# Lecture 13: Linked Lists

*CS2504/CS4092– Algorithms and Linear Data Structures*

Dr. Kieran T. Herley

Department of Computer Science
University College Cork

2013/14

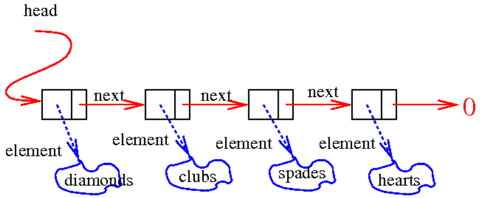
## Summary

*Concept of a linked lists: nodes and links. Basic manipulations of linked lists: simple list traversals, insertions and removals from linked lists. Linked lists as data structures.*

## (Singly) Linked Lists

**Linked List** set of *nodes* (rectangular objects below)

- element bearing
- threaded together



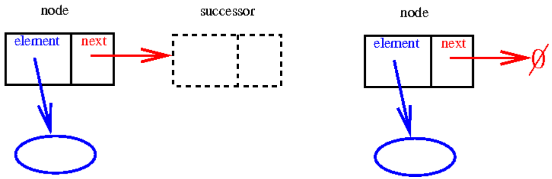
**Note** Useful basis for container representation

## Nodes

### Node

- *element*– object (e.g. String) associated with node
- *next* – a reference to
  - the node's to successsor (left),
  - null (right)

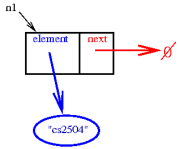
### Pic

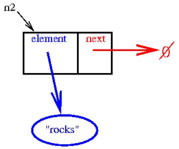


## Class LLNode

```
public class LLNode<EltType>
{   public LLNode()
    {   element = null;
        next = null;
    }
    public LLNode(LLNode<EltType> n,
                  String s)
    {   element = s; next = n;}

    public void setElement(EltType e)
    {   element = e; }
    public EltType getElement()
    {   return element; }

    // other getters and setters

    private EltType element;
    private LLNode<EltType> next;
}
```

```
LLNode<String> n1 =
    new LLNode<String>(null, "cs2504");
LLNode<String> n2 =
    new LLNode<String>(null, "rocks");
```





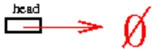
## Linked Lists

**Linked List** chain of zero or more element-bearing nodes

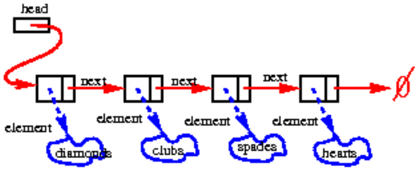
- last node "points to" null
- other nodes each "point to" successor

**Head** reference to first node in the list (null for empty list)

**(Tail– optional)** reference to last node/null



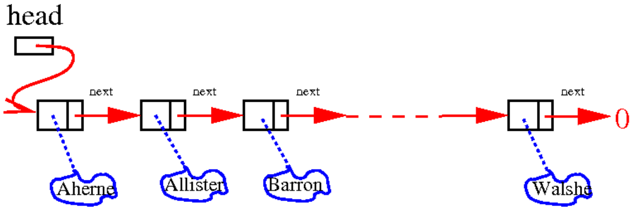
Empty list



Non–empty list

## Usefulness of Linked Lists

- Basis for container representations

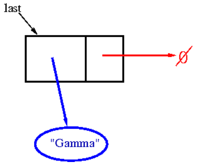


- Advantages of linked lists over arrays:
  - No predetermined size
  - Space usage proportional to size
  - Some manipulations more efficient than arrays
- (Some disadvantages too)

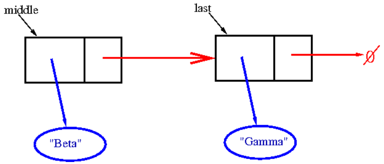
## Creating a List

```
LLNode<String> last = new LLNode<String>(null, "Gamma");
LLNode<String> middle = new LLNode<String>(last, "Beta");
LLNode<String> head = new LLNode<String>(middle, "Alpha");
```

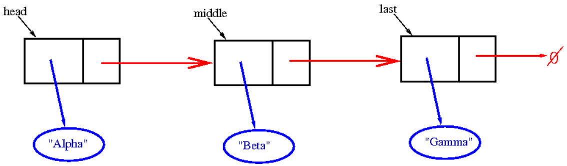
```
LLNode<String> last = new LLNode<String>(null, "Gamma");
LLNode<String> middle = new LLNode<String>(last, "Beta");
LLNode<String> head = new LLNode<String>(middle, "Alpha");
```



```
LLNode<String> last = new LLNode<String>(null, "Gamma");
LLNode<String> middle = new LLNode<String>(last, "Beta");
LLNode<String> head = new LLNode<String>(middle, "Alpha");
```



