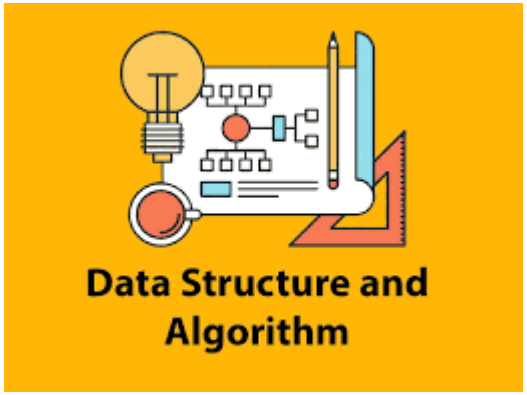
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

NGÀY 3.2: DANH SÁCH LIÊN KẾT



Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Thư viện mẫu trên STL
- Khái niệm DSLK
- Các phép toán trên DSLK
- Phân loại DSLK

1. Thư viện khuân mẫu chuẩn STL

- <array>
- <vector>
- <deque>
- **<forward_list>**
- **<list>**
- <stack>
- <queue>
- <priority_queue>
- <set>
- <multiset>
- <map>
- <multimap>
- <unordered_set>
- <unordered_multiset>
- <unordered_map>
- <unordered_multimap>
- <bitset>
- <valarray>

1. Thư viện khuân mẫu chuẩn STL

- Ví dụ 1: push_front

```
#include <iostream>
#include <list>
using namespace std;

// list::push_front
int main(){
    list<int> mylist(2,100); // 2 bien nguyen gia tri 100
    mylist.push_front(200);
    mylist.push_front(300);
    cout << "mylist chua day so:";
    list<int>::iterator it;
    for (it=mylist.begin(); it!=mylist.end(); ++it)
        cout << ' ' << *it;
    cout << '\n';

    return 0;
}
```

1. Thư viện khuân mẫu chuẩn STL

- Ví dụ 2: pop_front

```
#include <iostream>
#include <list>

// list::pop_front
using namespace std;
int main(){
    list<int> mylist;
    mylist.push_back(100);
    mylist.push_back(200);
    mylist.push_back(300);
    cout << "Thuc hien pop_front cac phan tu trong mylist:";
    while(!mylist.empty()) {
        cout << ' ' << mylist.front();
        mylist.pop_front();
    }
    cout << "\nKich thuoc cuoi cung cua mylist: "
         << mylist.size() << '\n';

    return 0;
}
```

1. Thư viện khuân mẫu chuẩn STL

- Tham khảo http://www.cplusplus.com/reference/list/list/
- Các phép toán
  - push_front
  - pop_front
  - push_back
  - pop_back
  - insert
  - erase

