Danh sách liên kết (Linked Lists)

Nguyễn Mạnh Hiển
hiennm@tlu.edu.vn

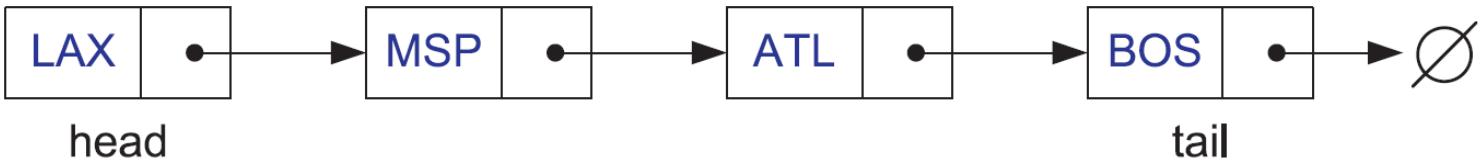
Nội dung

1. Danh sách liên kết
2. Danh sách liên kết đơn
3. Danh sách liên kết đôi
4. Danh sách liên kết vòng tròn

1. Danh sách liên kết

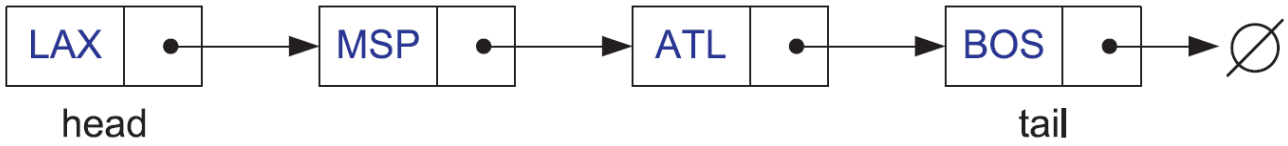
Danh sách liên kết

- Là một tập nút liên kết với nhau theo trật tự tuyến tính
- Mỗi nút chứa:
  - một phần tử
  - một hoặc nhiều liên kết tới các nút lân cận
- Các nút nằm rải rác trong bộ nhớ máy tính (trong khi các phần tử của mảng và vector nằm liên tục)

