The Python List

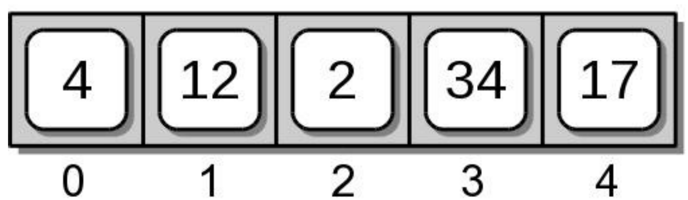
- A mutable sequence type container.
  - Provides operations for managing the collection.
  - Can grow and/or shrink as needed.
  - Implemented using an array.

List: Construction

- The Python list interface provides an abstraction to the actual underlying implementation.

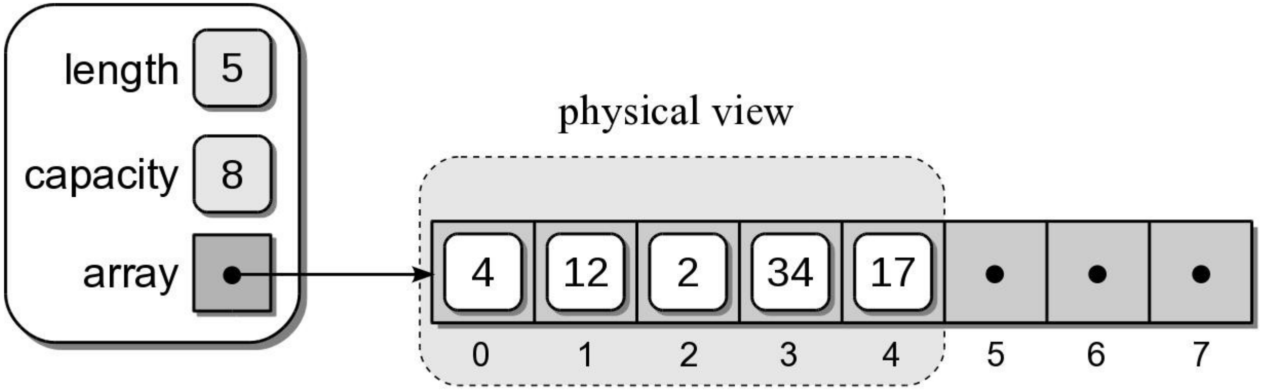
```
pyList = [ 4, 12, 2, 34, 17 ]
```

abstract view



List: Implementation

- An array is used to store the items of the list.
  - Created larger than needed.
  - **T**he items are stored in a contiguous subset of the array.

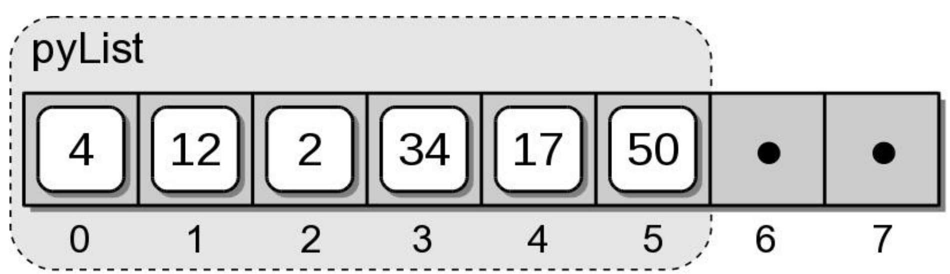


List: Appending an Item

- New items can be added at the end of the list.

```
pyList.append(50)
```

- When space is available, the item is stored in the next slot.

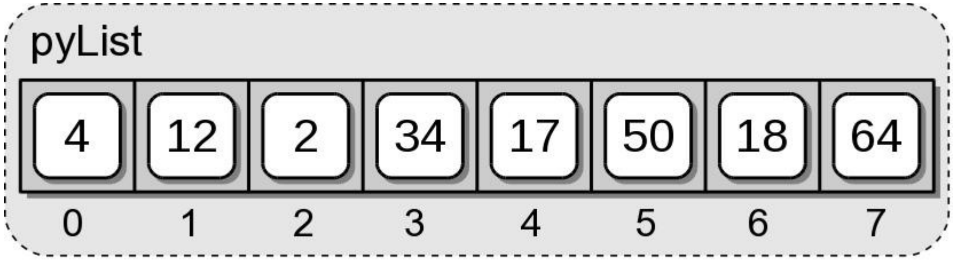


List: Appending an Item

- What happens when the array becomes full?

