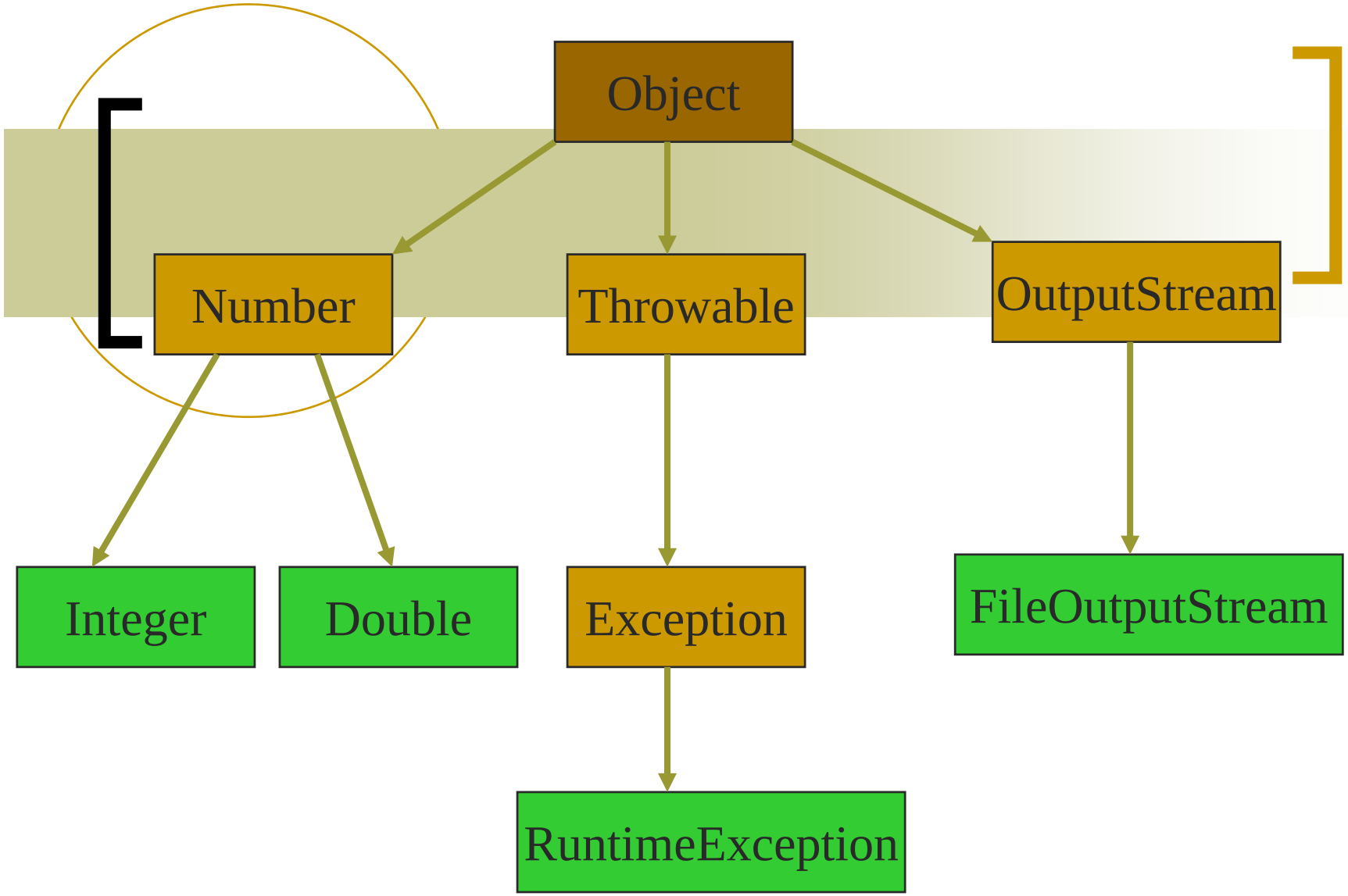
Subtrees


Object
root
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
RuntimeException

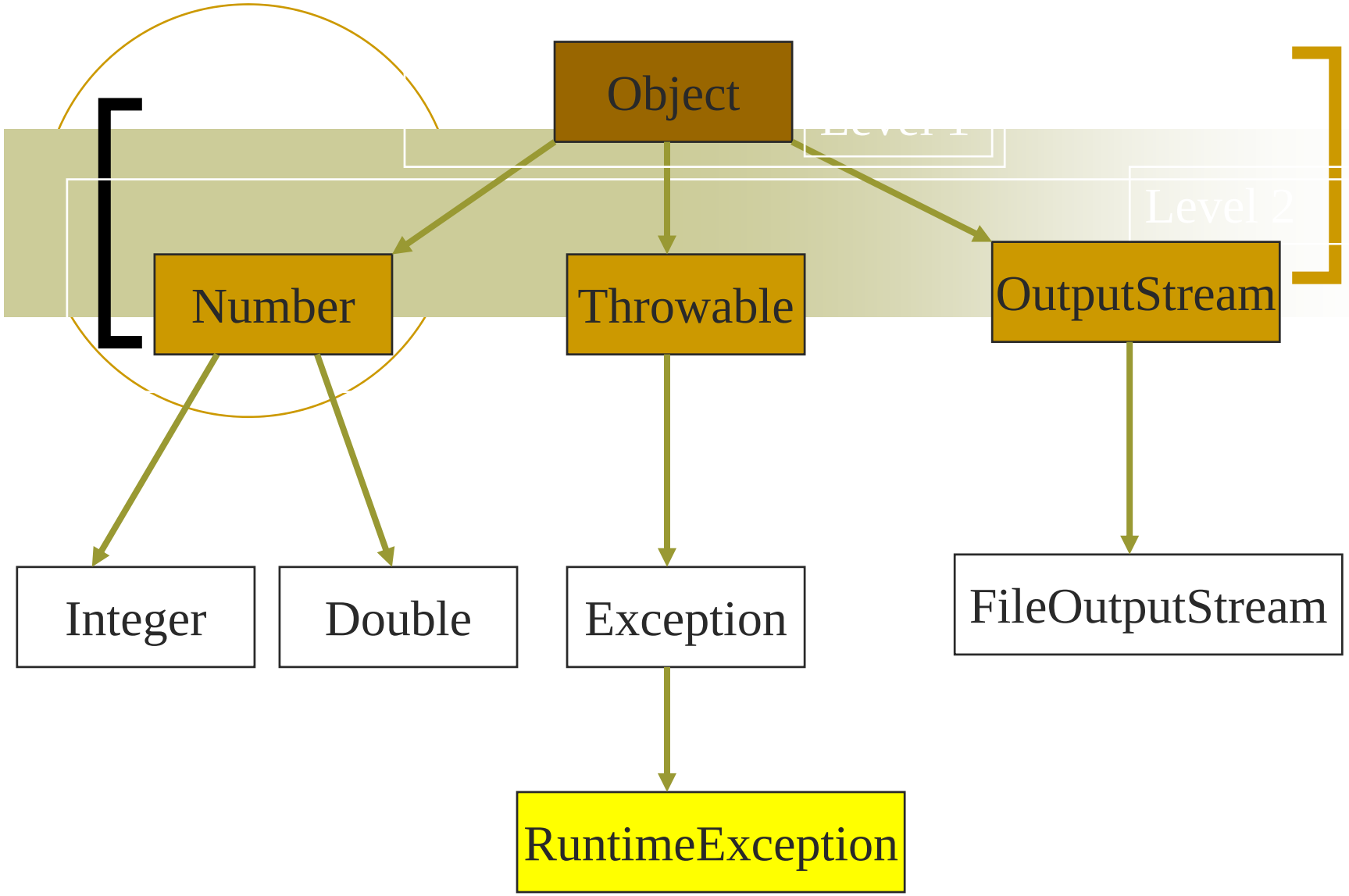
Leaves


Object
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
RuntimeException

Parent, Grandparent, Siblings, Ancestors, Descendants



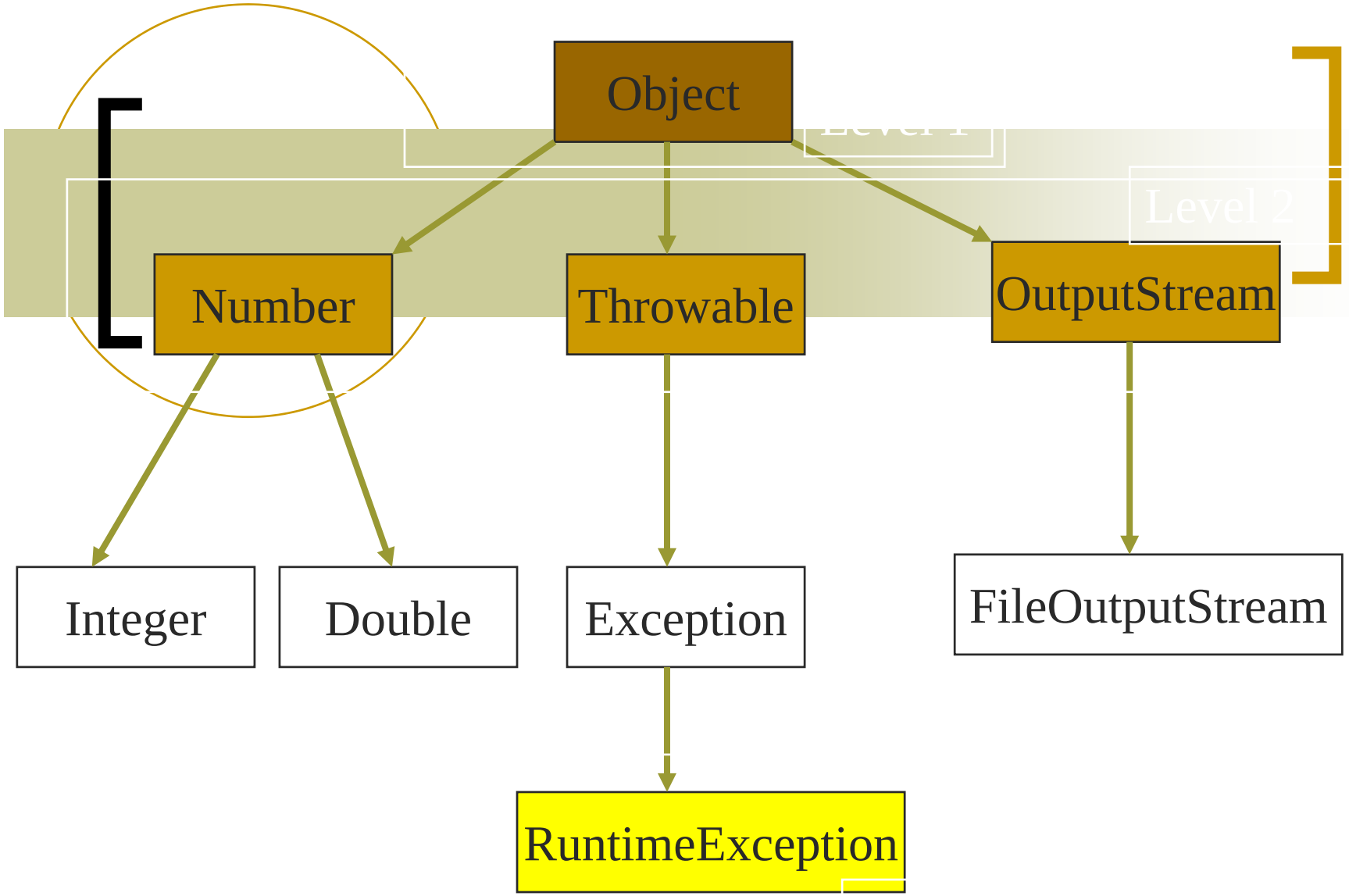
Levels


Object
Level 1
Level 2
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
RuntimeException

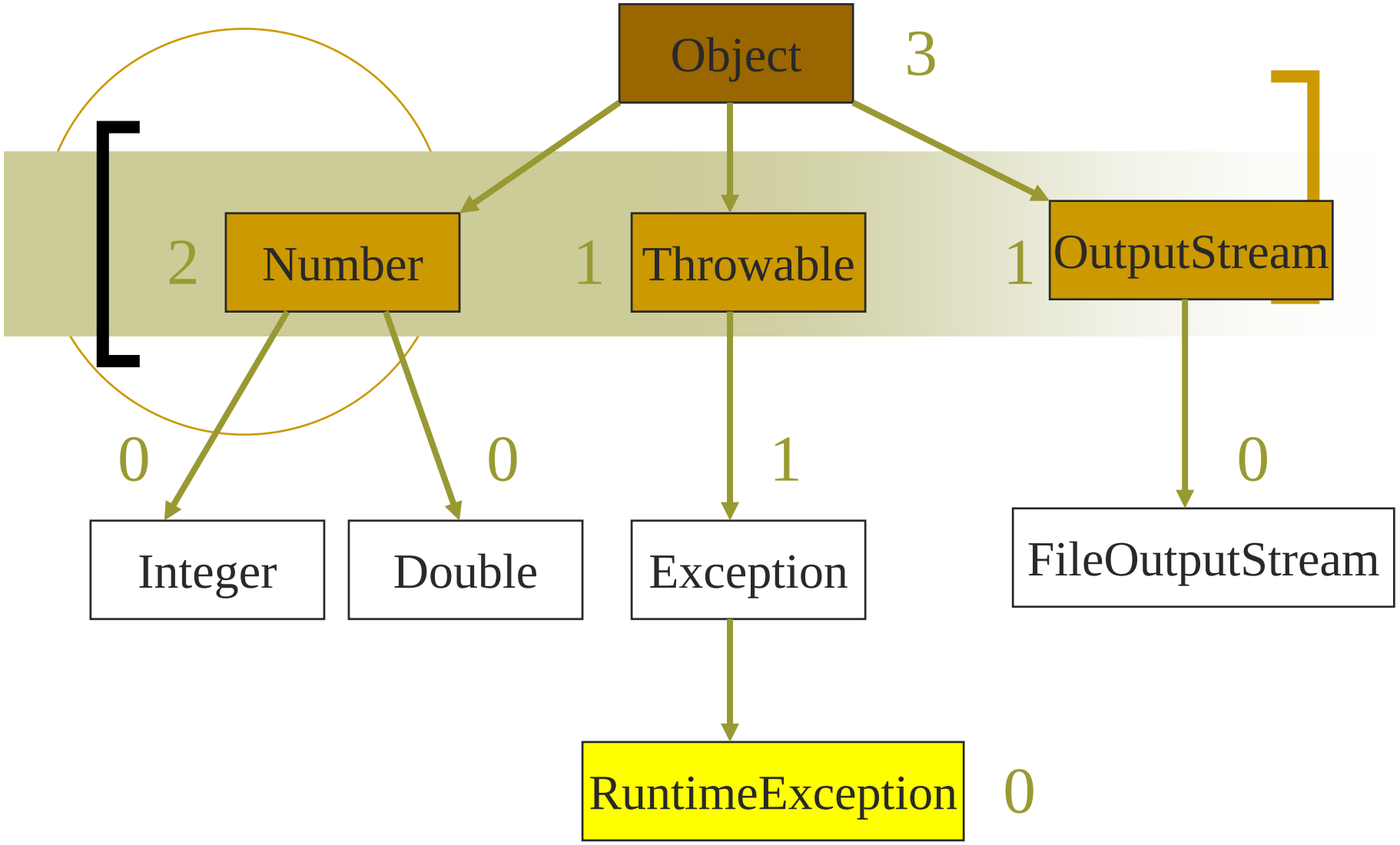

Caution

- به هرسطح یا Level در درخت یک شماره میدهیم:
- ریشه در 0 level قرار دارد.
- زیر درخت های ریشه از سطح ۱شروع می شوند.
- البته level ریشه می تواند از سطح یک نیز باشد بنابراین levelمربوط به زیر درخت های ریشه ا سطح ۲ شروع میشود.

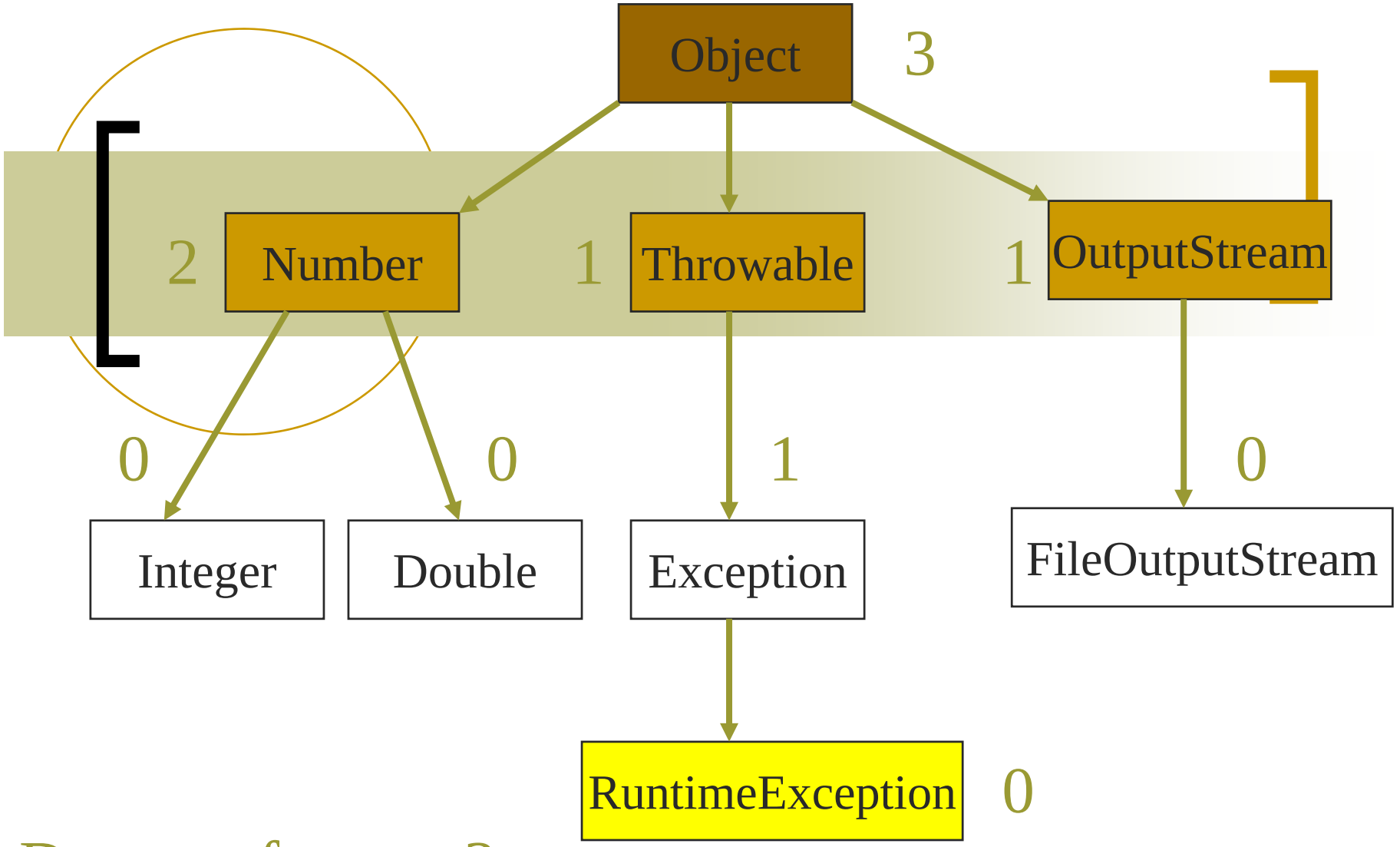
height = depth = number of levels



Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

- درخت دودویی از تعداد محدودی عنصر تشکیل شده است.
- اولین عنصر آن ریشه می باشد.
- درخت دودویی درختی است که درجه هر گره در آن یا صفر باشد یادو.
- تعریف:درجه گره:شامل تعداد زیر درخت های آن گره میباشد.
- اگر درجه گره دو باشد دو زیر درخت آن زیر درخت های چپ و راست نامیده می شوند.

Differences Between A Tree & A Binary Tree

- هر گره در درخت باینری درجه بیشتر از دو ندارد.
- یک درخت دودویی می تواند خالی باشد اما درخت معمولی نمی تواند خالی باشد.

Differences Between A Tree & A Binary Tree

- در درخت باینری زیر درخت ها دارای ترتیب می باشند.اما در درخت معمولی برای زیر درخت ها ترتیبی قائل نیستیم.

- ترتیب زیر درخت ها:
- متفاوت هستنداگر به دیددرخت باینری به آنها نگاه کرد.
- یکسان هستند اگر به دیددرخت معمولی به آنها نگاه کرد

Arithmetic Expressions

- (a + b) * (c + d) + e – f/g*h + 3.25
- عبارت محاسباتی بالا تسکیل شده از:
  - Operators (+, -, /, *). //عملگرها
  - Operands (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.). //عملوند ها
  - Delimiters ((,)).

Operator Degree

- انواع عملگر ها:
- عملگر های دودویی:نیازمند دو عملوند هستند.

  a + b

  c / d

  e - f
- عملگر های یکانی:نیازمند یک عملوند هستند.

  + g

  - h

Infix Form

- روش معمولی برای نوشتن یک عبارت.
- عملگر های دودوئی میان دو عملوند قرار می گیرند.

a * b

a + b * c

a * b / c

(a + b) * (c + d) + e – f/g*h + 3.25

Operator Priorities

- کدام عملگر زودتر اعمال می شود؟

  a + b * c

  a * b + c / d

- برای عملگر هاترتیب یا اولویت قائل می شویم

priority(*) = priority(/) > priority(+) = priority(-)

- عملگری زودتر اعمال می شود که بالاترین اولویت را داشته باشد.

Tie Breaker

- اگر عملگرهایی که در عبارت قرار می گیرند دارای اولویت یکسان باشند از چپ به راست عملگر ها را روی عملوندها اعمال کنید.

```
a + b - c
a * b / c / d
```

Infix Expression Is Hard To Parse

- سه روش برای نمایش یک عبارت محاسباتی داریم:
- Infix:که درآن ابتدا عملوندچپ /بعدعملگردر وسط/ ونهایتاعملوند راست قرار می گیرد.

prefix :که درآن ابتدا عملگریا ریشه / بعدعملوندچپ / ونهایتاعملوند راست قرار می گیرد.

postfix:که درآن ابتدا عملوندچپ عملگریا ریشه / بعدعملوندراست /ونهایتا ریشه قرار می گیرد.

Postfix Form

- مثال:

Infix = a + b

Postfix = ab+

Postfix Examples

Infix = a + b * c

a b c * +

- Infix = a * b + c

  a b * c +

- Infix = (a + b) * (c – d) / (e + f)

  a b + c d - * e f + /

Unary Operators

- Replace with new symbols.

+ a => a @

+ a + b => a @ b +

- a => a ?

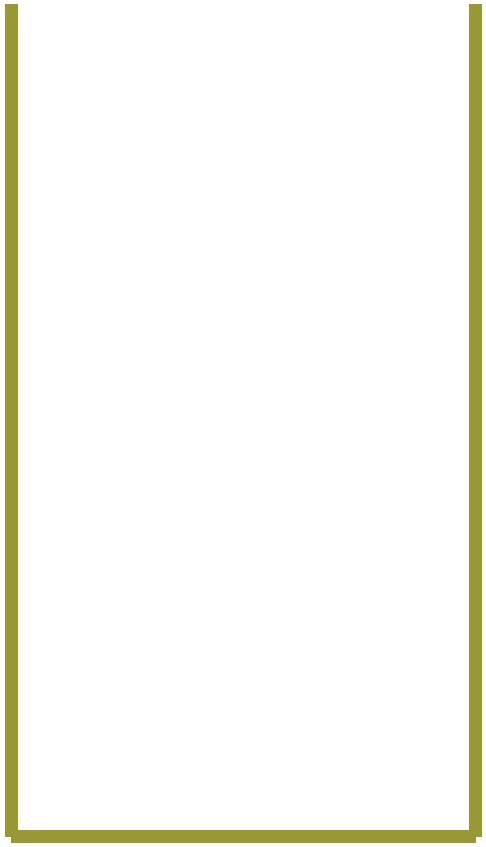
- a-b => a ? b -

Postfix Evaluation

- عبارت محاسباتی را از چپ به راست بخوانید
- به هر oprandیا عملوند رسیددر پشته push کنید.
- زمانی که به عملوند رسیدید از پشته popکنید وعملگر را روی آن اعمال کنید.
- این روش برای این است که در postfix عملگر بعد از عملوند ها می آید.

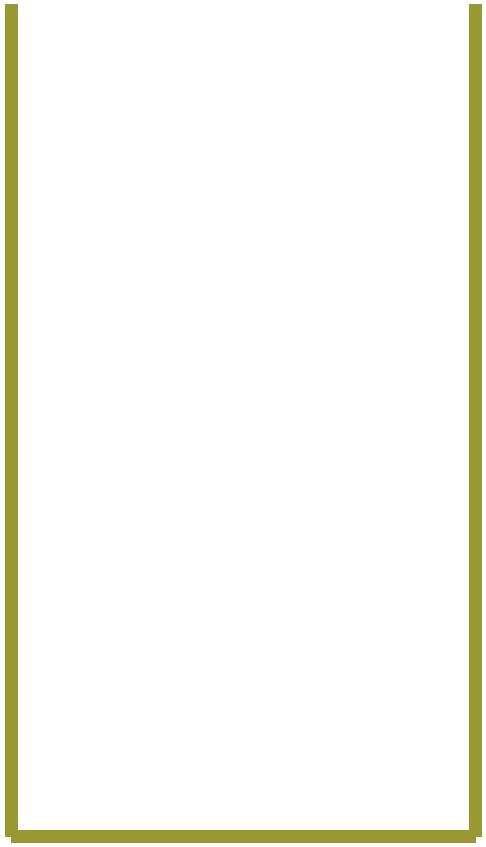
Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- b + c d - * e f + /
- + c d - * e f + /
- c d - * e f + /



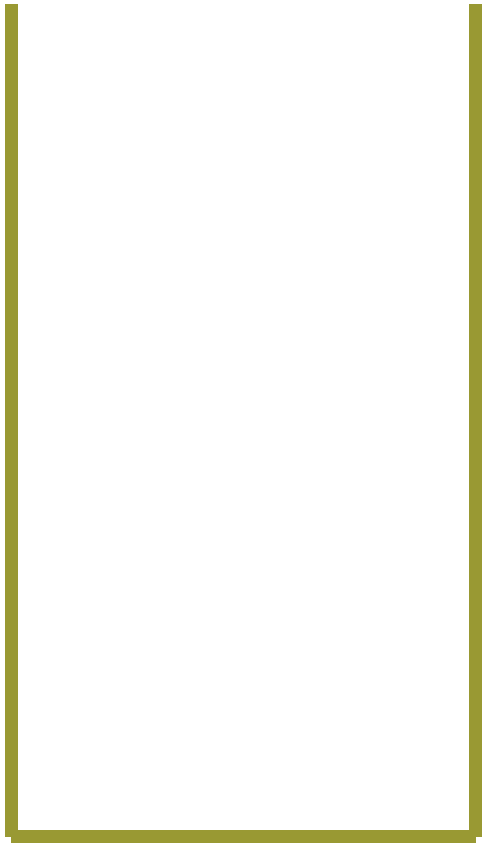
Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- b + c d - * e f + /
- + c d - * e f + /
- c d - * e f + /
- d - * e f + /
- - * e f + /
- * e f + /



Postfix Evaluation

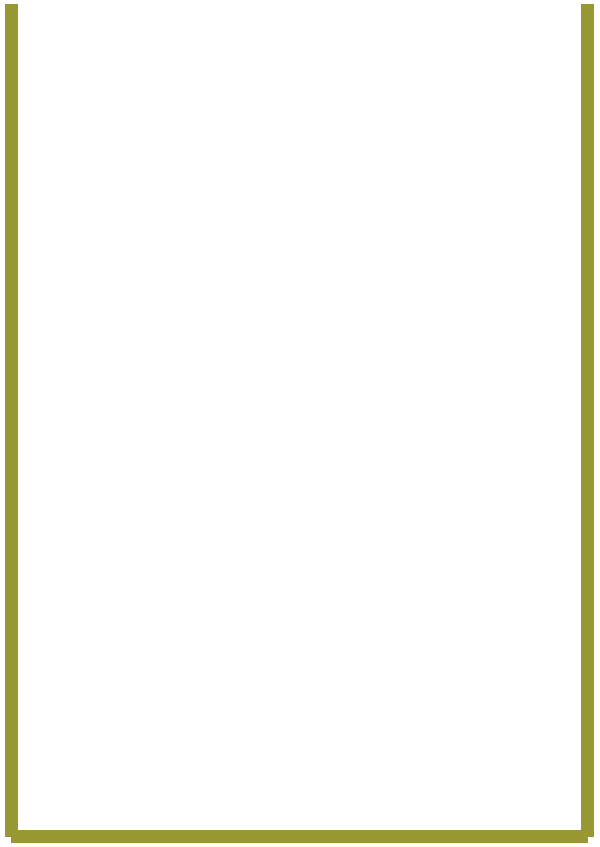
- (a + b) * (c – d) / (e + f)
- a b + c d – * e f + /



stack

Postfix Evaluation

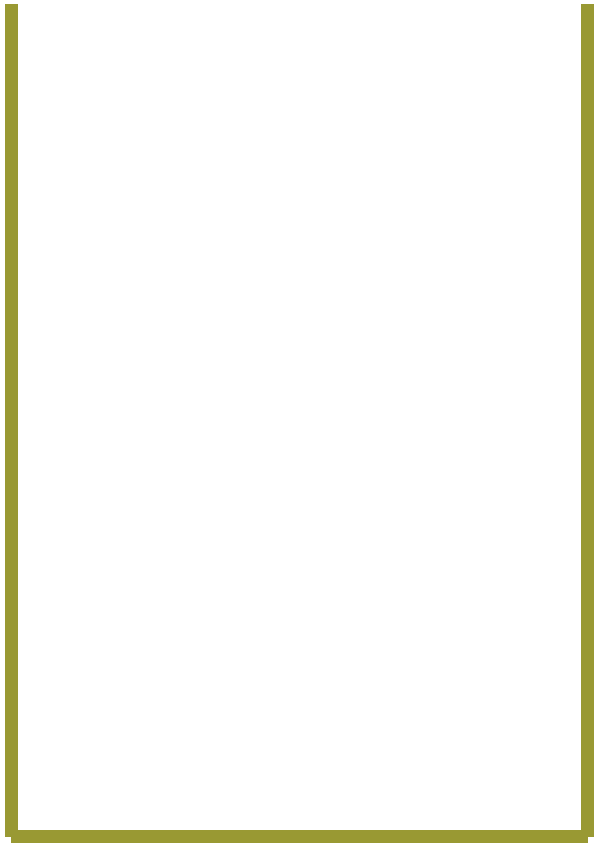
- (a + b) * (c – d) / (e + f)
- e f * + e / f + /
- f + /
- + /
- /



stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- e f * + e / f + /
- f + /
- + /
- /
-



stack

Prefix Form

- درprefix ابتدا عملگر و سپس عملوند چپ ونهایتا عموند راست می آید.

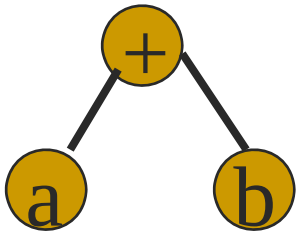
Infix = a + b

Postfix = ab+

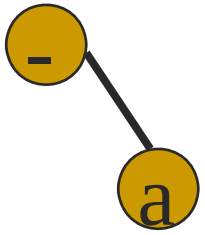
Prefix = +ab

Binary Tree Form

a + b

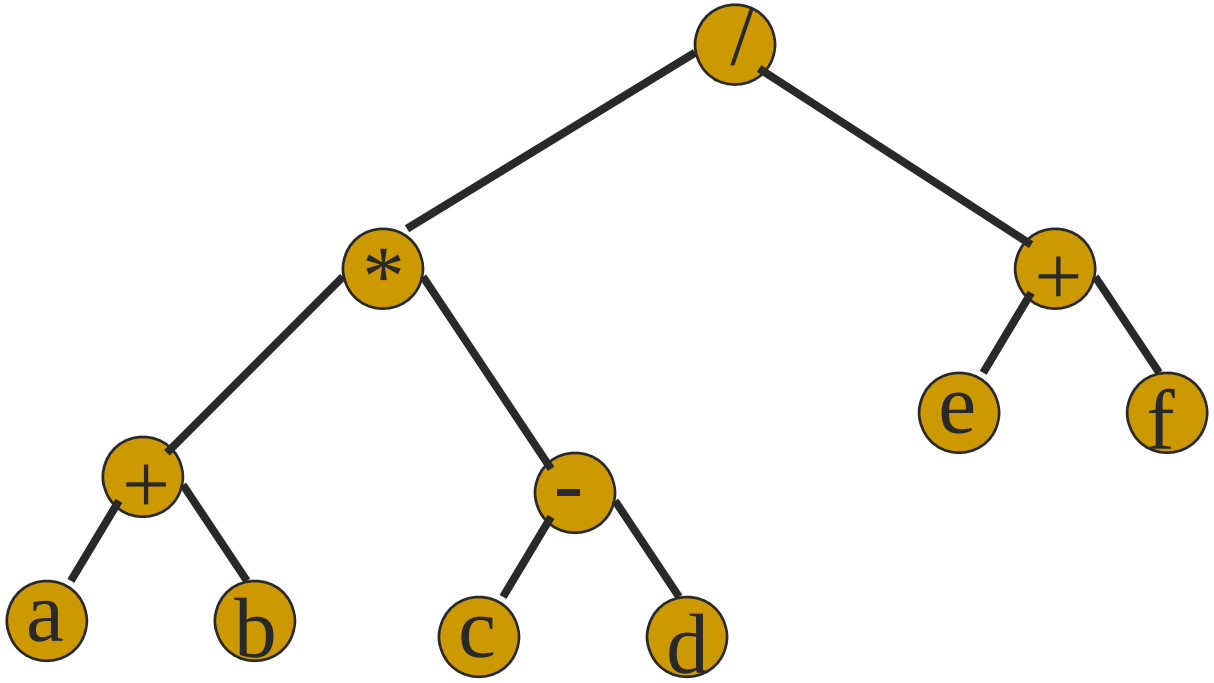


- - a



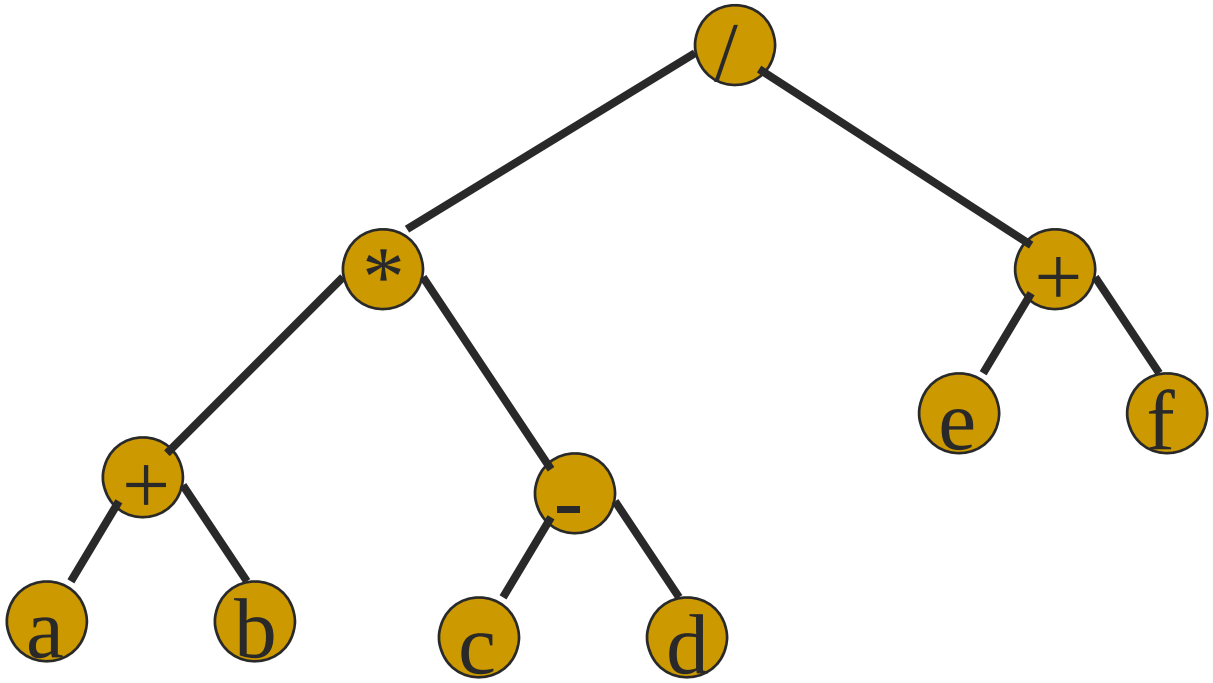
Binary Tree Form

(a + b) * (c – d) / (e + f)

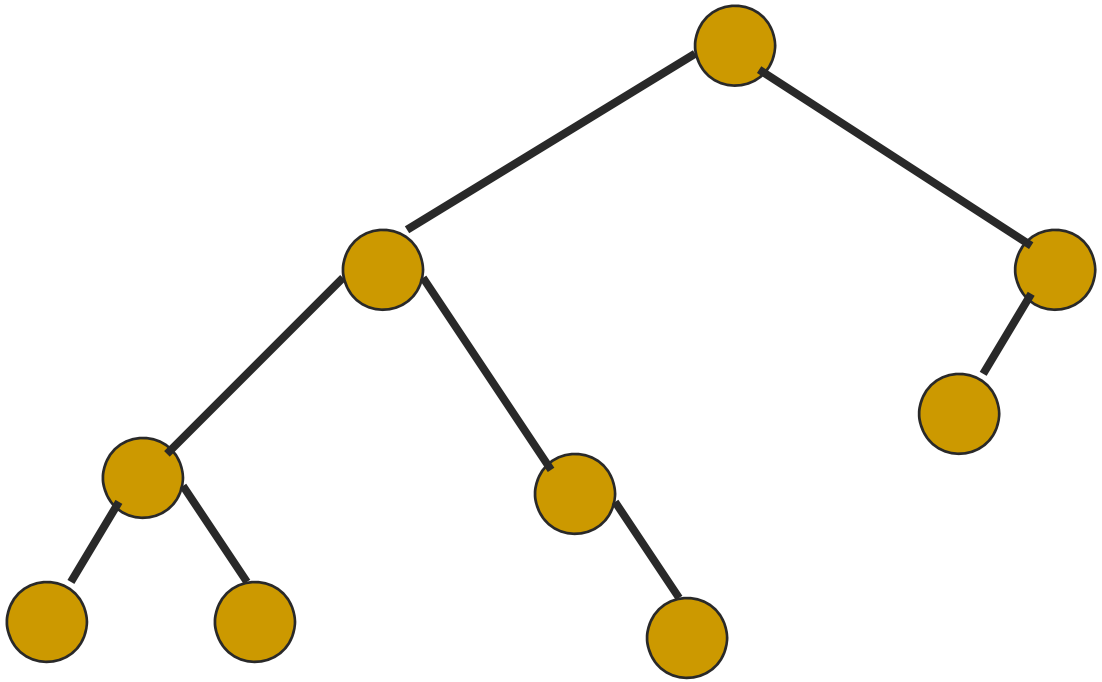


Merits Of Binary Tree Form

- کار با الگوریتم درخت باینری بهینه تر از عبارت محاسباتی است.
- Tree Binary

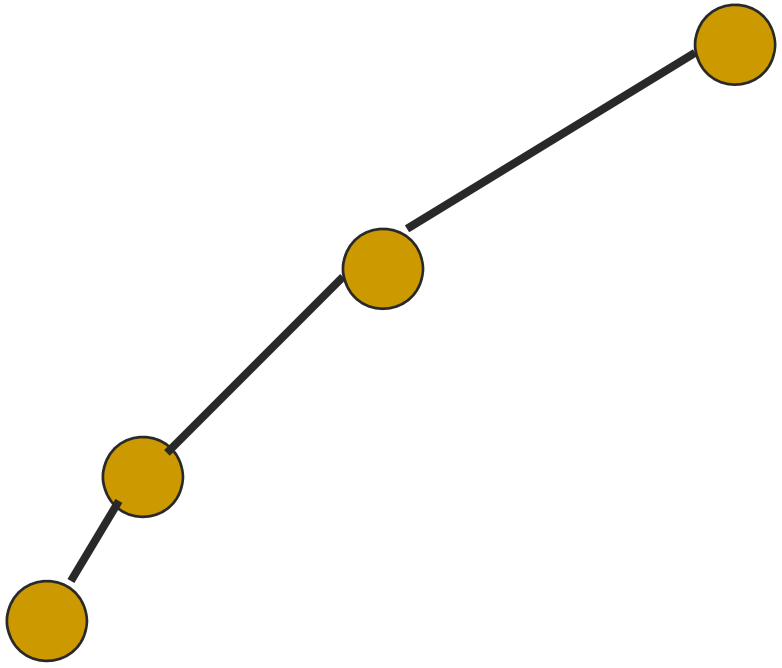


Binary Tree Properties & Representation



Minimum Number Of Nodes

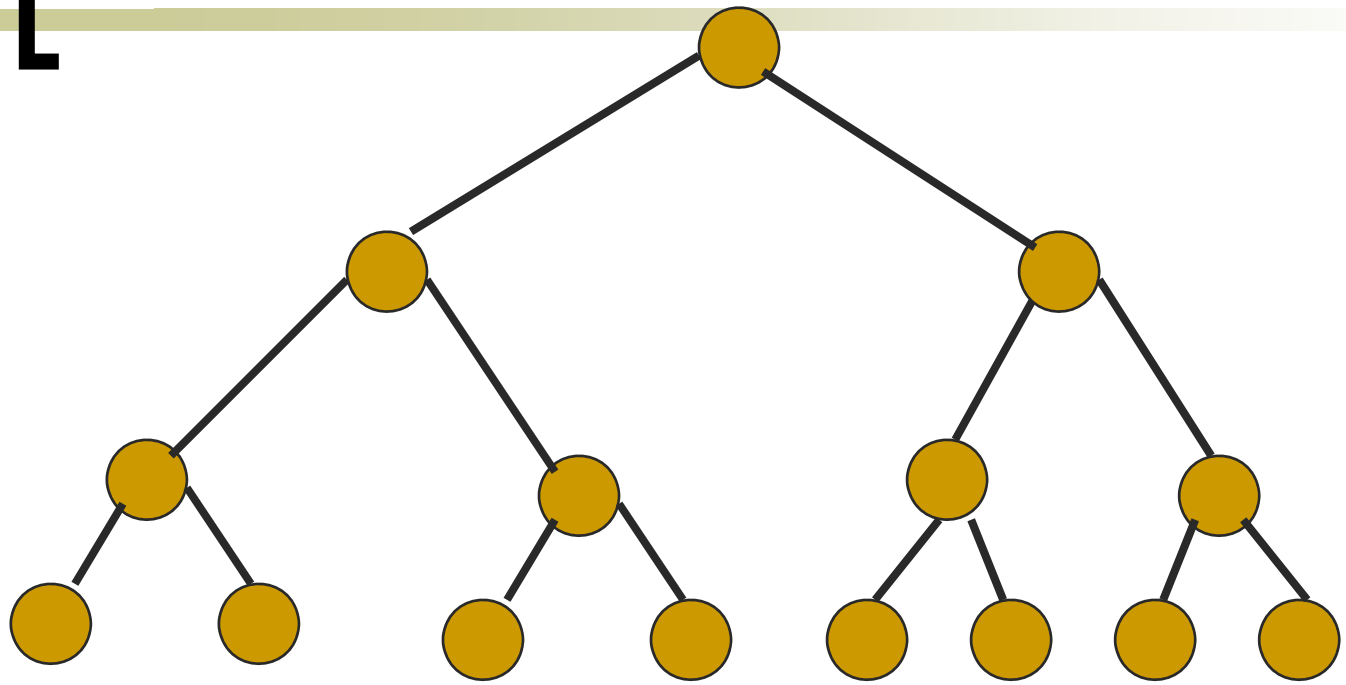
- حداقل تعداد گره های یک درخت باینری به اندازه ارتفاع آن میباشد.
- آخرین گره در درخت باینری معرف ارتفاع می باشد.



حداقل تعداد گره ها در درخت دودویی h میباشد.

Maximum Number Of Nodes

- برای درخت کامل:درختی که تمام گره های سطح آخرش وجود داشته باشد. برای محاسبه حداکثر تعداد گره هااز روش زیر:



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

$= 2^{h} - 1$

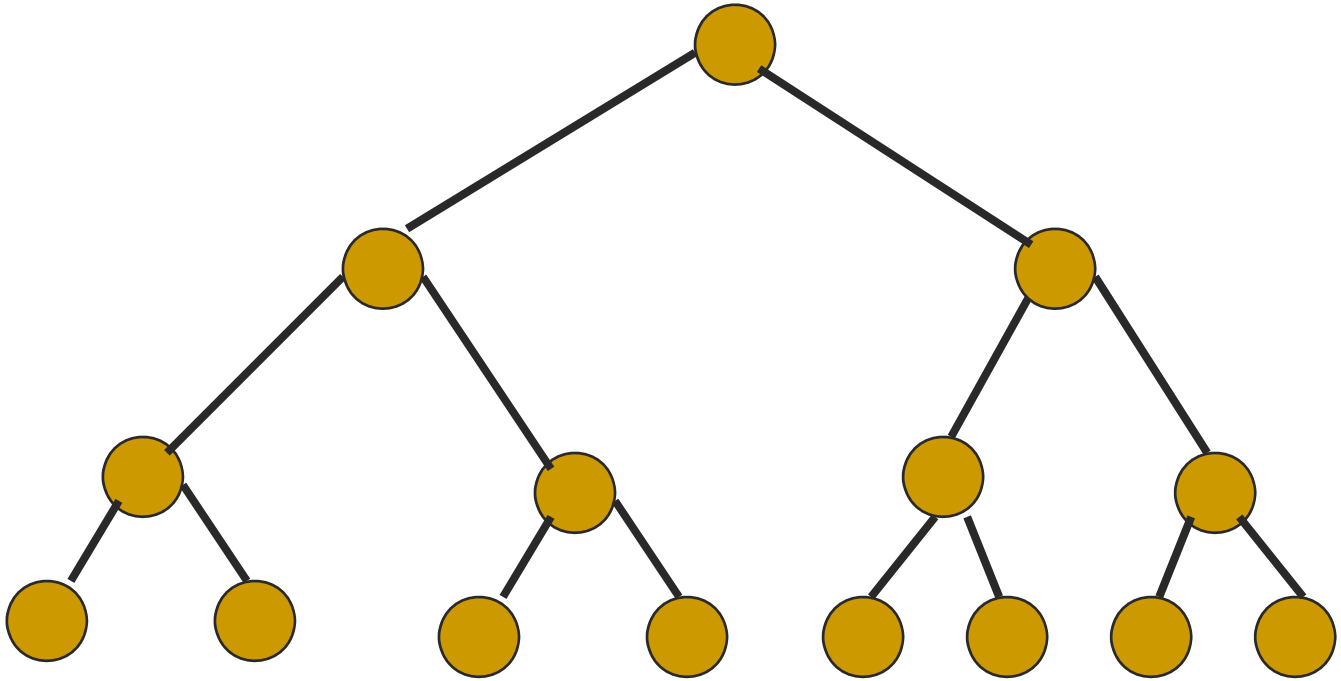
Number Of Nodes & Height

- اگر n تعداد گره های یک درخت باینری باشد محدوده آن :
- $h <= n <= 2^h - 1$
- $\log_2(n+1) <= h <= n$

Full Binary Tree

- تعدادگره ها در یک درخت دودویی به ارتفاع h:

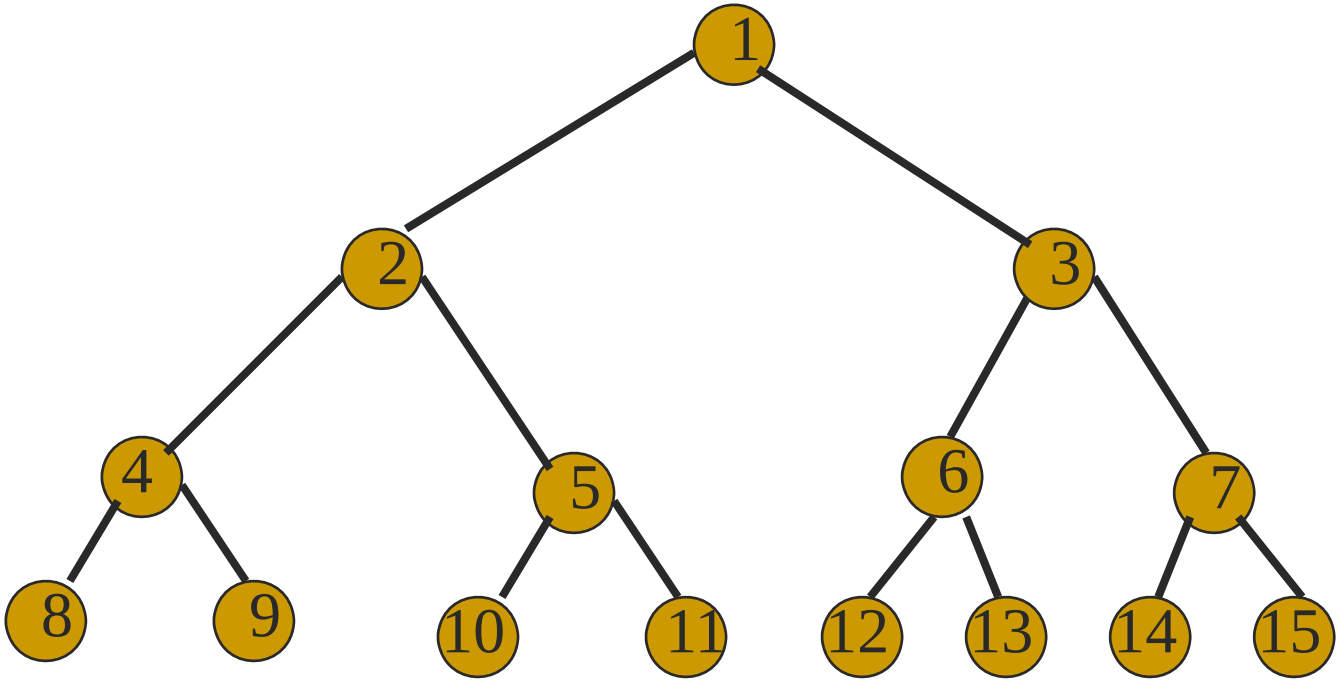
$2^h - 1$ میباشد.



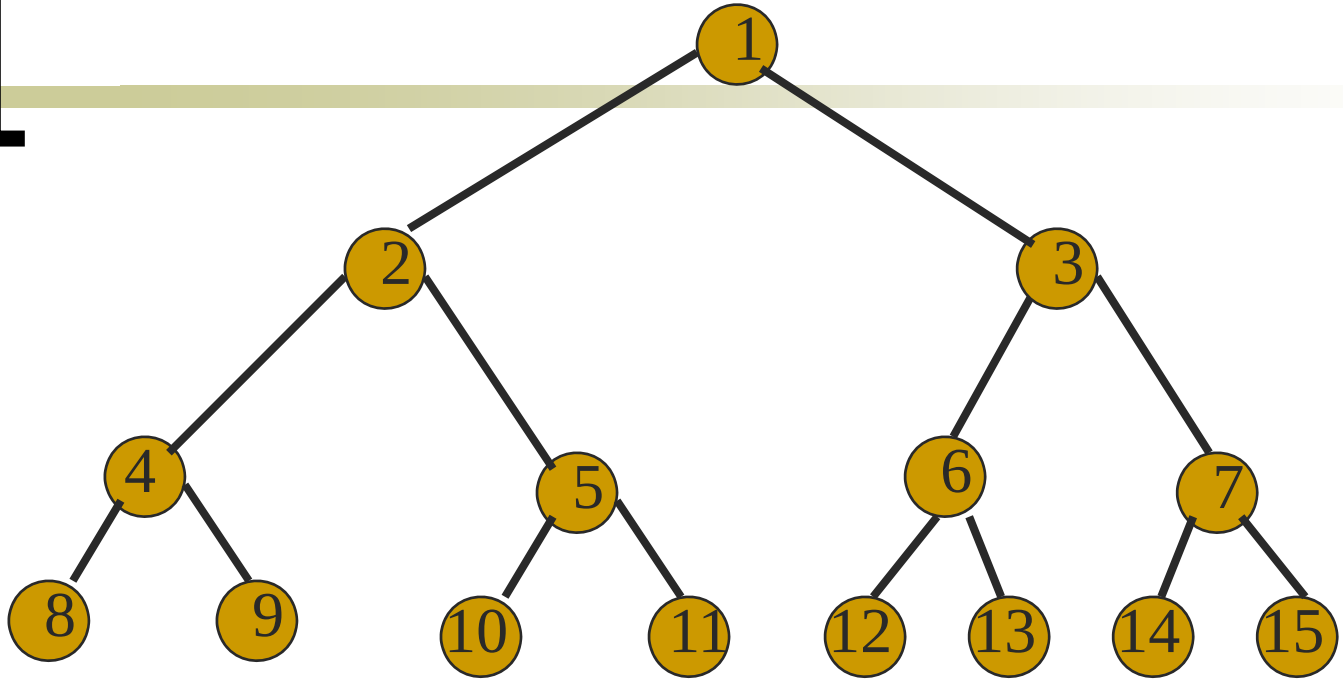
ارتفاع در درخت کامل بالا 4 میباشد.

