HK I, 2012-2013

Bài 5: Danh sách liên kết

Giảng viên: Hoàng Thị Điệp
Khoa Công nghệ Thông tin – Đại học Công Nghệ

3 cách để liên kết dữ liệu

- Mảng: tập hợp các phần tử cùng kiểu
- struct/class: tập hợp các thành phần có kiểu (có thể) khác nhau
- Con trỏ

Các KDLTT đã học

- KDLTT danh sách
  - Phép toán
    - insert
    - delete
    - append
    - at
    - length
    - empty
  - Cài đặt
    - mảng tĩnh
    - mảng động
- KDLTT tập động
  - Phép toán
    - insert
    - delete
    - search
    - max
    - min
    - empty
    - length
  - Cài đặt
    - mảng động không được sắp
    - mảng động được sắp

Nhận xét

- Độ phức tạp khi cài đặt danh sách bằng mảng
  - truy cập: getElement(A, i)
  - cập nhật: update(A, i)
  - xen thêm giá trị x: insert(A, i, x)
  - xóa bớt: del(A, i)
- Danh sách liên kết giúp insert và del hiệu quả hơn

KDLTT danh sách

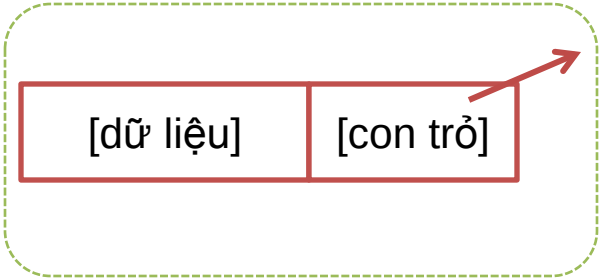
- Cài bằng mảng
  - at: O(1)
  - insert: O(N)
  - delete: O(N)
- Cài bằng danh sách liên kết

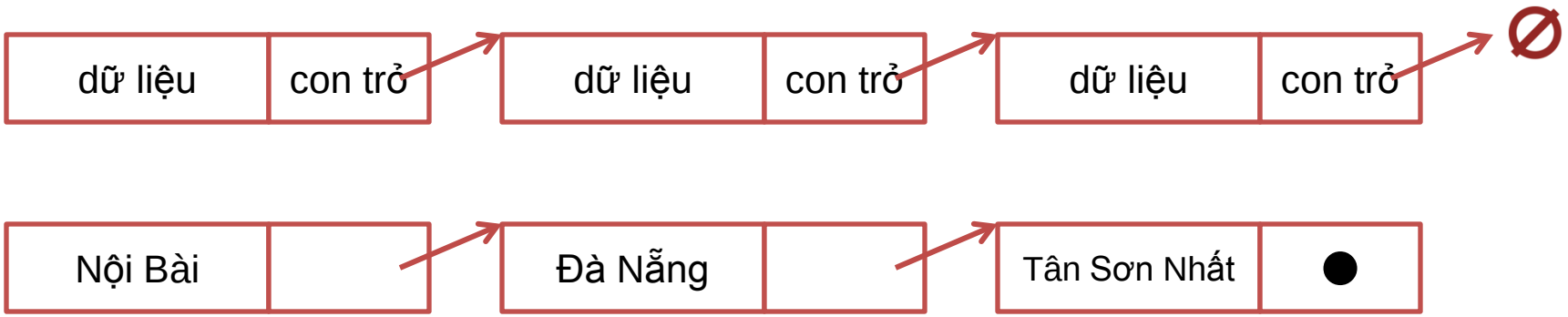
So sánh

	Mảng	**Danh sách liên kết**
Giống	Các phần tử có cùng kiểu và có thứ tự	
Khác	Bố cục logic giống với bố cục vật lý trong bộ nhớ máy tính	Bố cục logic không cần phải giống với bố cục vật lý

