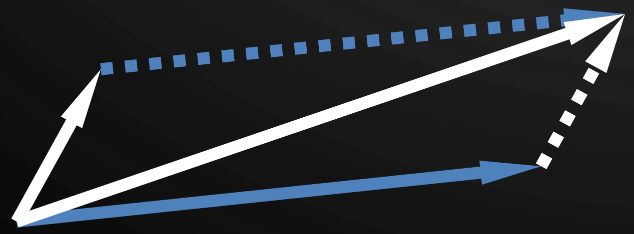
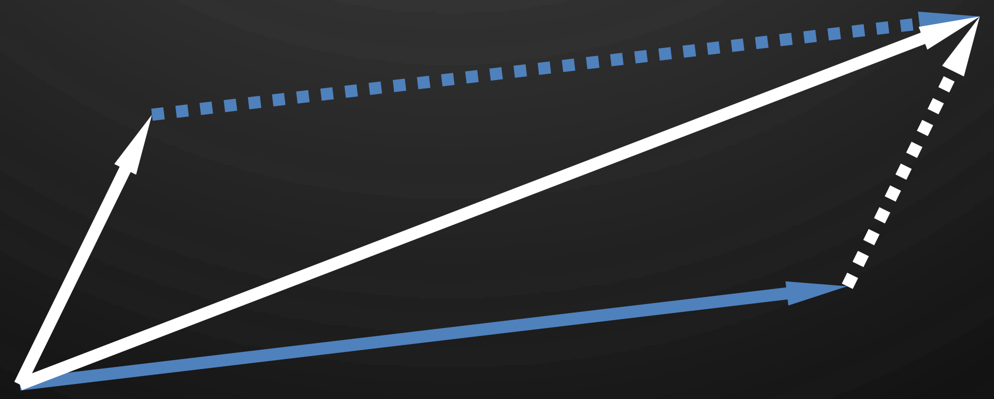
CH 6.
VECTORS, LISTS, AND SEQUENCES

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN C++, GOODRICH, TAMASSIA AND MOUNT (WILEY 2004) AND SLIDES FROM NANCY M. AMATO



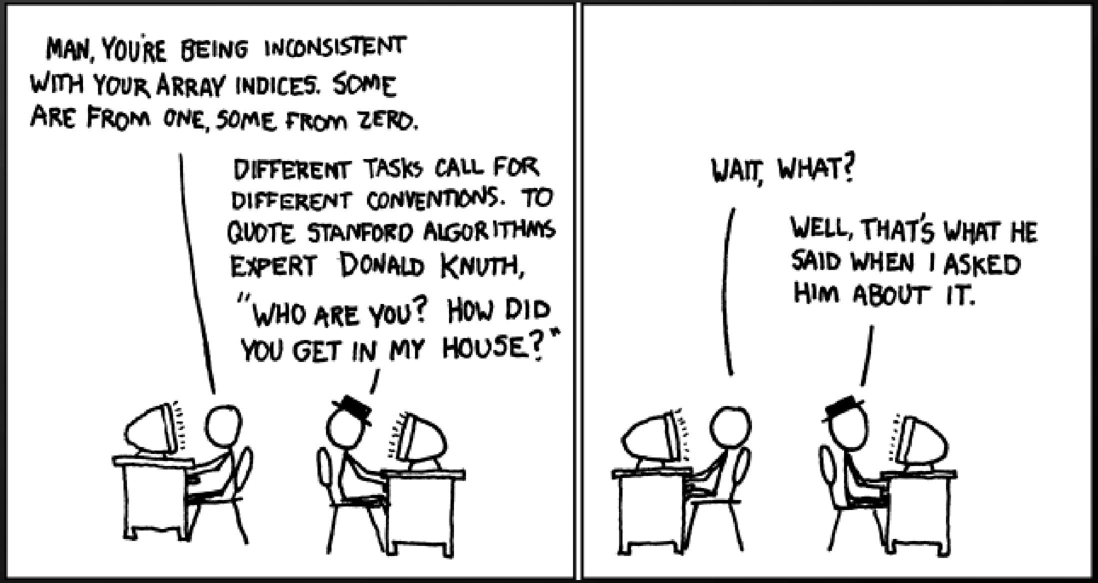
VECTORS

- The Vector ADT (Ch. 6.1.1)
- Array-based implementation (Ch. 6.1.2)



