So far, we have written and analysed algorithms using arrays.

An array is a kind of a "Data structure".

Did we make any assumptions about the design of the "array" data structure in analysing the algorithms? If so, what?

1) Every access takes same time

2) Size is known

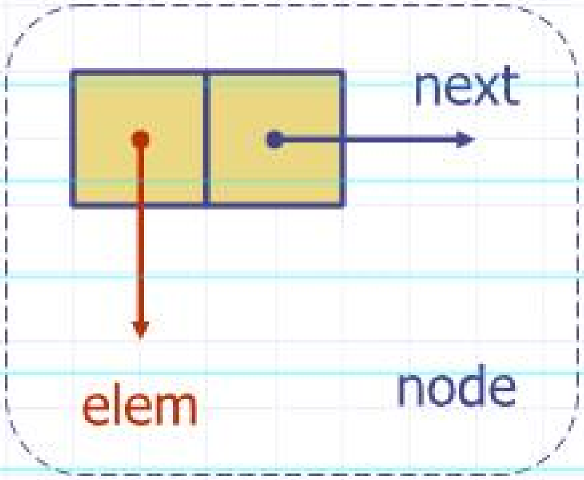
3) Address of first element is known
 . and can easily seek the ith element

The Node Class

```
public class   Node           {
   // Instance variables:
✓  private  Object element;
✓  private  Node next;
   /** Creates a node with null references to its element and next node. */
   public  Node()             {
     this(null,  null);
   }
   /** Creates a node with the given element and next node. */
   public  Node(Object e,  Node n)  {
      element  =  e;
      next  =  n;
   }
   // Accessor methods:
   public  Object getElement()  {
     return  element;
   }
   public  Node getNext()  {
     return  next;
   }
   // Modifier methods:
   public void  setElement(Object newElem)  {
      element  =  newElem;
   }
   public void  setNext(Node newNext)  {
      next  =  newNext;
   }
}
```

- Each node stores
  - element
  - link to the next node

Q: Uses of node?
Q: Modifications of node for more uses?



① Write down Qs for clarification.
② Write down Questions & Points for exploration (what uses can I make of node)

Pre-requirement

Structure, and objects, Pointers, Understanding of Array -Random Access, time-memory trade-off. *Exposure to different kinds of operations and tasks.

CS101 [Rubik cube, scientific calculator, connect 4]

1) What: (for the node class)

- *purpose, 'this' method, (data) type of element that can be stored in a node, what kind of nodes (parent/next) can a node link to making it different from array, can node element be changed, empty node possible, accessing a node from a collection, defining a node, way of referencing a node without referencing a collection, next of a node stores element of next node or its address, size limitation, can two nodes hold address of a common node, address (next) is relative or direct, is node size fixed, can node point to two different elements at the same time, what happens if a node gets `corrupt'?*

2) How:

- *Exposure to different kinds of operations and tasks such as fast (constant time) insertion and deletion which arrays cannot provide.*

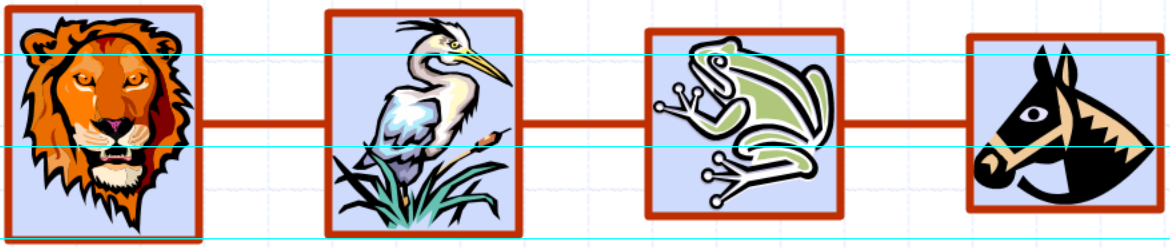
Exploratory Questions/Topics.

1) DS: Different ways of connecting nodes: flow charts, trees, "circular trees", Groups, ordered, based on element, non-contiguous or contiguous memory blocks, previous/next node, "contiguous" linked list, use collection as array, chain of nodes vs. array?, cyclic list of nodes possible, meshes, circular "queue" (by connecting last node to the first), social network graph
2) READ: Access time to other nodes (slowed down), no predefined length/size, 2 units to store a single node should it not make it inefficient?, can backtrack if "prev" link is defined,
3) WRITE: low time complexity for insertion, deletion (especially with an `expanded' node), unwanted elements removal, modifications are easy (?), Merging, splitting, other operations, direct link to an element, sorting is easier than in array -- change links instead of values,
4) EXPANDING NODE: Modifying the node DS: multiple quantities stored, multiple (100s of) pointers stored, node type can change at any node in the structure, along with element and next also store relation with another node in order to allow creation of binary and B+ trees, add more methods, graphs, inheritance from a node possible to give "multi-nodes", "previous" address, methods to modify/read extra pointers,