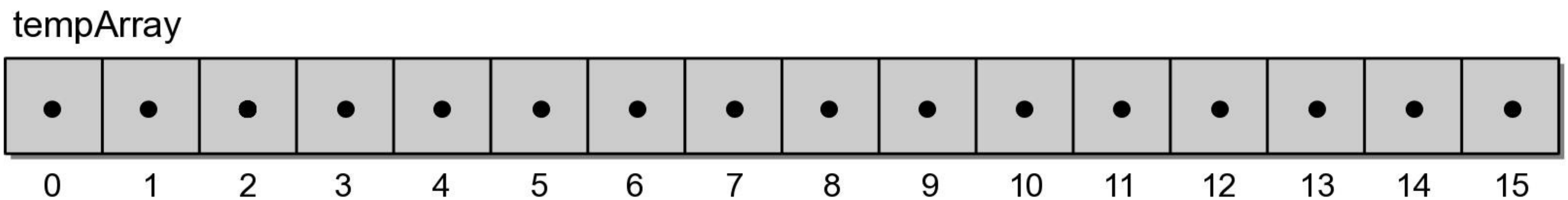
```
pyList.append(18)
pyList.append(64)
pyList.append(6)
```

- There is no space for value 6.

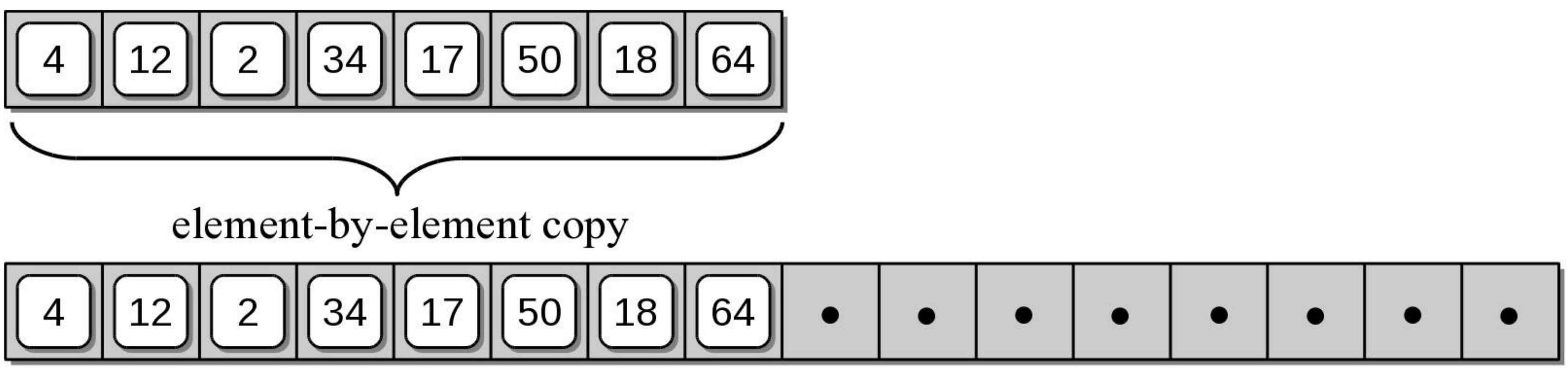


Expanding The Array

Step 1: create a new array, double the size.