APPLICATIONS OF VECTORS

- Direct applications
  - Sorted collection of objects (elementary database)
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures



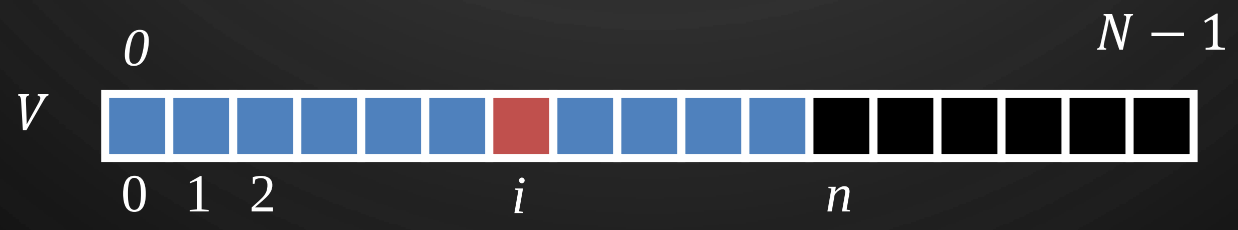
VECTOR ADT

- The Vector ADT extends the notion of array by storing a sequence of arbitrary objects
- An element can be accessed, inserted or removed by specifying its rank (number of elements preceding it)
- An exception is thrown if an incorrect rank is specified (e.g., a negative rank)

- Main vector operations:
  - $\text{at}(i)$: returns the element at index i
  - $\text{set}(i, e)$: replace the element at index i with e
  - $\text{insert}(i, e)$: insert a new element e to have index i
  - $\text{erase}(i)$: removes the element at index i
- Additional operations $\text{size}()$ and $\text{empty}()$

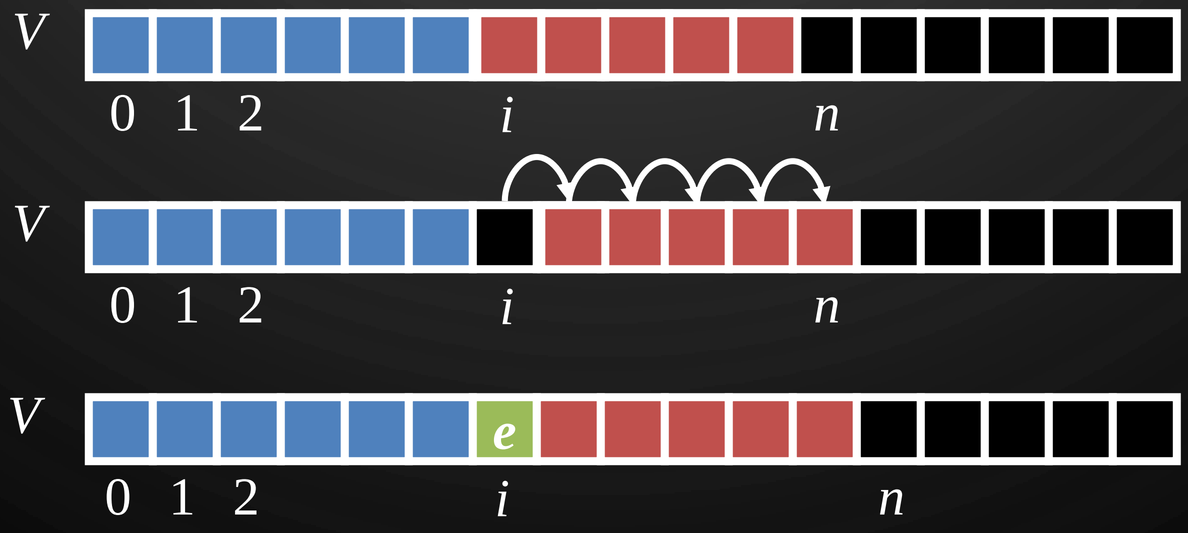
ARRAY-BASED VECTOR
STORAGE

- Use an array V of size N
- A variable n keeps track of the size of the vector (number of elements stored)
- $\text{at}(i)$ is implemented in $O(1)$ time by returning $V[i]$