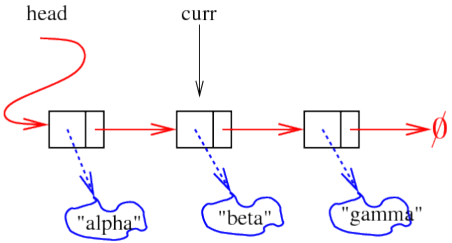
```
LLNode<String> last = new LLNode<String>(null, "Gamma");
LLNode<String> middle = new LLNode<String>(last, "Beta");
LLNode<String> head = new LLNode<String>(middle, "Alpha");
```



NB structure built “organically” node by node, not created all at once like an array

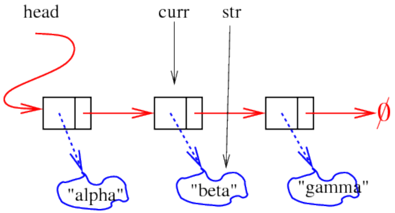
## Basic List Operation I

```
str = curr.getElement();
System.out. println ( str );
```



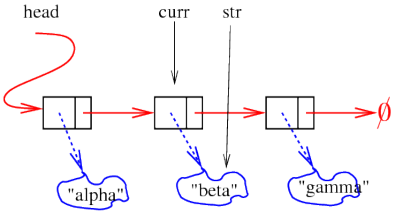
# Basic List Operation I

```
 str  = curr.getElement();
System.out. println ( str );
```



## Basic List Operation I

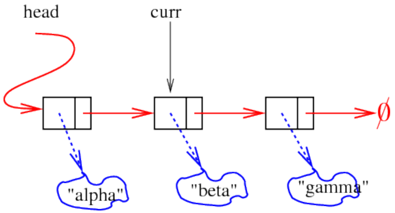
```
str = curr.getElement();
System.out. println ( str );
```



Prints "beta"

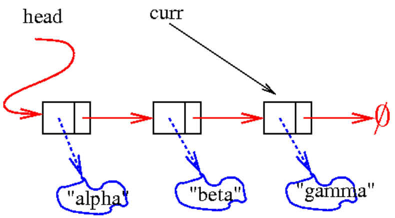
# Basic List Operation II

- curr = curr.getNext();



## Basic List Operation II

```
curr = curr.getNext();
```



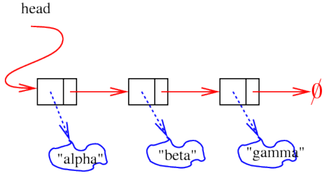
- Steps curr one node forwards in list
- One further execution of

  ```
  curr = curr.getNext();
  ```

  would set curr to null

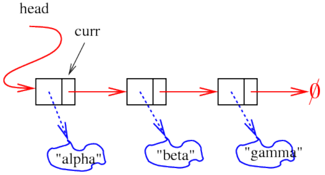
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



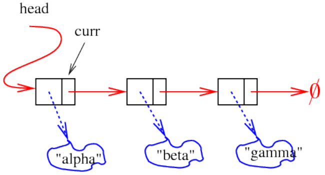
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



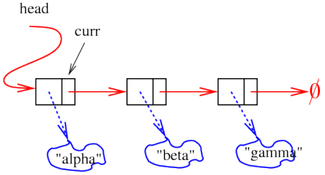
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



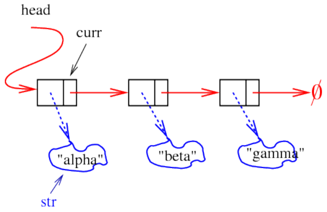
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{  str = curr.getElement();
   System.out.println(str);
   curr = curr.getNext();
}
```



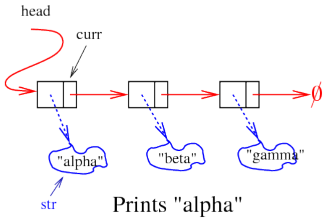
## Traversing a List

