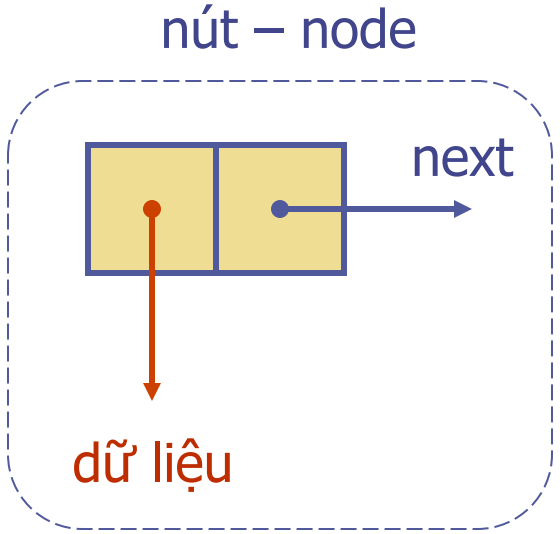
Ôn tập struct và bộ nhớ động
Linked Lists

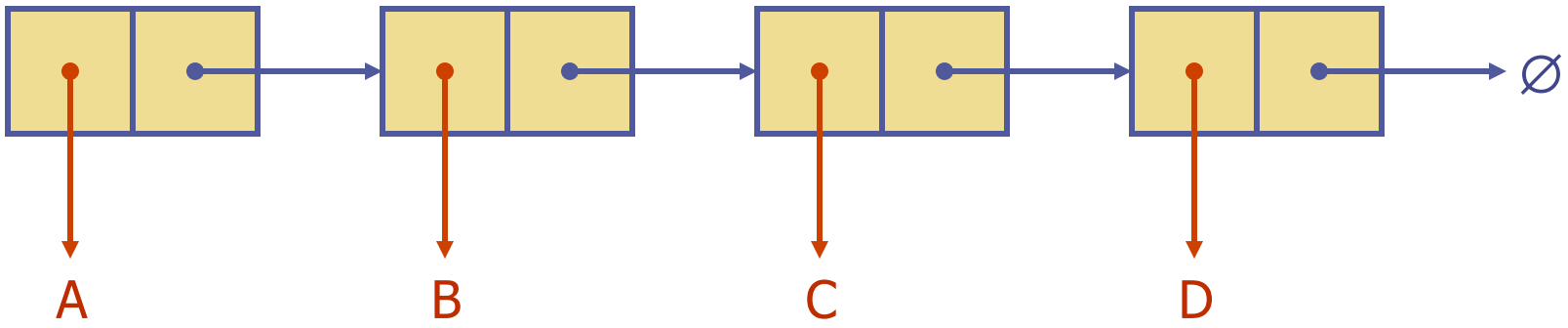
Lập trình nâng cao

Acknowledgement:
These slides are adapted from slides provided with *Data Structures and Algorithms in C++*
Goodrich, Tamassia and Mount (Wiley, 2004)

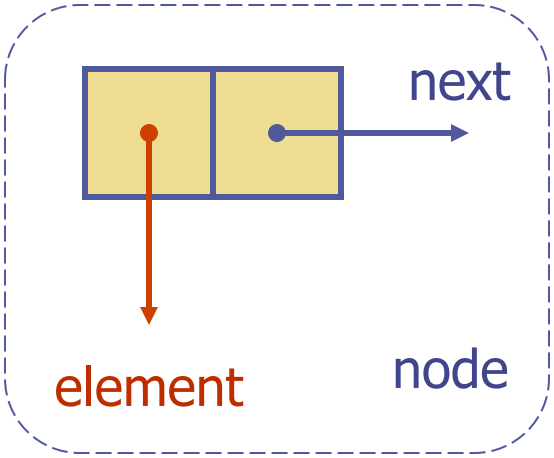
Singly Linked Lists
Danh sách liên kết đơn

- Danh sách liên kết đơn là một cấu trúc dữ liệu chứa một chuỗi các nút (node)
- Mỗi nút chứa
  - dữ liệu của nút đó
  - Con trỏ tới nút tiếp theo





Node chứa xâu kí tự




```
struct Node {
   // Instance variables
   const char* element;
   Node* next;

   // Initialize a node
   Node(const char* e = NULL, Node* n = NULL)
   {
      element = e;
      next = n;
   }
};
```

Trong ví dụ này, các nút chỉ dùng để móc nối dữ liệu chứ không để sửa dữ liệu, do đó dùng **const** cho element

Singly linked list

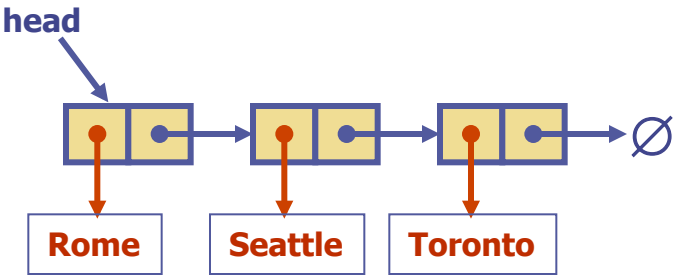
```
struct SLinkedList {
    Node* head; // con trỏ tới nút đầu danh sách
    long size;  // số nút trong danh sách, nếu cần

    /* Constructor mặc định tạo danh sách rỗng */
    SLinkedList() {
        head = NULL;
        size = 0;
    }
    // ... các hàm update và tìm kiếm ...
    bool isEmpty() { return head == NULL; }
    void insertAfter(Node * node, char* element) {...}
    ...
};
```