2. Khái niệm

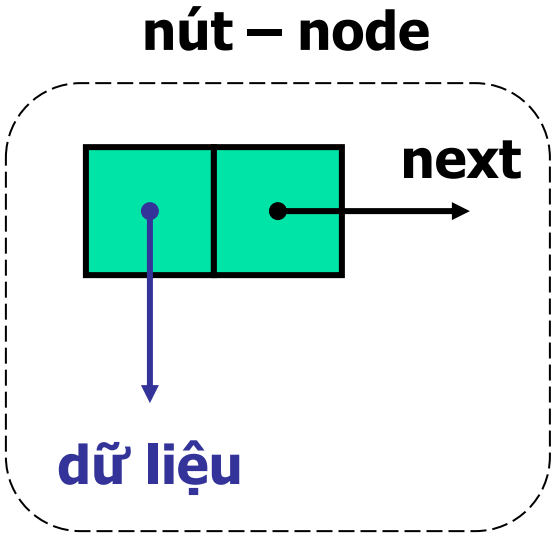
So sánh mảng và DSLK

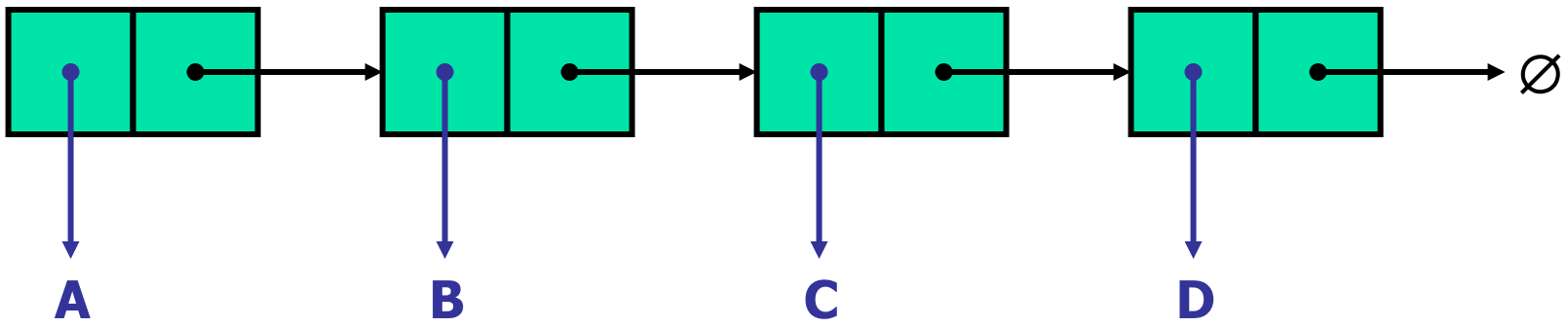
- Mảng
  - Lưu trữ tĩnh
  - Truy xuất nhanh
  - Kích thước bị giới hạn
  - Sử dụng bộ nhớ chưa hiệu quả
  - Thêm, bớt phần tử chậm (O(N))
- Danh sách liên kết
  - Lưu trữ động
  - Truy xuất tuần tự
  - Thêm, bớt phần tử nhanh
  - Kích thước không bị giới hạn
  - Tối ưu bộ nhớ

2. Khái niệm

DSLK được tạo thành từ các nút

- mỗi nút gồm 2 phần
  - phần dữ liệu: chứa phần tử dữ liệu
  - phần con trỏ: chứa 1 địa chỉ
- các nút liên kết với nhau thông qua con trỏ





2. Khái niệm

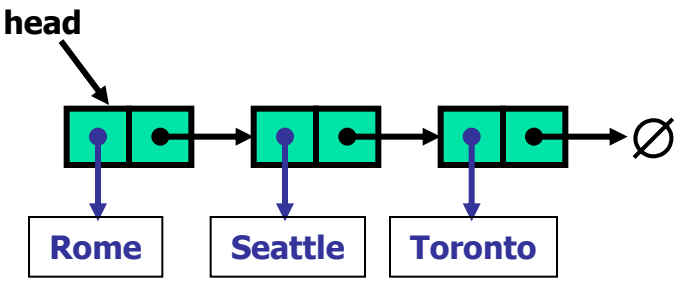
- Mỗi nút là một biến Node
- Nút cuối cùng có giá trị next bằng NULL
- Xác định DSLK bằng địa chỉ của

nút đầu tiên trong danh sách:

  - gọi biến lưu địa chỉ này là con trỏ đầu head
  - khởi tạo danh sách rỗng.

```
struct Node{
        Item data;
        Node * next;
};
```

```
Node * head = NULL;
```



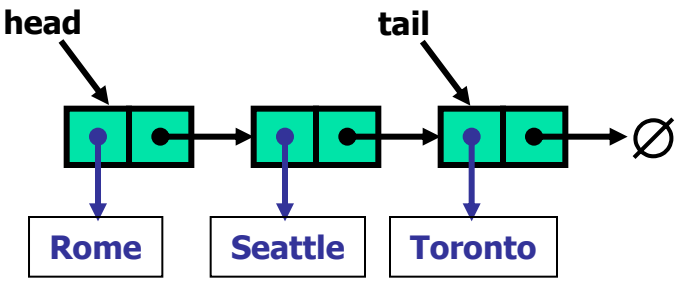
2. Khái niệm

- Có thể sử dụng them con trỏ đuôi tail

để các thao tác trên DSLK được thuận lợi

```
struct Node{
        Item data;
        Node * next;
};
Node * head;
Node * tail;
```

```
head = tail = NULL;
```



3. Các thao tác trên DSLK

- Thêm dữ liệu vào danh sách
  - Thêm vào đầu danh sách: addFirst(v)
  - Thêm vào sau nút pivot: insertAfter(pivot, v)
  - Thêm vào cuối danh sách: addLast(v)
- Duyệt danh sách: traverse()
- Xóa dữ liệu khỏi danh sách
  - Xóa nút đầu danh sách: removeFirst()
  - Xóa nút cuối danh sách: removeLast() ?
  - Xóa nút không phải đầu danh sách: remove(key)