Numbering Nodes In A Full Binary Tree

- تعداد نودهادر درخت باینری از 1 تا $2^h - 1$ متغیر است.
- شمارش سطوح از بالا به پایین می باشد.
- در هر سطح شمارش گره ها از چپ به راست می باشد.

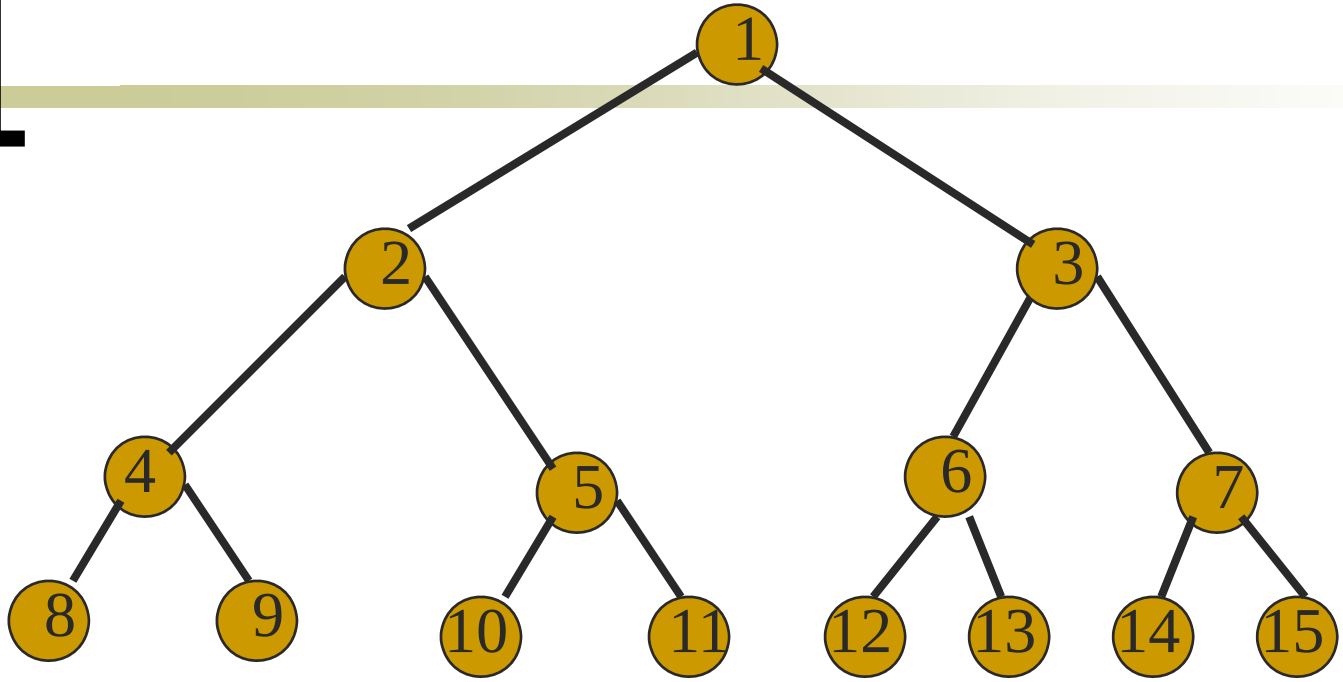


Node Number Properties



- والد هر گره مثل i درگره i / 2 آن میباشد ,البته به استثنای i = 1 زیرا:.
- i = 1 ریشه میباشد وریشه والد ندارد .

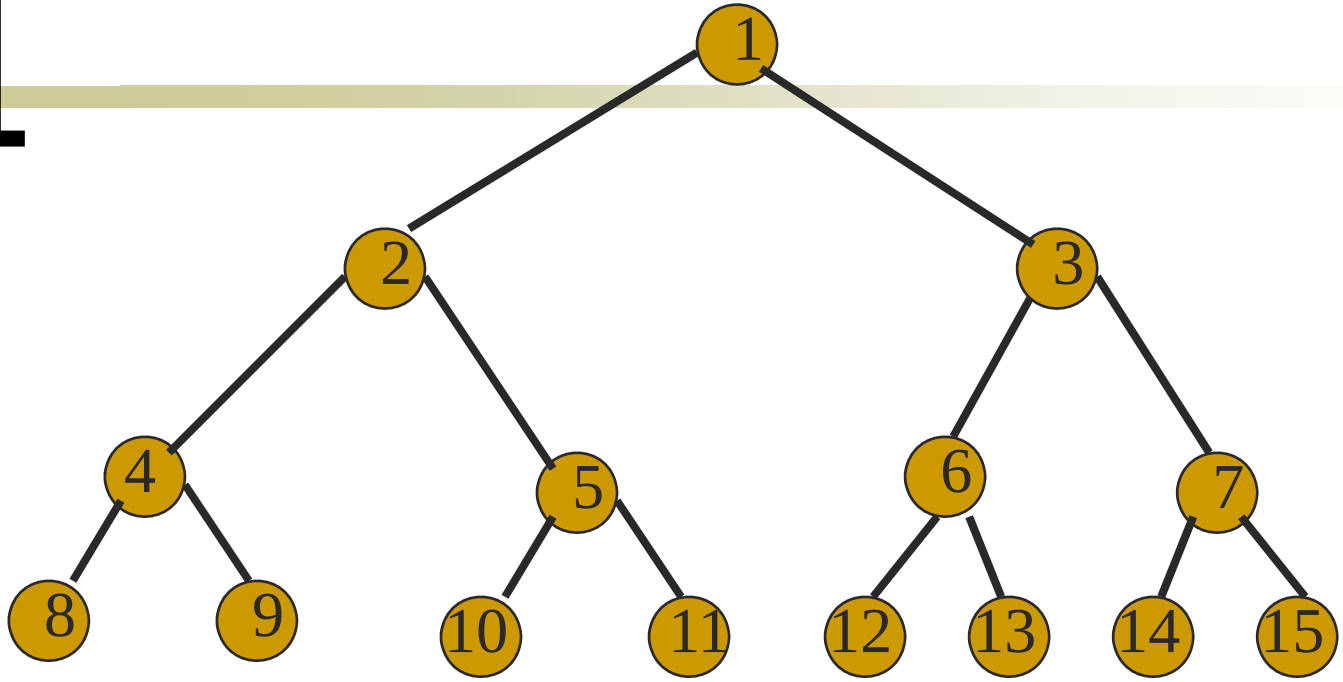
Node Number Properties



- گره چپ i در 2i میباشد.
- n تعدادگره های یک درخت باشد . اگر 2i > n باشد آنگاه i

دارای فرزند چپ نمی باشد.

Node Number Properties

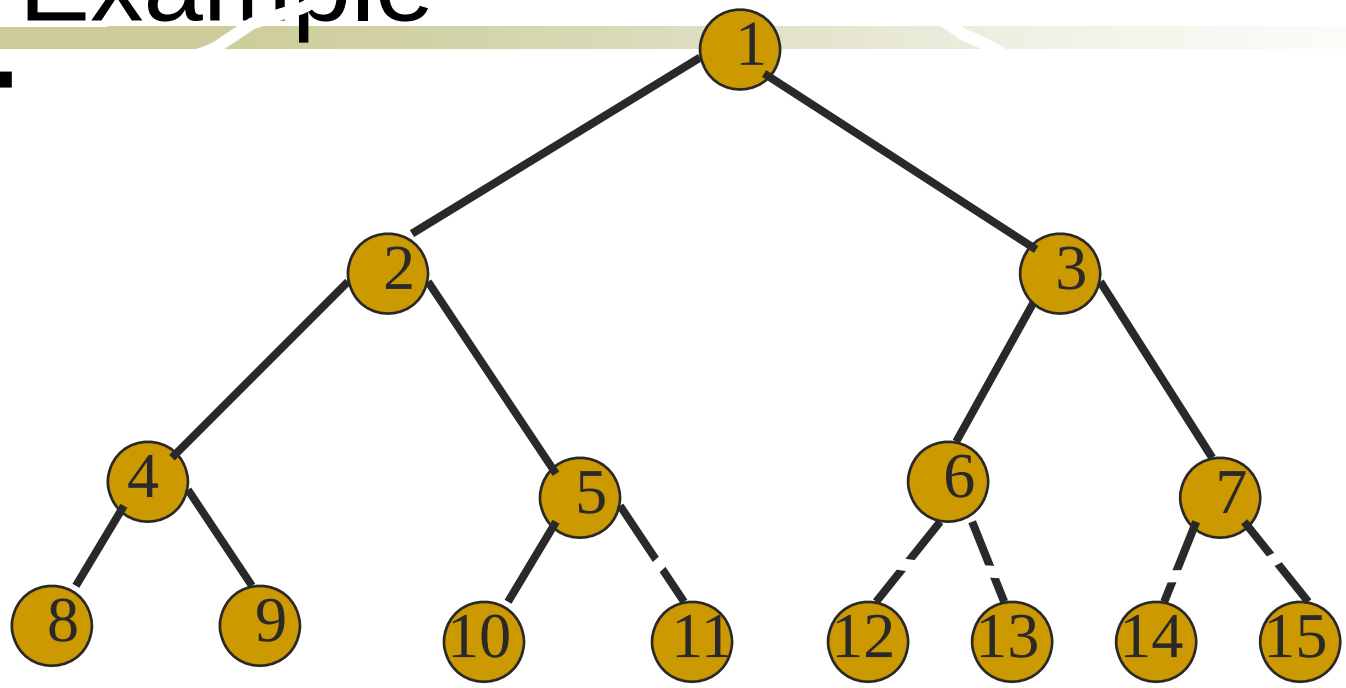


- گره راست i در 2i+1 میباشد.
- n تعدادگره های یک درخت باشد . اگر 2i+1 > n باشد آنگاه i دارای فرزند راست نمی باشد.

Complete Binary Tree With n Nodes

- در یک درخت باینری تعداد گره ها میان 1 تا n می باشد.
- اگر درخت شامل هر n تا باشد آنگاه به آن درخت درخت کامل گویند و سطر آخر کاملا پر نشده باشد, و گره ها ازچپ به راست قرار گرفته باشد.

Example



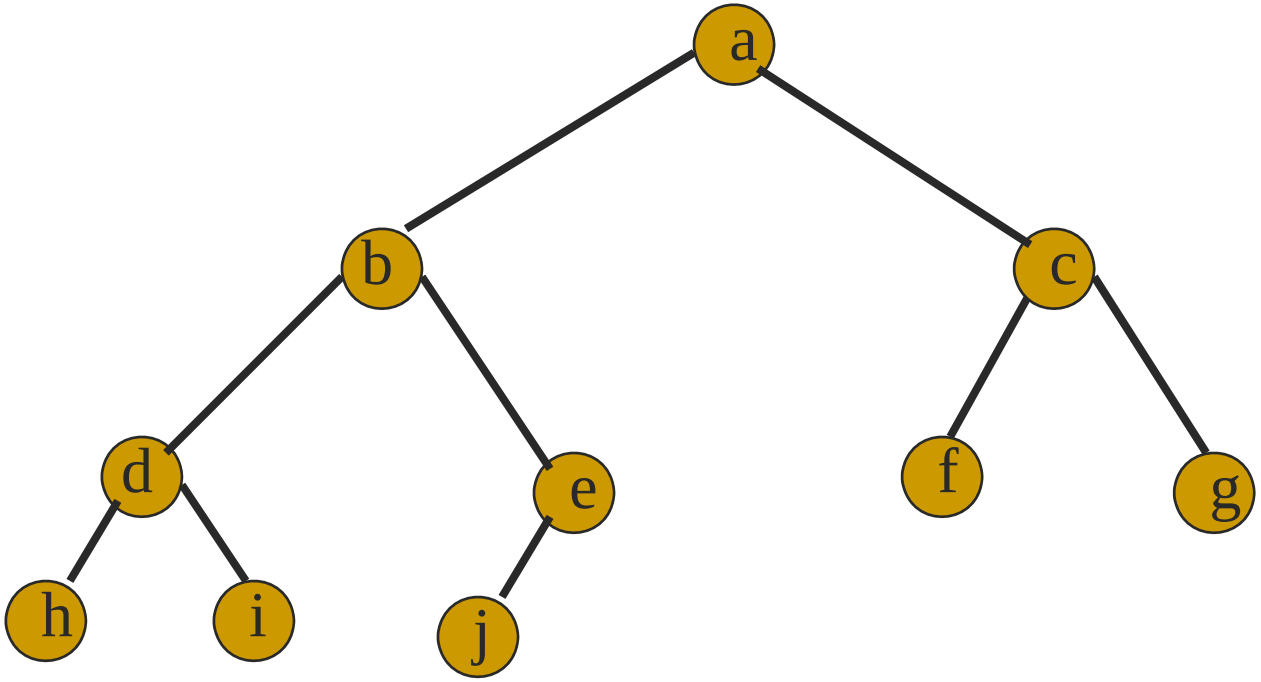
- مثال:درخت کامل را با 10 گره در نظر بگیرید.

Binary Tree Representation

- برای نمایش درخت دودویی می توان از آرایه استفاده کرد.
- نمایش با استفاده از لیست های پیوندی

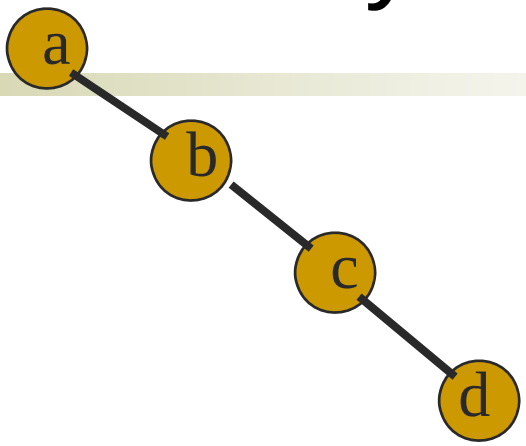
Array Representation

ابتداگره های یک درخت را به صورت سطحیLevel Order از چپ به راست شماره گذاری کرده سپس درخت را سطحی پیمایش کرده به این صورت که گره i در tree[i] قرار می گیرد.



tree[]		a	b	c	d	e	f	g	h	i	j

Right-Skewed Binary Tree



tree[]

	a	-	b	-	-	-	c	-	-	-	-	-	-	-	d

- برای نمایش یک درخت دودویی با استفاده ازیک آرایه با سایز $n+1$ تا 2^nنیاز داریم.

Linked Representation

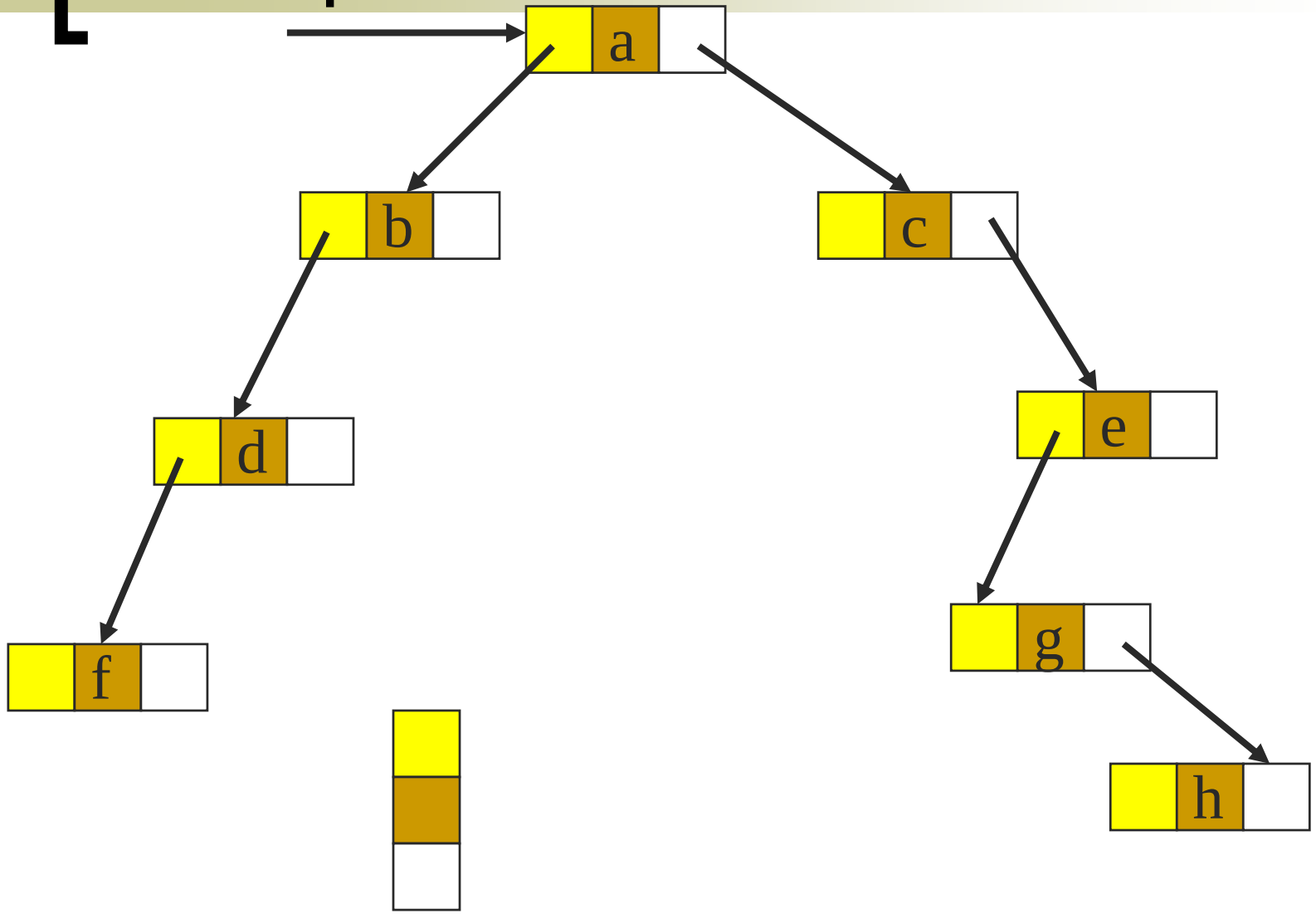
- تمام گره های درخت باینری در نمایش پیوندی به صورت یک نود نمایش داده می شود که دارای دو اشاره گر یکی به راست ویکی به چپ دارد.
- کل فضایی که نیاز داریم برای nگره درخت باینری جهت نمایش لیست پیوندی:

n * (space required by one node).

The Class BinaryTreeNode

```
package dataStructures;

BinaryTreeNode
{
Object element;
BinaryTreeNode leftChild; // زیر درخت چپ
BinaryTreeNode rightChild;// زیر درخت راست
}
```

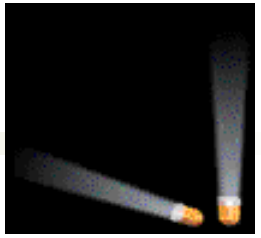
Linked Representation Example



Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level order

Binary Tree Traversal Methods



- در پیمایش درخت باینری هر گره فقط یک مرتبه visitمیشود.

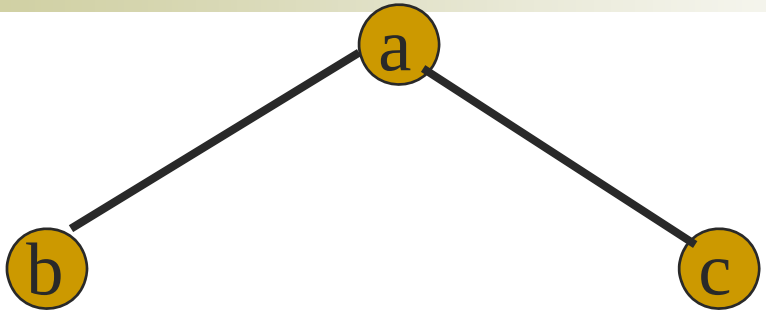
Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level order

Preorder Traversal

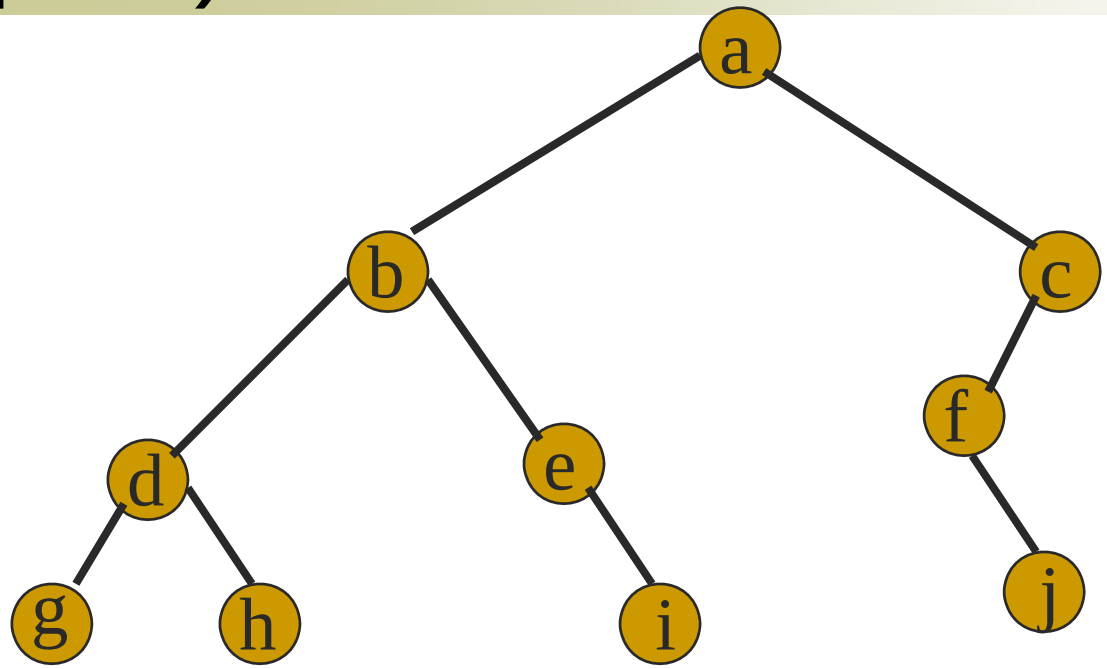
```
preOrder(BinaryTreeNode t)
{
(t != null)
{
visit(t);
preOrder(t.leftChild);
preOrder(t.rightChild);
}
}
```

Preorder Example (visit = print)



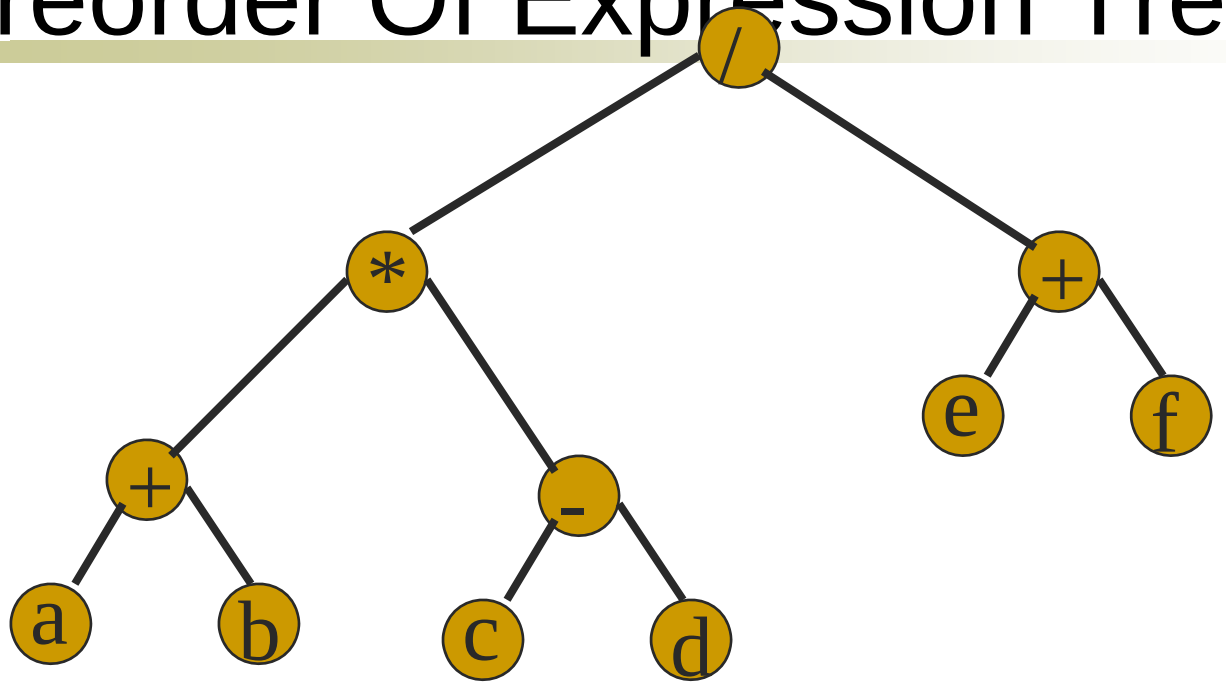
a b c

Preorder Example (visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



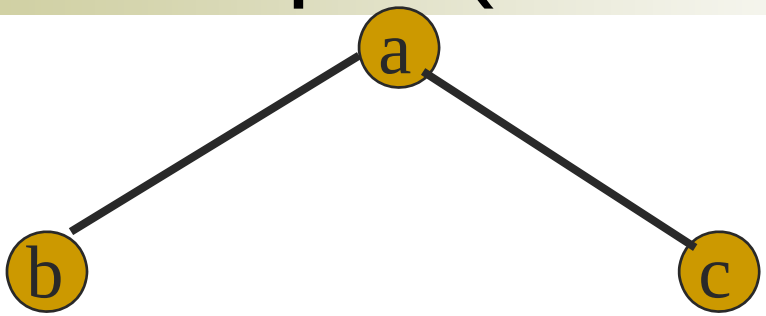
/ * + a b - c d + e f

معادل prefix در عبارت محاسباتی!

Inorder Traversal

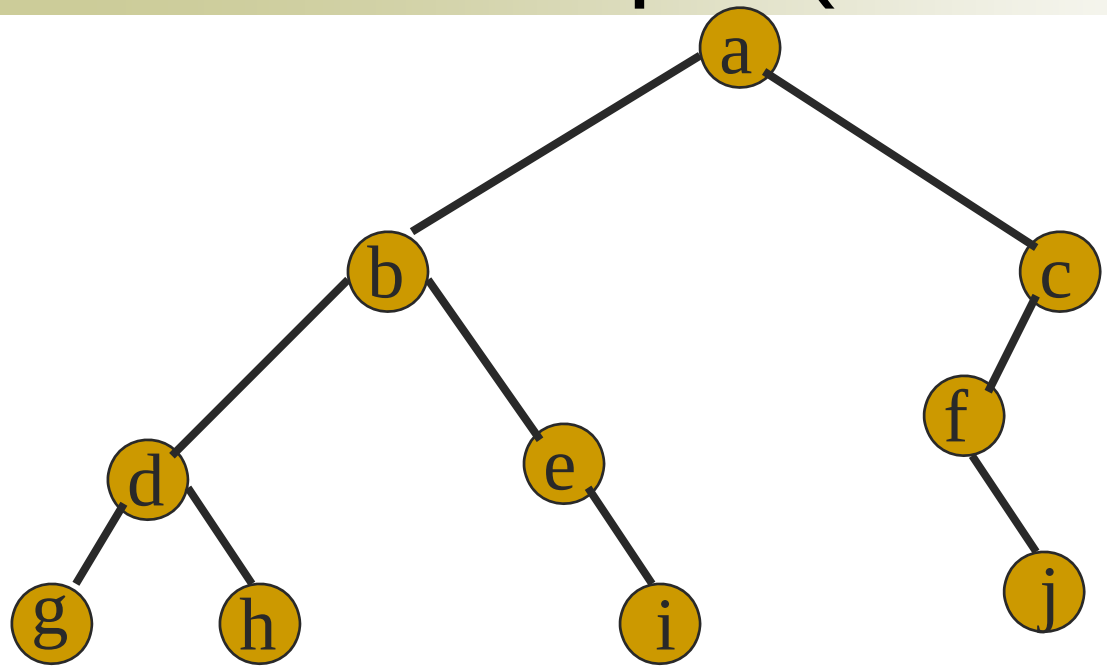
```
inOrder(BinaryTreeNode t)
{
(t != null)
{
inOrder(t.leftChild);
visit(t);
inOrder(t.rightChild);
}
}
```

Inorder Example (visit = print)



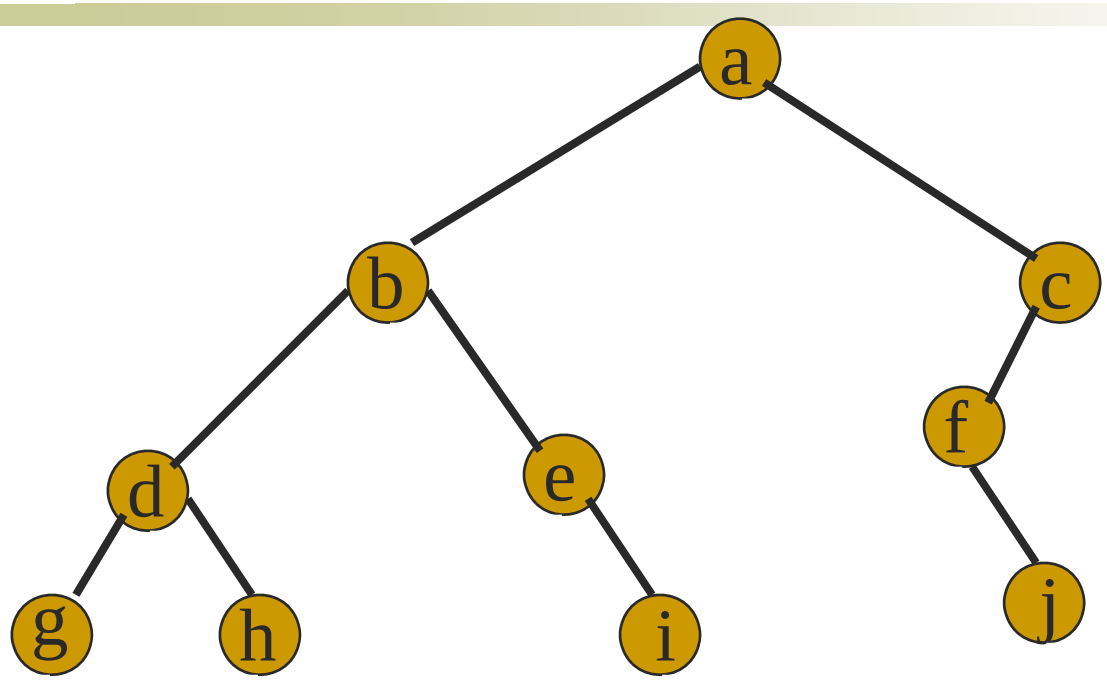
b a c

Inorder Example (visit = print)



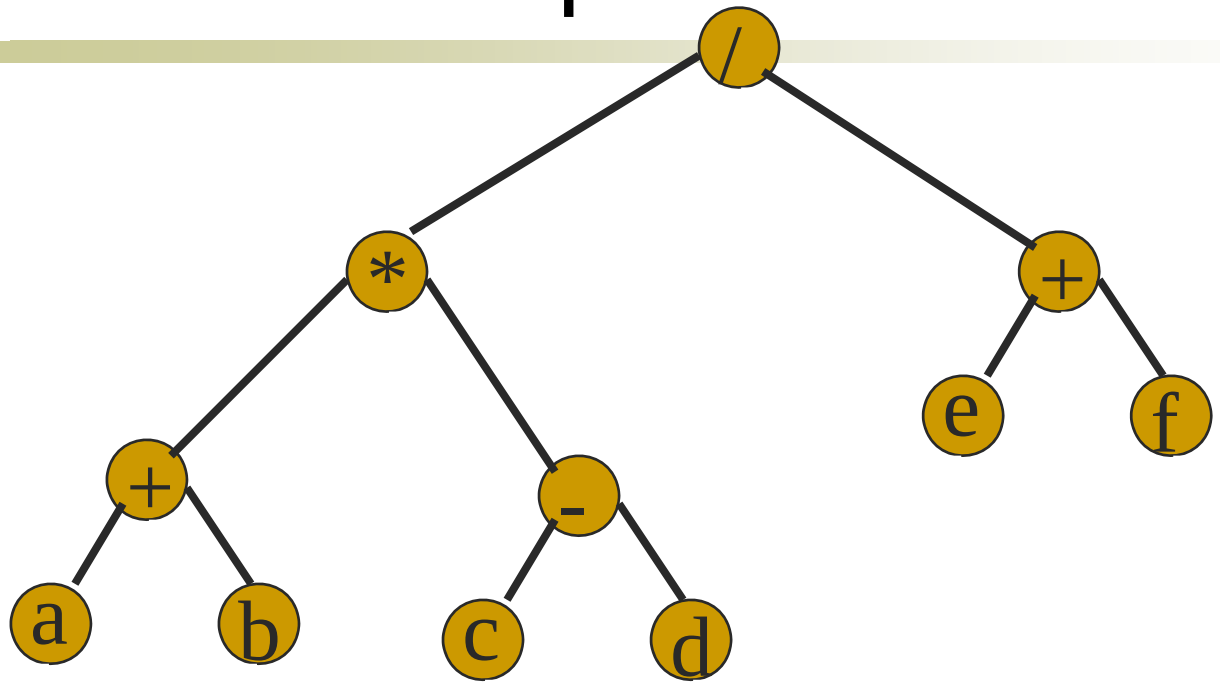
g d h b e i a f j c

Inorder By Projection (Squishing)



g d h b e i a f j c

Inorder Of Expression Tree

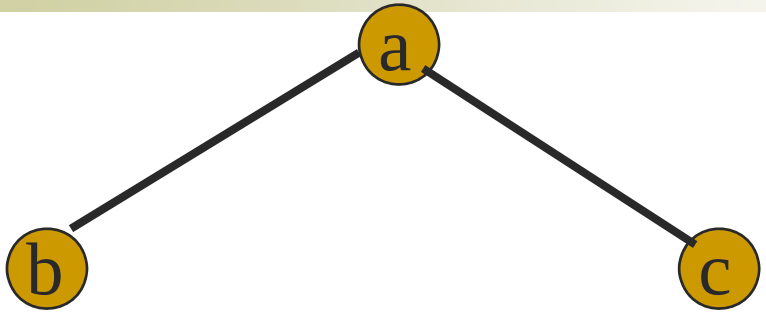


a + b * c - d / e + f

Postorder Traversal

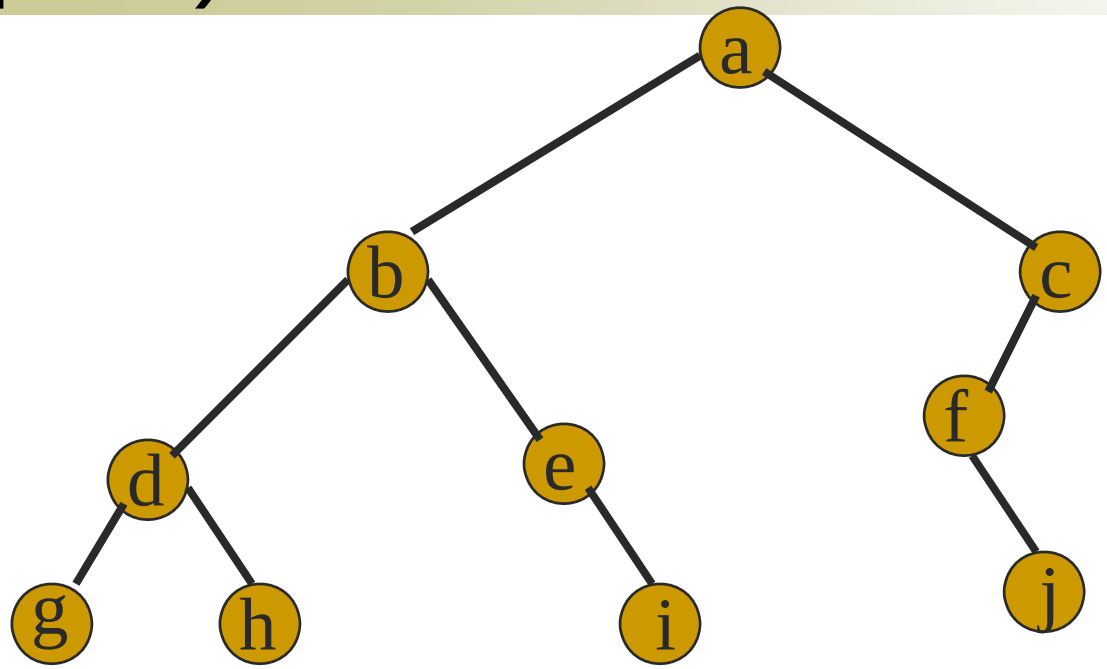
```
postOrder(BinaryTreeNode
                        t)
                        {
(t != null)
                        {
  postOrder(t.leftChild);
   postOrder(t.rightChild);
                    visit(t);
                        }
                        }
```

Postorder Example (visit = print)



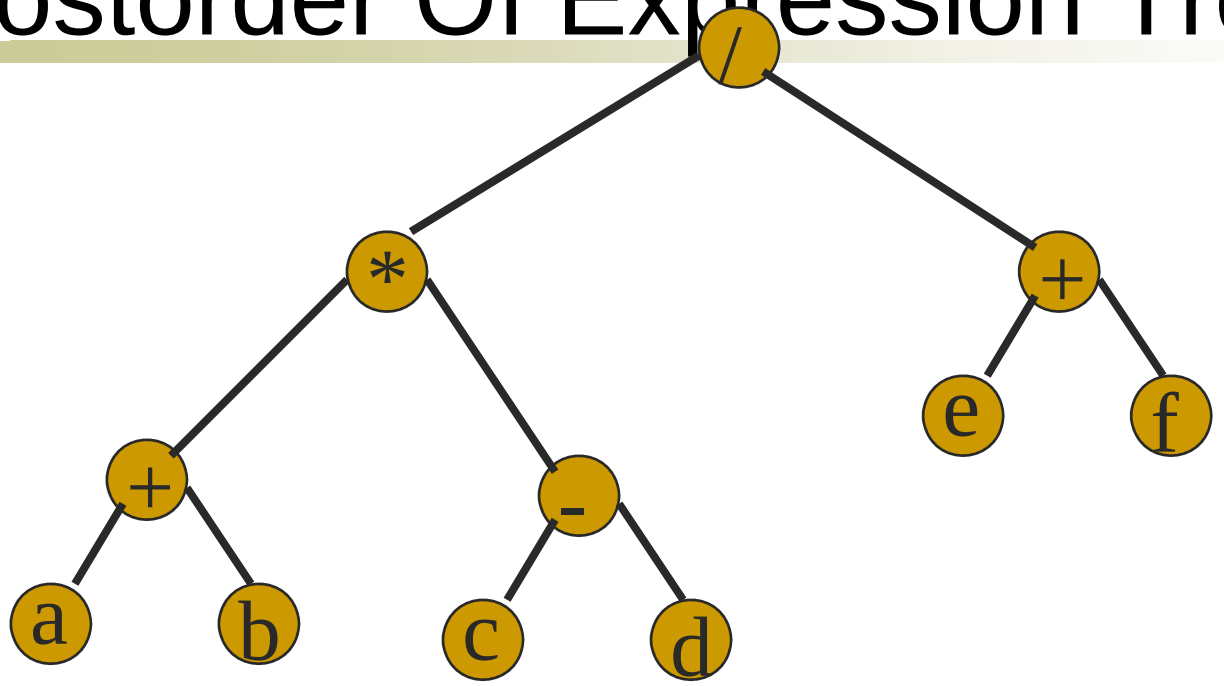
b c a

Postorder Example (visit = print)



g h d i e b j f c a

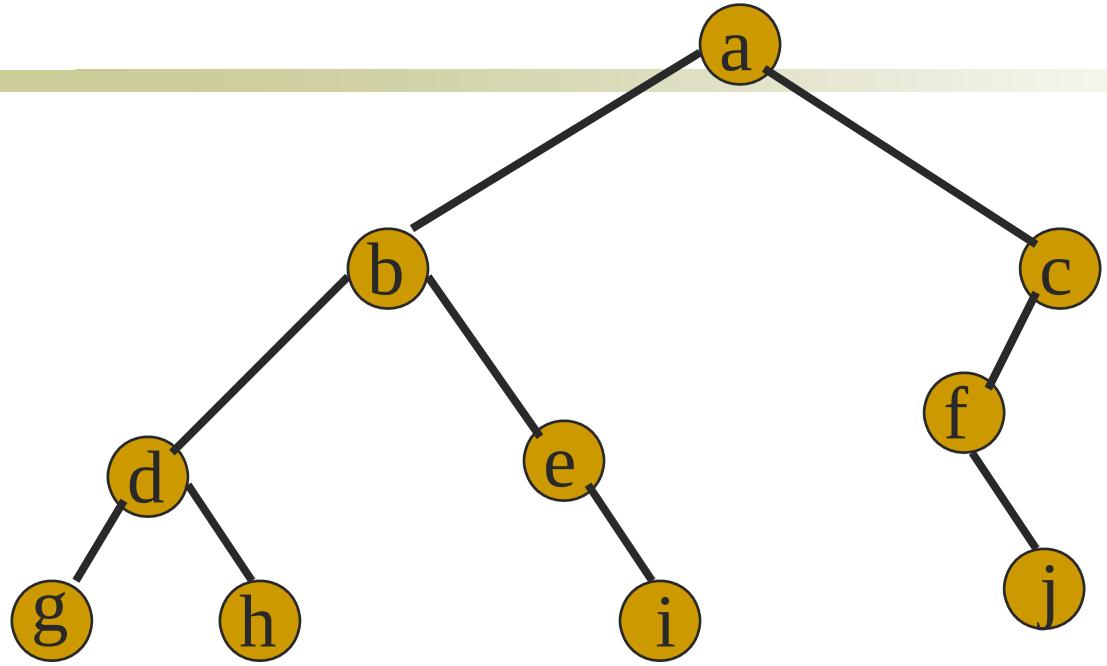
Postorder Of Expression Tree



a b + c d - * e f + /

معادل postfix در عبارت محاسباتی!

Traversal Applications



Level Order

Let t be the tree root.

(t != null)

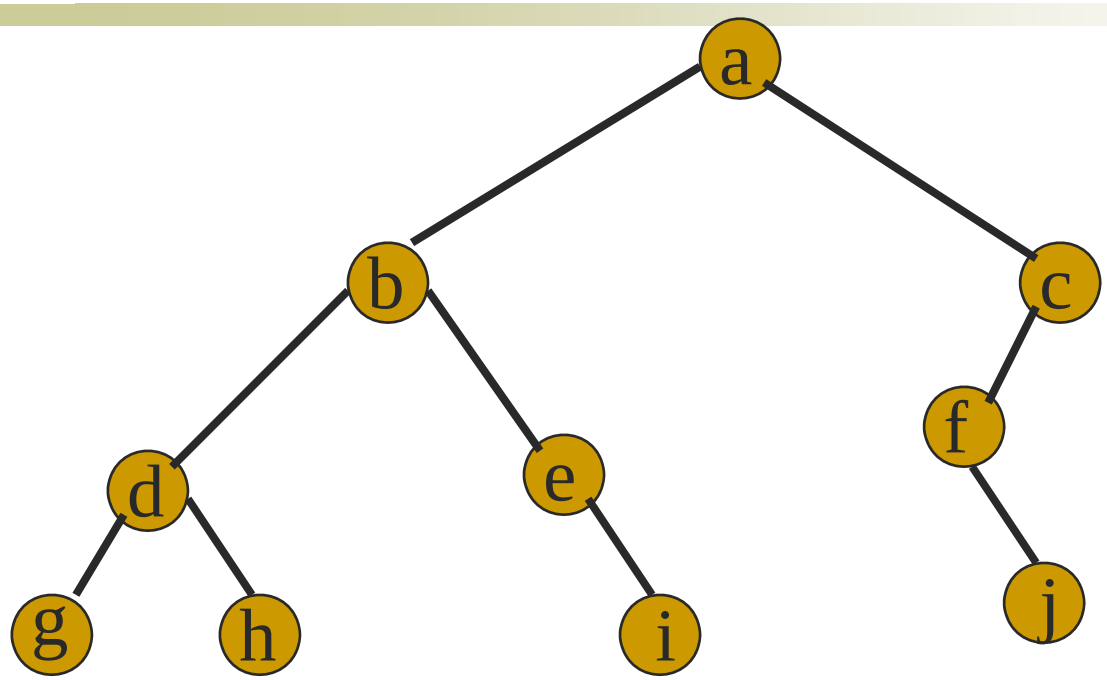
{

visit t and put its children on a FIFO queue;

remove a node from the FIFO queue and call it t;

}

Level-Order Example (visit = print)



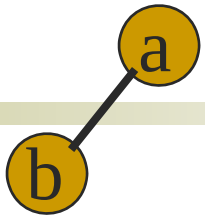
a b c d e f g h i j

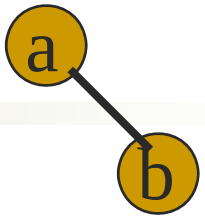
Some Examples

preorde

r =

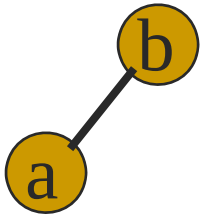
ab

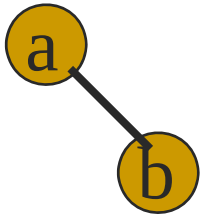




inorder

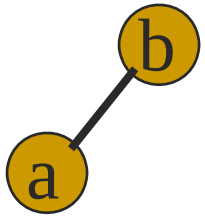
= ab

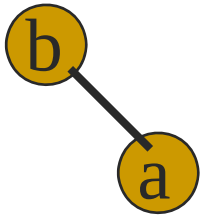




postorder

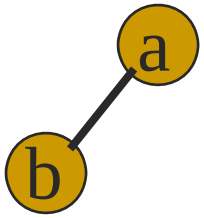
= ab

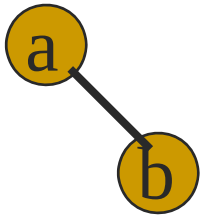




level order

= ab

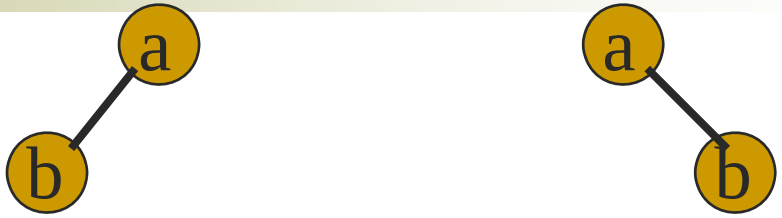




Preorder And Postorder

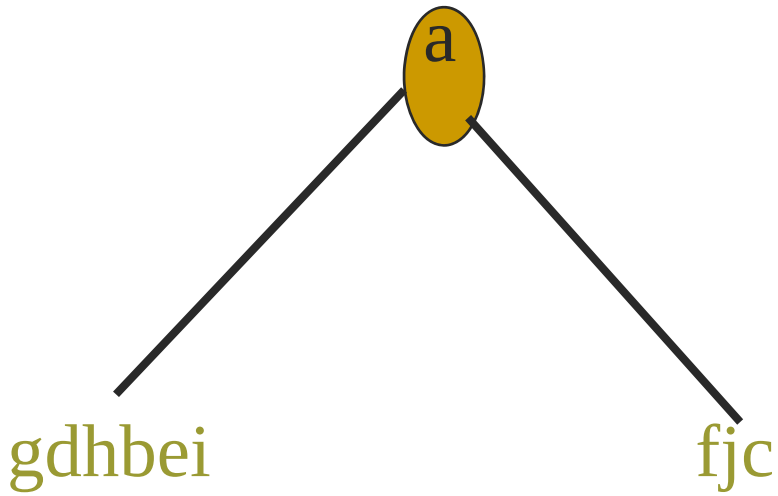
preorder = ab

postorder = ba

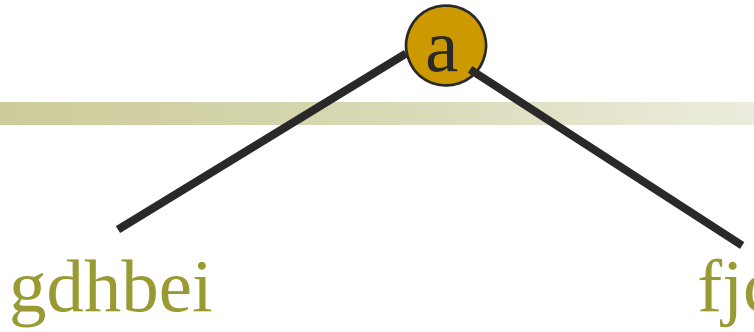


Inorder And Preorder

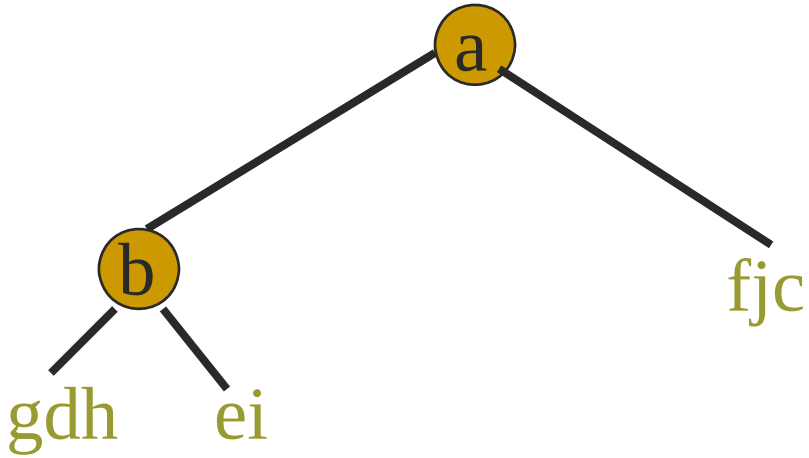
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j



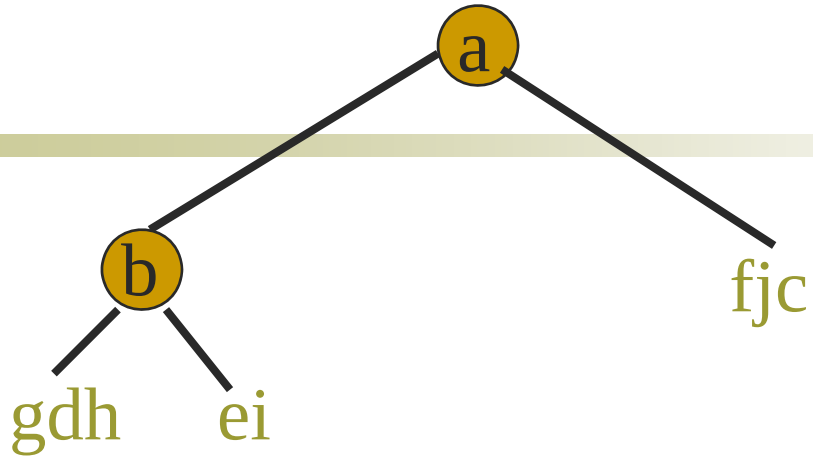
Inorder And Preorder



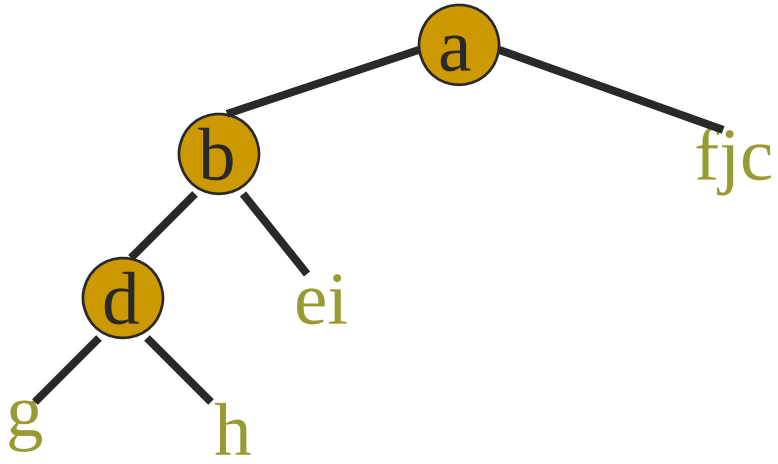
- preorder = b d g h e i c f j
- bریشه بعدی است/ gdh زیر درخت چپ / ei زیر درخت راست



Inorder And Preorder



- preorder = d g h e i c f j
- dریشه بعدی می باشد./g زیر درخت چپ/وhزیر درخت راست می باشد.