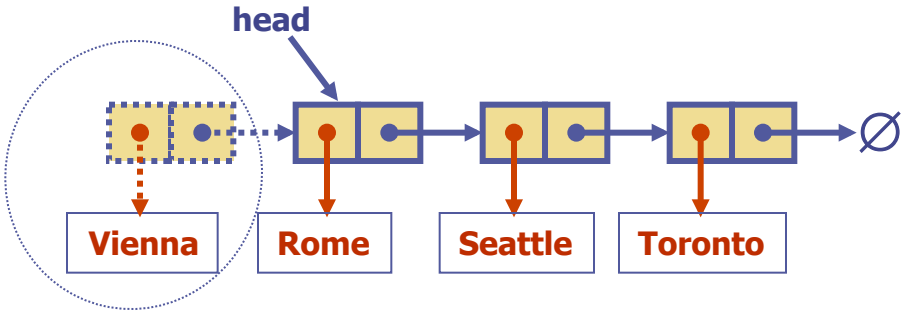
Thêm vào đầu danh sách

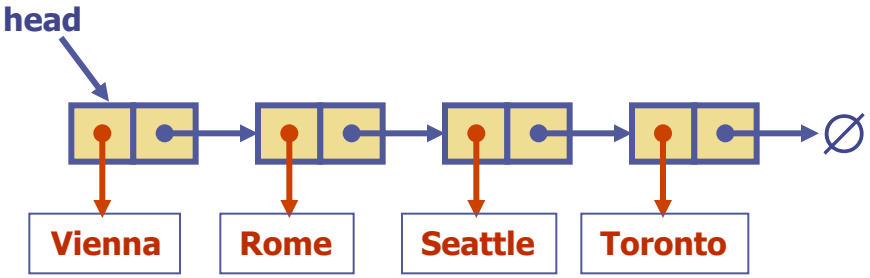
```
struct SLinkedList {
    Node* head;

  ...
```

Sử dụng:

```
SLinkedList list;
...
list.addFirst("Vienna");
```

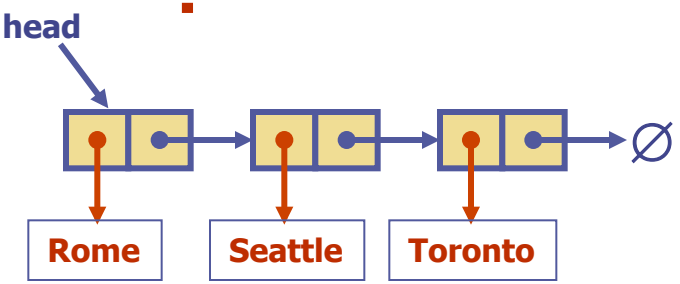




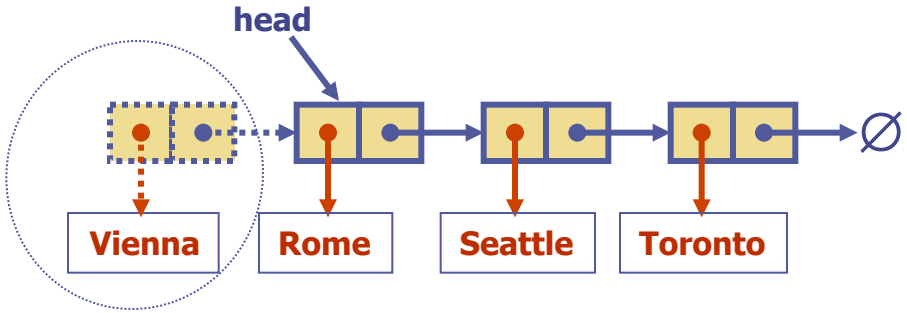


SLinkedList.addFirst – thuật toán

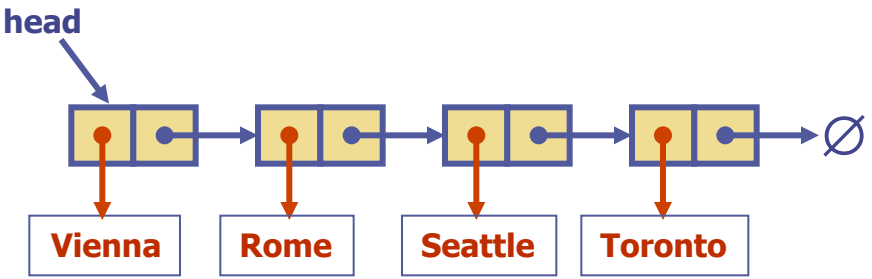
```
list.addFirst("Vienna");
```



1. Cấp phát nút mới
2. Đưa dữ liệu vào nút mới
3. Cho nút mới trỏ tới head cũ

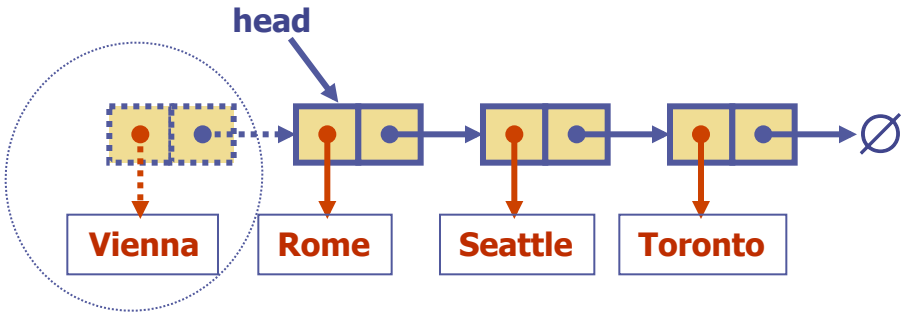


4. Sửa head để trỏ tới nút mới



SLinkedList.addFirst – cài đặt

```
list.addFirst("Vienna");
```

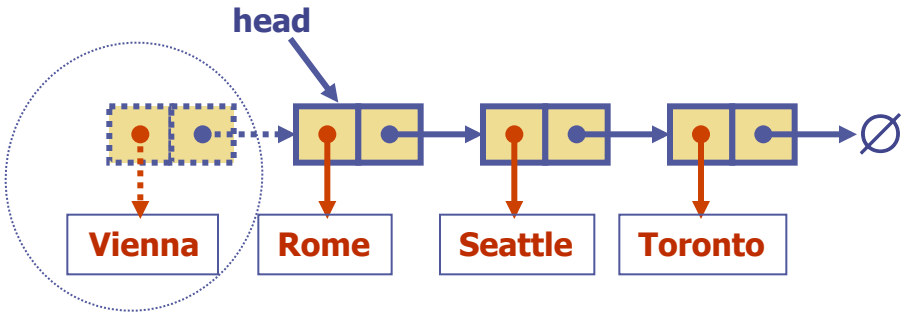



```
…
void addFirst(const char* s)
{
    Node* newNode = new Node;
    newNode->element = s;
    newNode->next = head;
    head = newNode;
}
```

