NHÓM 7 COMPUTER SCIENCE

Nguyễn Vũ Khang
Trương Hoàng Khiêm
Bùi Nhật Anh Khôi
Đinh Lê Bình An

OSA

```
struct Node
{
    int Info;
    struct Node* pNext;
};
typedef struct Node NODE;

struct List
{
    NODE* pHead;
    NODE* pTail;
};
typedef struct List LIST;
```

Cấu trúc dữ liệu của 1 node trong List đơn

```
struct NODE {
    data info;
    struct NODE* pNext;
};
```

Cấu trúc dữ liệu của DSLK đơn

```
struct LIST {
    NODE* pHead;
    NODE* pTail;
};
```

```
void Traverse(LIST L)
{
    NODE* p = L.pHead;
    if (p == NULL)
    {
        cout << "Danh sach rong!";
        return;
    }
    else
    {
        while (p != NULL)
        {
            cout << p->Info << " ";
            p = p->pNext;
        }
    }
}
```

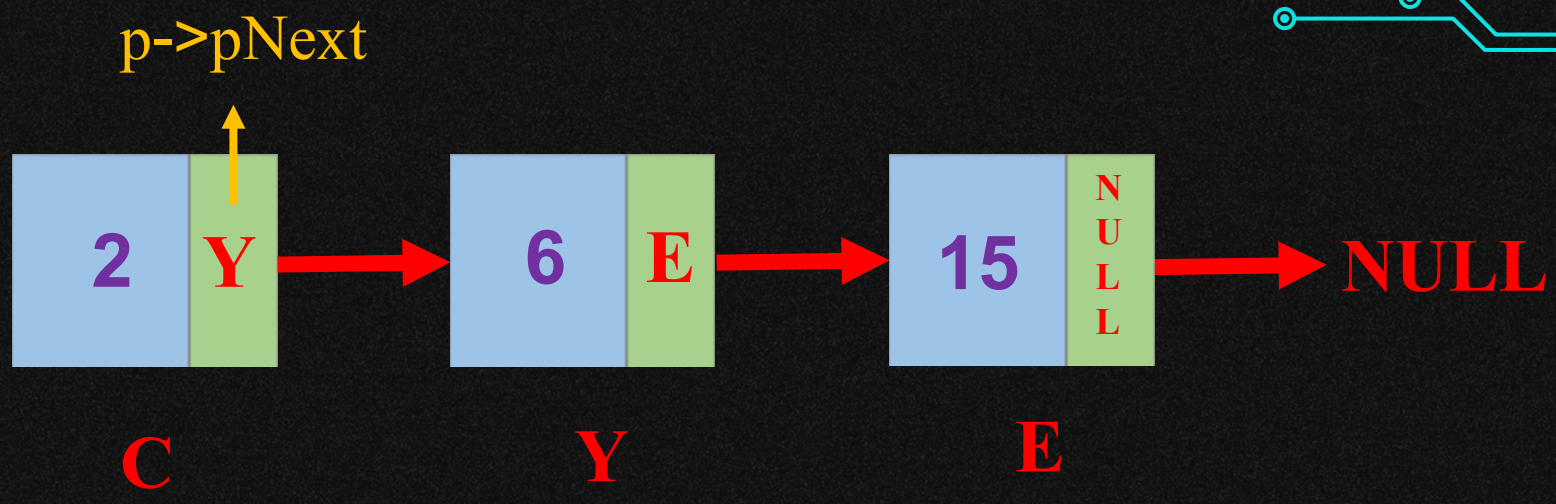
Duyệt danh sách

B1: cho node p = phead

B2: trong khi danh sách chưa duyệt hết

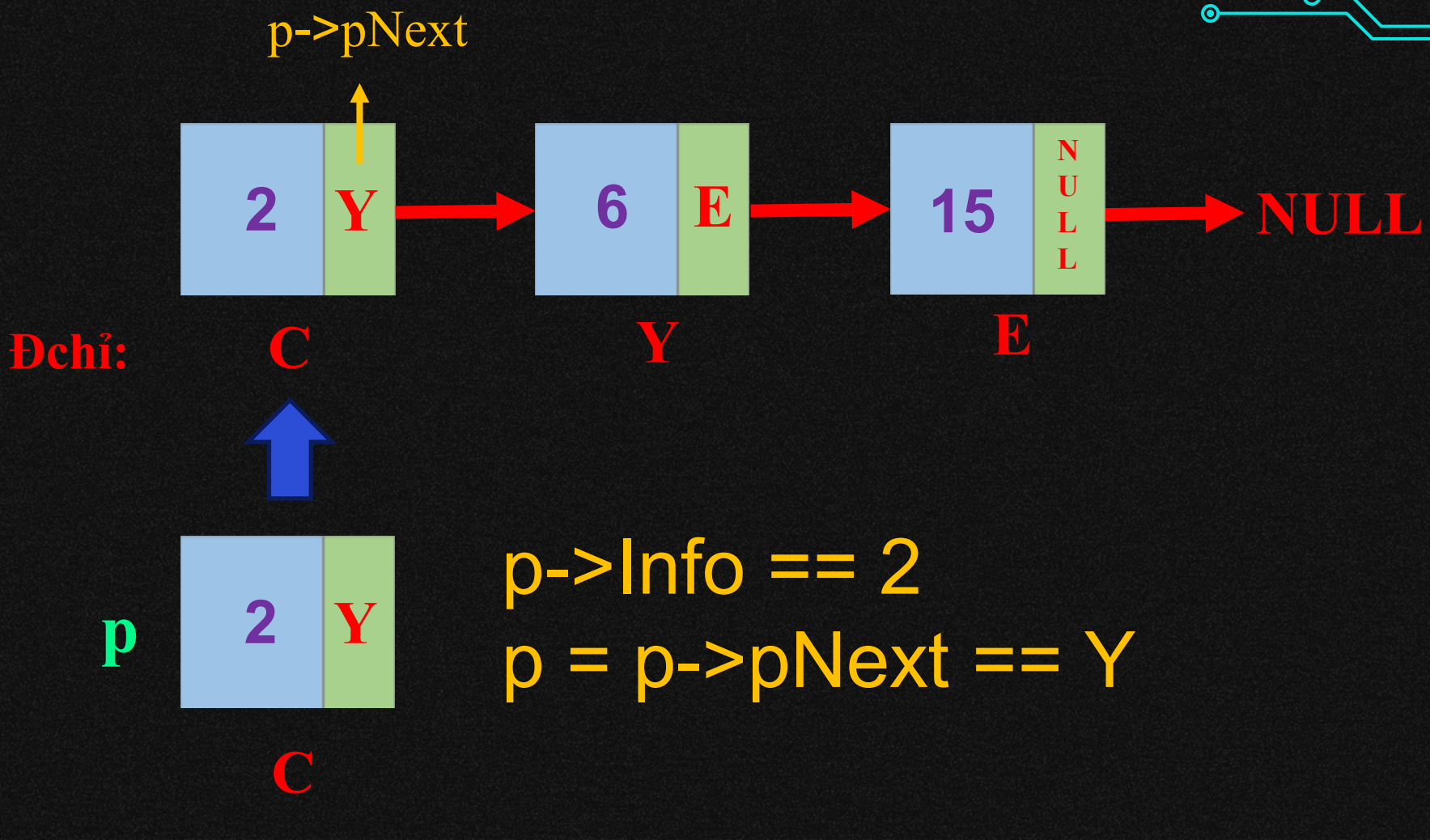
+ Xử lý phần tử p

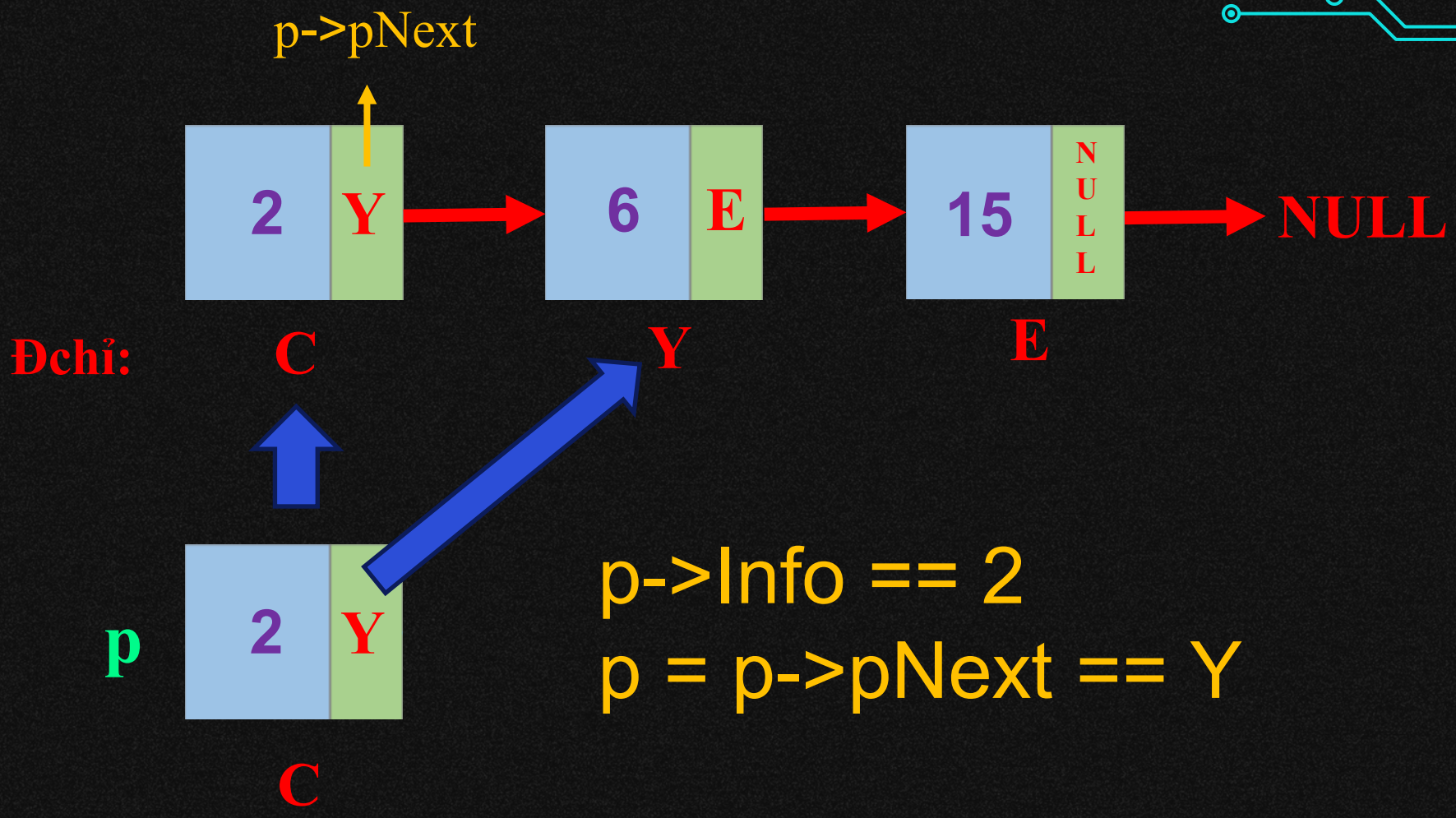
+ Cho p = pNext; // cập nhập lại biến con trỏ p để duyệt tới hết danh sách khi p == NULL

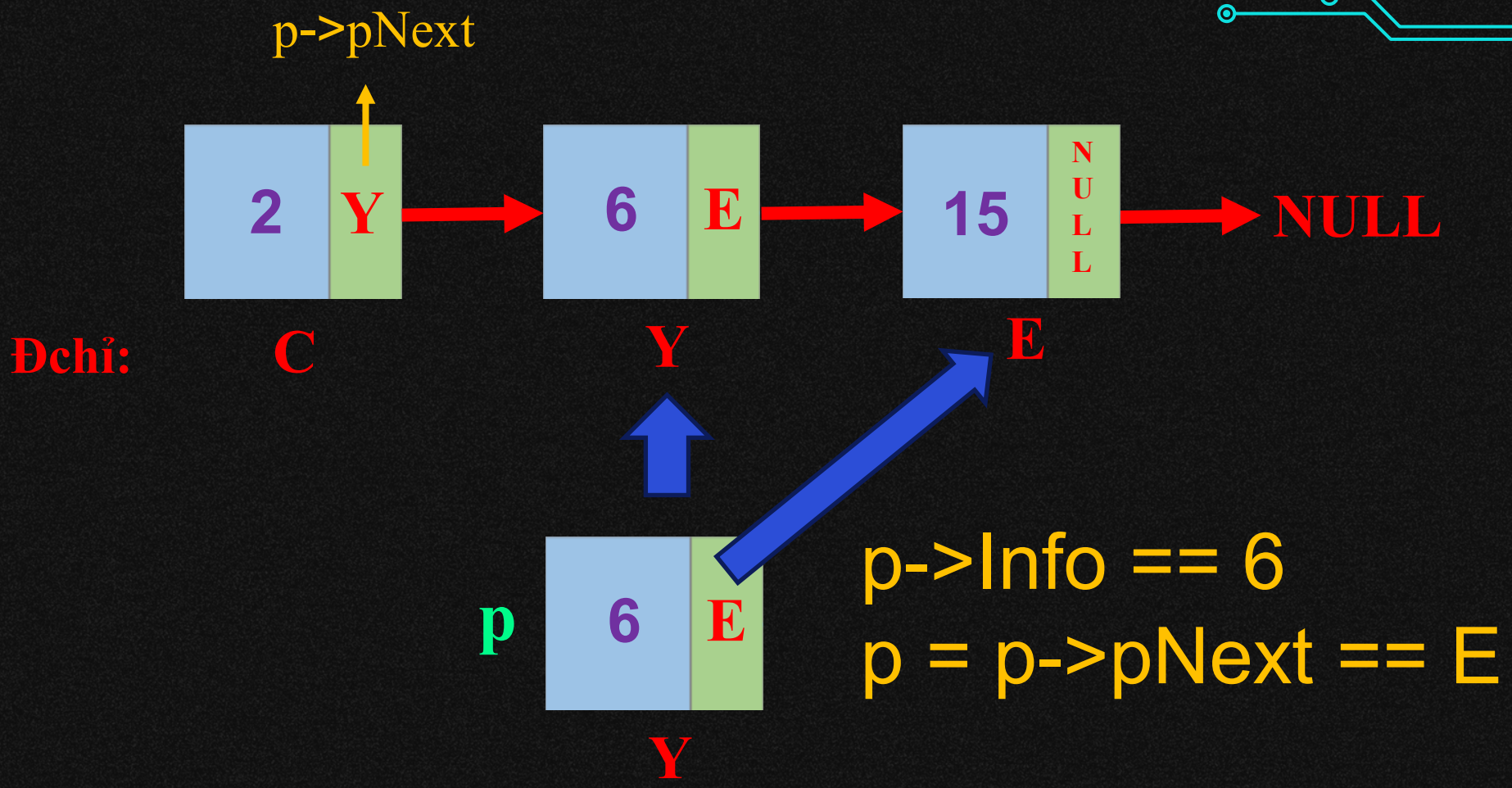


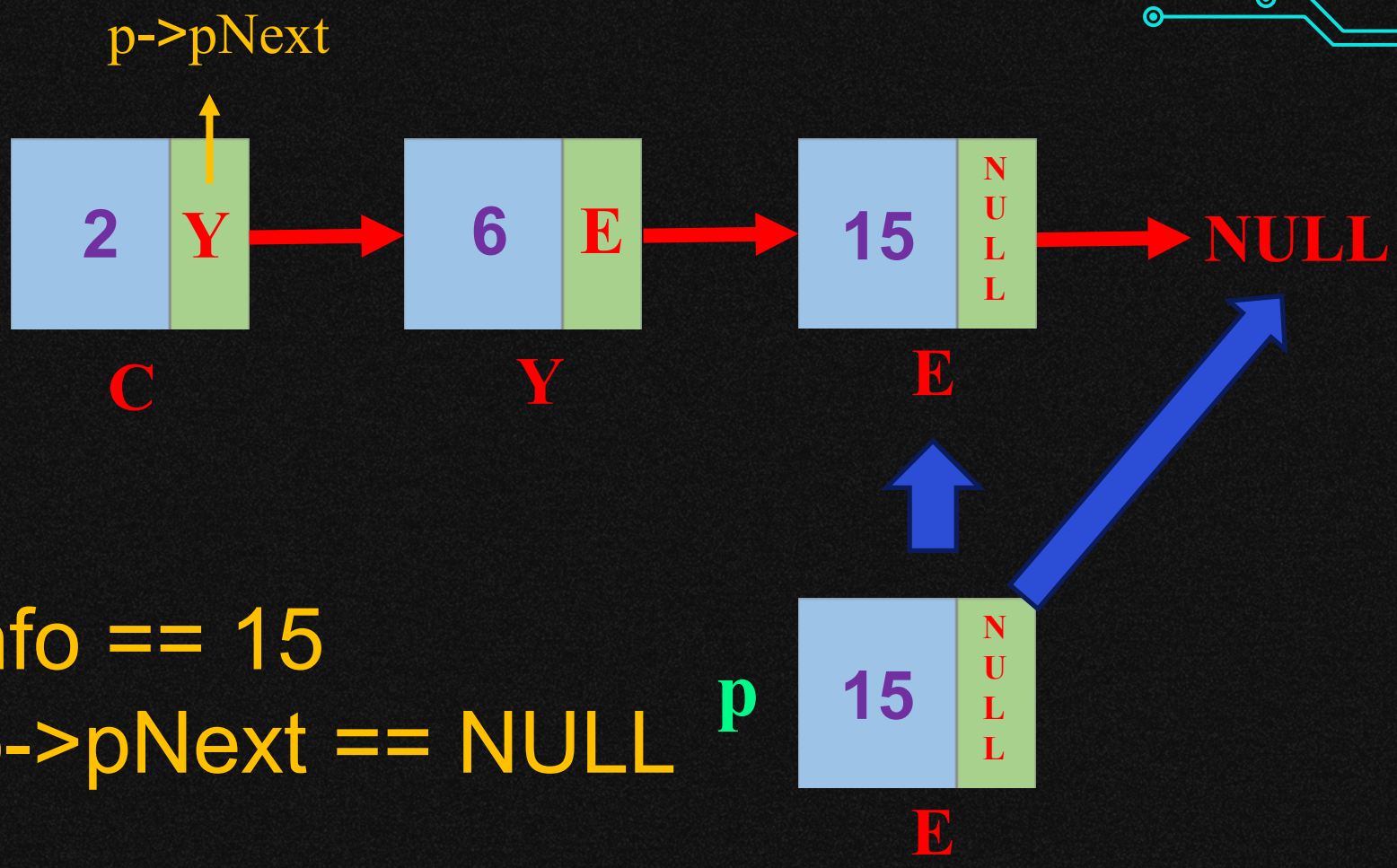
B1: cho node p = phead

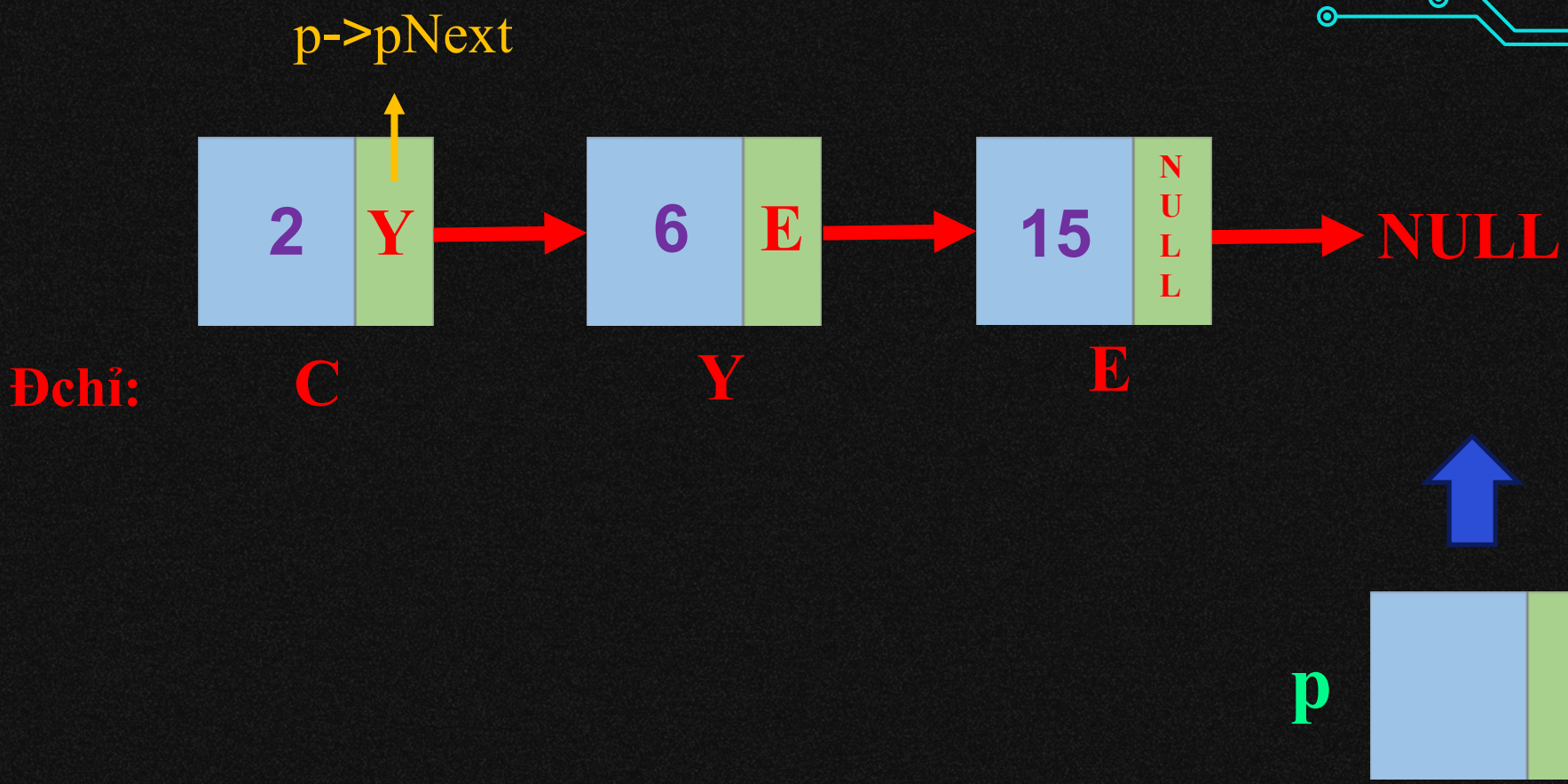
p = phead == C


p->pNext
2
Y
6
E
15
NULL
NULL
Đchỉ:
C
Y
E
p
2
Y
C
p->Info == 2
p = p->pNext == Y


p->pNext
2
Y
6
E
15
NULL
NULL
Đchỉ:
C
Y
E
p
2
Y
C
p->Info == 2
p = p->pNext == Y

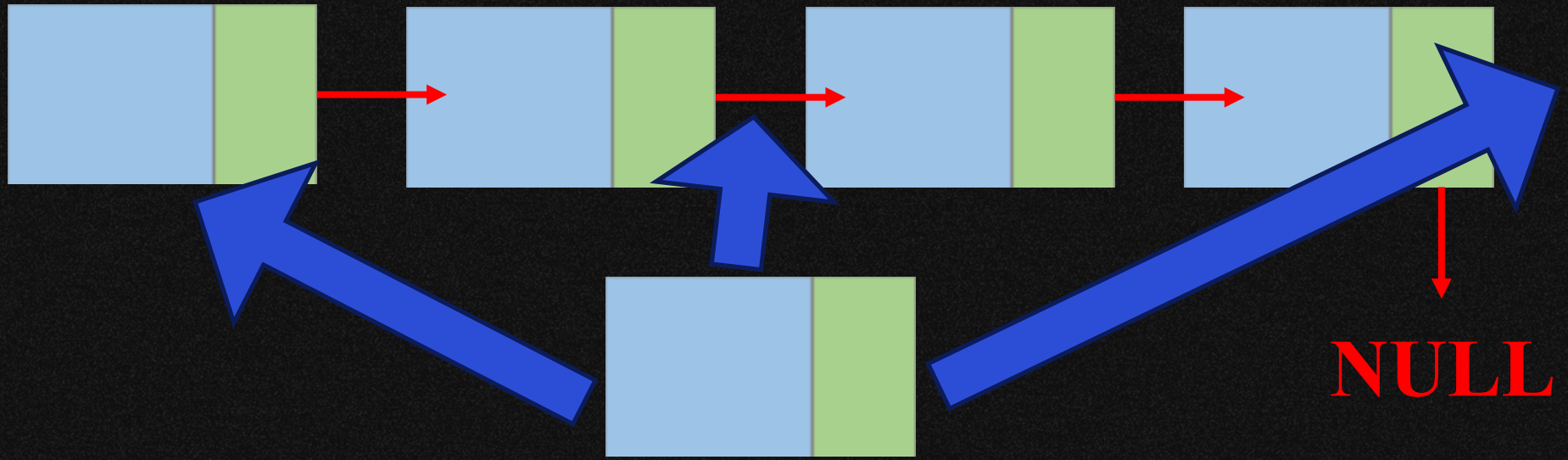

p->pNext
2
Y
6
E
15
NULL
NULL
Đchỉ:
C
Y
E
p
6
E
Y
p->Info == 6
p = p->pNext == E


p->pNext
2
Y
6
E
15
NULL
NULL
Đchỉ:
C
Y
E
p->Info == 15
p = p->pNext == NULL
p
15
NULL
E

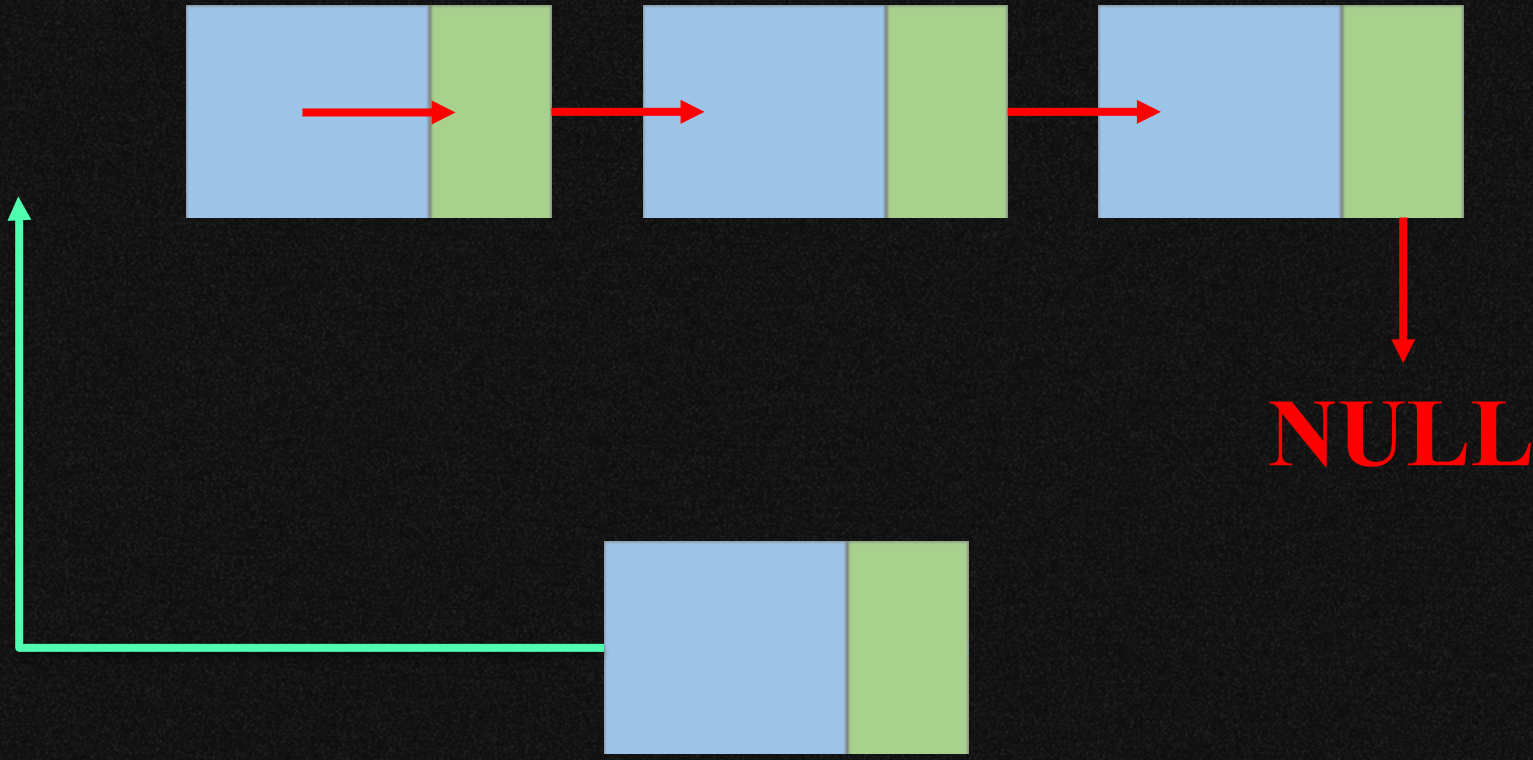

p->pNext
2
Y
6
E
15
NULL
NULL
Đchỉ:
C
Y
E
p

1. Thêm một phần tử có khóa x vào danh sách

- Thêm vào đầu List đơn
- Thêm vào cuối List
- Thêm vào vị trí bất kỳ.