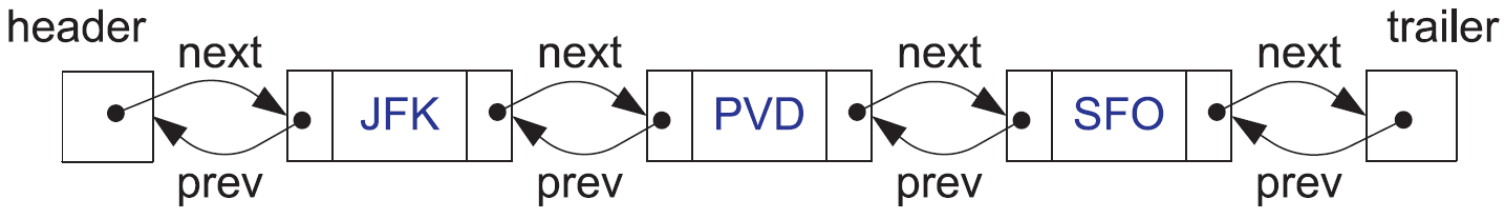


Các kiểu danh sách liên kết

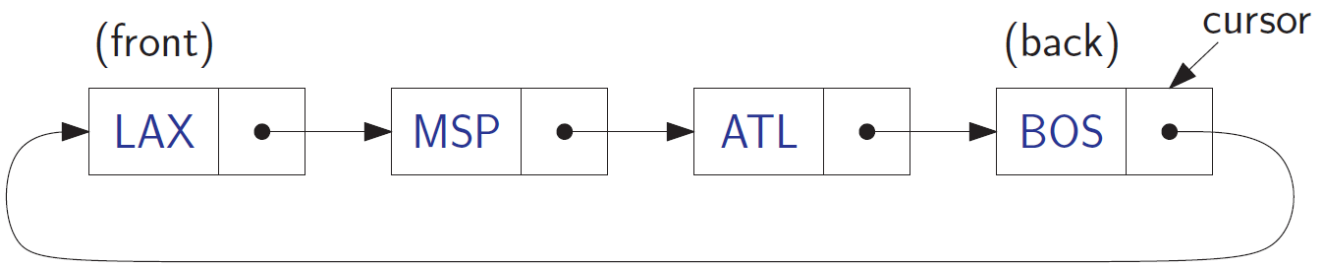
Danh sách liên kết đơn



Danh sách liên kết đôi

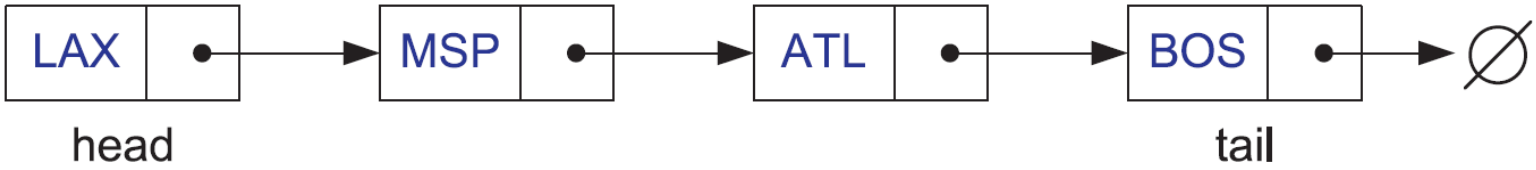


Danh sách liên kết vòng tròn



2. Danh sách liên kết đơn

Danh sách liên kết đơn



- Có một liên kết duy nhất giữa hai nút liên tiếp
- Các thao tác chính:
  - Chèn phần tử mới vào đầu danh sách
  - Xóa phần tử đầu danh sách
  - Lấy phần tử đầu danh sách

Cài đặt danh sách liên kết đơn

```
template <typename T>
class SingleList {
    public:
        hàm tạo, hàm hủy
        chèn/xóa ở đầu danh sách
        lấy phần tử đầu danh sách
        ...

    private:
        struct Node { ... }; // kiểu dữ liệu của các nút
        Node * head; // con trỏ tới nút đầu danh sách
};
```

Kiểu dữ liệu của các nút

```
struct Node {
    T elem;       // phần tử
    Node * next; // liên kết tới nút kế tiếp

    // Hàm tạo
    Node(T e, Node * n) {
        elem = e;
        next = n;
    }
};
```