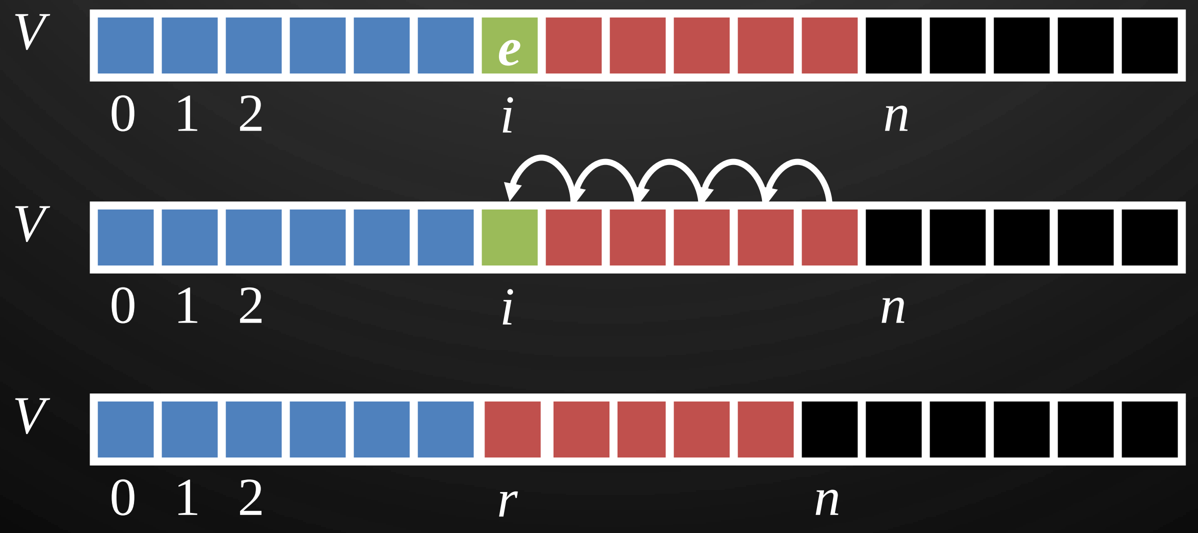
ARRAY-BASED VECTOR
INSERTION

- In $\text{insert}(i, e)$, we need to make room for the new element by shifting forward the $n - i$ elements $V[i], \ldots, V[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



ARRAY-BASED VECTOR
DELETION

- In $\text{erase}(i)$, we need to fill the hole left by the removed element by shifting backward the $\boldsymbol{n} - \boldsymbol{r} - 1$ elements $\boldsymbol{V}[\boldsymbol{r} + 1], \ldots, \boldsymbol{V}[\boldsymbol{n} - 1]$
- In the worst case ($\boldsymbol{r} = 0$), this takes $\boldsymbol{O}(\boldsymbol{n})$ time



PERFORMANCE

- In the array based implementation of a Vector
  - The space used by the data structure is $O(n)$
  - $\mathrm{size}()$, $\mathrm{empty}()$, $\mathrm{at}(i)$, and $\mathrm{set}(i, e)$ run in $O(1)$ time
  - $\mathrm{insert}(i, e)$, and $\mathrm{erase}(i)$ run in $O(n)$ time
- In an $\mathrm{insert}(i, e)$, when the array is full, instead of throwing an exception, we can replace the array with a larger one

EXERCISE:

- Implement the Deque ADT using Vector functions
  - Deque functions:
    - front(), back(), insertFront(e), insertBack(e), eraseFront(), eraseBack(), size(), empty()
  - Vector functions:
    - at(i), set(i, e), insert(i, e), erase(i), size(), empty()

EXERCISE SOLUTION:

Deque

- size() and empty()
- front()
- back()
- insertFront(e)
- insertBack(e)
- eraseFront()
- eraseBack()