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





DSLK bằng C++

- 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;
```

DSLK bằng C++

- Có thể sử dụng thêm con trỏ đuôi tail để các thao tác trên DSLK được thuận lợi
  - danh sách rỗng

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

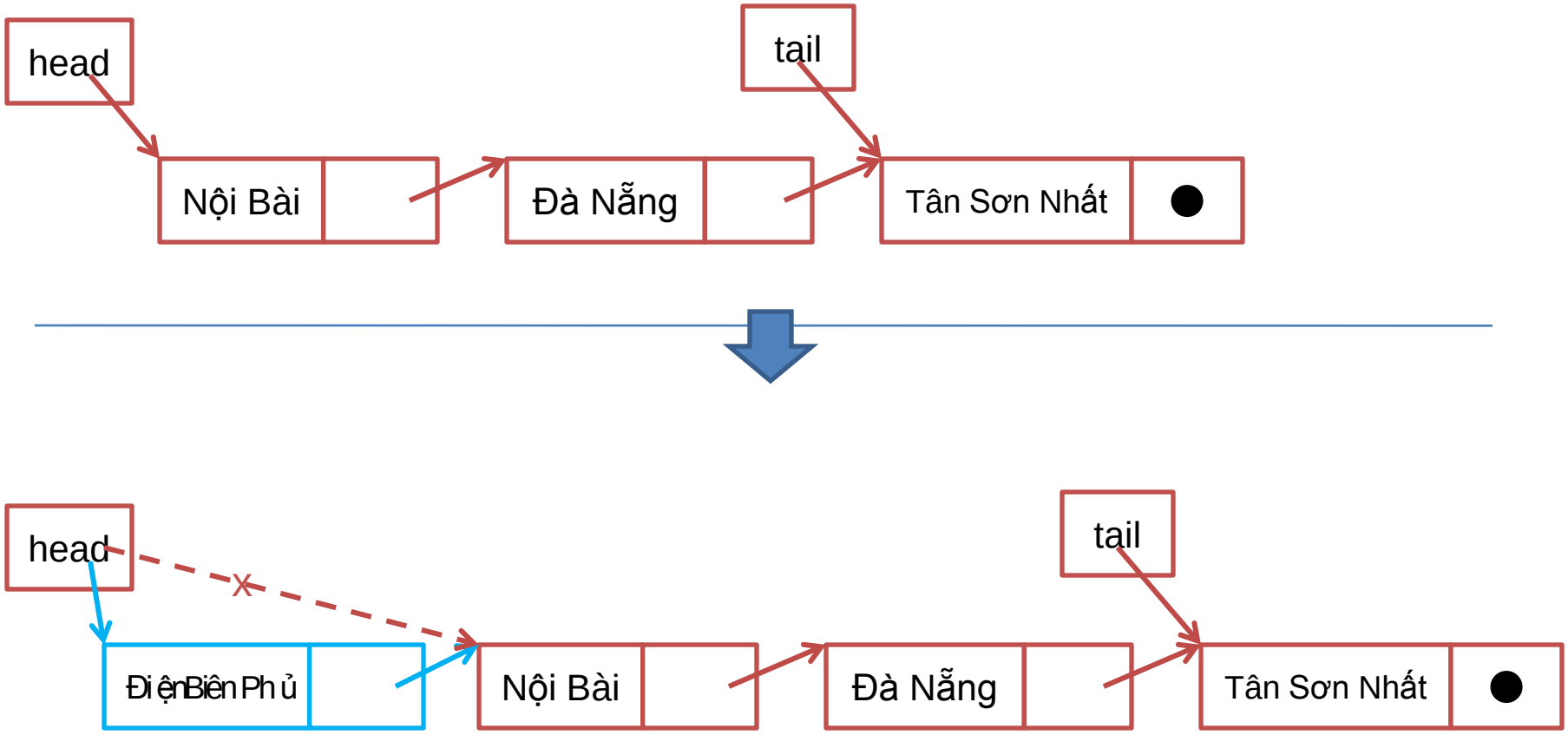
```
head = tail = NULL;
```

```
typedef int Item;

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

struct SList{
        Node * head;
        Node * tail;
        long size;

        SList();
        ~SList();

