SIT221: Data Structures and Algorithms

Lecture 3: Linked Lists

Updates

- Prac Recordings: available on Bb
- Workbook has been uploaded
- Week 2 prac solution has been uploaded
- Assignment 1 has been released and due **11:59pm Friday, Aug 11th**

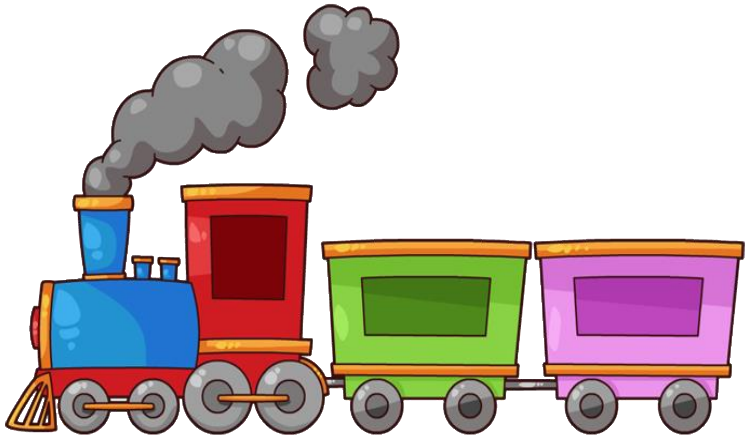
Lecture 2 Recap

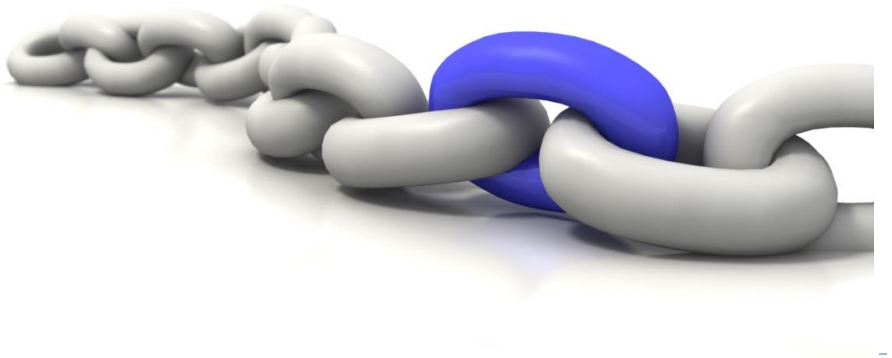
- Sorting
  - Bubble Sort
  - Insertion Sort
  - Selection Sort
  - Merge Sort
  - Quick Sort
- Search
  - Linear Search
  - Binary Search