3. Các thao tác trên DSLK

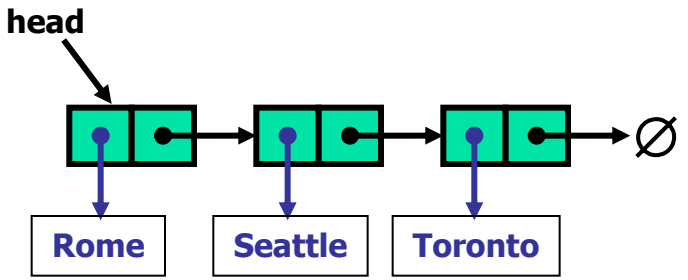
(1) Khởi tạo:

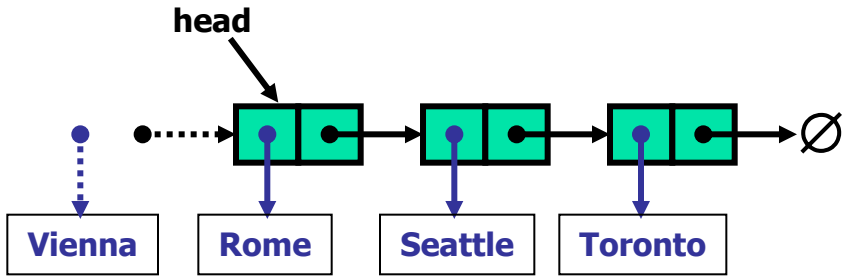
```
struct LinkedList {
    struct Node {
        string data;
        Node* next;
        Node(){}
        Node(string _data) {
            data = _data;
            next = nullptr;
        }
    };

    Node* head = nullptr;
```

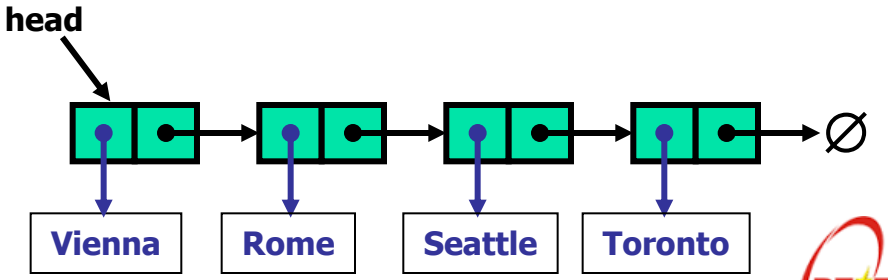
3. Các thao tác trên DSLK

(1.1) Thêm vào đầu danh sách:

- Cấp phát nút mới
- Đưa dữ liệu vào nút mới
- Cho nút mới trỏ tới head cũ
- Sửa head để trỏ tới nút mới