        Node* findPrevious(Node* p);
        Node* addFirst(const Item& v);
        Node* addLast(const Item& v);
        Node* insertAfter(Node* p, const Item& v);
        Node* insertBefore(Node* p, const Item& v);
        void removeFirst();
        void remove(Node*& p);
        void print();
};
```

Các phép toán trên DSLK

- Thêm dữ liệu vào danh sách
  - Thêm vào đầu danh sách: addFirst(v)
  - Thêm vào sau nút p: insertAfter(p, v)
  - Thêm vào trước nút p: insertBefore(p, v)
  - Thêm vào cuối danh sách: addLast(v)
- 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(p)
- Duyệt danh sách: traverse()

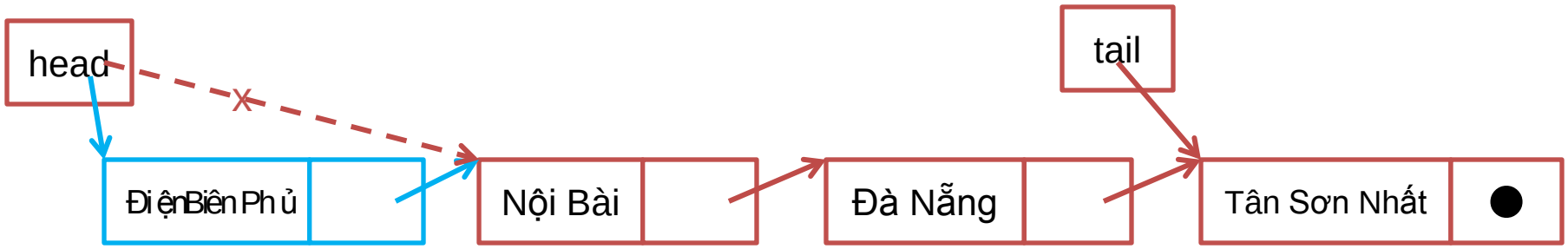