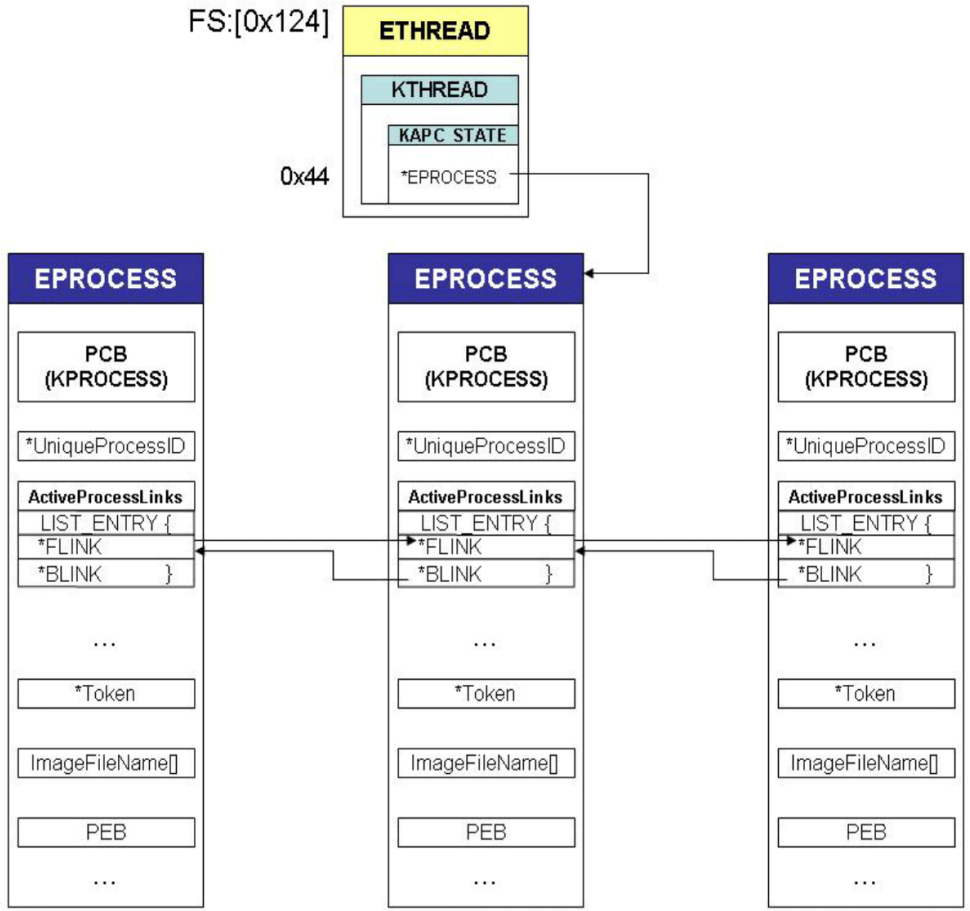
Lecture 2 Recap (2) – Practical

- Vector class
  - Sort()
  - Sort(Icomparer<T> comparer)
- Part class – IComparable<Part>
- Icomparer Interface
- Demo

Lecture 3 Linked List







Lists – Motivating scenario

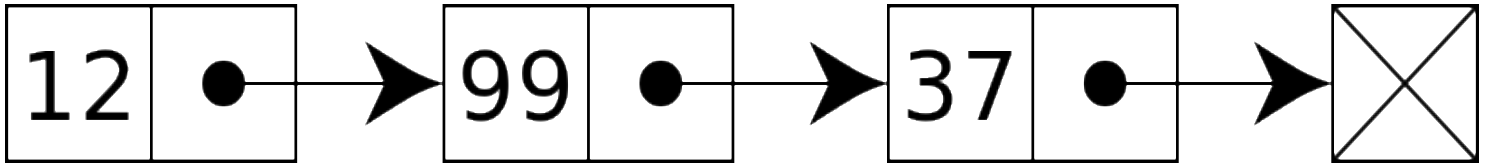
- How can you store a list of elements [integers, strings, parts, …]
  - Arrays?
  - What if I want to add element at the begining or somewhere in the middle?
  - What if you have millions of items?
  - What if I need to add element at the end of the array?
  - Do you have enough space? What if you do not?

What is wrong with Arrays?

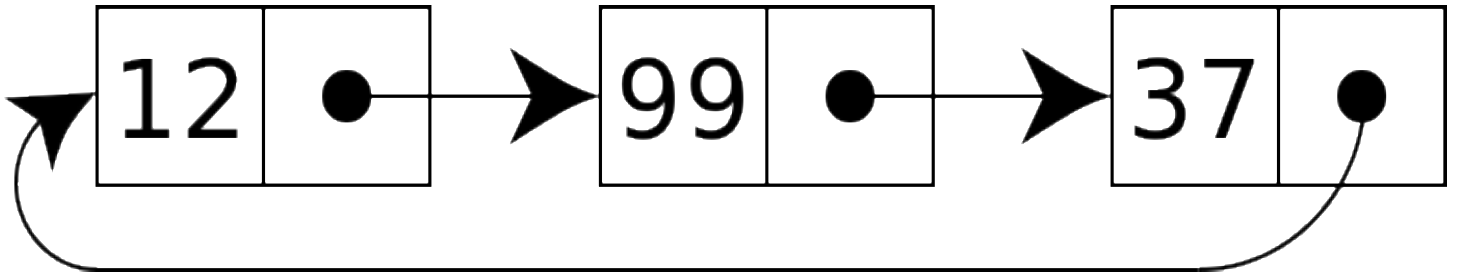
- Major problems with arrays:
  - Often need to shuffle elements when inserting or deleting an element
  - Cannot change size, can only reallocate and copy all elements from old to new array.
  - Or you pre-allocate and waste extra space.
- Linked lists solve these problems
  - However they introduce new problems!