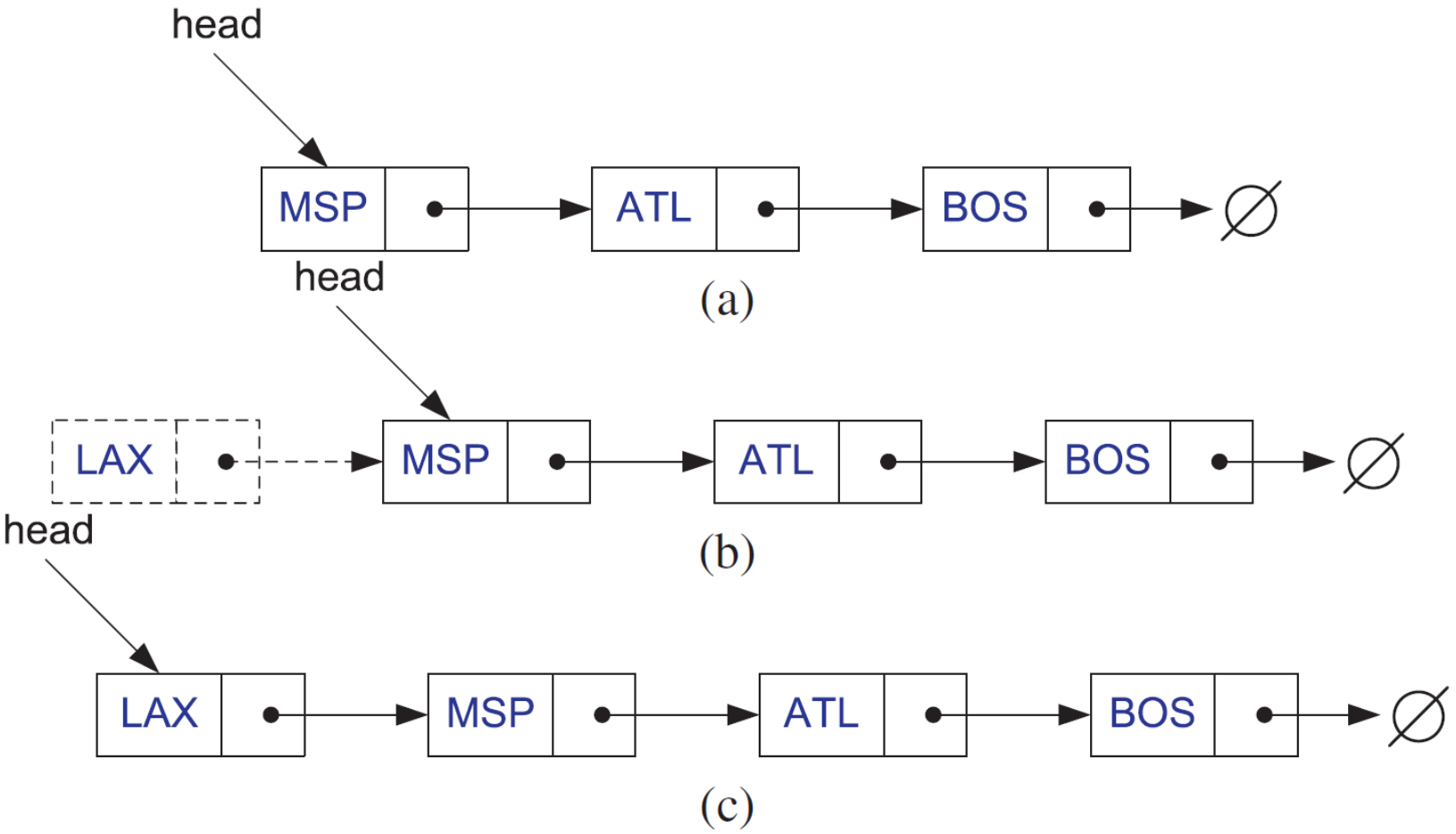
Hàm tạo và hàm hủy

```
SingleList() {
    head = NULL;
}

// Hàm empty kiểm tra trạng thái rỗng
// Hàm popFront xóa phần tử đầu danh sách
// (tham khảo các slide sau cho hai hàm đó)
~SingleList() {
    while (!empty())
        popFront();
}
```

Các hàm khác

```
// Kiểm tra danh sách có rỗng hay không
bool empty() {
    return (head == NULL);
}

// Lấy phần tử đầu danh sách
T front() {
    return head->elem;
}
```