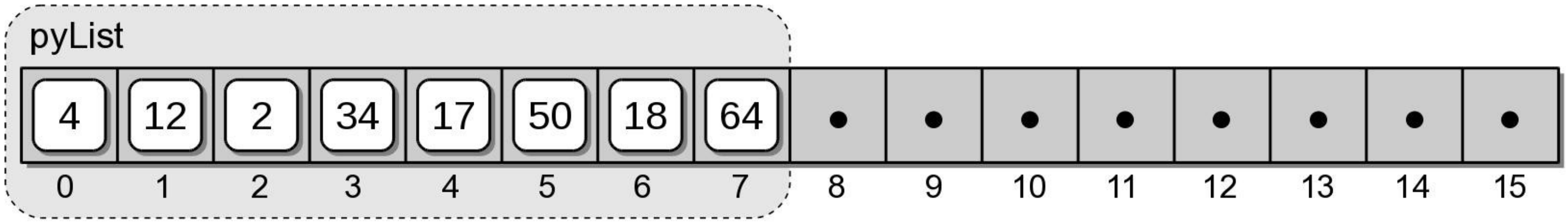


Step 2: copy the items from original array to the new array.

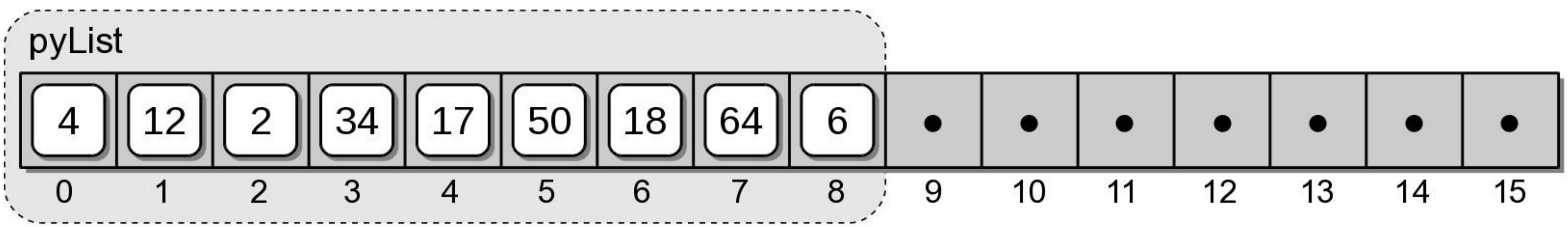


Expanding The Array

Step 3: replace the original array with the new array.



Step 4: store value 6 in the next slot of the new array.