Linked Lists – A recursive data structure

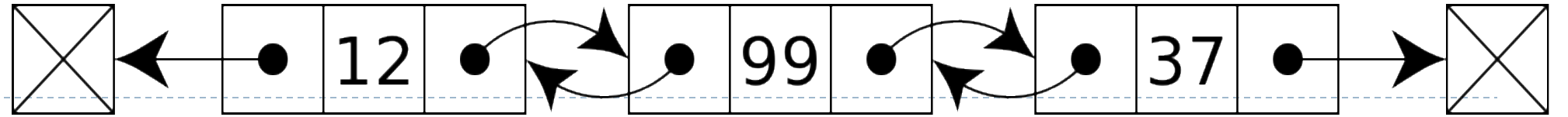
Singly linked lists



Circular linked lists

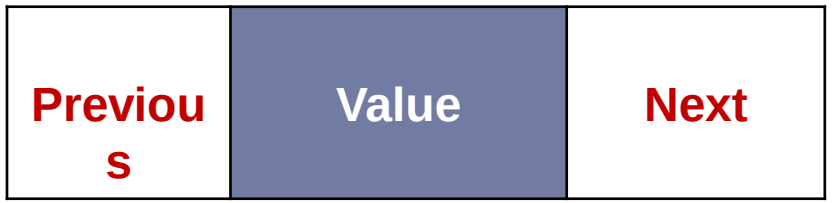


Doubly linked lists

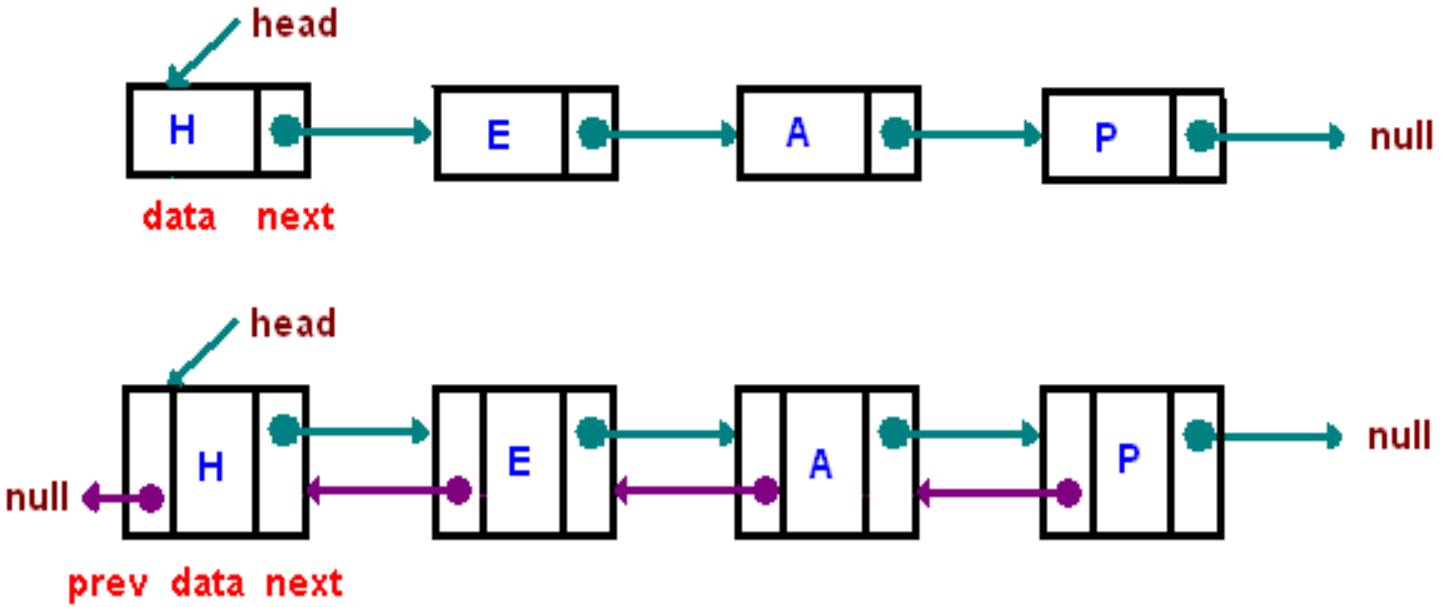


What are the building blocks of the linked list

- Node – what should we have hear? Value?
- Links – why do we need these? How many?
- Linked List Head – how this works?