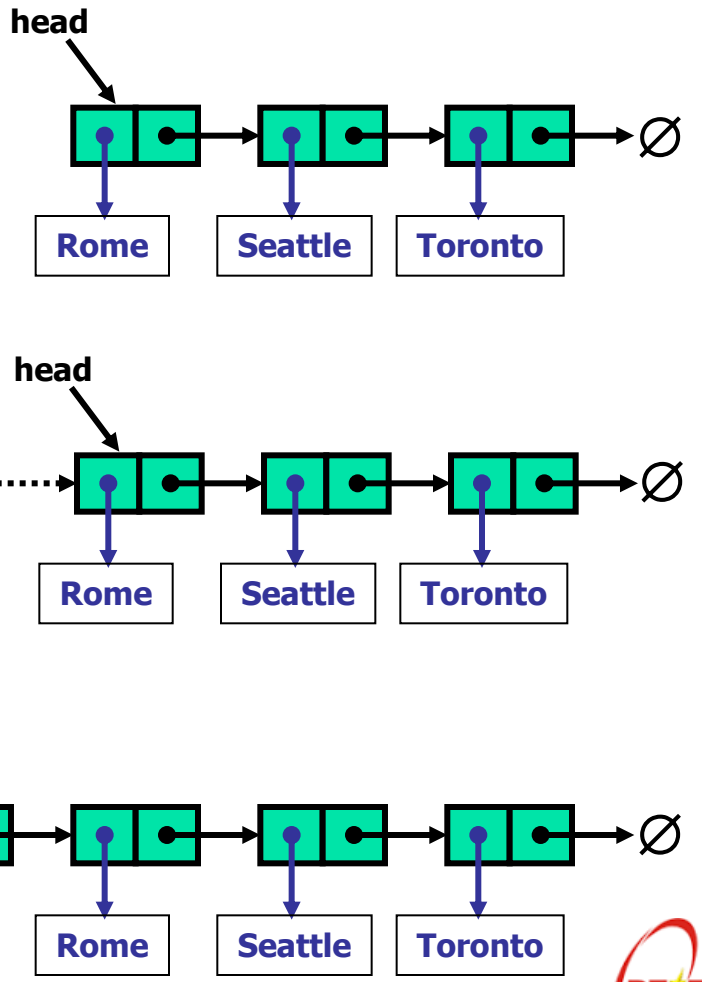


3. Các thao tác trên DSLK

(1.1) Thêm vào đầu danh sách:

- Cấp phát nút mới
- Đưa dữ liệu vào nút mới
- Cho nút mới trỏ tới head cũ
- Sửa head để trỏ tới nút mới

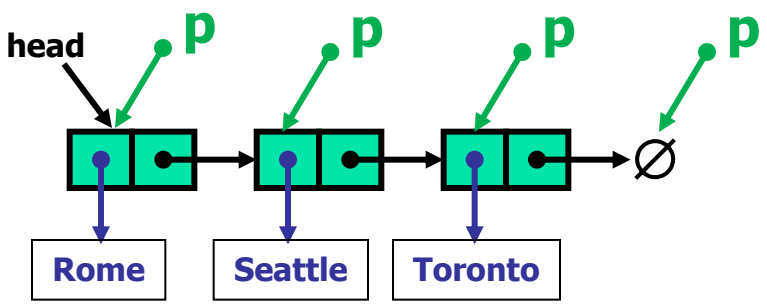
```
void addFirst(string v) {
    Node* new_node = new Node();
    new_node->data = v;
    new_node->next = head;
    head = new_node;
}
```



3. Các thao tác trên DSLK

(2) Duyệt DSLK:

- p bắt đầu từ head
- Nếu p không null thì
  - Xử lí dữ liệu tại p
  - Đẩy p tới nút tiếp theo
  - Quay lại bước 2



Rome Seattle Toronto done

```
void printList(){
    for(Node* p = head; p != NULL; p = p->next) {
        cout << p->data << " ";
    }
    cout << endl;
}
```

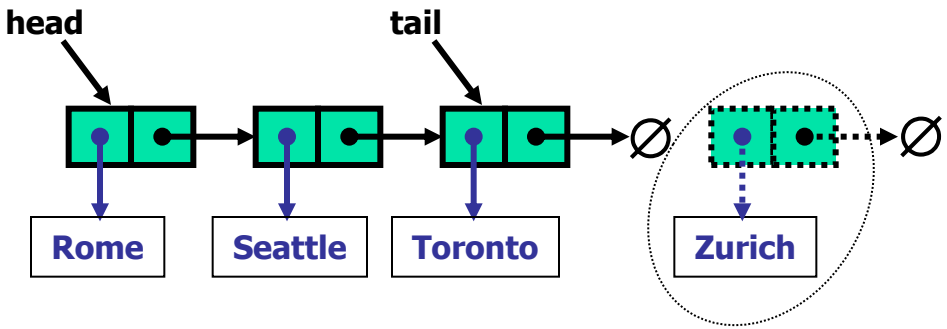
3. Các thao tác trên DSLK

(1.2) Thêm vào cuối danh sách:

- Cấp phát nút mới
- Đưa dữ liệu vào nút mới
- Nếu head là null thì gán head = nút mới
- Ngược lại, duyệt tới nút cuối cùng (last)