Realization using Vector Functions

- size() and empty()
- at(0)
- $\text{at}(size() - 1)$
- $\text{insert}(0, e)$
- $\text{insert}(size(), e)$
- erase(0)
- $\text{erase}(size() - 1)$

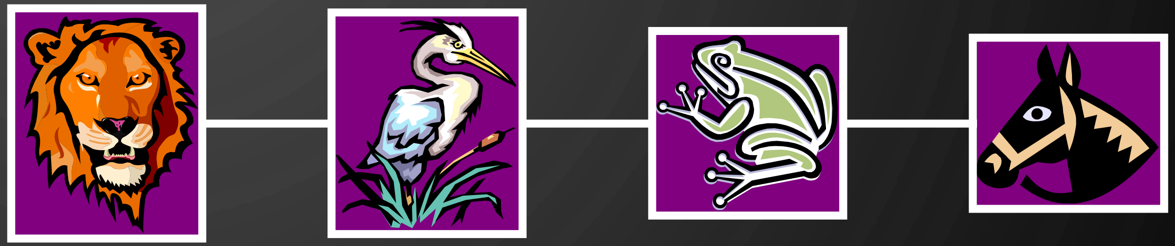
VECTOR SUMMARY

	Array Fixed-Size or Expandable	List Singly or Doubly Linked
$insert(i, e)$ and $erase(i)$	$O(1)$ Best Case ($i = n$) $O(n)$ Worst Case $O(n)$ Average Case	?
$at(i)$ and $set(i, e)$	$O(1)$	?
$size()$ and $empty()$	$O(1)$	?

ITERATORS AND POSITIONS

- An iterator abstracts the process of scanning through a collection of elements
- Can be implemented on most data structures in this course, e.g., vector and list
- Methods of the Iterator ADT:
  - hasNext() – returns whether another element follows
  - next() – returns iterator for next element
  - elem() – return element at position, also known as dereference in C++ (* operator)
- Iterators handle many operations in a uniform way
  - Example – insert for list and vector take iterators so the functions are called the same way
  - Traversal of data structure from begin() to end()