Thêm node p vào đầu danh sách liên kết đơn



Thêm node p vào đầu danh sách liên kết đơn

```
void AddHead(LIST& L, NODE* P)
{
    if (L.pHead == NULL)
    {
        L.pHead = P;
        L.pTail = L.pHead;
    }
    else
    {
        P->pNext = L.pHead;
        L.pHead = P;
    }
}
```

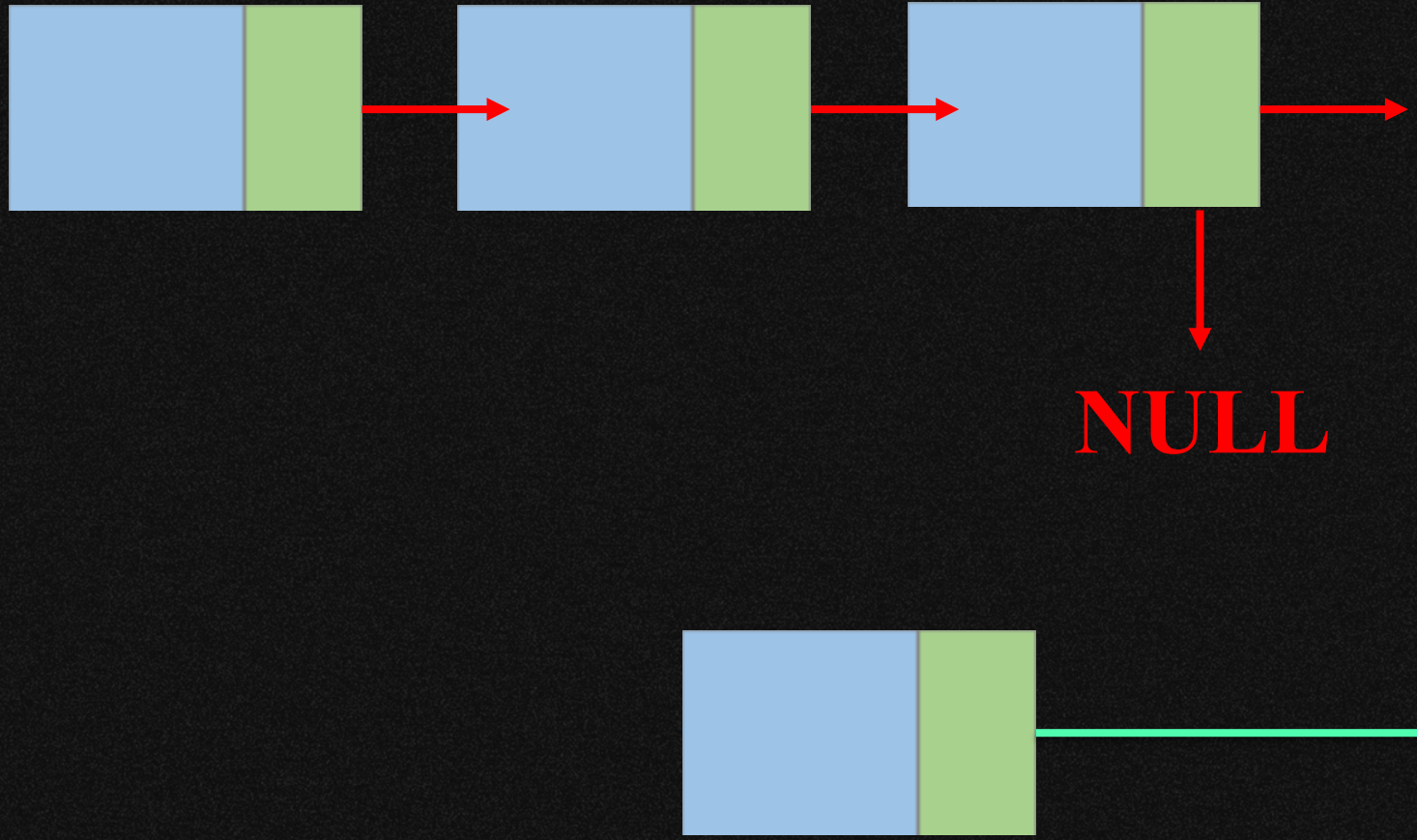
- Nếu List rỗng thì:

+ pHead = p;
+ pTail = pHead;

- Ngược lại (List không rỗng)

+ p->pNext = pHead;
+ pHead = p

Thêm node p vào cuối danh sách liên kết đơn



Thêm node p vào cuối danh sách liên kết đơn

```
void AddTail(LIST& L, NODE* P)
{
    if (L.pHead == NULL)
        AddHead(L, P);
    else
    {
        L.pTail->pNext = P;
        L.pTail = P;
    }
}
```

- Nếu List rỗng thì

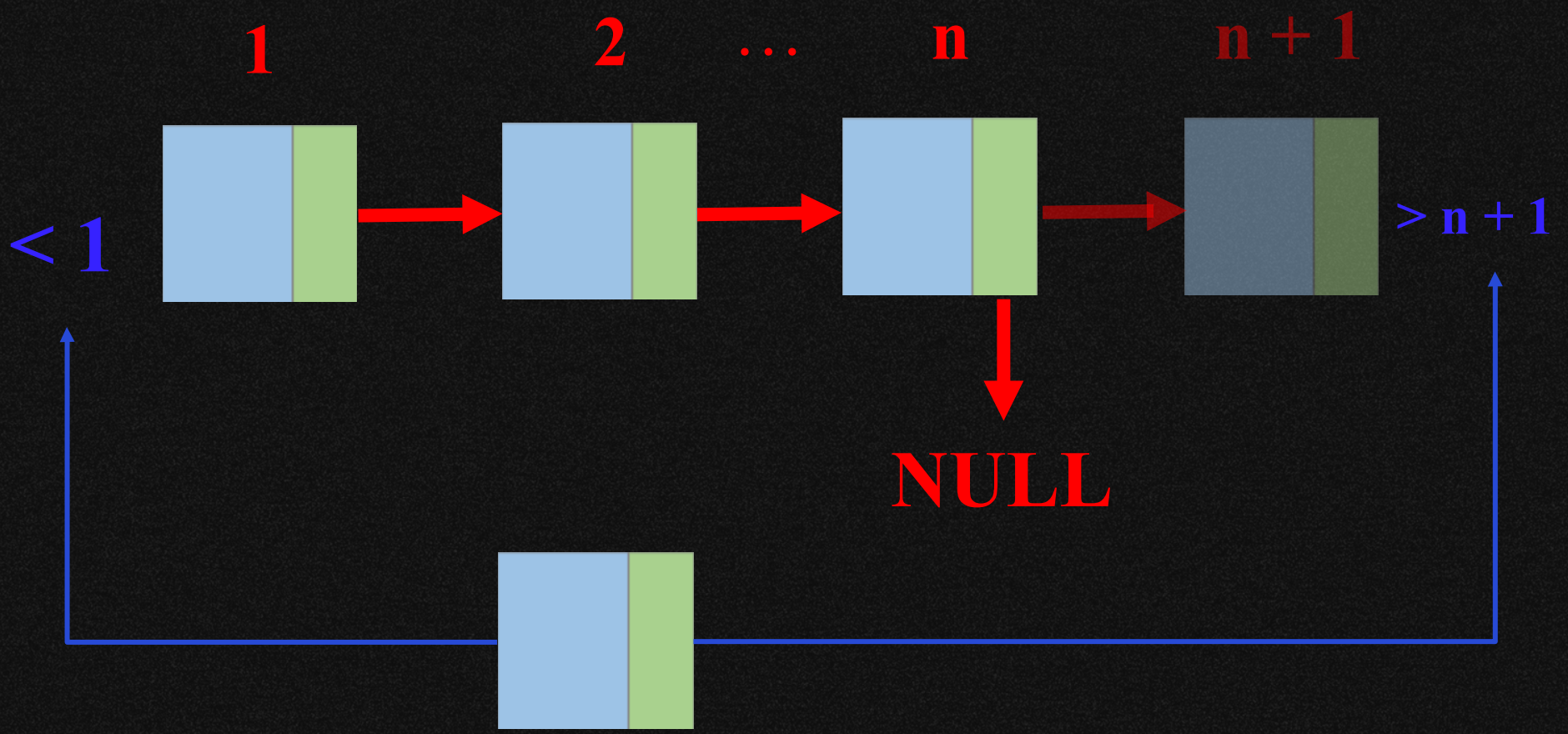
-> Thêm Head cho list rỗng:

+ pHead = p;
+ pTail = pHead;

- Ngược lại (List không rỗng)

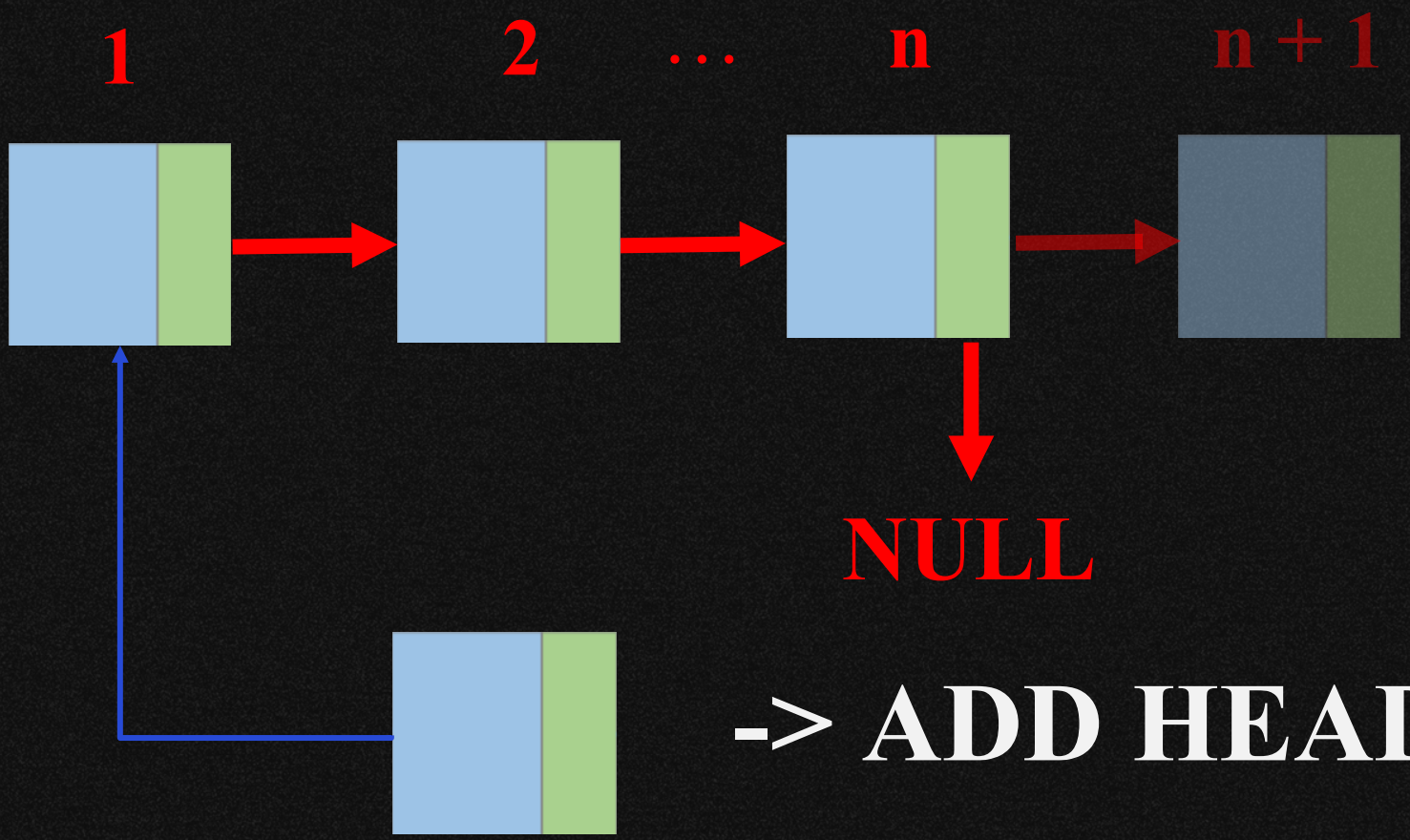
+ pTail->pNext=p;
+ pTail=p

Thêm node p vào vị trí bất kỳ

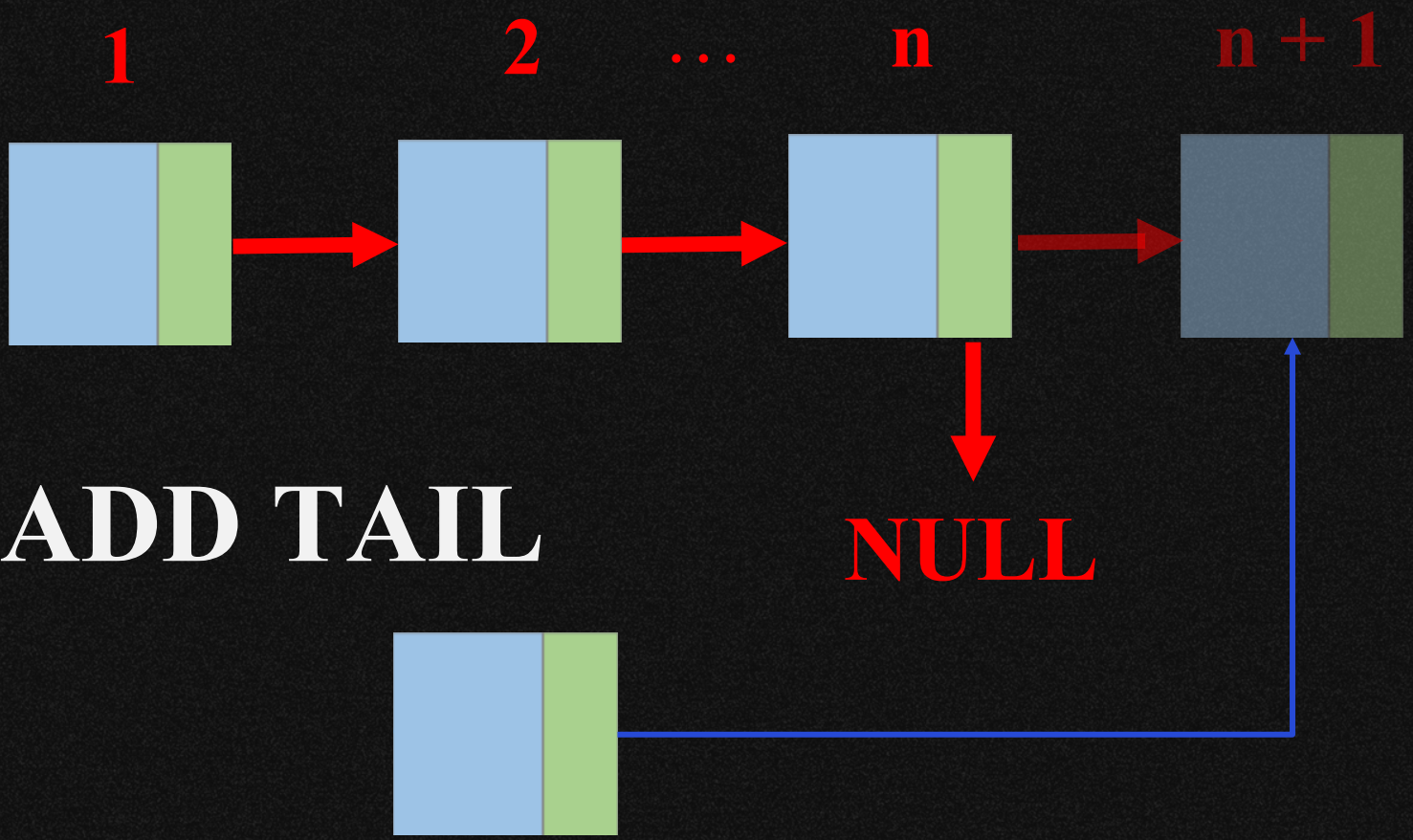


Thêm node p vào vị trí bất kỳ

Thêm node p vào vị trí bất kỳ

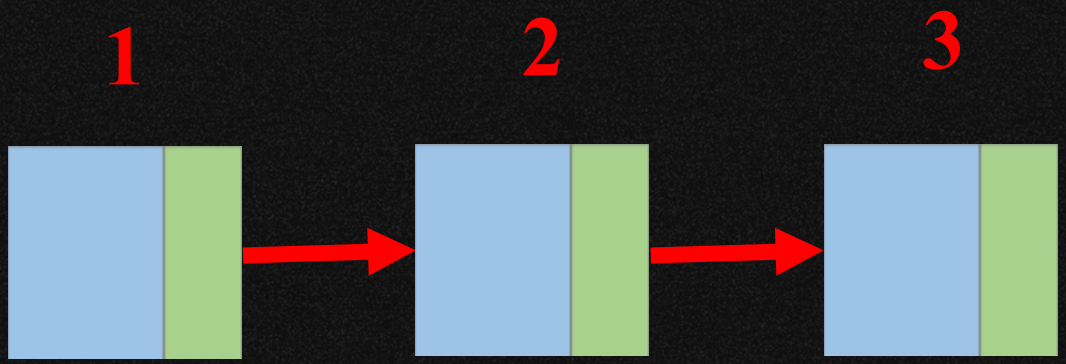


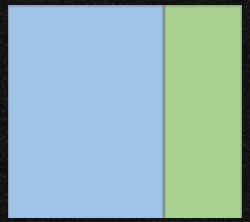
Thêm node p vào vị trí bất kỳ



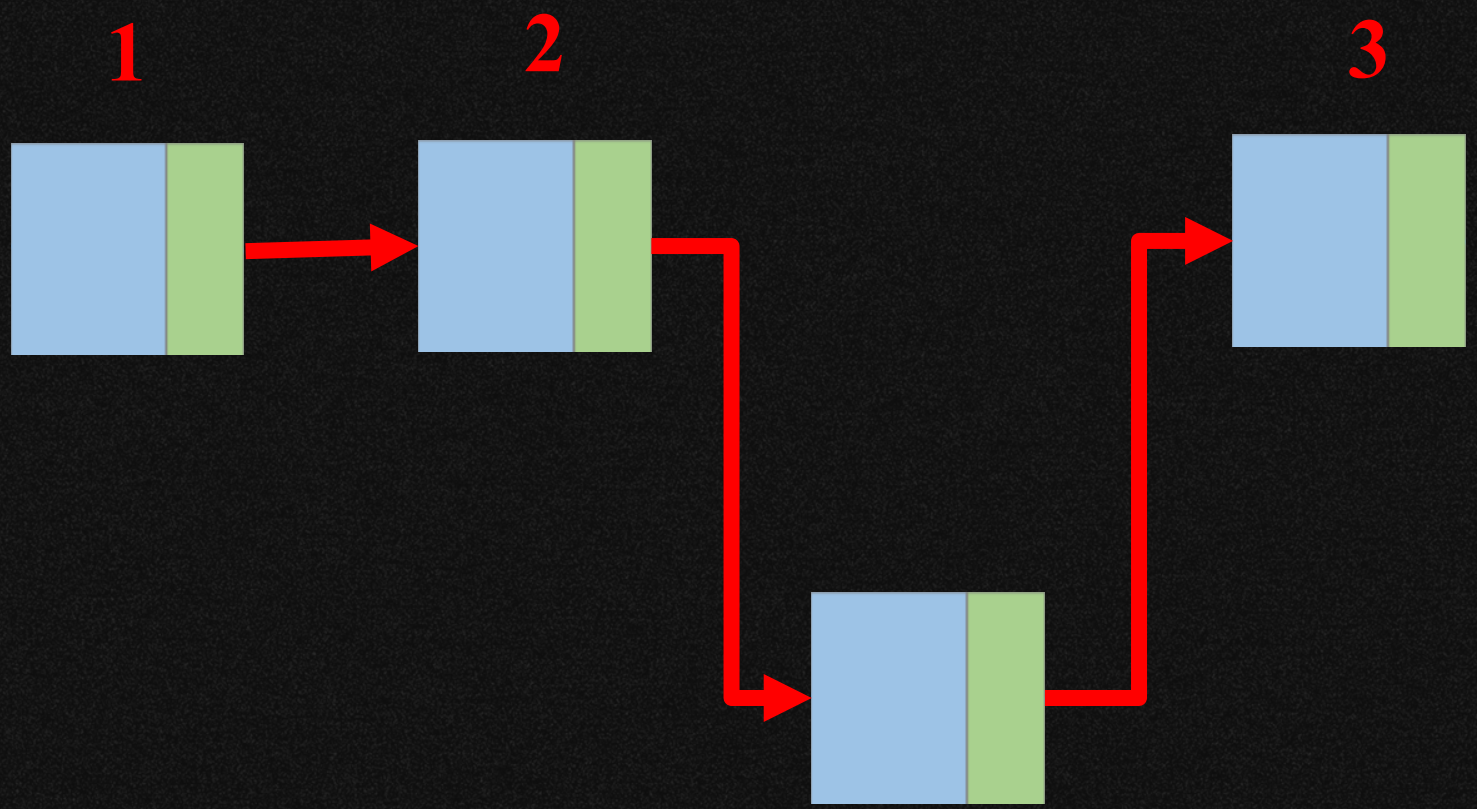
-> ADD TAIL

Thêm node p vào vị trí bất kỳ

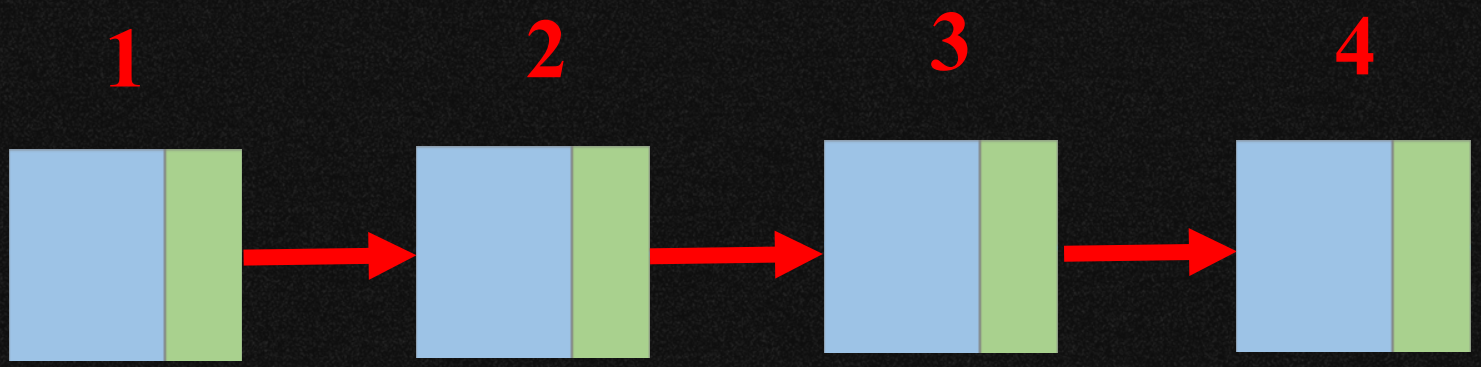




Thêm node p vào vị trí bất kỳ



Thêm node p vào vị trí bất kỳ



```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Đếm số node từ đầu tới cuối danh sách liên kết

```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Các phần từ có thể thêm sẽ nằm trong đoạn từ [1,n+1].

Nếu vị trí chèn < 1 hoặc lớn hơn n + 1 thì báo vị trí chèn không hợp lệ và kết thúc hàm

```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Nếu vị trí cần thêm là 1: Gọi hàm thêm vào đầu

```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Nếu vị trí cần thêm là n+1: Gọi hàm thêm vào cuối

```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Các vị trí còn lại:

+ Khởi tạo node p.

+ Cho vòng lặp chạy đến trước vị trí cần chèn một node.