Thêm vào đầu danh sách



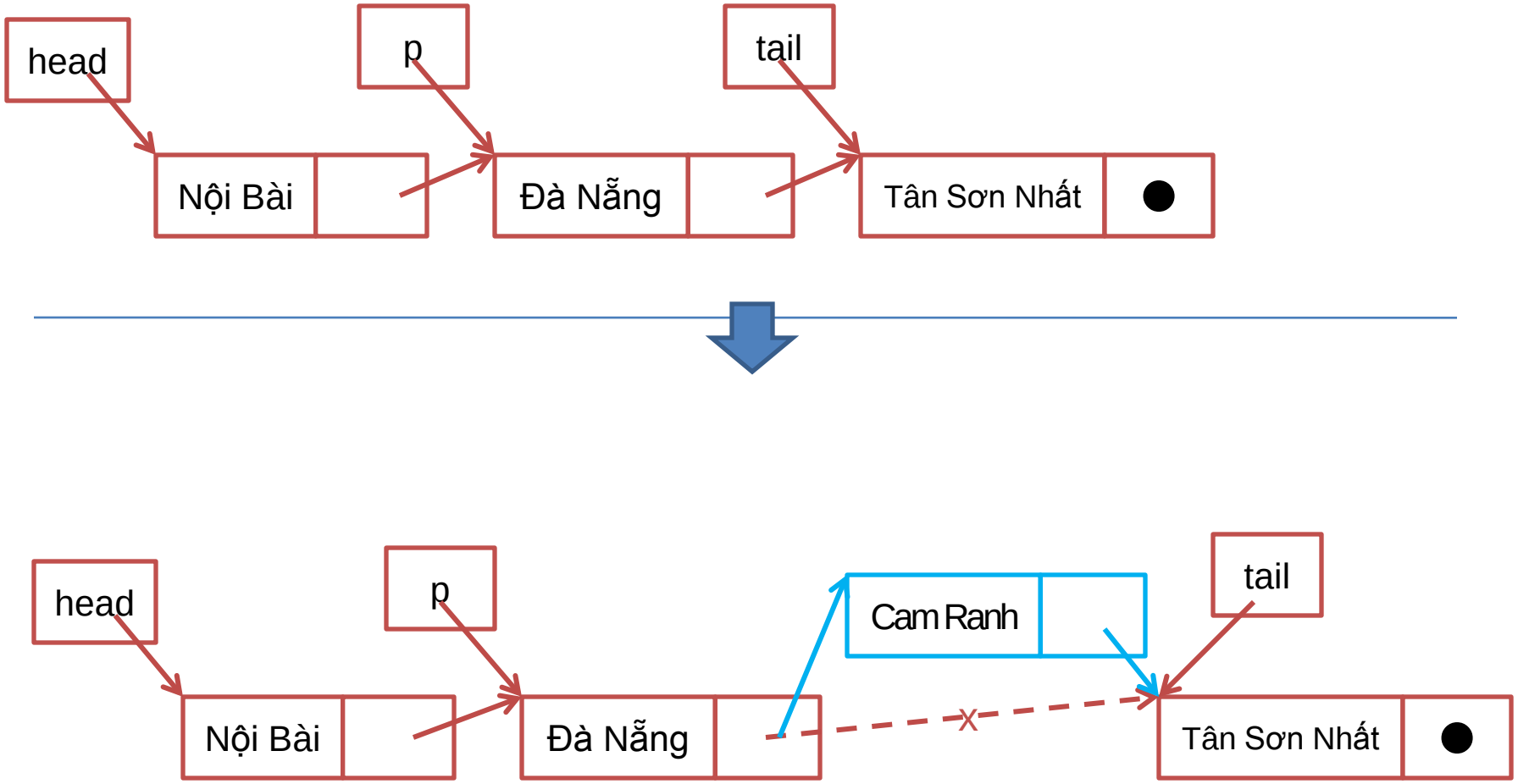
addFirst(v)

Algorithm ***addFirst(v)***
Input **dữ liệu v cần thêm vào đầu danh sách**
Output

q* ← new *Node()	{tạo nút mới}
(*q).data* ← *v	{nút mới chứa dữ liệu v}
(*q).next* ← *head	{nút mới chứa con trỏ đến nút head cũ}
head* ← *q	{trỏ head đến nút mới}
size* ← *size + 1	{tăng biến đếm nút}
cập nhật tail	



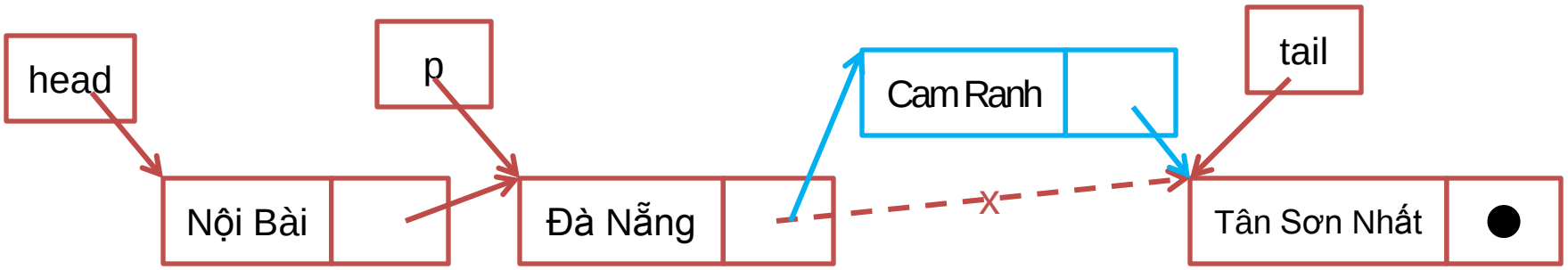
Thêm vào sau nút p



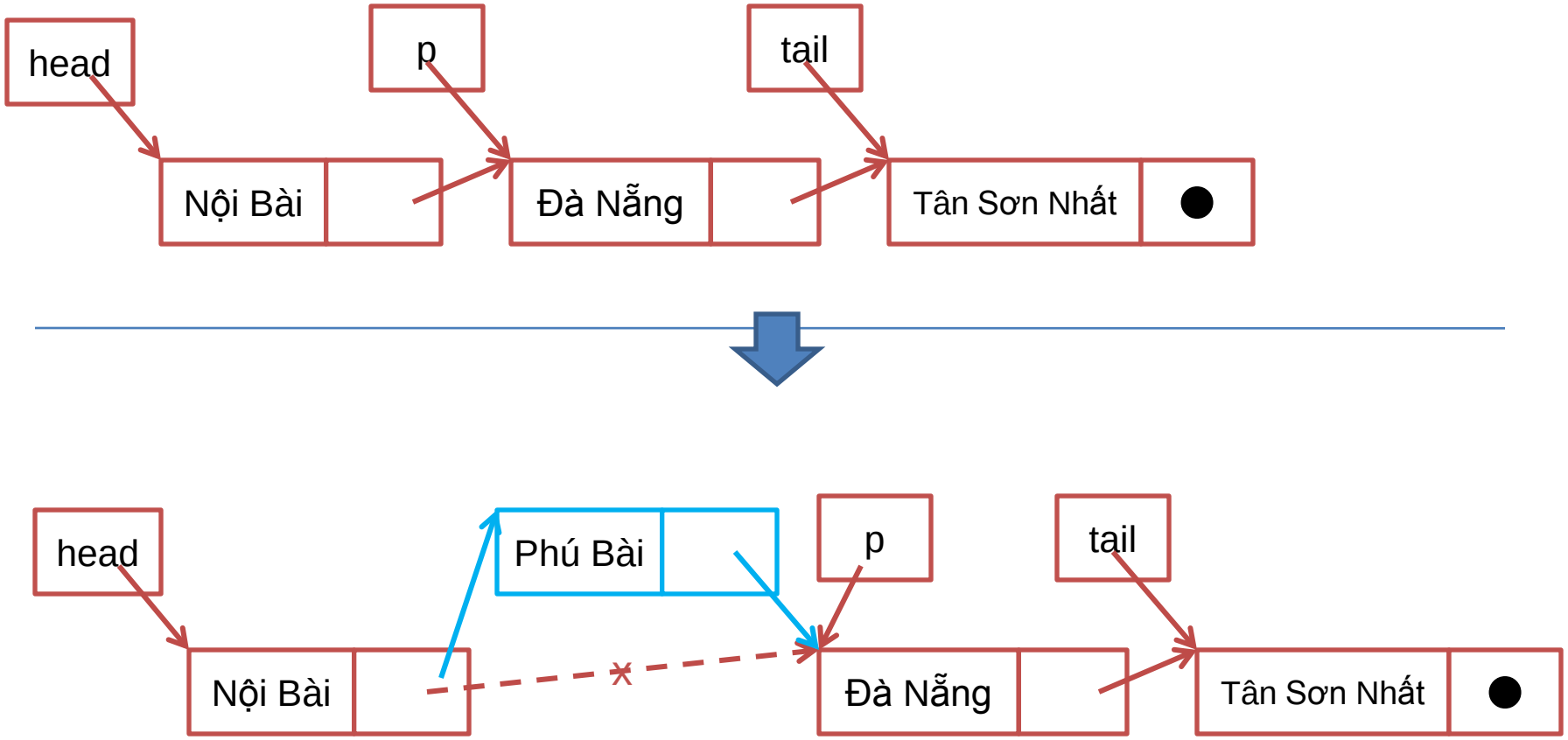
insertAfter(p, v)

Algorithm ***insertAfter(p, v)***
Input dữ liệu v cần thêm vào sau nút p trong danh sách
Output

q* ← new *Node()	{tạo nút mới}
(*q).data* ← *v	{nút mới chứa dữ liệu v}