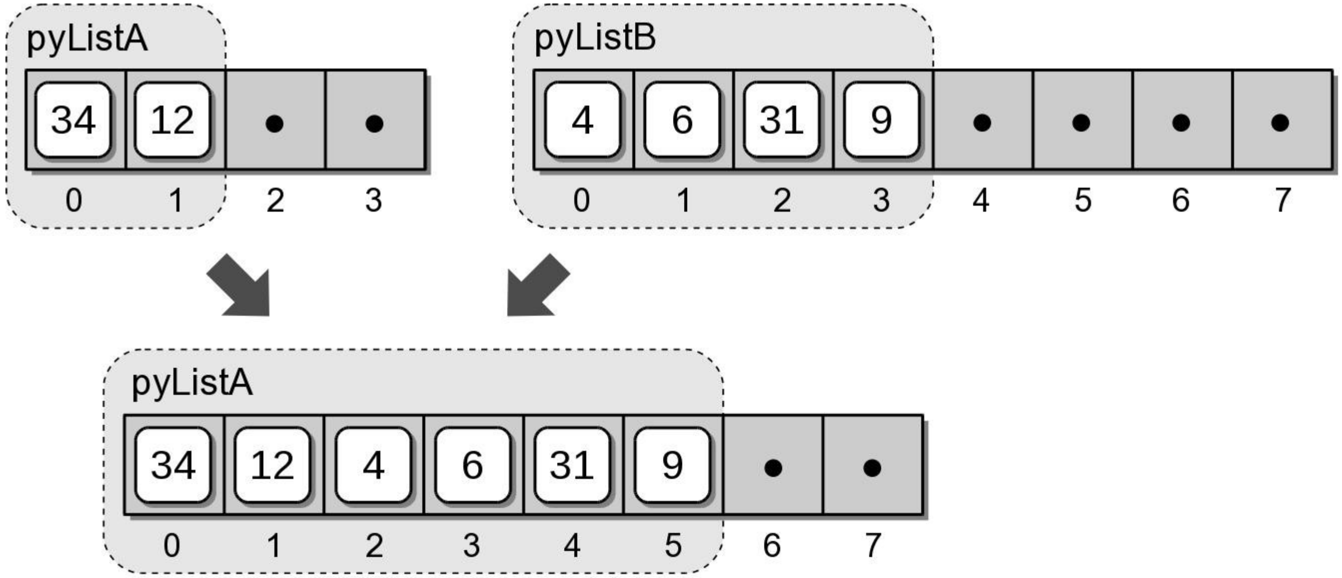
Big-O Analysis of Append

- Best Case Analysis?
- Worst Case Analysis?
- More later...

List: Extending

- The entire contents of a list can be appended to a second list.

```
pyListA = [34, 12]
pyListB = [4, 6, 31, 9]
pyListA.extend( pyListB )
```