1. Cấp phát nút mới
2. Đưa dữ liệu vào nút mới
3. Cho nút mới trỏ tới head cũ
4. Sửa head để trỏ tới nút mới

SLinkedList.addFirst – cài đặt

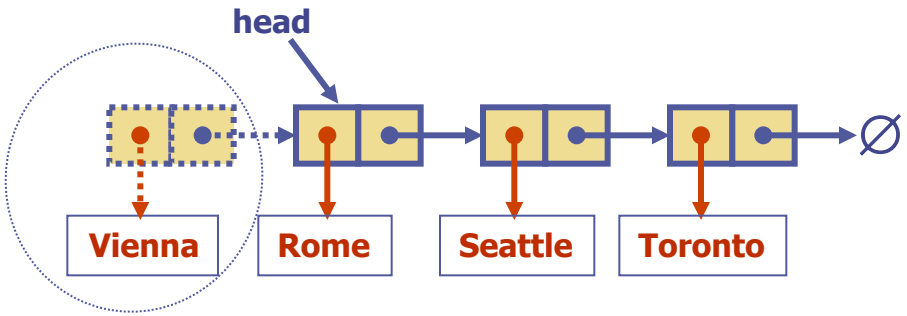
```
list.addFirst("Vienna");
```




```
…
void addFirst(const char* s)
{
    Node* newNode = new Node(s, head);
    head = newNode;
}
```

1. Cấp phát nút mới
2. Đưa dữ liệu vào nút mới
3. Cho nút mới trỏ tới head cũ
4. Sửa head để trỏ tới nút mới

Câu hỏi



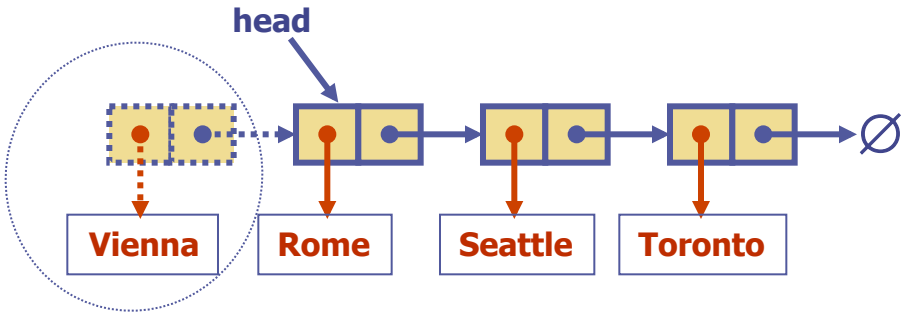

```
…
void addFirst(const char* s)
{
    Node* newNode = new Node(s, head);
    head = newNode;
}
```

Tại sao không thể thay bằng lệnh sau?

```
Node newNode(s, head);
```

SLinkedList.addFirst – cài đặt

```
list.addFirst("Vienna");
```

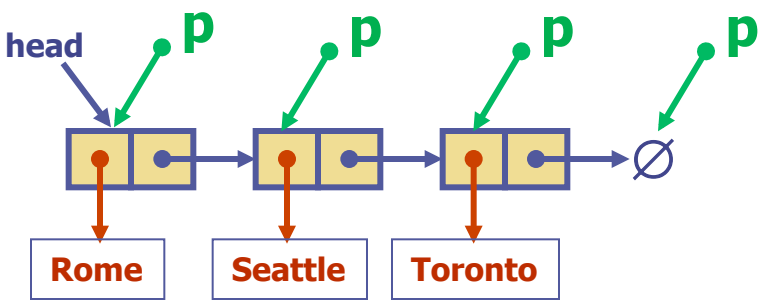



```
…
void addFirst(const char* s)
{
   head = new Node(s, head);
}
```

1. Cấp phát nút mới
2. Đưa dữ liệu vào nút mới
3. Cho nút mới trỏ tới head cũ
4. Sửa head để trỏ tới nút mới

Duyệt danh sách liên kết

1. P bắt đầu từ head
2. Nếu p không null thì
    1. Xử lý dữ liệu tại p
    2. Đẩy p tới nút tiếp theo
    3. Quay lại 2



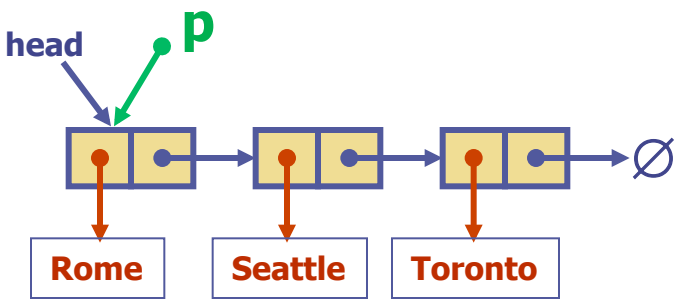
Rome **Seattle** **Toronto** done

```
list.print(); // in danh sách
…
void print() {
   for (Node *p = head; p != NULL; p = p->next) {
       cout << p->element << ".";
   }
}
```

Câu hỏi

Tại sao lại cần p?

Sao không dùng chính head để chạy con trỏ?