Inorder And Postorder

- در level orderپیمایش از چپ به راست می باشد
- در Post orderپیمایش از راست به چپ می باشد.

- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a;
- gdhbei // زیر درخت چپ
- fjc// زیردرخت راست

Inorder And Level Order

- در level order پیمایش از چپ به راست می باشد.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a;
- gdhbei// زیر درخت چپ
- fjc// زیردرخت راست

Priority Queues

- دونوع صف اولویت دار تعریف می کنیم:
- صف با الویت بالا
- صف با الویت پایین

Min Priority Queue

- یک مجموعه از عناصر میباشد.
- برای هر عنصر یک کلید اولویت تعریف می کنیم.
- اعمال زیر را می توان روی آن انجام داد.
  - چک کردن اینکه صف خالی است یا نه؟
  - سایز و اندازه صف
  - اضافه کردن یک عنصر به لیست
  - پیدا کردن عنصر با کمترین اولویت
  - حذف عنصری با کمترین اولویت

Max Priority Queue

- یک مجموعه از عناصر میباشد.
- برای هر عنصر یک کلید اولویت تعریف می کنیم.
- اعمال زیر را می توان روی آن انجام داد.
  - چک کردن اینکه صف خالی است یا نه؟
  - سایز و اندازه صف
  - اضافه کردن یک عنصر به لیست
  - پیدا کردن عنصر با بیشترین اولویت
  - حذف عنصری با بیشترین اولویت

Complexity Of Operations

دو نمونه خوب می توان از heaps leftist و leftist trees نام برد.

isEmpty, size, and get => O(1) time

put and remove => O(log n) time

که در آن n طول لیست می باشد

Applications

Sorting

- کلید هر عنصر به عنوان اولویت آن محسوب می شود.
- عناصر را مرتب شده در لیست اولویت قرار دهید
- گزینش عنصر با توجه به کلید اولویت آن:
  - اگر عنصری با کمترین اولویت را خواستیم انتخاب کنیم باید صف به صورت صعودی مرتب شده باشد.
  - اگر عنصری با بیشترین اولویت را خواستیم انتخاب کنیم باید صف به صورت نزولی مرتب شده باشد.

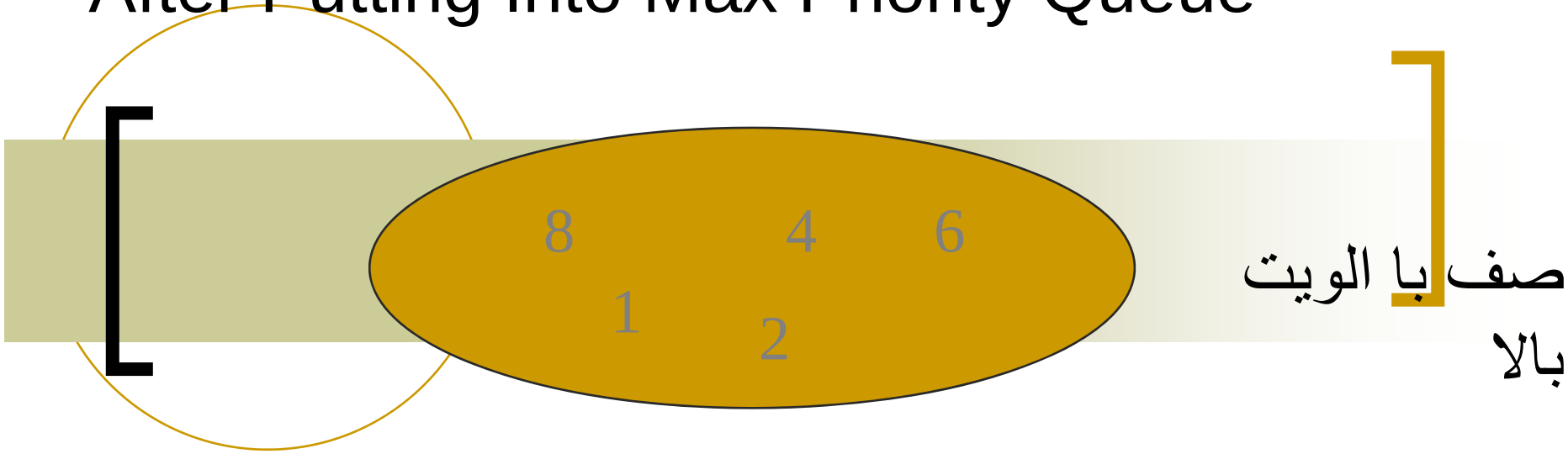
Sorting Example

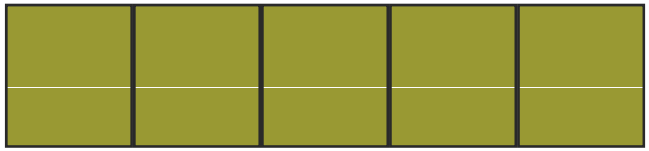
مرتب کنیدپنج عنصری که کلید های آن به این قرار باشد:

6, 8, 2, 4, 1

- از صف با الویت بالا استفاده کنید.
  - ابتدا عناصر را در یک صف ب الویت بالا قرار دهید
  - پنج مرتبه عمل حذف را انجام دهید در نهایت عناصر از راست به چپ مرتب می شوند.

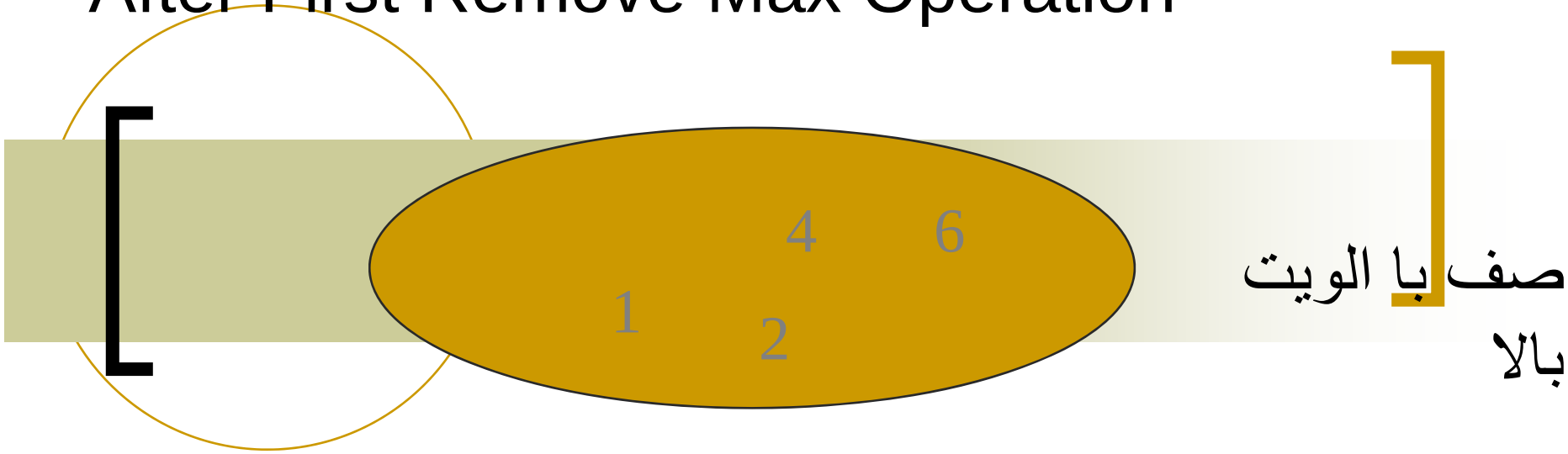
After Putting Into Max Priority Queue

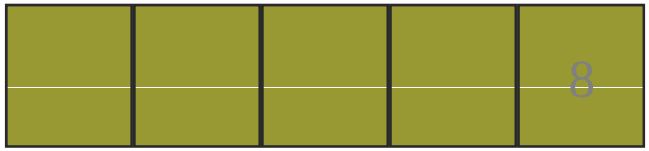




Sorted Array

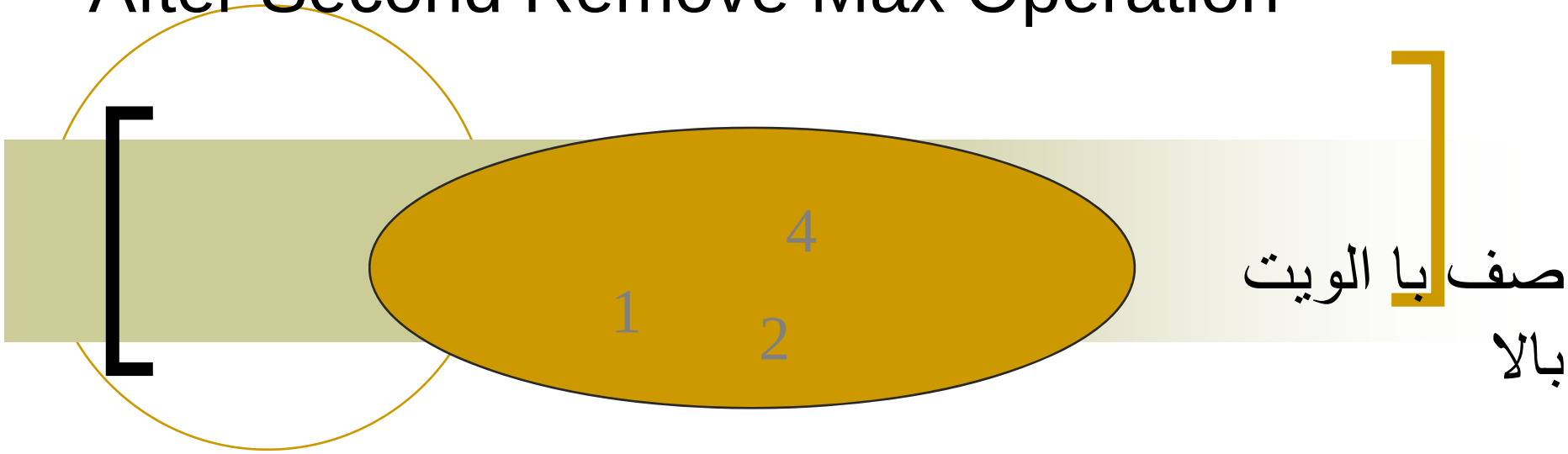
After First Remove Max Operation

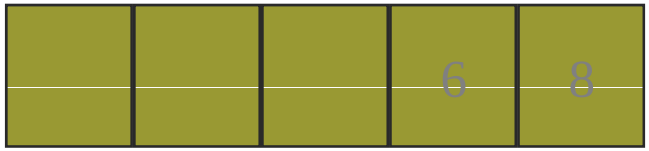




Sorted Array

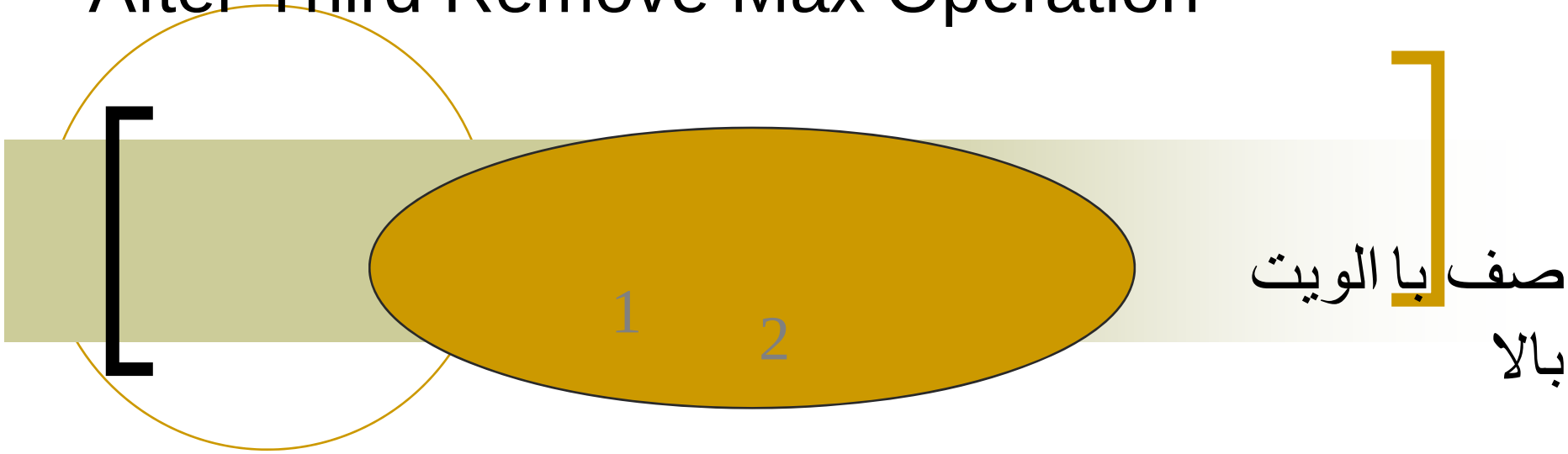
After Second Remove Max Operation

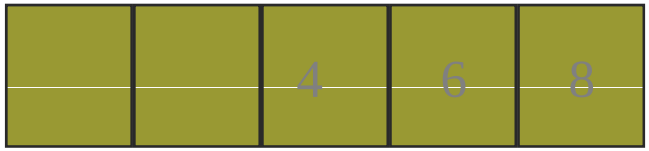




Sorted Array

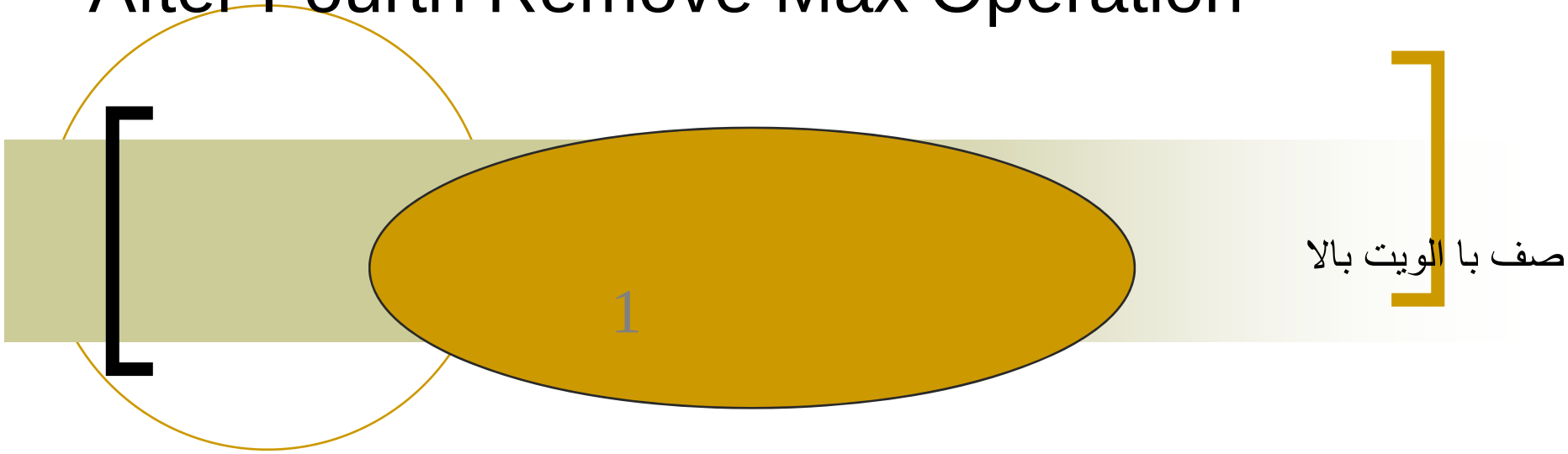
After Third Remove Max Operation

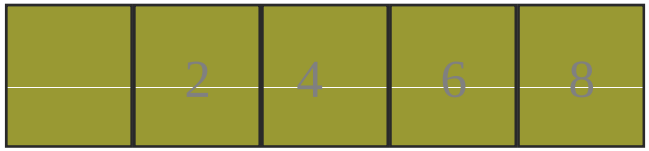




Sorted Array

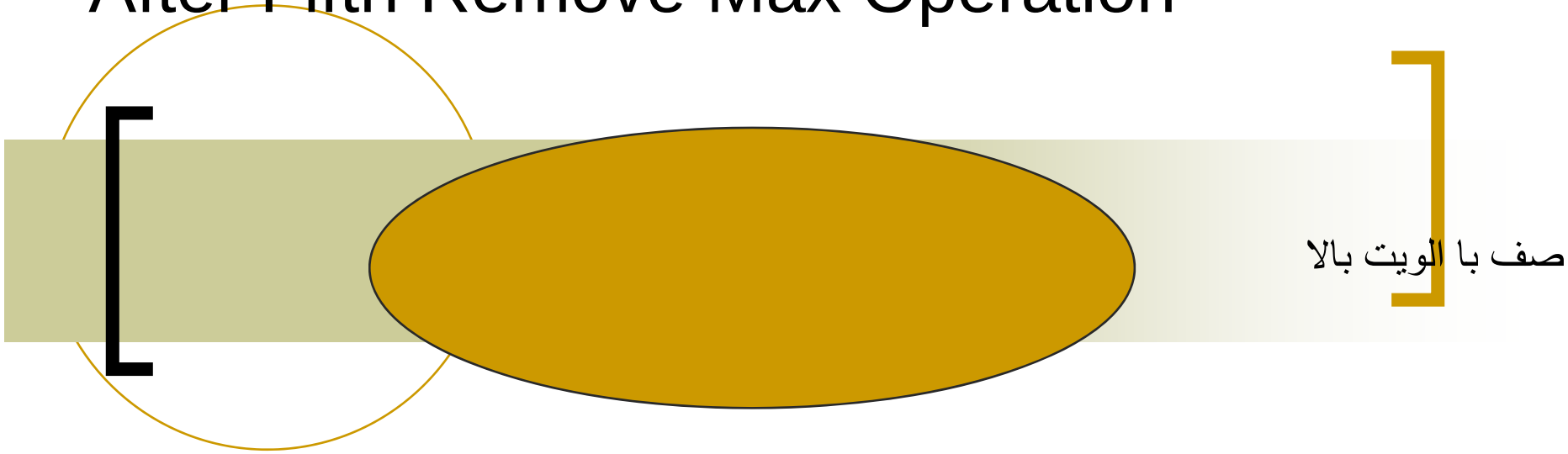
After Fourth Remove Max Operation

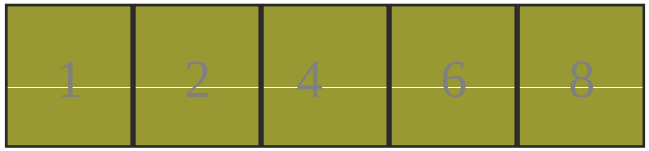




Sorted Array

After Fifth Remove Max Operation





Sorted Array

Complexity Of Sorting

مرتب سازی nعنصر

- n put operations => O(n log n) time.
- n remove max operations => O(n log n) time.
- total time is O(n log n).
- compare with $O(n^2)$

Heap Sort

کاربرد صف با الویت بالا در heap دیده می شود.

این مرتب سازی با O(n) قابل انجام است.

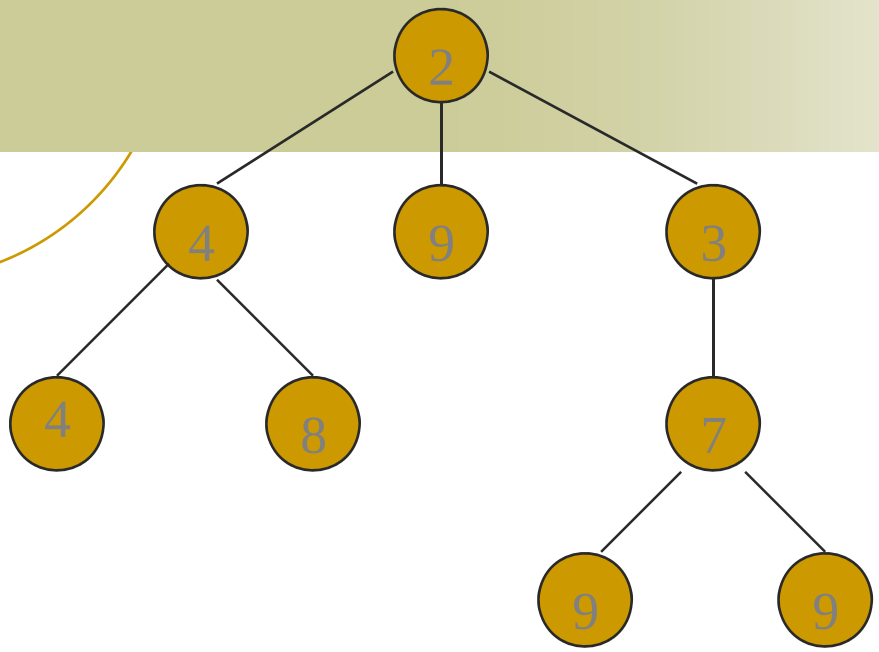
Min Tree Definition

هر درختی دارای value یا یک مقدار می باشد

Min tree:درختی است که در آن valueریشه از value

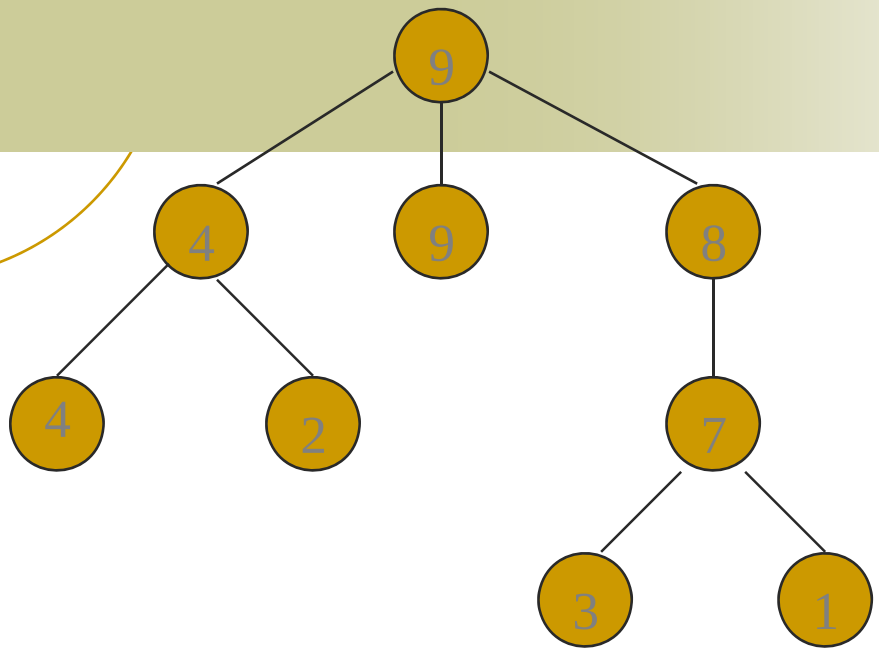
تمام زیر درختها با فرزندانش کمتر باشد.

Min Tree Example



ریشه کمترین مقدار را دارد

Max Tree Example

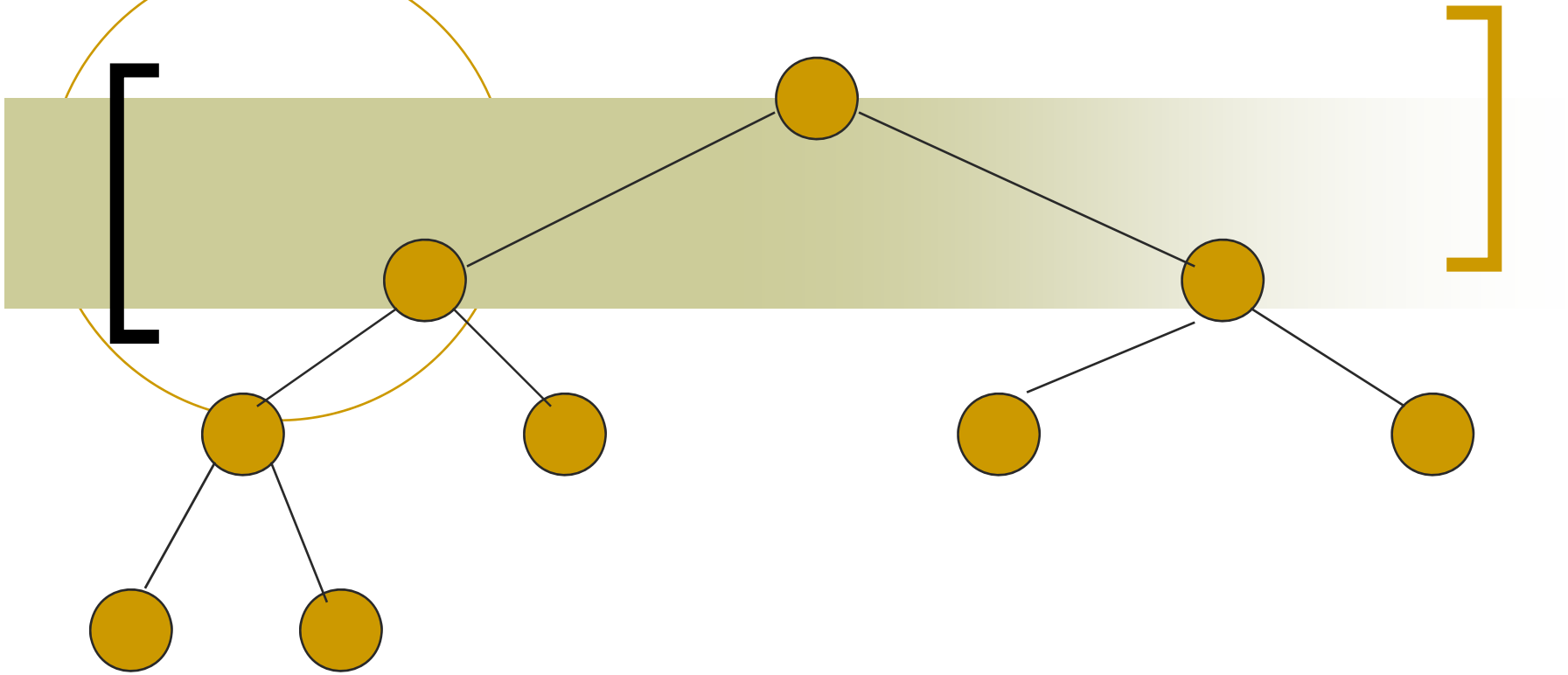


ریشه بیشترین مقدار را دارد

Min Heap Definition

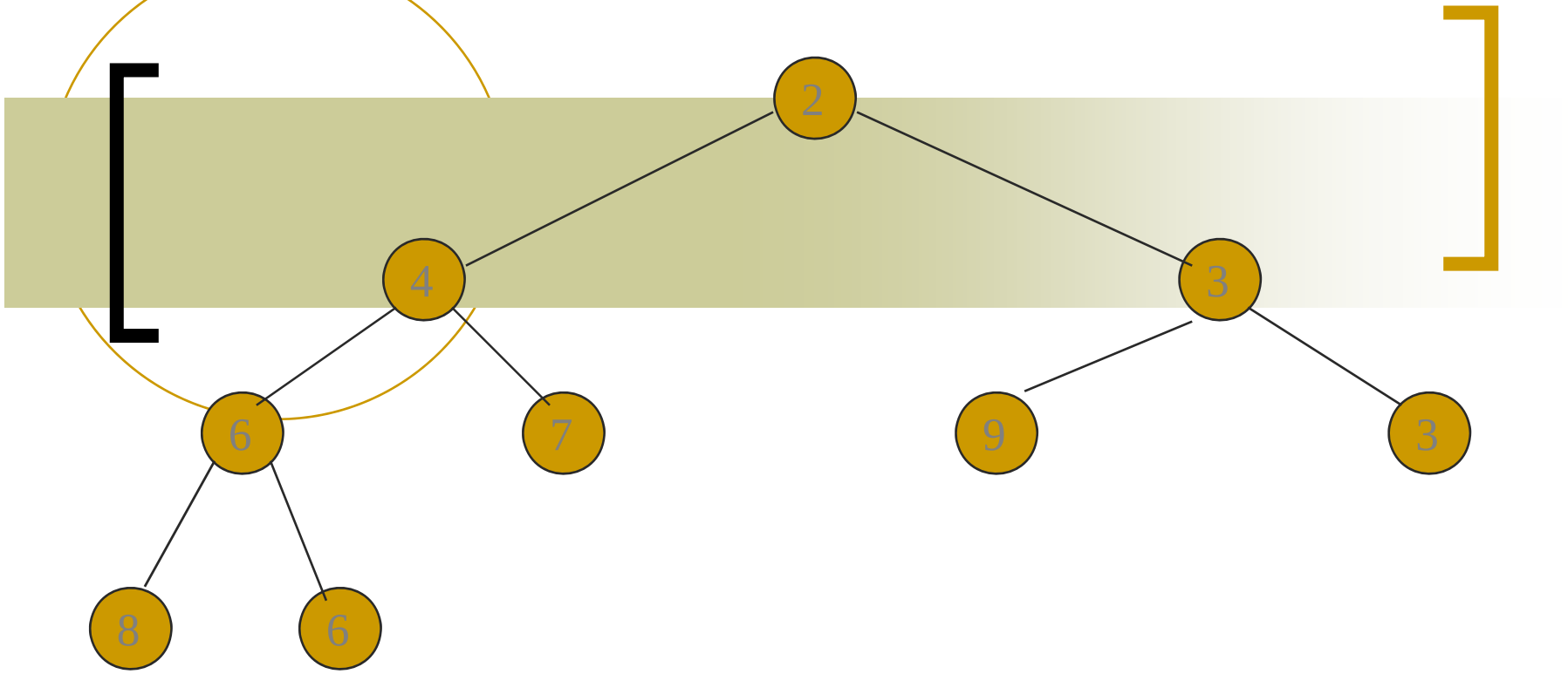
- درخت باینری کامل
- min tree

Min Heap With 9 Nodes



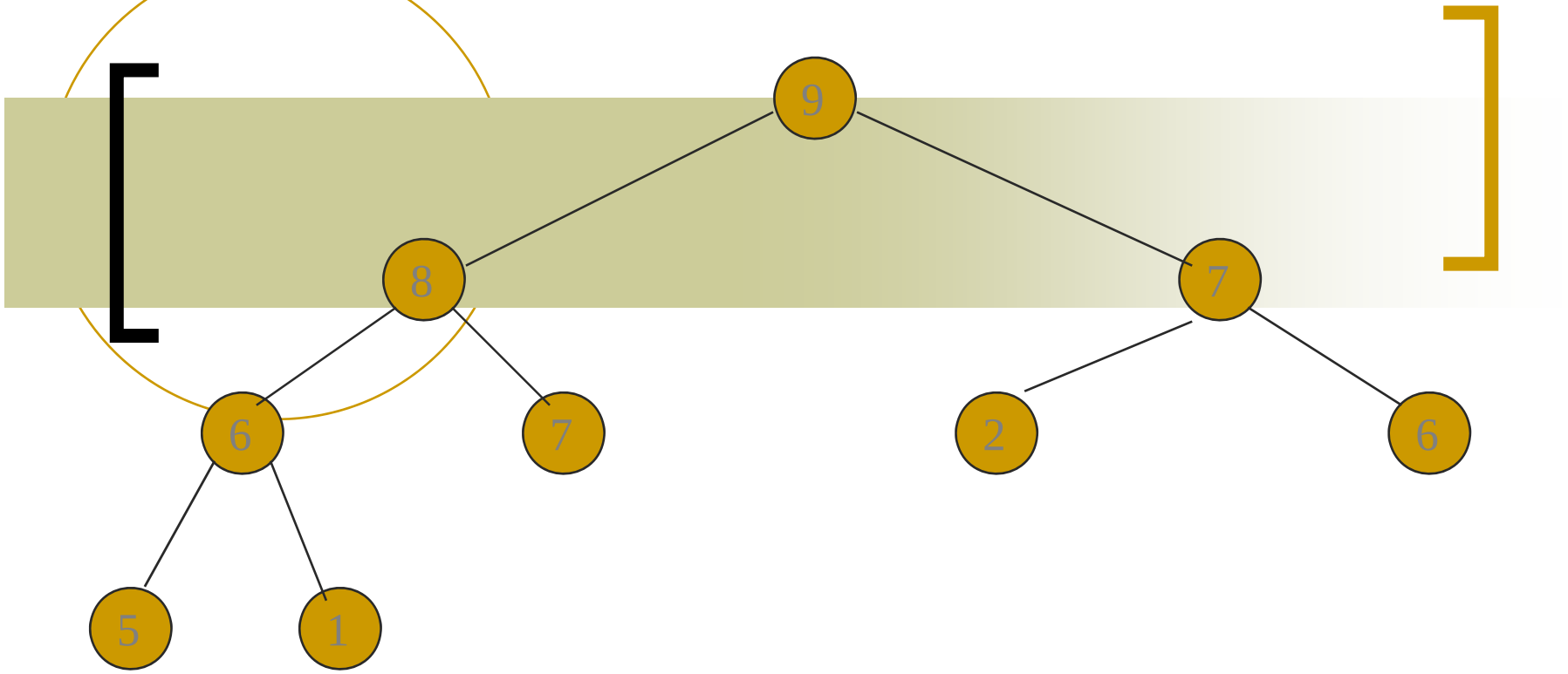
درخت کامل با 9 گره

Min Heap With 9 Nodes



درخت کامل با 9 گره و همچنین min tree

Max Heap With 9 Nodes

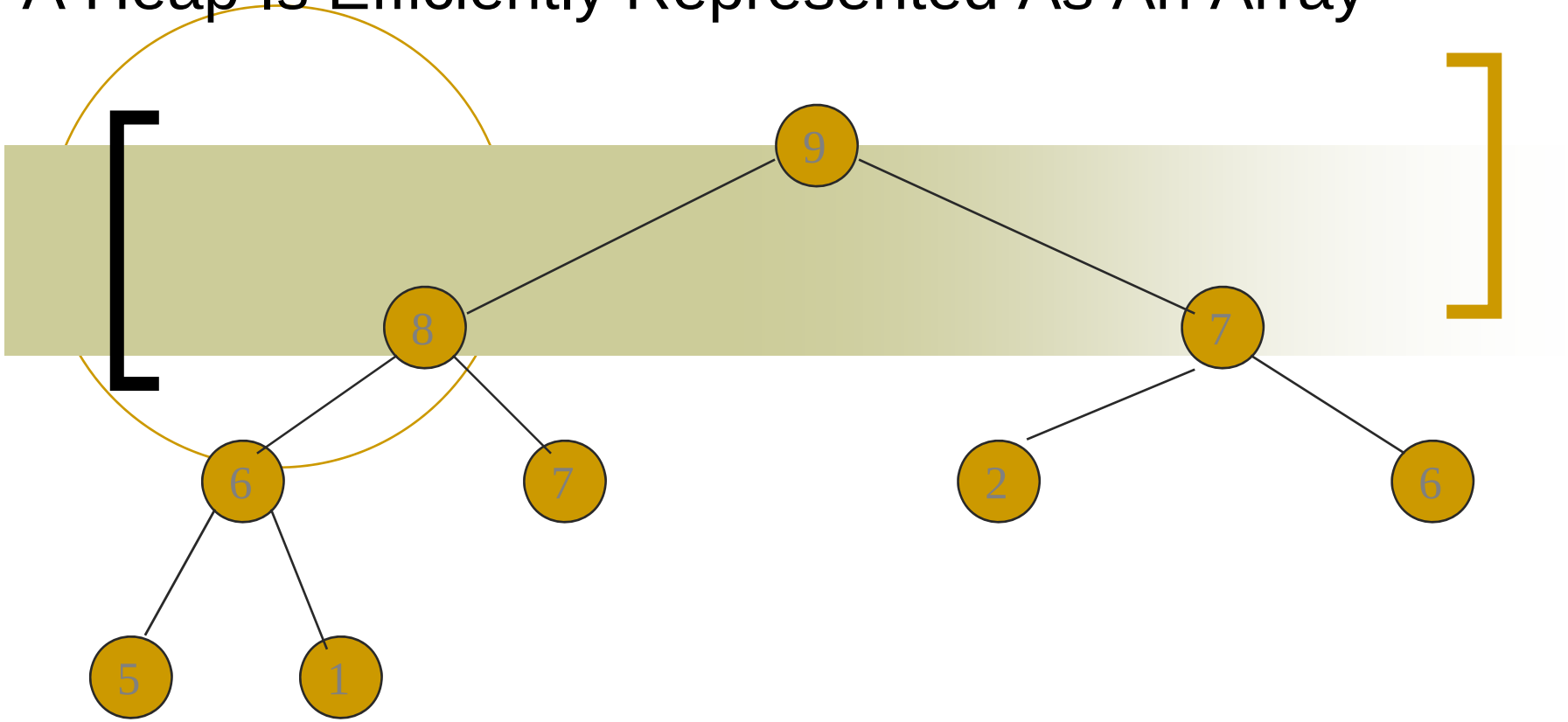


درخت کامل با 9 گره و همچنین max tree

Heap Height

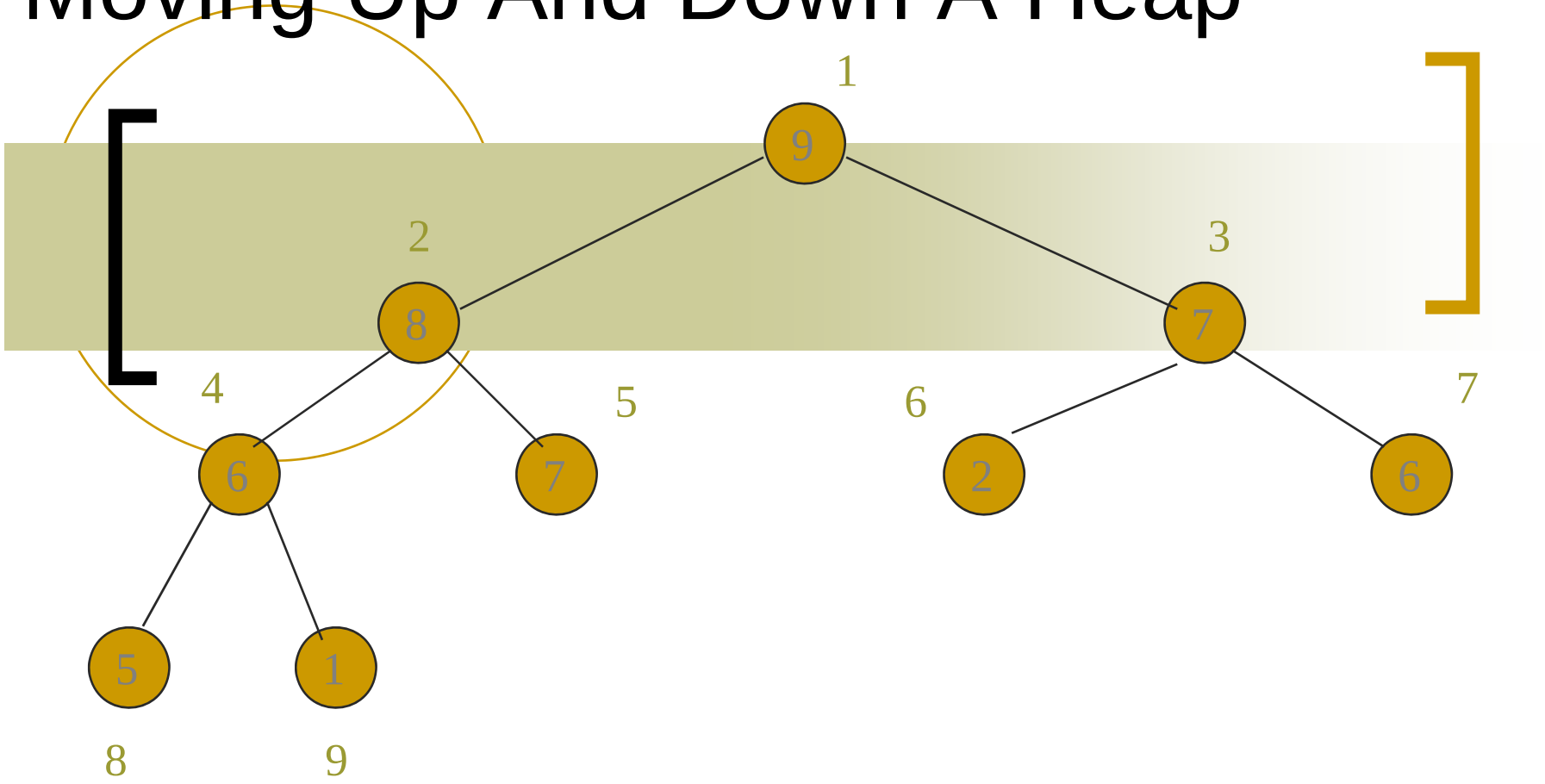
درختی کامل می باشد که ارتفاع ان Heap
است $\log_2 (n+1)$.