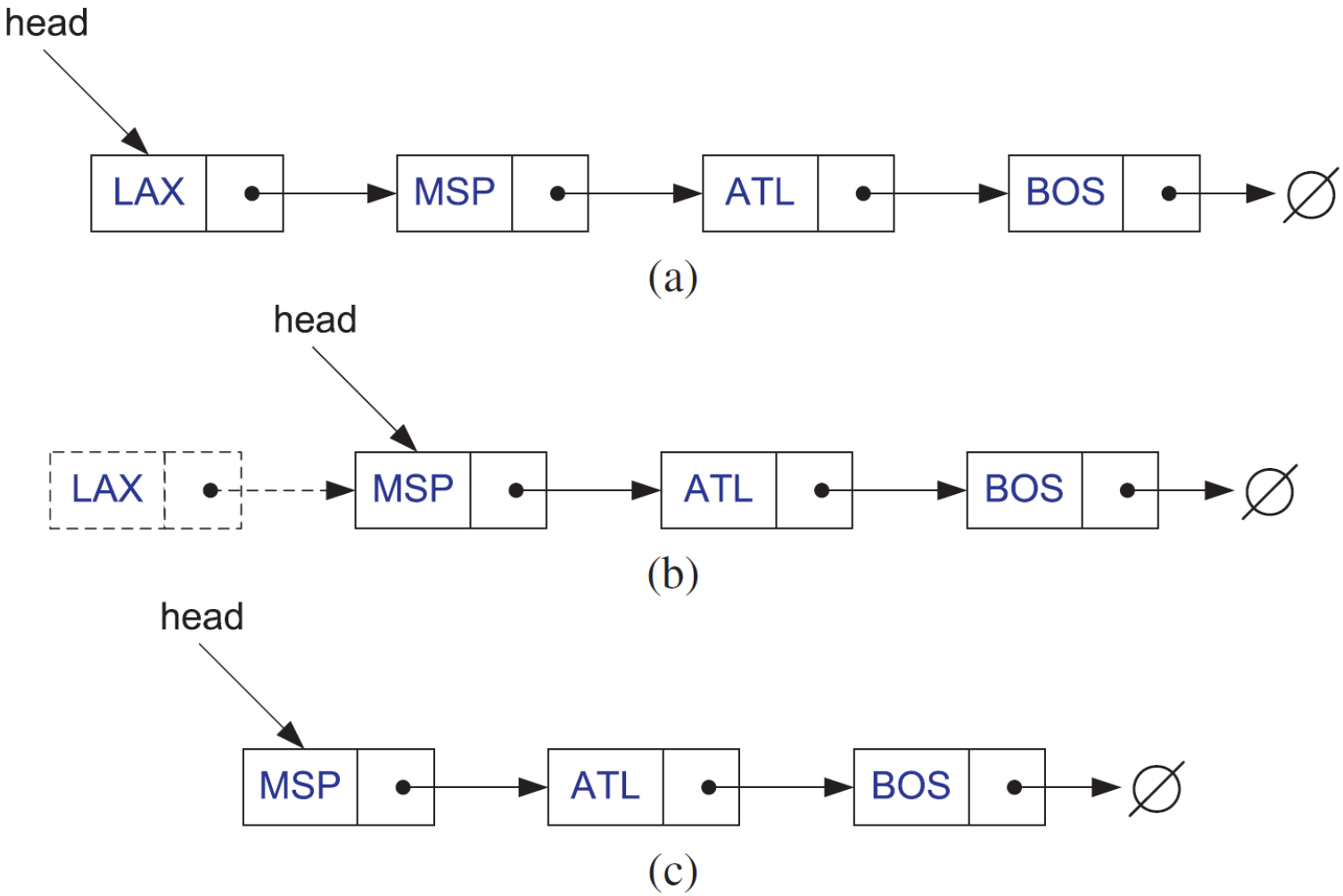
Chèn vào đầu danh sách



Chèn vào đầu danh sách (tiếp)

```
// e (element) là phần tử cần chèn
void pushFront(T e) {
    // v là nút mới, trong đó v.next = head có
    // nghĩa là v trỏ tới nút đầu danh sách.
    Node * v = new Node(e, head);

    // Nút đầu danh sách bây giờ là v, vì vậy
    // phải cập nhật con trỏ head.
    head = v;
}
```

Xóa phần tử đầu danh sách



Xóa phần tử đầu danh sách (tiếp)

```
void popFront() {
    // Giữ lại nút đầu danh sách
    Node * old = head;

    // Nhảy sang nút kế tiếp
    head = head->next;

    // Xóa nút đầu danh sách cũ
    delete old;
}
```