Rome Seattle Toronto

```
list.print(); // in danh sách
…
void print() {
   for (Node *p = head; p != NULL; p = p->next) {
       cout << p->element << ".";
   }
}
```

Xóa nút đứng đầu danh sách

SLinkedList list;….

const char* s
= list.removeFirst(); //Vienna

1. Sửa head để trỏ tới nút thứ hai trong danh sách

Node* old = head;

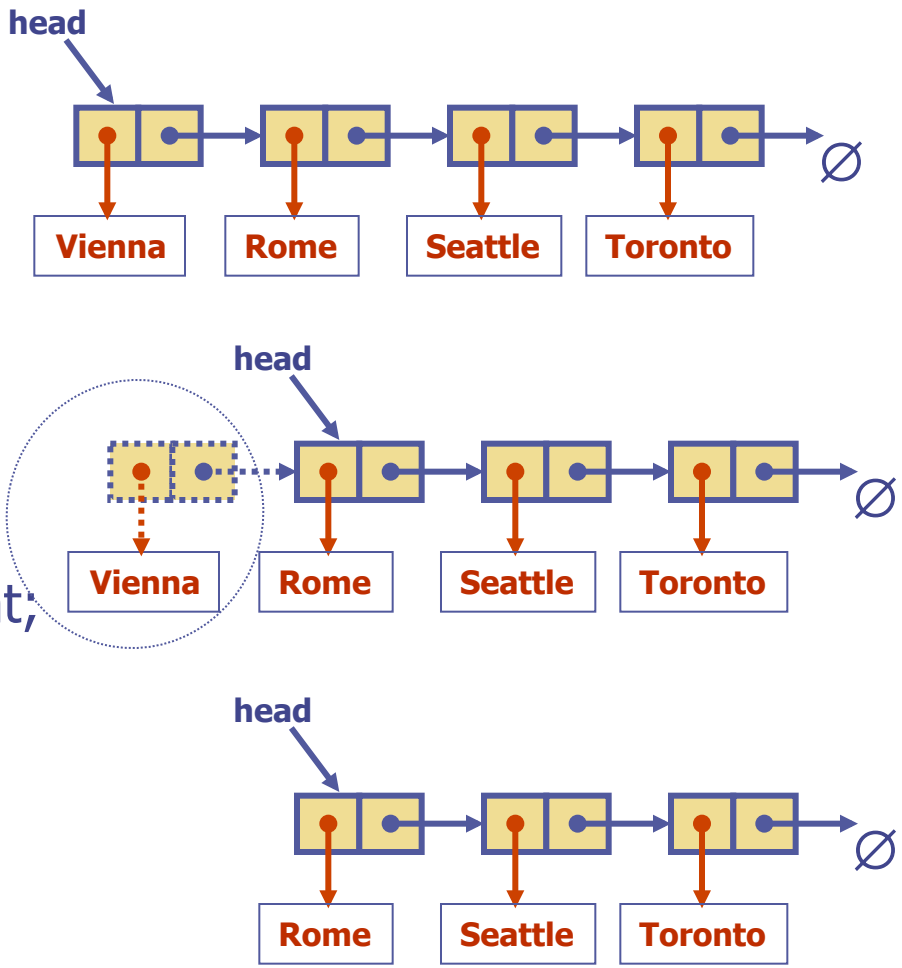
const char* old_element = old.element;

head = head.next;

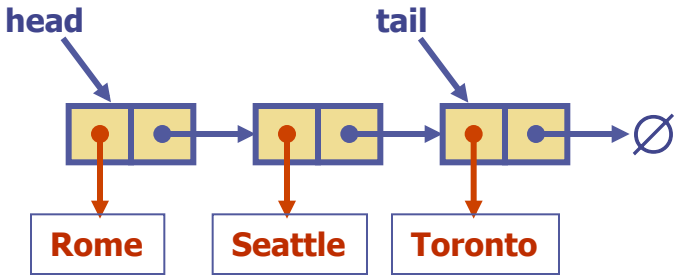
1. Giải phóng nút đứng đầu

delete old;

return old_element;



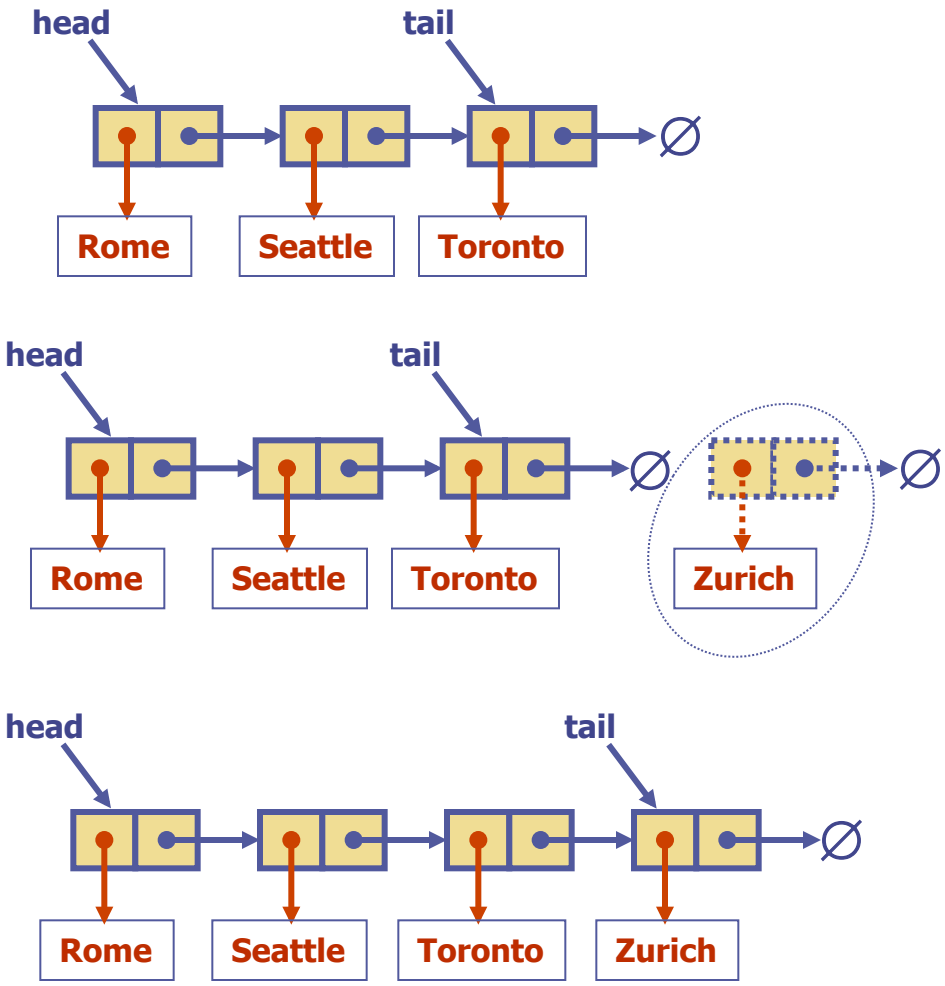
DS liên kết với con trỏ chặn cuối




```
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 ...
}
```