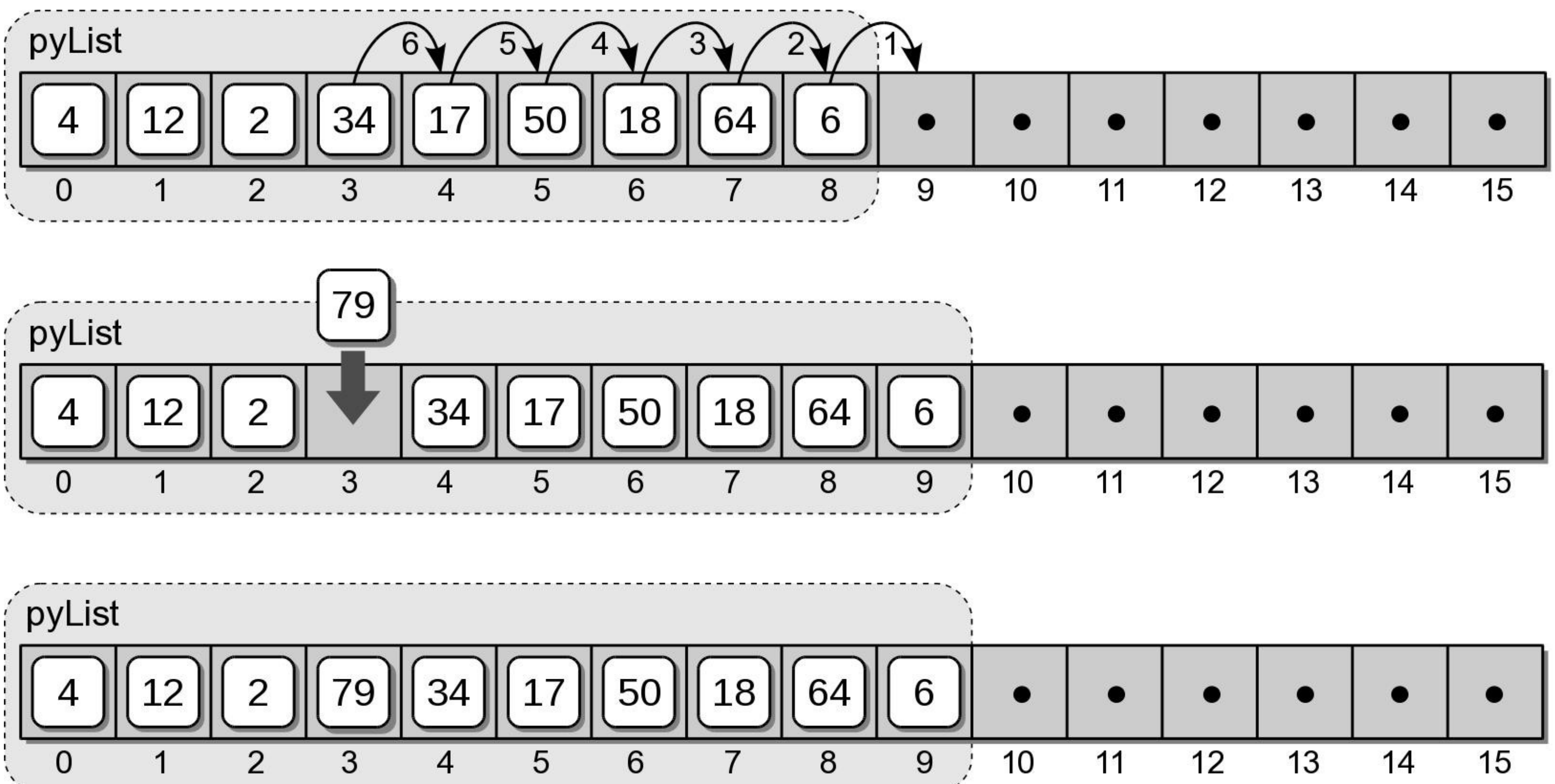
Big-O Analysis of Extend

- How should we measure input size?
- Best Case Analysis?
- Worst Case Analysis?

List: Inserting Items

- An item can be inserted anywhere within the list.

```
pyList.insert( 3, 79 )
```