```
LLNode<String> curr = head;
String  str ;
while (curr  != null)
{   str  = curr.getElement();
    System.out. println ( str );
    curr  = curr.getNext();
}
```



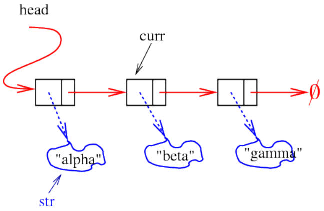
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



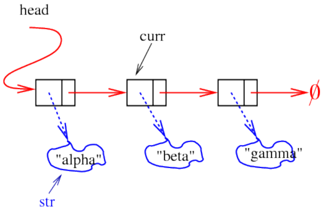
# Traversing a List

```
LLNode<String> curr = head;
String  str ;
while (curr  != null)
{   str  = curr.getElement();
    System.out. println ( str );
    curr  = curr.getNext();
}
```



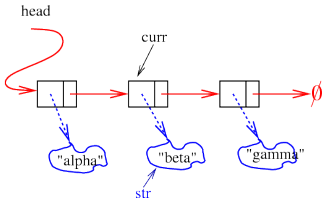
## Traversing a List

```
LLNode<String> curr = head;
String  str ;
while (curr != null)
{   str  = curr.getElement();
    System.out.println ( str );
    curr  = curr.getNext();
}
```



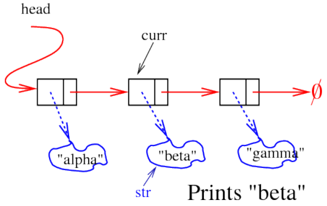
## Traversing a List

```
LLNode<String> curr = head;
String str ;
while (curr != null)
{   str = curr.getElement();
    System.out. println ( str );
    curr = curr.getNext();
}
```



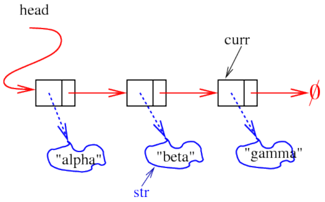
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



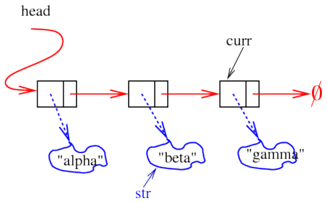
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



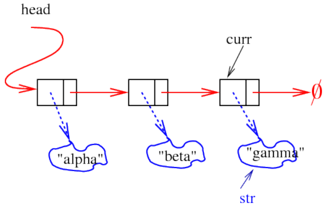
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



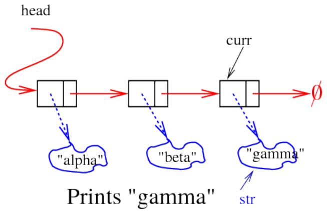
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



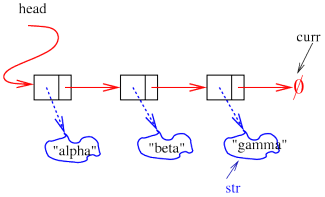
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



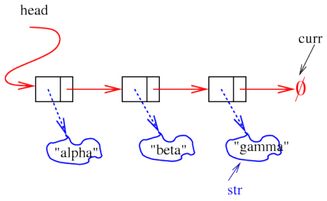
## Traversing a List

```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



## Traversing a List

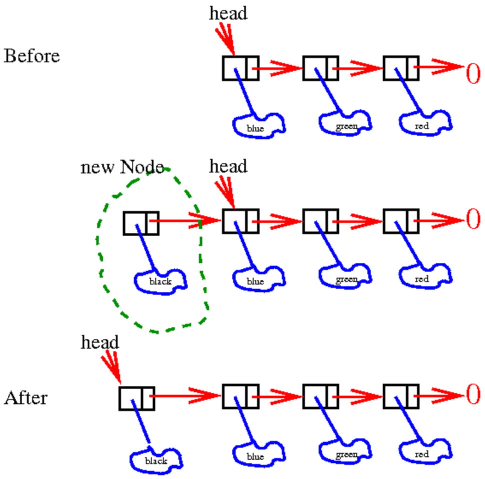
```
LLNode<String> curr = head;
String str;
while (curr != null)
{   str = curr.getElement();
    System.out.println(str);
    curr = curr.getNext();
}
```



## Inserting Into List

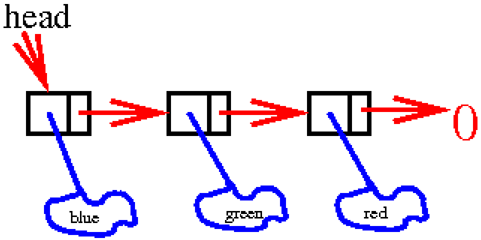
Insertion of a new LLNode at the head of the list:

- new LLNode created with next set to old head
- head redirected to point to new LLNode



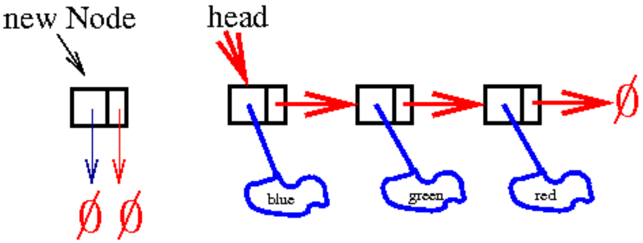
## Inserting . . .

```
LLNode<String> newNode = new LLNode<String>();
newNode.setElement("black");
newNode.setNext(head);
head = newNode;
```



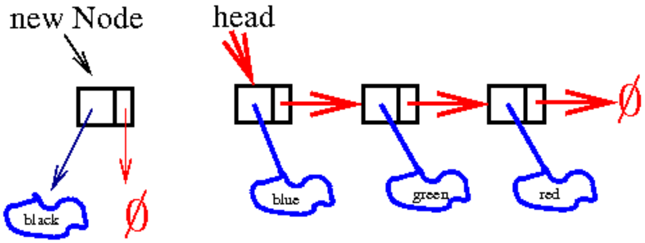
## Inserting . . .

```
LLNode<String> newNode = new LLNode<String>();
newNode.setElement("black");
newNode.setNext(head);
head = newNode;
```



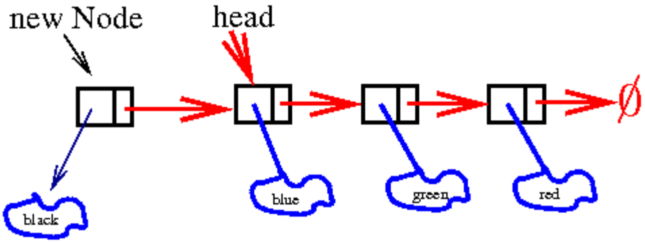
## Inserting . . .

```
LLNode<String> newNode = new LLNode<String>();
newNode.setElement("black");
newNode.setNext(head);
head = newNode;
```



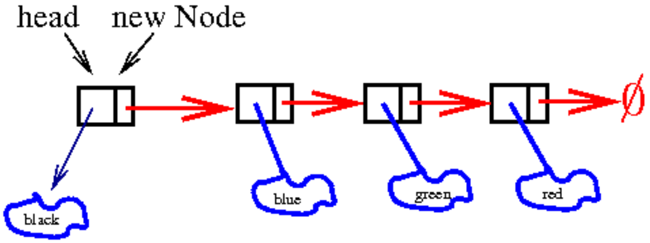
## Inserting . . .

```
LLNode<String> newNode = new LLNode<String>();
newNode.setElement("black");
newNode.setNext(head);
head = newNode;
```



## Inserting . . .

```
LLNode<String> newNode = new LLNode<String>();
newNode.setElement("black");
newNode.setNext(head);
head = newNode;
```

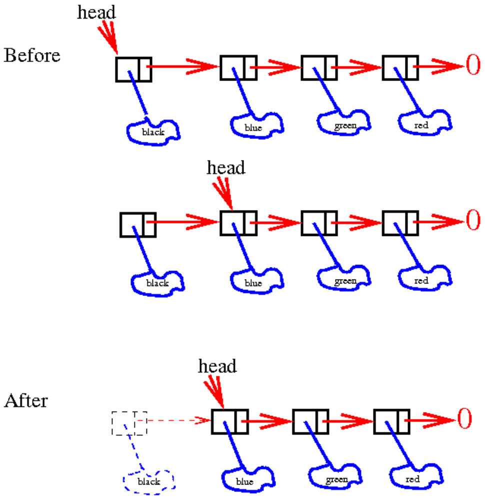


## Removing First List Node

Removal at the head of the list:

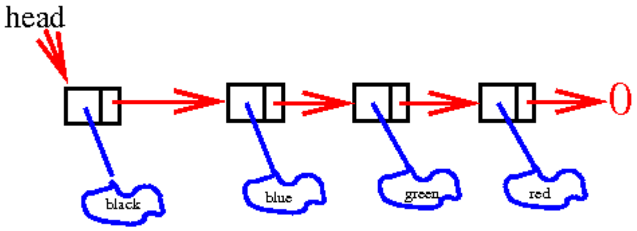
- re-direct head to point to point to next-to-first node;
- (if no other node, make head null)

Note: removed LLNode will eventually be garbage collected.