Thêm vào cuối danh sách

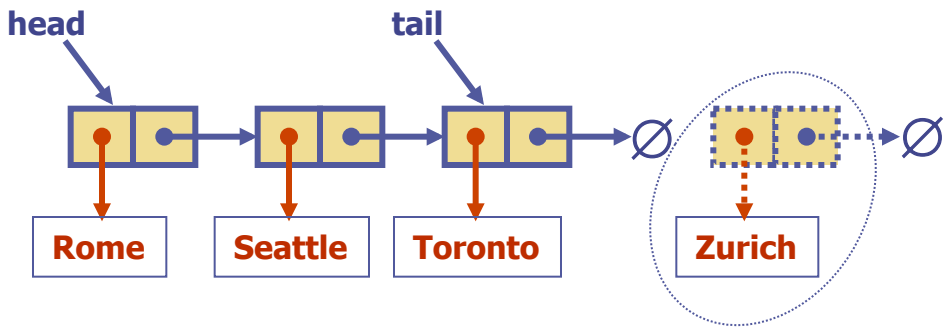
1. Cấp phát nút mới
2. Lắp dữ liệu
3. Cho nút mới trỏ tới null
4. Cho nút cuối trỏ tới nút mới
5. Sửa tail để trỏ tới nút mới – đuôi mới



addLast(const char* s)

1. Cấp phát nút mới
2. Lắp dữ liệu
3. Cho nút mới trỏ tới null

4. Cho nút cuối trỏ tới nút mới
5. Sửa tail để trỏ tới nút mới – đuôi mới

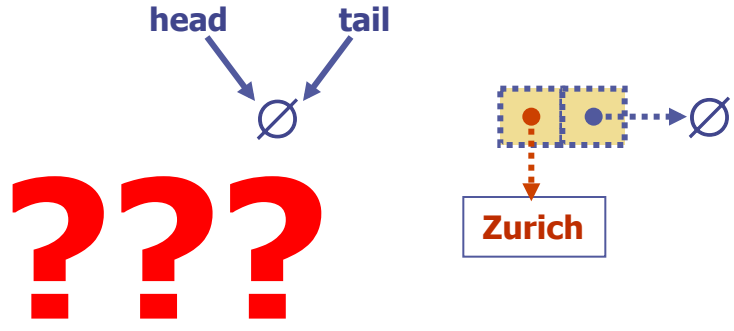


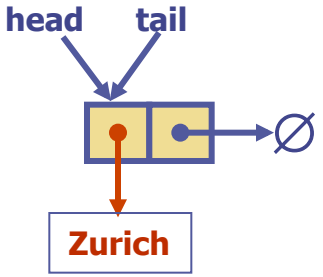

```
struct SLinkedListWithTail {
    Node* head; // head node
    Node* tail; // tail node of the list

    void addLast(const char *s) {
       Node* newNode = new Node(s, NULL);      //1,2,3
       tail->next = newNode;                   //4
       tail = newNode;                         //5
    }
...
```

addLast(const char* s)

1. Cấp phát nút mới
2. Lắp dữ liệu
3. Cho nút mới trỏ tới null

4. **Cho nút cuối trỏ tới nút mới**
5. Sửa tail để trỏ tới nút mới – đuôi mới

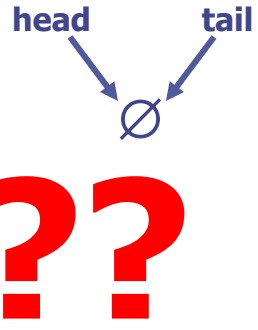


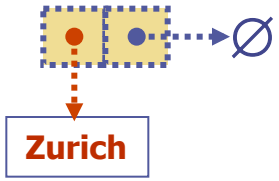


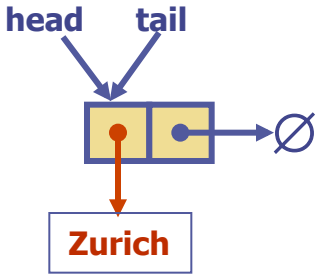

```
struct SLinkedListWithTail {
    Node* head; // head node
    Node* tail; // tail node of the list

    void addLast(const char *s) {
       Node* newNode = new Node(s, NULL);      //1,2,3
       tail->next = newNode;                   //4
       tail = newNode;                         //5
    }
...
```

addLast(const char* s)

1. Cấp phát nút mới
2. Lắp dữ liệu
3. Cho nút mới trỏ tới null
4. **Nếu list rỗng thì trỏ head tới nút mới**
   nếu không, cho nút cuối trỏ tới nút mới
5. Sửa tail để trỏ tới nút mới – đuôi mới