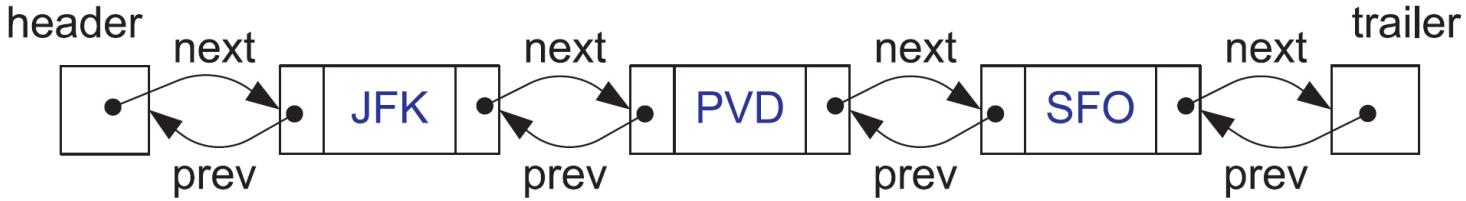
Phân tích thời gian chạy

- Hàm tạo: O(1)
- Hàm hủy: O(n) – vì phải xóa n phần tử
- Kiểm tra rỗng: O(1)
- Lấy phần tử đầu danh sách: O(1)
- Chèn/xóa ở đầu danh sách: O(1)

Vì sao không nên chèn/xóa ở cuối danh sách liên kết đơn?

3. Danh sách liên kết đôi

Danh sách liên kết đôi



- Mỗi nút chứa hai liên kết:
  - Liên kết tới nút tiếp theo
  - Liên kết về nút phía trước
- Các thao tác chính:
  - Chèn/xóa ở đầu, cuối hoặc vị trí hiện hành
  - Lấy phần tử ở đầu, cuối hoặc vị trí hiện hành
  - Duyệt danh sách tiến hoặc lùi
- Chú ý: header và trailer là những nút giả (không chứa phần tử), được dùng để thuận tiện cho việc lập trình

Cài đặt danh sách liên kết đôi

```
template <typename T>          // T là kiểu phần tử
class DoubleList {
    public:
        hàm tạo, hàm hủy, kiểm tra rỗng
        các thao tác chèn/xóa
        các thao tác lấy phần tử
        các thao tác duyệt danh sách
    private:
        struct DNode { ... }; // kiểu của các nút
        DNode * header;       // đầu danh sách
        DNode * trailer;      // cuối danh sách
        DNode * currentPos;   // vị trí hiện hành
};
```

Kiểu dữ liệu của các nút

```
struct DNode {
    T elem;         // phần tử
    DNode * next; // liên kết về phía sau
    DNode * prev; // liên kết về phía trước
};
```