(*q).next* ← *(*p).next	{nút mới chứa con trỏ đến nút sau p}
(*p).next* ← *q	{nút p chứa con trỏ đến nút mới}
size* ← *size + 1	{tăng biến đếm nút}
cập nhật tail	

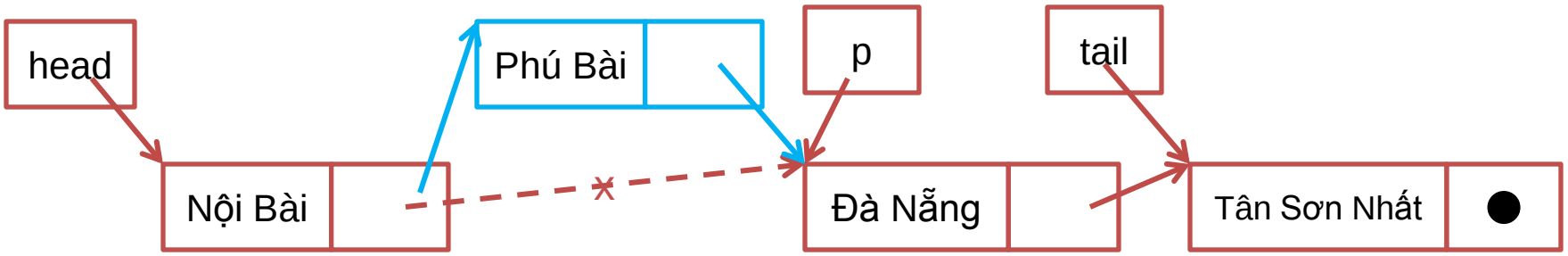


Thêm vào trước nút p

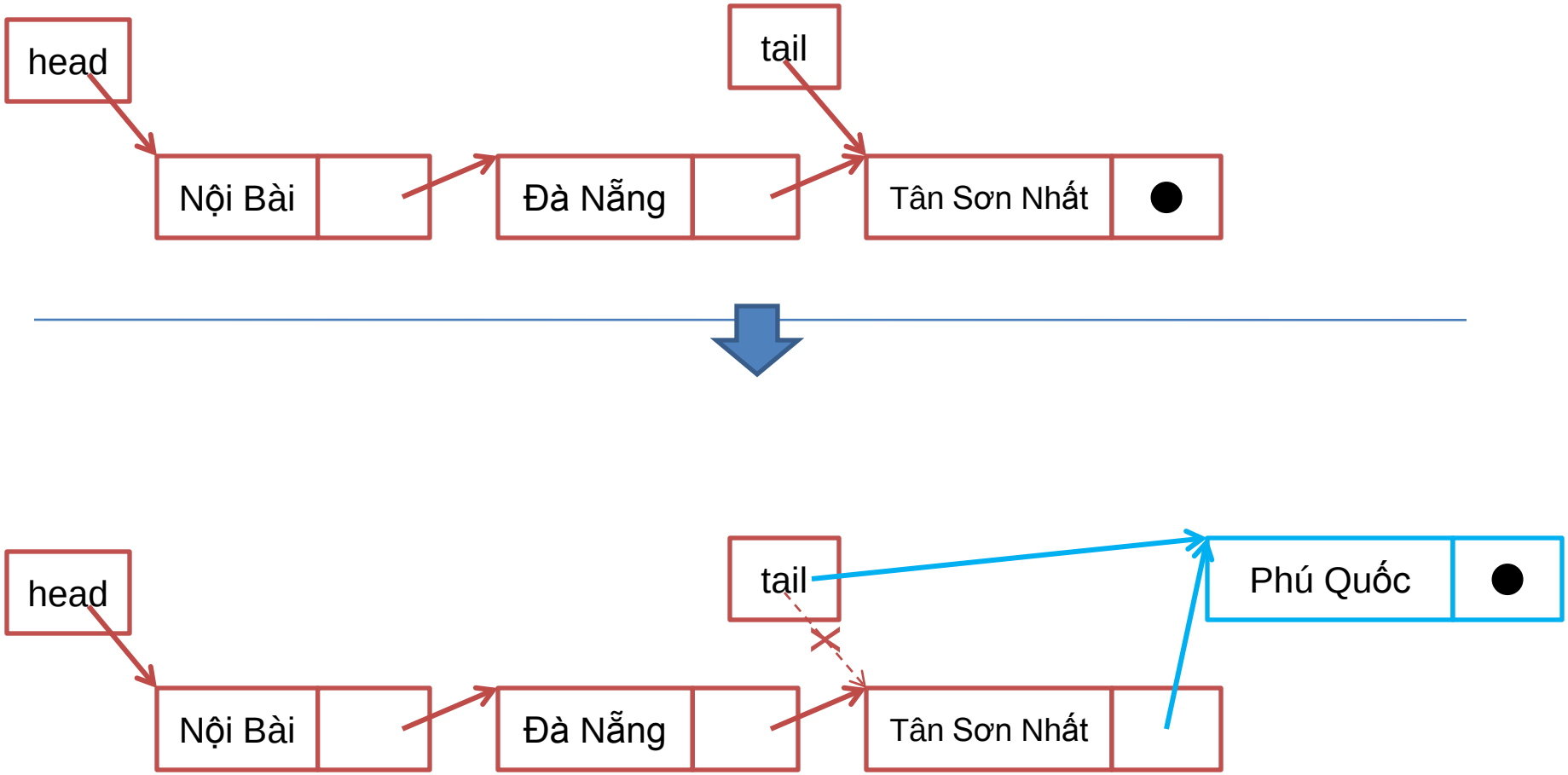


insertBefore(p, v)

Algorithm ***insertBefore(p, v)***
Input dữ liệu v cần thêm vào trước nút p trong danh sách
Output
if *p = head* **then**
addFirst(v)
else
tìm nút pre liền trước p
insertAfter(pre, v)



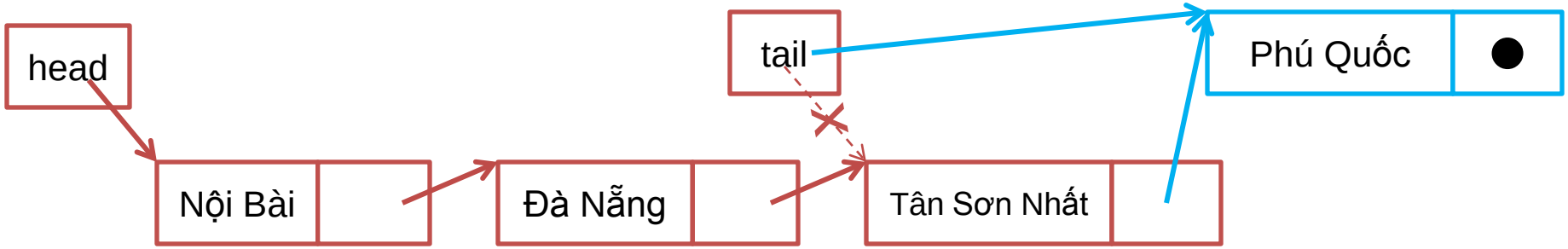
Thêm vào cuối danh sách



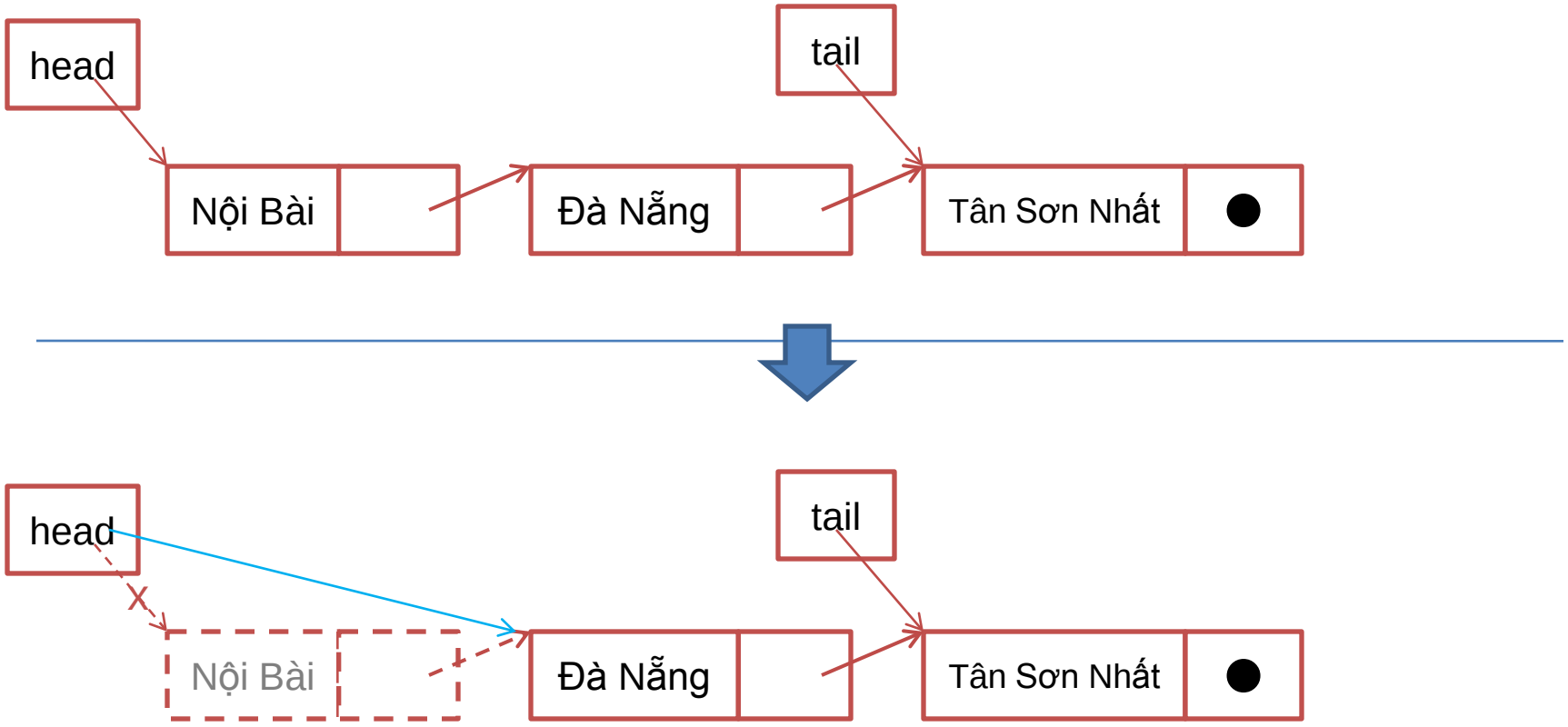
addLast(v)

Algorithm ***addLast(v)***