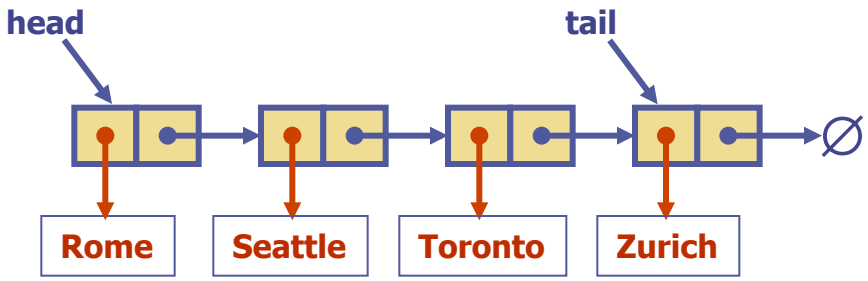




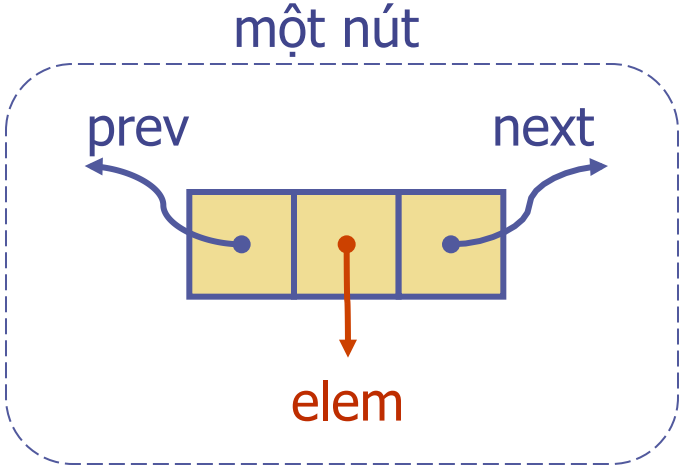

```
struct SLinkedListWithTail {
    Node* head; // head node
    Node* tail; // tail node of the list

    void addLast(const char *s) {
       Node* newNode = new Node(s, NULL);      //1,2,3
       if (head == NULL) head = newNode;       //4
       else tail->next = newNode;
       tail = newNode;                         //5
    }
```

Xóa ở cuối danh sách

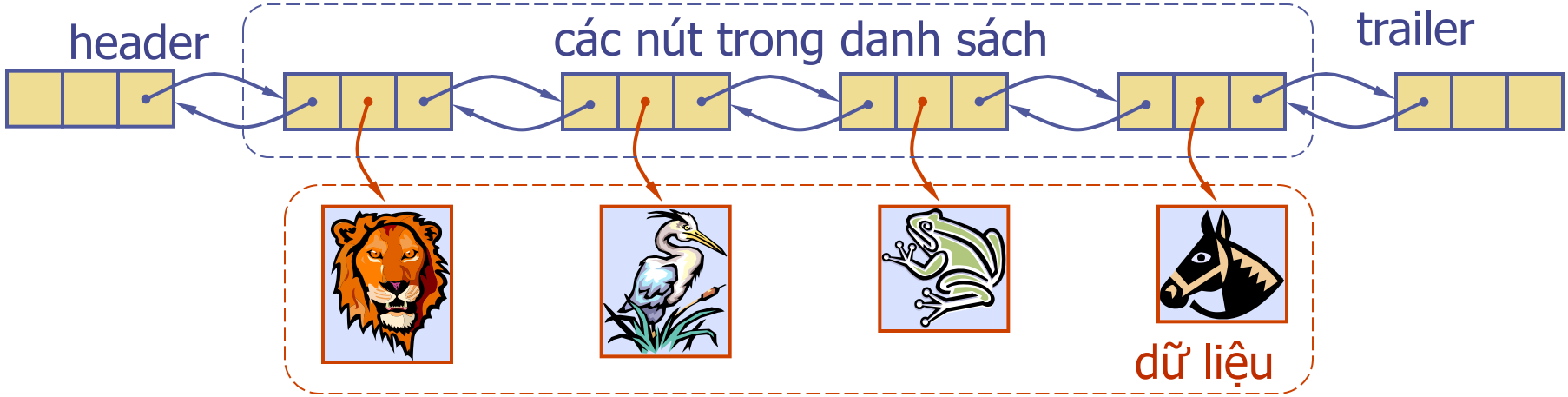
- Đối với danh sách liên kết đơn, không có cách nào ngoài việc lấy con trỏ chạy từ đầu đến nút ngay trước tail
- Không hiệu quả!



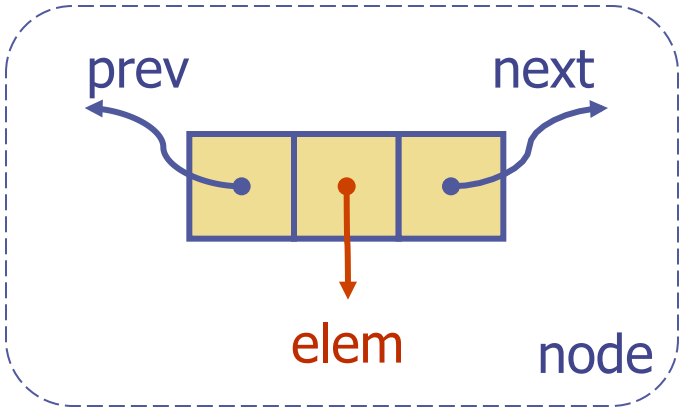
Danh sách liên kết đôi
Doubly Linked List



- Thuận tiện hơn!
- Mỗi nút chứa:
  - Dữ liệu
  - Con trỏ tới nút liền trước
  - Con trỏ tới nút liền sau
- Danh sách liên kết chứa:
  - header và trailer trỏ tới nút đặc biệt chặn đầu và cuối danh sách