```
void AddPosition(LIST& L, NODE* P, int position)
{
    int size = CountNode(L);
    if (position < 1 || position > size + 1)
    {
        cout << "Vi tri chen khong hop le!";
        return;
    }
    else if (position == 1)
    {
        AddHead(L, P);
        return;
    }
    else if (position == size + 1)
    {
        AddTail(L, P);
        return;
    }

    NODE* CurrentNode = L.pHead;
    for (int i = 1; i <= position - 2; i++)
    {
        CurrentNode = CurrentNode->pNext;
    }

    // CurrentNode lúc này sẽ ở vị trí NODE trước vị trí cần chèn
    P->pNext = CurrentNode->pNext;
    CurrentNode->pNext = P;
}
```

Các vị trí còn lại:

+ Cho con trỏ Node p trỏ vào node kế tiếp tại vị trí node đang giữ

+ Cho con trỏ node tại vị trí hiện tại trỏ vào node p

Hủy phần tử trong DSLK đơn

- **Nguyên tắc: Phải cô lập phần tử cần hủy trước hủy.**

Hủy phần tử đầu trong DSLK

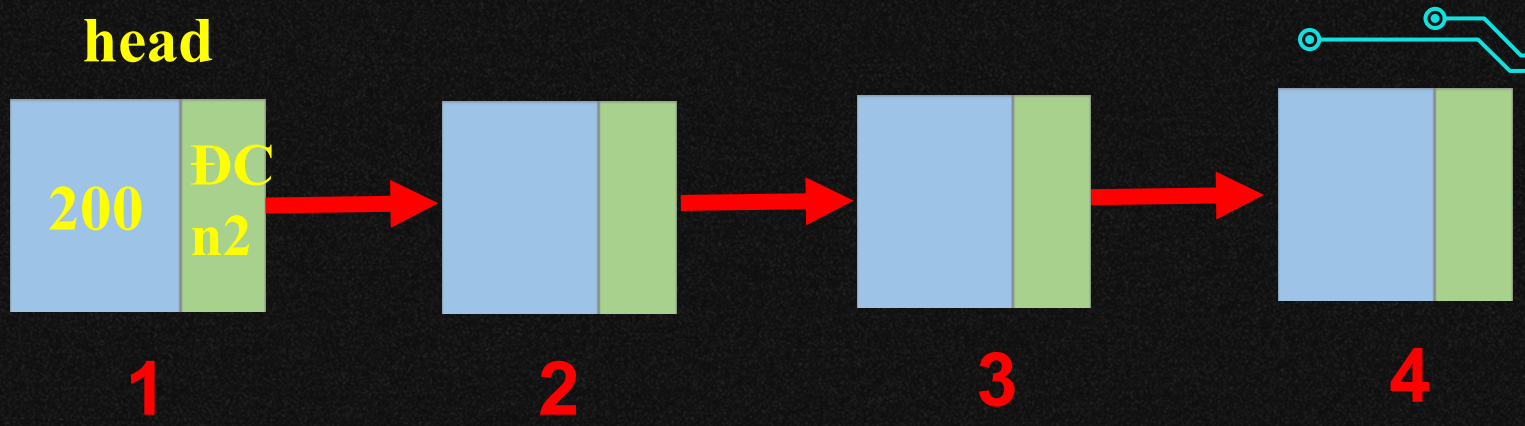
```
int DeleteHead(LIST& L, int& x)
{
    NODE* p;
    if (L.pHead != NULL)
    {
        x = L.pHead->Info;
        // Bước 1
        p = L.pHead;
        // Bước 2
        L.pHead = L.pHead->pNext;

        if (L.pHead == NULL)
            L.pTail = NULL;
        delete p;
        return 1;
    }
    return 0;
}
```

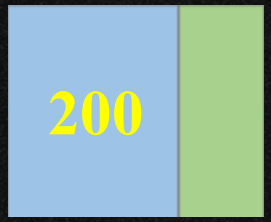
Bắt đầu:

Nếu pHead == NULL thì pTail=NULL

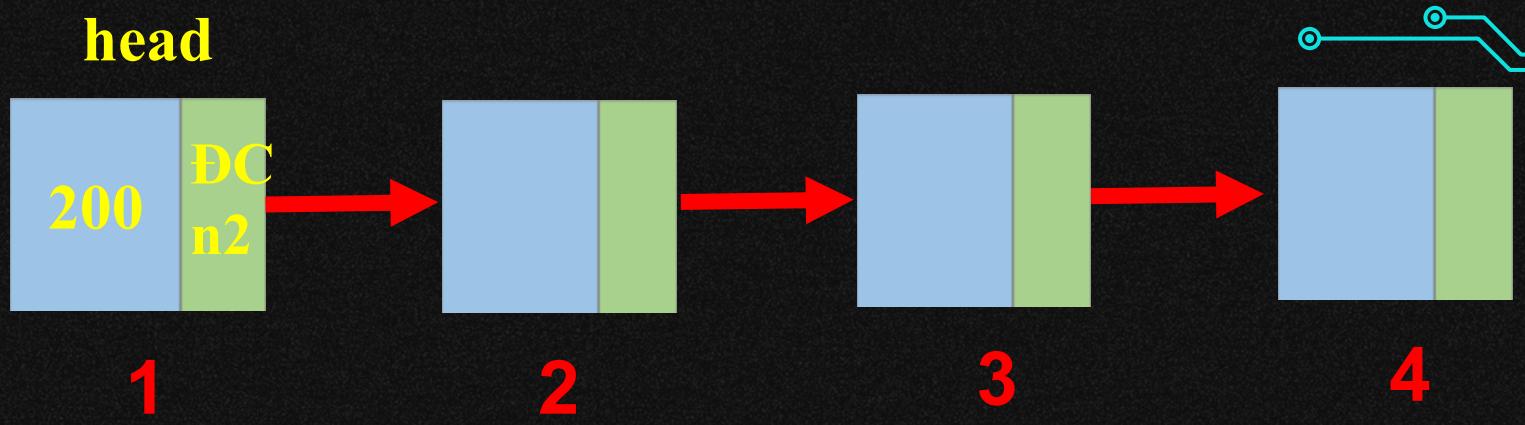
Nếu (pHead != NULL) thì

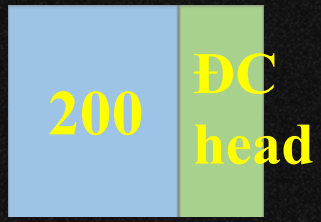
- B1: p = pHead (NODE Thế mạng)
- B2: pHead = pHead->pNext
- B3: delete (p)



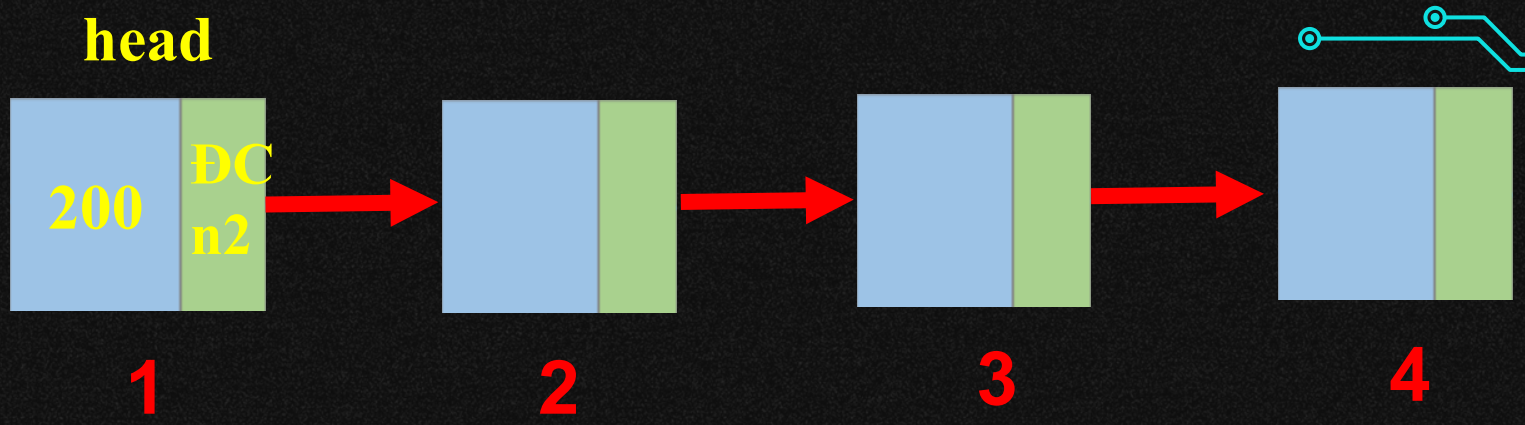


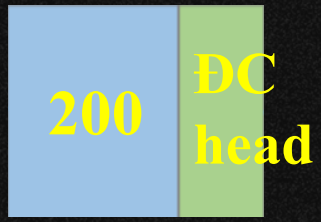

```
if (L.pHead != NULL)
{
    x = L.pHead->Info;
```



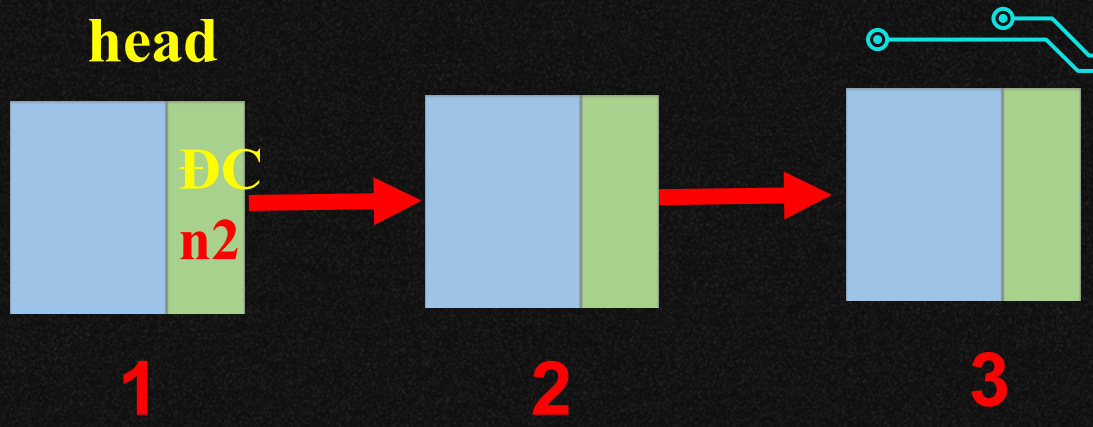


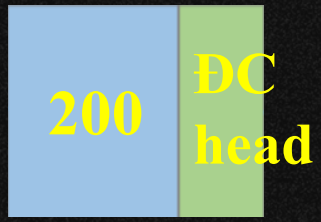

```
// Bước 1
p = L.pHead;
```



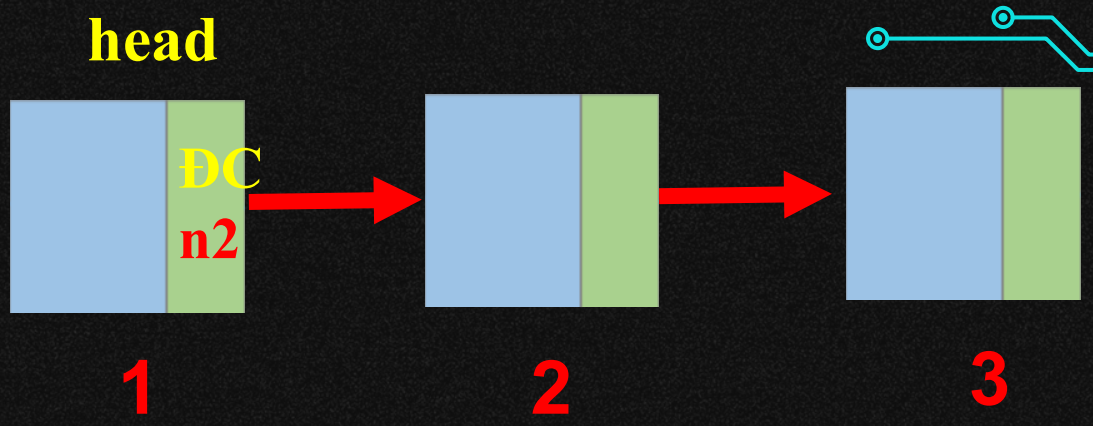


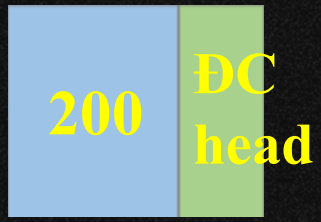

```
// Bước 2
L.pHead = L.pHead->pNext;
```






```
// Bước 2
L.pHead = L.pHead->pNext;
```





p

```
delete p;
```

```
int DeleteHead(LIST& L, int& x)
{
    NODE* p;
    if (L.pHead != NULL)
    {
        x = L.pHead->Info;
        // Bước 1
        p = L.pHead;
        // Bước 2
        L.pHead = L.pHead->pNext;

        if (L.pHead == NULL)
            L.pTail = NULL;
        delete p;
        return 1;
    }
    return 0;
}
```

// Đã xóa phần tử

// List rỗng không có phần tử để xóa

Câu hỏi: Làm ngược lại quá trình được không?

Hủy phần tử cuối trong DSLK

Nếu pHead==NULL thì pTail=NULL

Nếu (pHead!=NULL) thì

+ Nếu pHead->pNext == NULL

DeleteHead(L, x)

+ Ngược lại:
  - B1: p = pTail (NODE Thế mạng)
  - B2: pTail->pNext = NULL
  - B3: Xóa p

```
int DeleteTail(LIST& L, int& x)
{

    if (L.pHead == NULL)
    {
        L.pTail = NULL;
        return 0;
    }
    else if (L.pHead->pNext == NULL)
    {
        return DeleteHead(L, x);
    }
    else
    {
        NODE* CurrentNode = L.pHead;

        // duyệt node đến trước vị trí Node kế cuối
        while (CurrentNode->pNext->pNext != NULL)
            CurrentNode = CurrentNode->pNext;

        // CurrentNode lúc này sẽ ở vị trí Node kế Node Tail

        NODE* p;
        x = L.pTail->Info;
        p = L.pTail;
        CurrentNode->pNext = NULL;
        L.pTail = CurrentNode;

        delete p;
        return 1;
    }
}
```

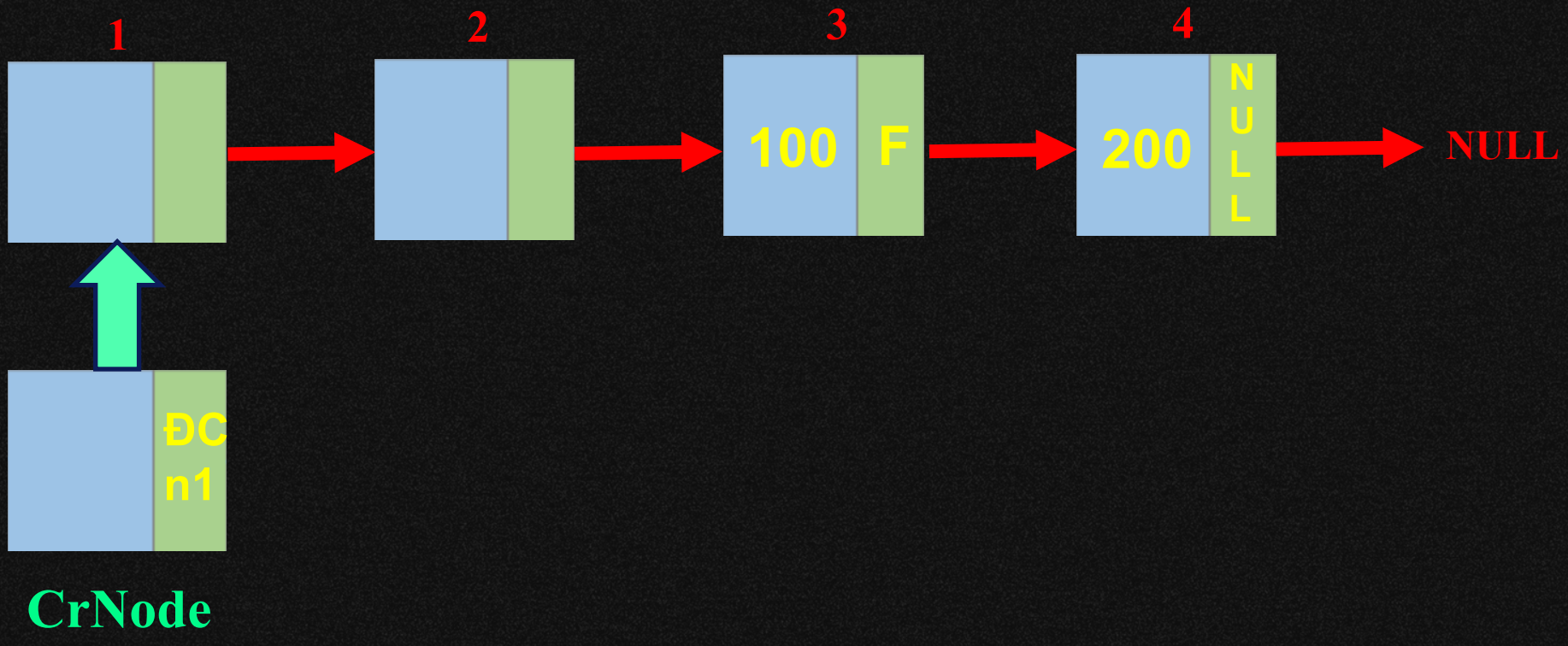
MẢNG RỖNG

```
int DeleteTail(LIST& L, int& x)
{

    if (L.pHead == NULL)
    {
        L.pTail = NULL;
        return 0;
    }
    else if (L.pHead->pNext == NULL)
    {
        return DeleteHead(L, x);
    }
    else
    {
        NODE* CurrentNode = L.pHead;

        // duyệt node đến trước vị trí Node kế cuối
        while (CurrentNode->pNext->pNext != NULL)
            CurrentNode = CurrentNode->pNext;

        // CurrentNode lúc này sẽ ở vị trí Node kế Node Tail

        NODE* p;
        x = L.pTail->Info;
        p = L.pTail;
        CurrentNode->pNext = NULL;
        L.pTail = CurrentNode;

        delete p;
        return 1;
    }
}
```

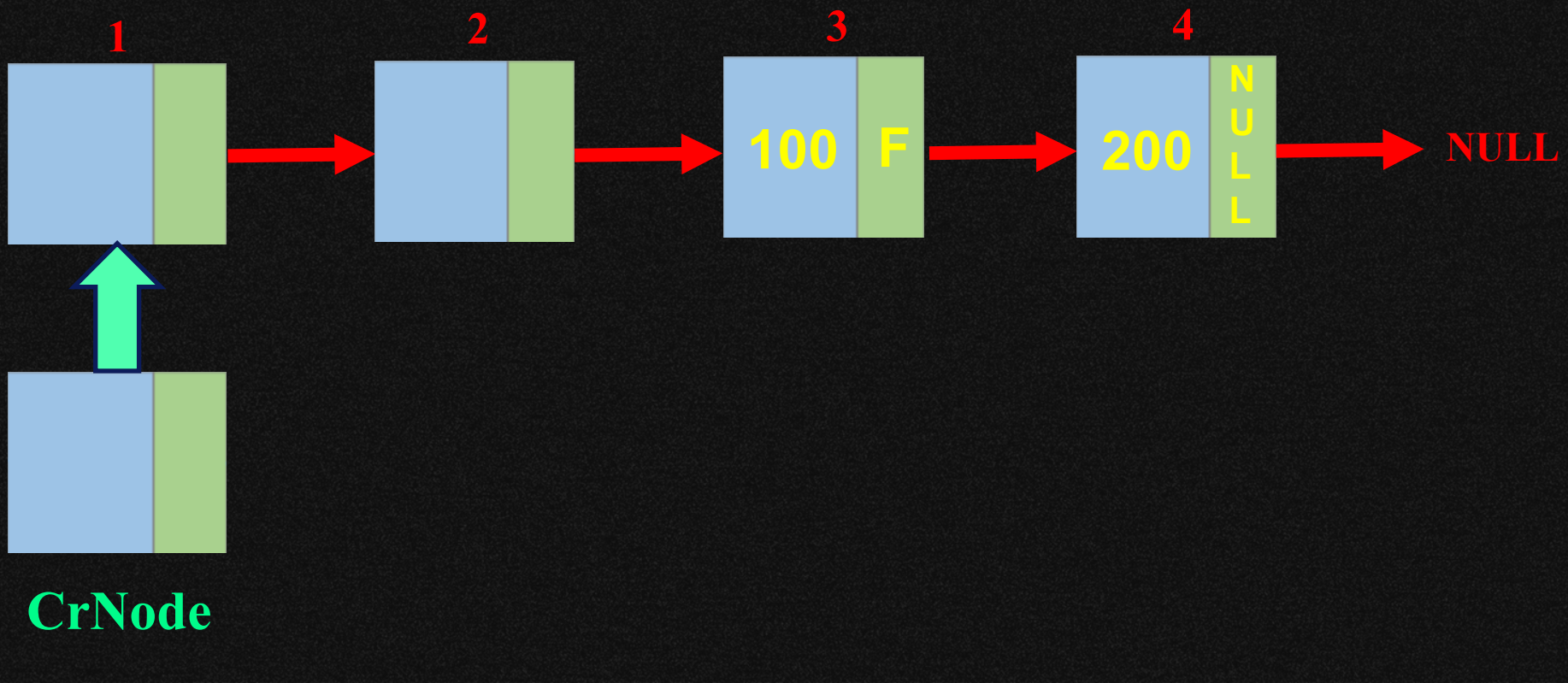
MẢNG CHỈ CÓ 1 NODE

```
int DeleteTail(LIST& L, int& x)
{

    if (L.pHead == NULL)
    {
        L.pTail = NULL;
        return 0;
    }
    else if (L.pHead->pNext == NULL)
    {
        return DeleteHead(L, x);
    }
    else
    {
        NODE* CurrentNode = L.pHead;

        // duyệt node đến trước vị trí Node kế cuối
        while (CurrentNode->pNext->pNext != NULL)
            CurrentNode = CurrentNode->pNext;

        // CurrentNode lúc này sẽ ở vị trí Node kế Node Tail

        NODE* p;
        x = L.pTail->Info;
        p = L.pTail;
        CurrentNode->pNext = NULL;
        L.pTail = CurrentNode;

        delete p;
        return 1;
    }
}
```

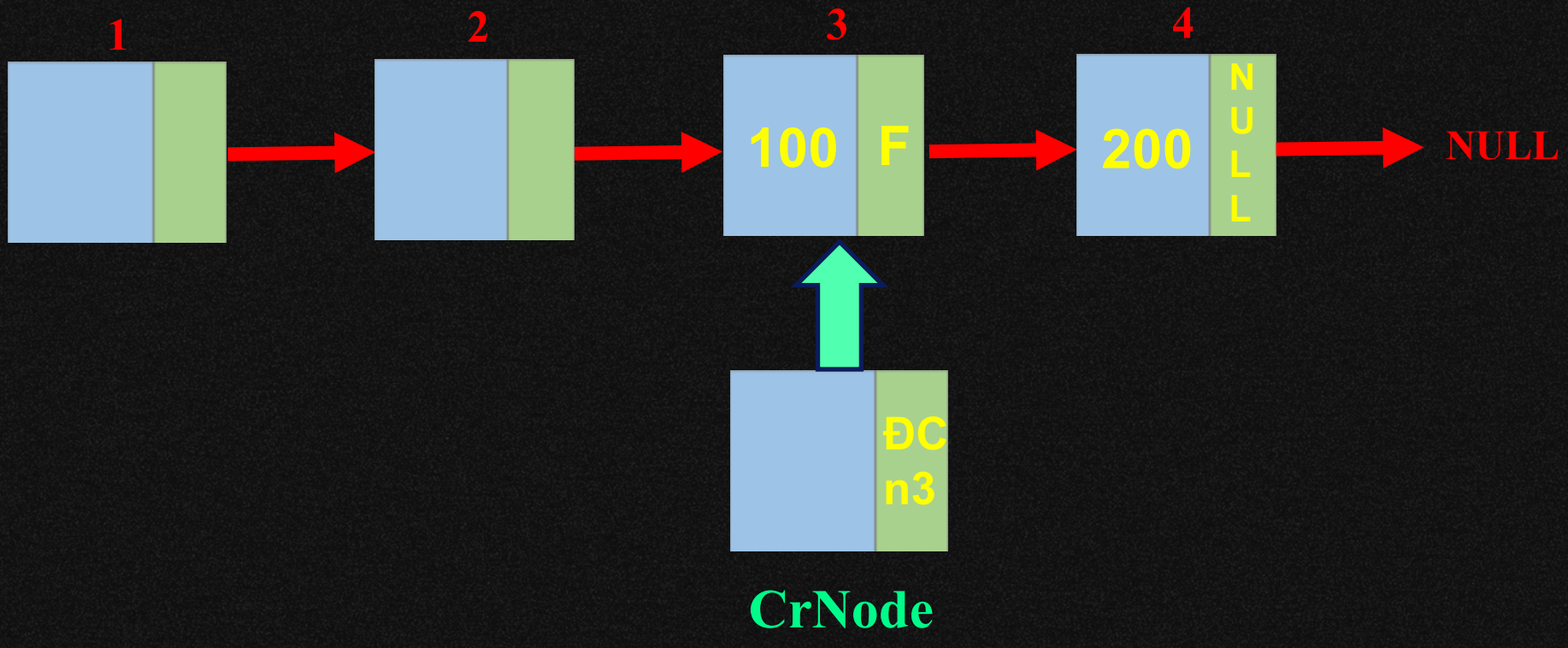
```
NODE* CurrentNode = L.pHead;
```



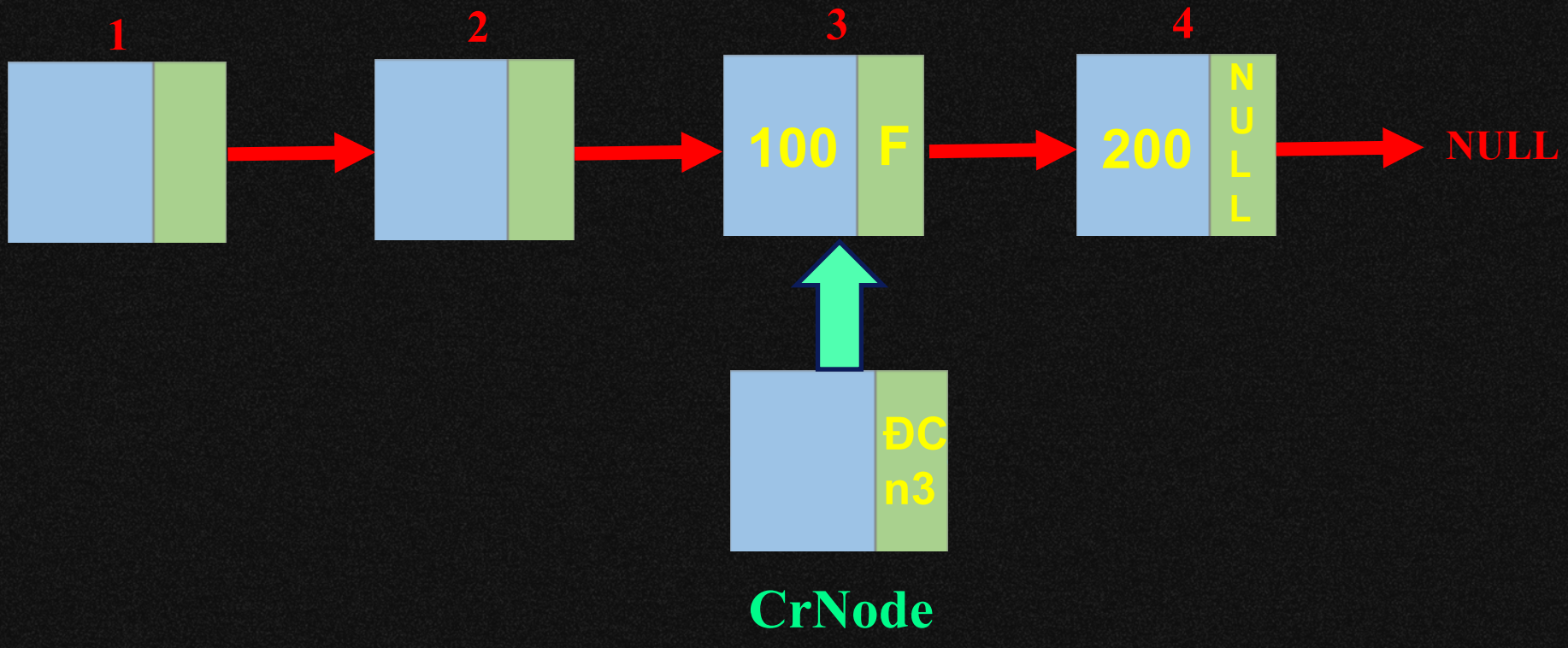
```
// duyệt node đến trước vị trí Node kế cuối
while (CurrentNode->pNext->pNext != NULL)
    CurrentNode = CurrentNode->pNext;