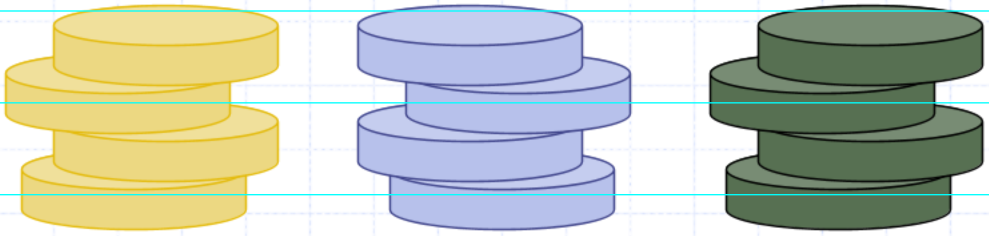
Abstract Data Types (ADTs)

- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- Example: ADT modeling a simple stock trading system
  - The data stored are buy/sell orders
  - The operations supported are
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - Error conditions:
    - Buy/sell a nonexistent stock
    - Cancel a nonexistent order

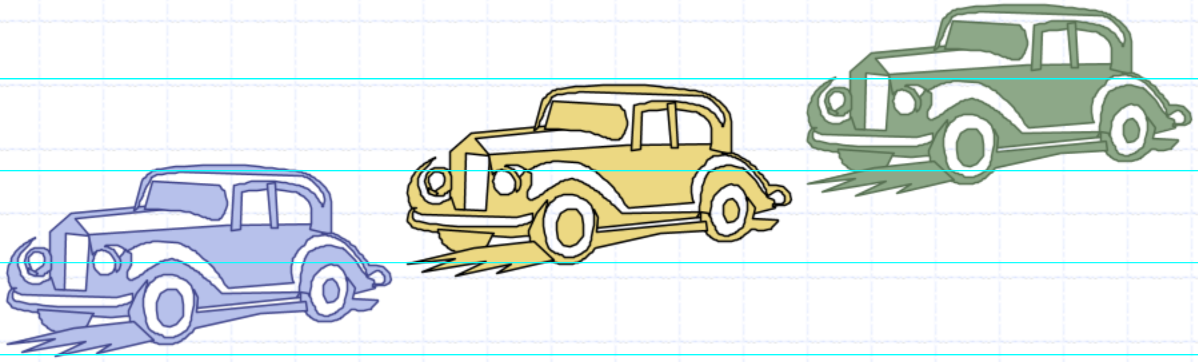
Linked Lists



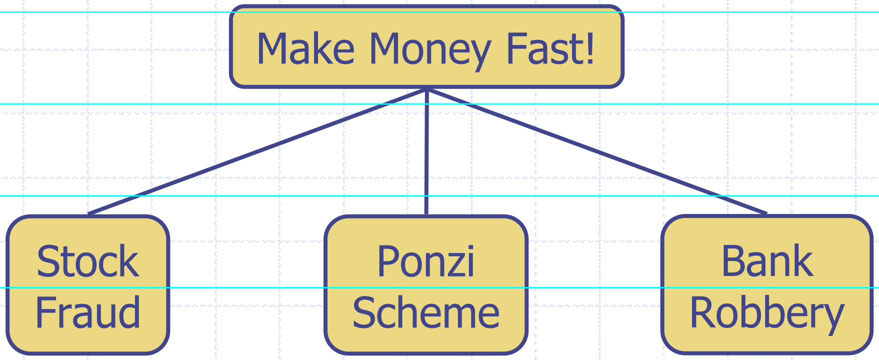
Stacks [LIFO]



Queues [FIFO]

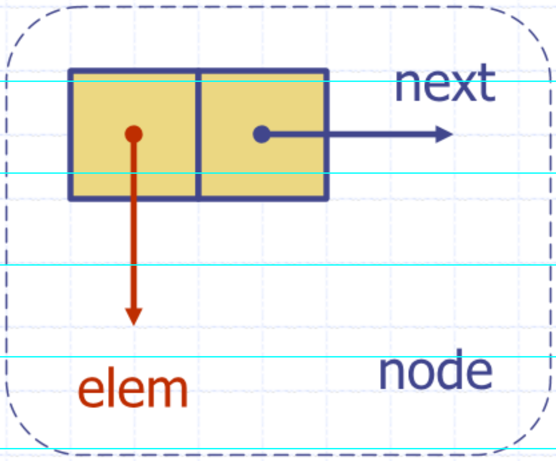


Trees



Singly Linked List (§ 4.4.1)

- A singly linked list is a concrete data structure consisting of a sequence of nodes
- Each node stores
  - element
  - link to the next node


next
elem
node