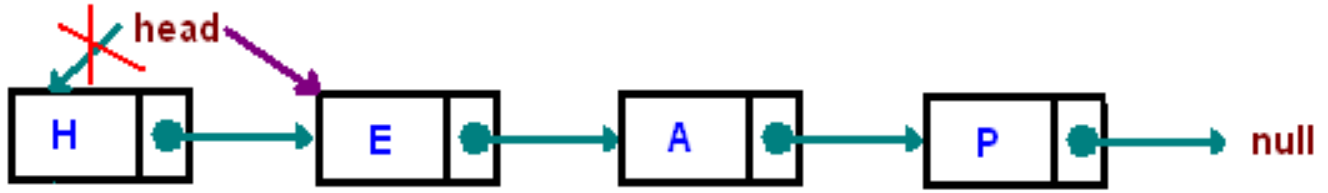


Node

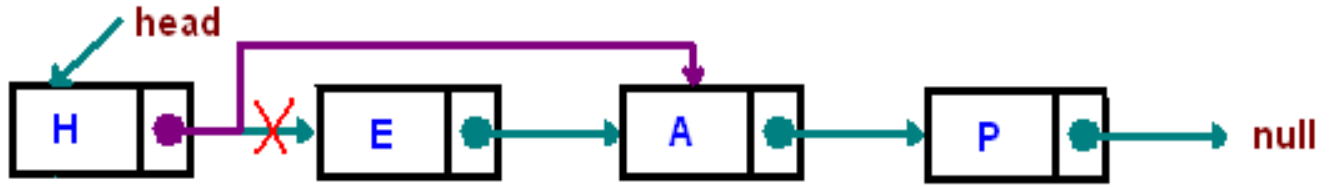


Linked Lists Basics

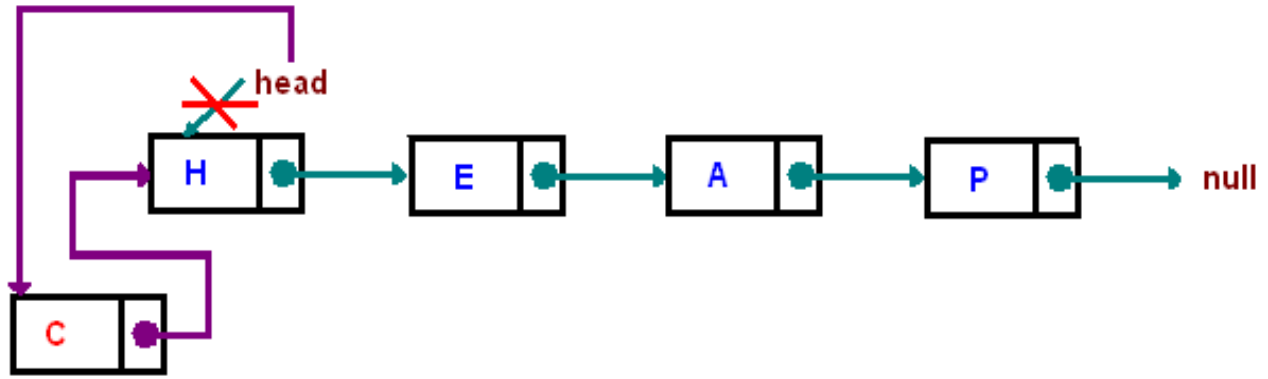
head = head.next



head = head.next.next



newNode = new Node(…);
newNode.next = head;
Head = newNode;

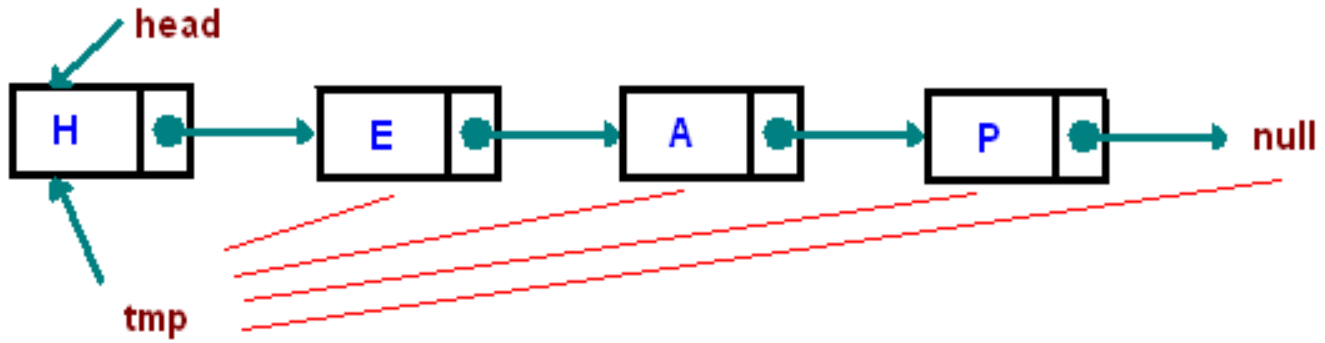


Key Linked List Operations

- Traversal
- Nth Node
- Add
- Insert
- RemoveAt

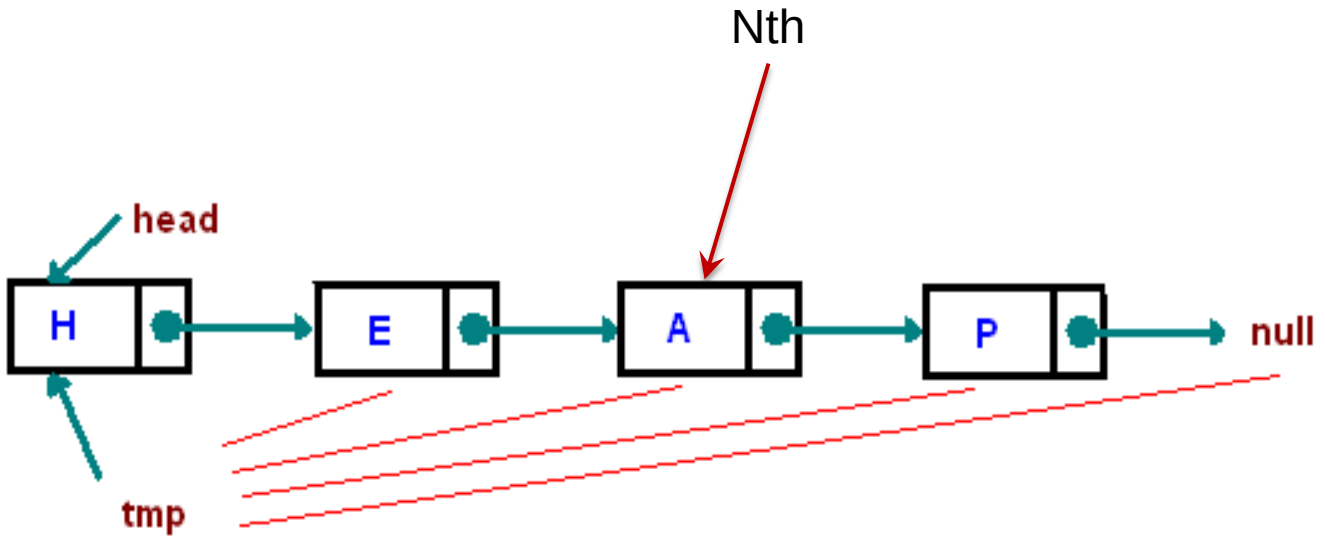
Linked Lists Traversal

```
tmp = head;
While(tmp != null)
{
   //add your code here
    tmp = tmp.next;
}
```