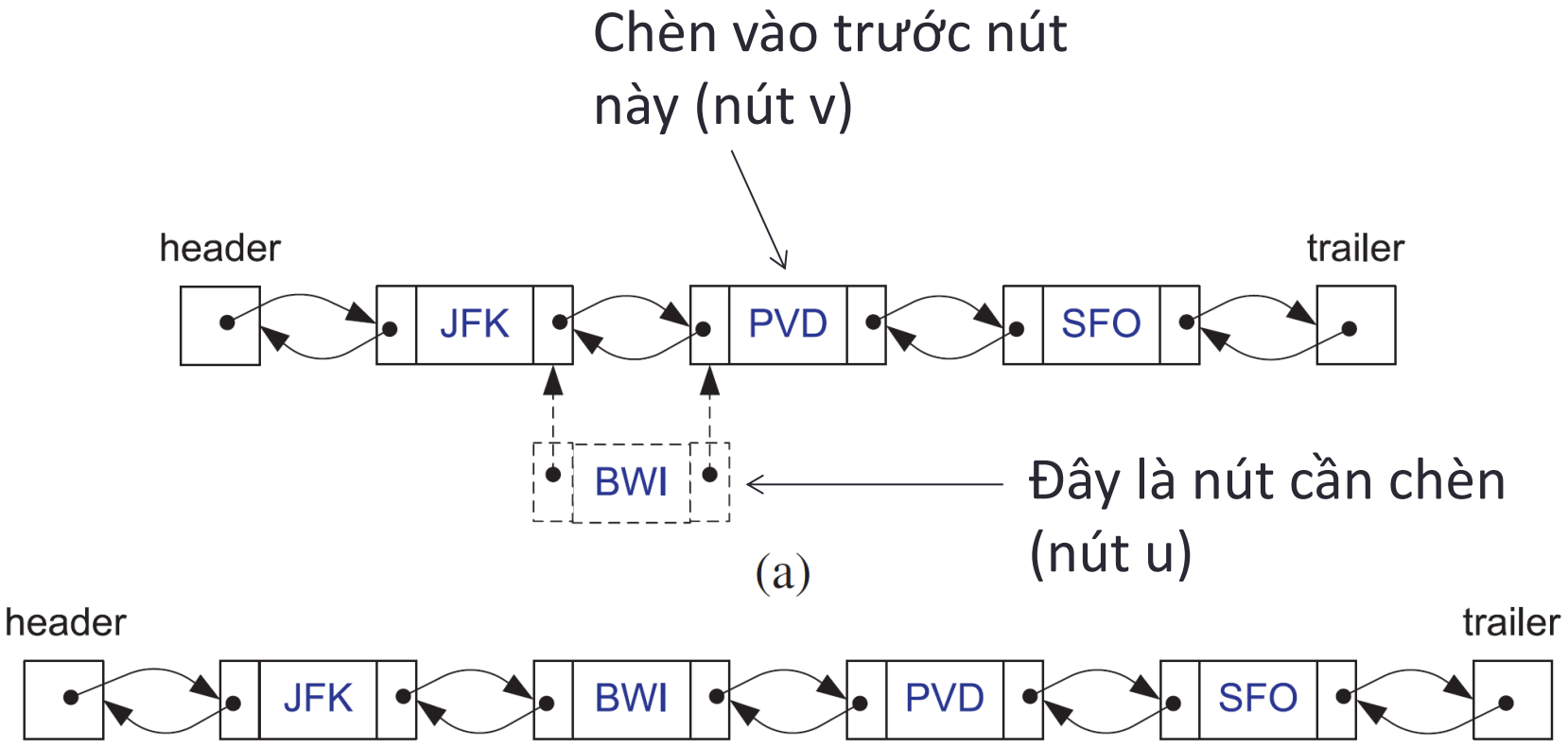
Khai báo các thao tác

```
DoubleList();          // hàm tạo
~DoubleList();         // hàm hủy
bool empty();          // kiểm tra rỗng
T front();             // lấy phần tử đầu danh sách
T back();              // lấy phần tử cuối danh sách
T current();           // lấy phần tử hiện hành
bool moveNext();       // chuyển sang nút tiếp theo
bool movePrevious(); // chuyển về nút phía trước
void moveFront();      // chuyển về đầu danh sách
void moveBack();       // chuyển về cuối danh sách
```

Khai báo các thao tác (tiếp)

```
void pushFront(T e); // chèn vào đầu danh sách
void pushBack(T e);  // chèn vào cuối danh sách
void popFront();     // xóa ở đầu danh sách
void popBack();      // xóa ở cuối danh sách

// Chèn vào trước vị trí hiện hành
void insert(T e);

// Xóa ở vị trí hiện hành
void remove();
```