A Heap Is Efficiently Represented As An Array

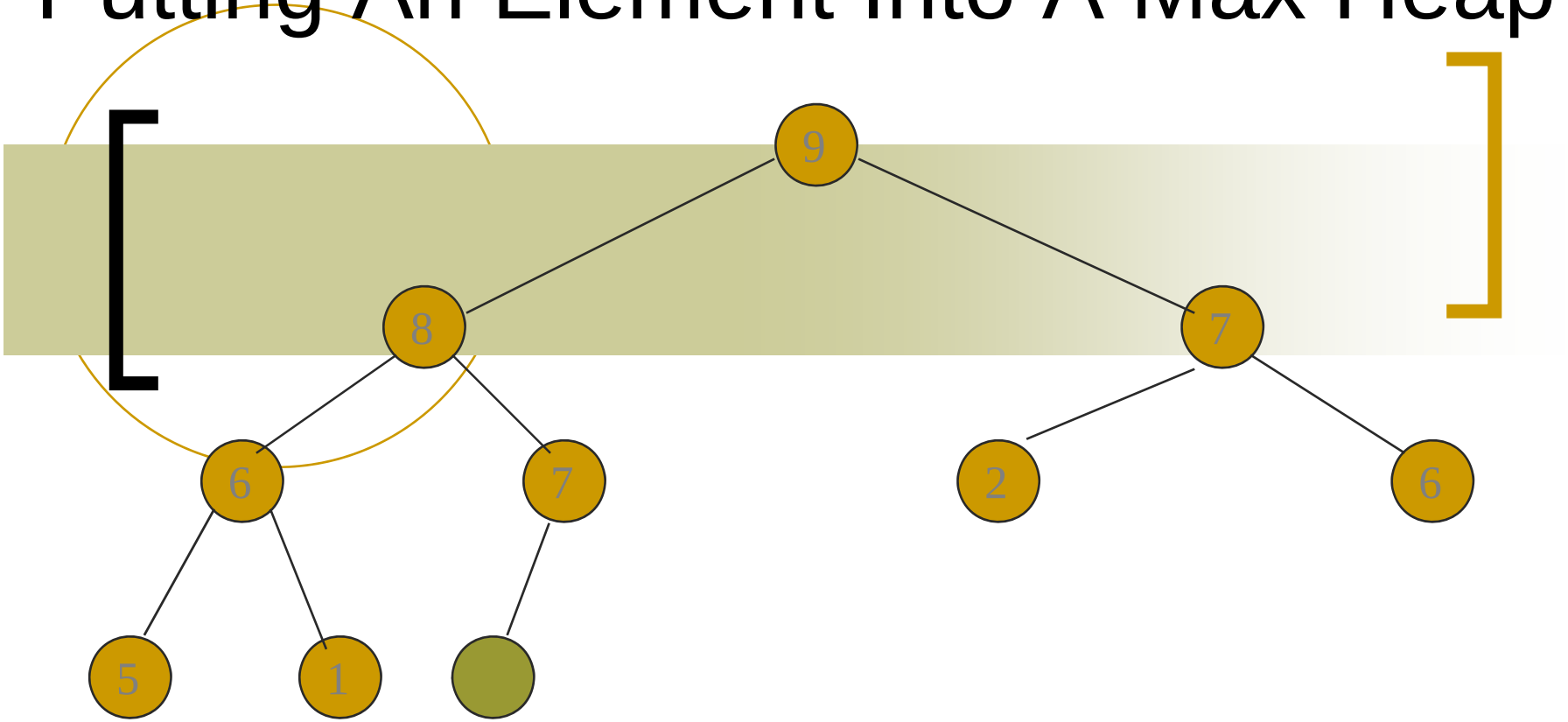


0 1 2 3 4 5 6 7 8 9 10

Moving Up And Down A Heap

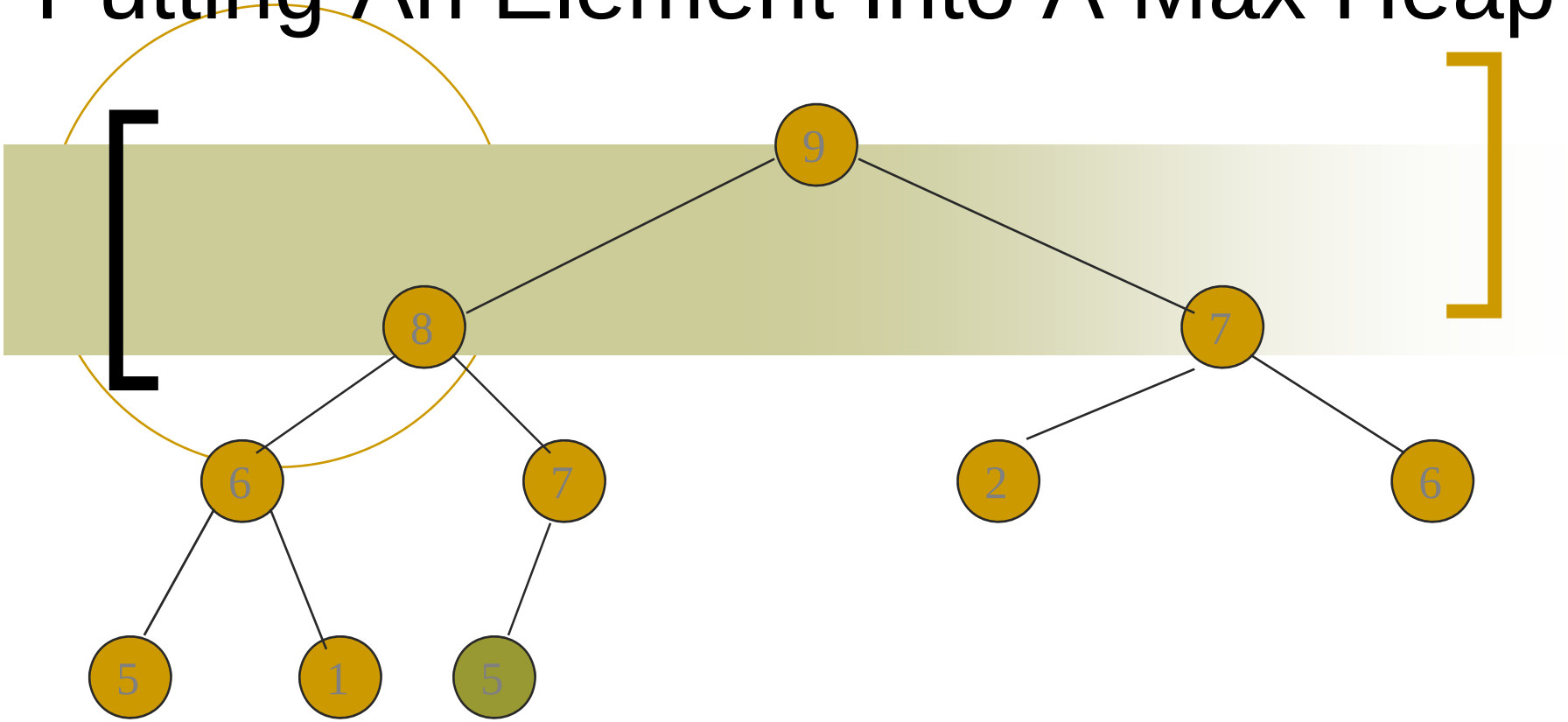


Putting An Element Into A Max Heap



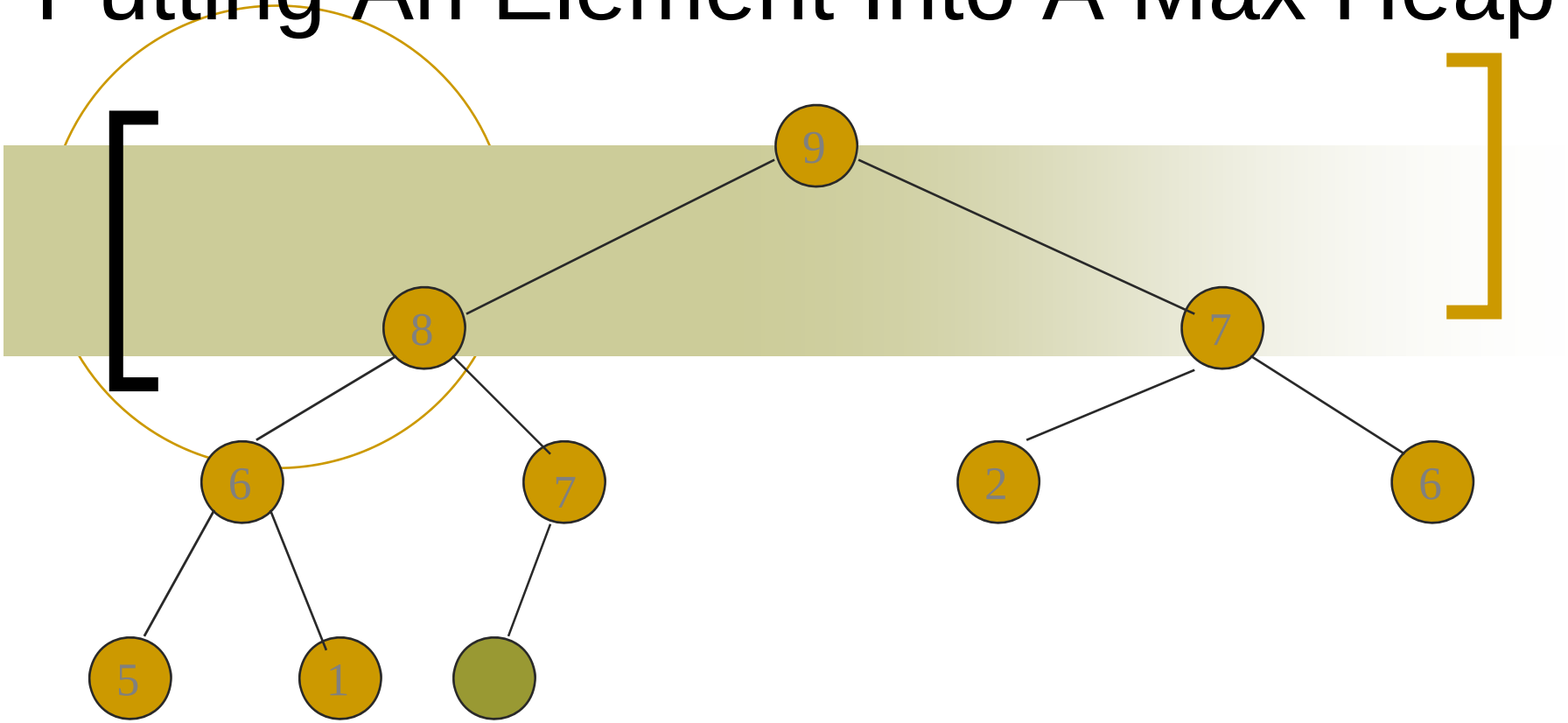
درخت کامل با 10گره

Putting An Element Into A Max Heap



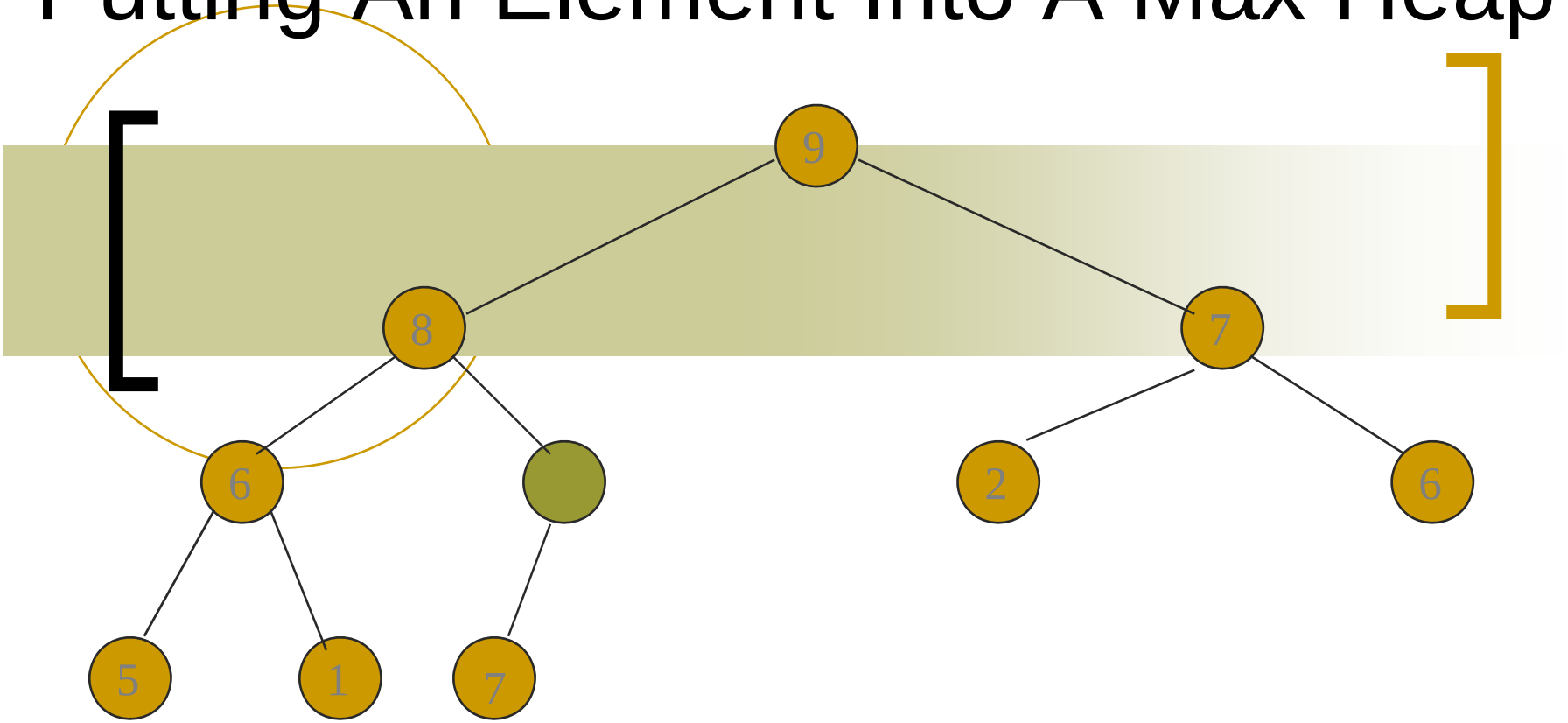
5 گره جدید می باشد

Putting An Element Into A Max Heap



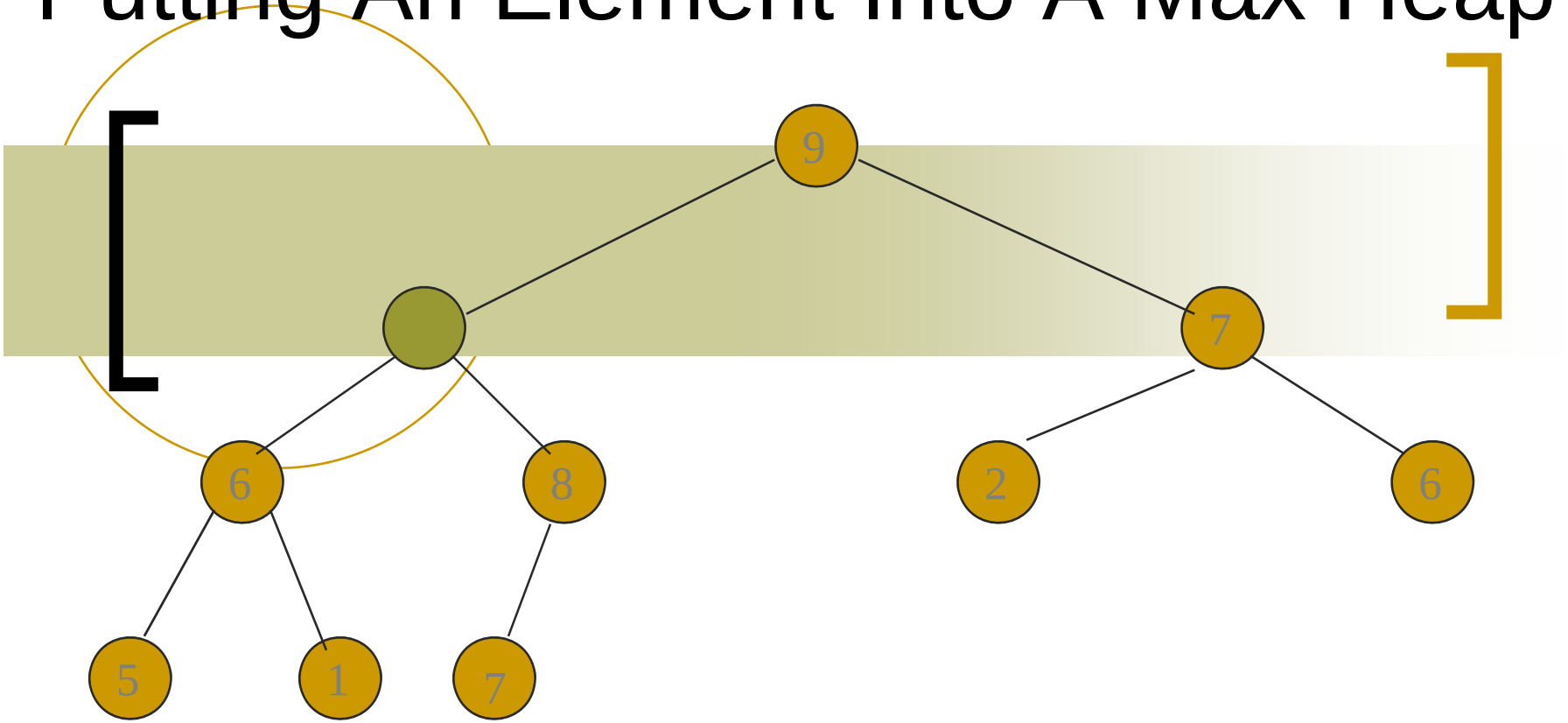
20 گره جدید می باشد

Putting An Element Into A Max Heap



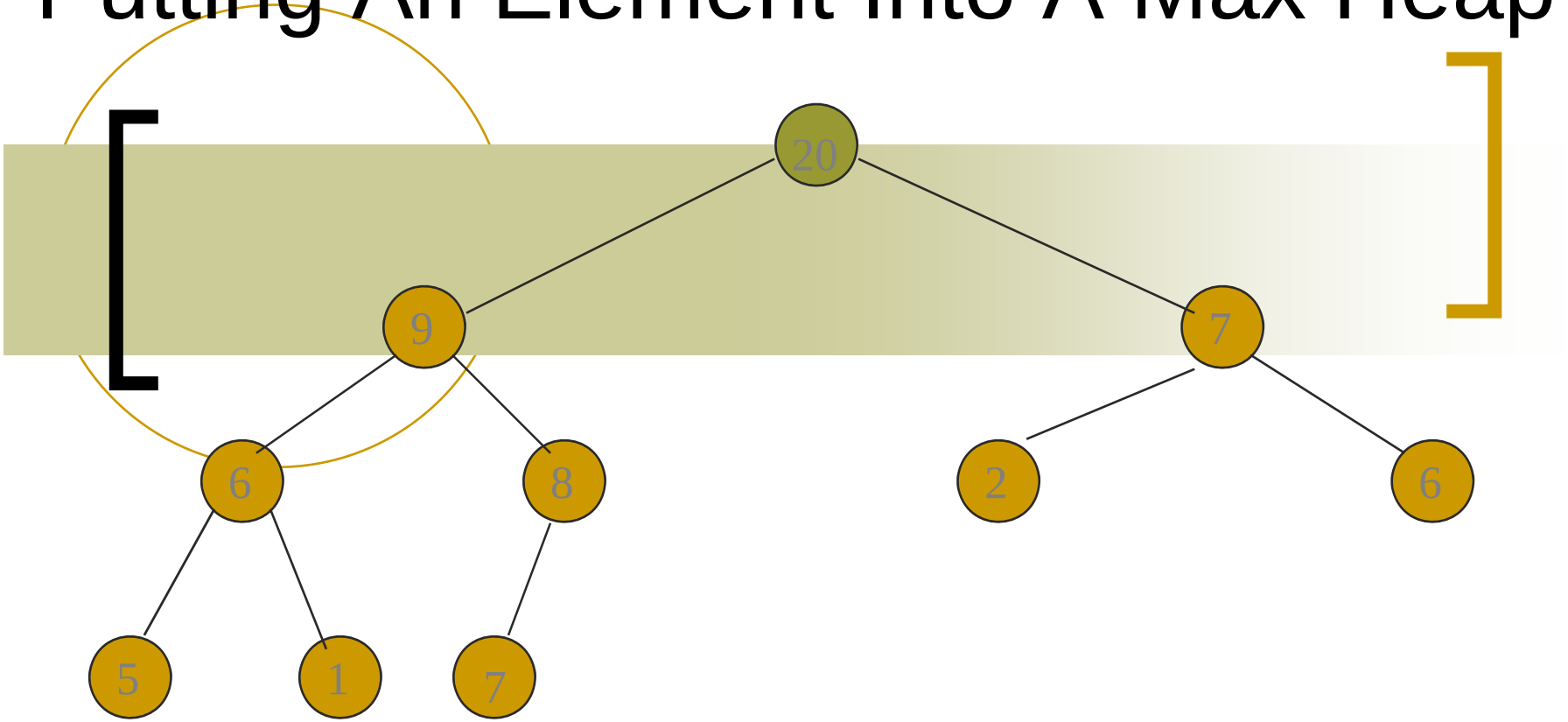
20 گره جدید می باشد

Putting An Element Into A Max Heap



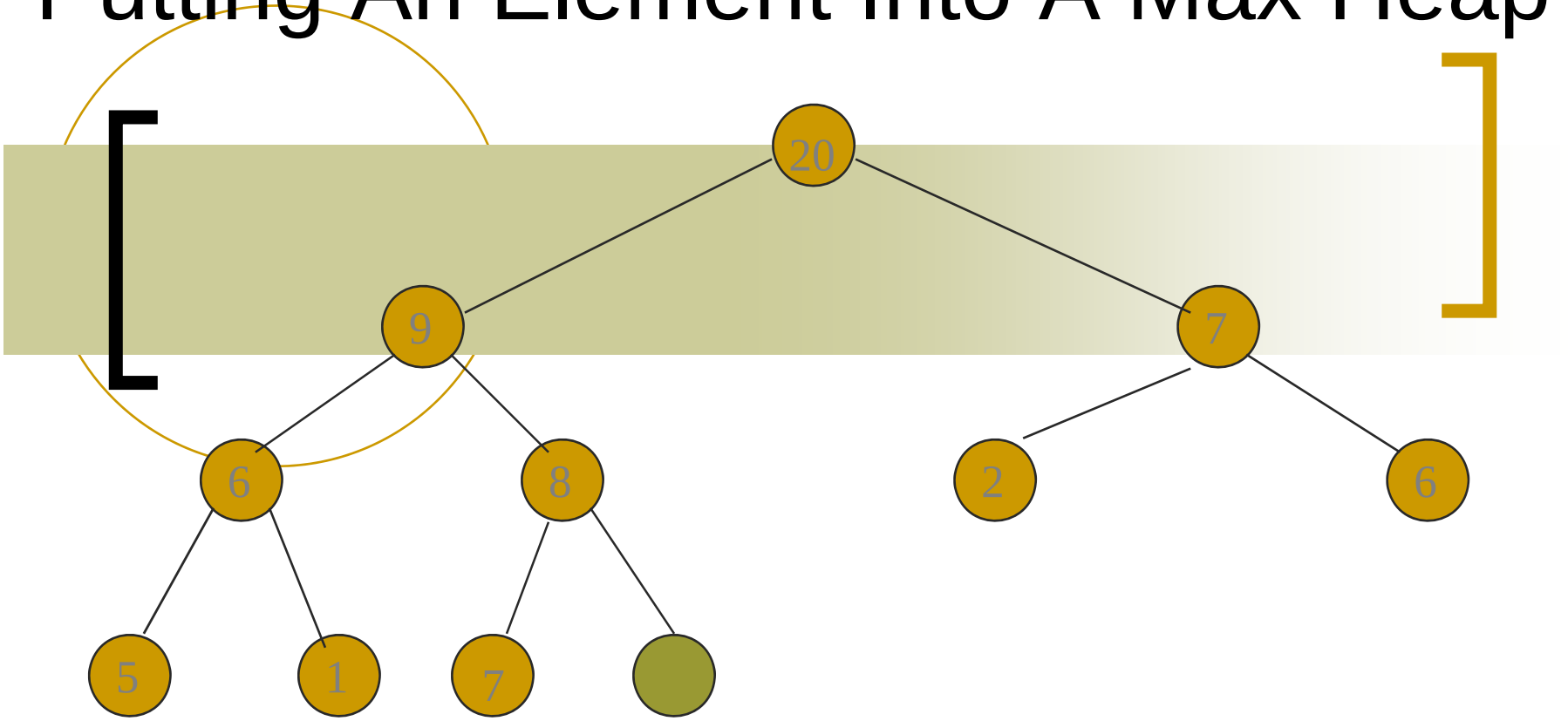
20 گره جدید می باشد

Putting An Element Into A Max Heap



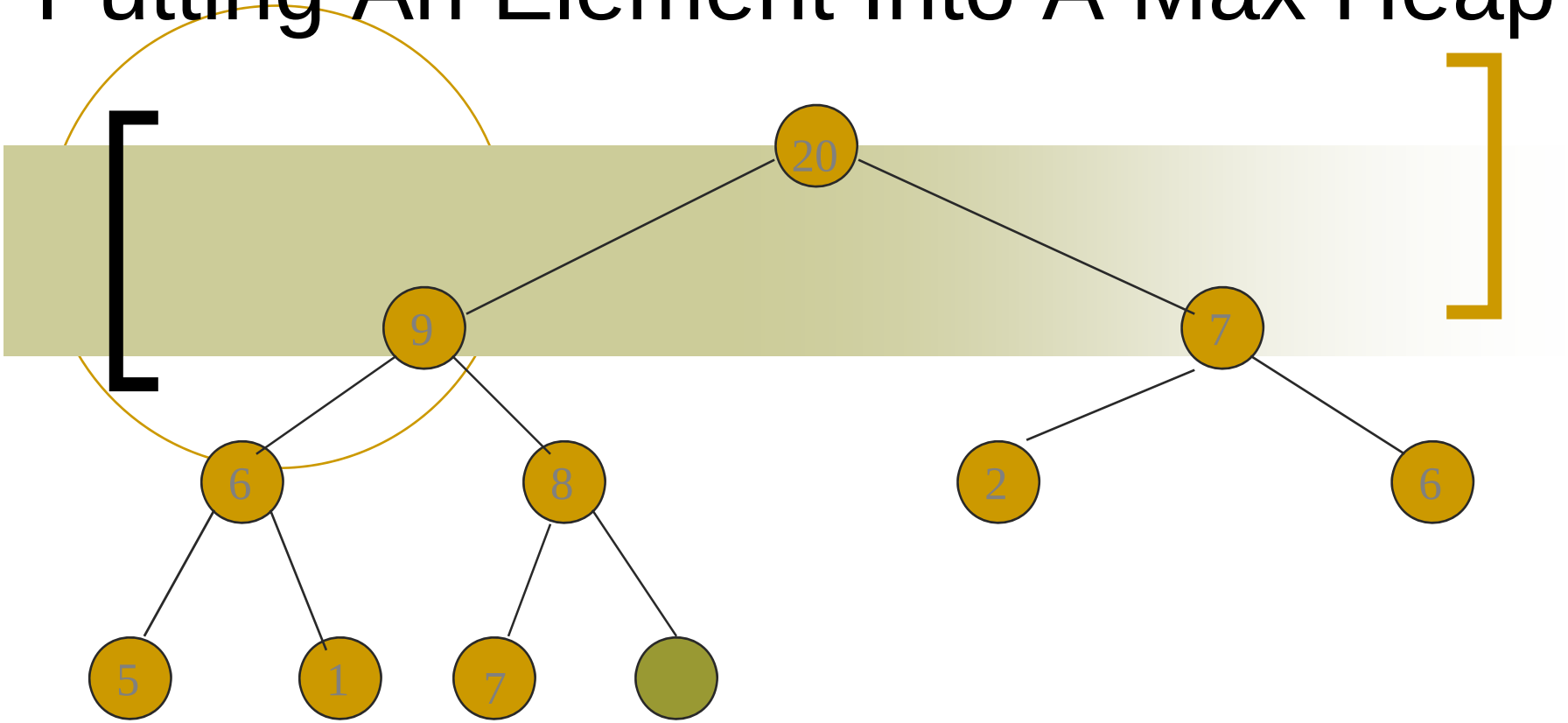
20 گره جدید می باشد

Putting An Element Into A Max Heap



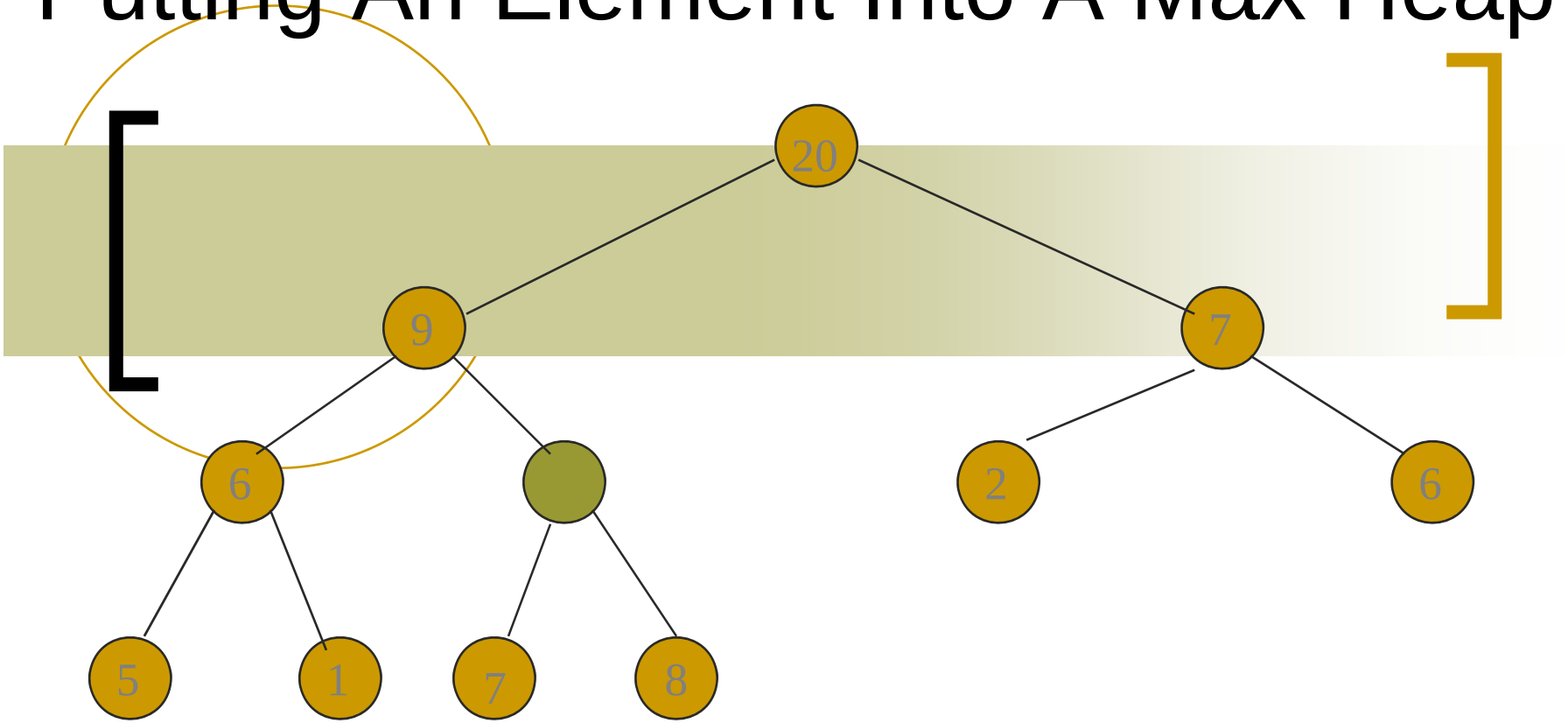
درخت کامل با 11 گره

Putting An Element Into A Max Heap



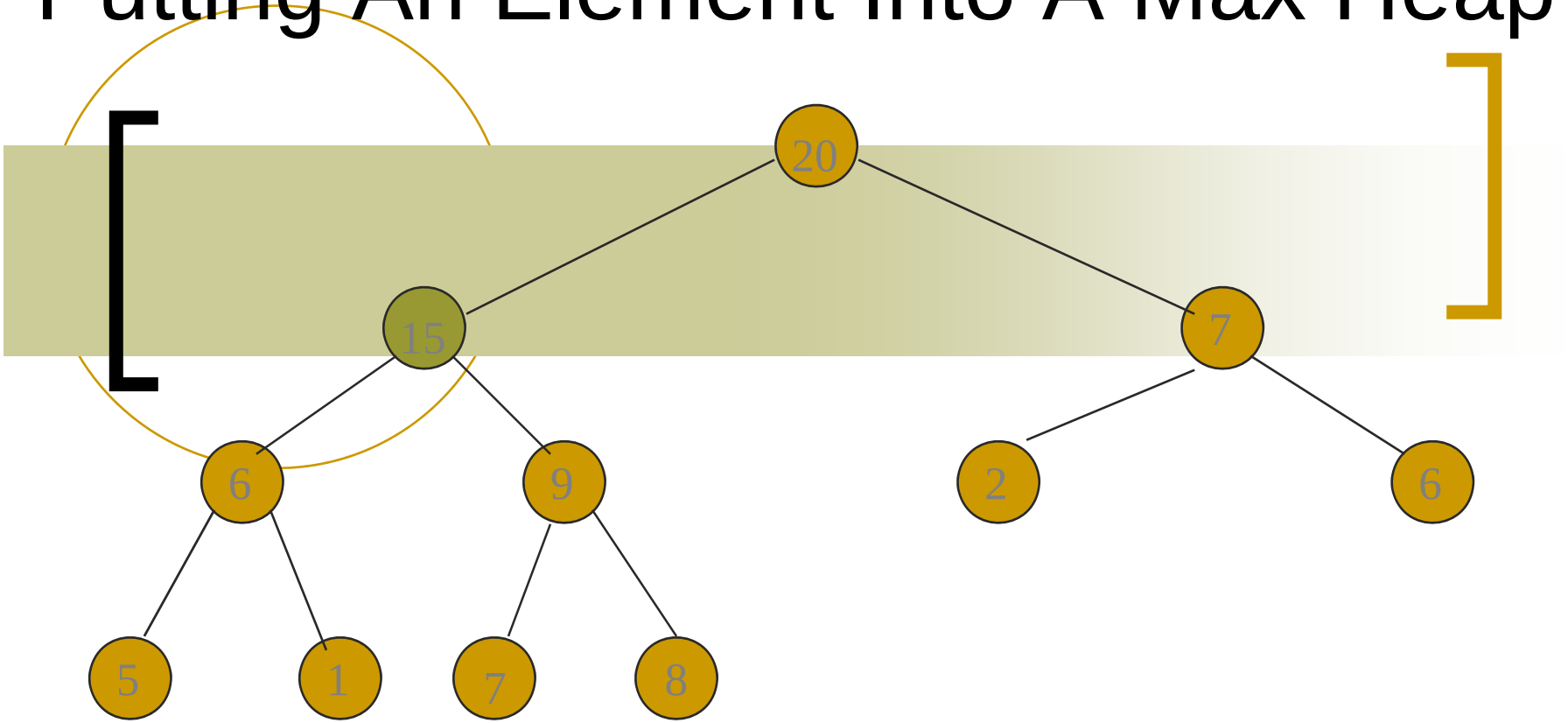
15 گره جدید می باشد

Putting An Element Into A Max Heap



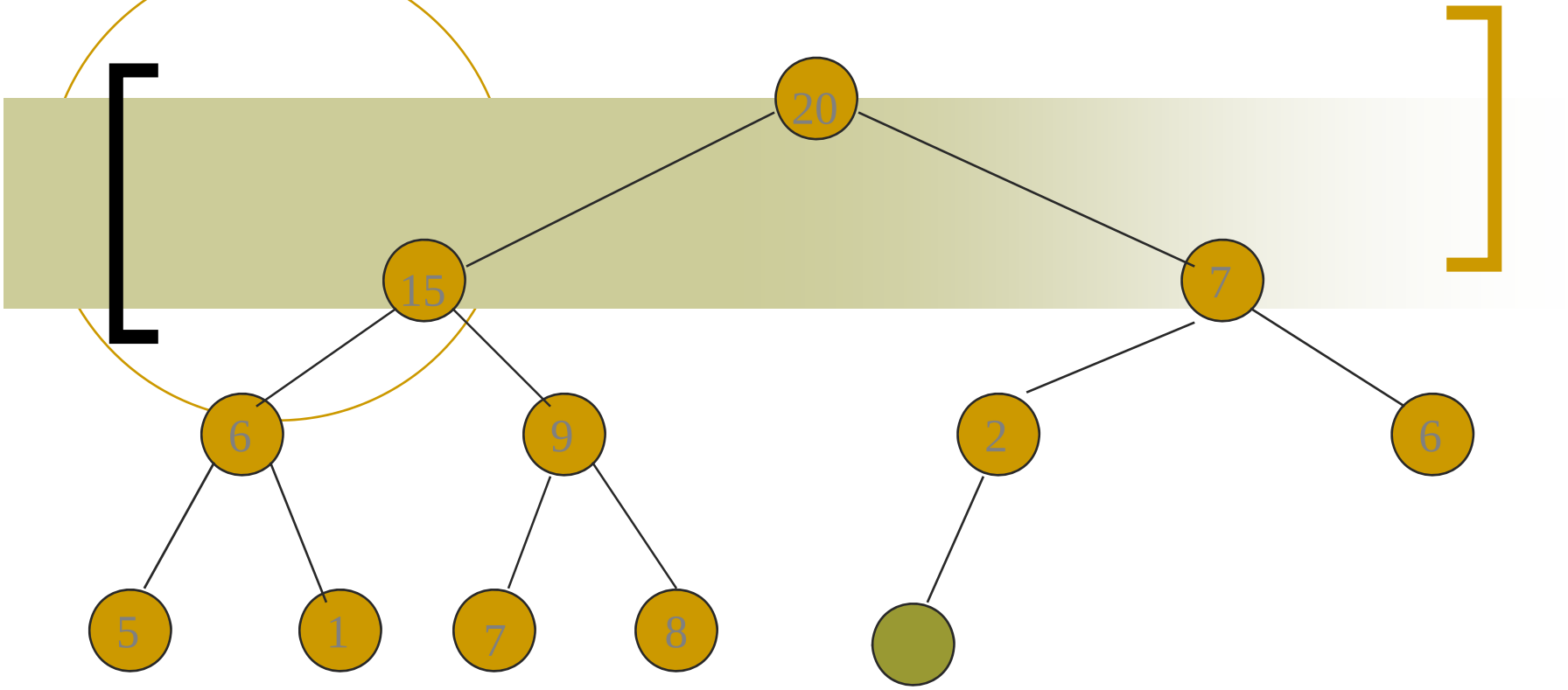
15 گره جدید می باشد

Putting An Element Into A Max Heap



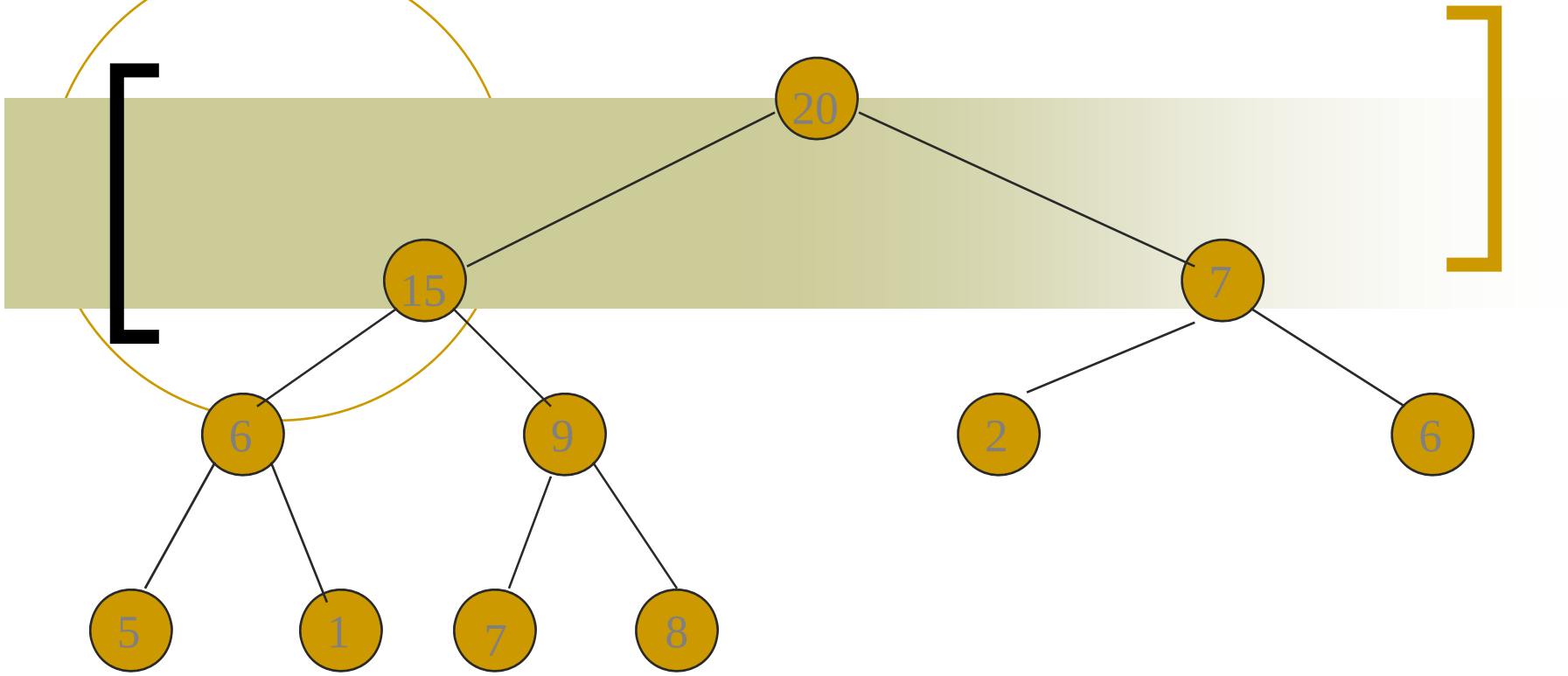
15 گره جدید می باشد

Complexity Of Put



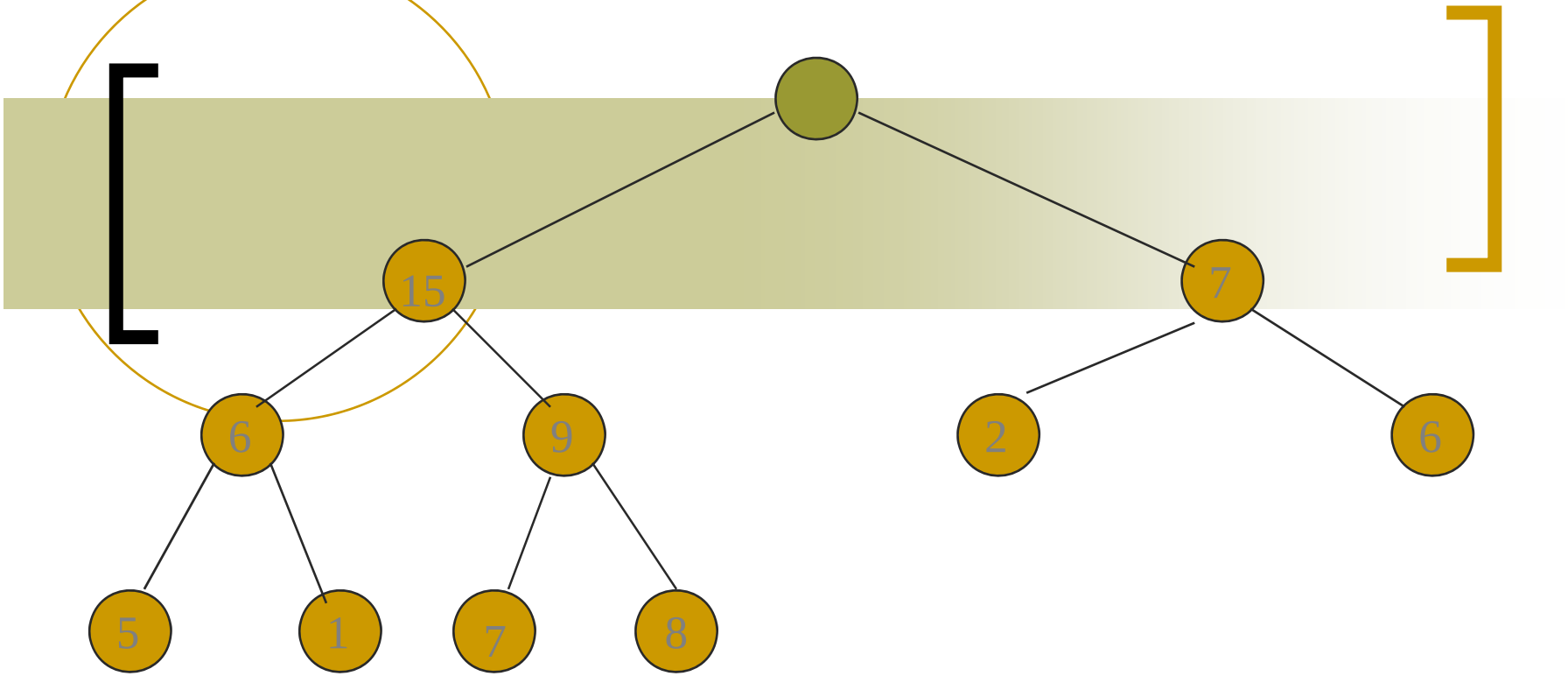
پیچیدگی آن O(log n) زمانیکه سایز heapمقدار n باشد

Removing The Max Element



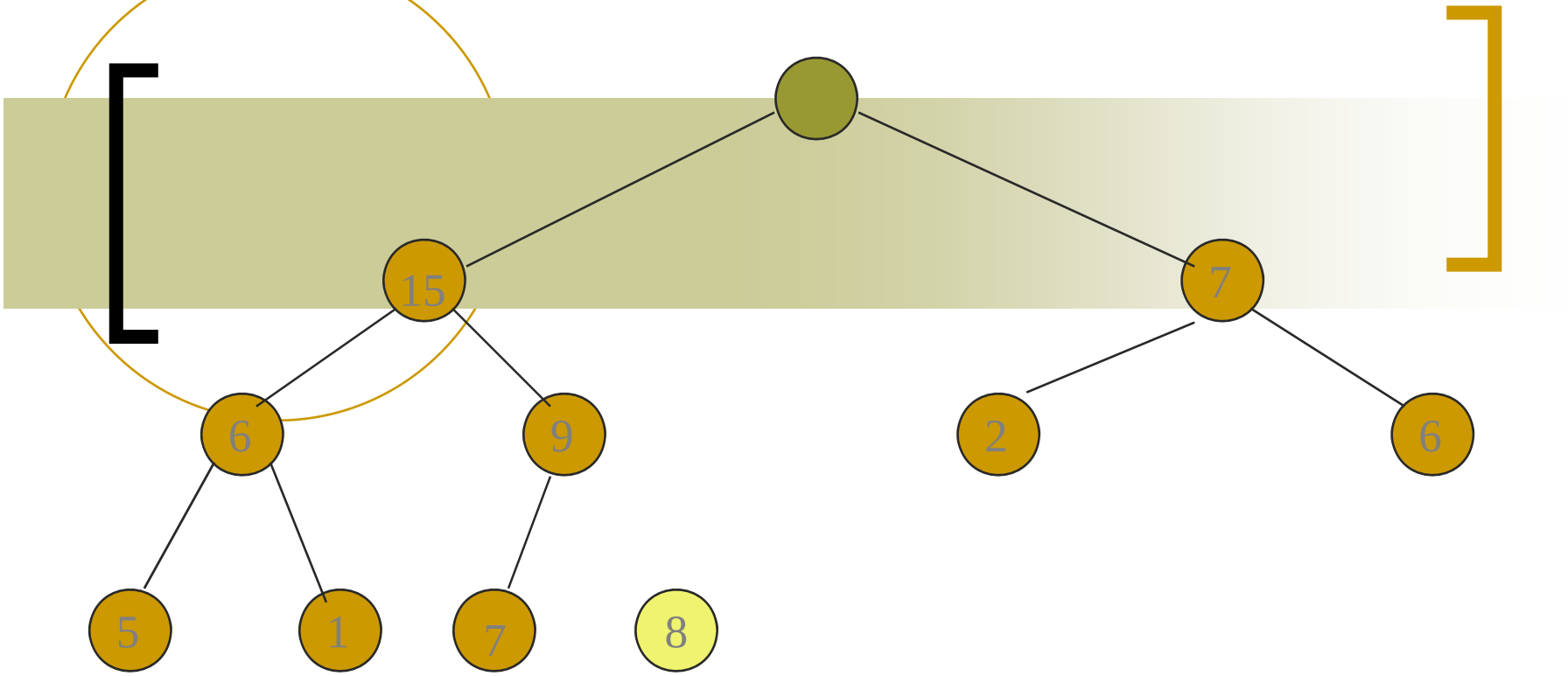
بزرگترین مقدلر در ریشه قرار دارد.

Removing The Max Element



پس از آنکه بزرگترین مقدار حذف شد

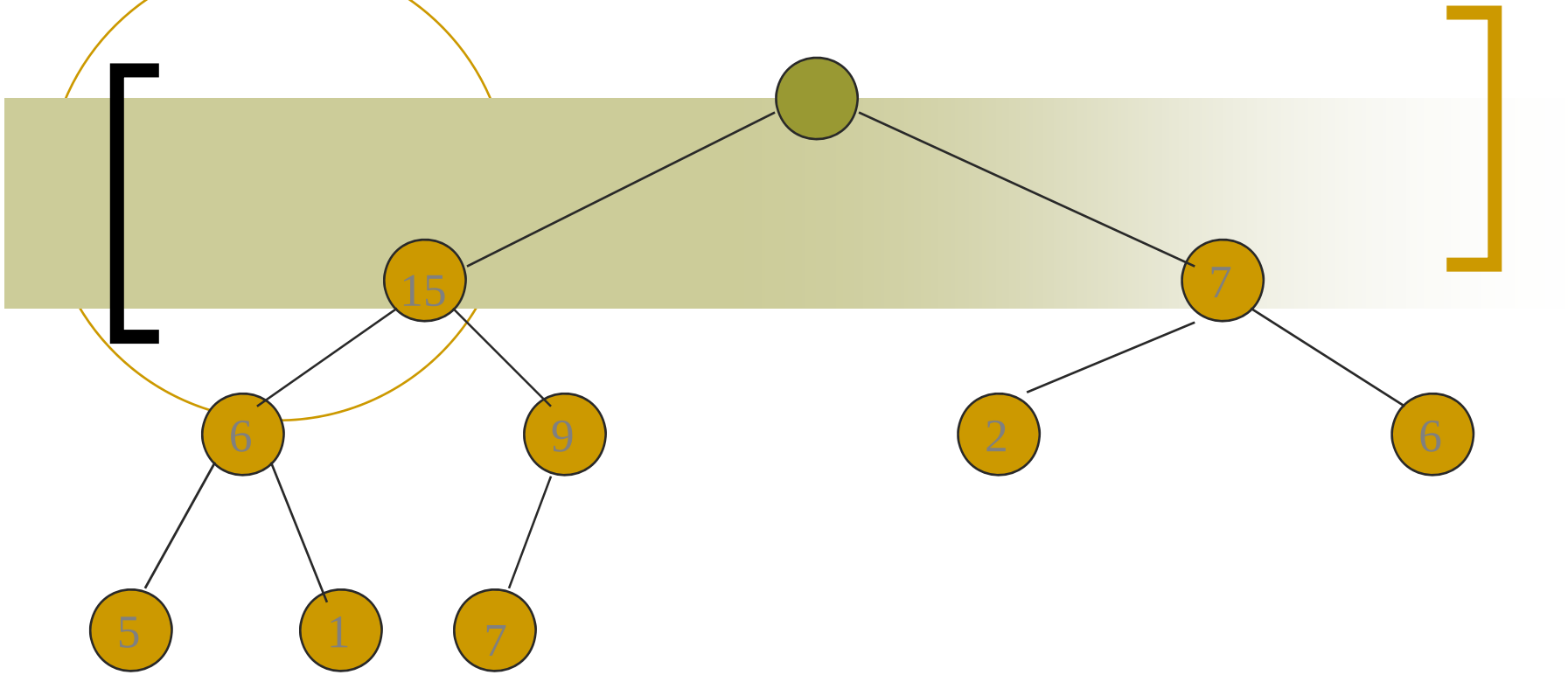
Removing The Max Element



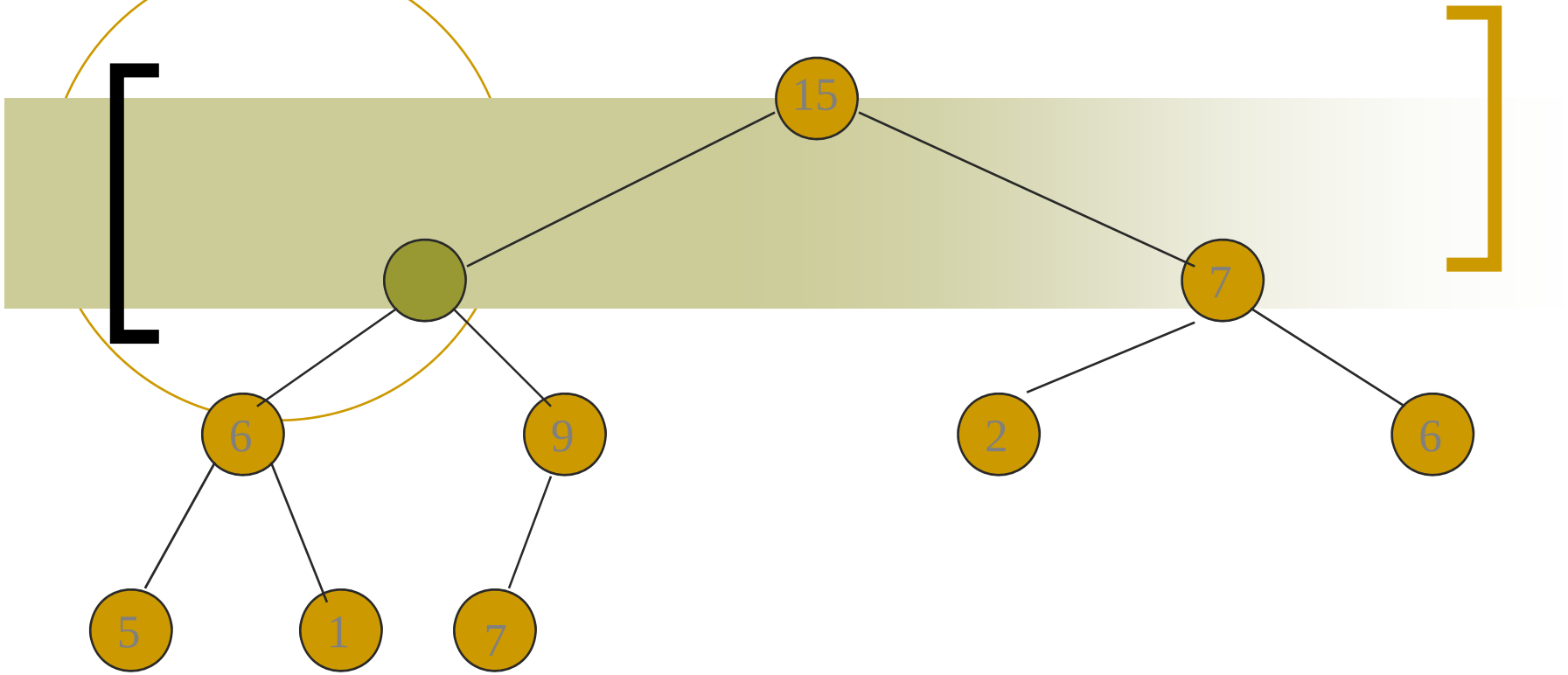
در heapتعداد 10عنصر قرار می گیرد.

8 را به heapاضافه کنید.