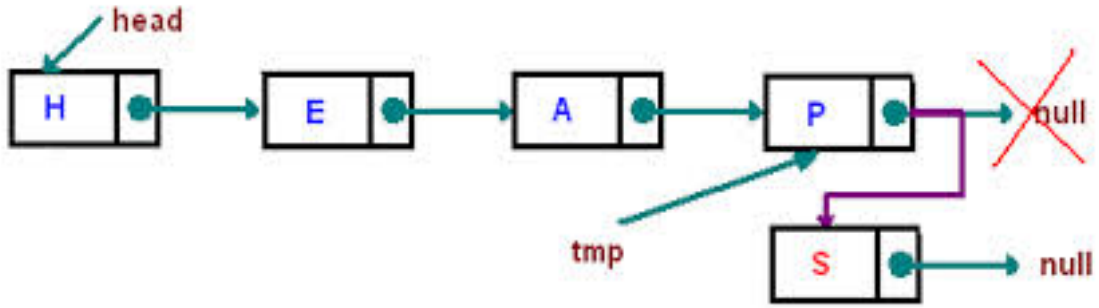
Nth node

```
tmp = head;
int index = 0;
While(tmp != null && index < nth )
{
        tmp = tmp.next;
        index++;
}
return tmp;
```



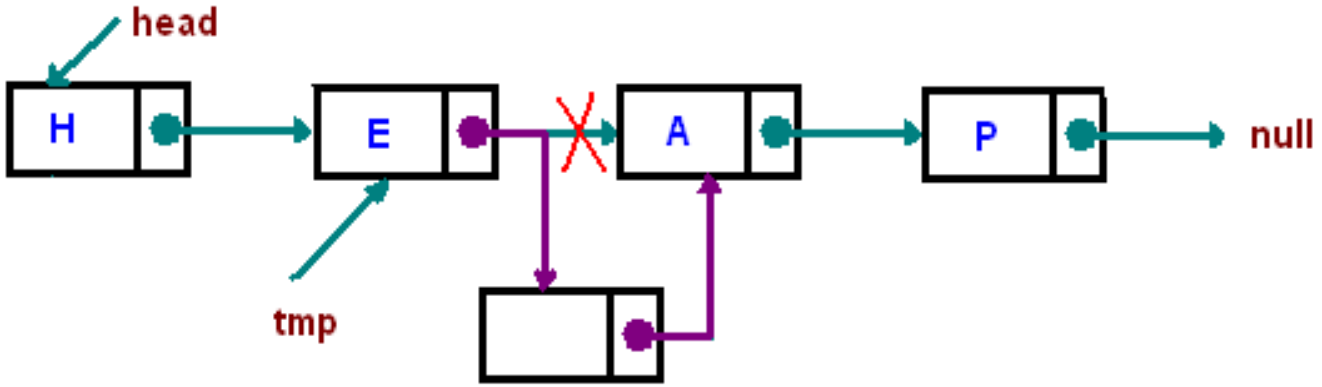
Add node

```
If (head == null)
        head = newNode;
else
{
        tmp = head;
        While (tmp.Next != null) tmp = tmp.Next;
        tmp.Next = newNode;
        newNode.Next = null;
}
```