Input **dữ liệu v cần thêm vào cuối trong danh sách**
Output

q ← **new** ***Node()***	{tạo nút mới}
(*q).data ← ***v***	{nút mới chứa dữ liệu v}
(*q).next ← ***NULL***	{nút mới chứa con trỏ NULL}
(*tail).next ← ***q***	{nút tail cũ chứa con trỏ đến nút mới}
tail ← ***q***	{tail trỏ đến nút mới}
size ← ***size + 1***	{tăng biến đếm nút}

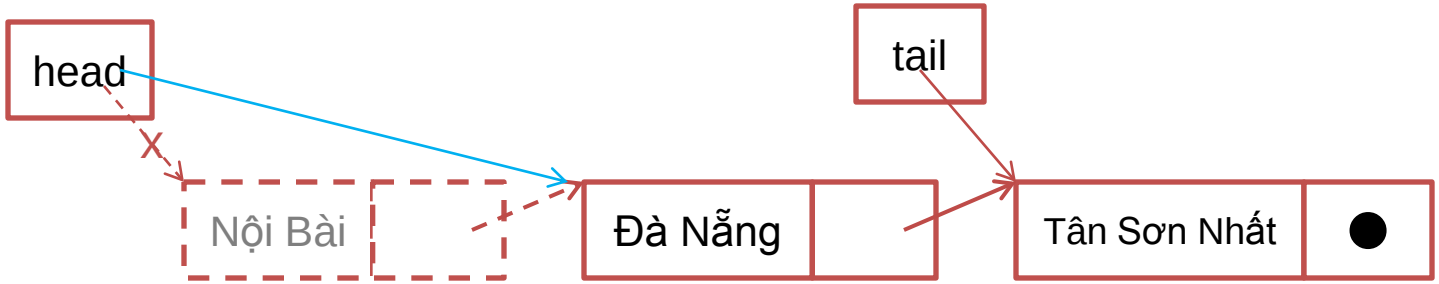


Xóa nút đầu danh sách

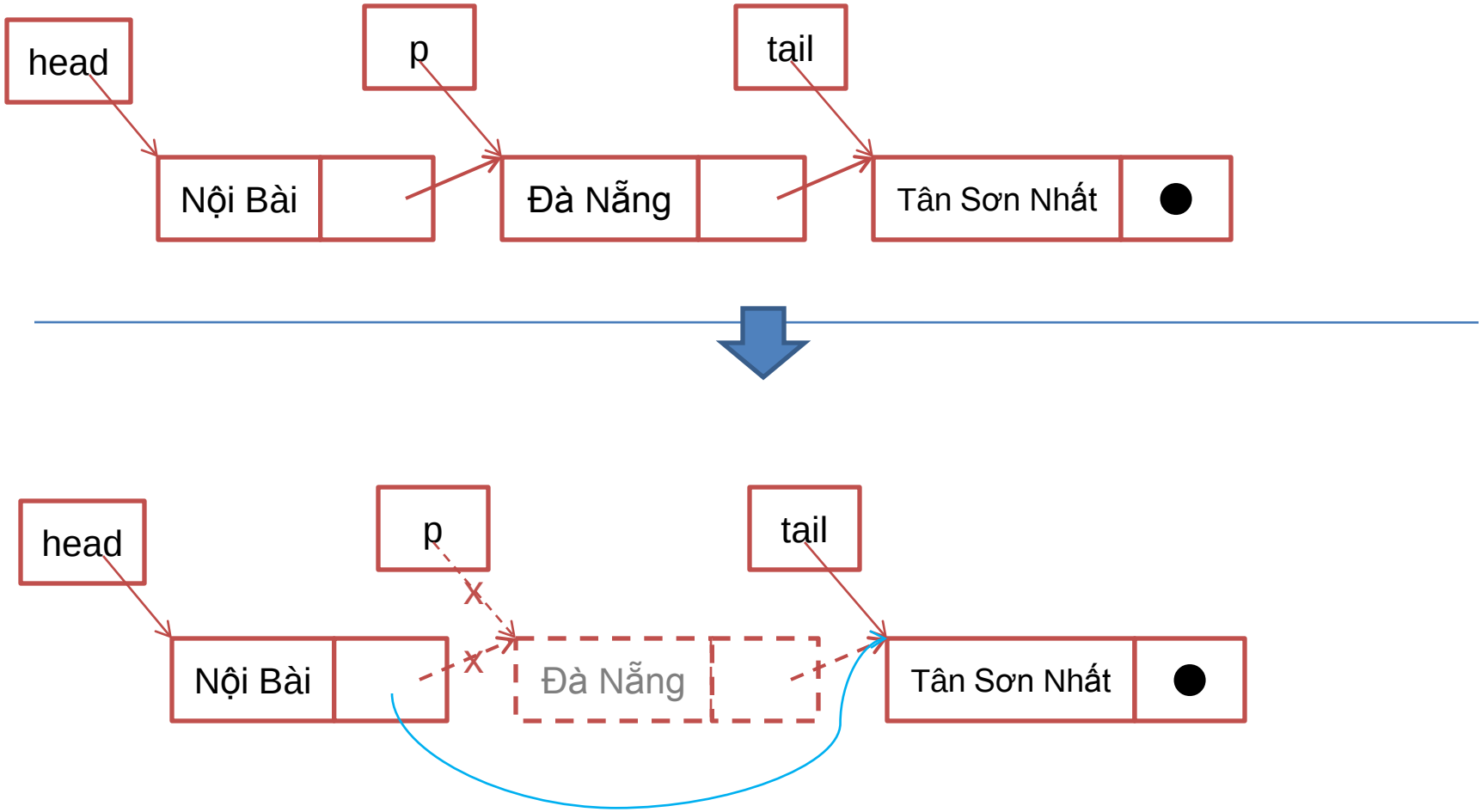


removeFirst()

Algorithm ***removeFirst()***
Input
Output
if *head* = **null then**
báo lỗi: danh sách rỗng
t ← *head*
head ← *(*head).next* {trỏ head đến nút sau nó}
delete *t* {giải phóng vùng nhớ trỏ bởi head cũ}