LISTS AND SEQUENCES

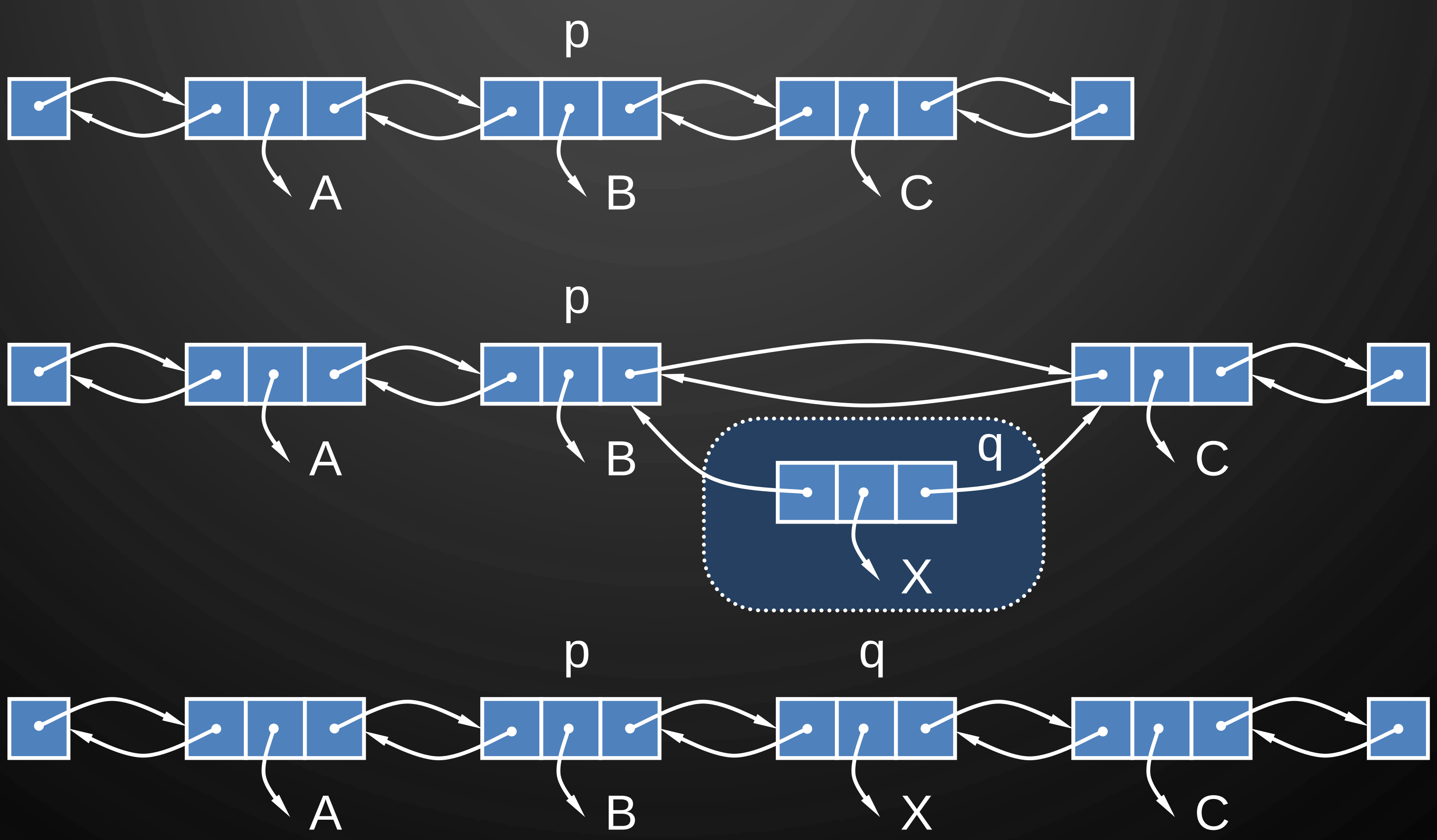


- Iterators (Ch. 6.2.1)
- List ADT (Ch. 6.2.2)
- Doubly linked list (Ch. 6.2.3)
- Sequence ADT (Ch. 6.3.1)
- Implementations of the sequence ADT (Ch. 6.3.2-3)