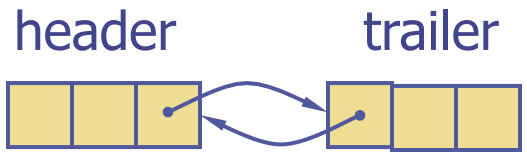
DNode struct




```
/* Node of a doubly linked list of strings */
struct DNode {
   const char* element;
   DNode *next, *prev; // Pointers to next and previous node

   /* Initialize a node. */
   DNode(const char* e = NULL, DNode* p = NULL, DNode *n = NULL)
   {
      element = e;
      prev = p;
      next = n;
   }
};
```

Cài đặt doubly linked list DList

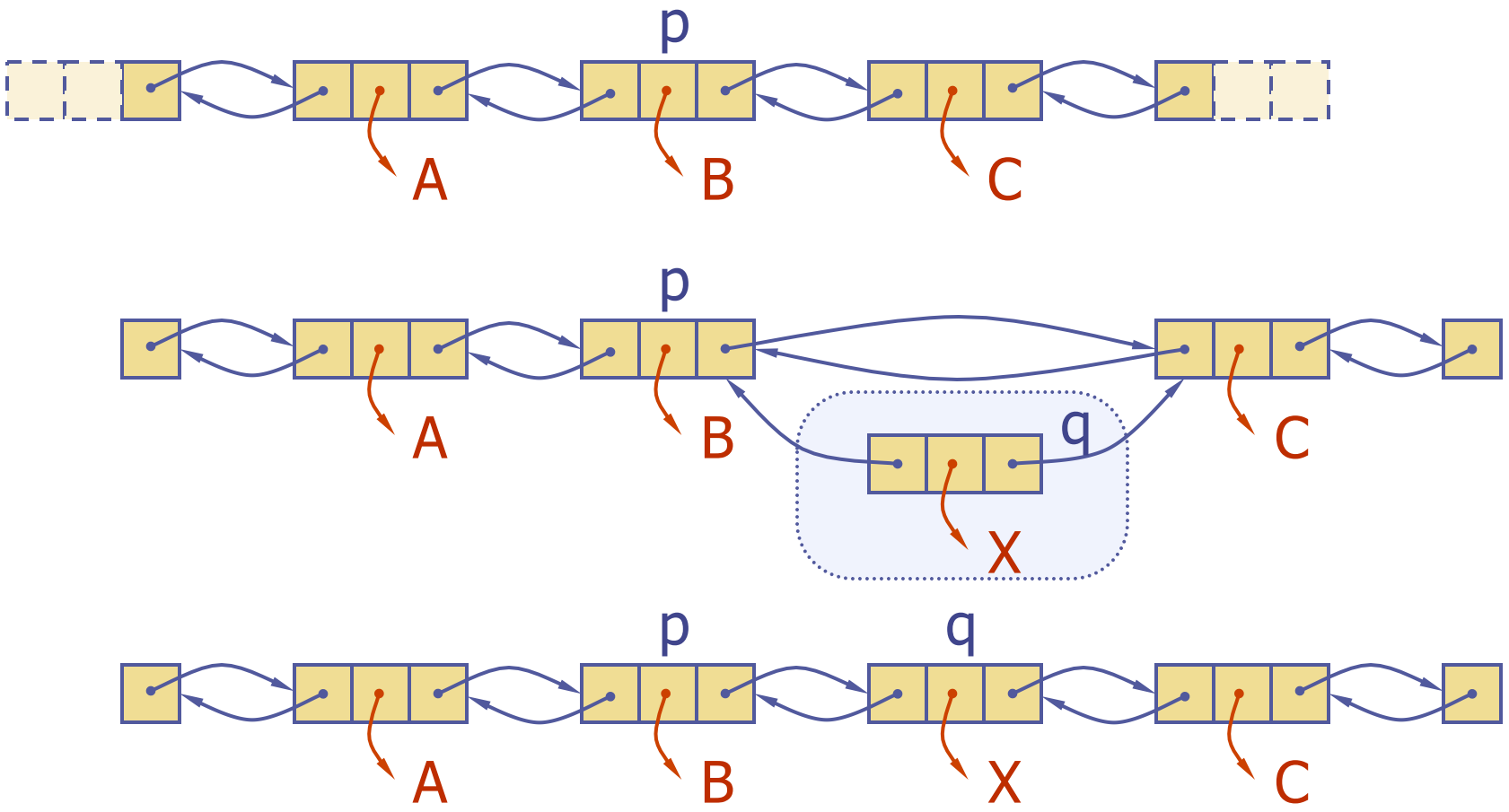
```
struct DList {
    DNode header, trailer;// sentinels

    /* Default constructor that creates an empty list */
    DList() {
        header.next = &trailer;
        trailer.prev = &header;
    }
```



...

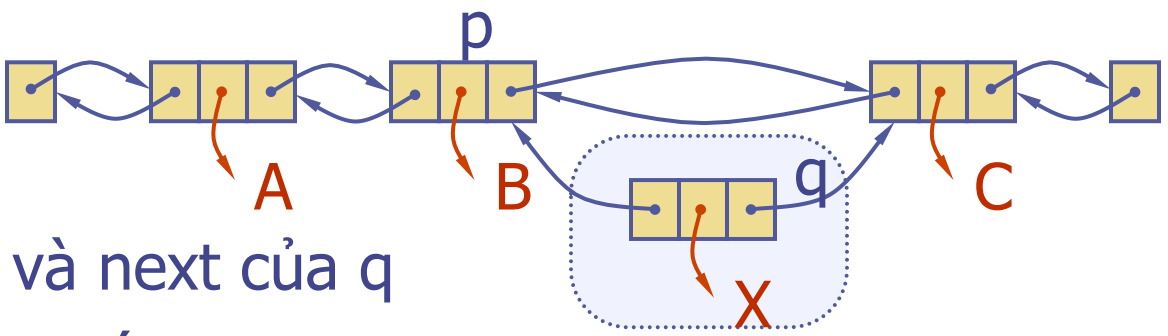
insertAfter – chèn vào giữa

hàm insertAfter(p, X), chèn X vào sau p và trả về địa chỉ nút mới q



insertAfter – cài đặt

hàm insertAfter(p, X):



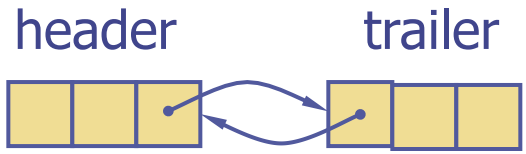
1. Tạo nút mới q
2. Lắp dữ liệu, nối prev và next của q
3. Nối prev của nút sau p với q
4. Nối next của p với q

```
DNode insertAfter(DNode* p, const char* s) {
    DNode* newNode = new DNode(s, p, p->next);
    p->next->prev = newNode;
    p->next = newNode;
    return newNode;
}
```

Câu hỏi

- Trường hợp danh sách rỗng, thêm vào đầu, thêm vào cuối danh sách thì sao?