size ← *size - 1* {giảm biến đếm nút}
cập nhật tail

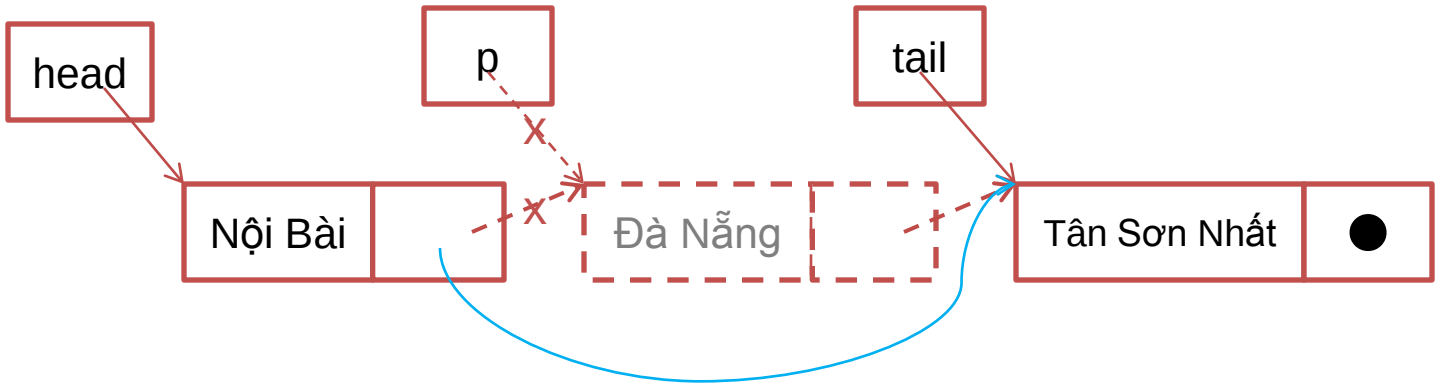


Xóa nút không phải đầu danh sách



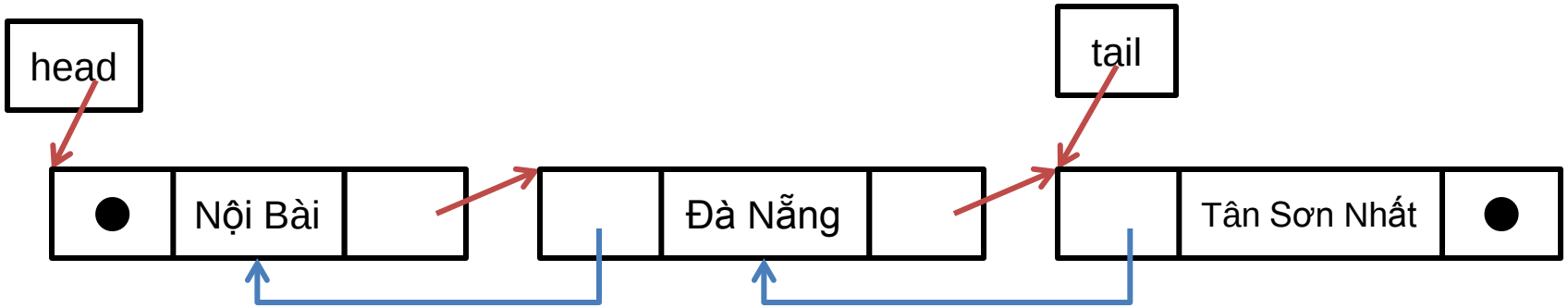
remove(p)

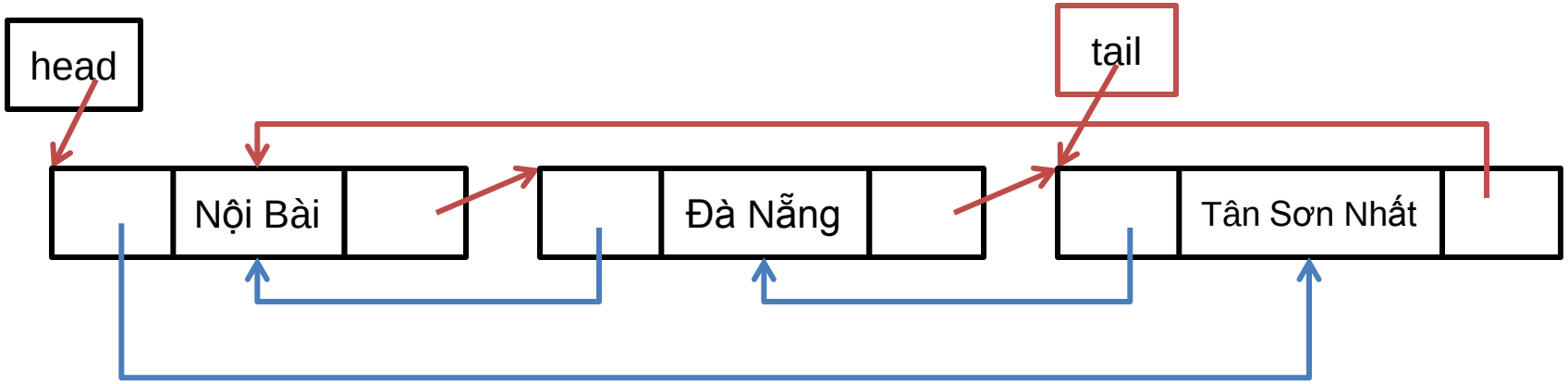
Algorithm ***remove(p)***
Input *nút p cần xóa khỏi danh sách*
Output
if *p=head* **then**
removeFirst()
else
tìm nút pre liền trước p
*(*pre).next ← (*p).next* {pre chứa con trỏ đến nút sau p}
delete *p* {giải phóng vùng nhớ trỏ bởi p}
size ← size - 1 {giảm biến đếm nút}
cập nhật tail



Các dạng DSLK

- DSLK đơn
  - singly linked list, uni-directional list, one-way list
- DSLK kép
  - doubly linked list, bi-directional list
- DSLK vòng tròn
  - ring list


head
tail
Nội Bài
Đà Nẵng
Tân Sơn Nhất


head
tail
Nội Bài
Đà Nẵng
Tân Sơn Nhất

Cài đặt danh sách bởi DSLK

- Sinh viên tự nghiên cứu chương 5.4, 5.5.

Câu hỏi

- Hãy mô tả cấu trúc của DSLK được định nghĩa bởi đoạn mã sau.

```
typedef struct Node ListNode;
struct Node{
        int data;
        ListNode *next;
}

typedef struct FirstNode *LinkedList;
struct FirstNode{
        ListNode *first;
}
```

```
typedef struct Node ListNode;
struct Node{
        int data;
        ListNode* next;
}

typedef struct FirstNode* LinkedList;
struct FirstNode{
        ListNode* first;
}
```

Chuẩn bị bài tới

- Đọc chương 6, 7 giáo trình.