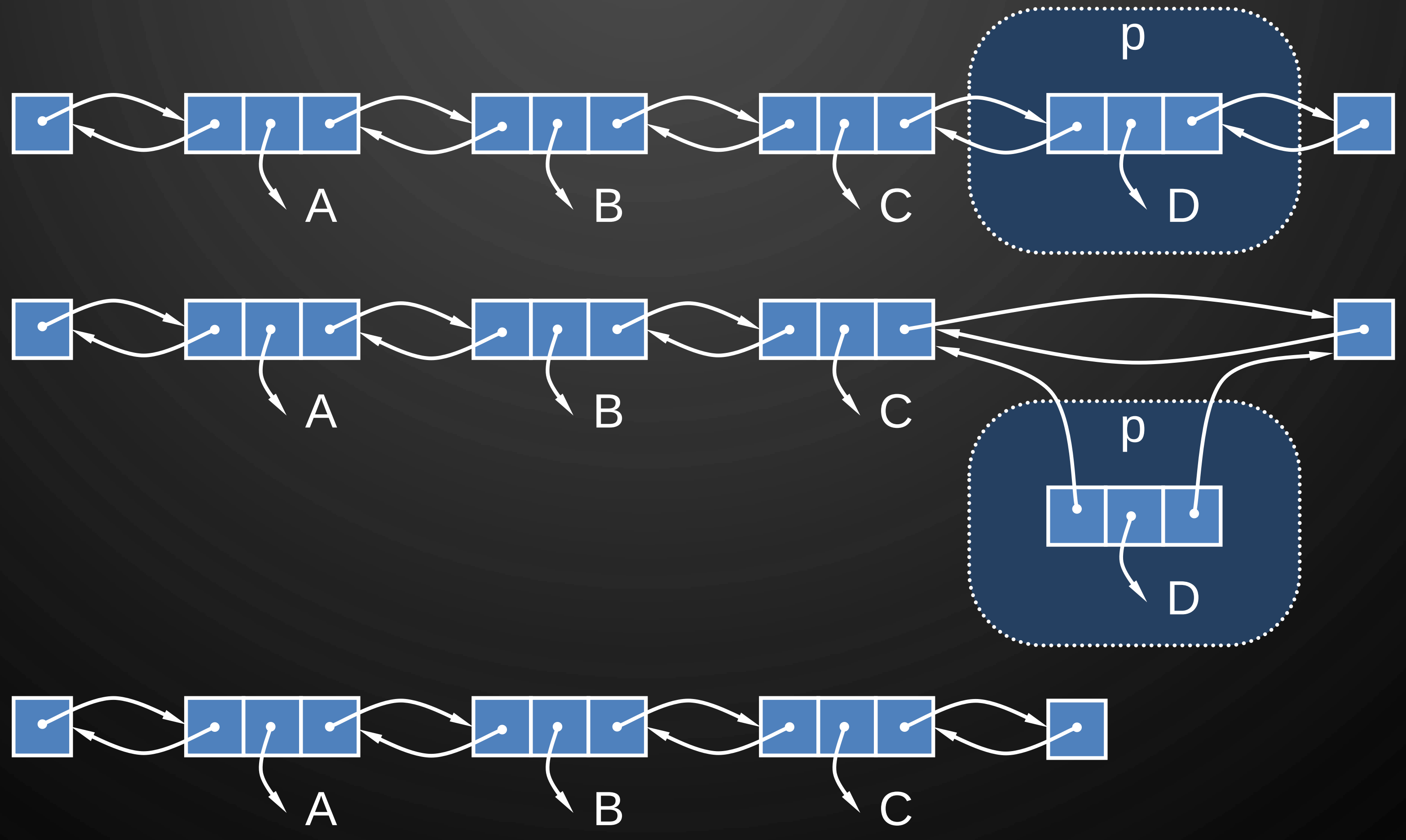
LIST ADT

- The List ADT models a sequence of **positions** storing arbitrary objects
  - establishes a before/after relation between positions
- It allows for insertion and removal in the "middle"
- Generic methods:
  - size() and empty()
- Accessor methods:
  - begin() and end()
- Update methods:
  - insertFront(e), insertBack(e), insert(p, e) – Note insert will insert e before iterator p
  - eraseFront(), eraseBack(), erase(p)

INSERT(p, e)



ERASE(p)



PERFORMANCE

- Assume doubly-linked list implementation of List ADT
  - The space used by a list with n elements is $O(n)$
  - The space used by each iterator of the list is $O(1)$
  - All the operations of the List ADT run in $O(1)$ time

LIST SUMMARY

	List Singly-Linked	List Doubly- Linked
begin(), end(), insertFront(), insertBack(), eraseFront()	$O(1)$	$O(1)$
insert(p, e), eraseBack(), erase()	$O(n)$ Worst and Average case $O(1)$ Best case	$O(1)$
size() and empty()	$O(1)$	$O(1)$