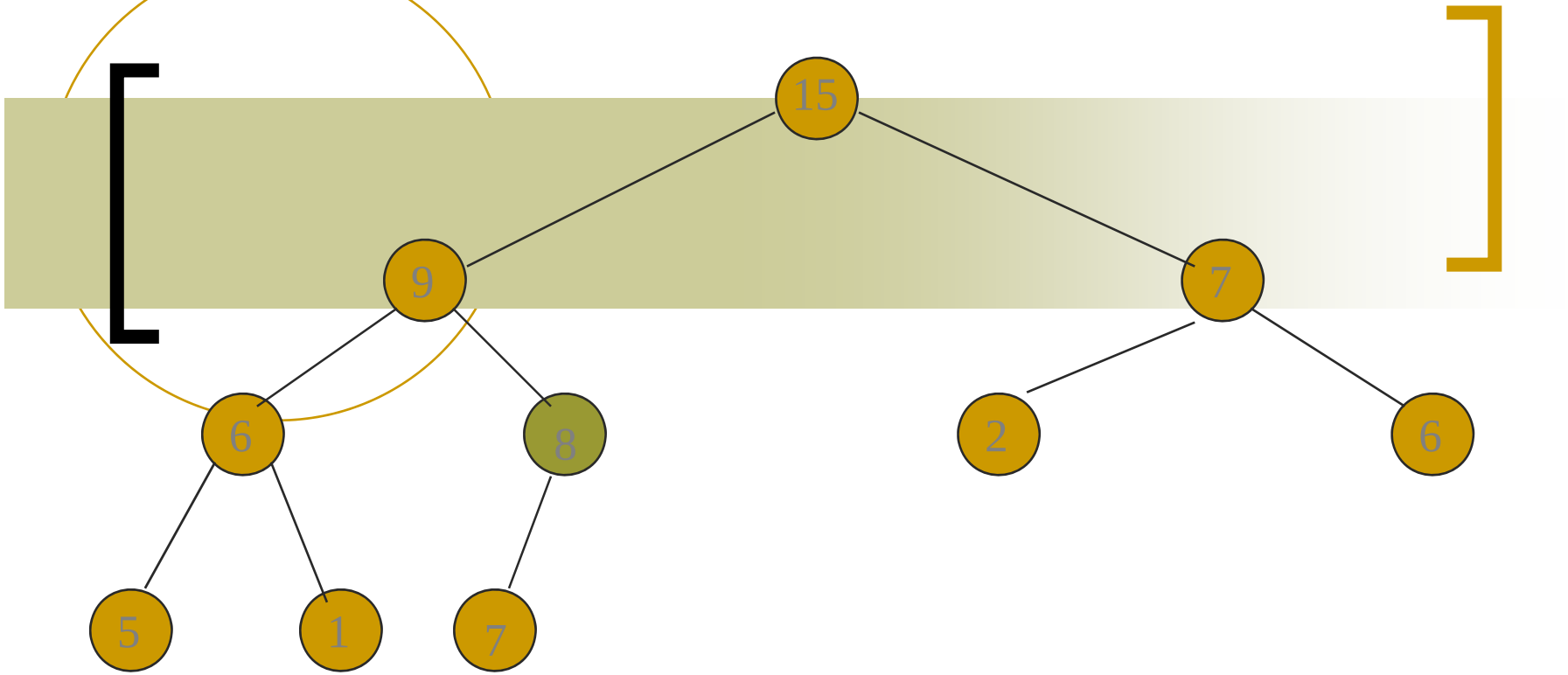
Removing The Max Element



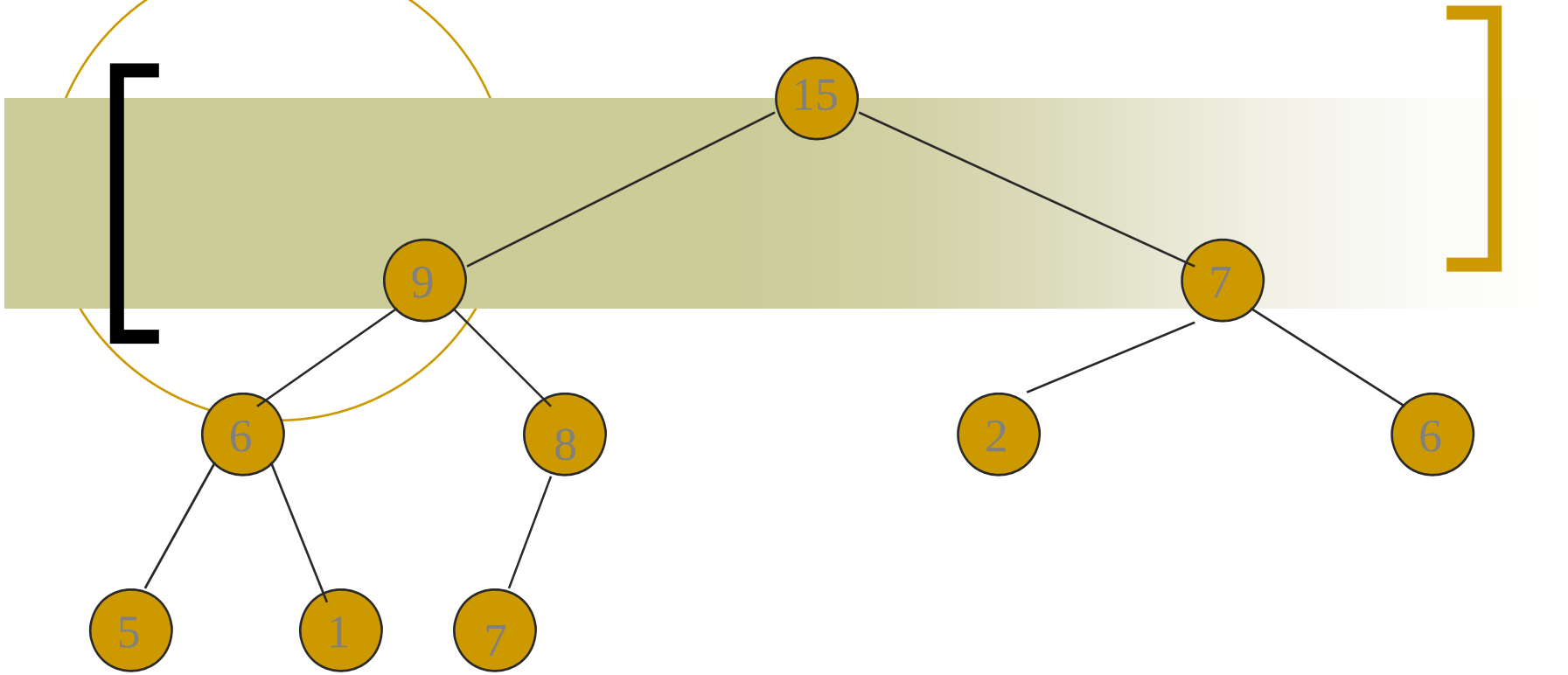
Removing The Max Element



Removing The Max Element

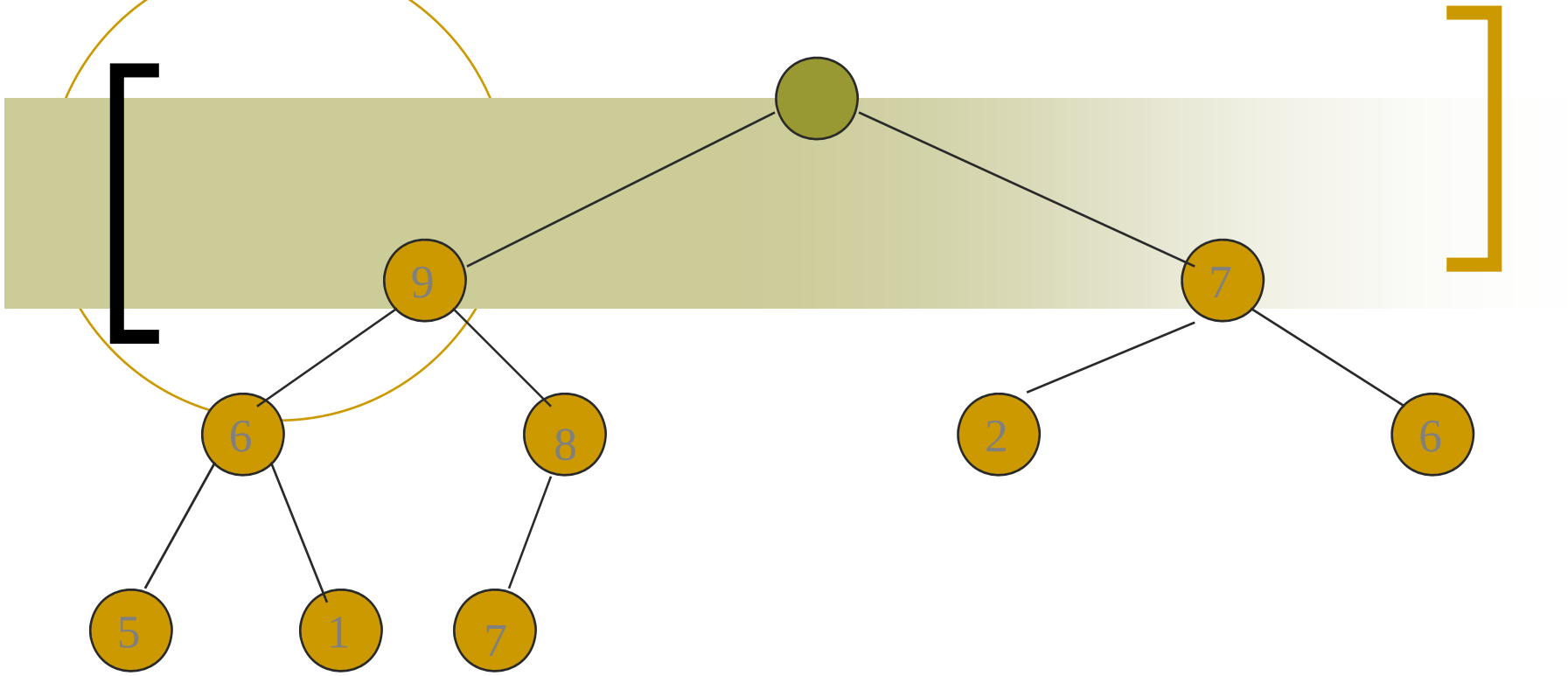


Removing The Max Element



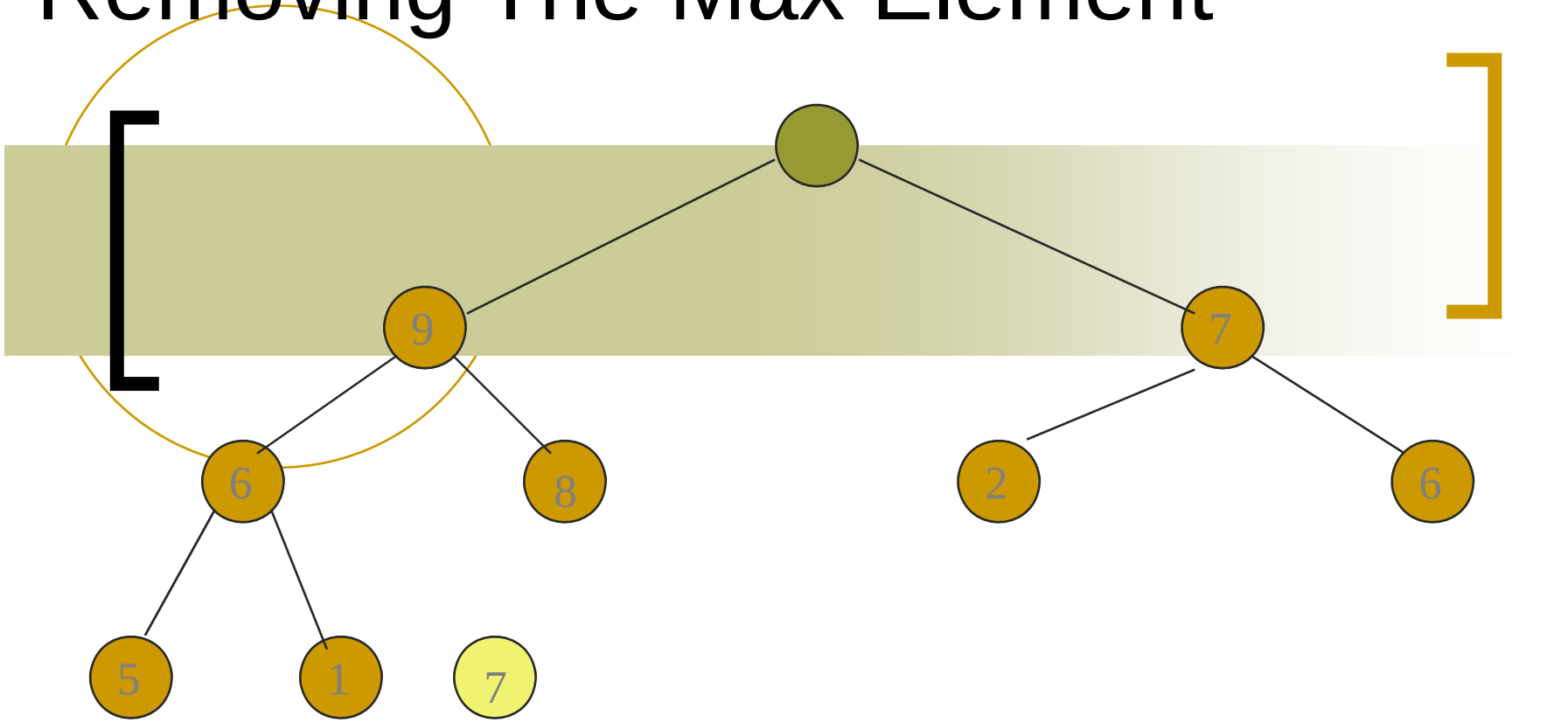
15 بزرگترین عنصر قرار می گیرد.

Removing The Max Element



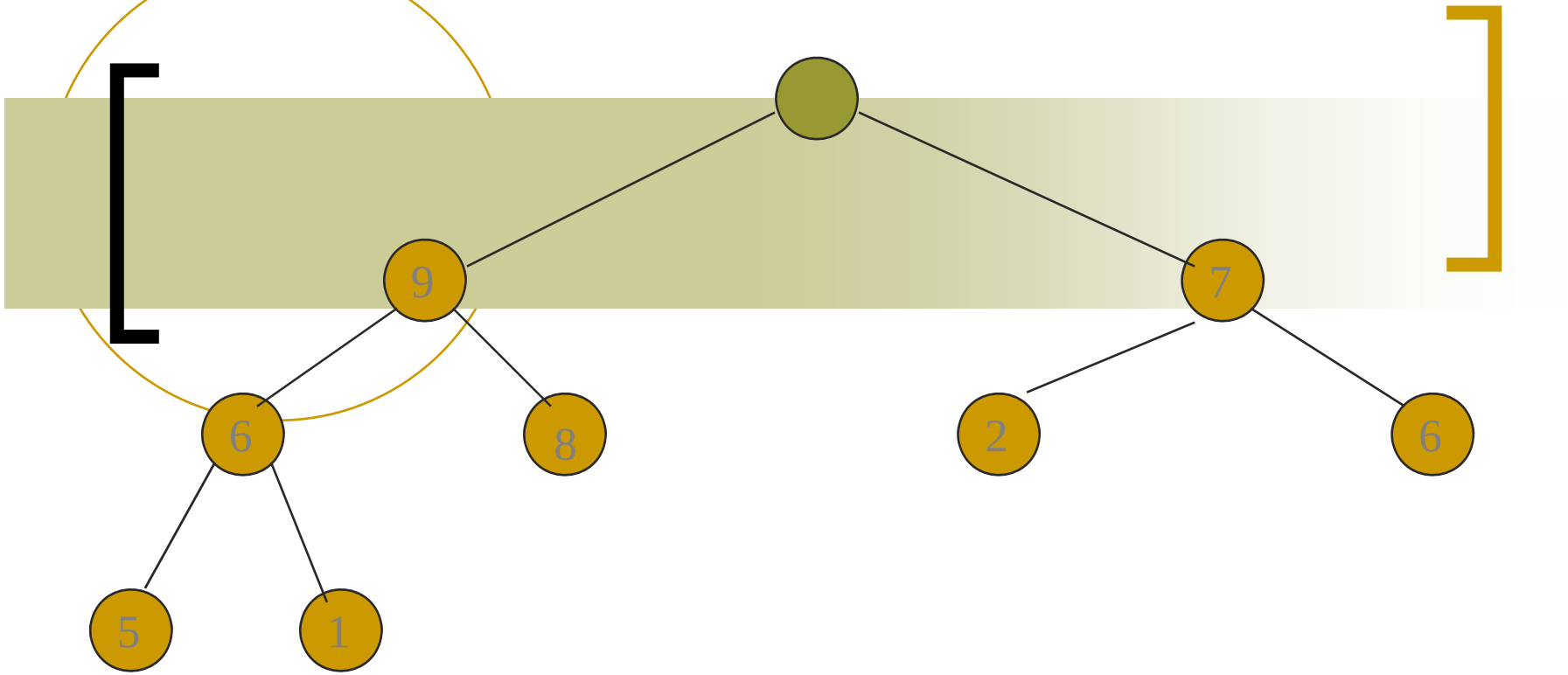
پس از آنکه بزرگترین مقدار حذف شد

Removing The Max Element

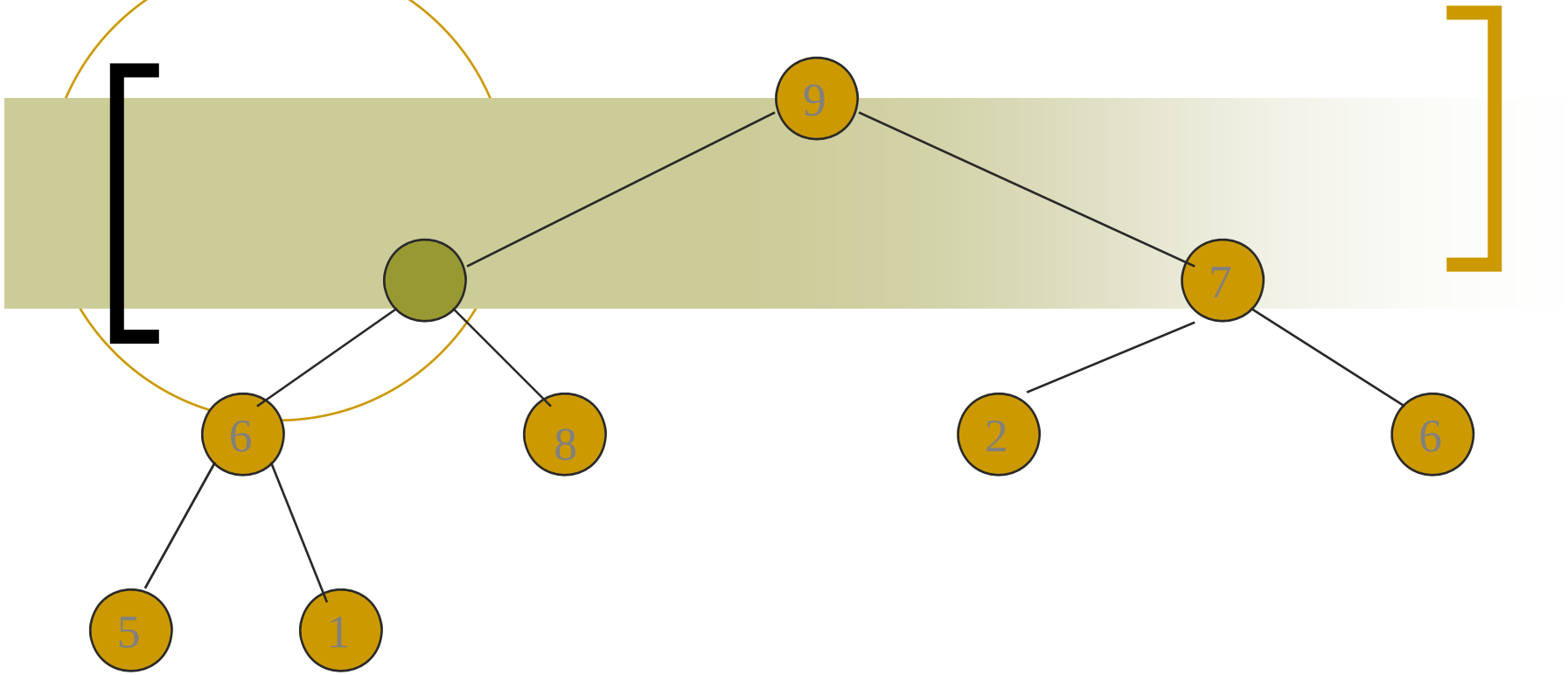


اکنون در heap تعداد9 عنصر قرار می گیرد.

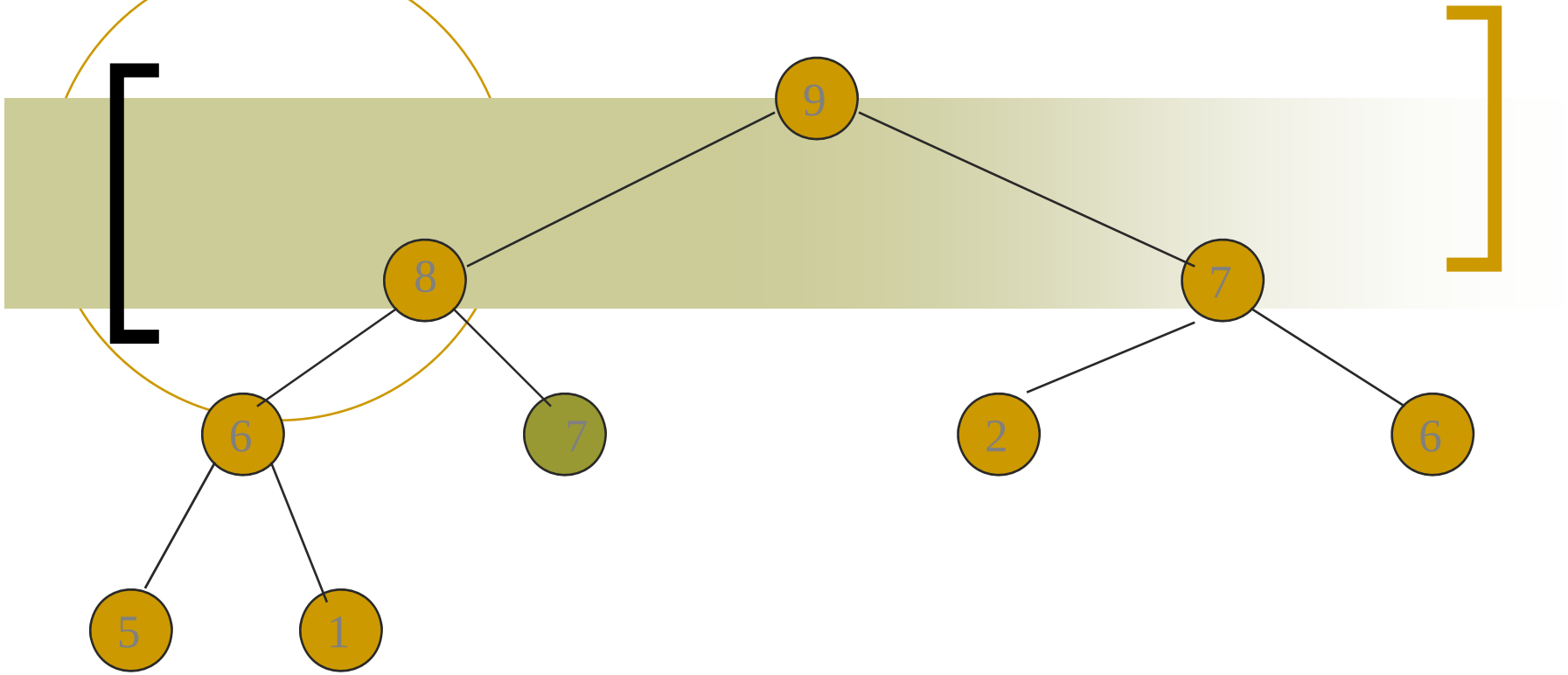
Removing The Max Element



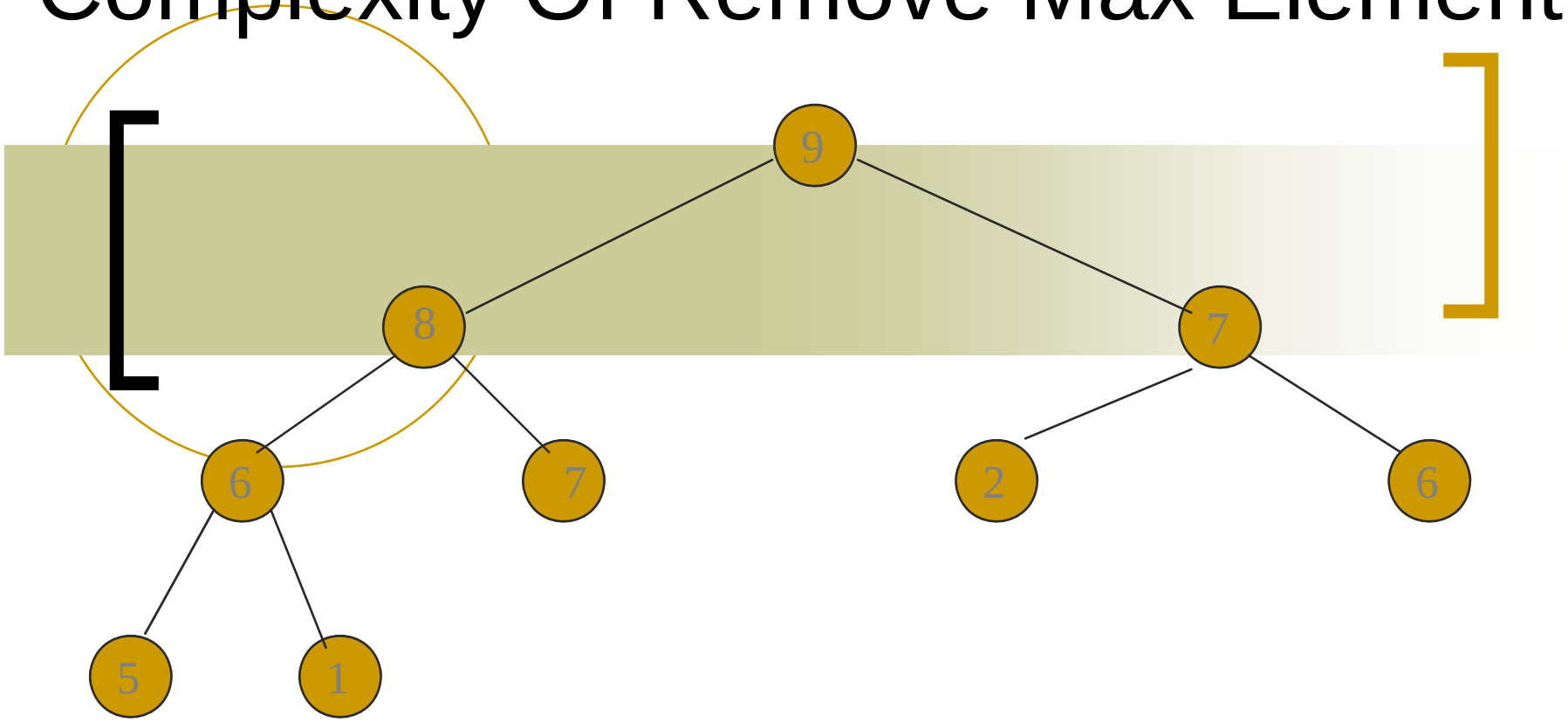
Removing The Max Element



Removing The Max Element

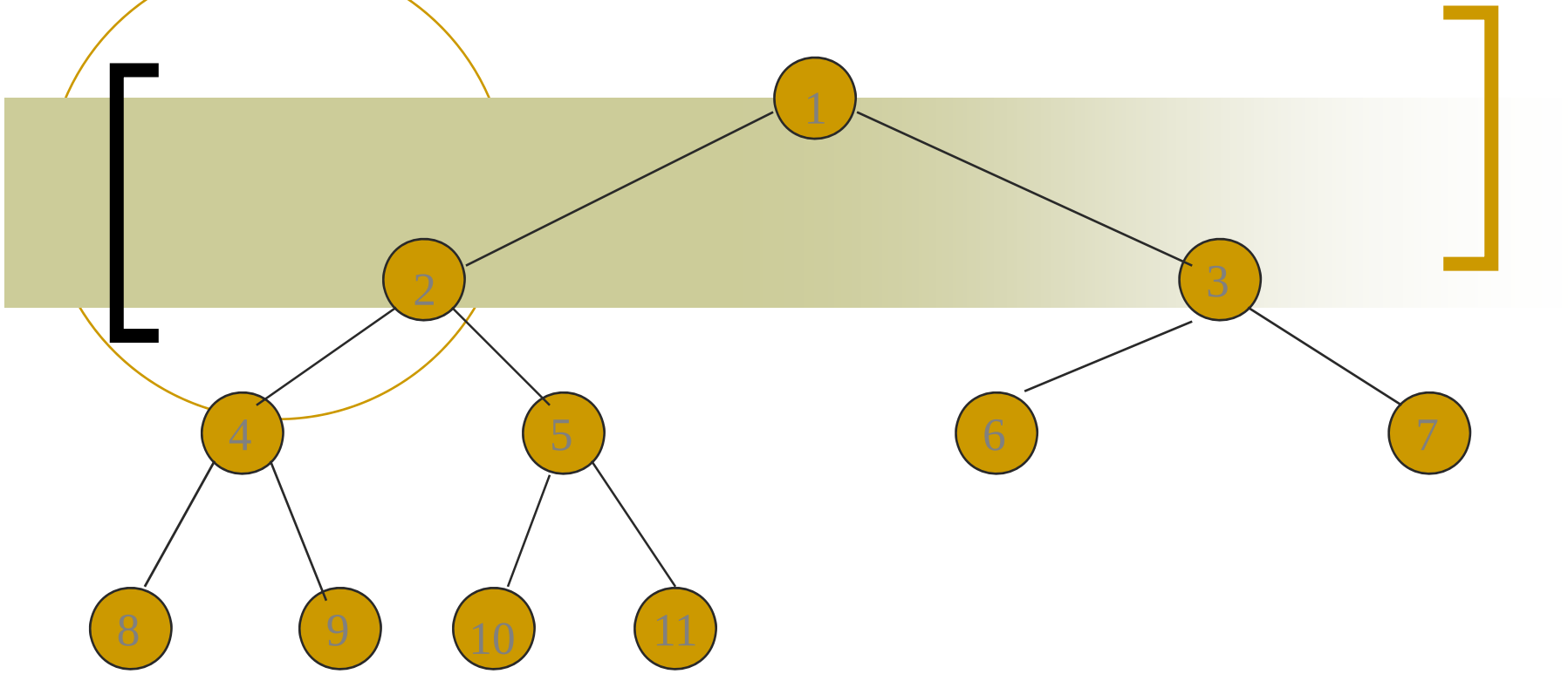


Complexity Of Remove Max Element



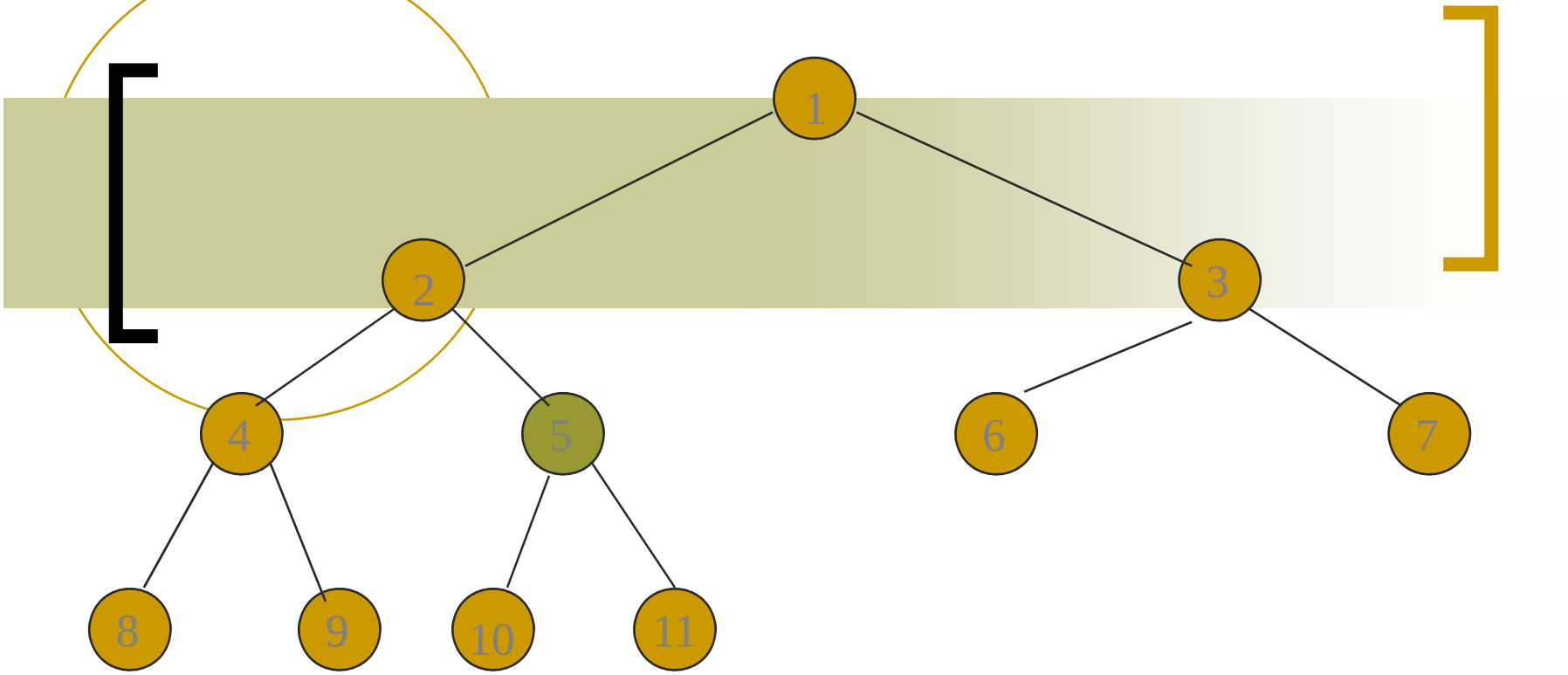
پیچیدگی =O(log n)

Initializing A Max Heap



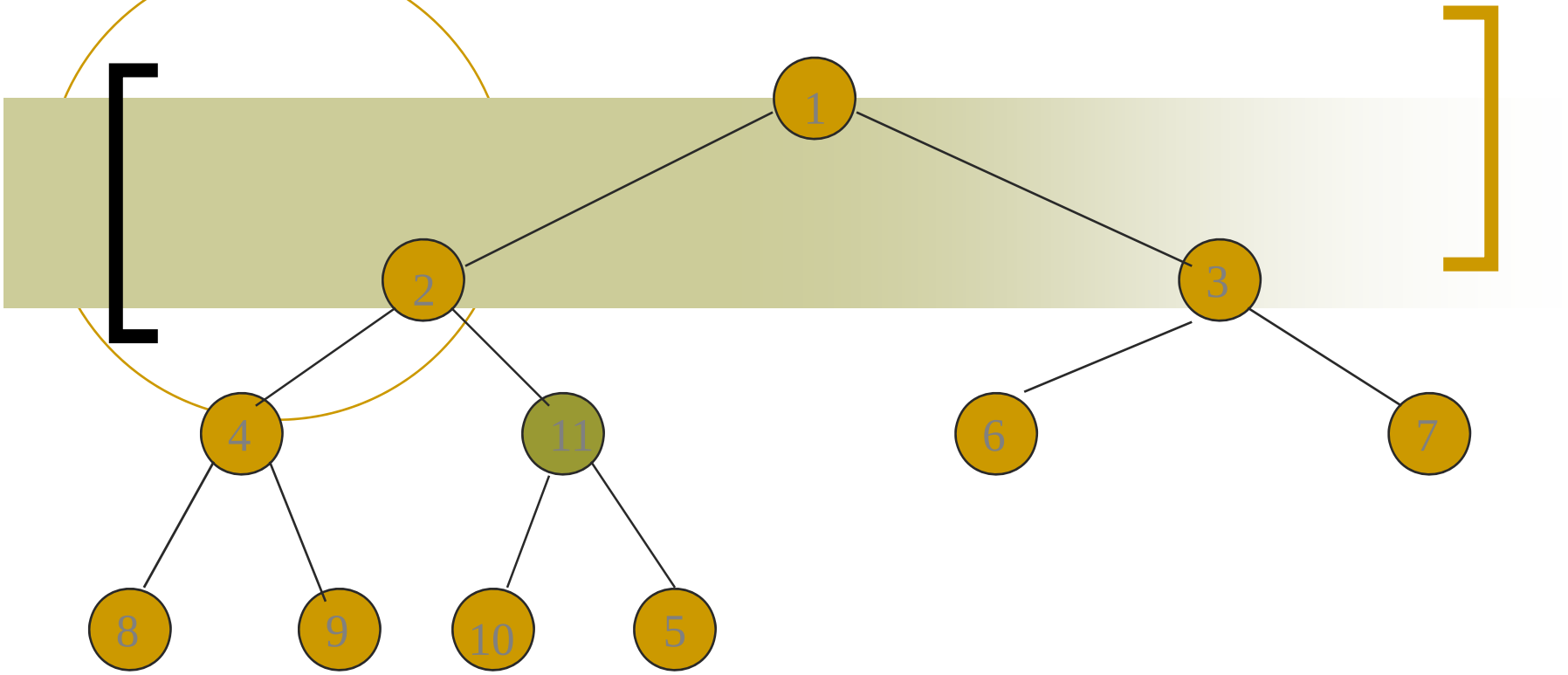
input array = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Initializing A Max Heap



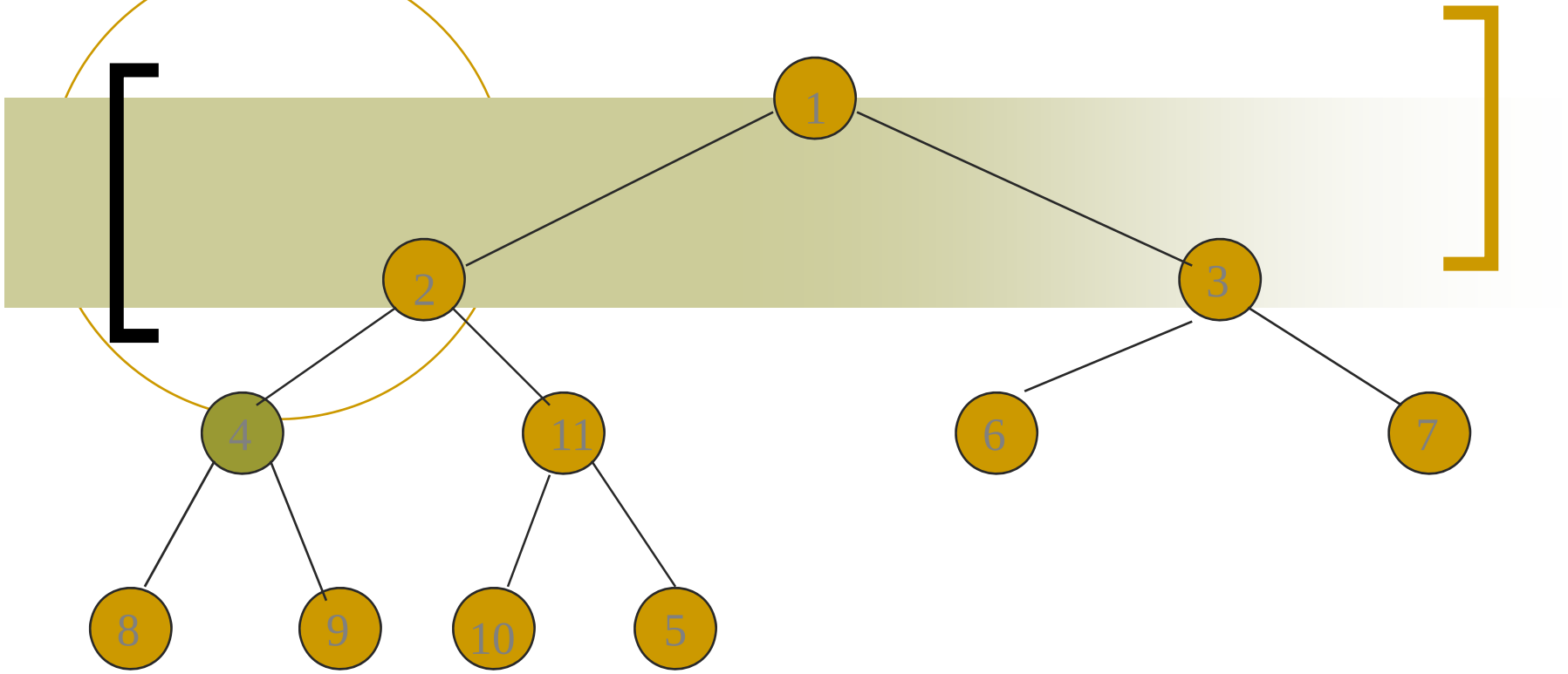
ابتدا موقعیت n/2 را در آرایه فوق محاسبه کرده و سپس از آن گره در درخت شروع کنید

Initializing A Max Heap

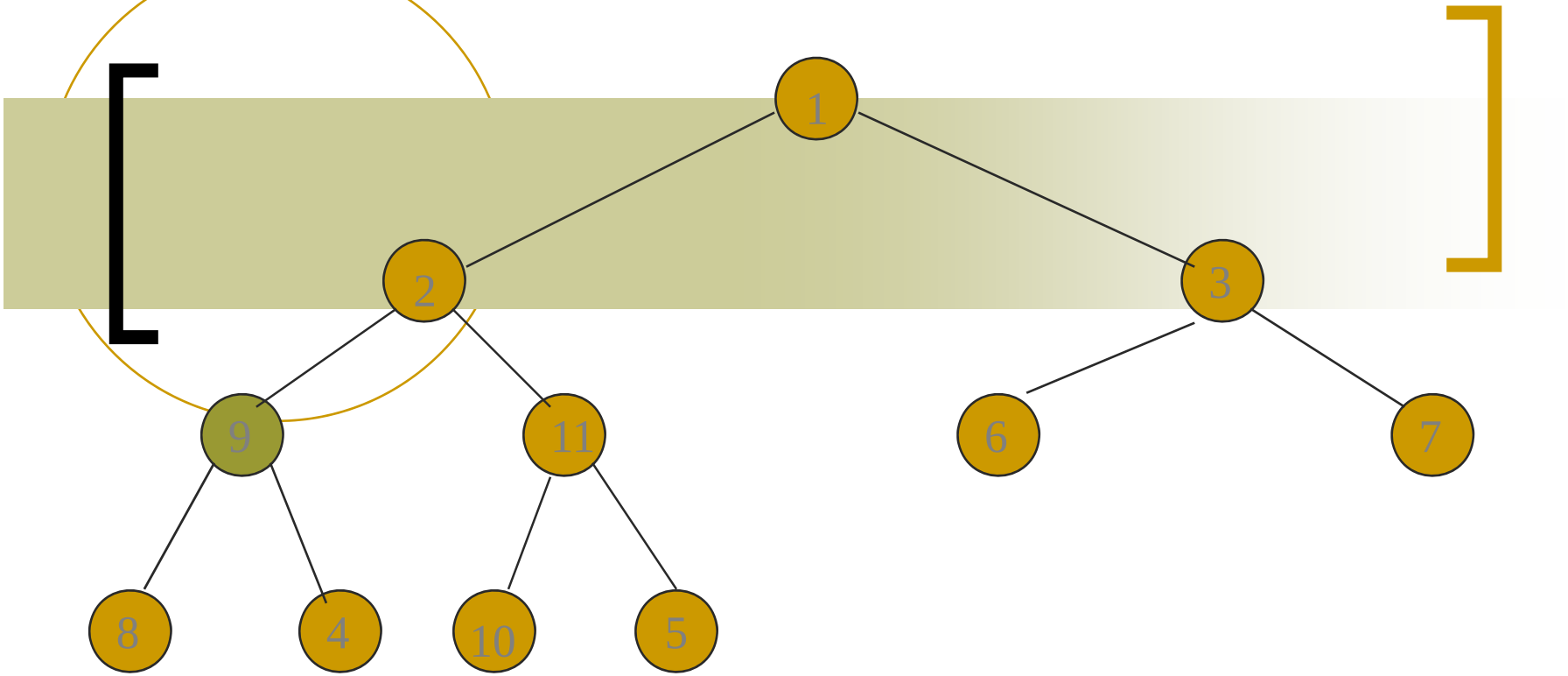


Max heap بودن را در درخت چک کنید.در درخت max heapمحتوای ریشه از کلیه عناصر آن بزرگتر می باشد.

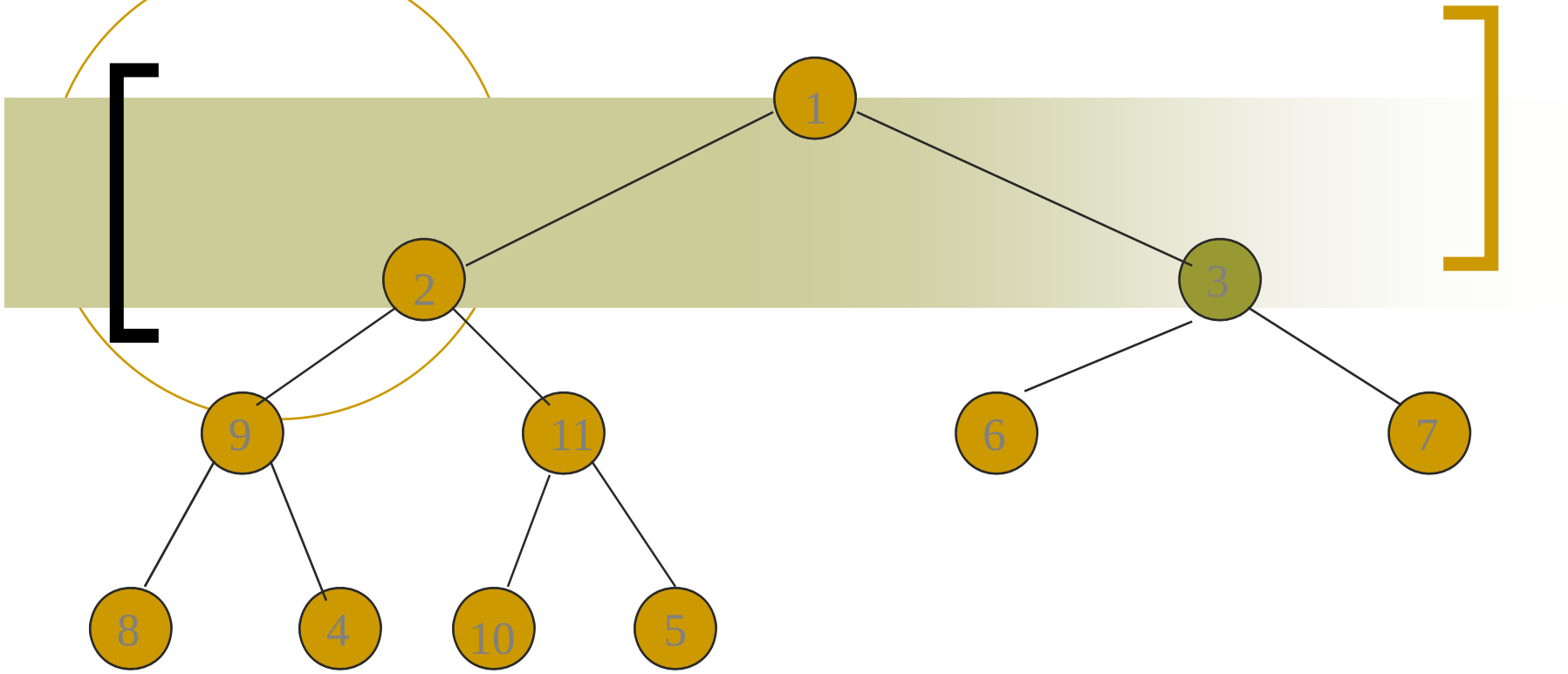
Initializing A Max Heap



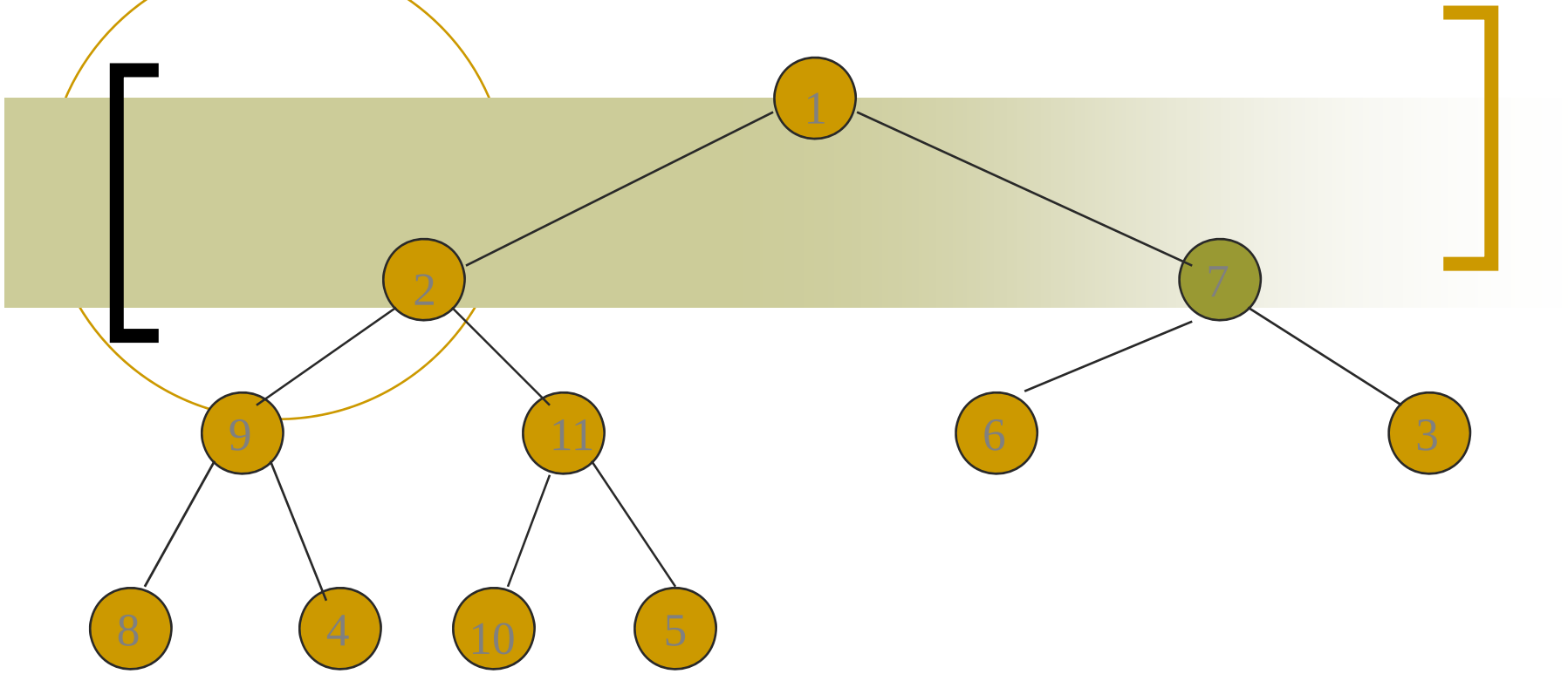
Initializing A Max Heap



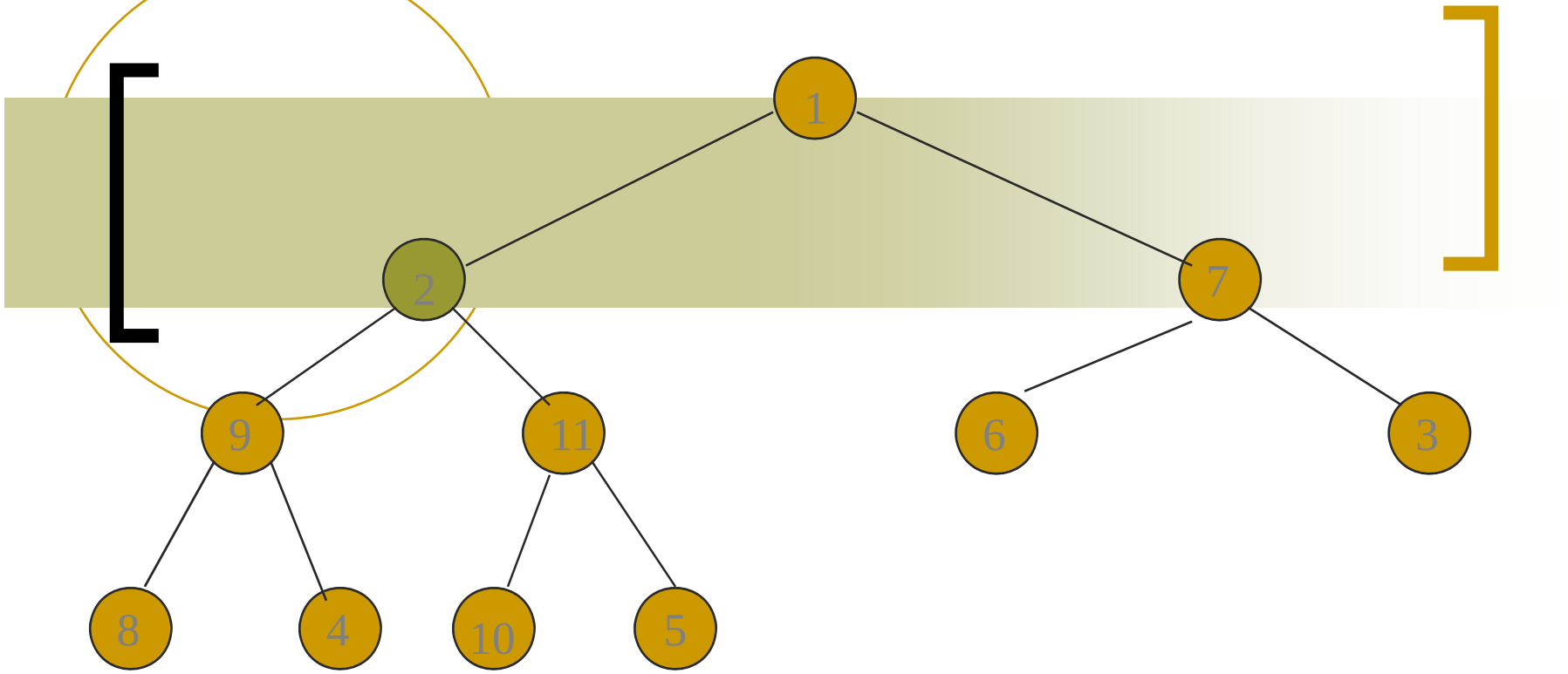
Initializing A Max Heap



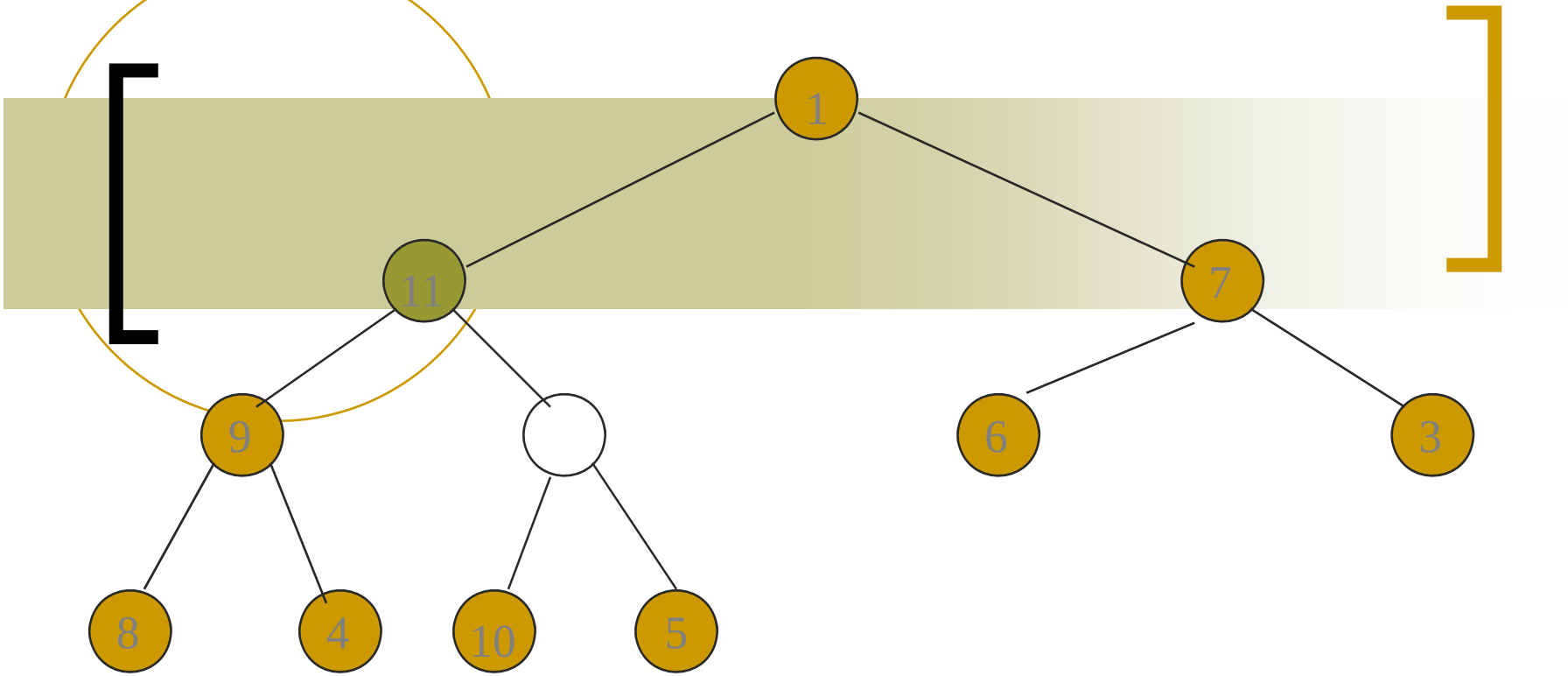
Initializing A Max Heap



Initializing A Max Heap

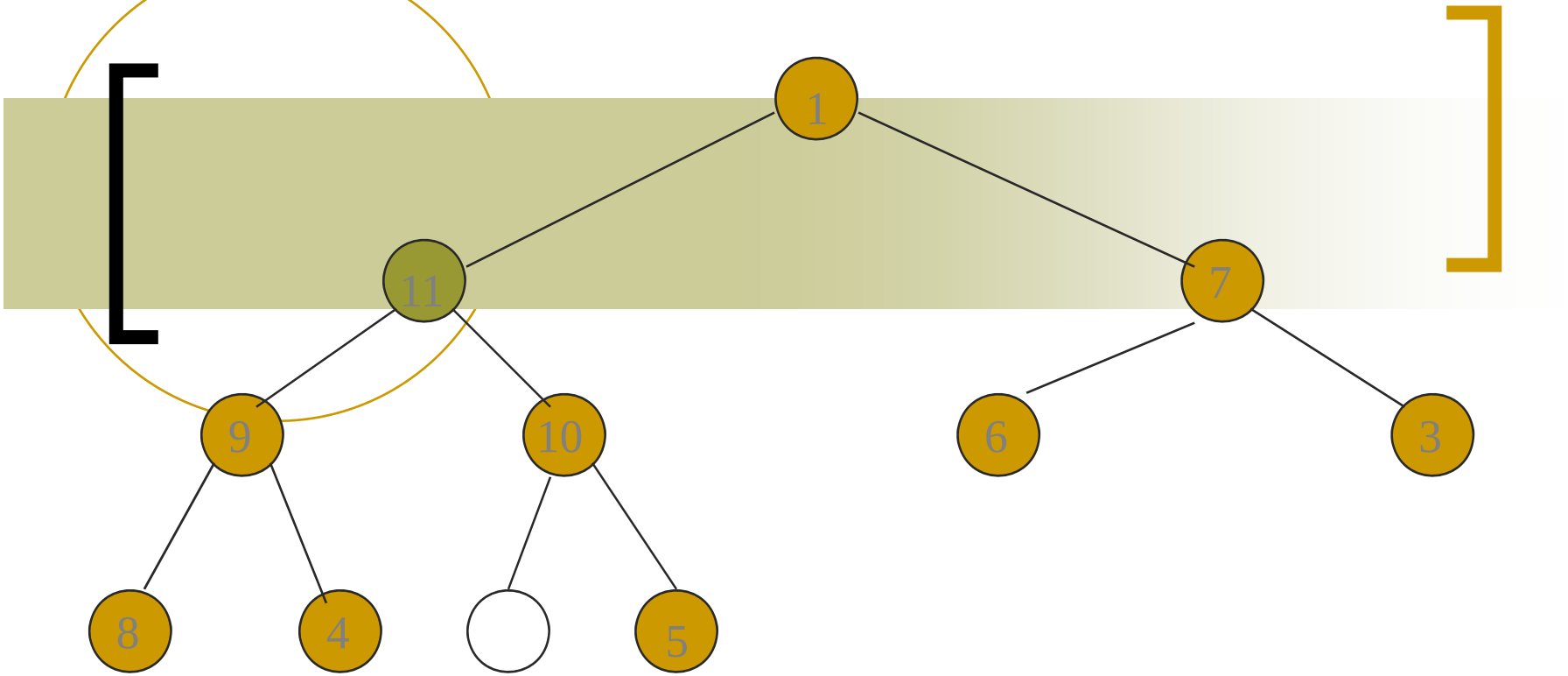


Initializing A Max Heap



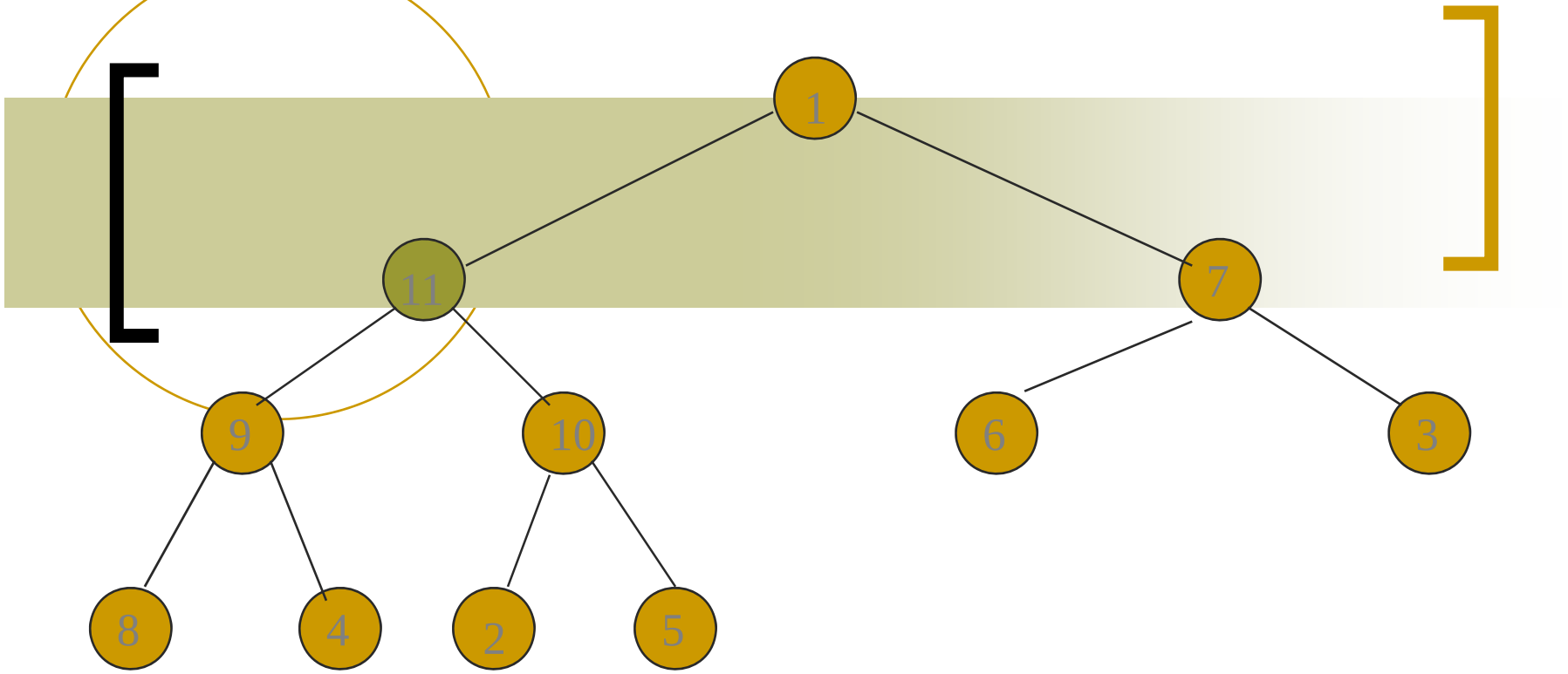
مکان 11 با 2عوض می شود.