// CurrentNode lúc này sẽ ở vị trí Node kế Node Tail
```



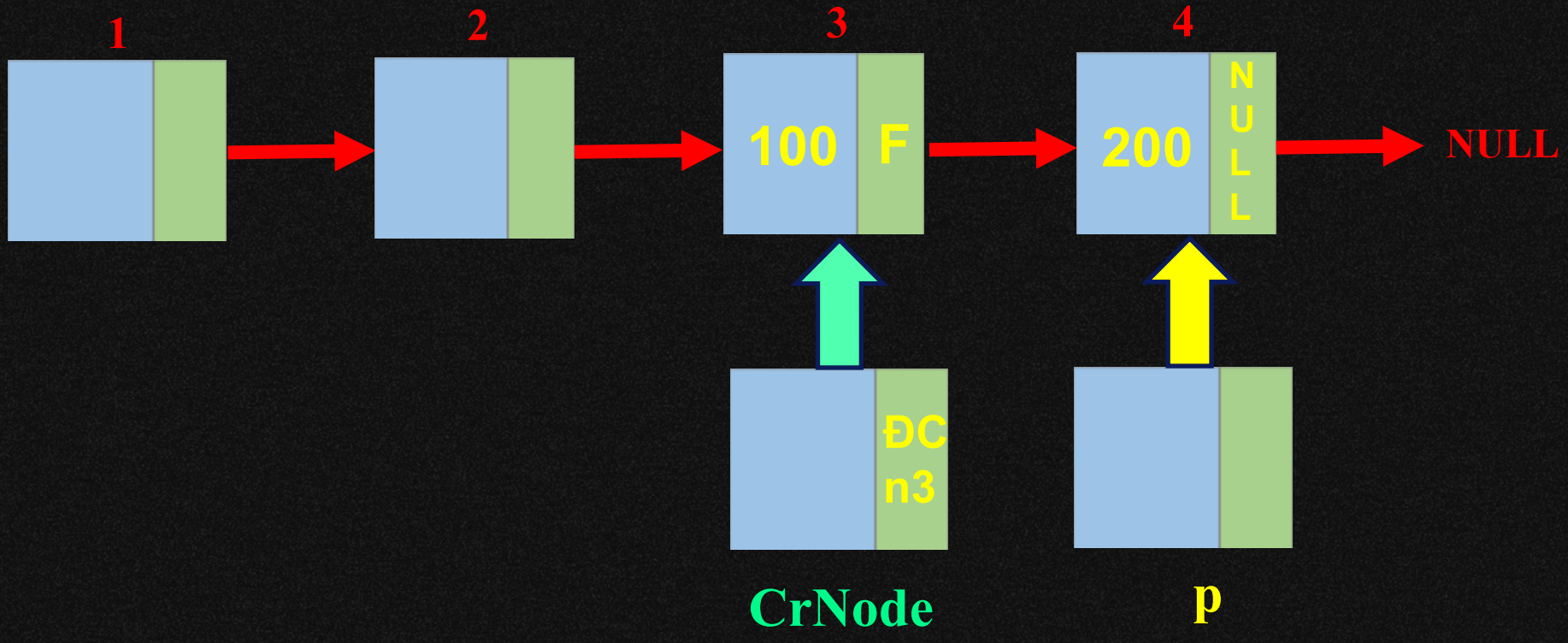
```
// duyệt node đến trước vị trí Node kế cuối
while (CurrentNode->pNext->pNext != NULL)
    CurrentNode = CurrentNode->pNext;

// CurrentNode lúc này sẽ ở vị trí Node kế Node Tail
```



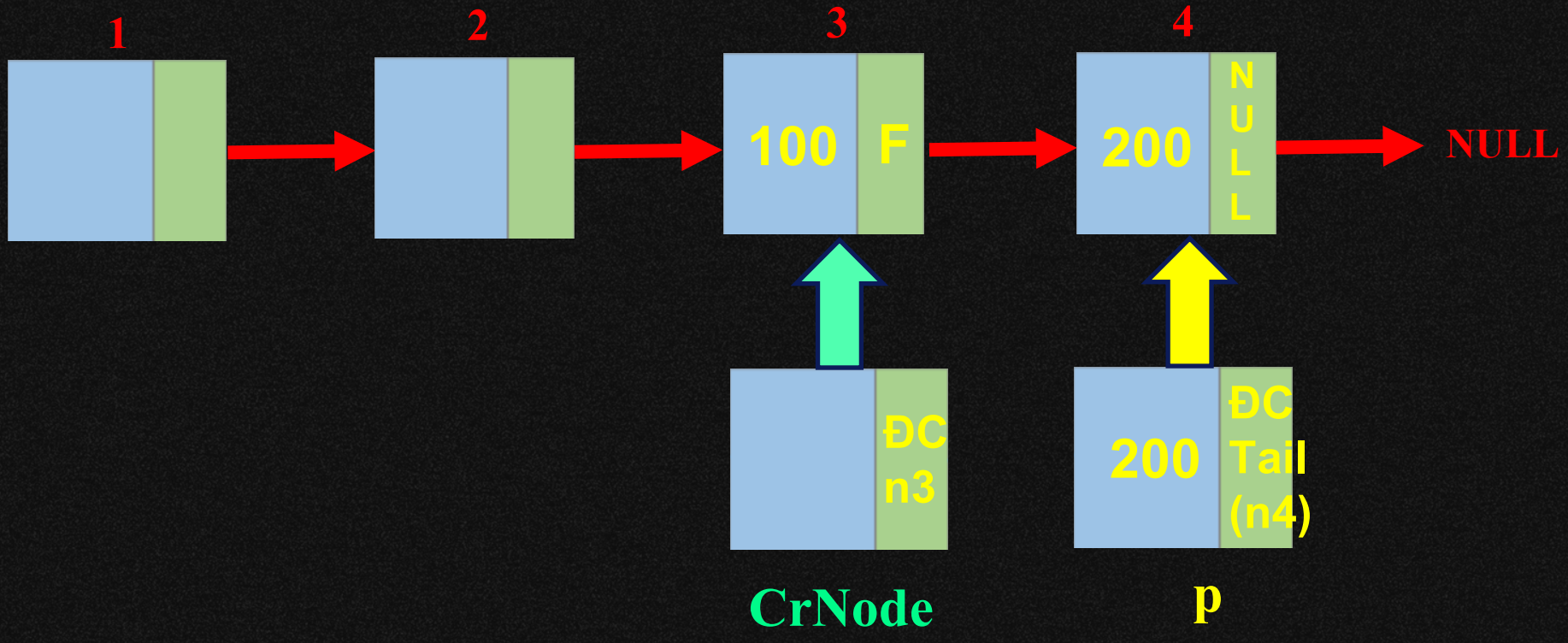
```
// duyệt node đến trước vị trí Node kế cuối
while (CurrentNode->pNext->pNext != NULL)
    CurrentNode = CurrentNode->pNext;

// CurrentNode lúc này sẽ ở vị trí Node kế Node Tail
```



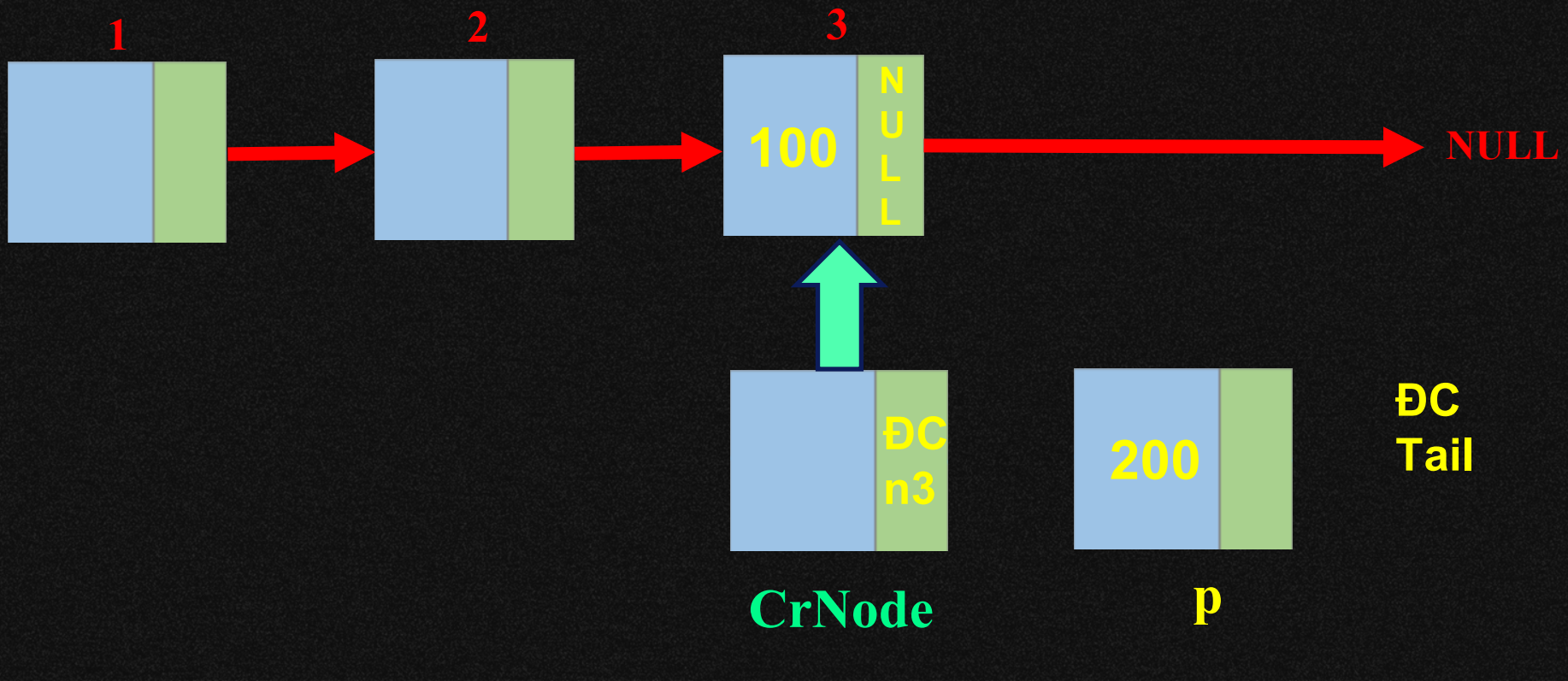
```
NODE* p;
x = L.pTail->Info;
p = L.pTail;
```



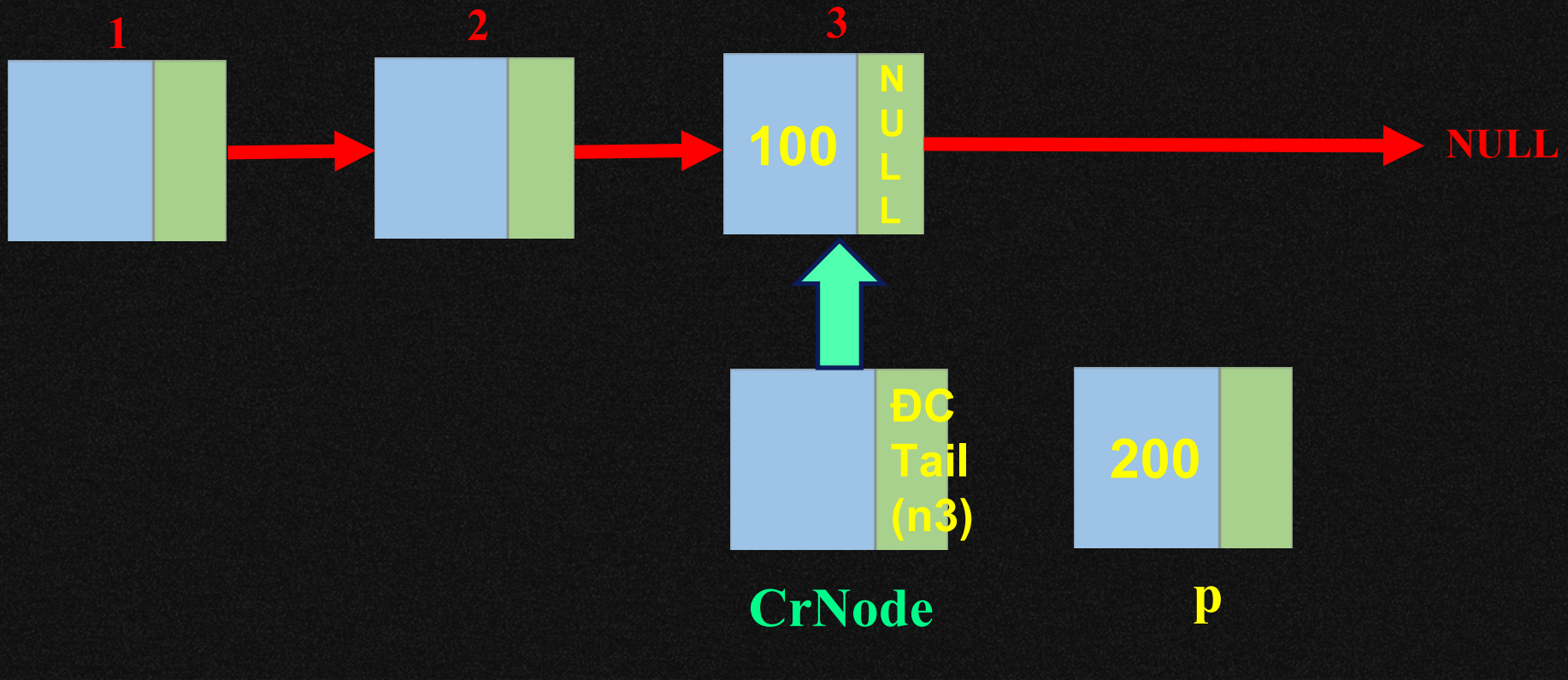
```
NODE* p;
x = L.pTail->Info;
p = L.pTail;
```



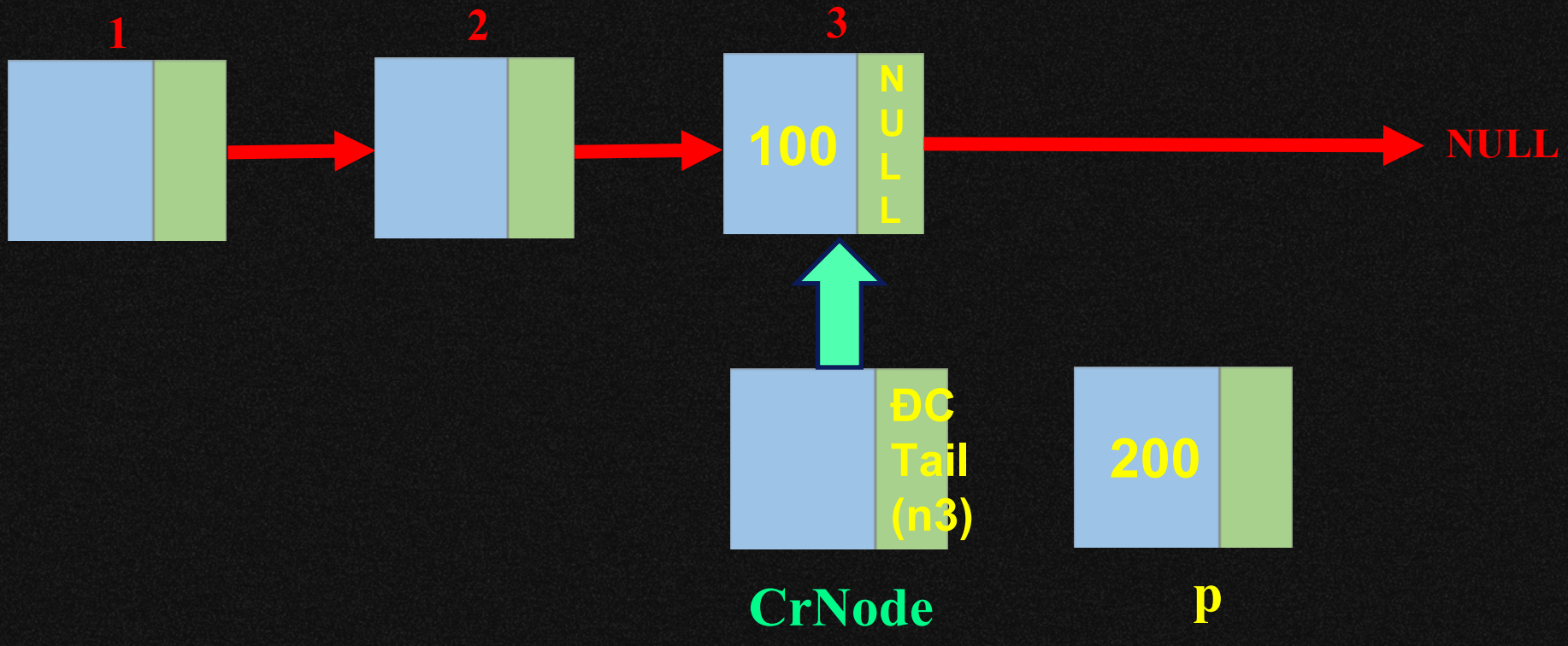
```
CurrentNode->pNext = NULL;
L.pTail = CurrentNode;
```



```
CurrentNode->pNext = NULL;
L.pTail = CurrentNode;
```



```
delete p;
```



Thuật toán hủy phần tử đứng sau phần tử Q

```
int RemoveAfterQ(LIST& L, NODE* Q, int& x)
{
    NODE* p;
    if (Q != NULL)
    {
        // p giữ địa chỉ node cần xóa
        p = Q->pNext;

        // Nếu p không phải là node cuối => xóa node cuối
        if (p != NULL)
        {
            if (p == L.pTail)
                L.pTail = Q;
            Q->pNext = p->pNext;
            x = p->Info;
            delete p;
        }
        return 1;
    }
    return 0;
}
```

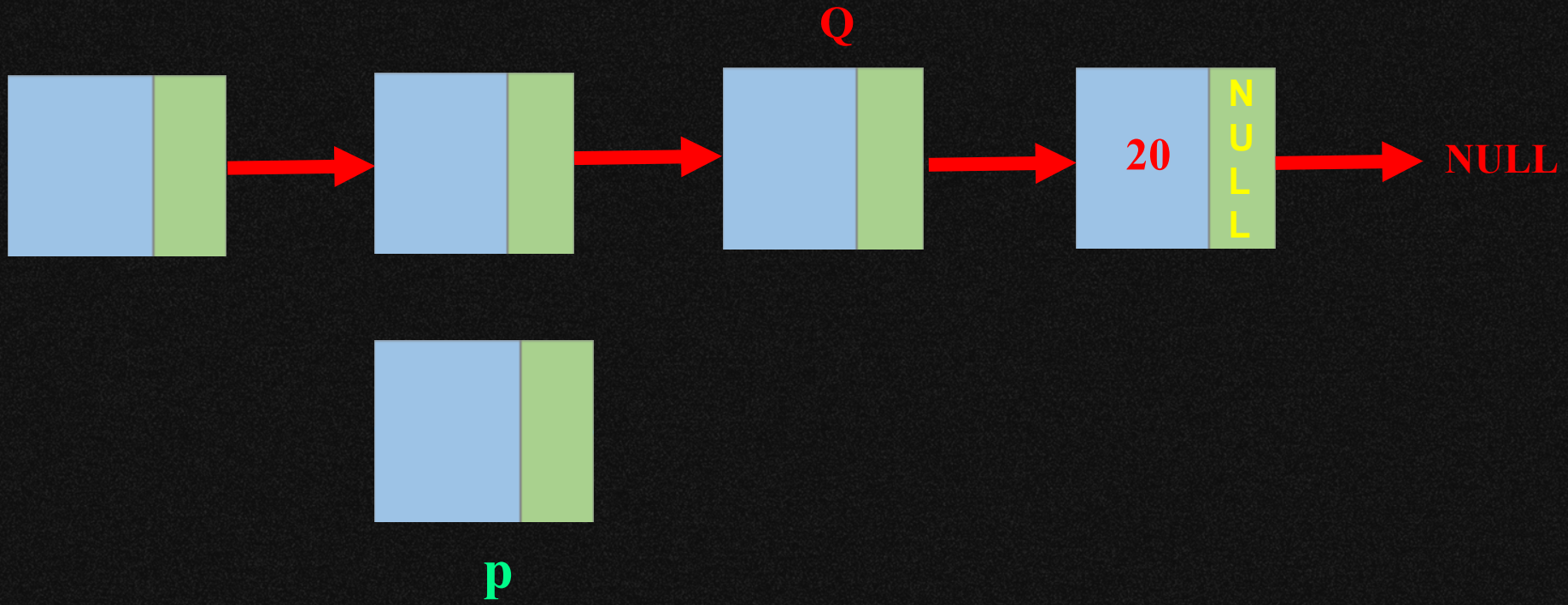
Nếu (Q != NULL) thì: Q tồn tại trong List

- B1: p = Q ->pNext; (NODE thế mạng)

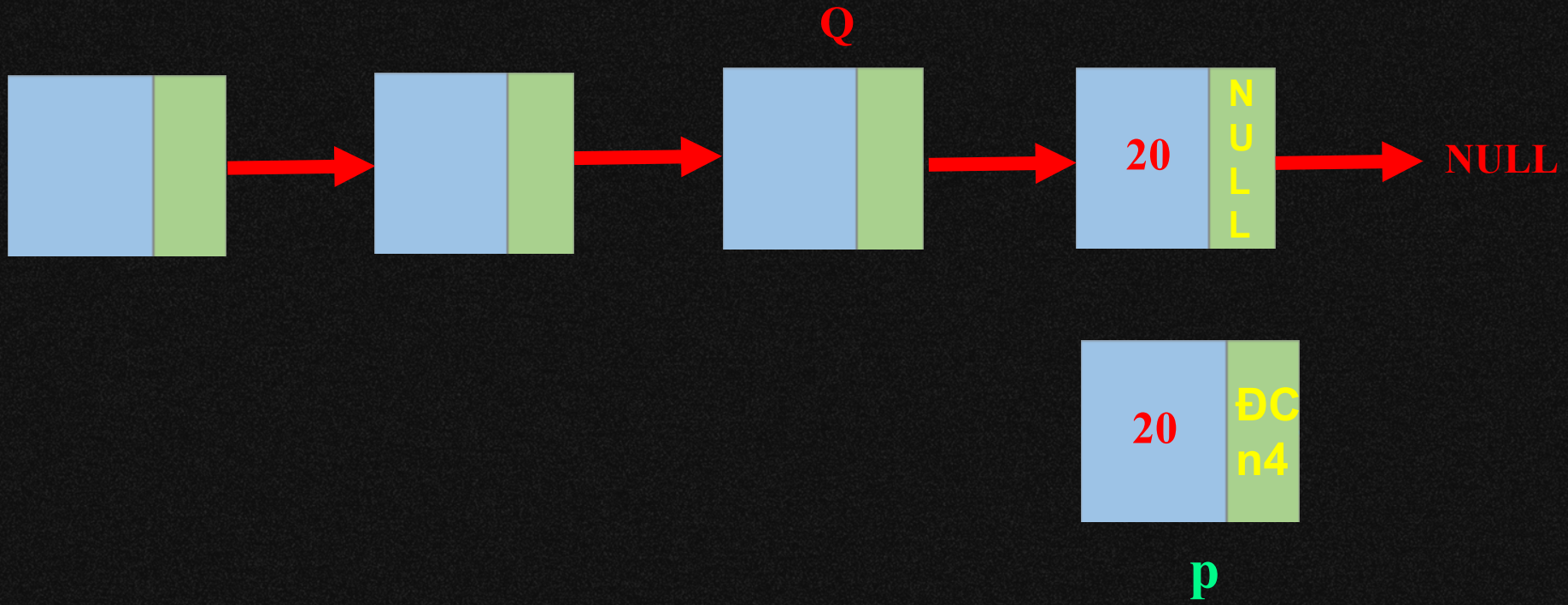
- B2: Nếu (p != NULL) thì Q không là phần tử cuối
  + Q ->pNext=p->pNext; // tách p ra khỏi xâu
  + Nếu (p == pTail) // nút cần hủy là nút cuối
  pTail= Q;
  + delete p; // hủy p

```cpp
int RemoveAfterQ(LIST& L, NODE* Q, int& x)
{
    NODE* p;
    if (Q != NULL)
    {
        // p giữ địa chỉ node cần xóa
        p = Q->pNext;