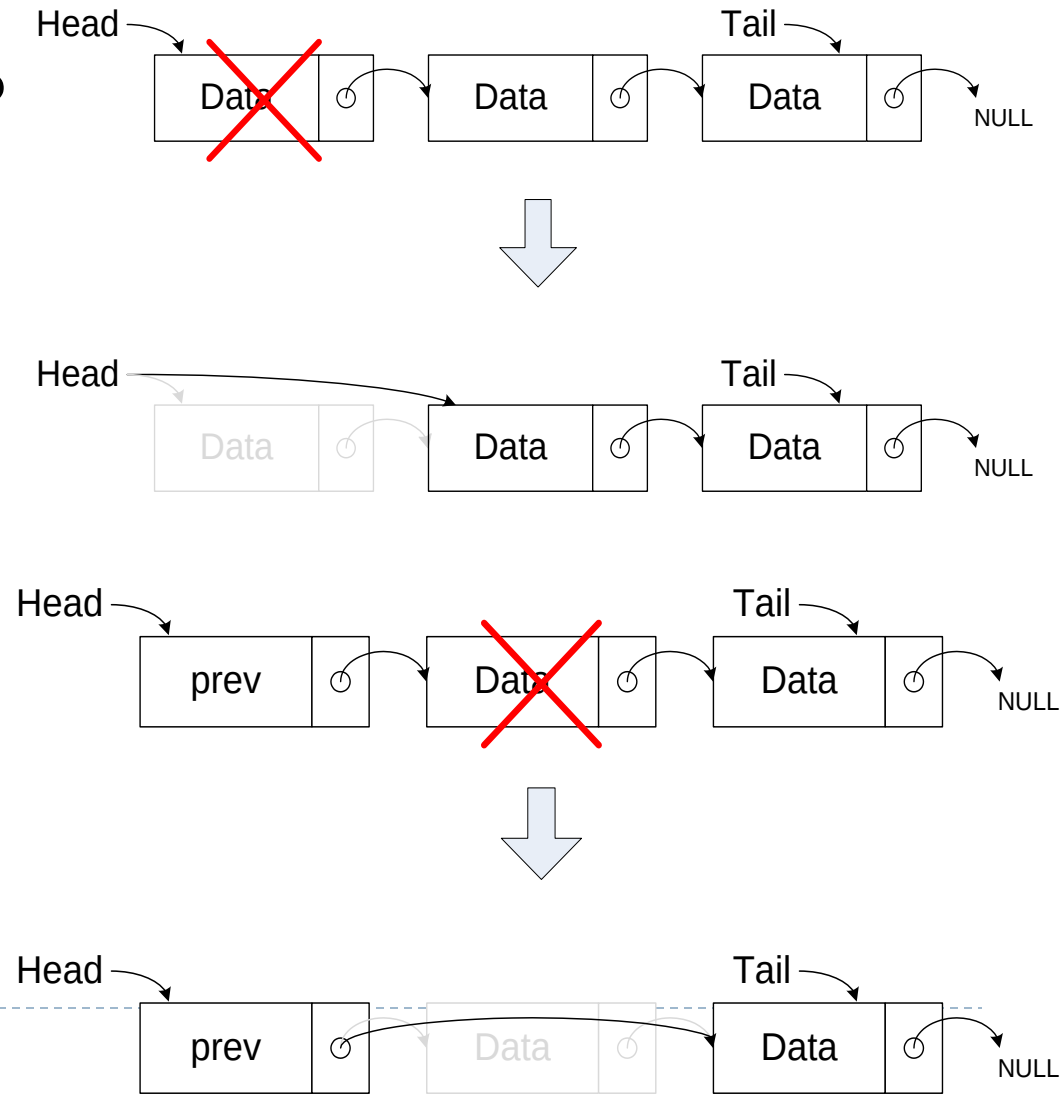
Insert node

- New head – insert at zero?
  - newNode.Next = head;
  - head = newNode;
- At nth position?
  - Tmp = Get nth – 1
  - newNode.Next = tmp.Next
  - tmp.Next = newNode;



Delete node

- Delete head – Delete at zero?
  - Head = head.Next;
- At nth position?
  - Tmp = Get nth – 1
  - Tmp.Next = tmp.Next.Next;