Initializing A Max Heap

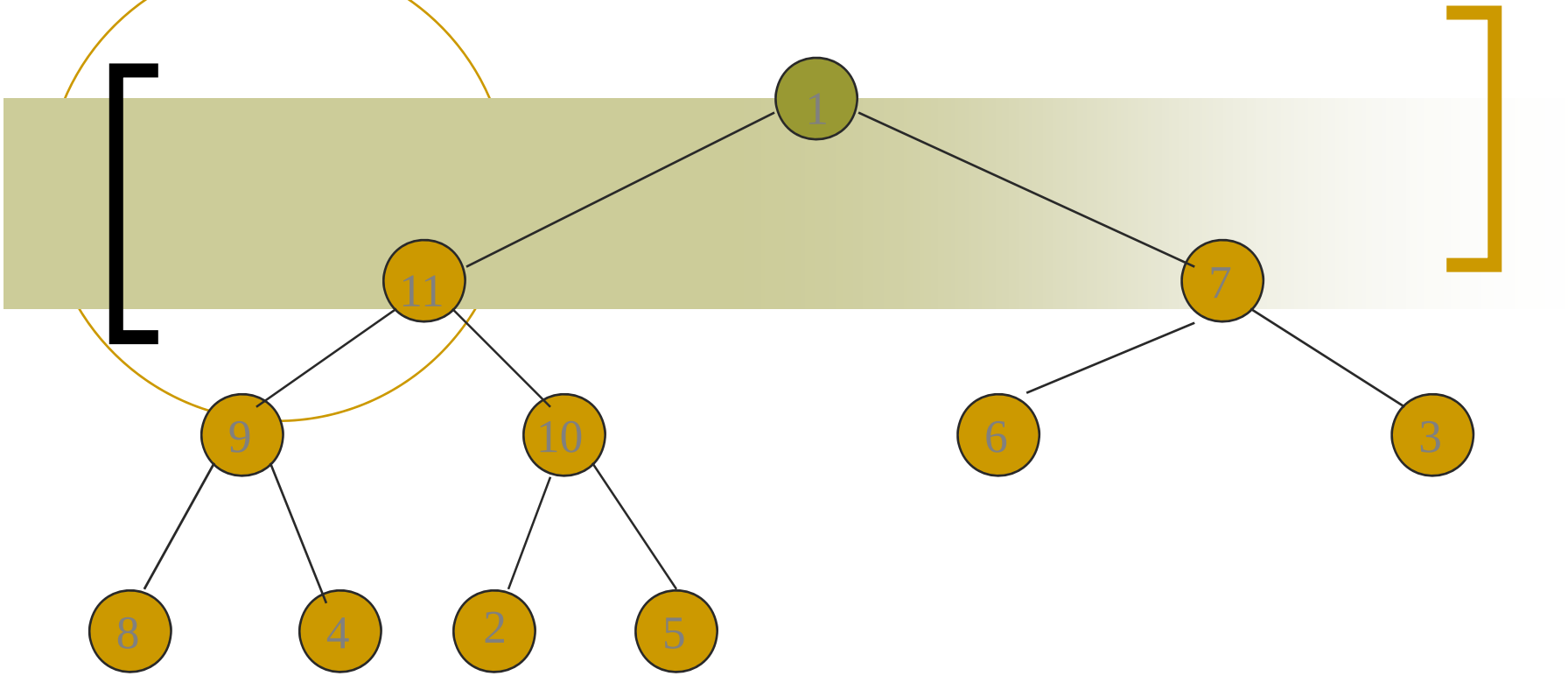


مکان 10 با 2عوض می شود.

Initializing A Max Heap

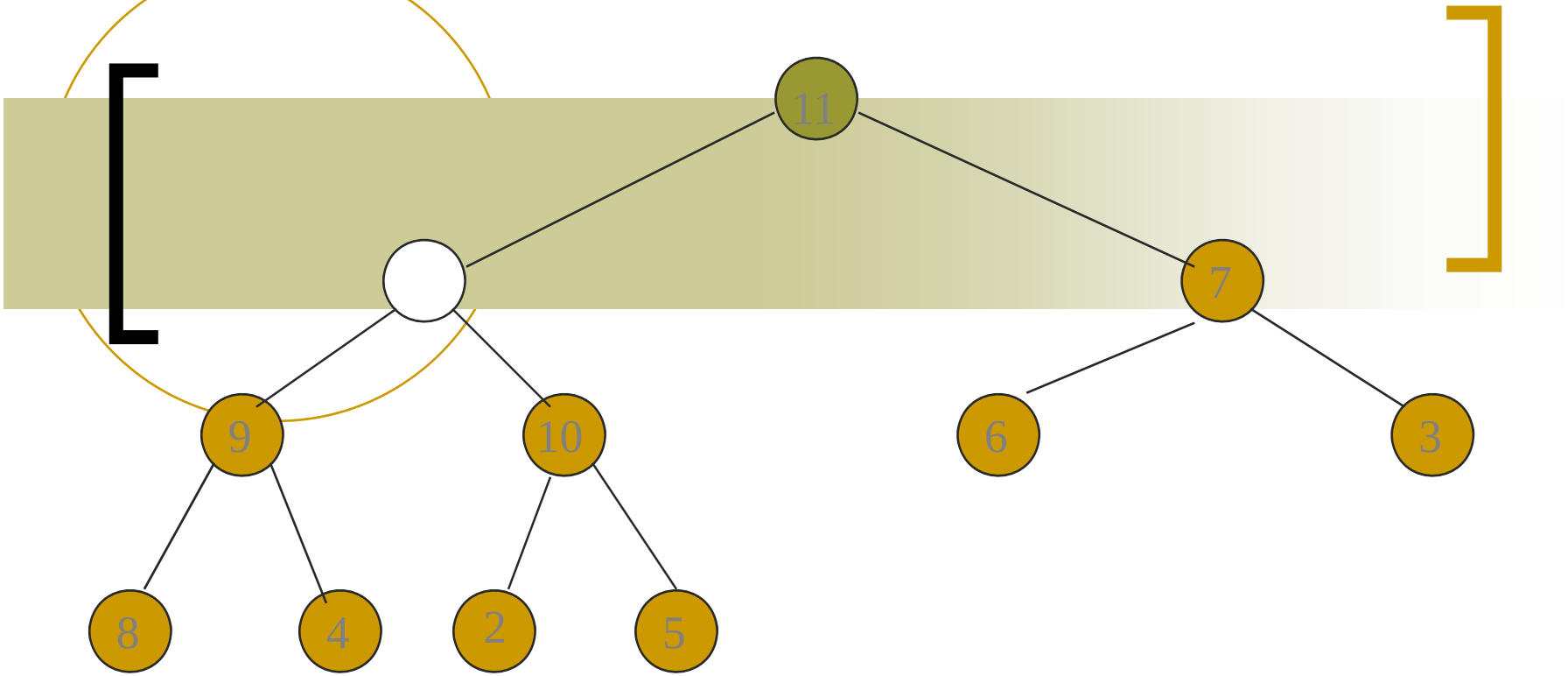


Initializing A Max Heap



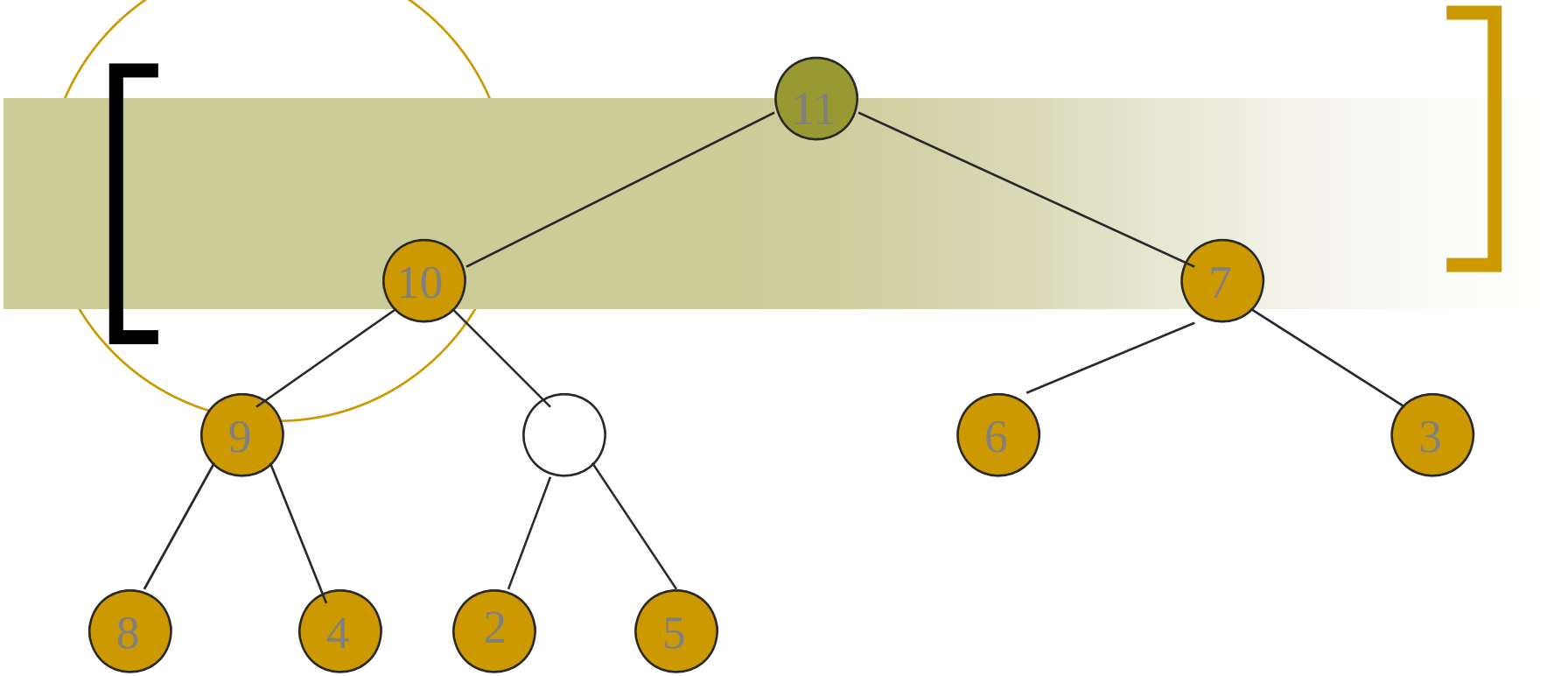
مکان 1را بیابید؟

Initializing A Max Heap

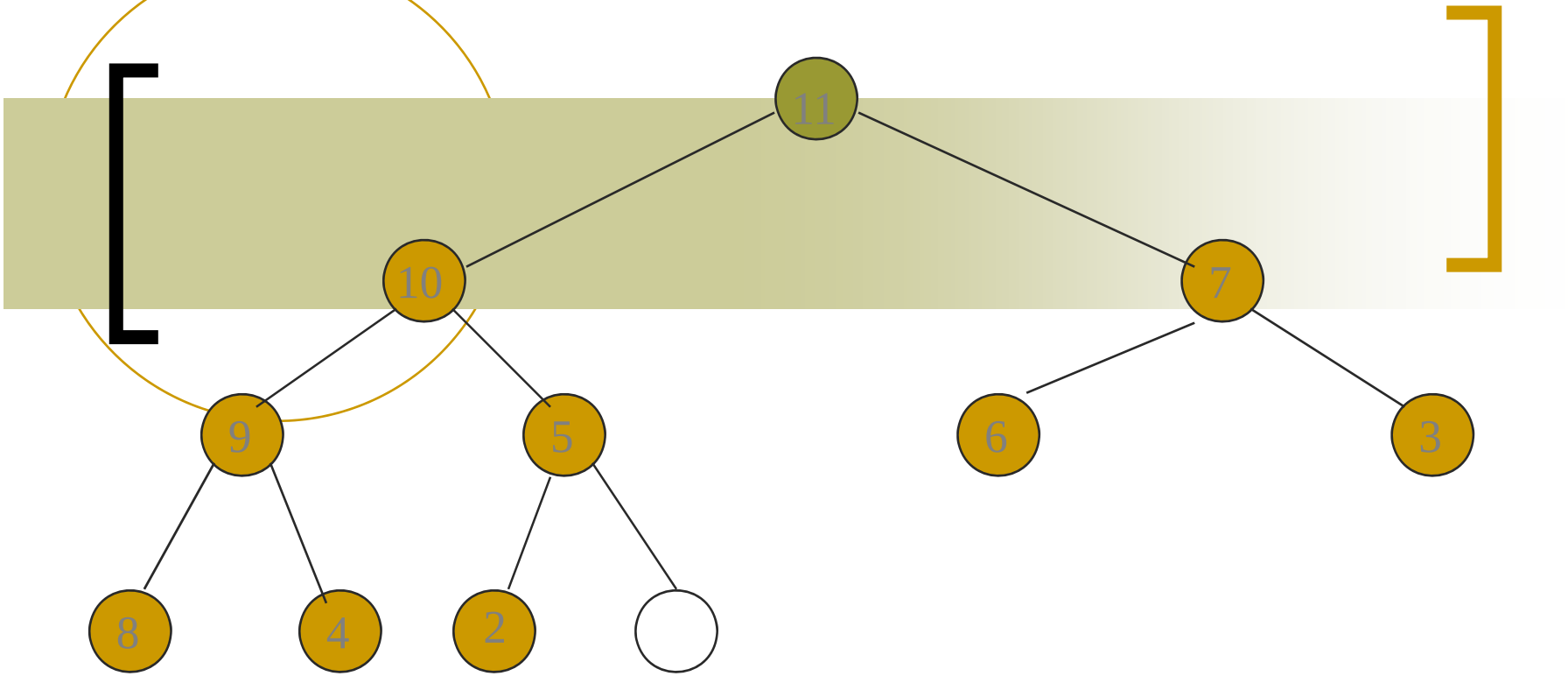


یافتن مکان 1

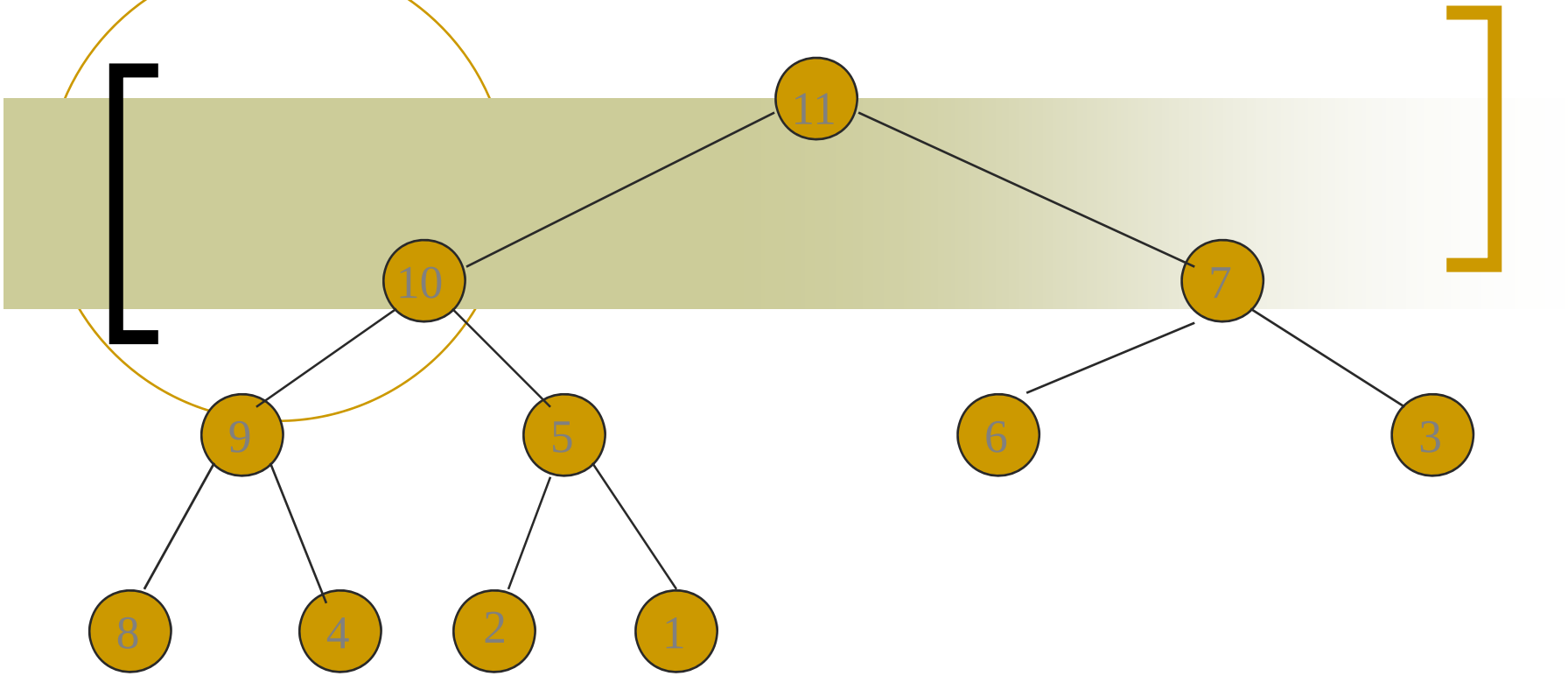
Initializing A Max Heap



Initializing A Max Heap

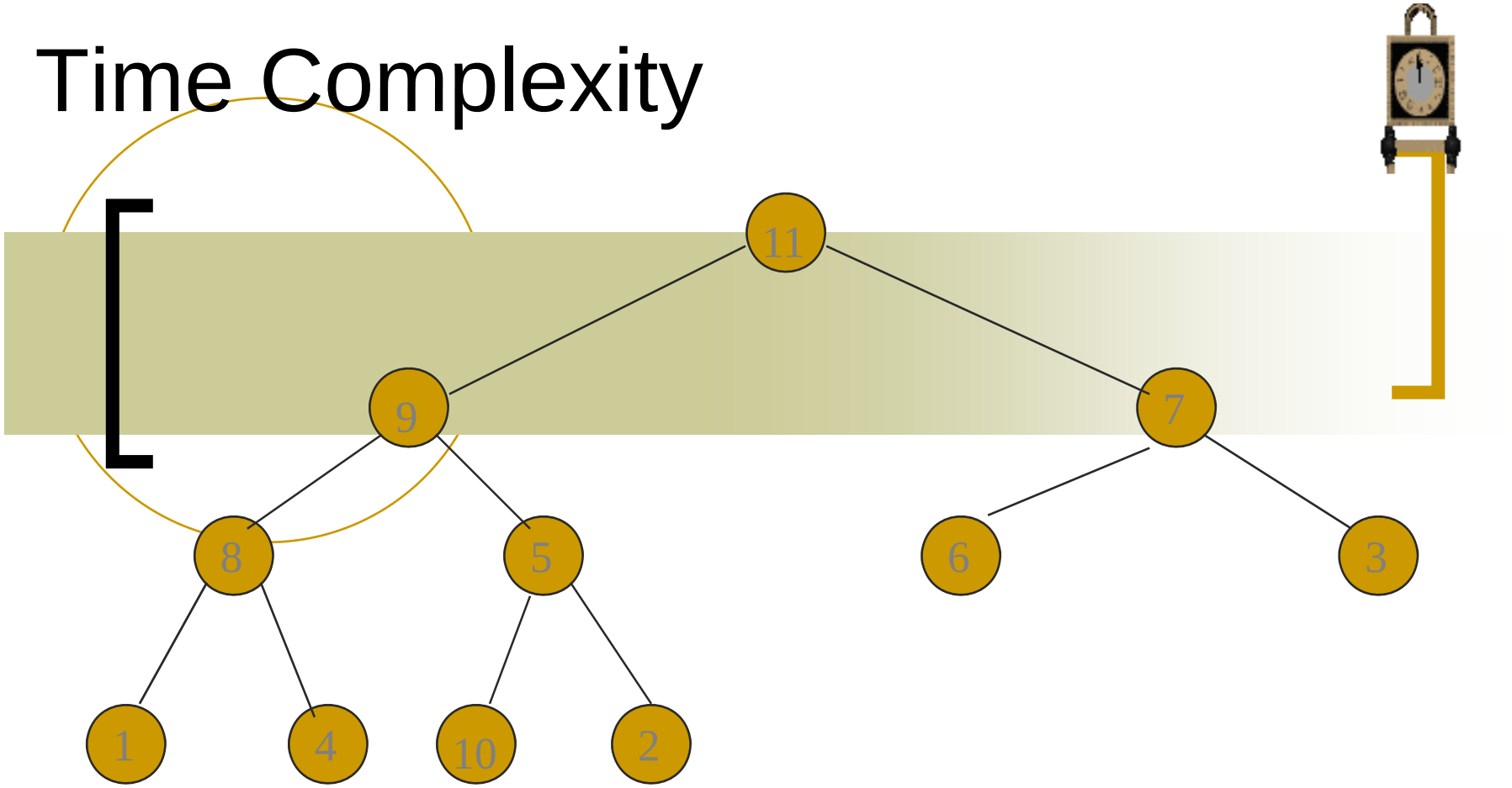


Initializing A Max Heap



انجام شد.

Time Complexity



ارتفاع درخت heap = h

تعداد زیر درخت ها با ریشه در level j $= 2^{j-1}$

پیچیدگی زمانی برای هر زیر درخت = O(h-j+1)

Complexity

پیچیدگی زمانی برای زیر درخت سطح j $2^{j-1}(h-j+1) = t(j) <= j$

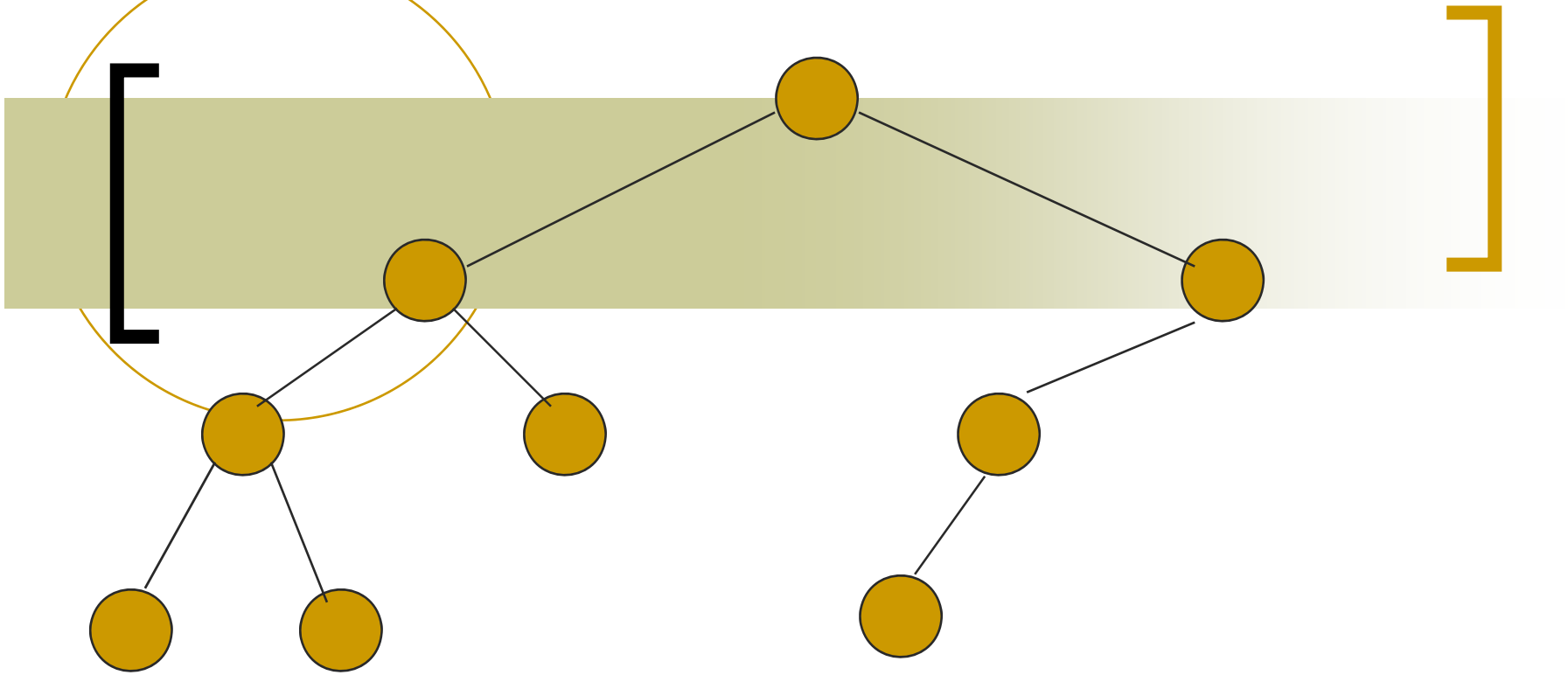
زمان کل $t(1) + t(2) + \ldots + t(h-1) = O(n)$

Extended Binary Trees

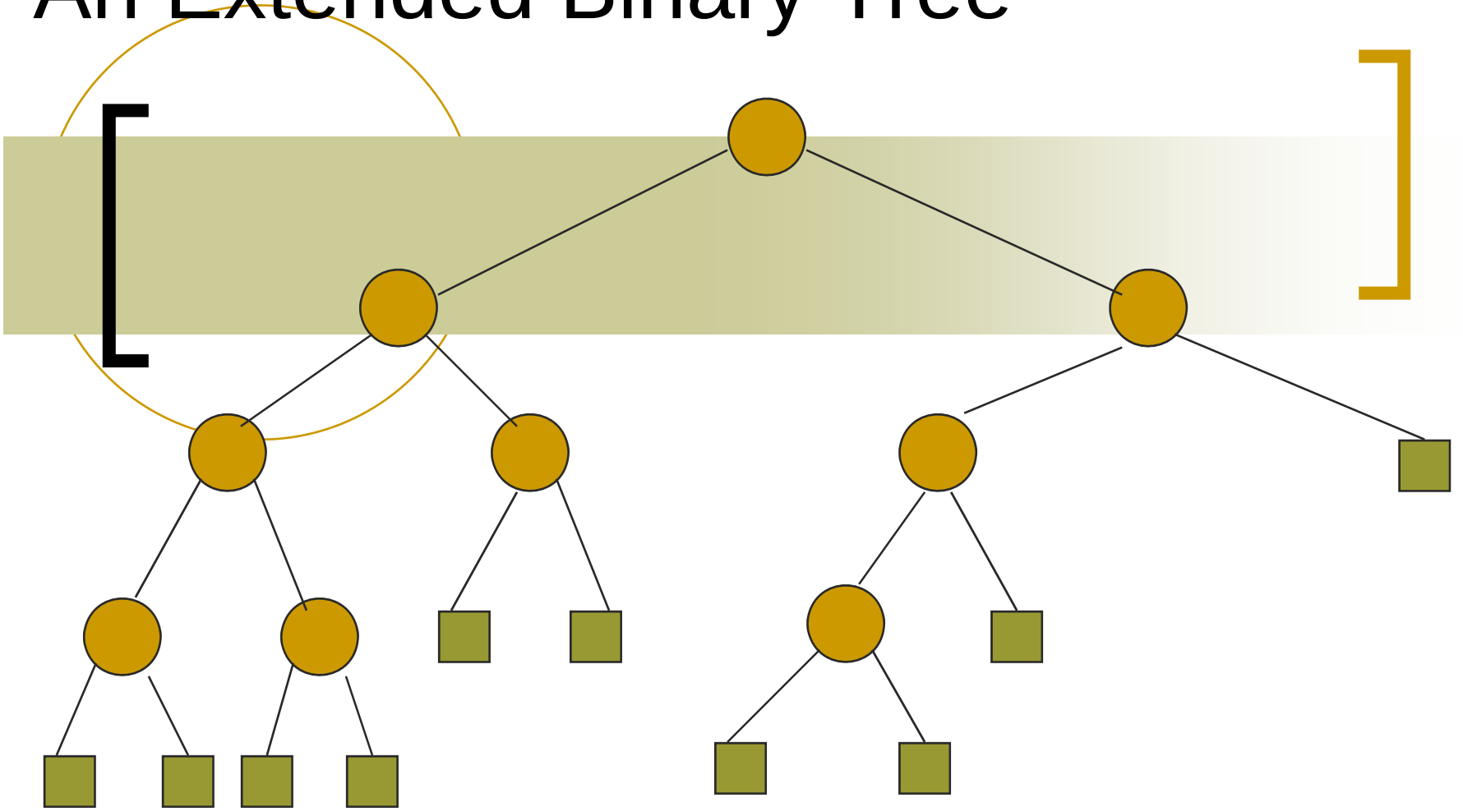
یک درخت باینری را در نظر بگیریدو اضافه کنید یک نود خارجی هر جایی زیر درخت نداشته باشد.

به همین دلیل به آن درخت باینری گسترش یافته می گویند.

A Binary Tree



An Extended Binary Tree



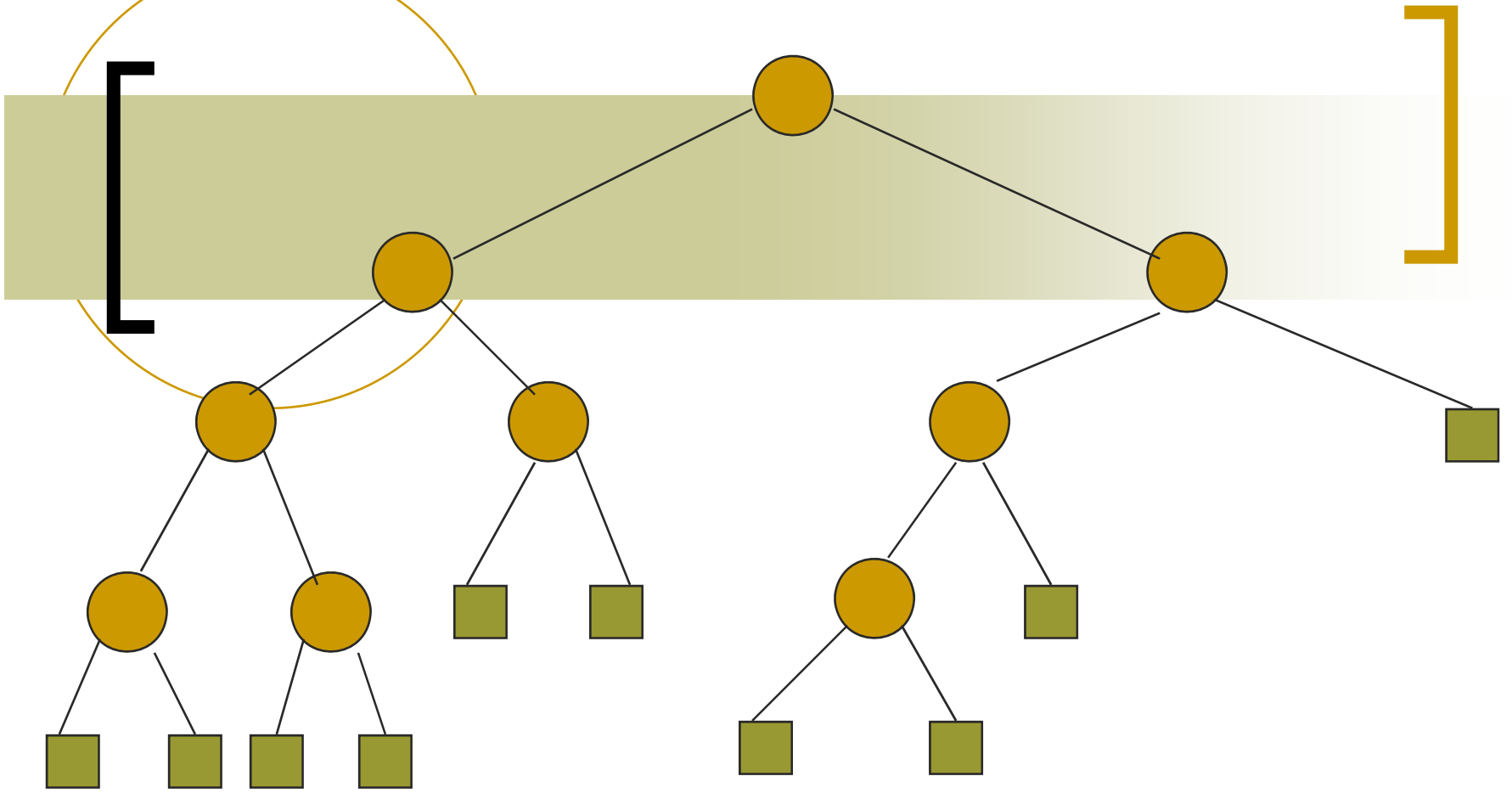
تعداد گره های خارجی = n+1

The Function s()

برای هر گره x در درخت باینری گسترش یافته با استفاده از تابع s(x) قادریم:

طول کوتاهترین مسیر از x به گره خارجی در زیر درخت های ریشه محاسبه کرد.

s() Values Example



Binary Search Trees

- Dictionary Operations:
  - get(key)
  - put(key, value)
  - remove(key)
- Additional operations:
  - ascend()
  - get(index) (اندیس گره در درخت باینری را برمی گرداند.)
  - remove(index) (محتوای اندیس در درخت باینری را حذف می‌کند)

Complexity Of Dictionary Operations

Data Structure	Worst Case	Expected
Hash Table	O(n)	O(1)
Binary Search Tree	O(n)	O(log n)
Balanced Binary Search Tree	O(log n)	O(log n)

Complexity Of Other Operations

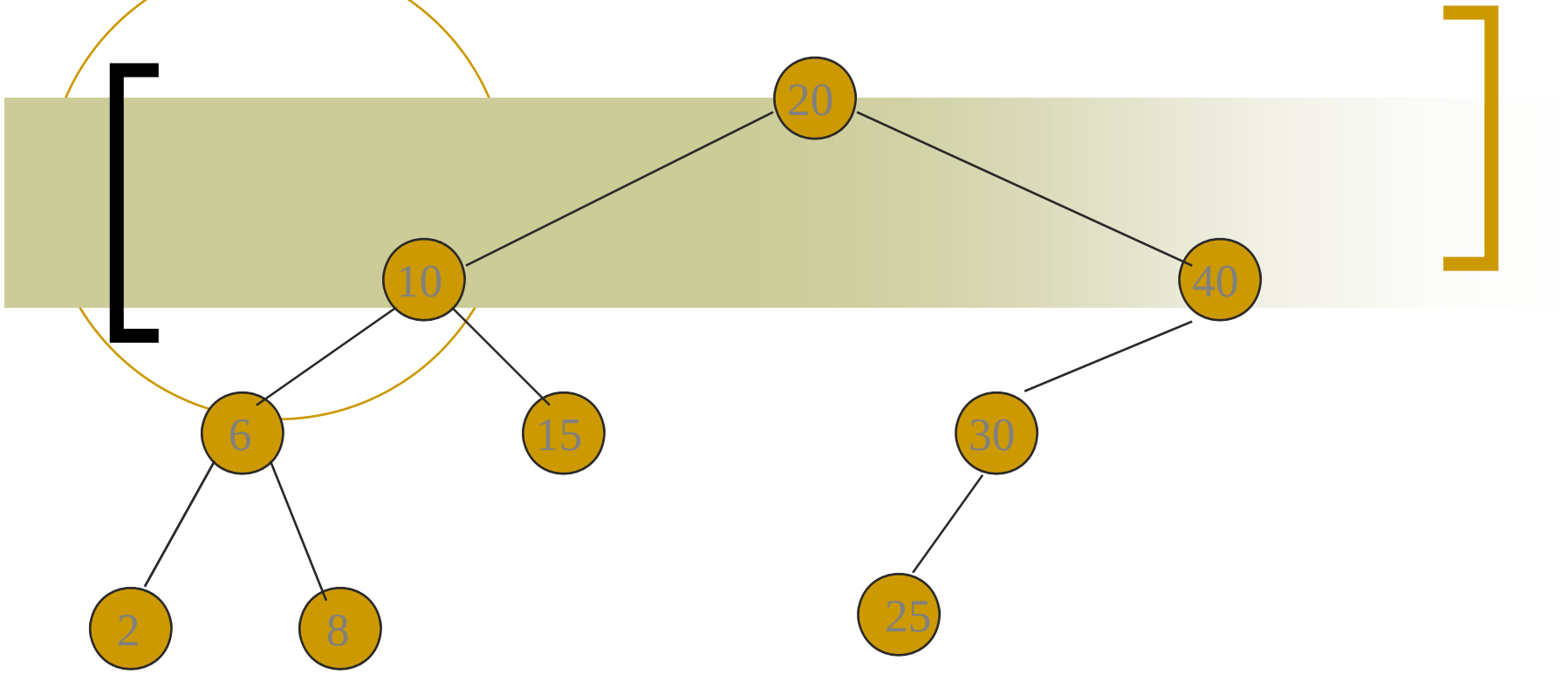
Data Structure	ascend	get and remove
Hash Table	O(D + n log n)	O(D + n log n)
Indexed BST	O(n)	O(n)
Indexed Balanced BST	O(n)	O(log n)

D is number of buckets

Definition Of Binary Search Tree

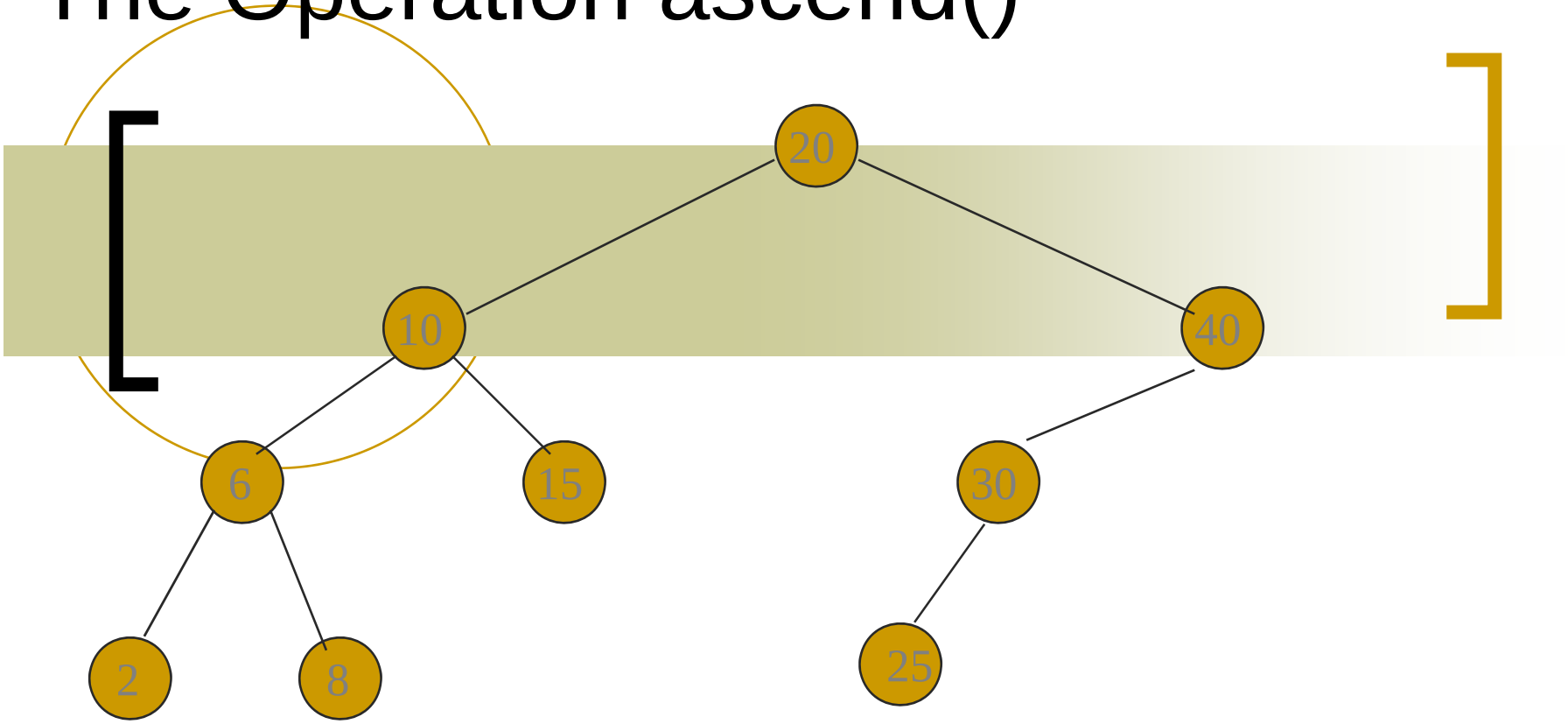
- در یک درخت باینری
- هر گره دارای (key, value)میباشد.
- برای هر گره دلخواه x کلیه عناصر زیر درخت چپ آن کوچکتر از آن وکلیه عناصر زیر درخت راست آن بزرگتر از آن می باشد.

Example Binary Search Tree



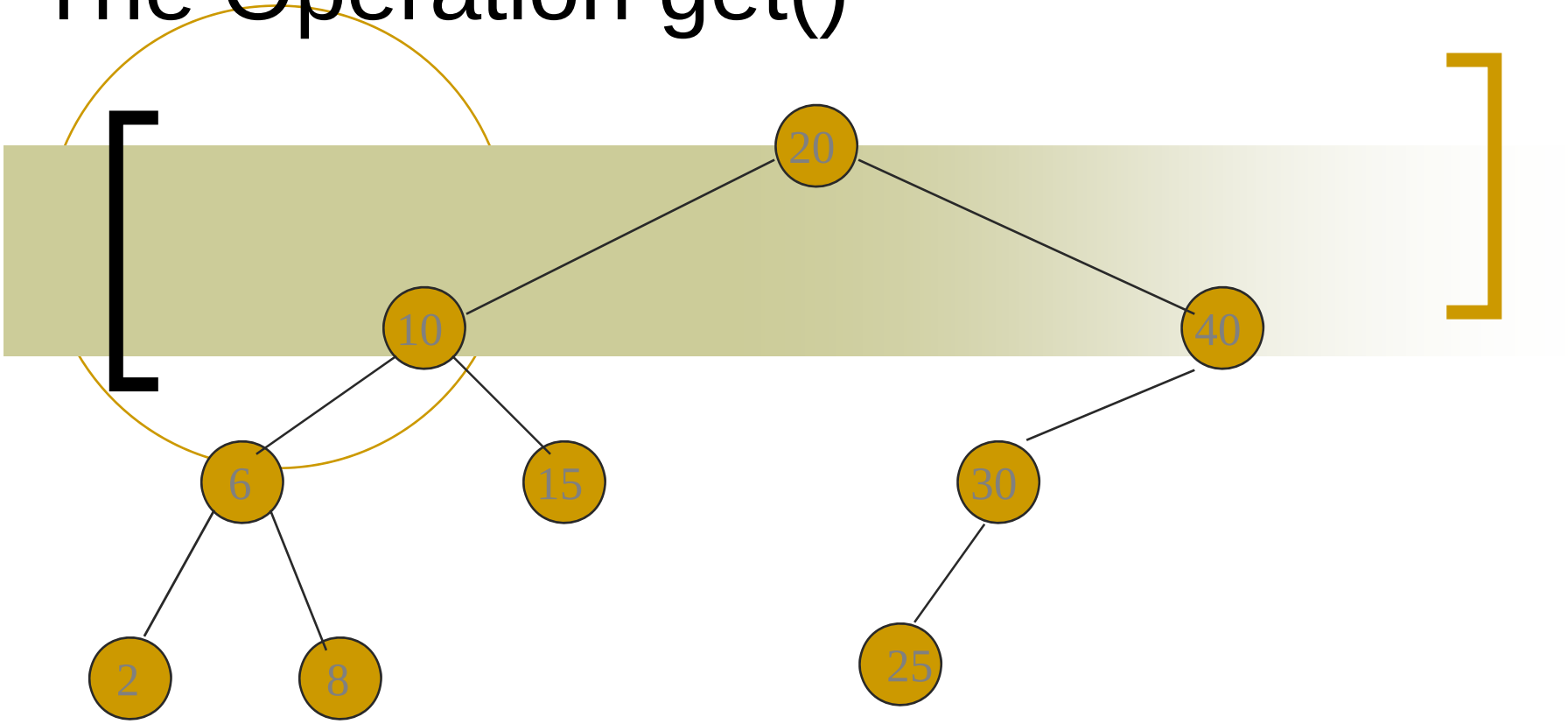
فقط کلید ها را نشان داده است.