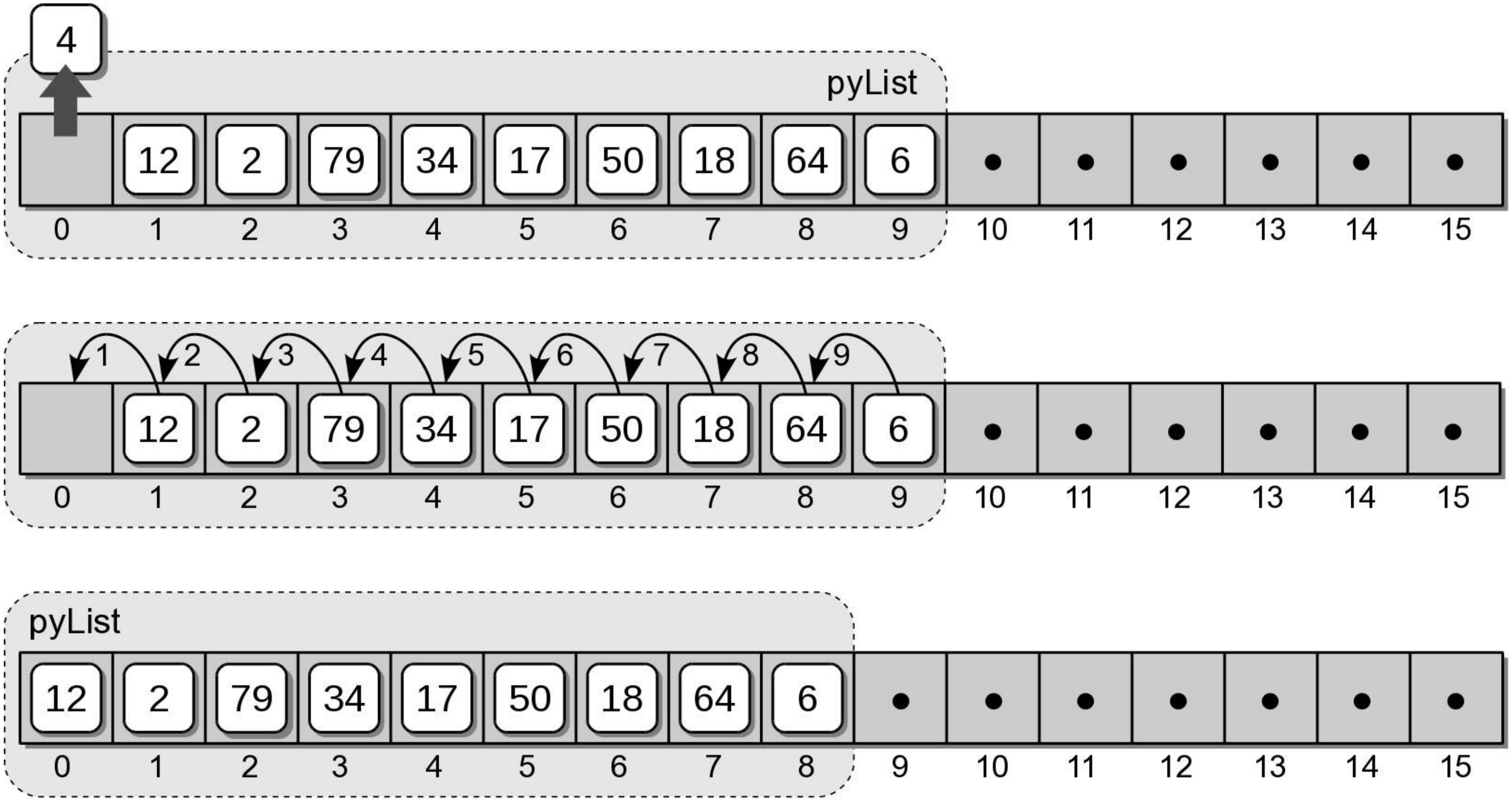
Big-O Analysis of Insert

- Best Case Analysis?
- Worst Case Analysis?

List: Removing Items

- An item can be removed from position of the list.