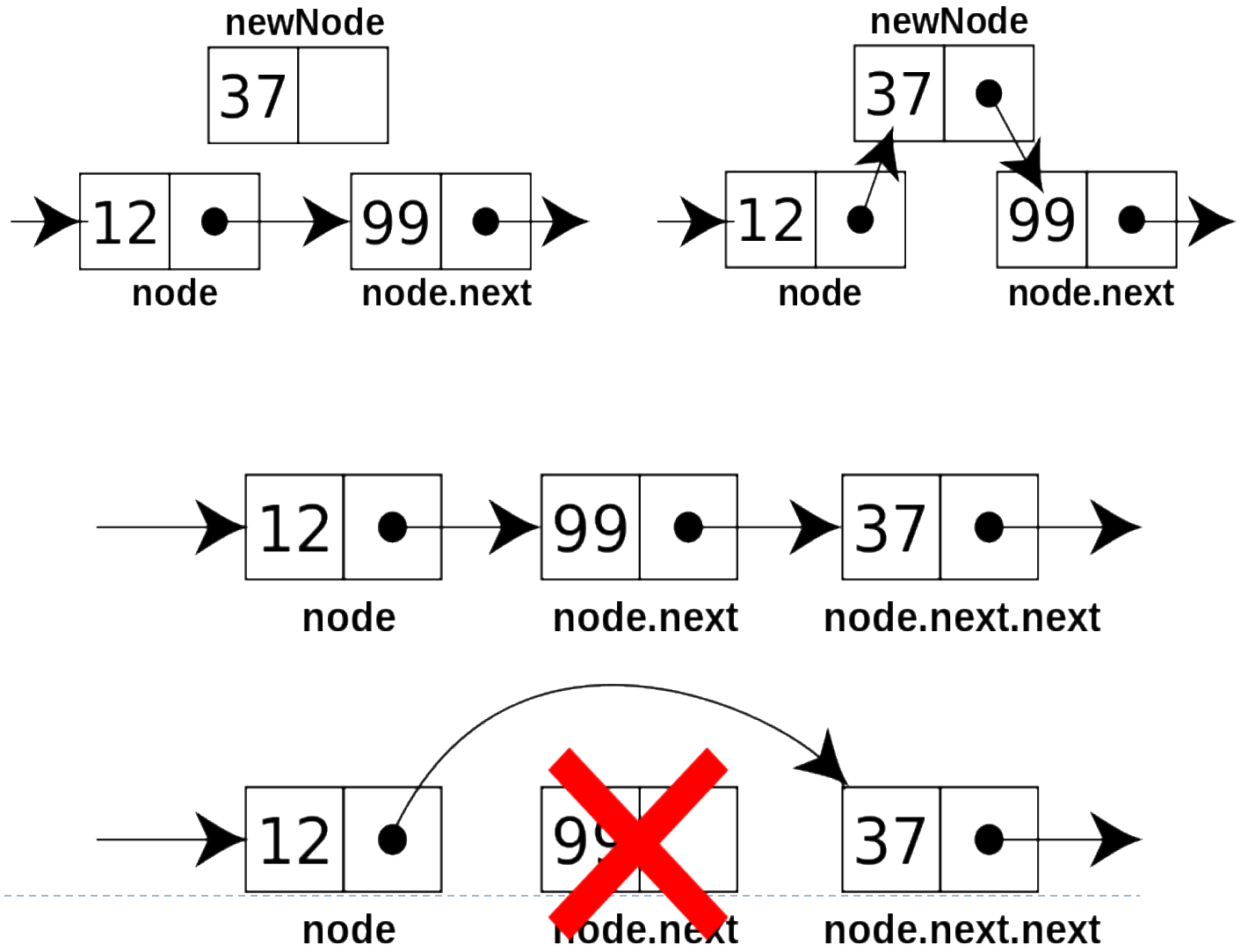
Complexity

- Linked Lists:
  - Constant time insert and delete, BUT polynomial to access.
- Arrays:
  - Constant time random access, better cache locality, BUT polynomial time insert & delete.

	Linked list	Array	Dynamic array
Indexing	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Insert/delete at beginning	$\Theta(1)$	N/A	$\Theta(n)$
Insert/delete at end	$\Theta(n)$ when last element is unknown; $\Theta(1)$ when last element is known	N/A	$\Theta(1)$ amortized
Insert/delete in middle	search time + $\Theta(1)$[5][6][7]	N/A	$\Theta(n)$
Wasted space (average)	$\Theta(n)$	0	$\Theta(n)$[8]