        // Nếu p không phải là node cuối => xóa node cuối
        if (p != NULL)
        {
            if (p == L.pTail)
                L.pTail = Q;
            Q->pNext = p->pNext;
            x = p->Info;
            delete p;
        }
        return 1;
    }
    return 0;
}
```

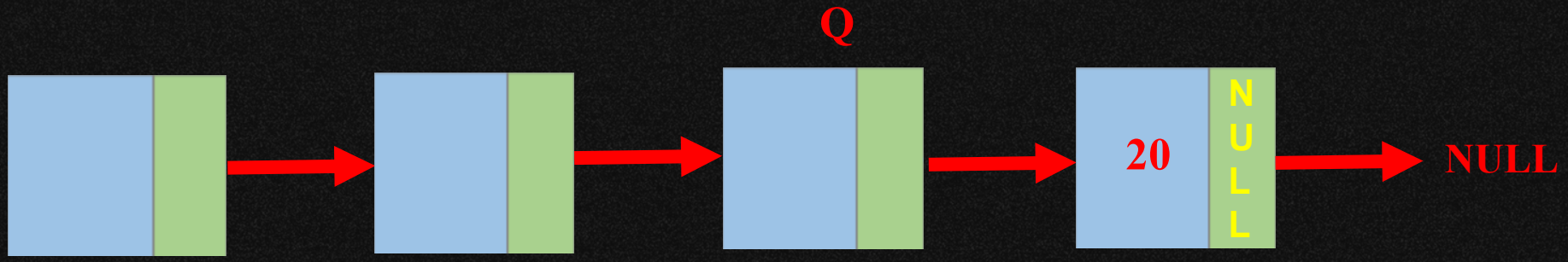
```
if (Q != NULL)
{
    // p giữ địa chỉ node cần xóa
    p = Q->pNext;
```



```
if (Q != NULL)
{
    // p giữ địa chỉ node cần xóa
    p = Q->pNext;
```

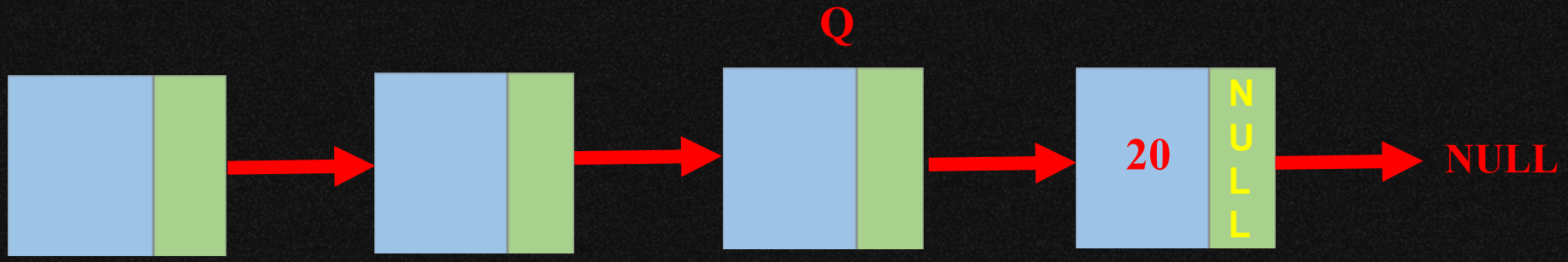


```
if (p != NULL)
{
    if (p == L.pTail)
        L.pTail = Q;
```



P == L.pTail

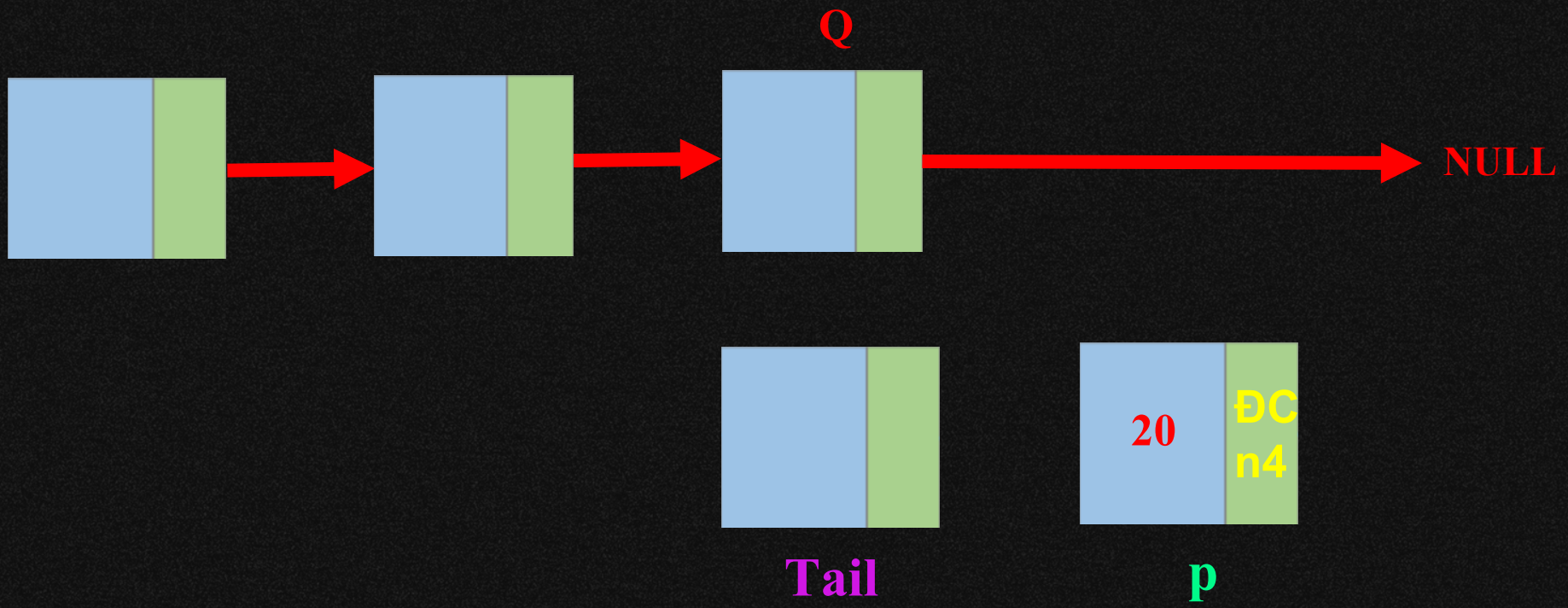
```
if (p != NULL)
{
    if (p == L.pTail)
        L.pTail = Q;
```



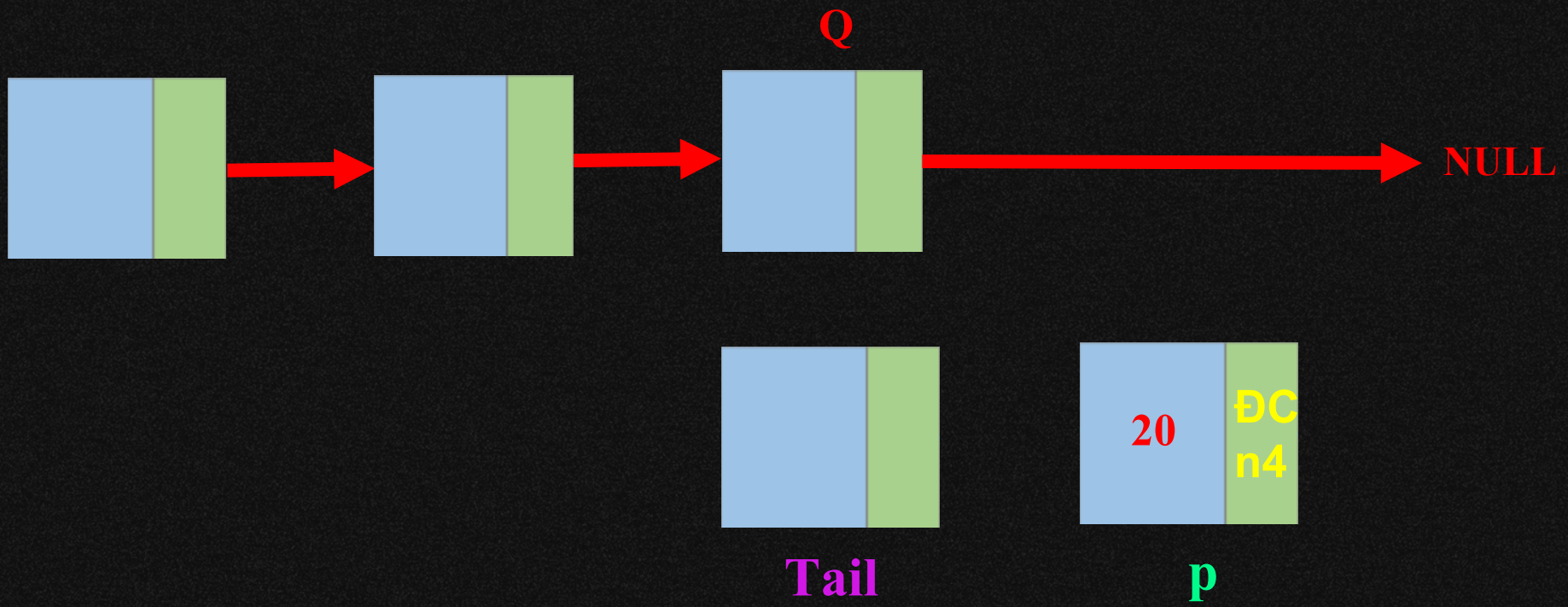
p ≡ Tail

p ≡ Tail

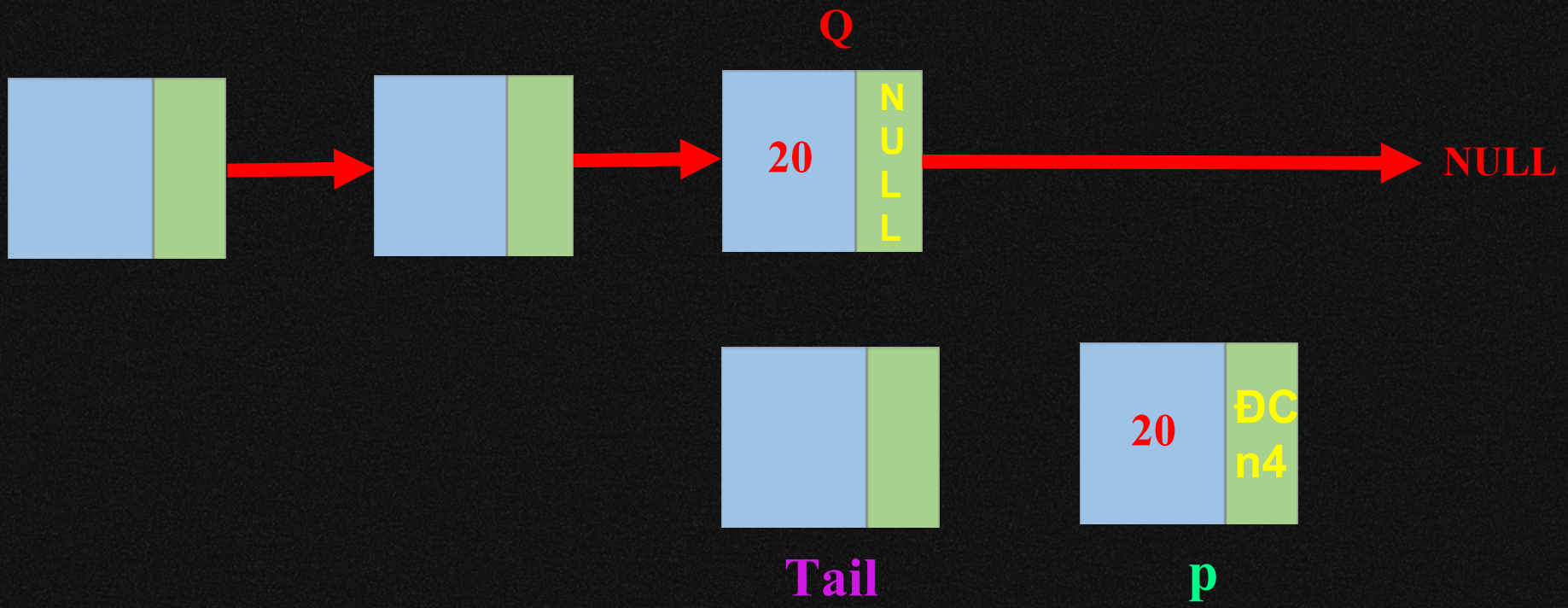
```
if (p != NULL)
{
    if (p == L.pTail)
        L.pTail = Q;
```



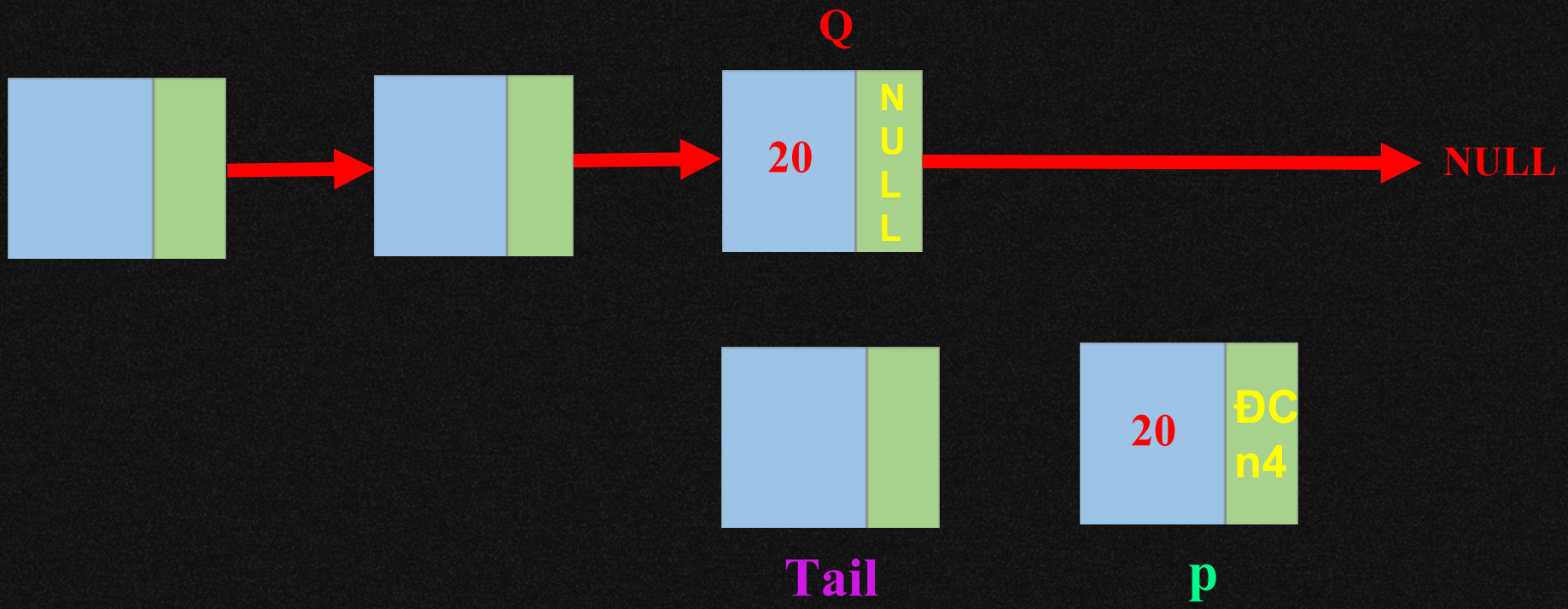
```
Q->pNext = p->pNext;
x = p->Info;
```



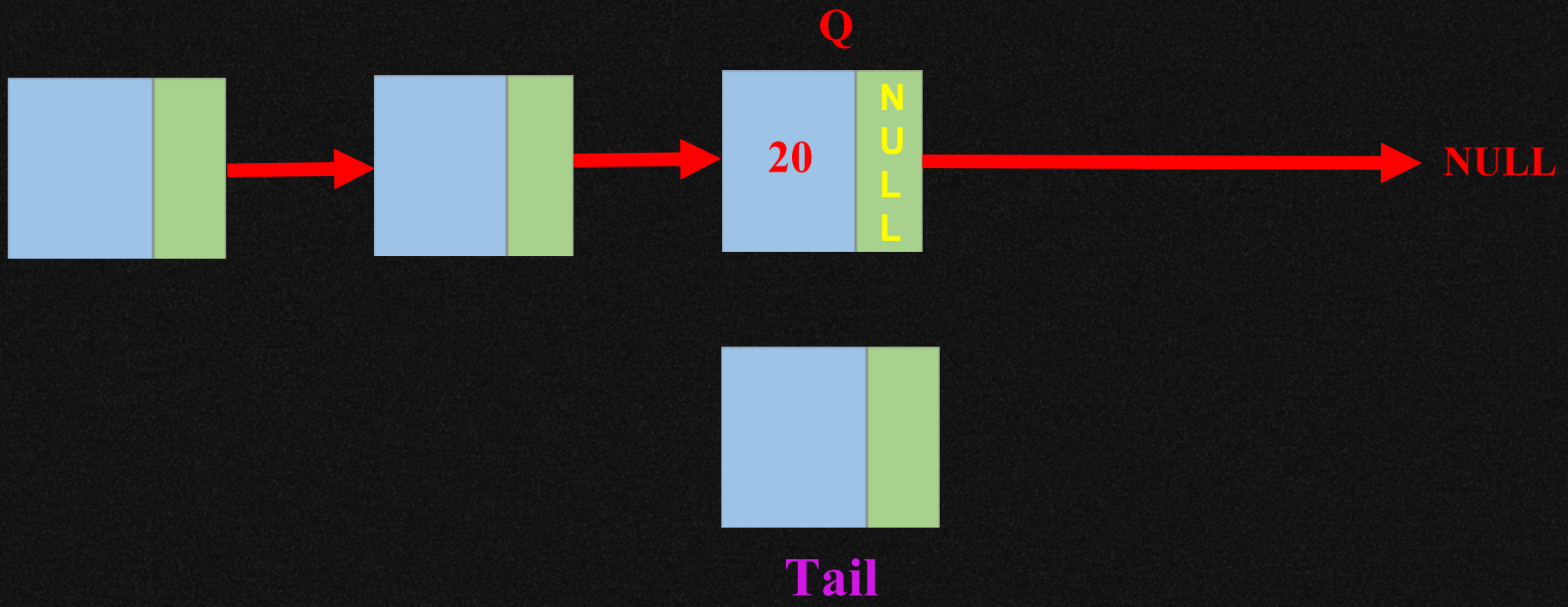
```
Q->pNext = p->pNext;
x = p->Info;
```



```
delete p;
```



```
delete p;
```



```
int  RemoveX(LIST& L, int x)
{
    NODE* Q, *p;
    Q = NULL;
    p = L.pHead;
    while (p != NULL && p->Info != x)
    {
        Q = p;
        p = p->pNext;
    }

    if (p == NULL)
        return 0;
    if (Q != NULL)
        RemoveAfterQ(L, Q, x);
    else
        DeleteHead(L, x);
    return 1;
}
```

Xóa vị trí có khóa x trong danh sách

Bước 1: Tìm phần tử p có khoá bằng x, và Q đứng trước p

Bước 2:

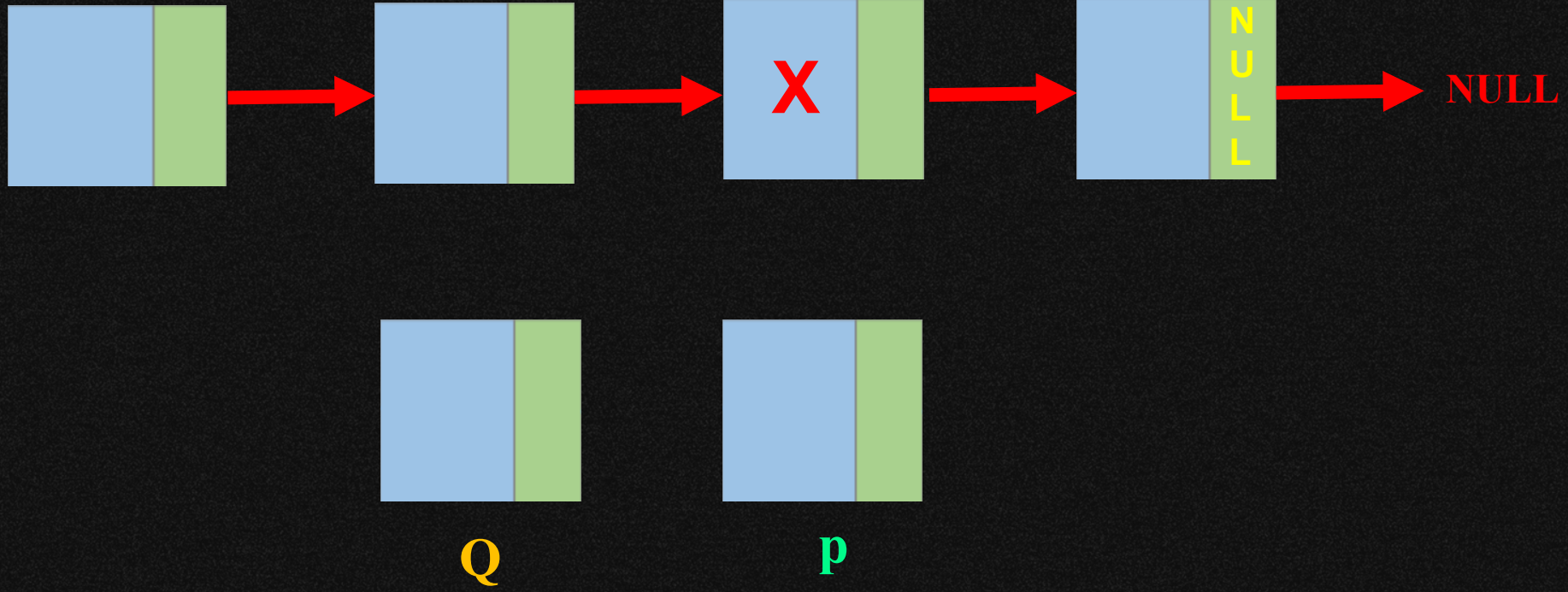
+ Nếu (p!=NULL) thì //tìm thấy phần tử có khoá bằng x
+ Hủy p ra khỏi List bằng cách hủy phần tử đứng sau Q

Ngược lại: Báo không tìm thấy phần tử có khoá x

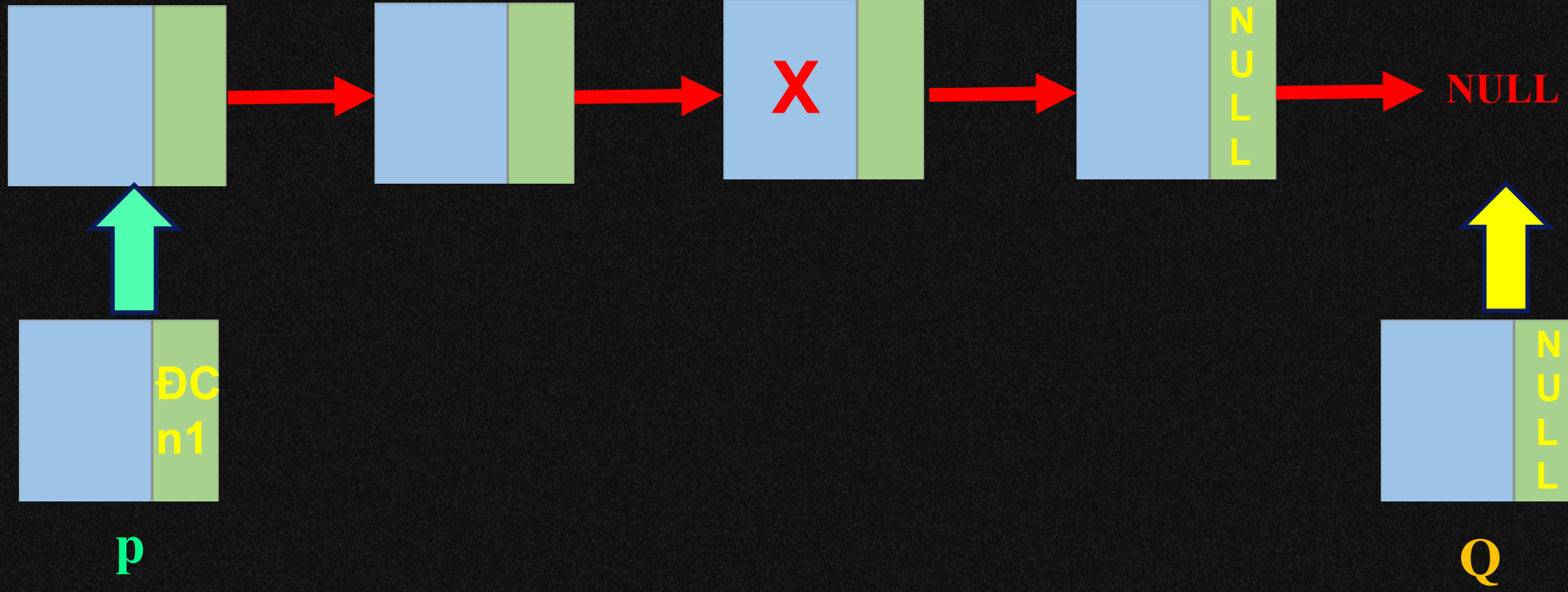
```
int  RemoveX(LIST& L, int x)
{
    NODE* Q, *p;
    Q = NULL;
    p = L.pHead;
    while (p != NULL && p->Info != x)
    {
        Q = p;
        p = p->pNext;
    }