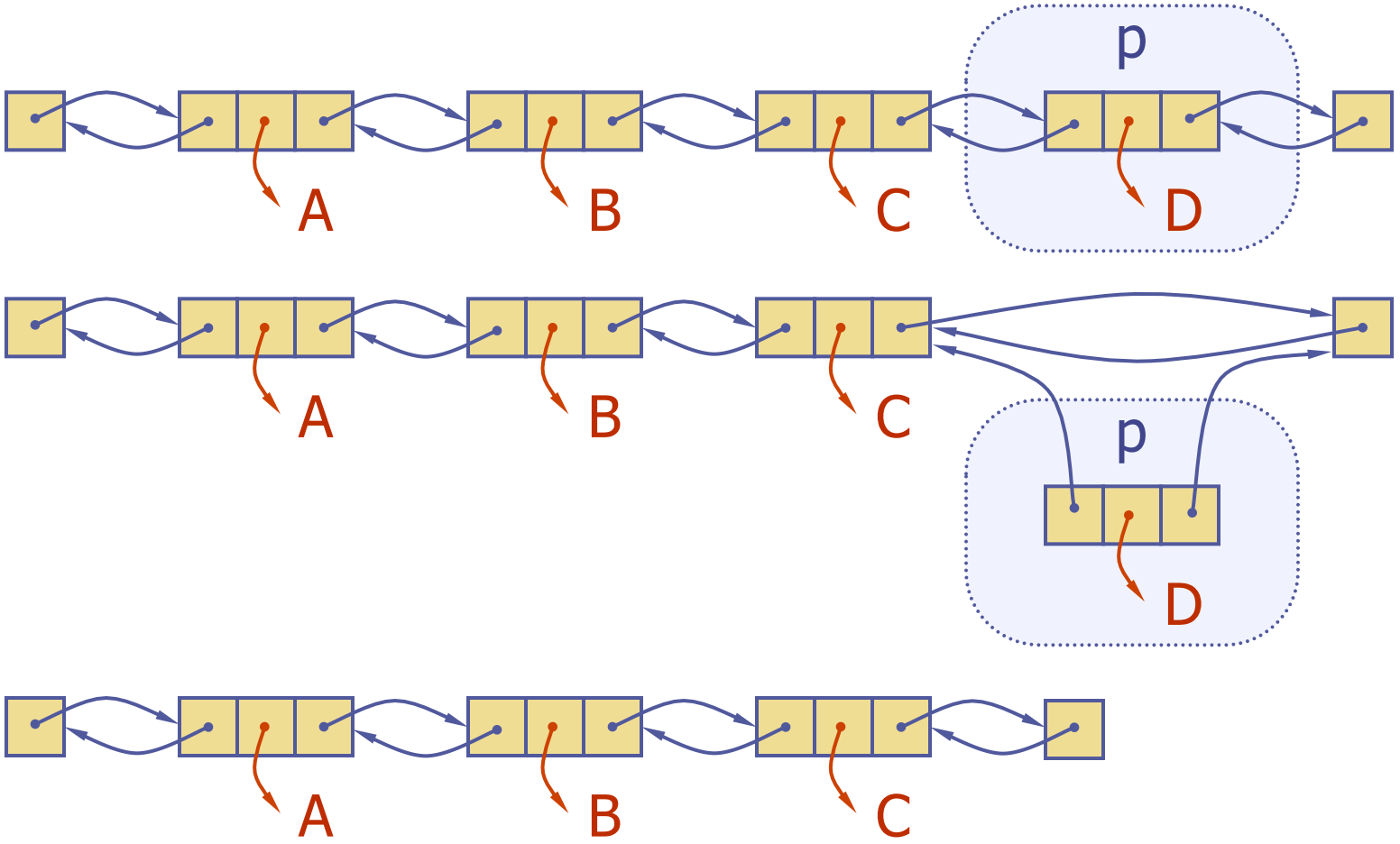
```
DNode insertAfter(DNode* p, const char* s) {
    DNode* newNode = new DNode(s, p, p->next);
     p->next->prev = newNode;
     p->next = newNode;
    return newNode;
}
```



Do luôn có hai nút header chặn đầu và trailer chặn cuối, nên danh sách thực tế không bao giờ rỗng, không bao giờ cho phép chèn vào trước header hoặc sau trailer
→ Không cần xử lý trường hợp đặc biệt cho thiết kế này

Xóa

- remove(p) xóa nút mà p trỏ tới



Destructor

- Destructor của List:
  - **Phải** giải phóng các nút hiện nằm trong danh sách
- Destructor của Node:
  - **Không được** giải phóng dữ liệu (elements)
- Ai cấp phát cái gì thì có trách nhiệm giải phóng cái đó
  - List cấp phát các Node, do đó List phải tự giải phóng các Node
    - Lập trình viên nào delete một list, sau đó lại truy nhập vào một node trong list đó thì tự chịu hậu quả.
  - Node không cấp phát element, nó không thể biết dữ liệu đó có phải dữ liệu động hay không và hiện có còn ai đang dùng đến → không được giải phóng element