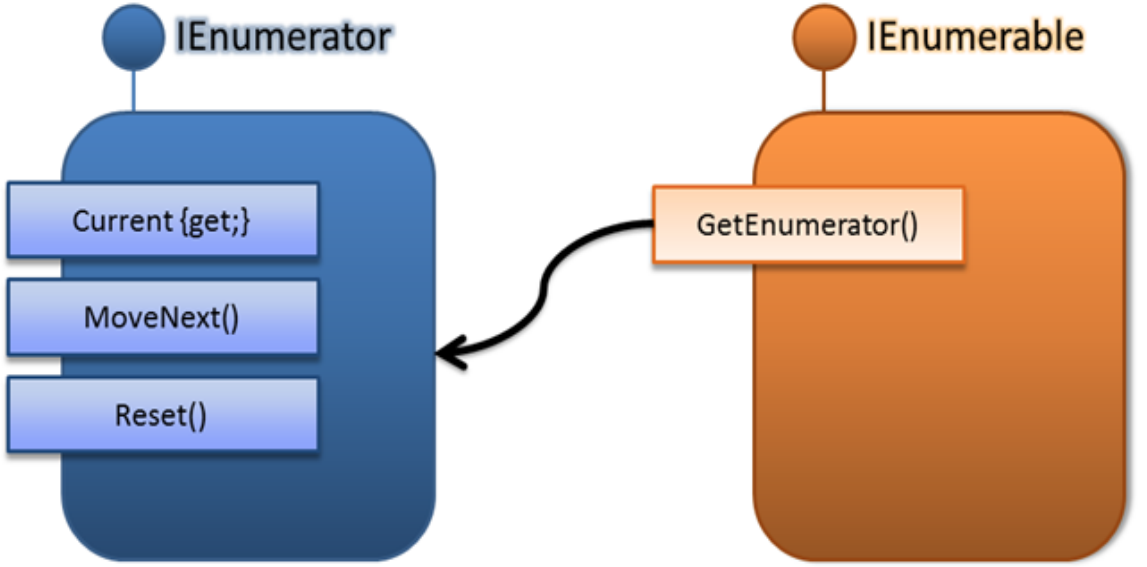
Practical - 1

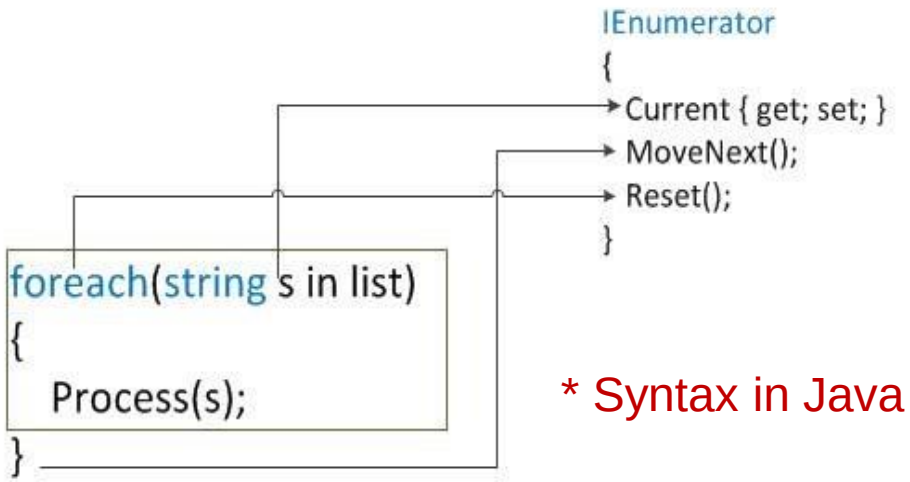
- A singly linked list.
  - IndexOf
  - Insert
  - RemoveAt
  - GetEnumerator
  - MoveNext



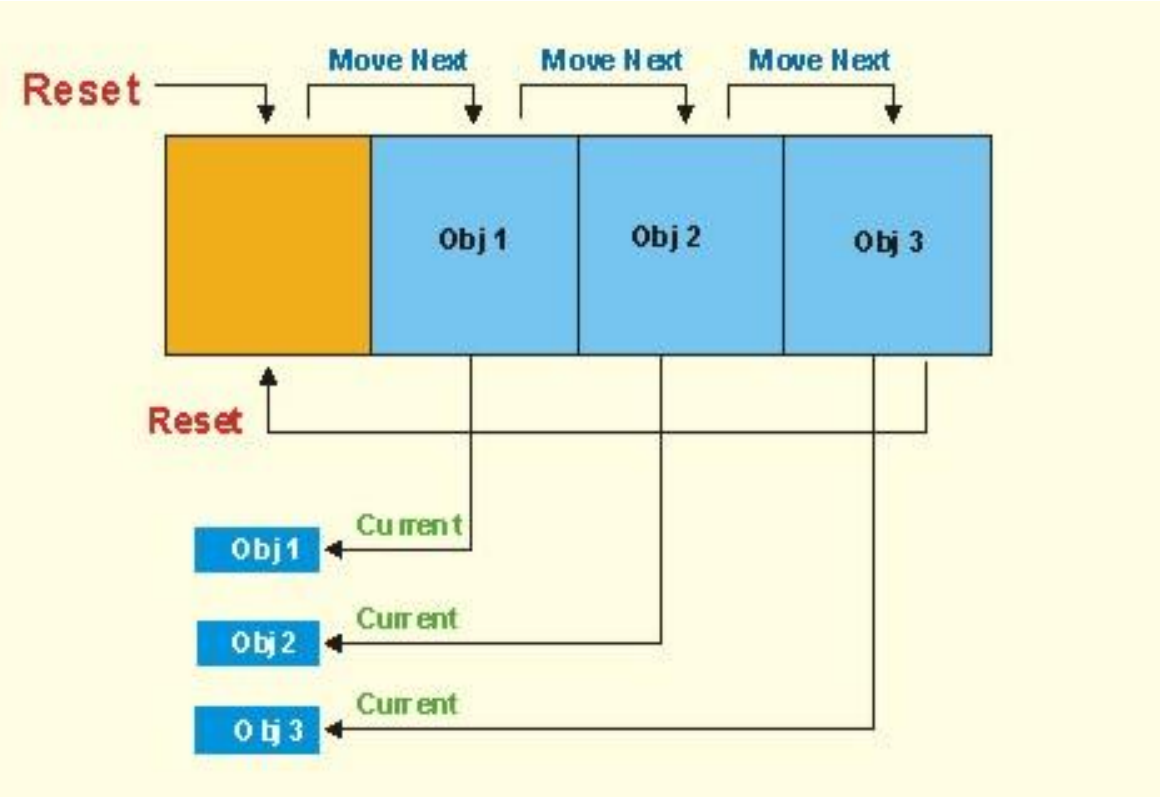
Practical - 2

- A singly linked list.
  - IndexOf
  - Insert
  - RemoveAt
  - GetEnumerator
  - MoveNext





* Syntax in Java



Unit testing – Get ready for next week

- How/what to test a given class/method?
- After you have implemented your Add method, how can you test it's functioning properly?
- What test cases you need to check for?
- Next week, we will have a task to write unit test cases using NUnit.
- Test-driven Development?