    if (p == NULL)
        return 0;
    if (Q != NULL)
        RemoveAfterQ(L, Q, x);
    else
        DeleteHead(L, x);
    return 1;
}
```

// Tìm không thấy x

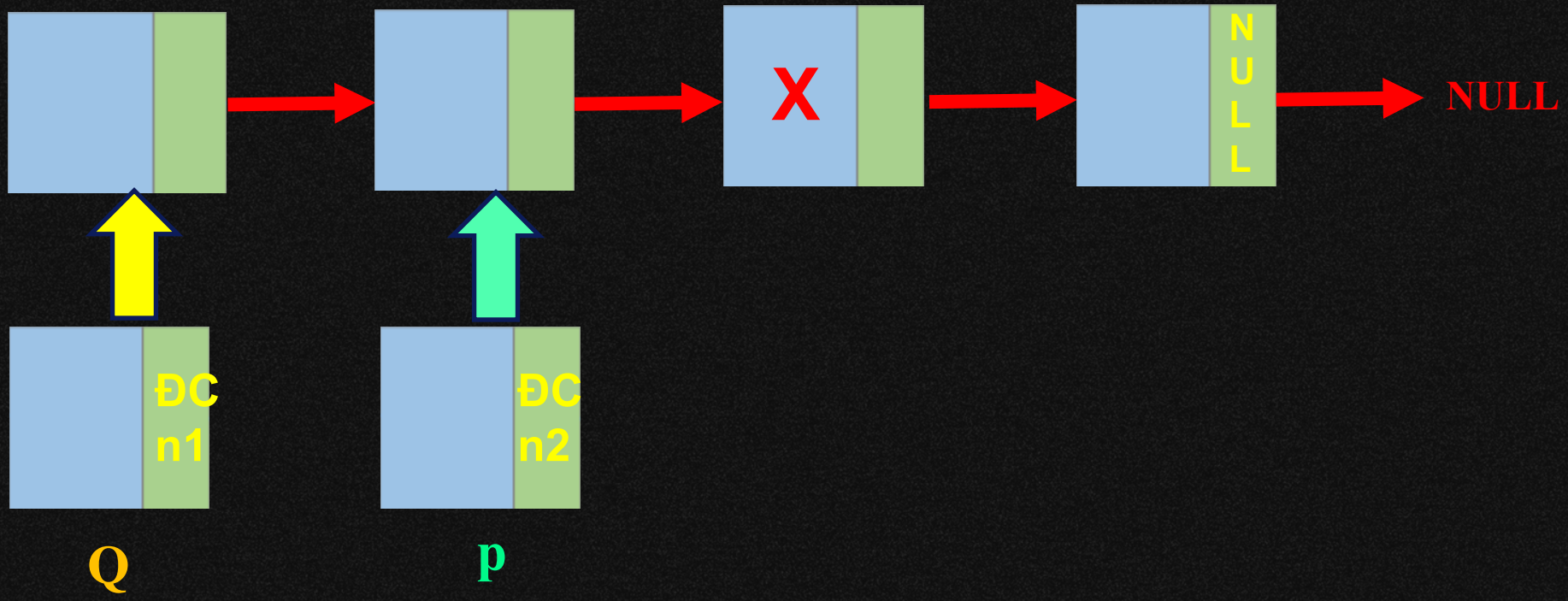
```
NODE* Q, *p;
```



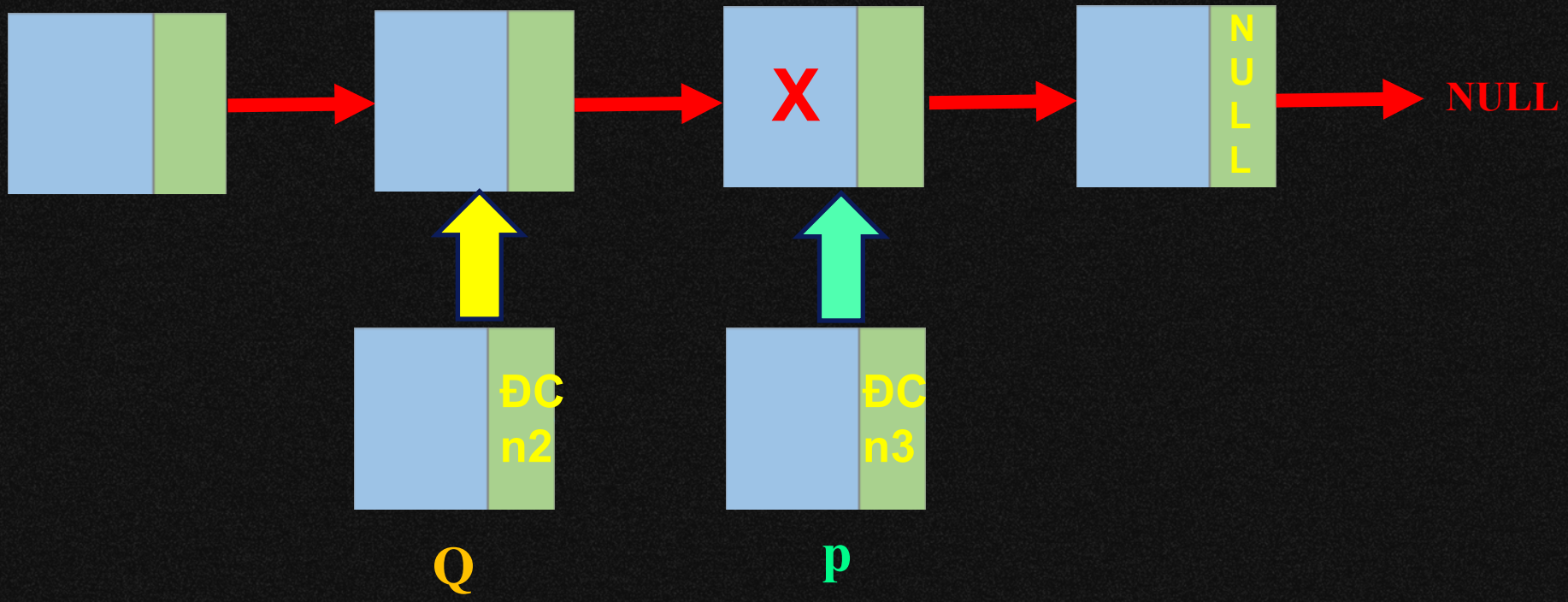
```
Q = NULL;
p = L.pHead;
```



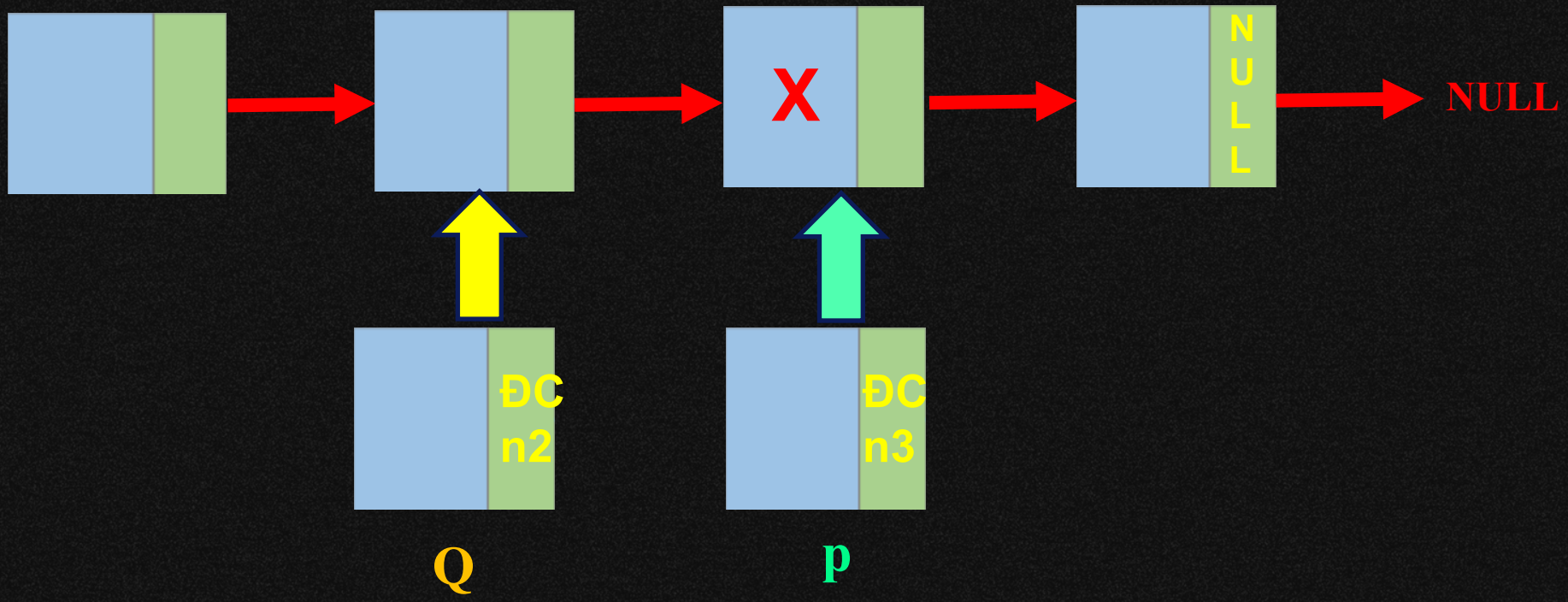
```
while (p != NULL && p->Info != x)
{
    Q = p;
    p = p->pNext;
}
```



```
while (p != NULL && p->Info != x)
{
    Q = p;
    p = p->pNext;
}
```



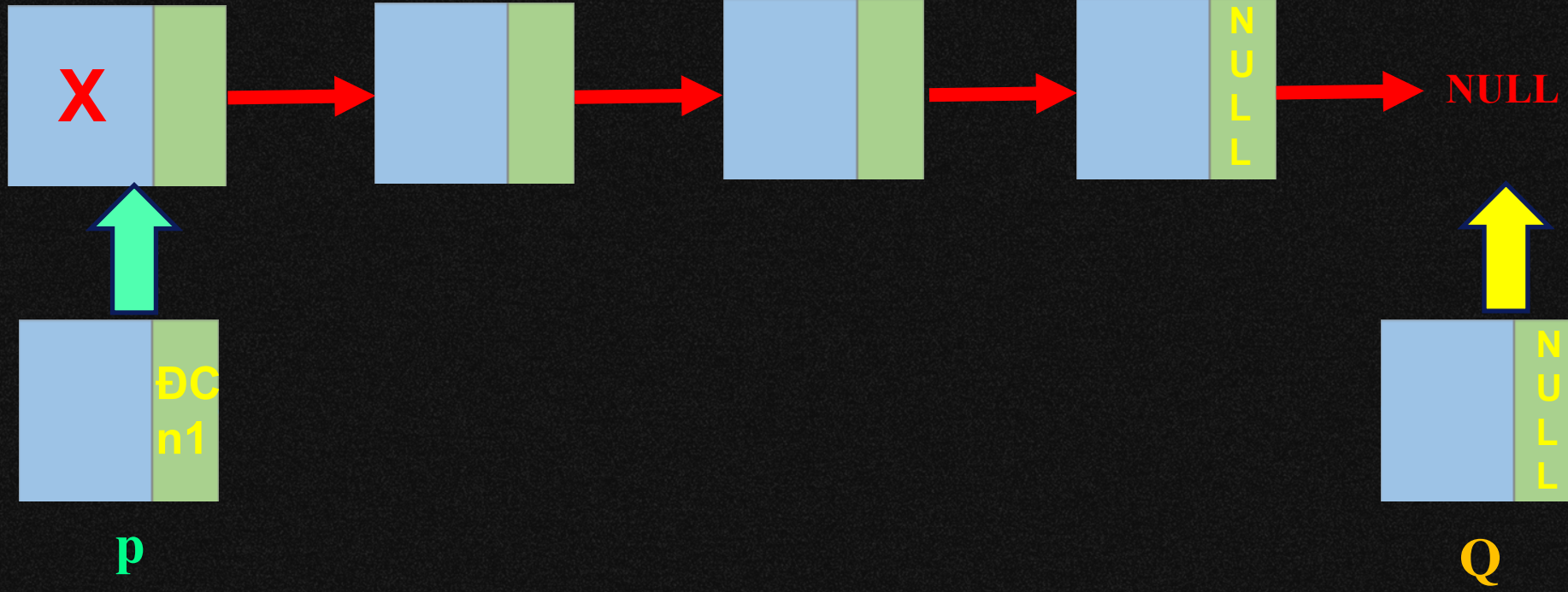
```
if (Q != NULL)
    RemoveAfterQ(L, Q, x);
```



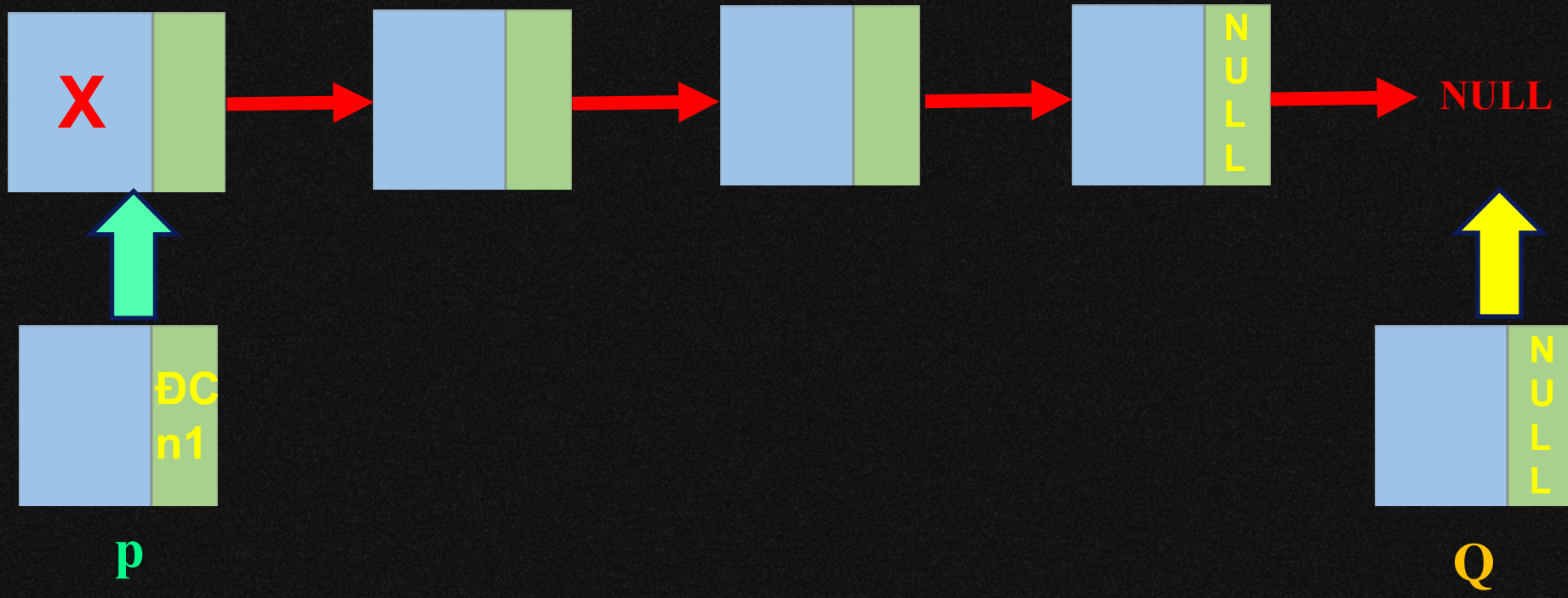
```
if (Q != NULL)
    RemoveAfterQ(L, Q, x);
```

TH: X ở node đầu

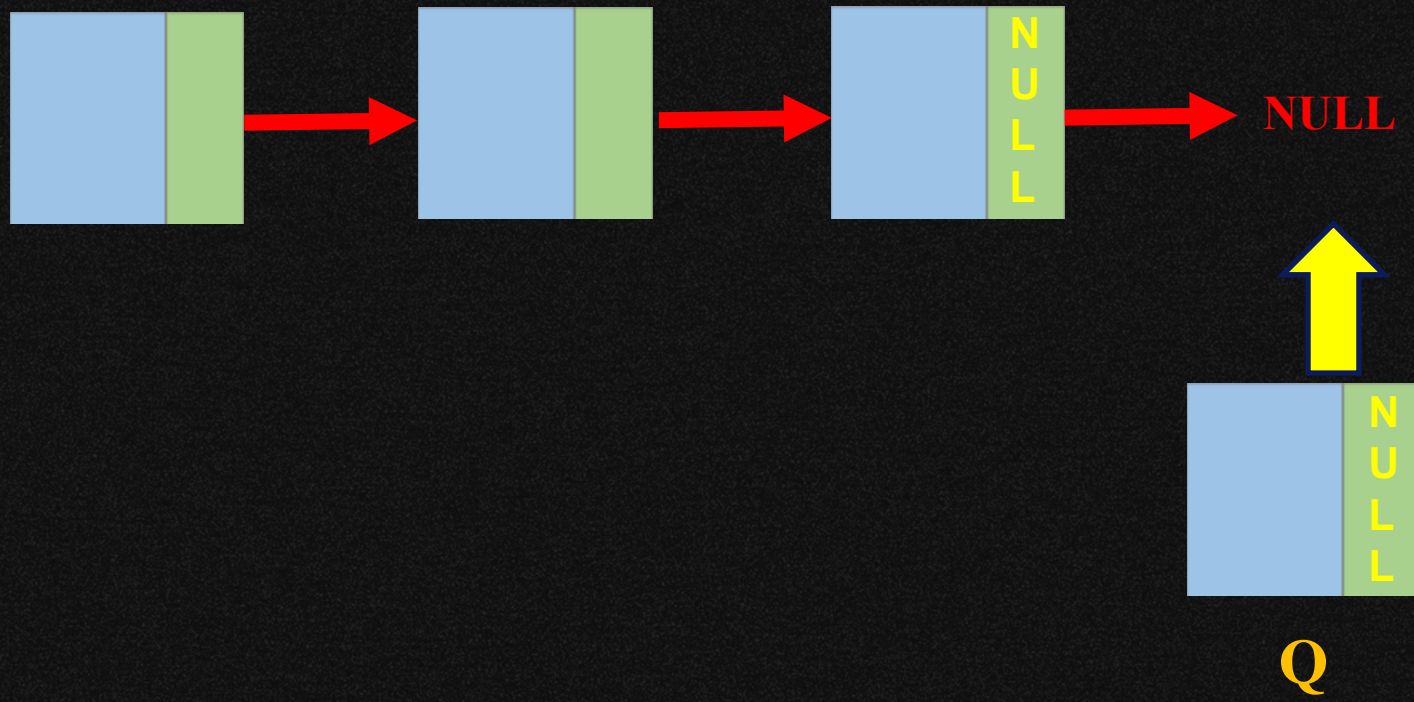
```
Q = NULL;
p = L.pHead;
```



```
while (p != NULL && p->Info != x)
{
    Q = p;
    p = p->pNext;
}
```



```
else // Q == NULL
    DeleteHead(L, x);
```



Có tìm kiếm trên linked list bằng binary search được không?

Vì sao ngta ưu tiên sài linear search thay binary search?

Sắp xếp danh sách bằng Selection Sort

CÓ 2 CÁCH TIẾP CẬN:

C1: Thay đổi thành phần Info

C2: Thay đổi thành phần pNext (thay đổi trình tự móc nối sao cho tạo nên thứ tự mong muốn)

Ưu và nhược điểm của 2 cách

	Cách 1	Cách 2
ƯU:	Đơn giản, như sắp xếp mảng	- Kích thước mảng không thay đổi - Thao tác nhanh

Ưu và nhược điểm của 2 cách

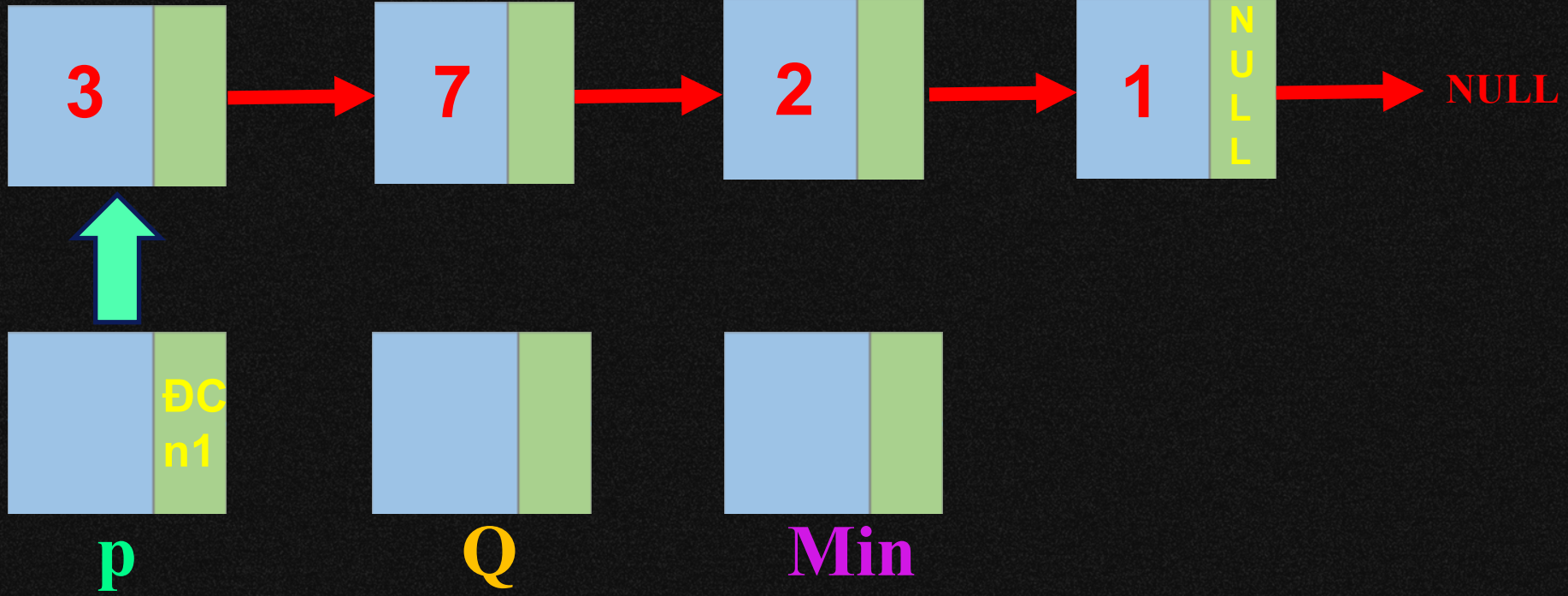
	Cách 1	Cách 2
NHƯỢC:	Đòi hỏi thêm vùng nhớ khi hoán đổi thành phần data của 2 phần tử -> thích hợp với những dslk có info nhỏ Khi kích thước info lớn => chi phí thực hiện cũng lớn Làm thao tác chậm lớn	Cài đặt phức tạp

Ưu và nhược điểm của 2 cách

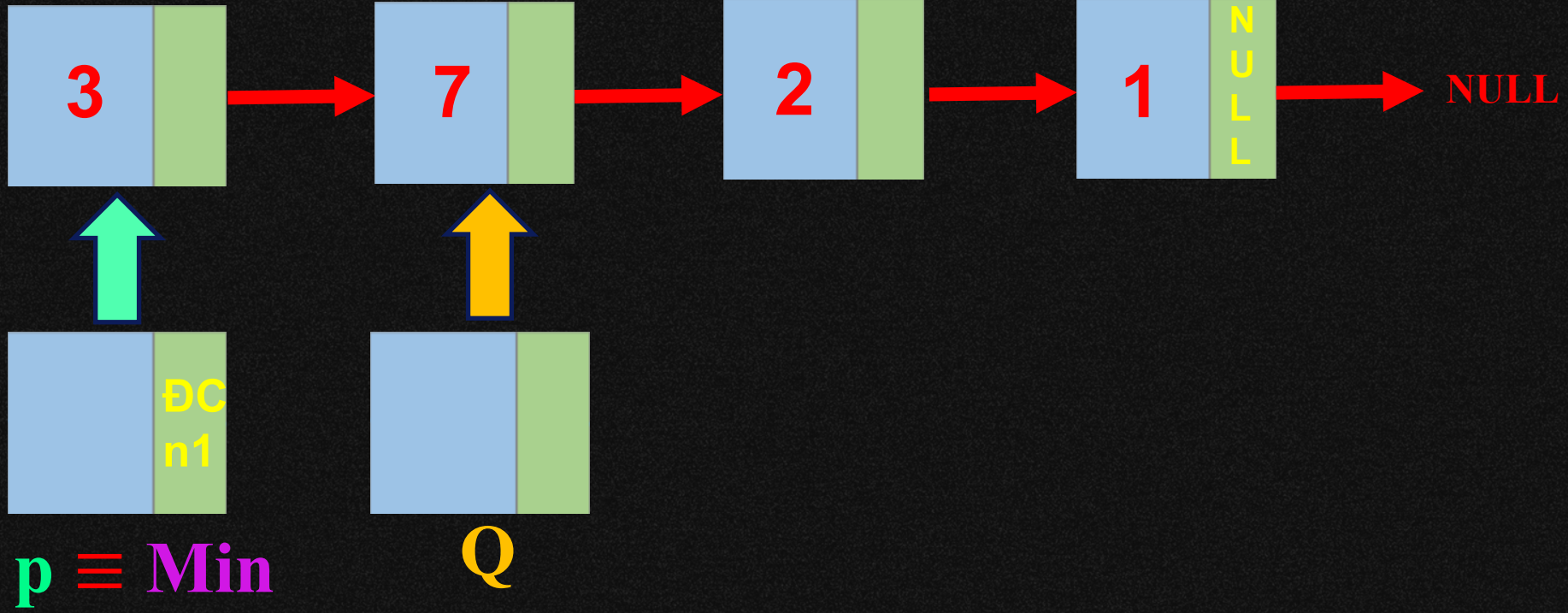
	Cách 1	Cách 2
ƯU:	Đơn giản, như sắp xếp mảng	Kích thước mảng không thay đổi Thao tác nhanh
NHƯỢC:	Đòi hỏi thêm vùng nhớ khi hoán đổi thành phần data của 2 phần tử -> thích hợp với những dslk có info nhỏ Khi kích thước info lớn => chi phí thực hiện cũng lớn Làm thao tác chậm lớn	Cài đặt phức tạp