The Operation ascend()



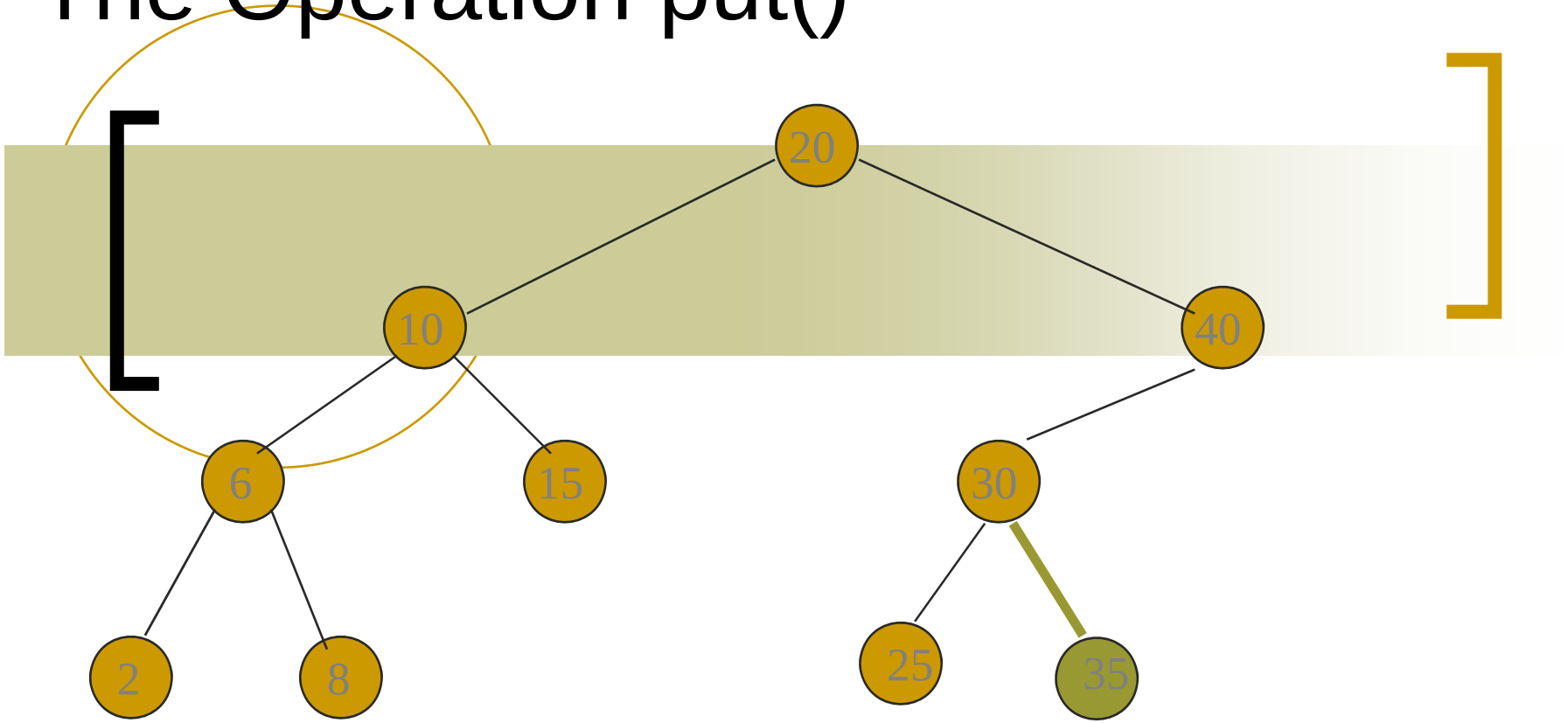
پیمایش میانوندی آن با $O(n)$ امکا ن پذیر است.

The Operation get()



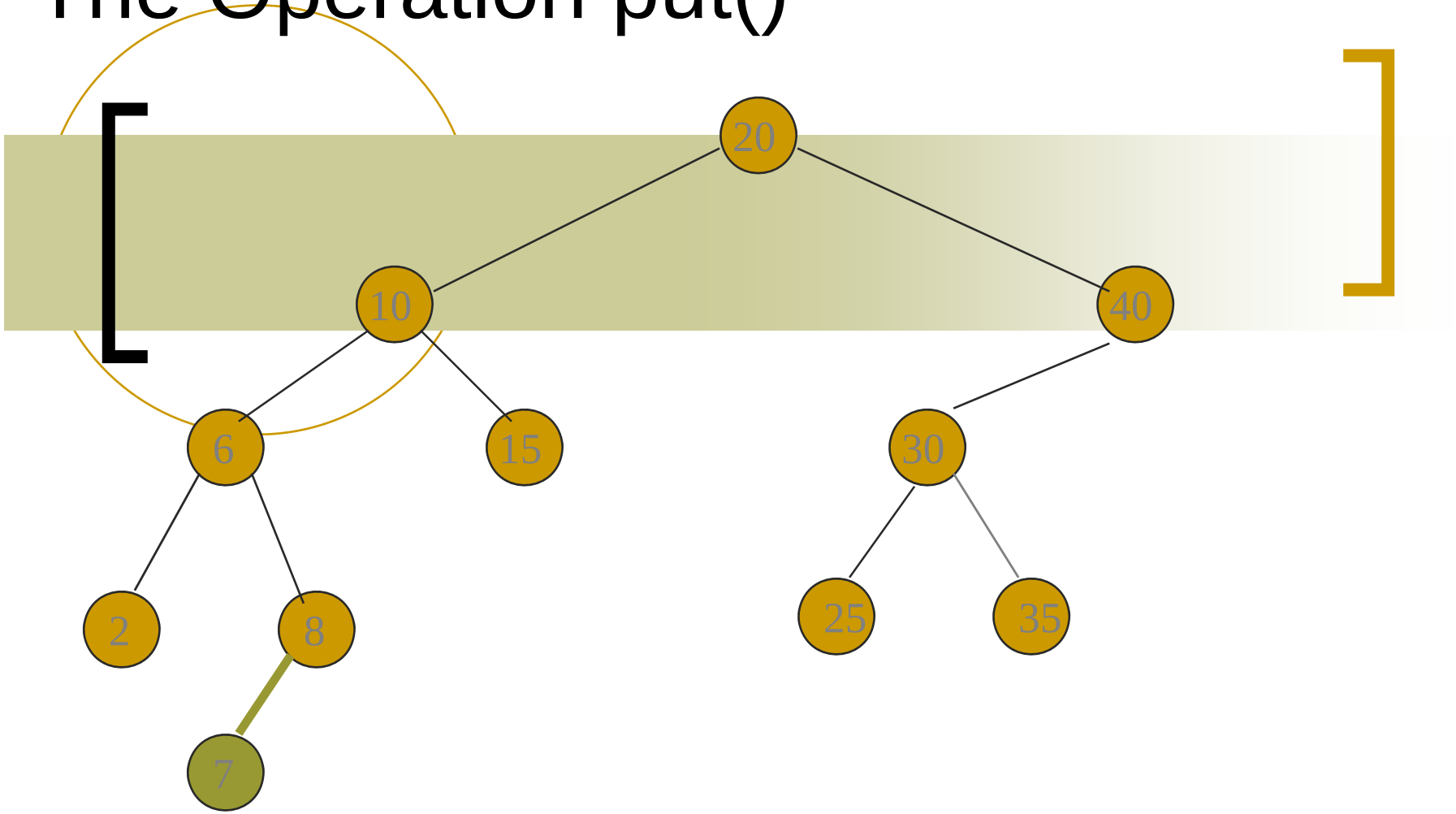
پیچیدگی = O(height) = O(n)

The Operation put()



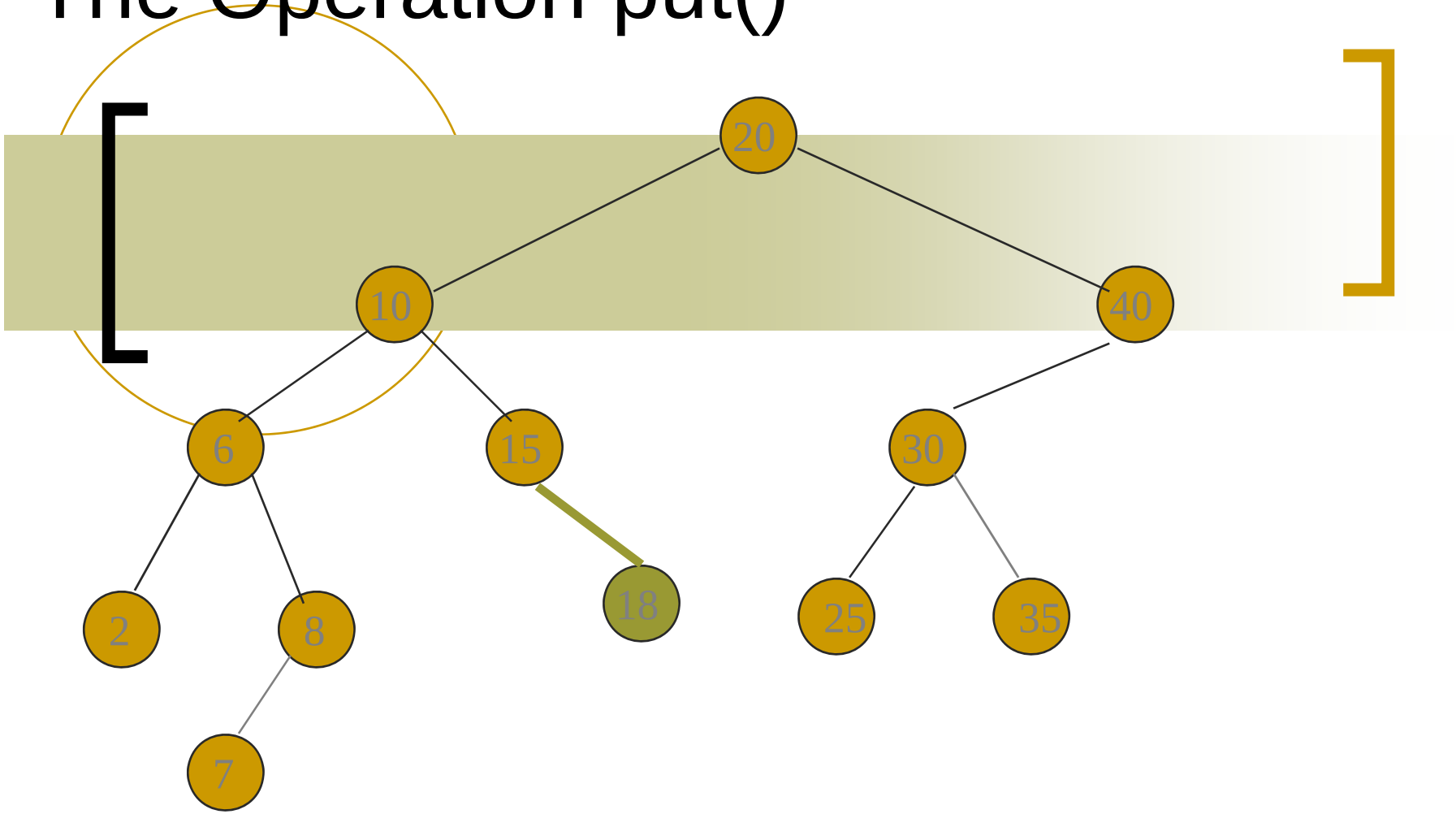
Put 35.

The Operation put()



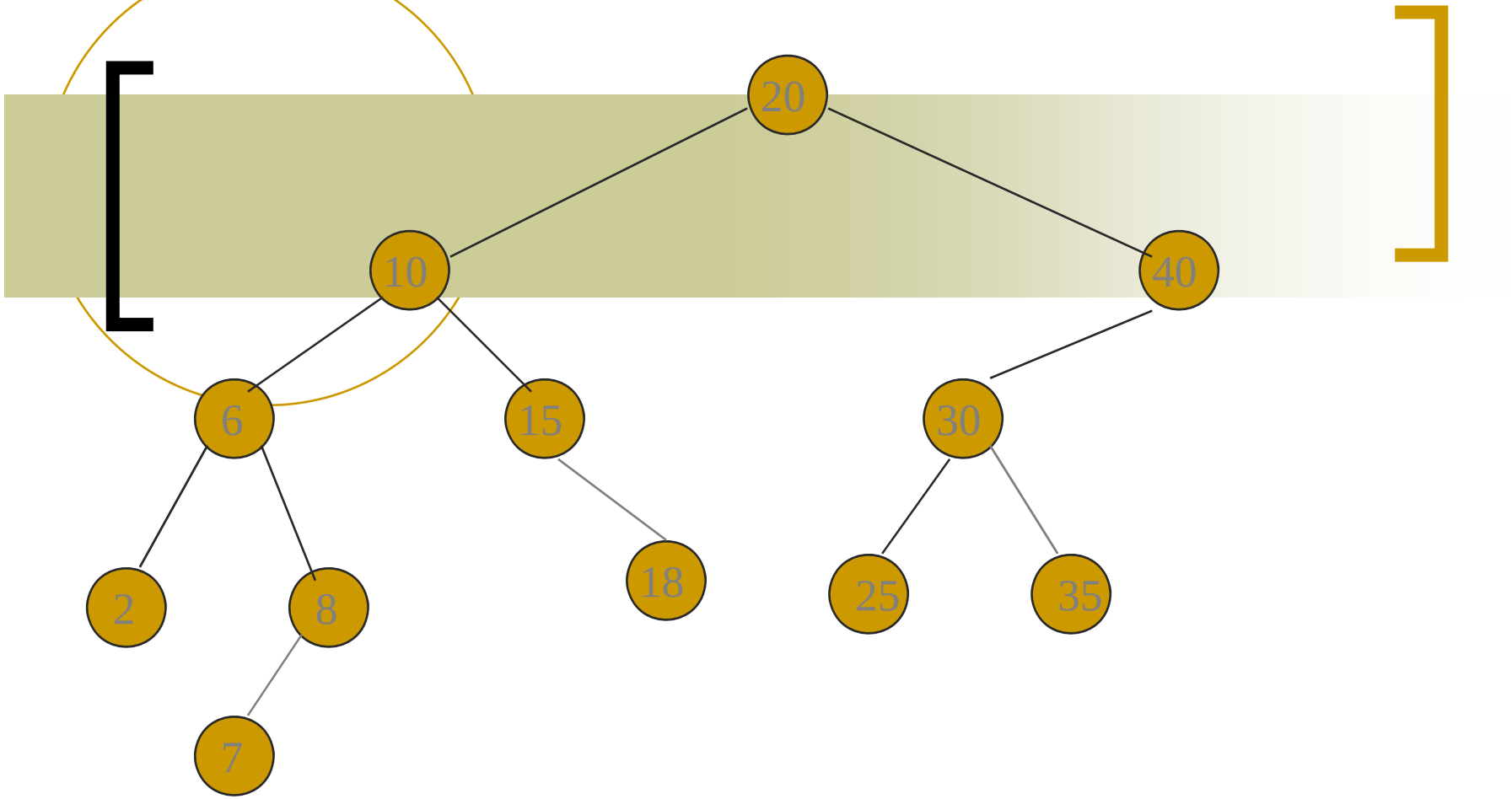
Put 7.

The Operation put()



Put 18.

The Operation put()



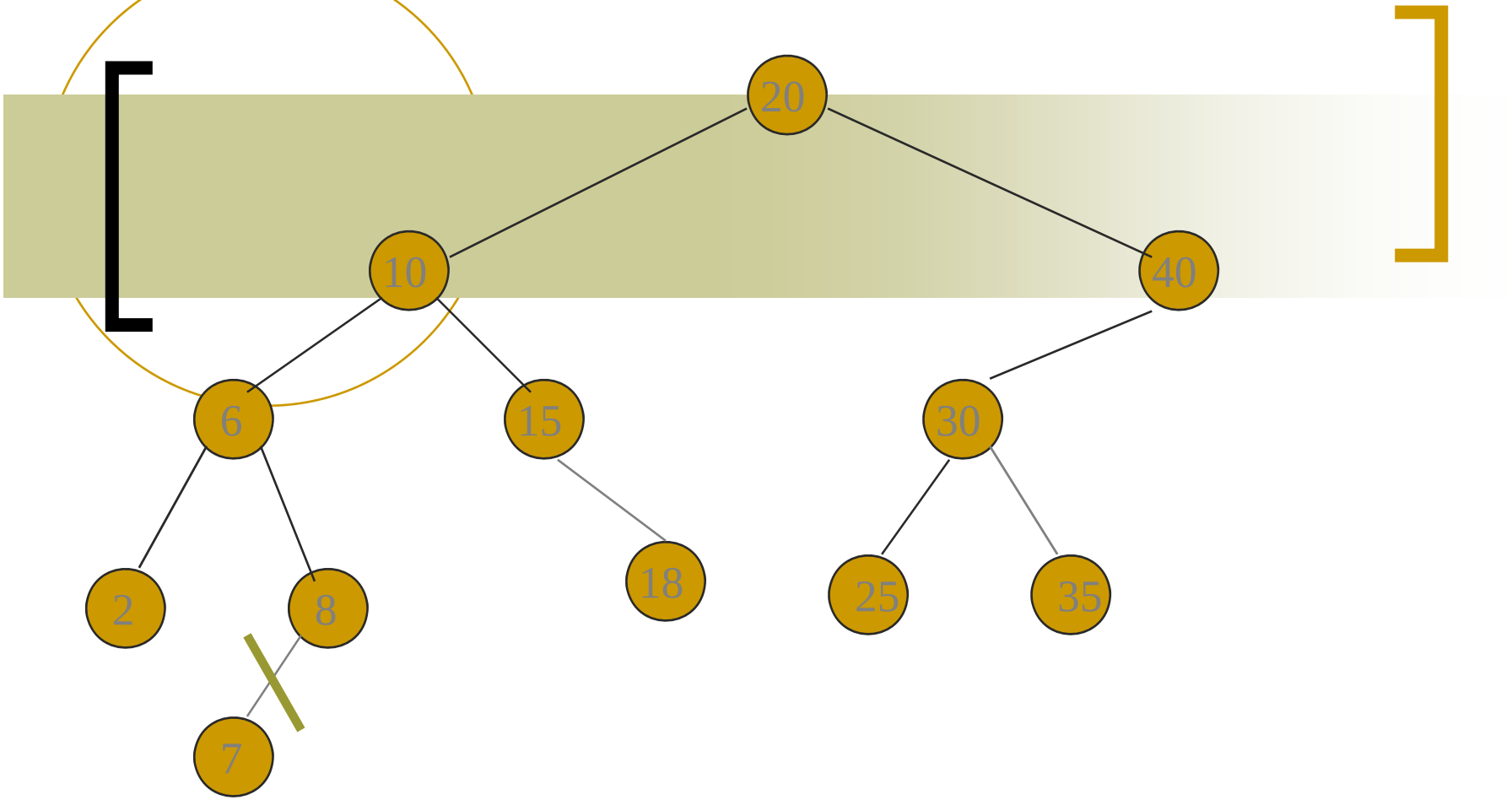
پیچیدگی => put() = O(height).

The Operation remove()

گره در وضعیت های:

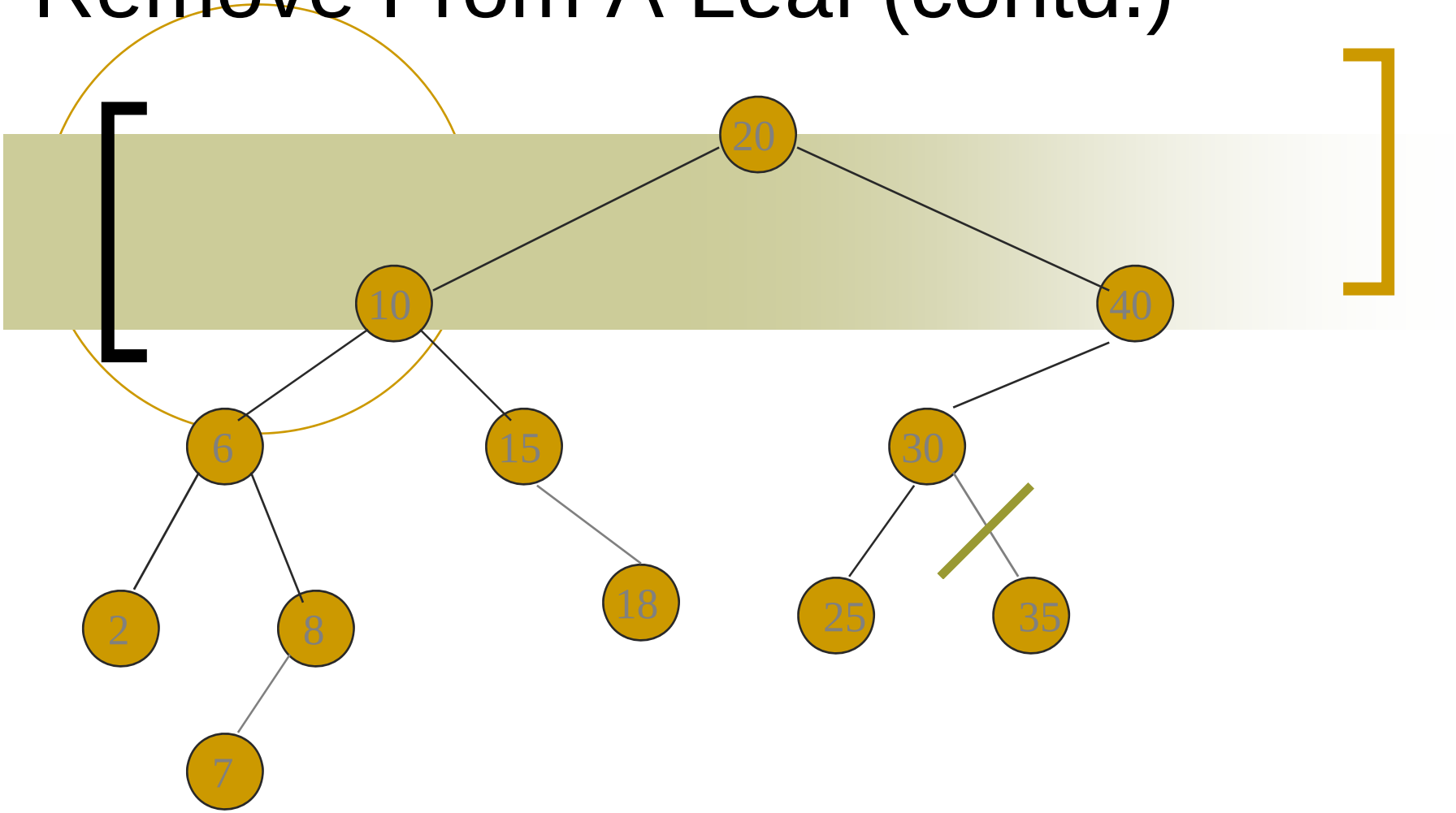
- اگر گره برگ باشد
- اگر گره از درجه یک باشد.
- اگر گره از درجه دو باشد

Remove From A Leaf



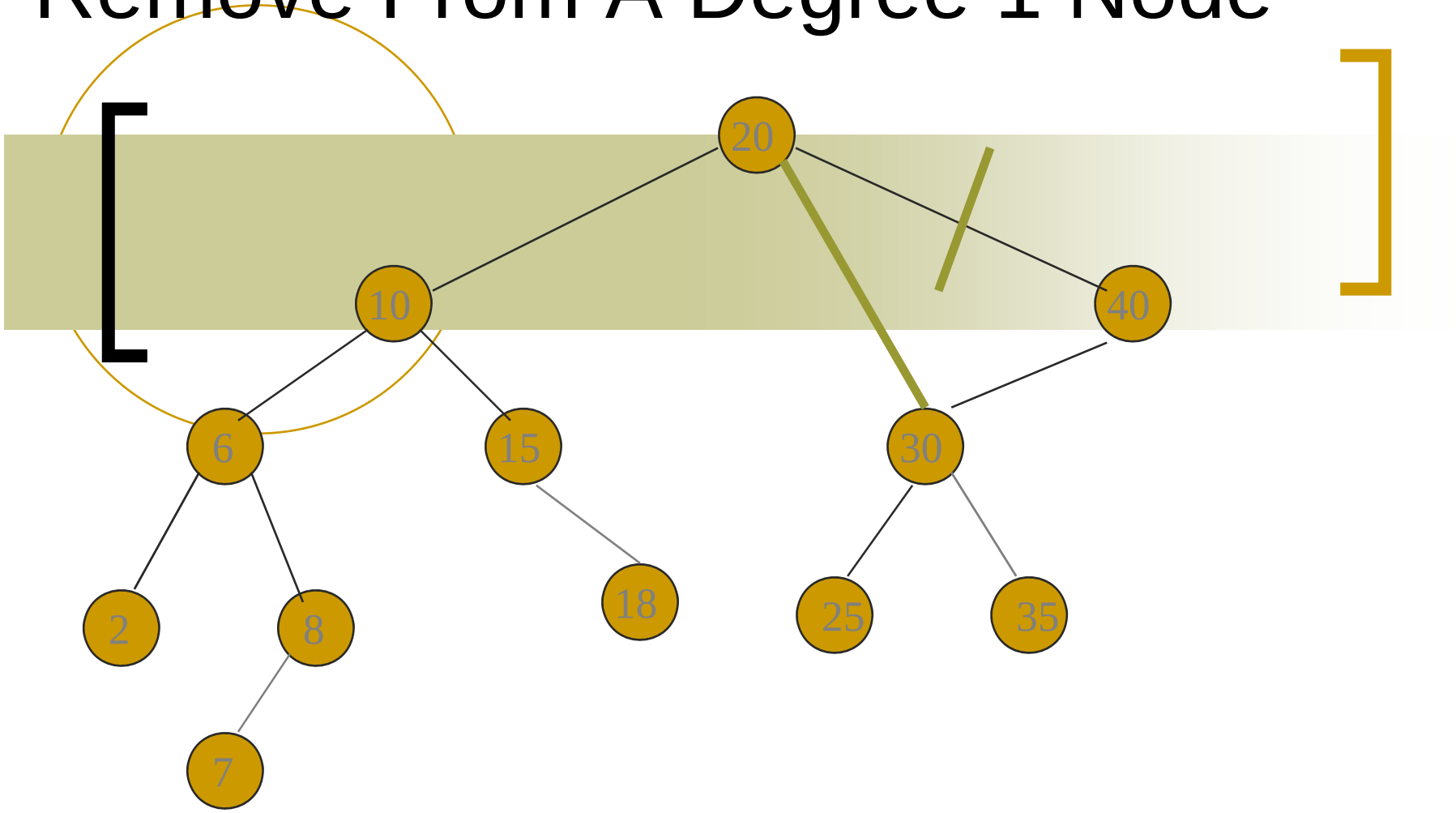
حذف گره برگ =7

Remove From A Leaf (contd.)



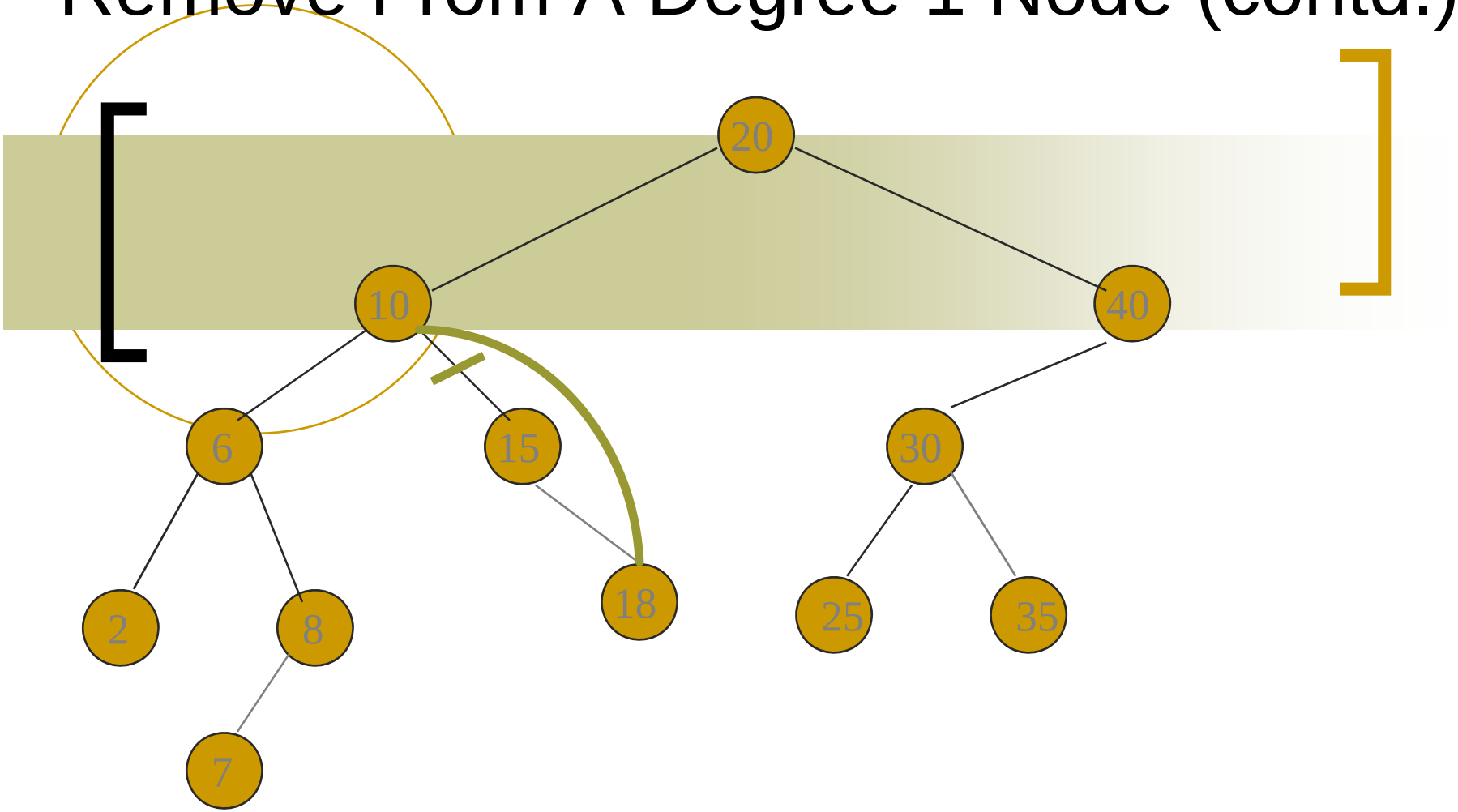
حذف گره برگ =35

Remove From A Degree 1 Node



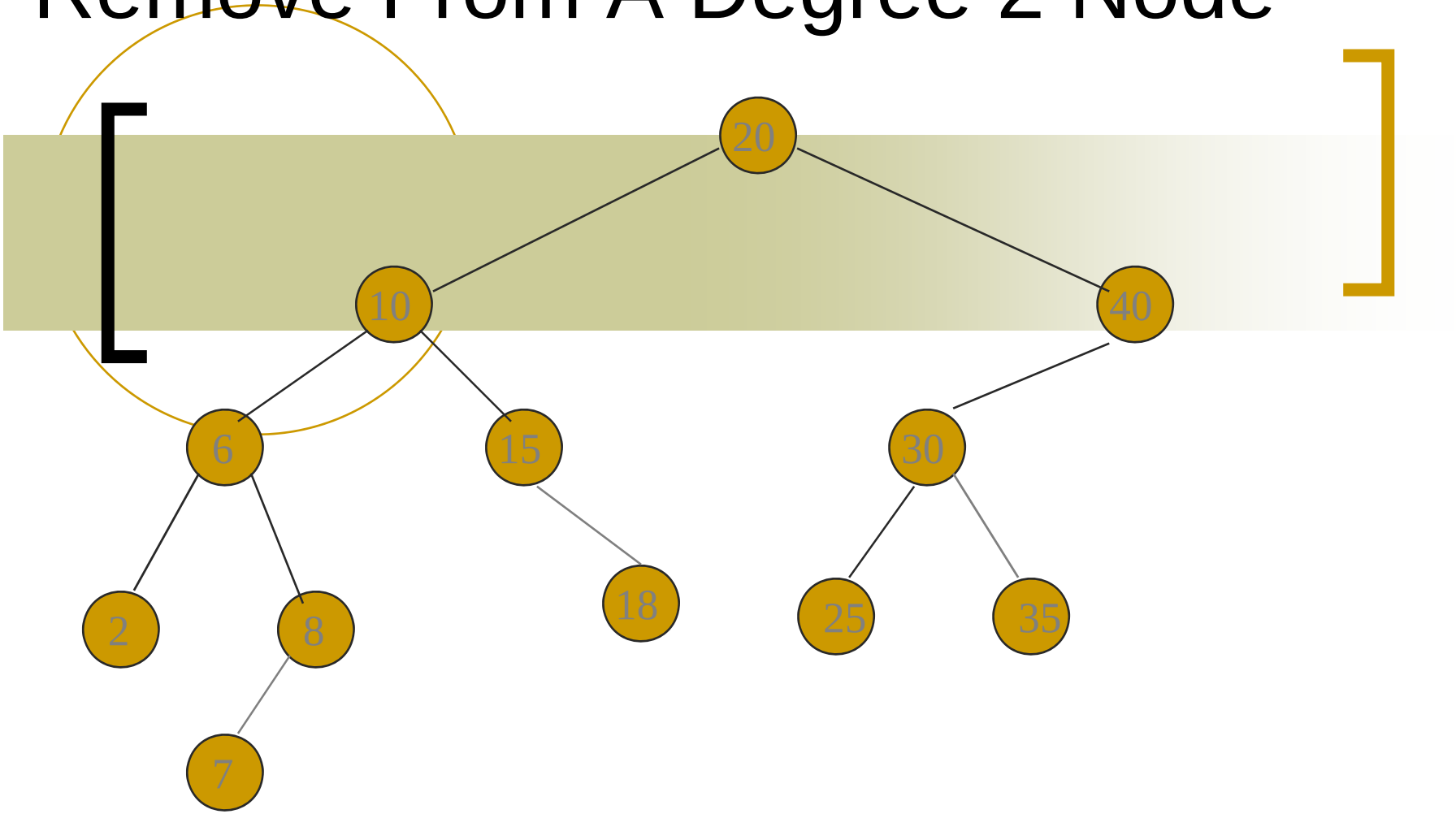
حذف گره تک فرزندی = 40

Remove From A Degree 1 Node (contd.)



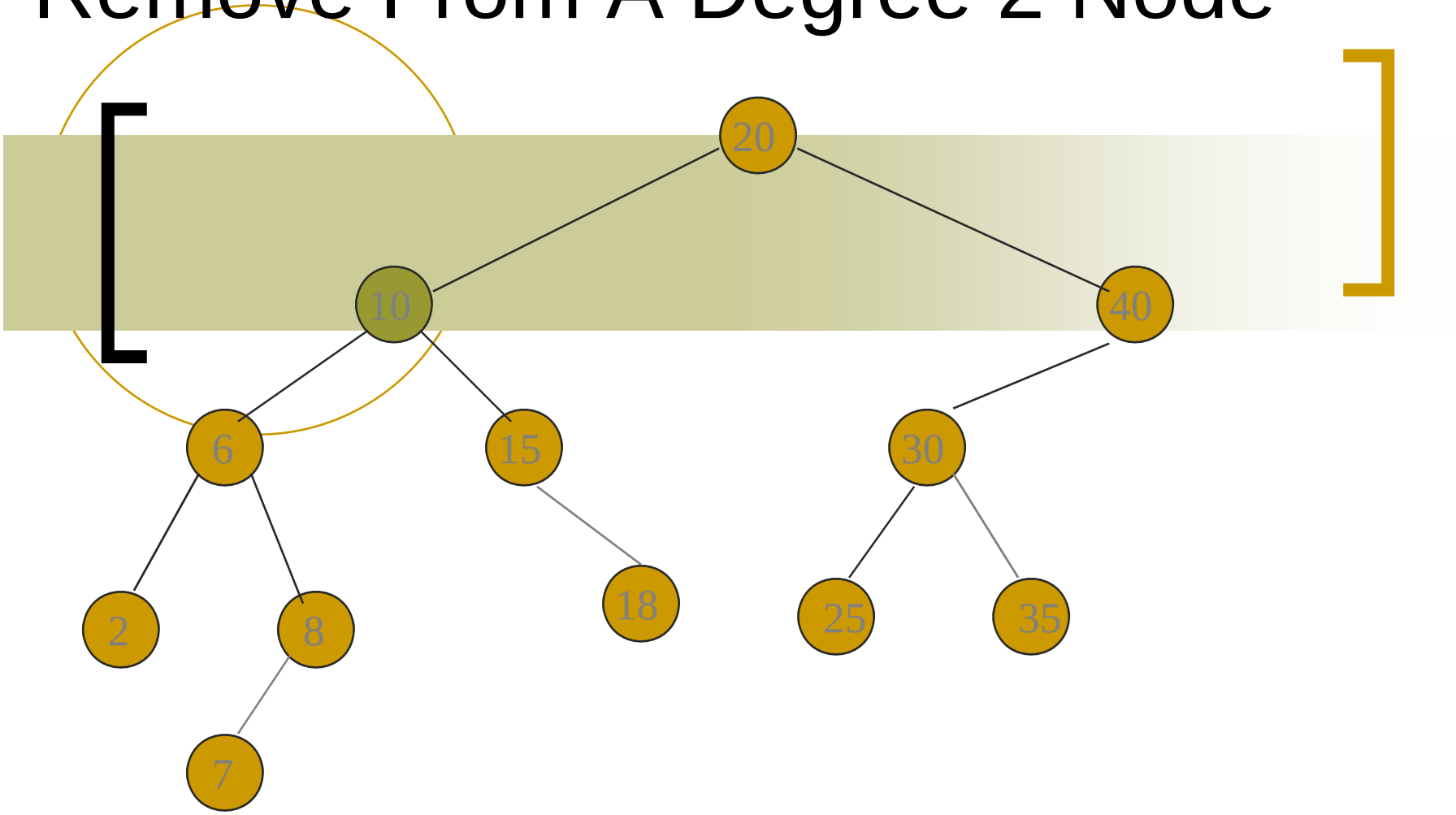
حذف گره تک فرزندی = 15

Remove From A Degree 2 Node



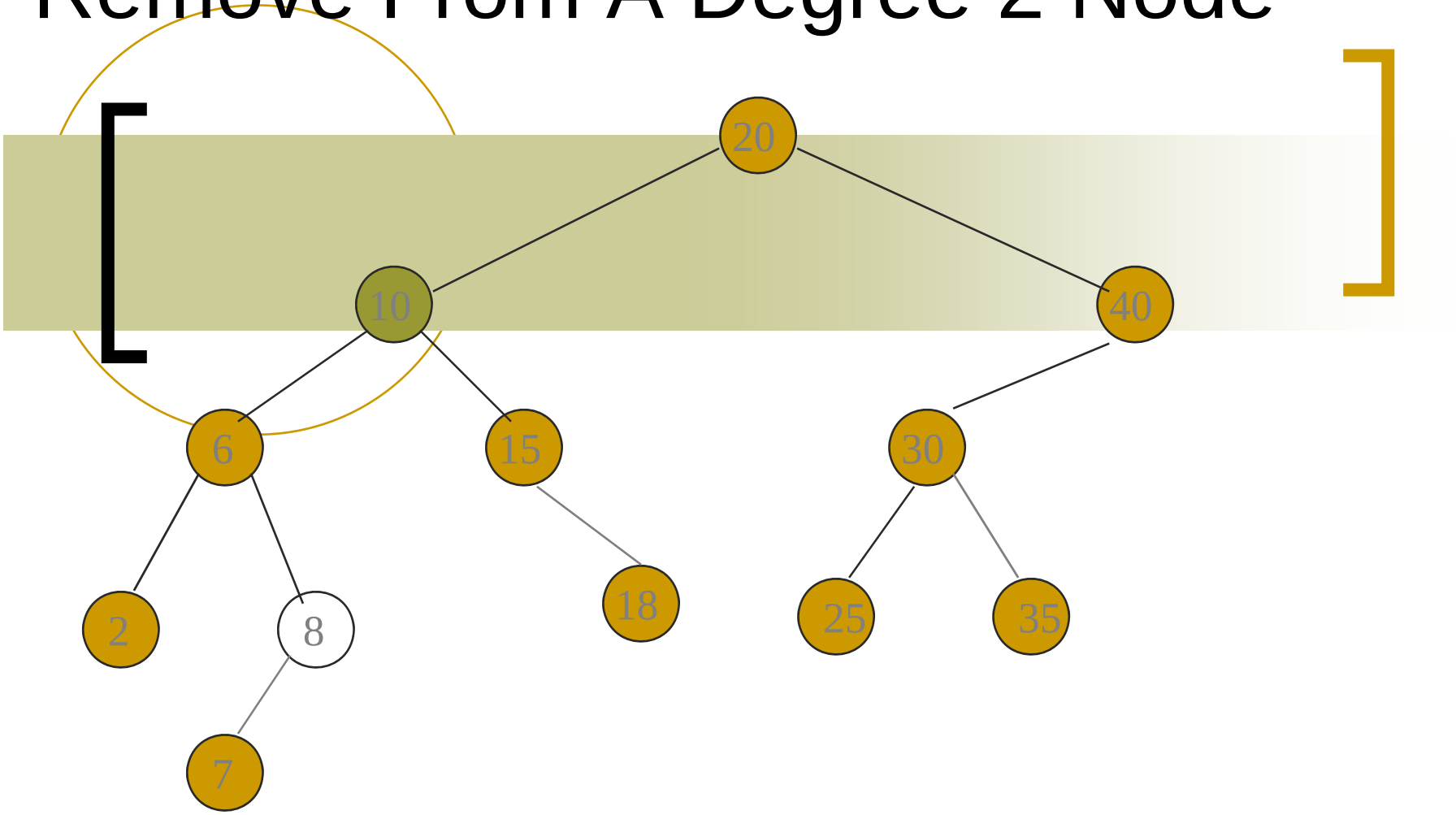
حذف گره دو فرزندی = 10

Remove From A Degree 2 Node



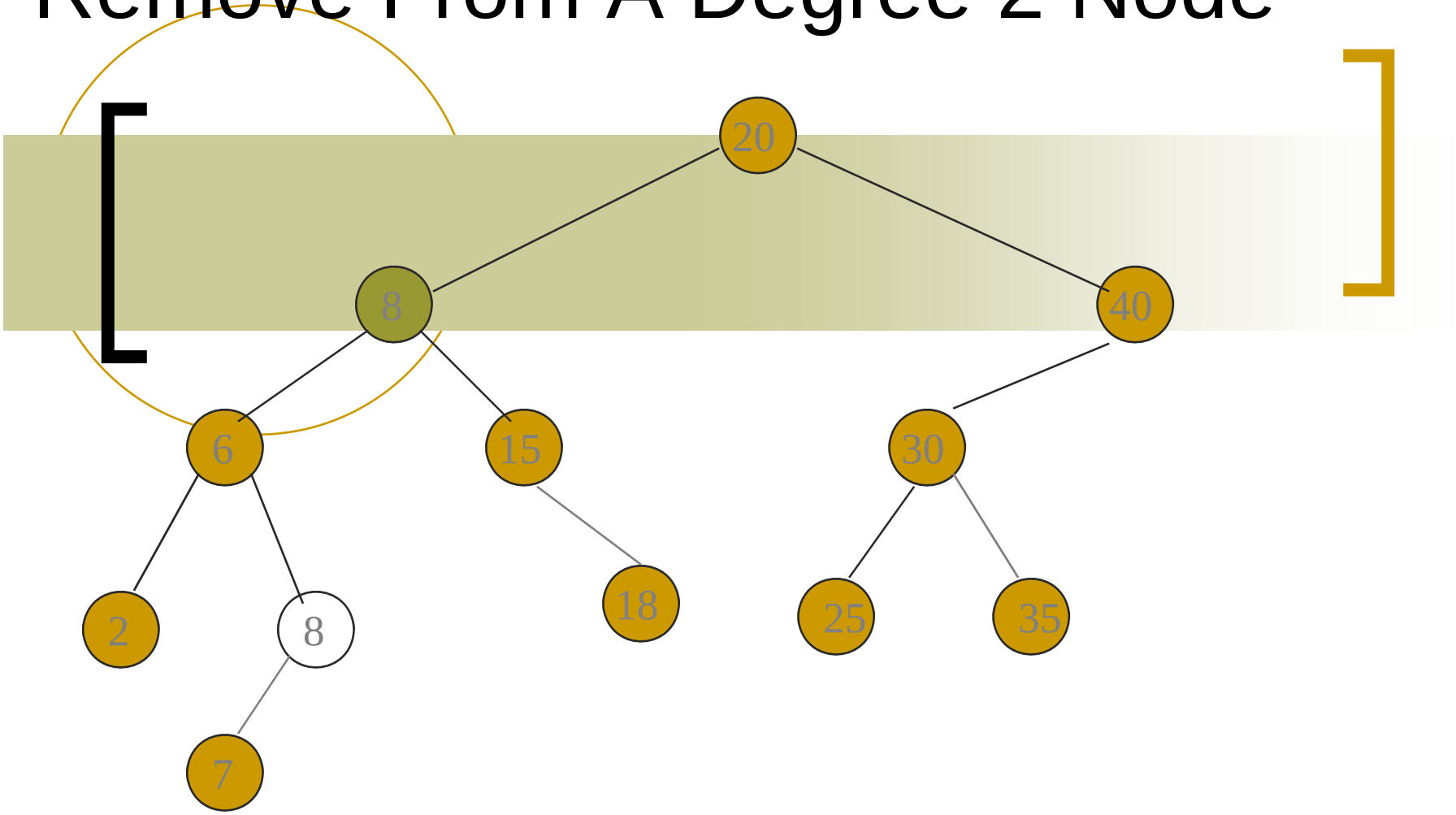
جابه جا کنید با بزرگترین در زیر درخت چپ ویا با کوچکترین در زیر درخت راست.

Remove From A Degree 2 Node



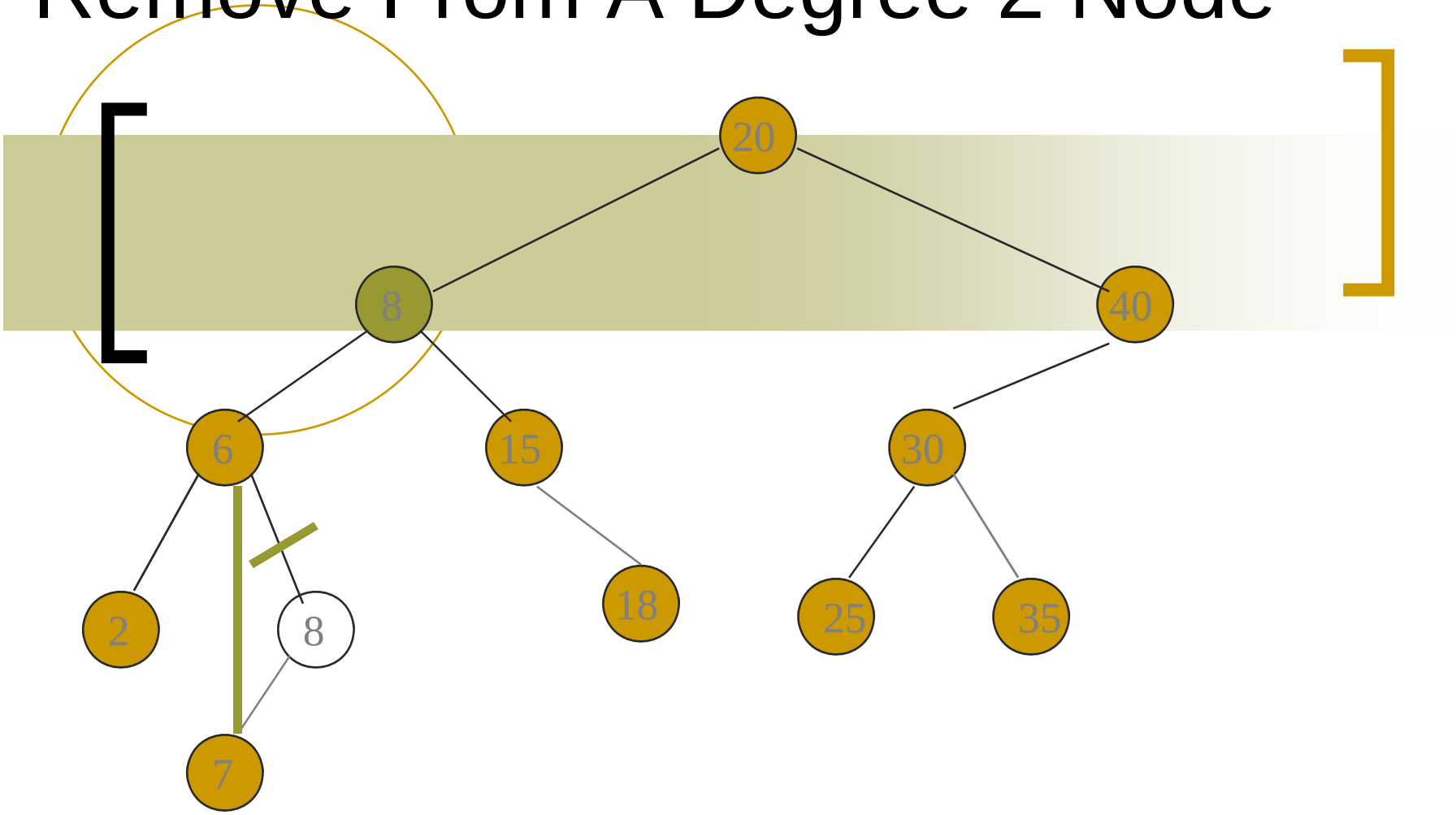
جابه جا کنید با بزرگترین در زیر درخت چپ ویا با کوچکترین در زیر درخت راست.

Remove From A Degree 2 Node



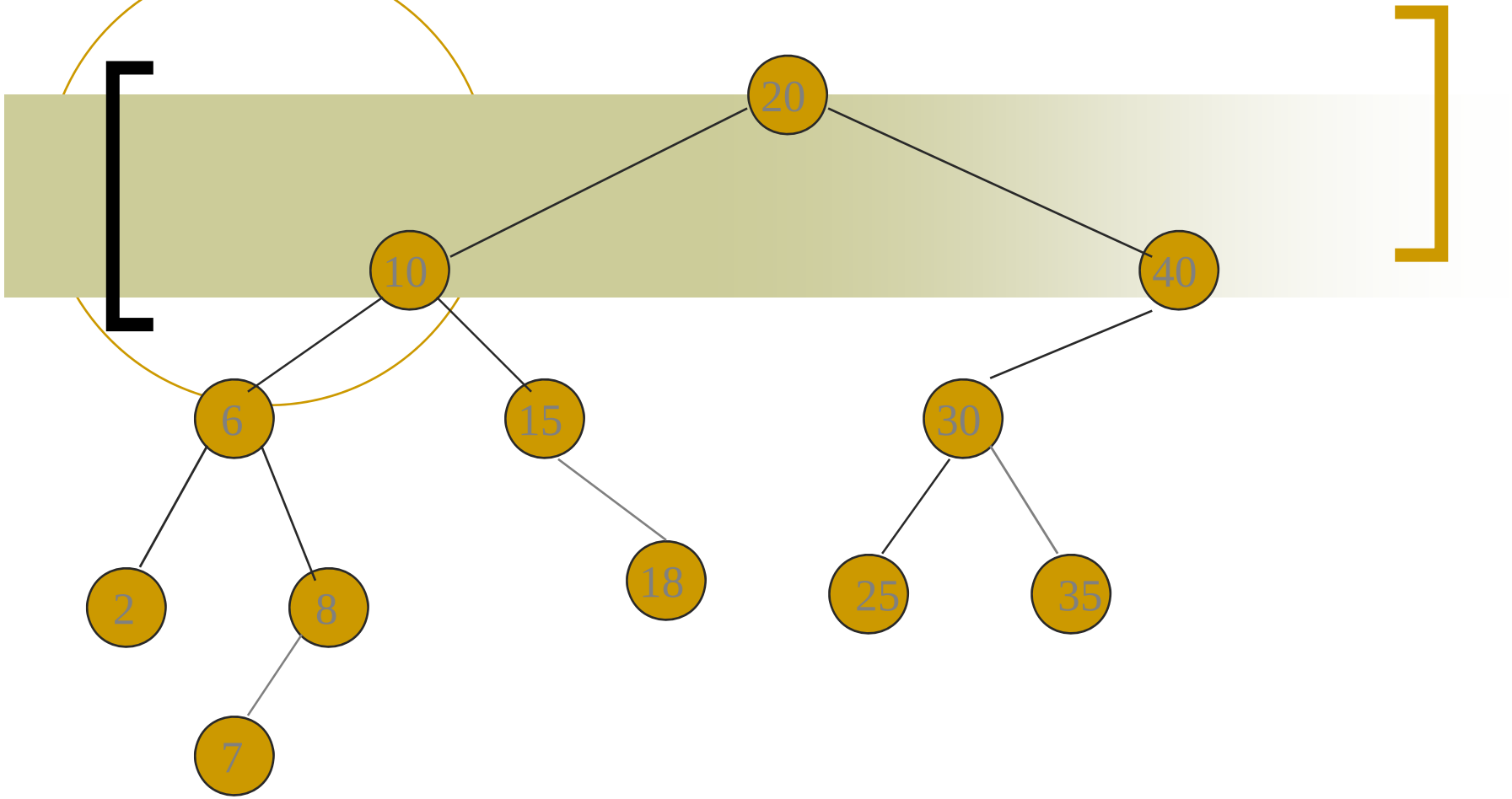
جابه جا کنید با بزرگترین در زیر درخت چپ ویا با کوچکترین در زیر درخت راست.