SEQUENCE ADT

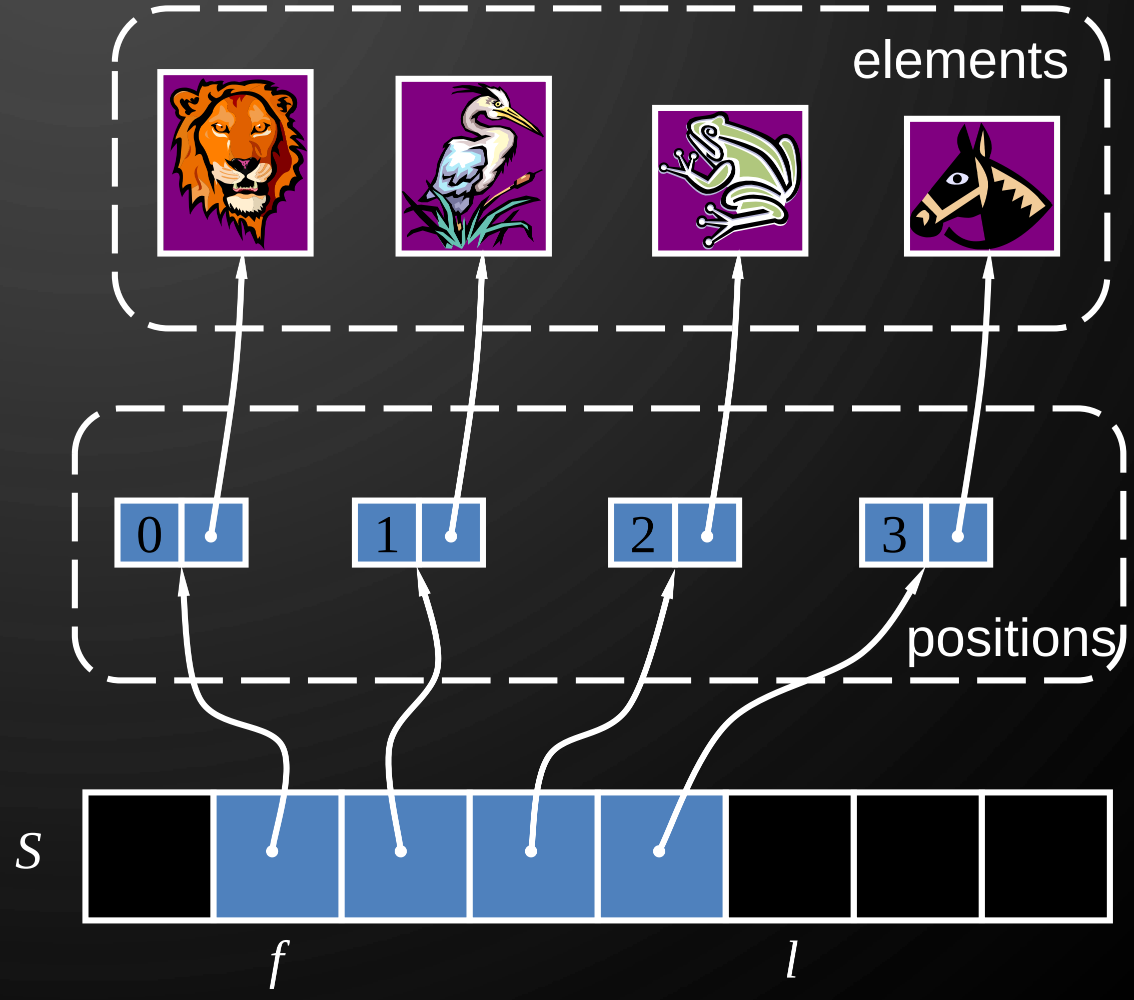
- The Sequence ADT is a combination of the Vector and List ADTs
- Elements accessed by
  - Index or
  - Iterator (Position)
- All items in the List ADT plus the following “bridging” functions:
  - $\text{atIndex}(i)$ – returns position of element at index i
  - $\text{indexOf}(p)$ – returns index of element at position p

APPLICATIONS OF SEQUENCES

- The Sequence ADT is a basic, general-purpose, data structure for storing an ordered collection of elements
- Direct applications:
  - Generic replacement for stack, queue, vector, or list
  - Small database (e.g., address book)
- Indirect applications:
  - Building block of more complex data structures

ARRAY-BASED IMPLEMENTATION

- We use a circular array storing positions
- A position object stores:
  - Element
  - Index
- Indices f and l keep track of first and last positions



SEQUENCE IMPLEMENTATIONS

	Circular Array	List Doubly- Linked
$size(), empty(), begin(), end(), insertFront(), insertBack()$	$O(1)$	$O(1)$
$atIndex(i)$ and $indexOf(p)$	$O(1)$	$O(n)$
$insert(p, e)$ and $erase(p)$	$O(n)$	$O(1)$

INTERVIEW QUESTION 1

- Write code to partition a list around a value x, such that all nodes less than x come before all nodes greater than or equal to x.

GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.

INTERVIEW QUESTION 2

- Implement a function to check if a list is a palindrome.

GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.