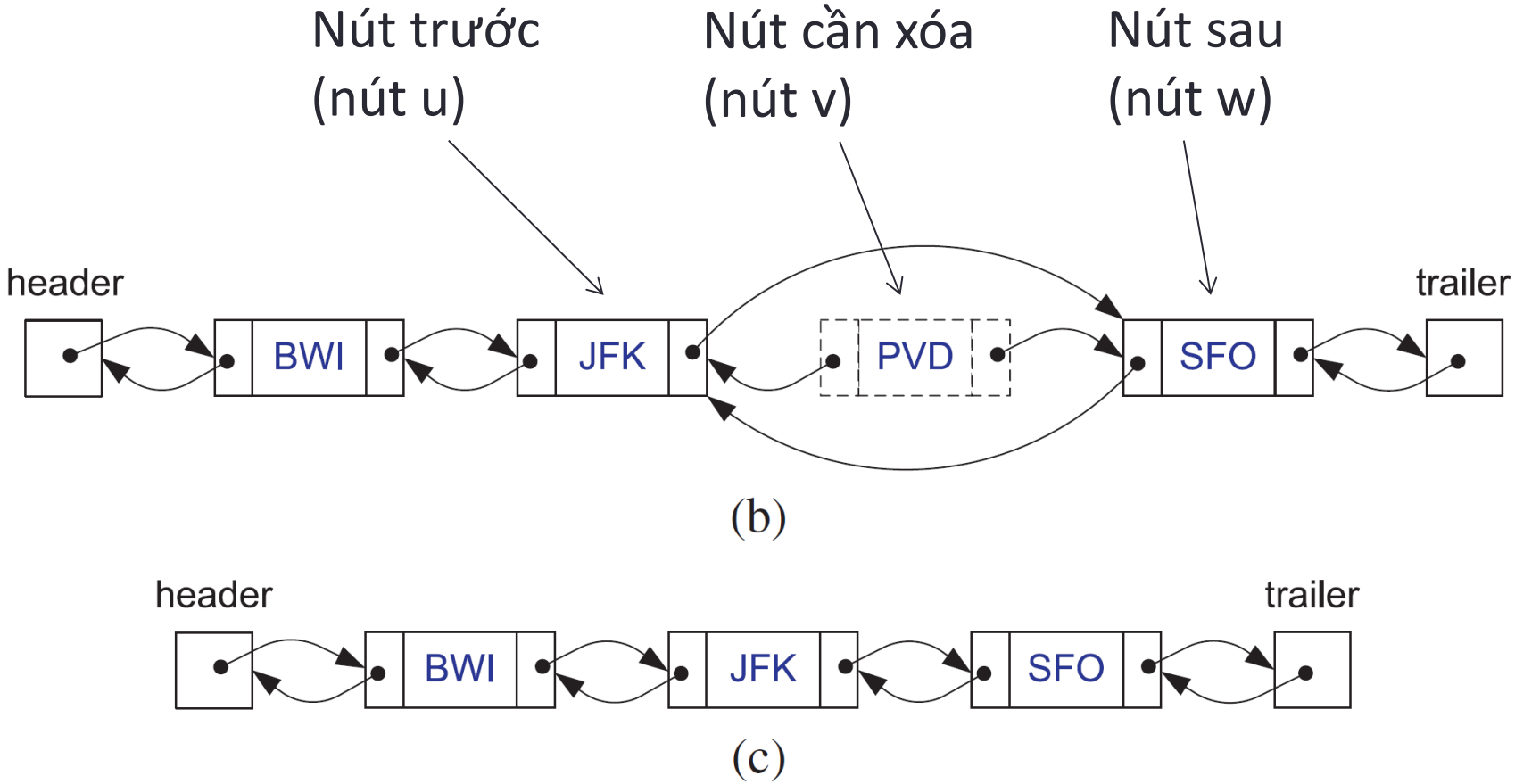
Chèn vào trước vị trí hiện hành



Chèn vào trước vị trí hiện hành (tiếp)

```
// Chèn phần tử e vào trước vị trí hiện hành
void insert(T e) {
    DNode * v = currentPos; // Nút hiện hành v
    DNode * u = new DNode;  // Nút mới u
    u->elem = e;            // Nút mới u chứa e,
    u->next = v;            // liên kết với nút sau và
    u->prev = v->prev; // liên kết với nút trước.
    v->prev->next = u; // Nút trước liên kết với u
    v->prev = u;            // Nút sau liên kết với u
}
```