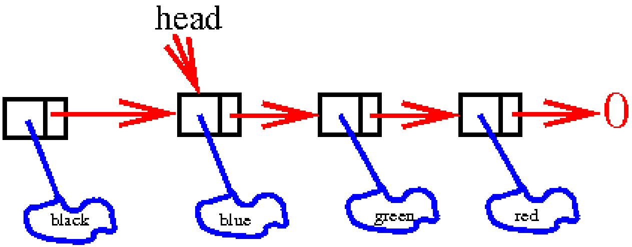
## Removing . . .

```
if (head != null)
{   head = head.getNext();
}
```



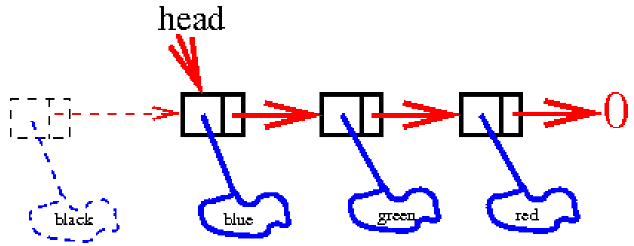
## Removing . . .

```
if (head != null)
{  head = head.getNext();
}
```



## Removing . . .

```
if (head != null)
{   head = head.getNext();
}
```

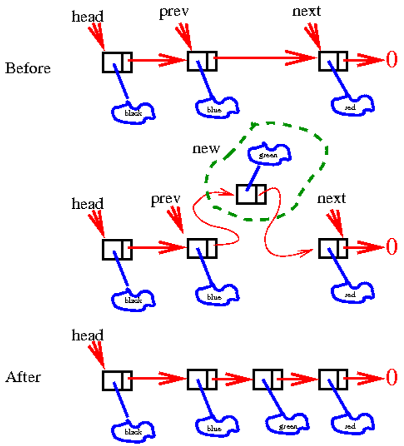


Eventually garbage collected

## Inserting "Interior" Node

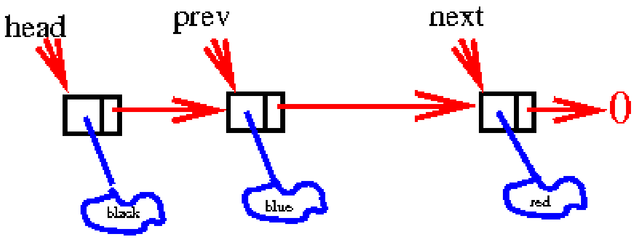
Insertion of a new node in the middle of the list ("between" prev and next):

- create new node, making next its successor
- re-direct prev's reference to make new node its new successor



## Inserting . . .

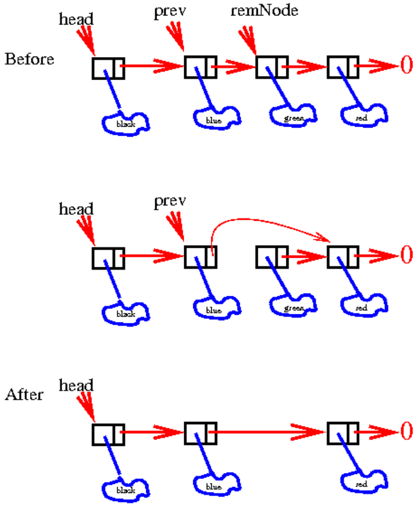
```
newNode = new LLNode(next, "green");
prev.setNext(newNode);
```



## Removing “Interior” Node

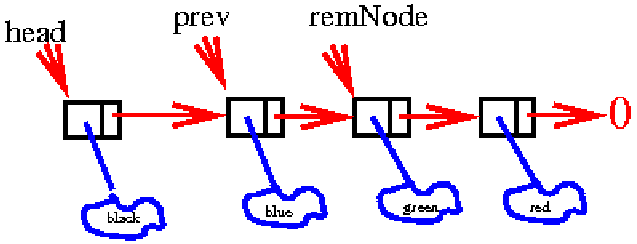
Removal of node in the middle of the list (remNode with predecessor prev):

- re-direct prev’s reference to point to reNode’s successor



## Removing . . .

```
if (remNode == head)
{   head = head.getNext();
}
else
{   prev.setNext(remNode.getNext());
}
```


head
prev
remNode
0
black
blue
green
red