```
void SelectionSortList(LIST& L)
{
    NODE* P, *Q, * MIN;
    P = L.pHead;
    while (P != L.pTail)
    {
        MIN = P;
        Q = P->pNext;
        while (Q != NULL)
        {
            if (Q->Info < MIN->Info)
                MIN = Q;
            Q = Q->pNext;
        }
        swap(MIN->Info, P->Info);
        P = P->pNext;
    }
}
```

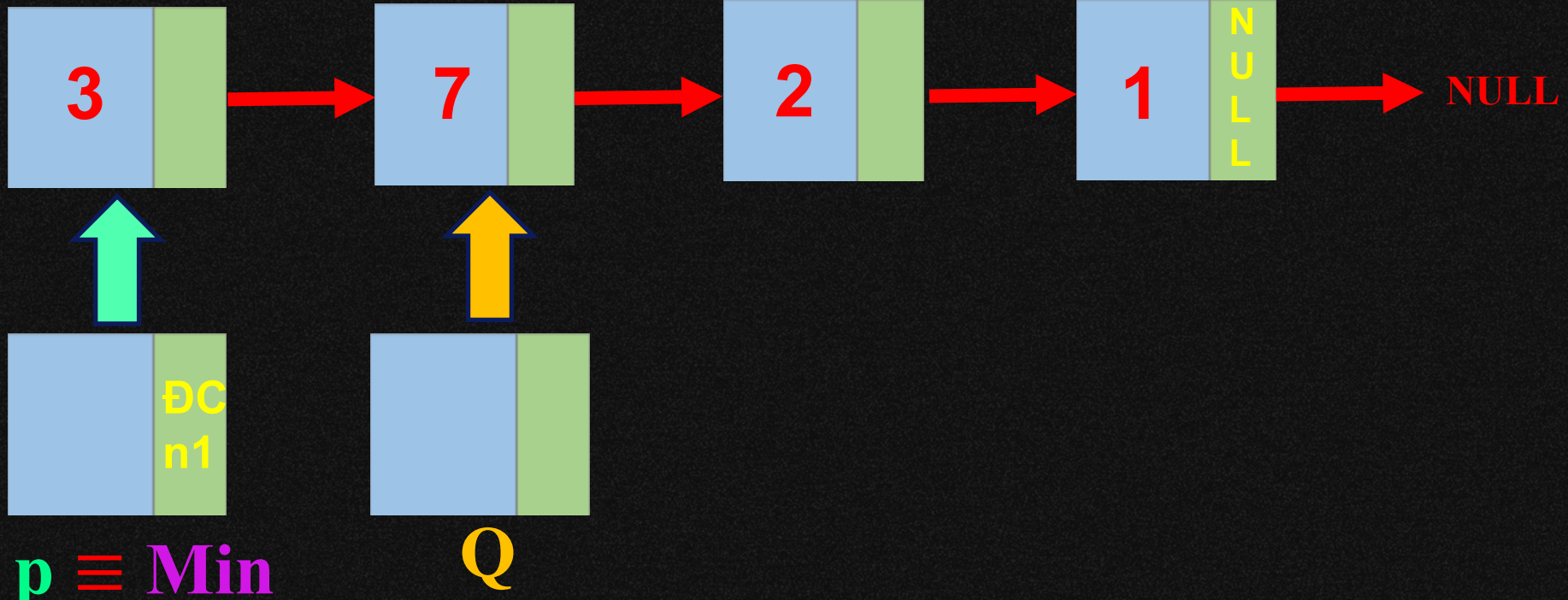
```
NODE* P, *Q, * MIN;
P = L.pHead;
```



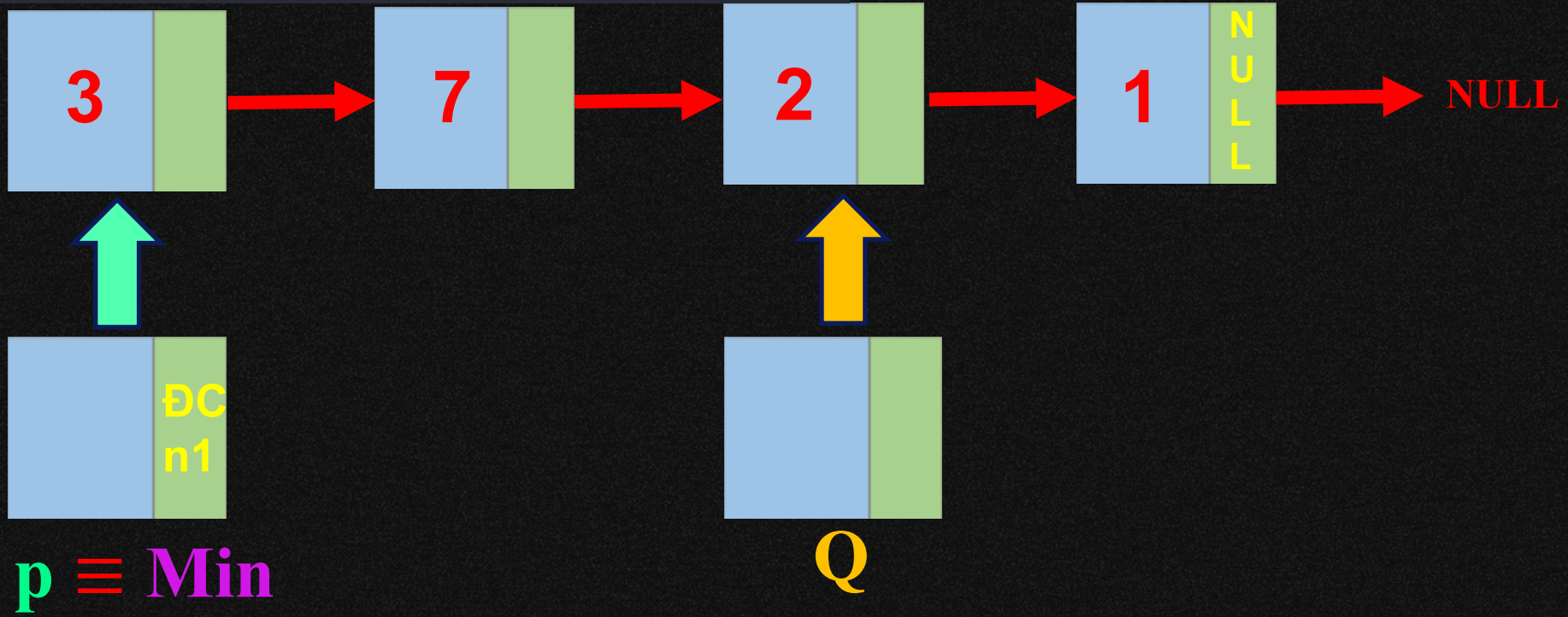
```
while (P != L.pTail)
{
    MIN = P;
    Q = P->pNext;
```



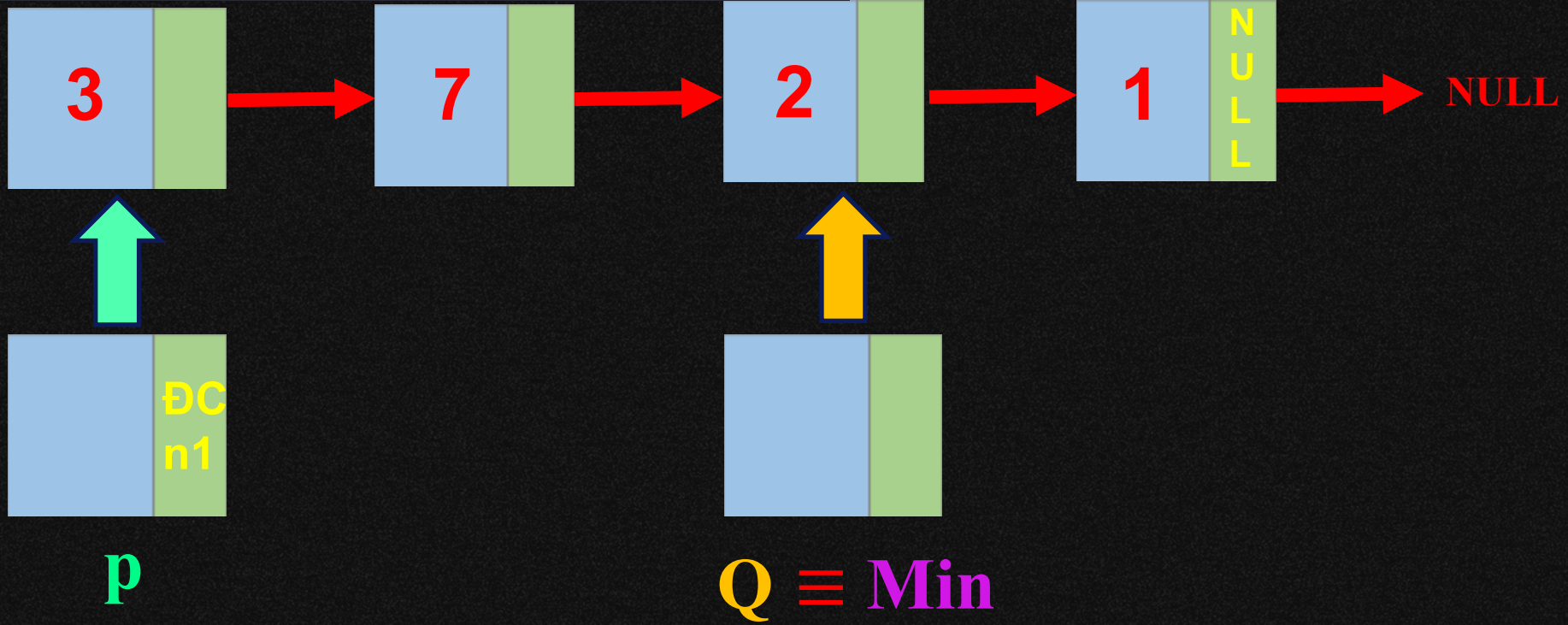
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



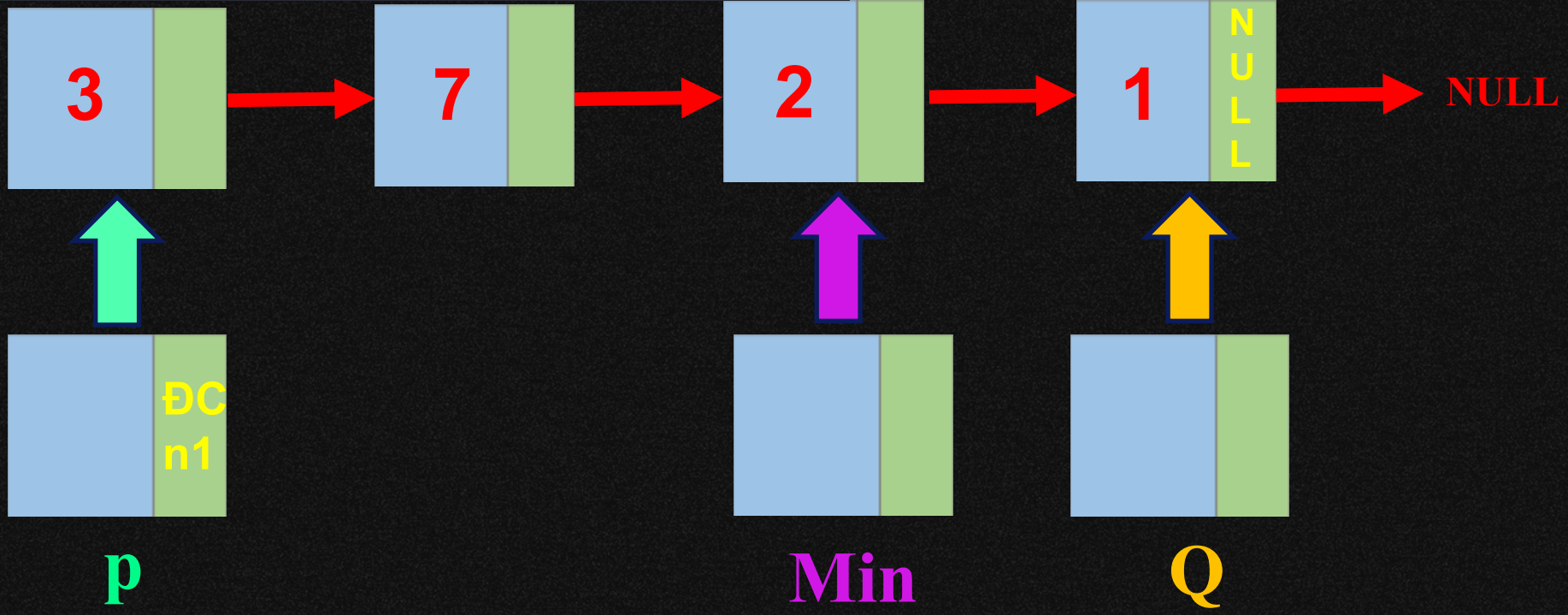
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



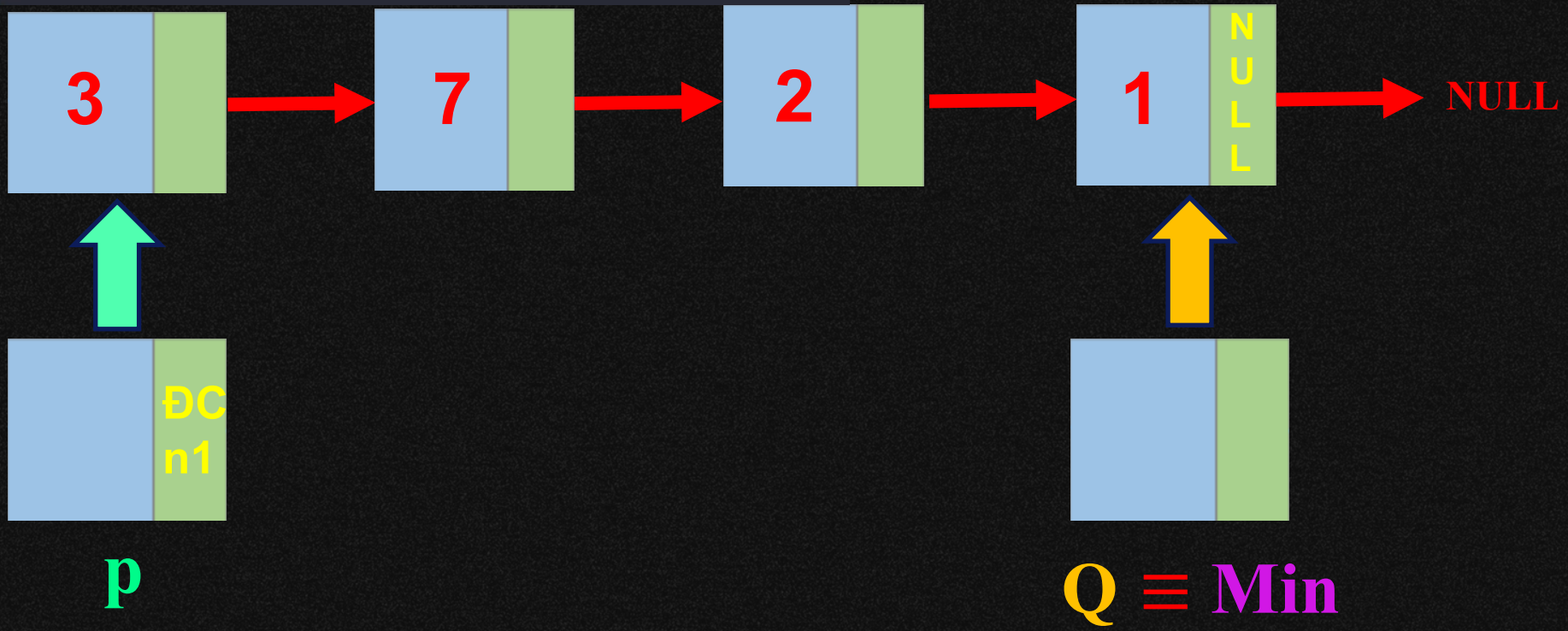
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



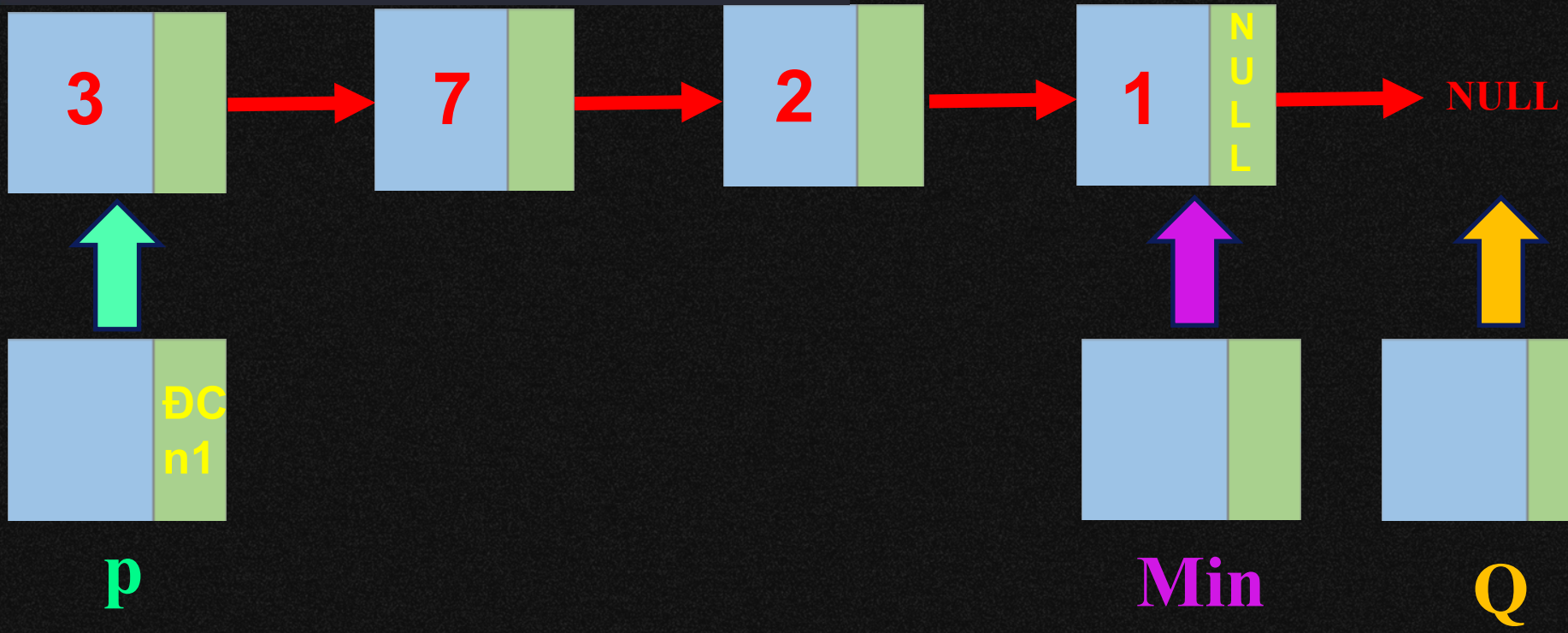
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



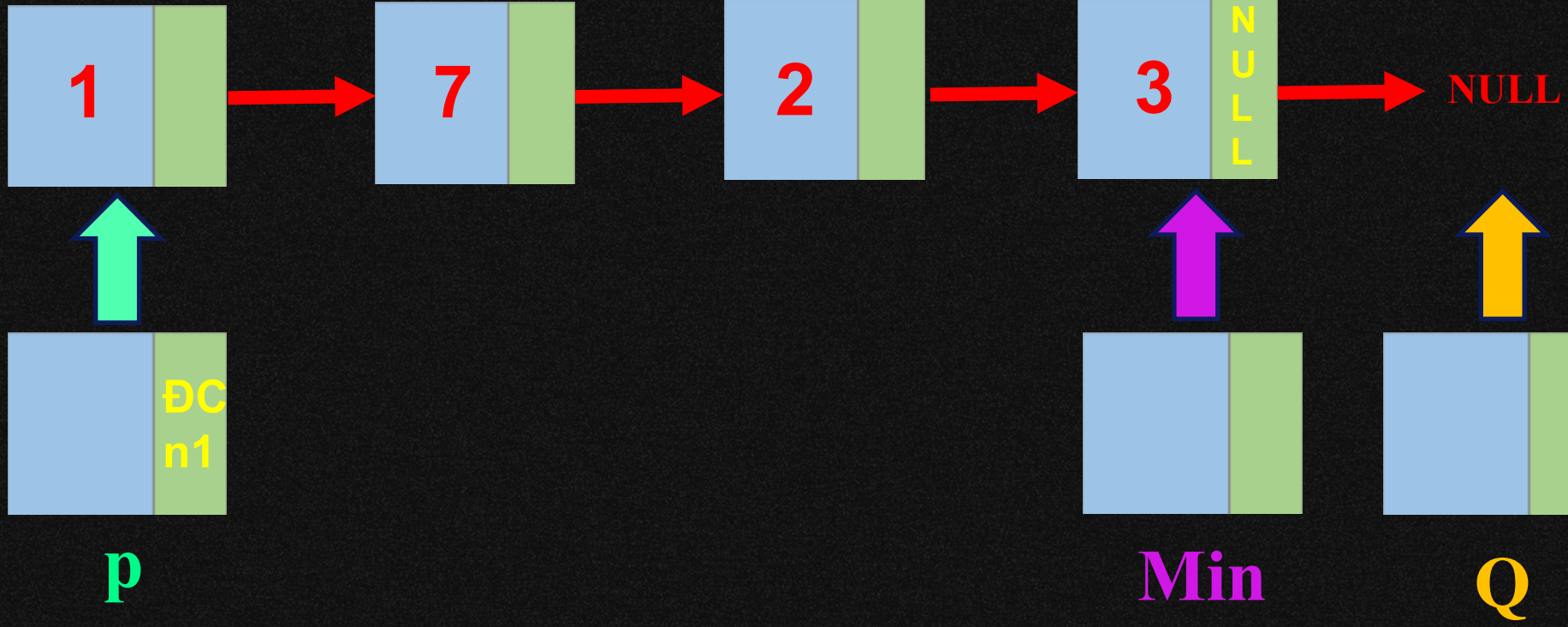
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



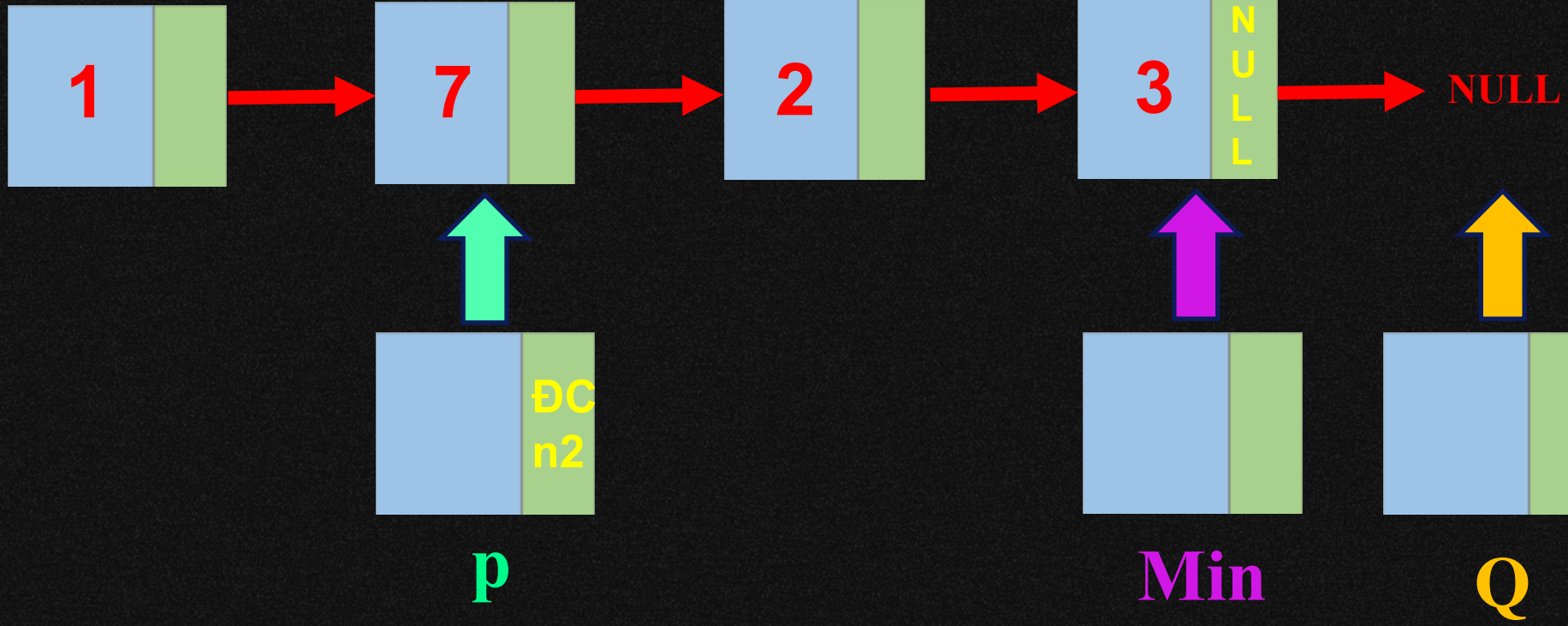
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



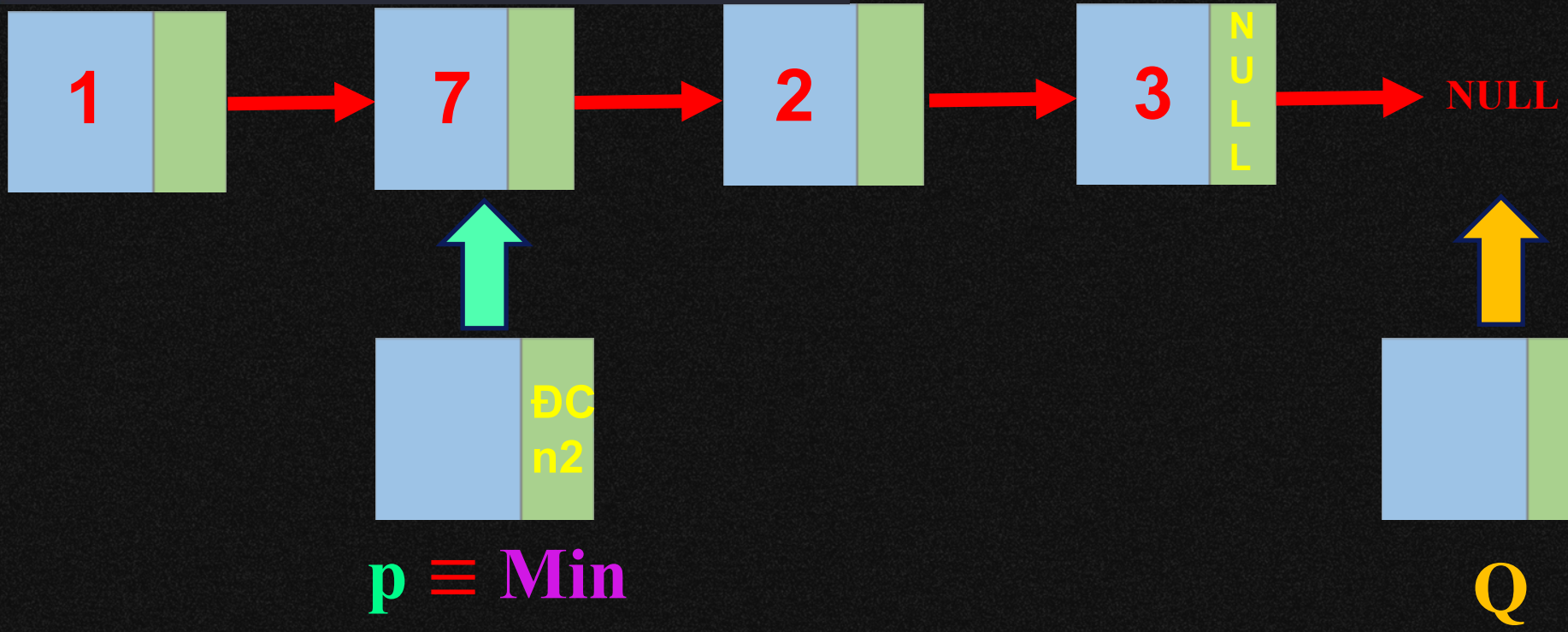