Rồi để last trỏ tới nút mới

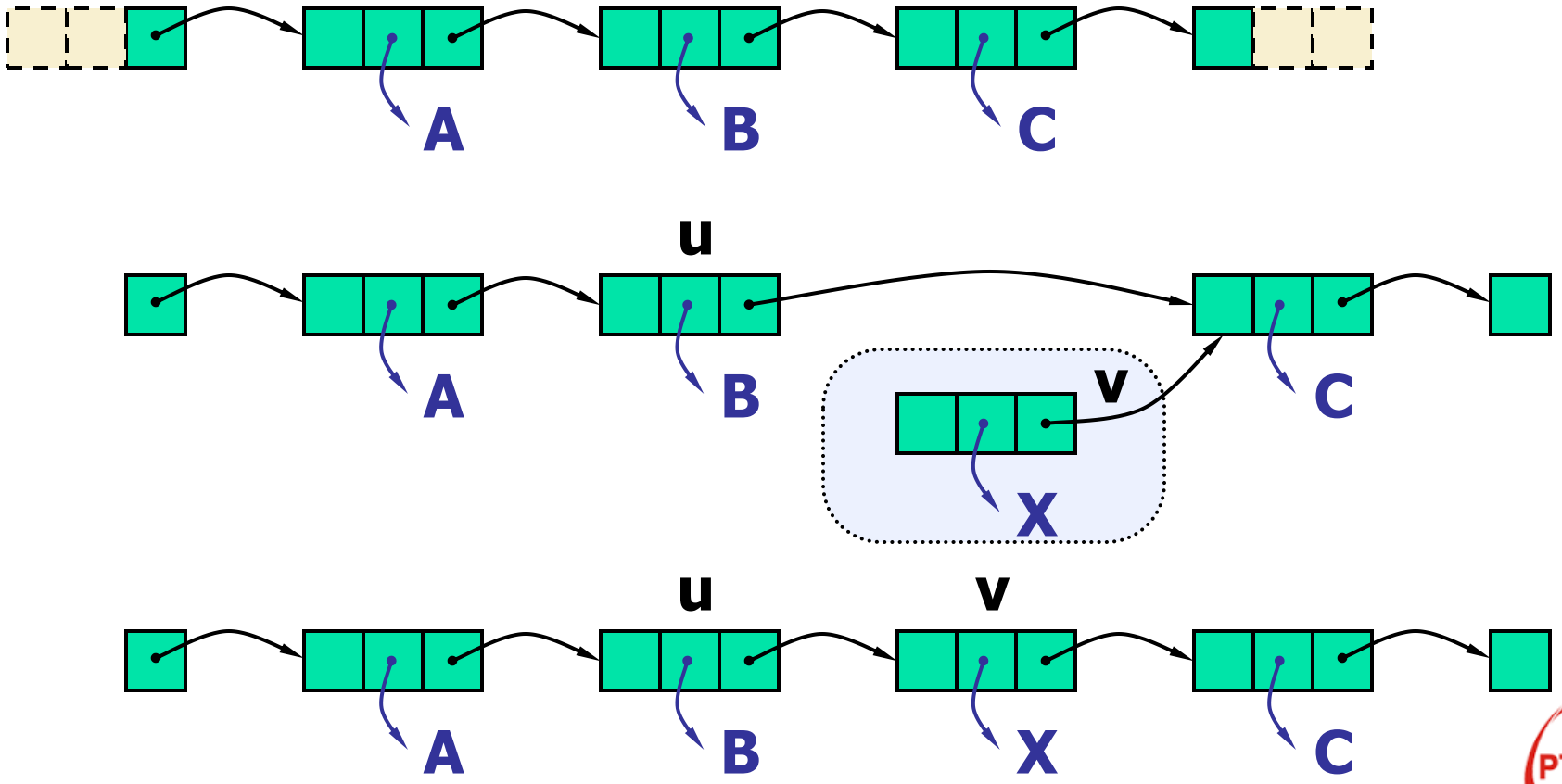
```
void addLast(string v) {
    Node* new_node = new Node();
    new_node->data = v;
    new_node->next = nullptr;

    if(head == NULL) {
        head = new_node;
    }
    else {
        Node* p = head;
        while(p->next != nullptr) {
            p = p->next;
        }
        p->next = new_node;
    }
}
```



3. Các thao tác trên DSLK

(1.3) Thêm phần tử key sau pivot:

- Ví dụ: thêm nút key = X sau pivot B



3. Các thao tác trên DSLK

(1.3) Thêm phần tử key sau pivot:

- Tạo nút mới new_node, gán giá trị key cho nút mới
- Tìm vị trị current_node có data là pivot
- Nối next của new_node tới next của current_node
- Nối next của current_node tới new_node