Xóa ở vị trí hiện hành

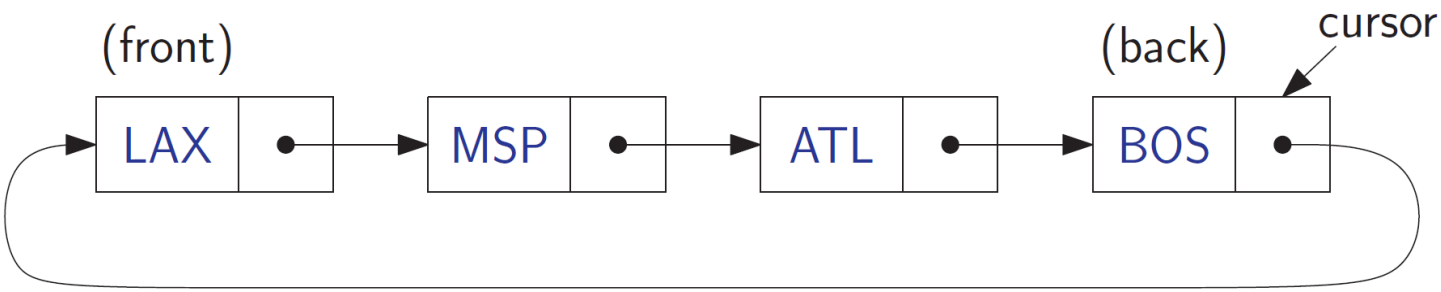


Xóa ở vị trí hiện hành (tiếp)

```
// Xóa nút v nằm ở vị trí hiện hành
void remove() {
    DNode * v = currentPos; // Nút hiện hành v
    DNode * u = v->prev;    // Nút trước nút v
    DNode * w = v->next;    // Nút sau nút v
    u->next = w;    // Nút trước trỏ tới nút sau
    w->prev = u;    // Nút sau trỏ về nút trước
    delete v;       // Xóa nút hiện hành cũ
    currentPos = w; // Vị trí hiện hành mới
}
```

4. Danh sách liên kết vòng tròn

Danh sách liên kết vòng tròn



- Cấu trúc tương tự như danh sách liên kết đơn
- Nhưng có thêm con trỏ đặc biệt **cursor** trỏ đến cuối danh sách (back), và liên kết next của nút cuối trỏ vòng về đầu danh sách (front)
- Các thao tác chính:
  - Chèn và xóa ở sau cursor
  - Lấy phần tử ở đầu và cuối danh sách
  - Dịch chuyển cursor sang vị trí tiếp theo

Cài đặt danh sách liên kết vòng tròn

```
template <typename T>          // T là kiểu phần tử
class CircleList {
    public:
        hàm tạo, hàm hủy, kiểm tra rỗng
        chèn/xóa ở sau cursor
        lấy phần tử ở đầu và cuối danh sách
        dịch chuyển cursor sang vị trí tiếp theo
    private:
        struct CNode { ... }; // kiểu của các nút
        CNode * cursor;       // con trỏ đặc biệt
};
```

Kiểu dữ liệu của các nút

```
struct CNode {
    T elem;         // phần tử
    CNode * next; // liên kết về phía sau
};
```

Khai báo các thao tác

```
CircleList();      // hàm tạo
~CircleList();     // hàm hủy
bool empty();      // kiểm tra rỗng
T front();         // lấy phần tử đầu danh sách
T back();          // lấy phần tử cuối danh sách
void moveNext();   // dịch chuyển cursor
void insert(T e); // chèn vào sau cursor
void remove();     // xóa nút sau cursor
```

Chèn vào sau cursor

```
// Chèn phần tử e vào sau cursor
void insert(T e) {
    CNode * v = new CNode; // tạo nút mới v
    v->elem = e;           // nút mới chứa e
    if (cursor == NULL) {  // nếu danh sách rỗng
        v->next = v;       // nút v trỏ tới chính nó
        cursor = v;        // cursor trỏ tới nút v
    }
    else {                 // nếu danh sách không rỗng
        v->next = cursor->next;
        cursor->next = v;
    }
}
```

Xóa nút sau cursor

```
void remove() {
    CNode * old = cursor->next; // nút cần xóa
    if (old == cursor) // nếu danh sách chỉ có một nút
        cursor = NULL; // cursor thành NULL sau khi xóa
    else               // nếu danh sách có nhiều nút
        cursor->next = old->next;
    delete old;
}
```