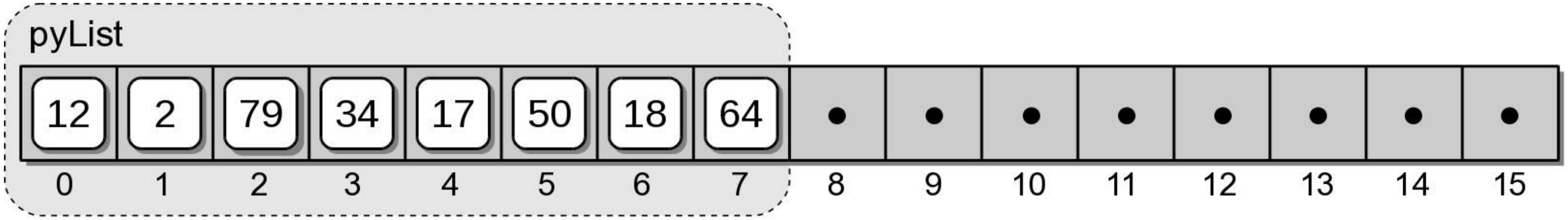
```
pyList.pop(0)
```



List: Removing Items

- Removing the last item in the list.

```
pyList.pop()
```



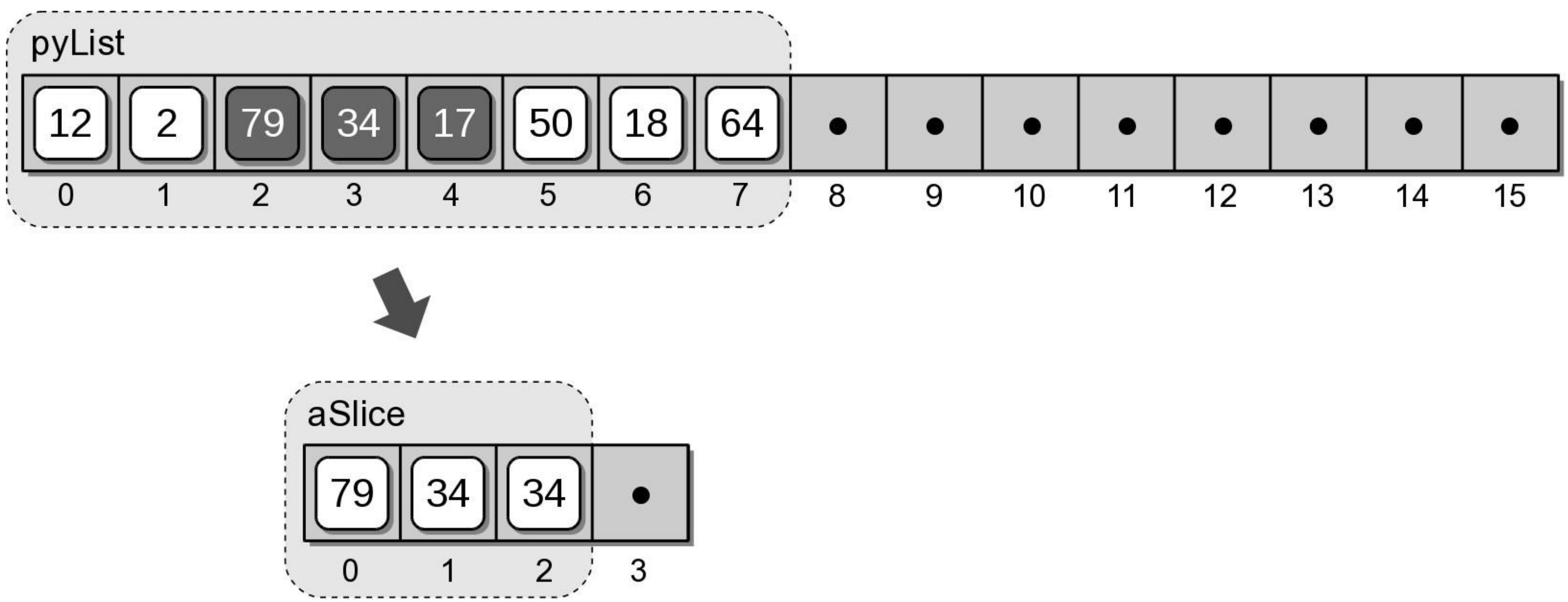
Big-O Analysis of Remove

- Best Case Analysis?
- Worst Case Analysis?

List: Slices

- Slicing a list creates a new list from a contiguous subset of elements.

```
aSlice = pyList[2:5]
```



Big-O Analysis of Slice

- How do we measure input size?
- Best Case Analysis?
- Worst Case Analysis?

Python List: Time-Complexities

List Operation	Worst Case
v = list()	?
len(v)	?
v = [0] * n	?
v[i] = x	?
v.append(x)	?
v.extend(w)	?
v.insert(x)	?
v.pop()	?
traversal	?

Python List: Time-Complexities

List Operation	Worst Case
v = list()	O(1)
len(v)	O(1)
v = [0] * n	O(n)
v[i] = x	O(1)
v.append(x)	O(n)
v.extend(w)	O(n) or O(n + m)
v.insert(x)	O(n)
v.pop()	O(n)
traversal	O(n)

Revisiting Analysis of Append

- Best Case: O(1)
- Worst Case: O(n)
- How would we analyze this:

```
for item in input:
    myList.append(i)
```

Amortized Analysis

- Compute the time-complexity by finding the **average cost** over a sequence of operations.
  - Cost per operation must be known.
  - Cost must vary, with
    - many ops contributing little cost.
    - only a few ops contributing high cost.

Append Example

i	s_i	e_i	Size	List Contents
1	1	-	1	[1]
2	1	1	2	[1][2]
3	1	2	4	[1][2][3][ ]
4	1	-	4	[1][2][3][4]
5	1	4	8	[1][2][3][4][5][ ][ ][ ]
6	1	-	8	[1][2][3][4][5][6][ ][ ]
7	1	-	8	[1][2][3][4][5][6][7][ ]
8	1	-	8	[1][2][3][4][5][6][7][8]
9	1	8	16	[1][2][3][4][5][6][7][8][9][ ][ ][ ][ ][ ][ ][ ]
10	1	-	16	[1][2][3][4][5][6][7][8][9][10][ ][ ][ ][ ][ ][ ]
11	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][ ][ ][ ][ ][ ]
12	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][12][ ][ ][ ][ ]
13	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][12][13][ ][ ][ ]
14	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][12][13][14][ ][ ]
15	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][ ]
16	1	-	16	[1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16]

Table 4.5: Using the aggregate method to compute the total run time for a sequence of 16 append operations.

Amoritized Cost of Append

- Total cost of resize operations:

$$\sum_{j=0}^{\lg n} 2^j \; < \; 2\text{n} \; \in \; O(n)$$

- Total cost of set operations: $\sum_{i=1}^{n} 1 = n$

- Average cost of n append operations:

$$\frac{\sum_{j=0}^{\lg n} 2^j + n}{n} \in O(1)$$