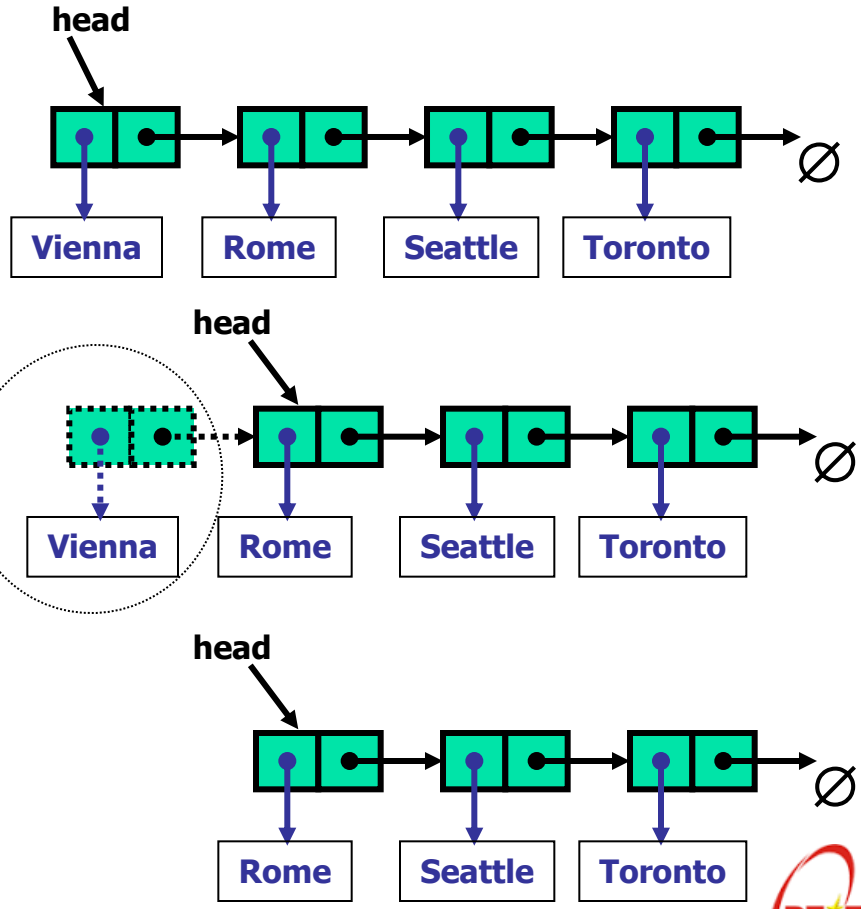
```
void insertAfter(string pivot, string newKey) {
    Node* current = head;
    /// jump current to the node that current->data == pivot

    Node* new_node = new Node;
    new_node->data = newKey;
    new_node->next = current->next;

    current->next = new_node;
}
```

3. Các thao tác trên DSLK

(3.1) Xóa phần tử ở đầu danh sách:

- Sửa head để trỏ tới nút thứ hai trong danh sách

- Giải phóng nút đứng đầu
  - delete old;



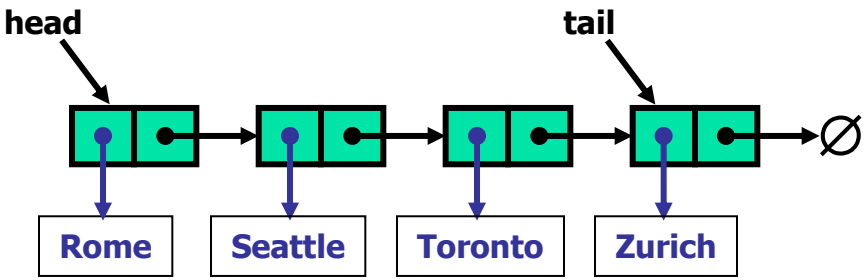

```
void removeFirst() {
    if(head == nullptr) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
```

3. Các thao tác trên DSLK

(3.2) Xóa phần tử cuối danh sách:

- Đối với danh sách liên kết đơn, không có cách nào ngoài việc lấy con trỏ chạy từ đầu đến nút ngay trước tail
- Không hiệu quả!

```
void removeLast() {
    if(head == nullptr) return;
    if(head->next == nullptr) return;

    Node* second_last = head;
    while(second_last->next->next != nullptr) {
        second_last = second_last->next;
    }
    second_last->next = nullptr;

    delete(second_last->next);
}
```