Remove From A Degree 2 Node



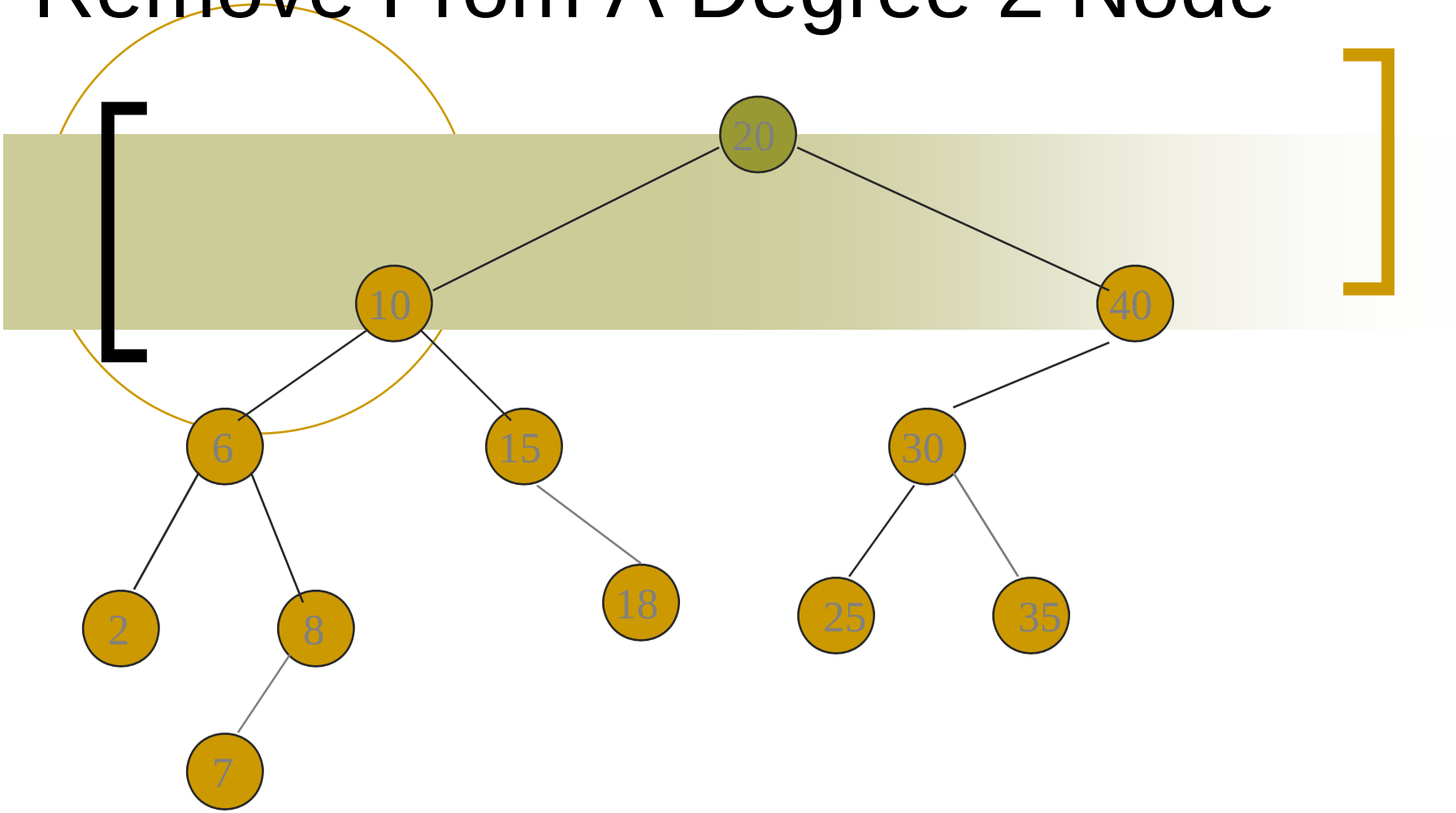
بزرگترین کلید یا باید برگ باشد یا گره تک فرزندی

Another Remove From A Degree 2 Node



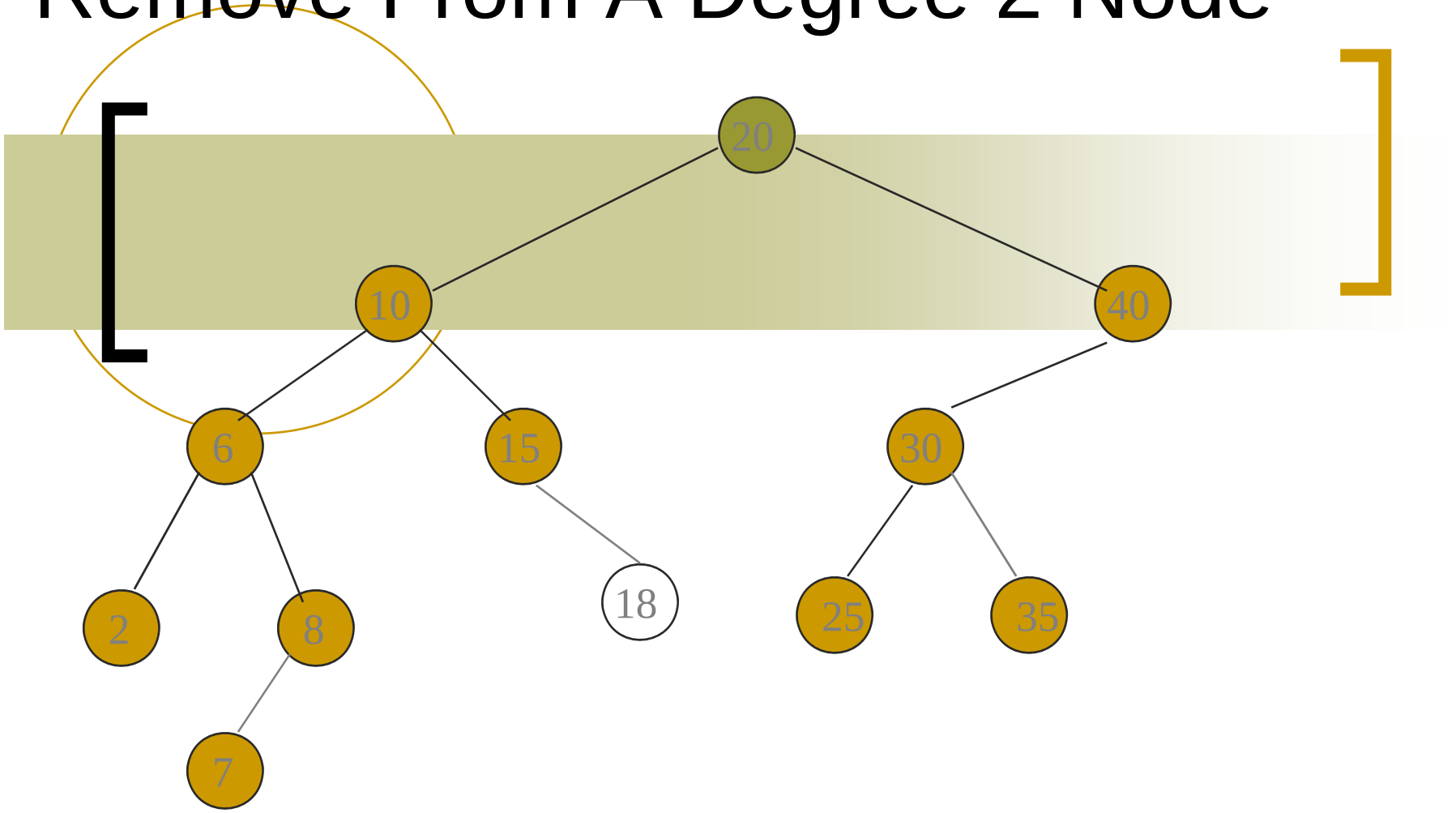
حذف گره دو فرزندی = 20

Remove From A Degree 2 Node



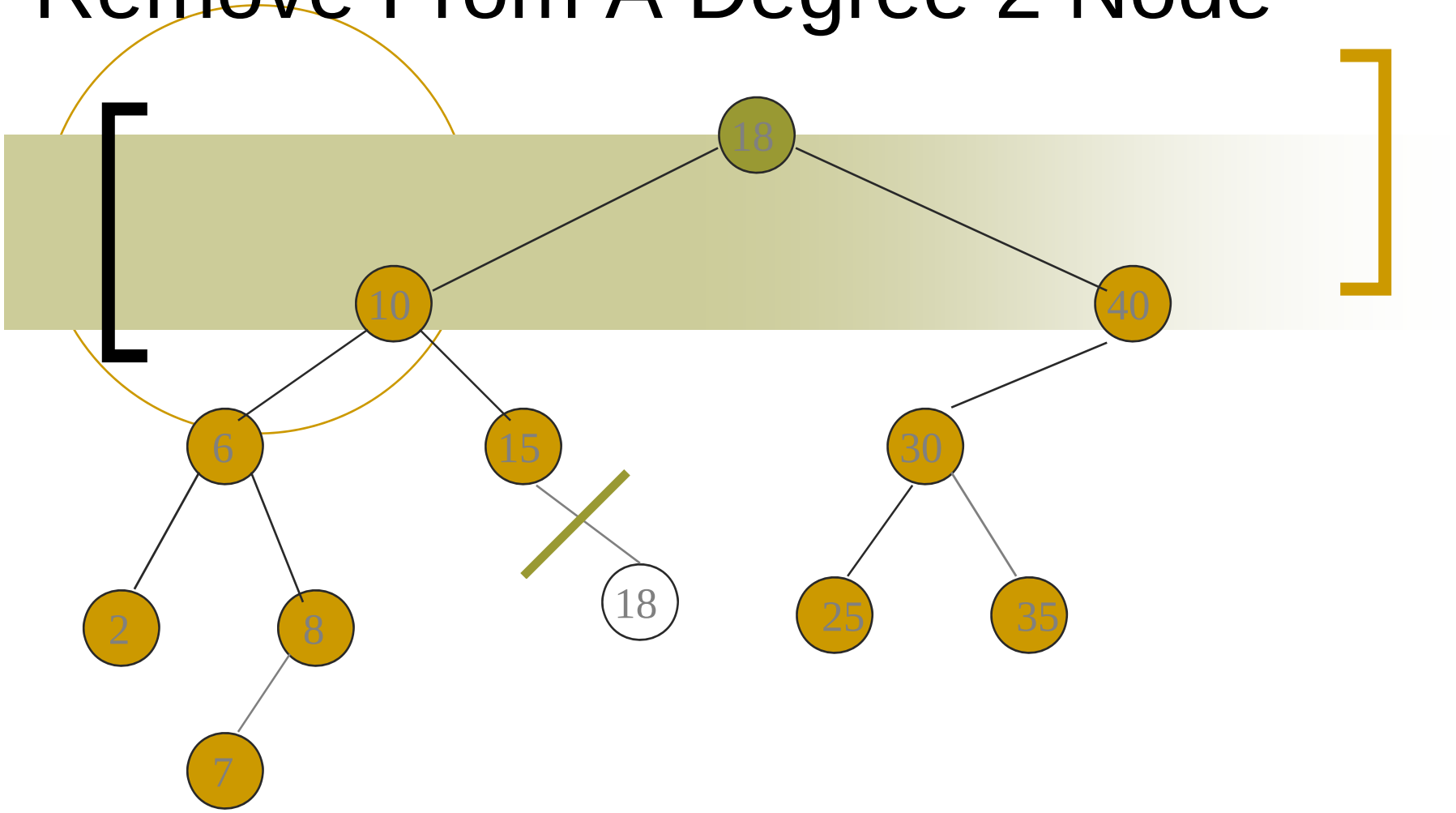
جابه جا کنید با بزرگترین در زیر درخت چپ

Remove From A Degree 2 Node



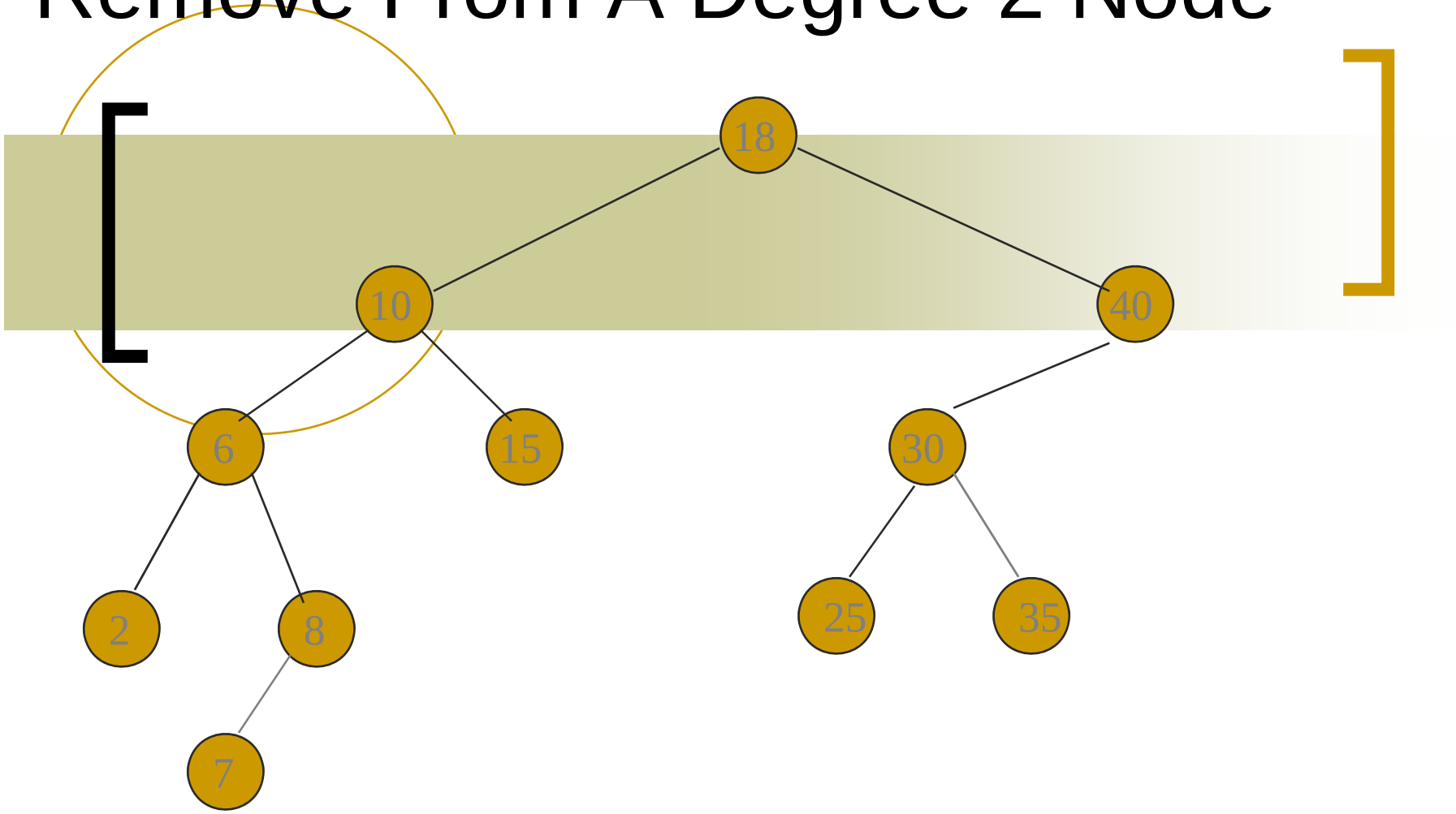
جابه جا کنید با بزرگترین در زیر درخت چپ

Remove From A Degree 2 Node



جابه جا کنید با بزرگترین در زیر درخت چپ

Remove From A Degree 2 Node

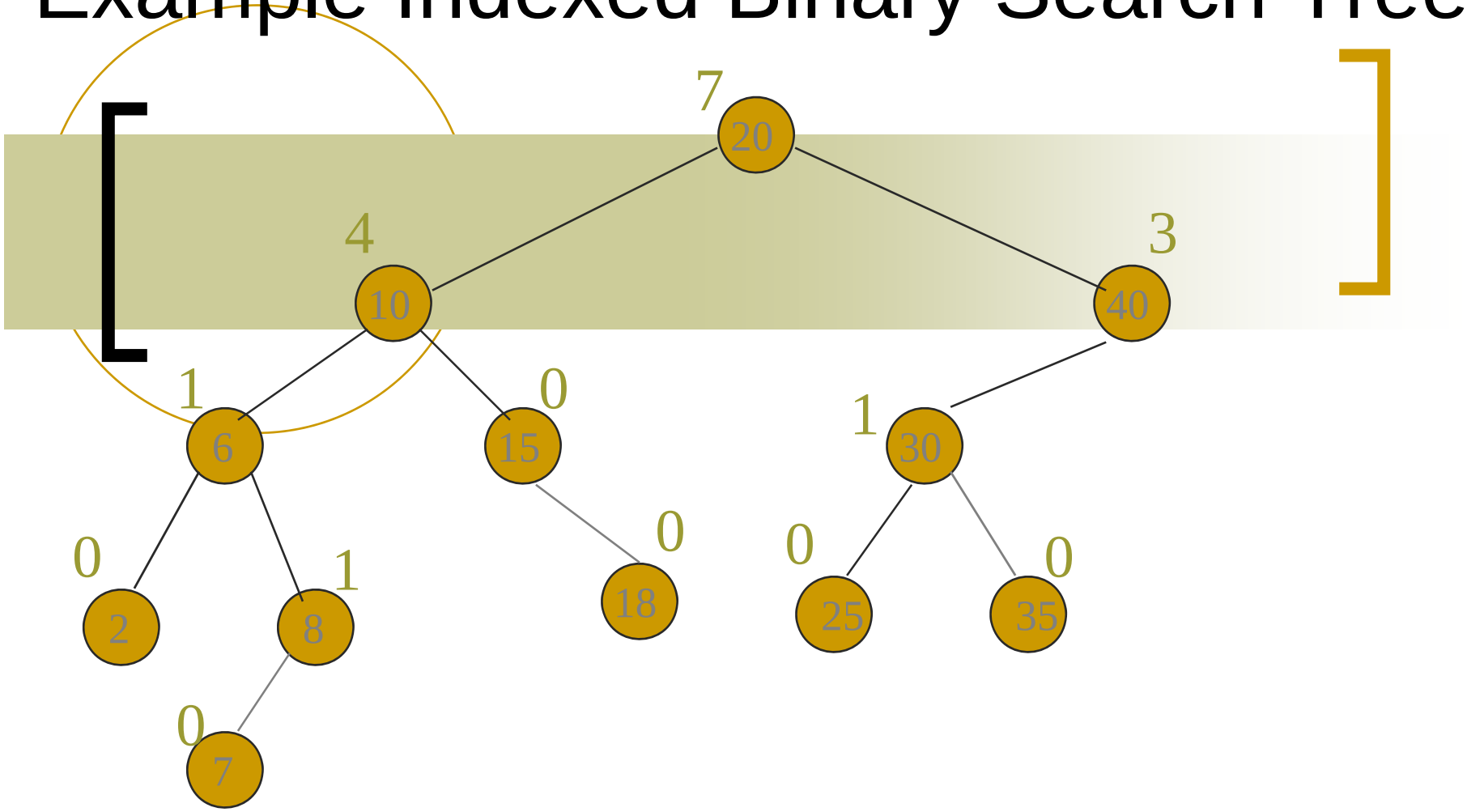


پیچیدگی = O(height).

Indexed Binary Search Tree

- جستجوی باینری
- در هر نود یک فیلد اضافی وجود دارد:
  - leftSize= شمارش تعداد نود های چپ درخت

Example Indexed Binary Search Tree



leftSize =red

leftSize And Rank

Rank=موقعیت عناصر در inorder

(inorder = مرتب کلید ها به صورت صعودی).

[2,6,7,8,10,15,18,20,25,30,35,40]

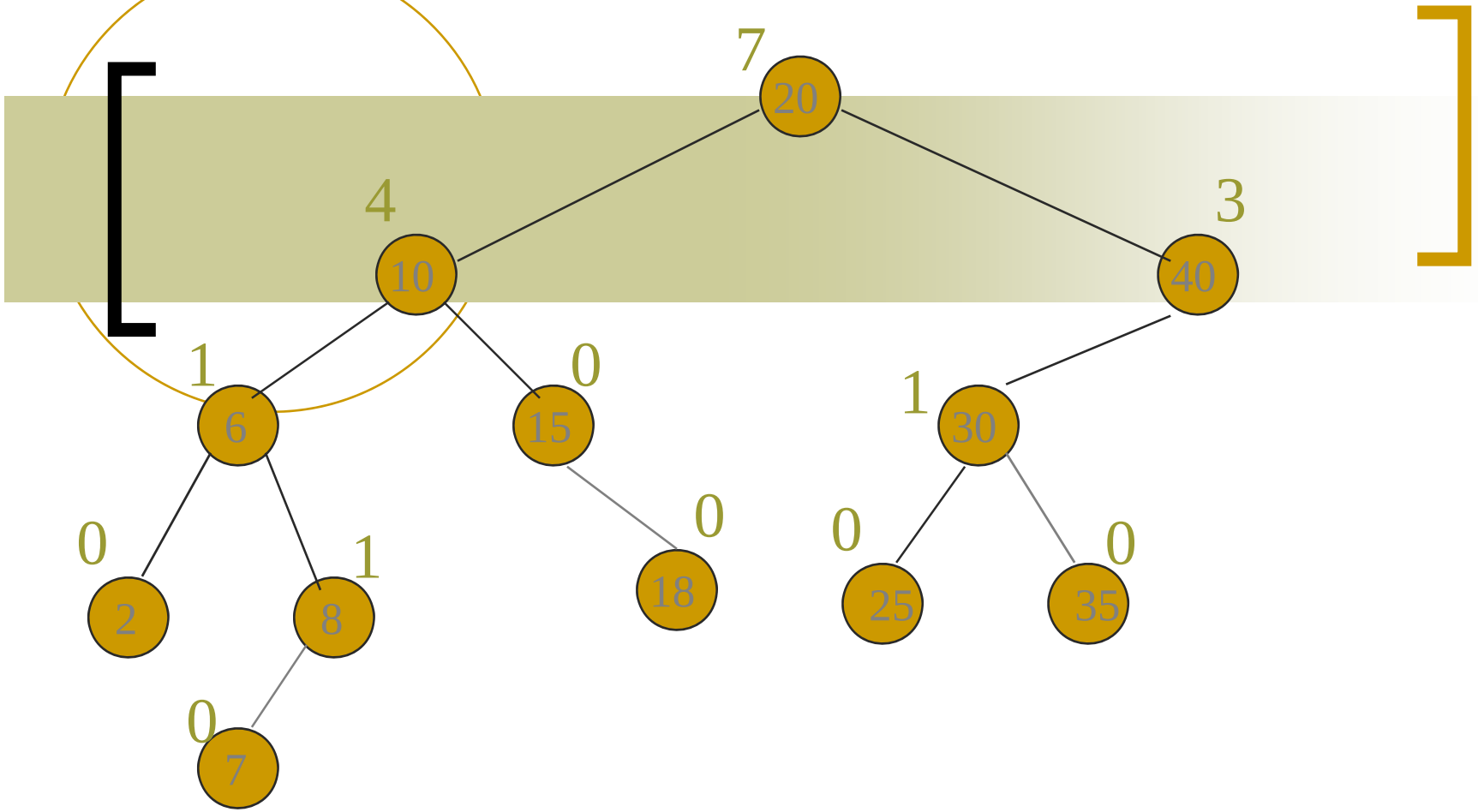
rank(2) = 0

rank(15) = 5

rank(20) = 7

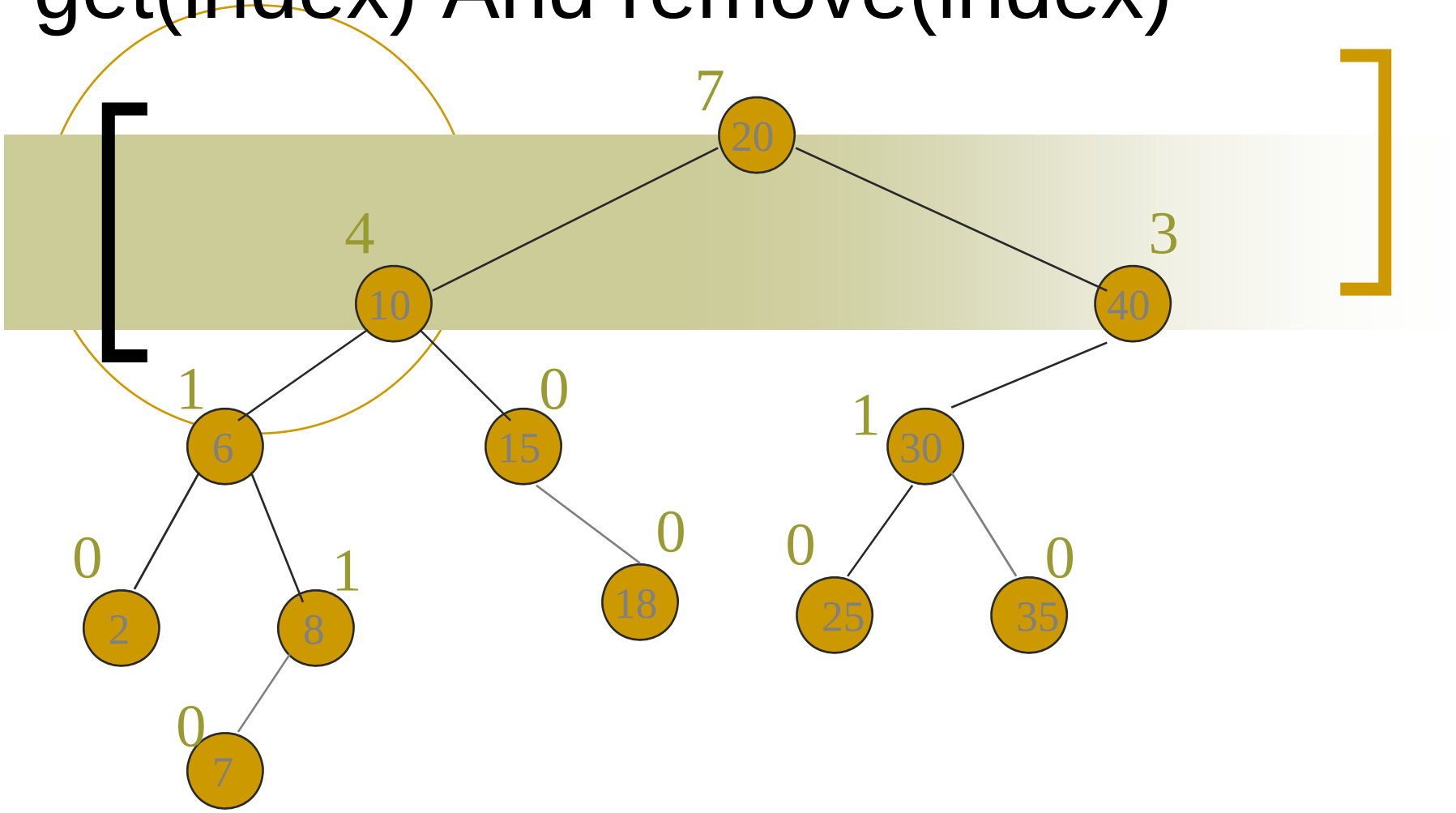
leftSize(x) = rank(x)

leftSize And Rank



sorted list = [2,6,7,8,10,15,18,20,25,30,35,40]

get(index) And remove(index)

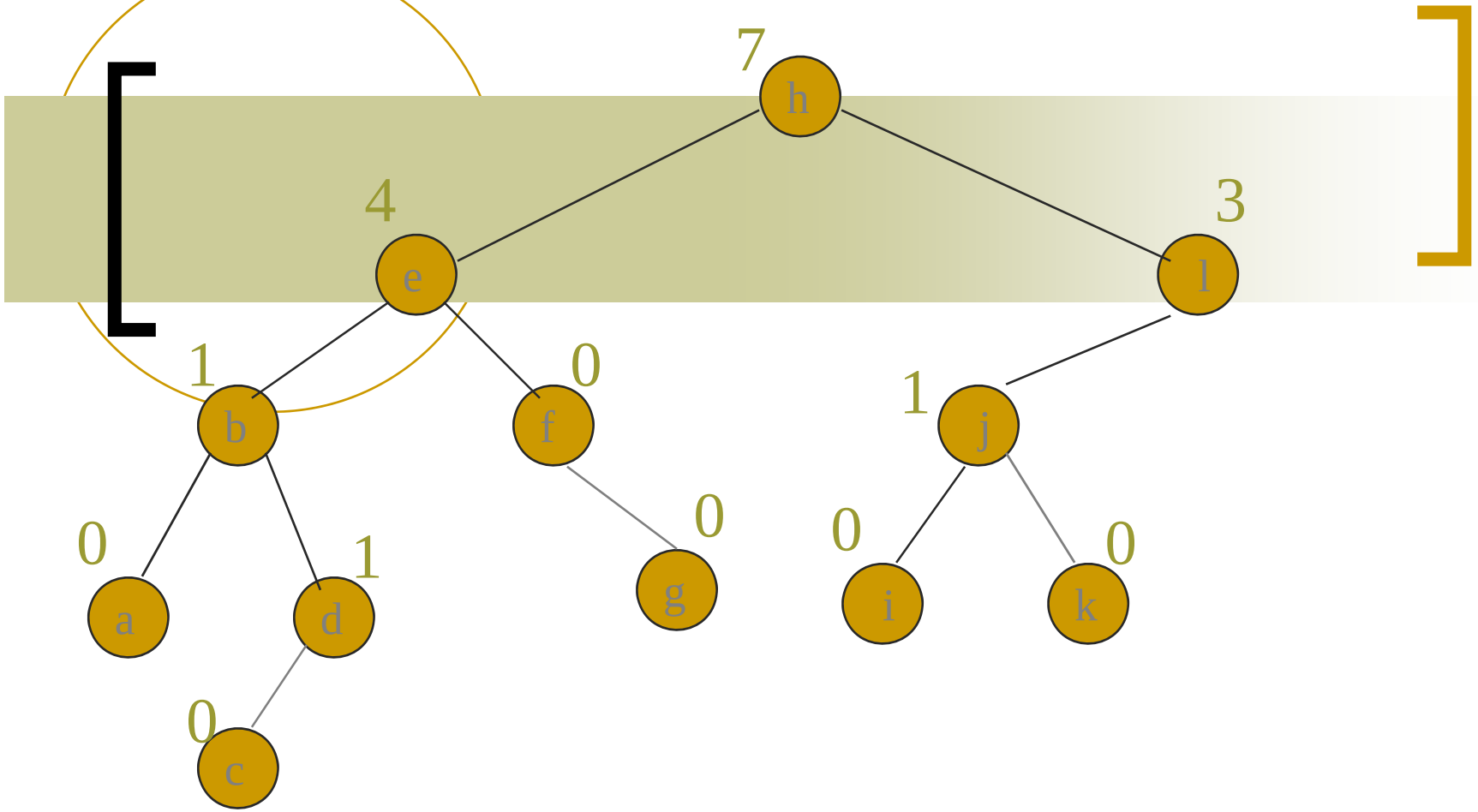


sorted list = [2,6,7,8,10,15,18,20,25,30,35,40]

get(index) And remove(index)

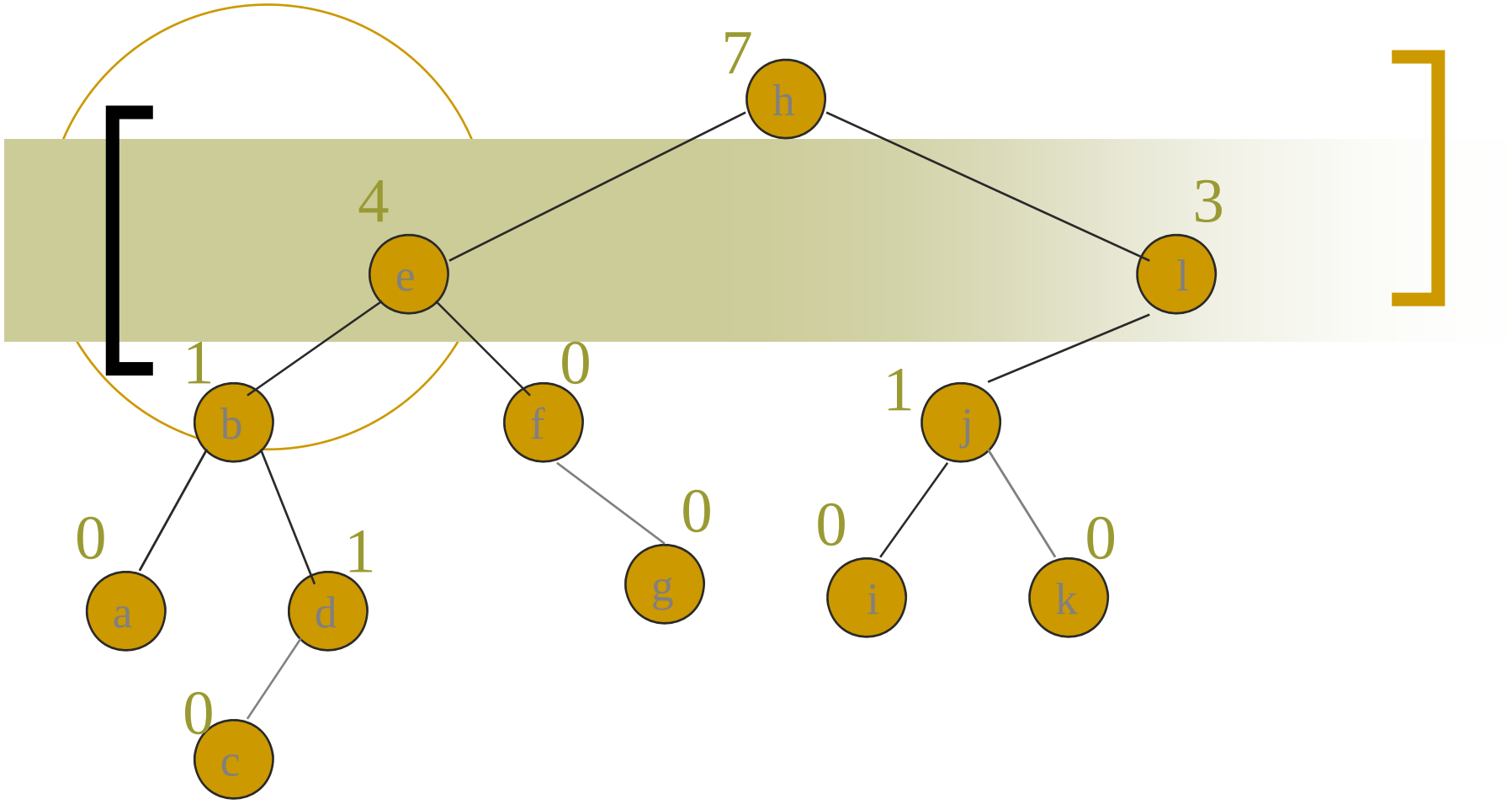
- if index = x.leftSize
  x.element را انتخاب کن .
- if index < x.leftSize
  عنصری انتخاب می شود که محتوای indexآن در زیر درخت چپ xمی باشد.
- if index > x.leftSize
  عنصری انتخاب می شودکه محتوای (index - x.leftSize-1) آن در زیر درخت راست xمی باشد.

Linear List As Indexed Binary Tree



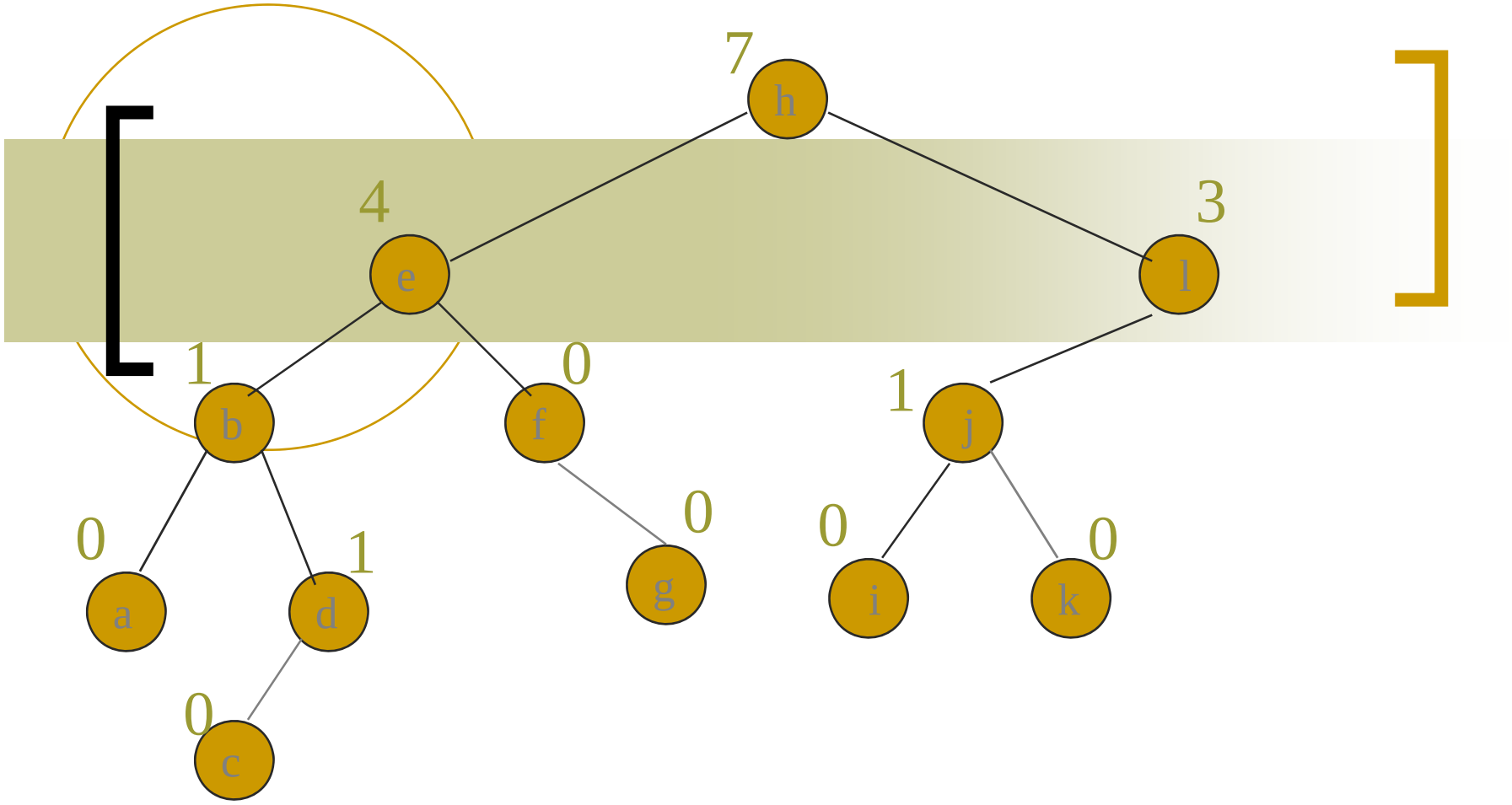
list = [a,b,c,d,e,f,g,h,i,j,k,l]

add(5,’m’)



list = [a,b,c,d,e,f,g,h,i,j,k,l]

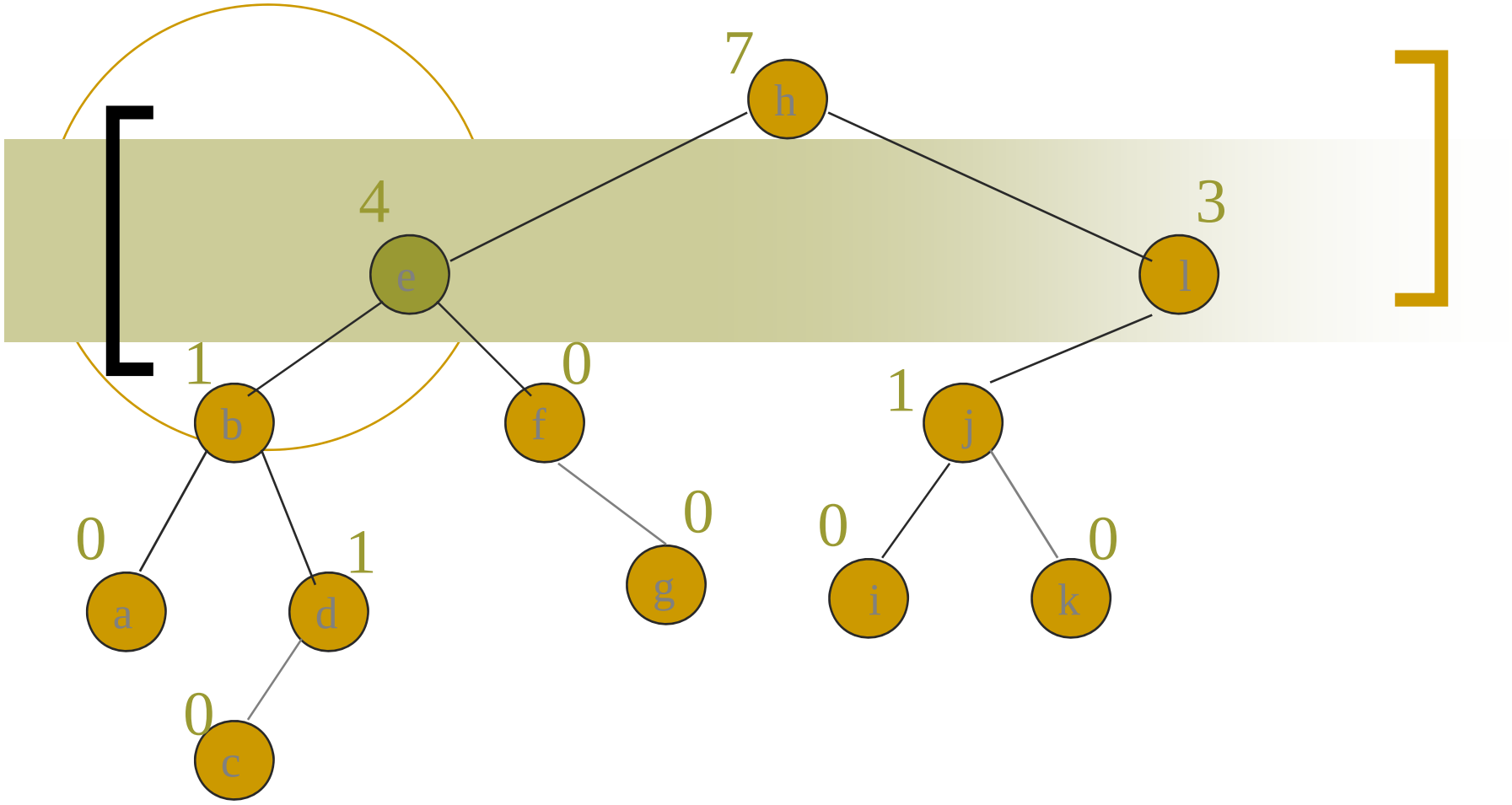
add(5,'m')



list = [a,b,c,d,e, m,f,g,h,i,j,k,l]

find node with element 4 (e)

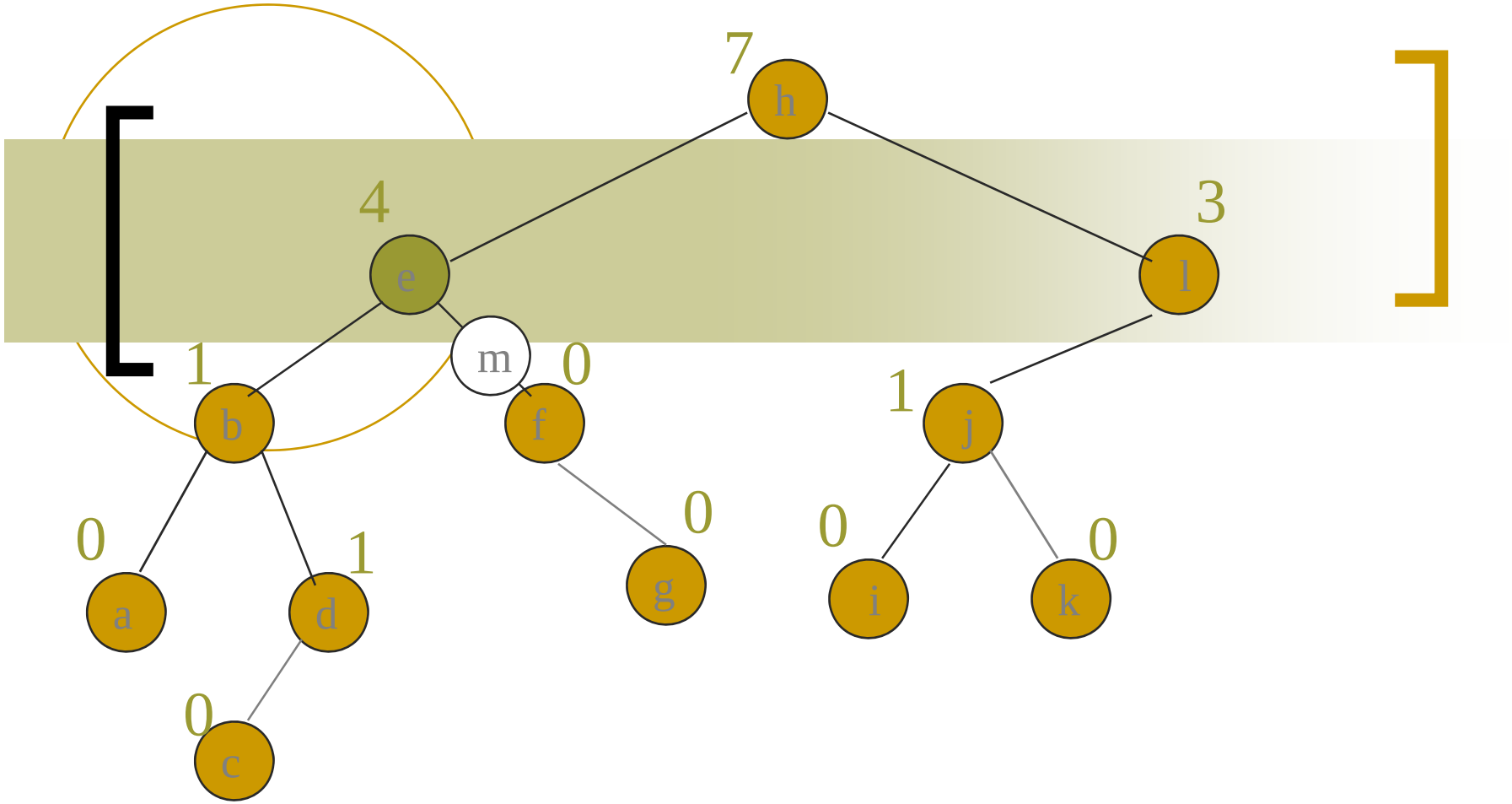
add(5,'m')



list = [a,b,c,d,e, m,f,g,h,i,j,k,l]

find node with element 4 (e)

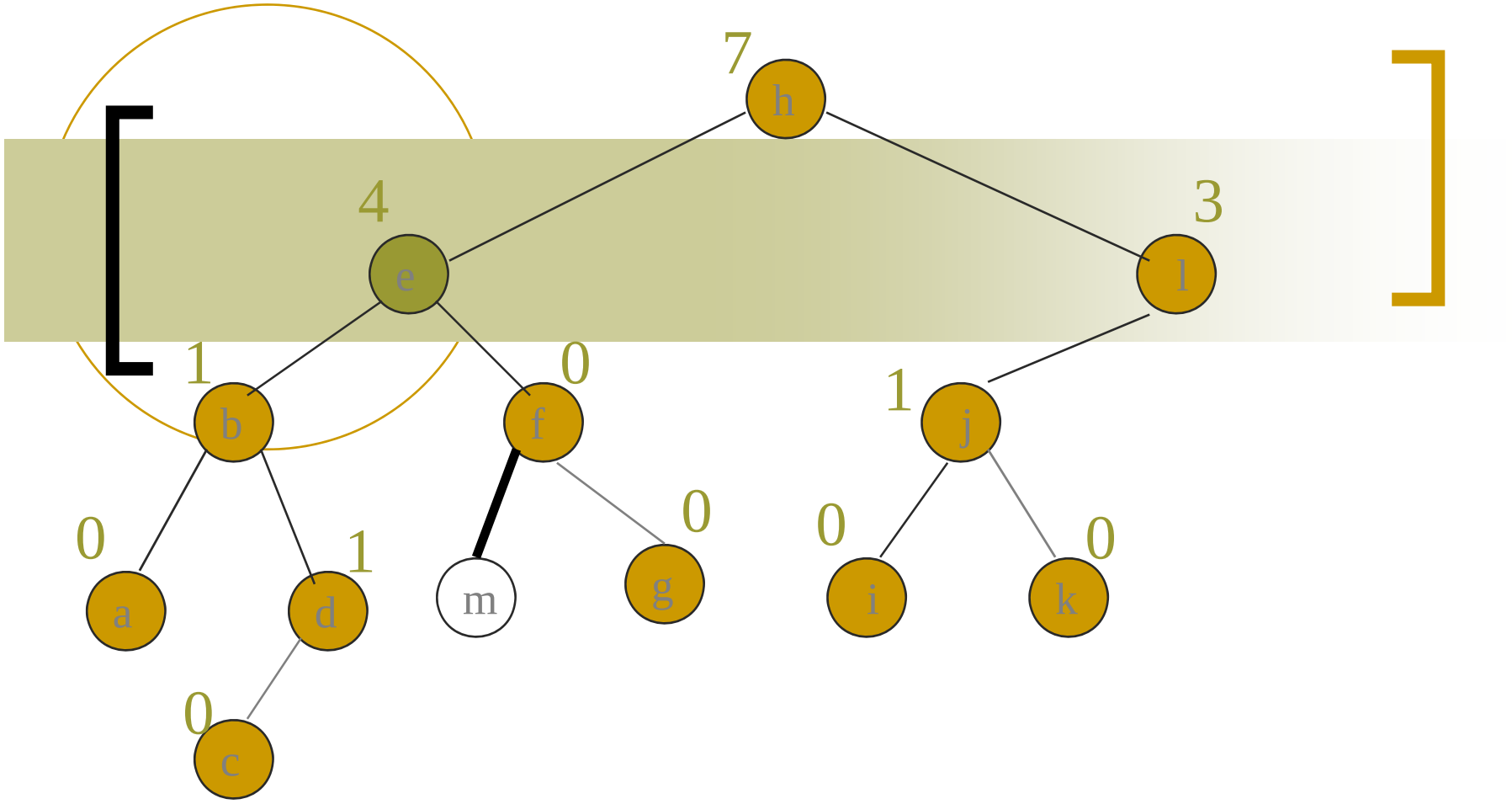
add(5,’m’)



m را به راست گره e اضافه کرده است

در نتیجه زیر درخت راست e به عنوان زیر درخت راست m قرار می گیرد.

add(5,’m’)



mرا بعنوان فرزند چپ در زیر درخت راست گره eقرار دهید.

add(5,’m’)

- پیچیدگی =O(height)

سایت مرجع دانشجوی پیام نور

- ✓ نمونه سوالات پیام نور : بیش از ۱۱۰ هزار نمونه سوال همراه با پاسخنامه تستی و تشریحی
- ✓ کتاب ، جزوه و خلاصه دروس
- ✓ برنامه امتحانات
- ✓ منابع و لیست دروس هر ترم
- ✓ دانلود کاملا رایگان بیش از ۱۴۰ هزار فایل مختص دانشجویان پیام نور