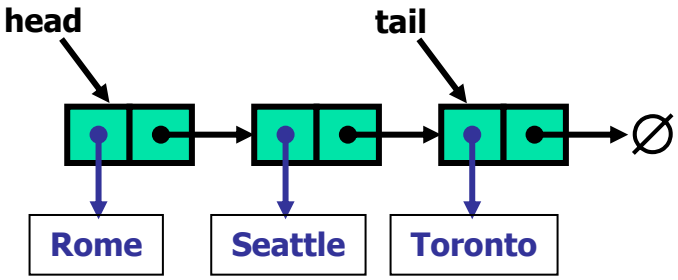


3. Các thao tác trên DSLK

Cải tiến cho 3.2:

Thêm con trỏ chặn cuối tail

→ Các thao tác ở cuối DSLK trở nên dễ dàng hơn




```
struct SLinkedListWithTail {
    Node* head; // head node
    Node* tail; // tail node of the list

      /* Default constructor that creates an empty list */
    SLinkedListWithTail() {
        head = null;
        tail = null;
    }
    // ... update and search methods would go here ...
}
```

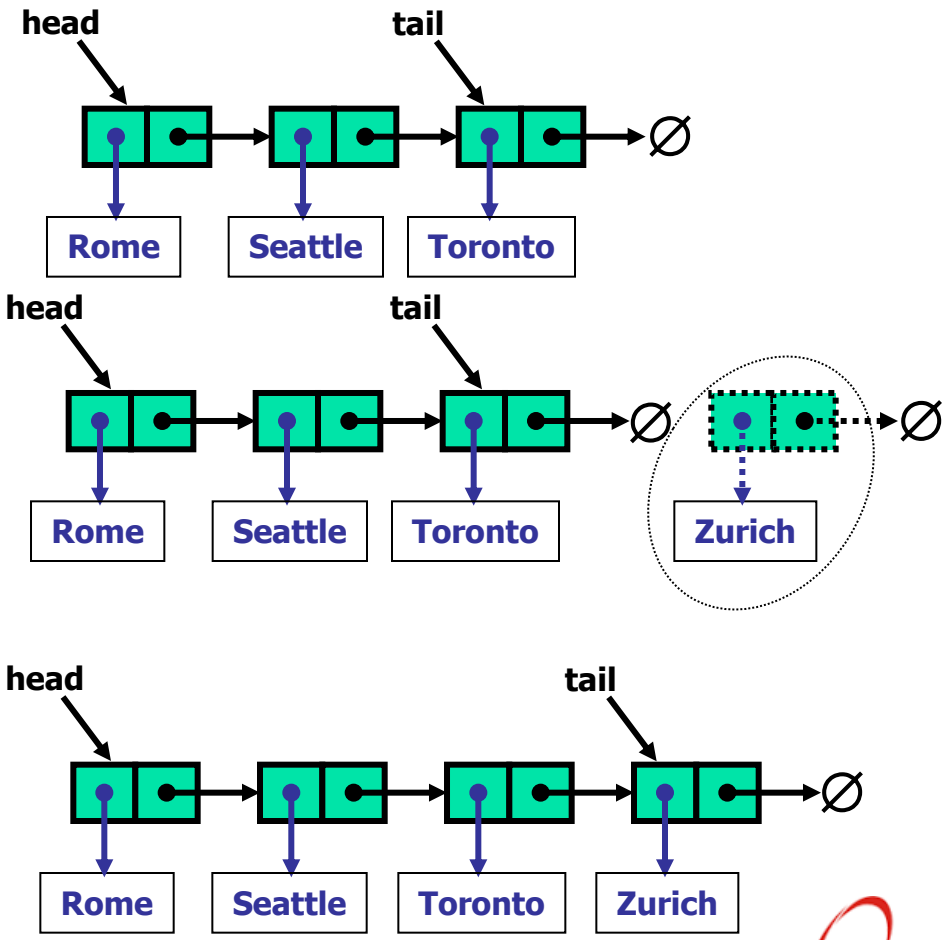
3. Các thao tác trên DSLK

Cải tiến cho 3.2:

Thêm con trỏ chặn cuối tail

Thao tác thêm vào cuối DSLK

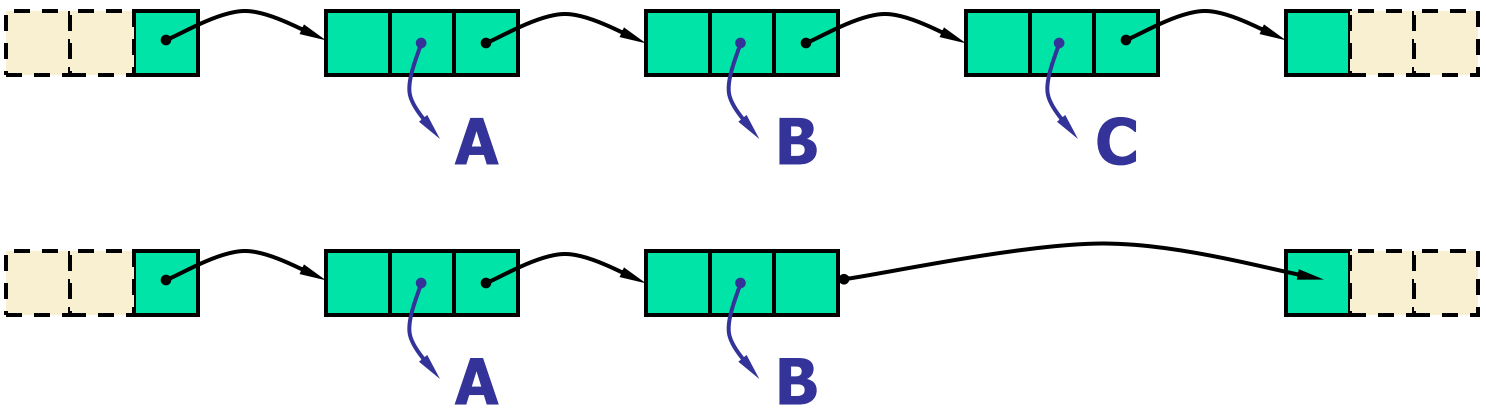
- Cấp phát nút mới
- Lắp dữ liệu
- Cho nút mới trỏ tới null
- Cho nút cuối trỏ tới nút mới
- Sửa tail để trỏ tới nút mới – đuôi mới



3. Các thao tác trên DSLK

(3.3) Xóa bỏ nút có giá trị bằng key:

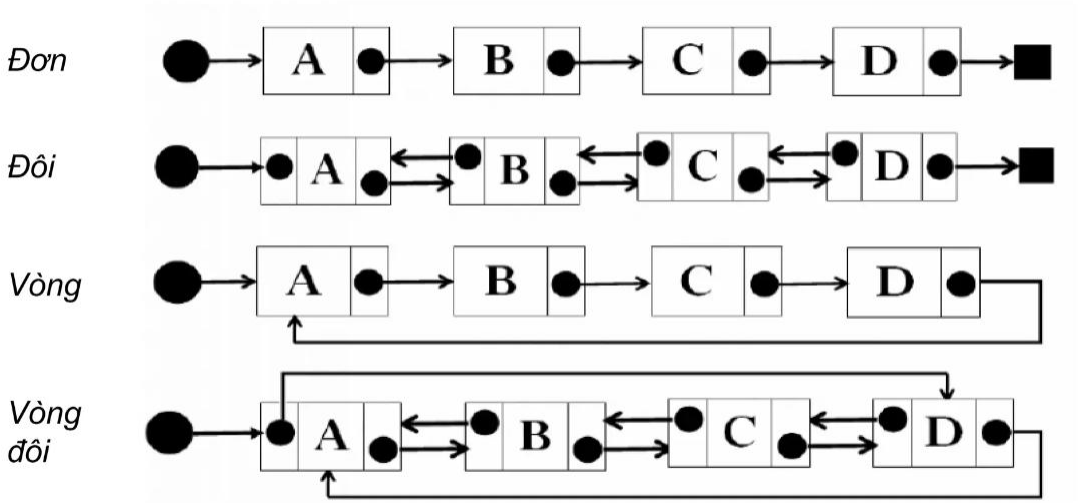
- Tìm kiếm nút có giá trị bằng key
- Nếu tồn tại, xóa bỏ nút này
- Tìm nút trước nút chứa key (node B)
- Nối B→next = C→ next
- Delete C



4. Phân loại DSLK

Có 4 loại phổ biến:

- DSLK đơn
- DSLK đôi
- DSLK vòng
- DSLK vòng đôi



QUESTIONS & ANSWERS