```
swap(MIN->Info, P->Info);
P = P->pNext;
```



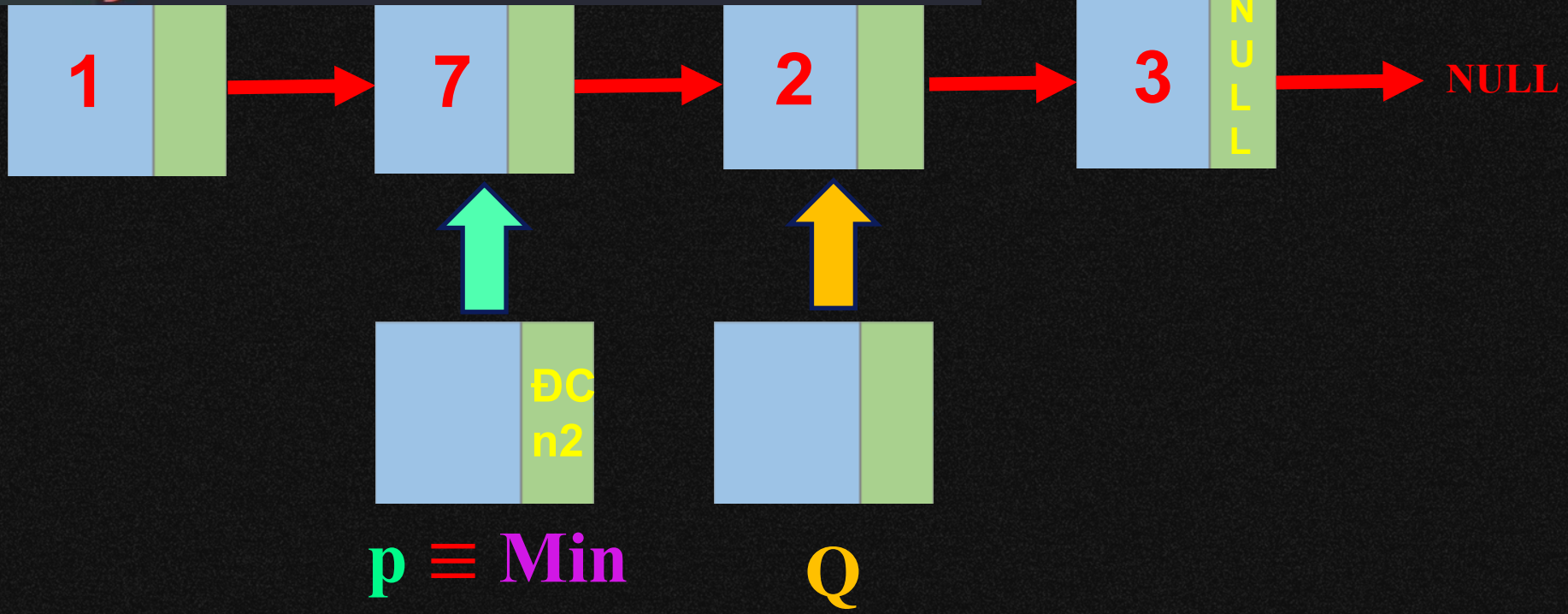
```
swap(MIN->Info, P->Info);
P = P->pNext;
```



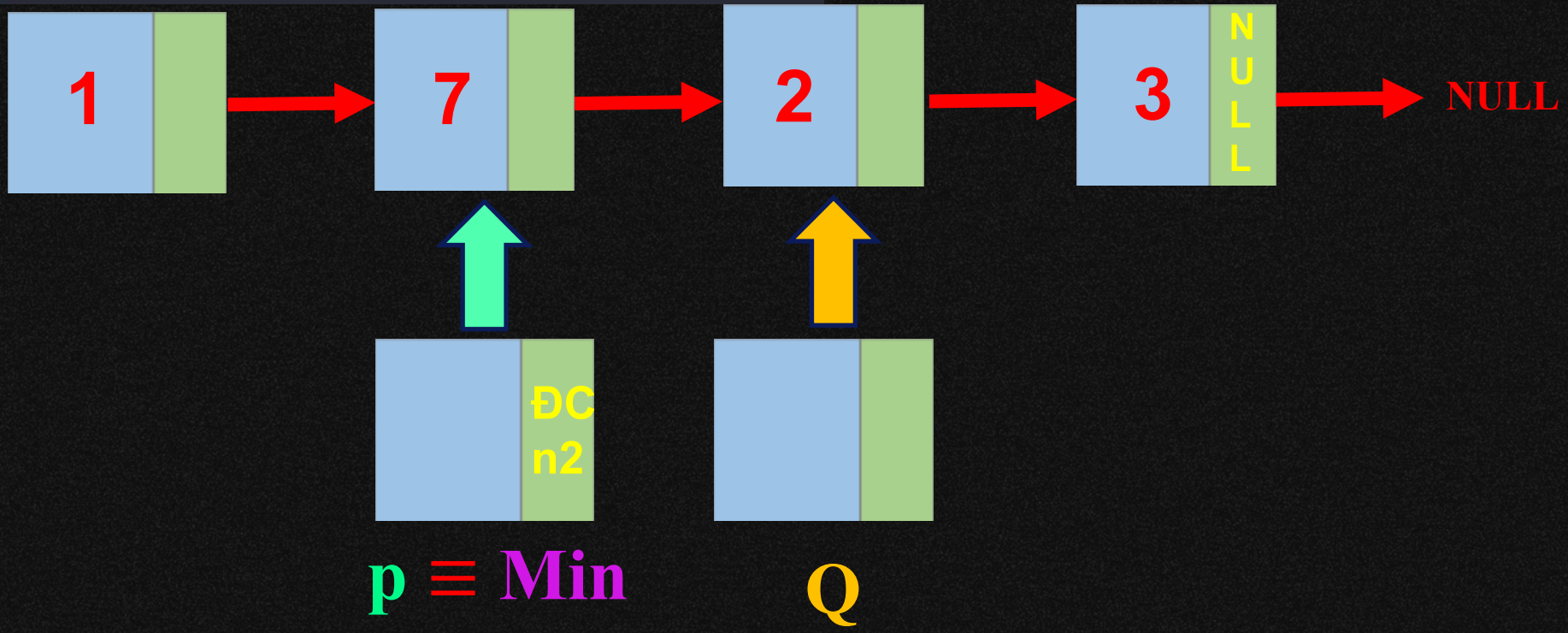
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



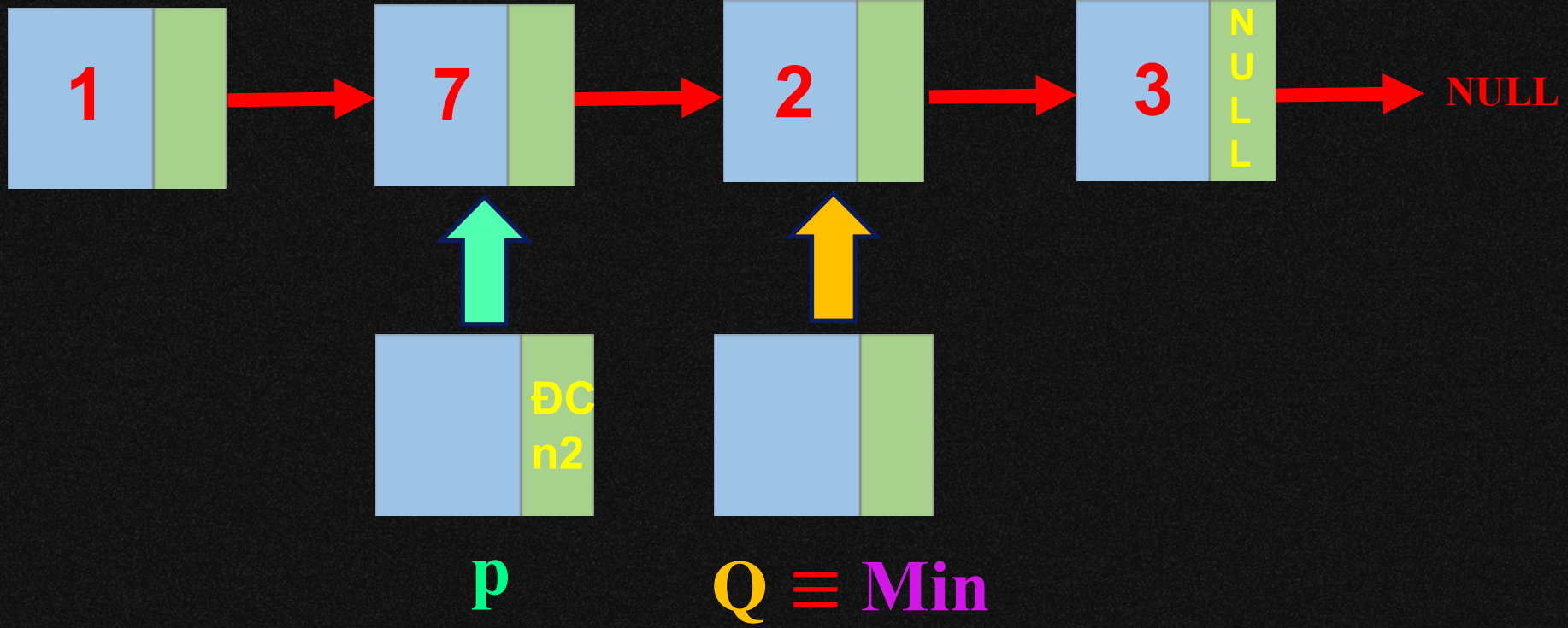
```
while (P != L.pTail)
{
    MIN = P;
    Q = P->pNext;
    while (Q != NULL)
    {
        if (Q->Info < MIN->Info)
            MIN = Q;
        Q = Q->pNext;
    }
```



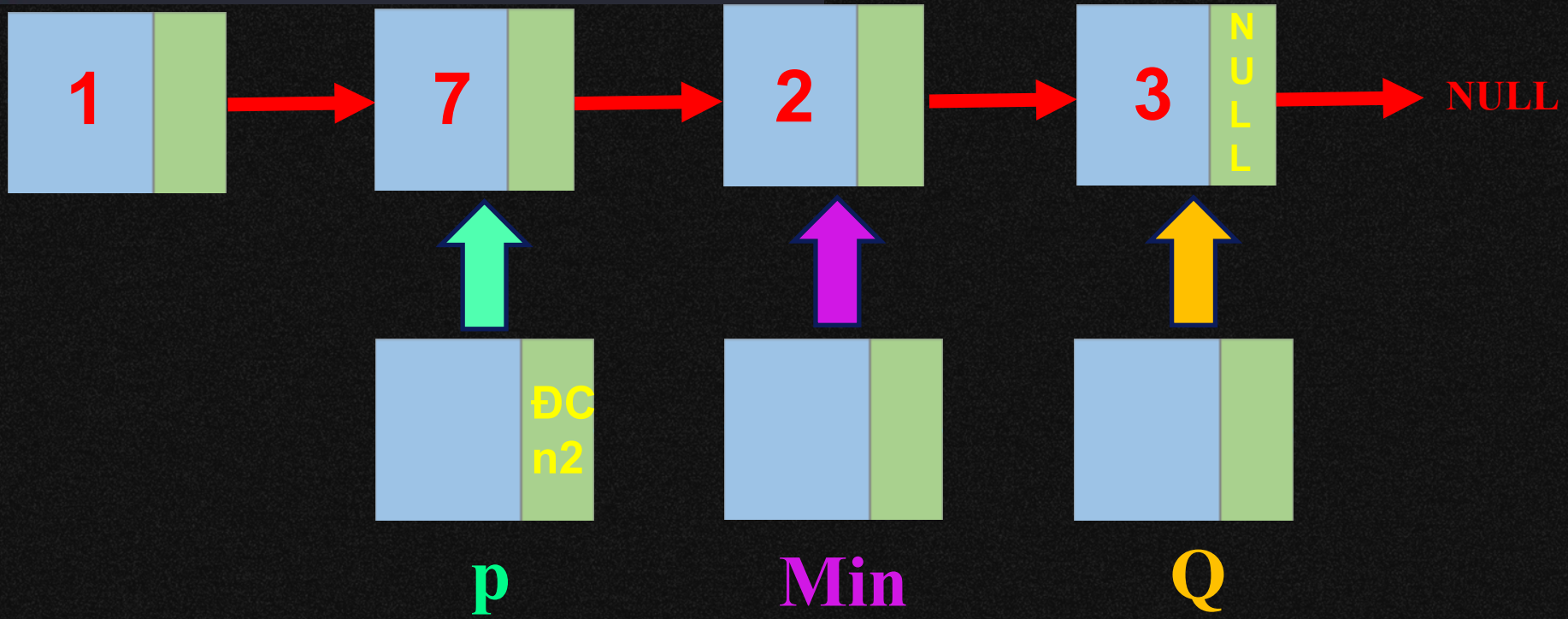
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



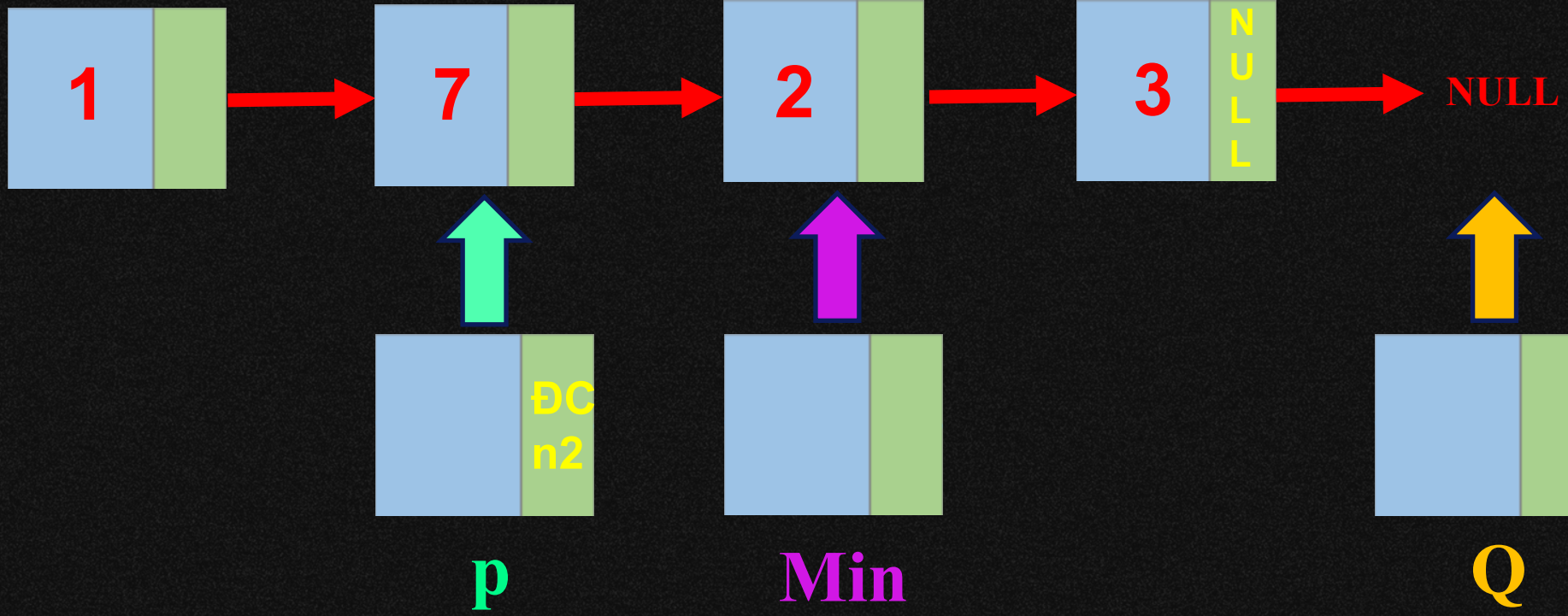
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



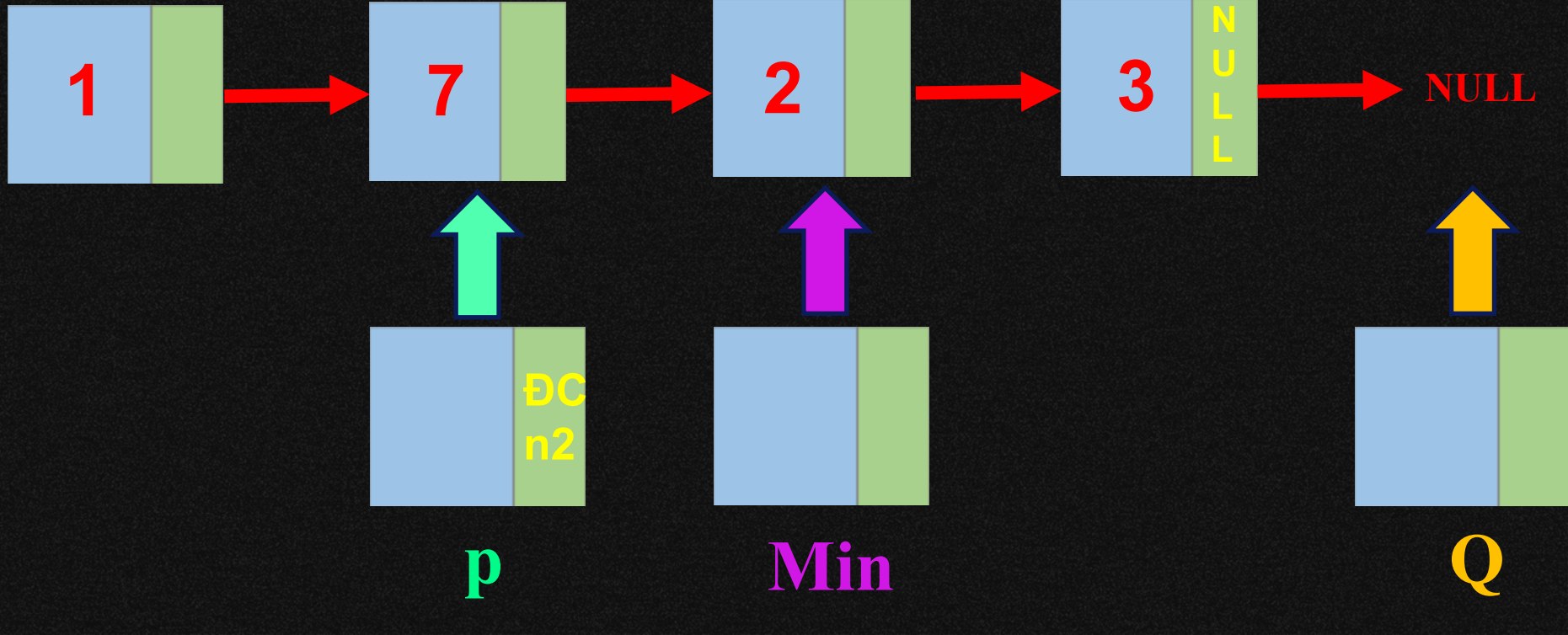
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



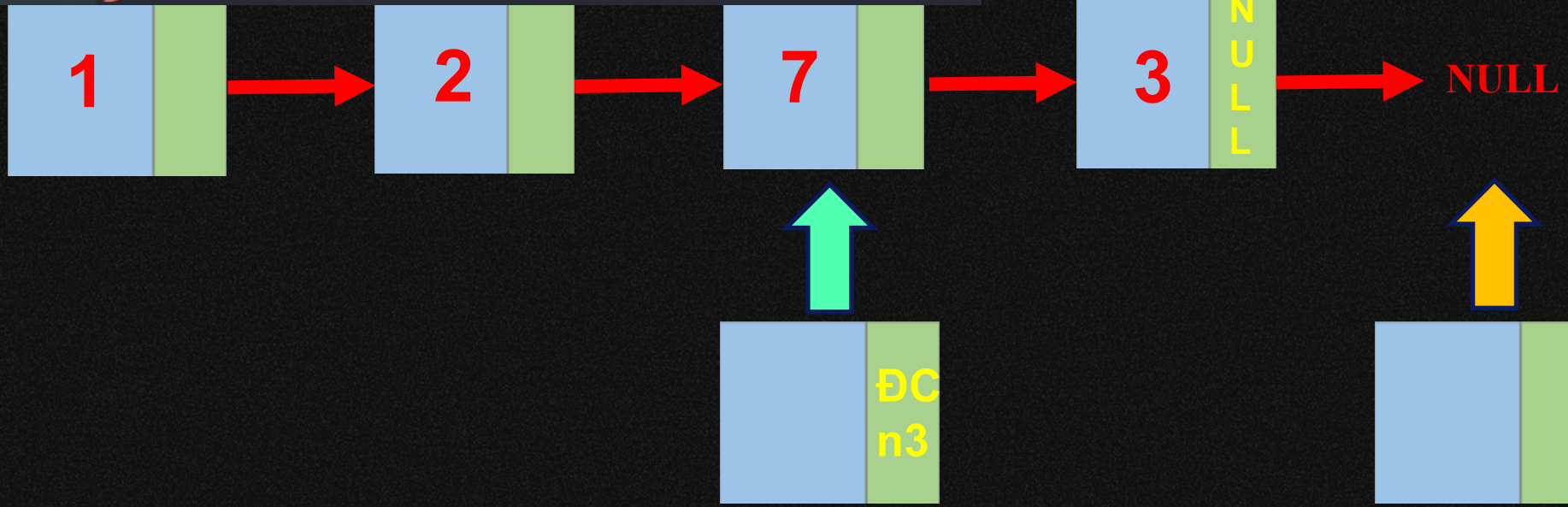
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



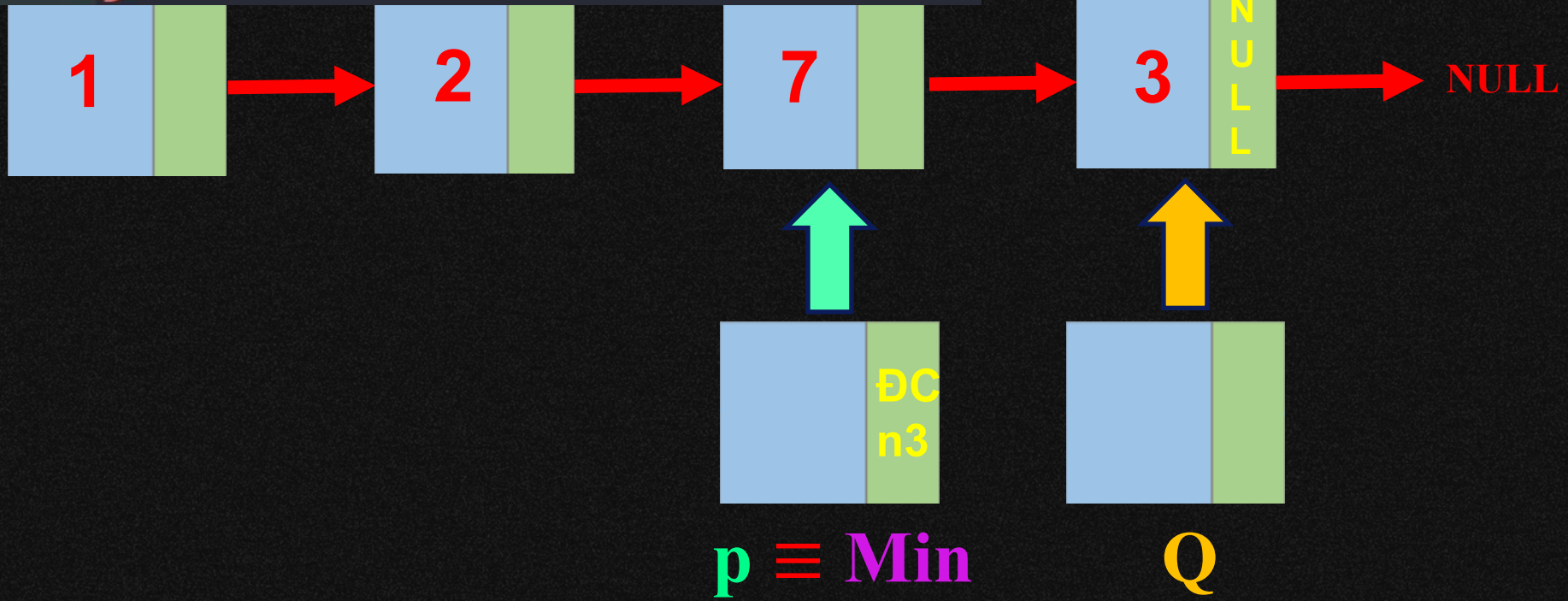
```
swap(MIN->Info, P->Info);
P = P->pNext;
```



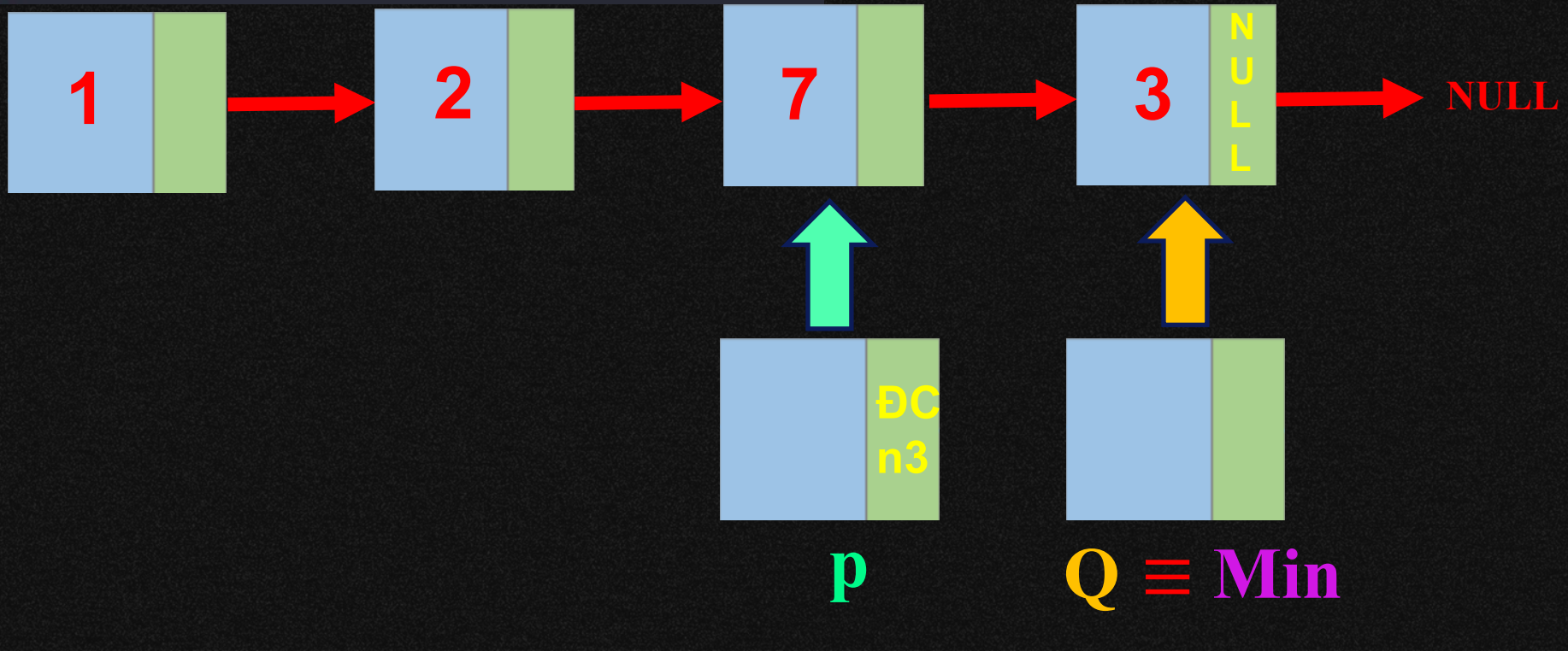
```
while (P != L.pTail)
{
    MIN = P;
    Q = P->pNext;
    while (Q != NULL)
    {
        if (Q->Info < MIN->Info)
            MIN = Q;
        Q = Q->pNext;
    }
```



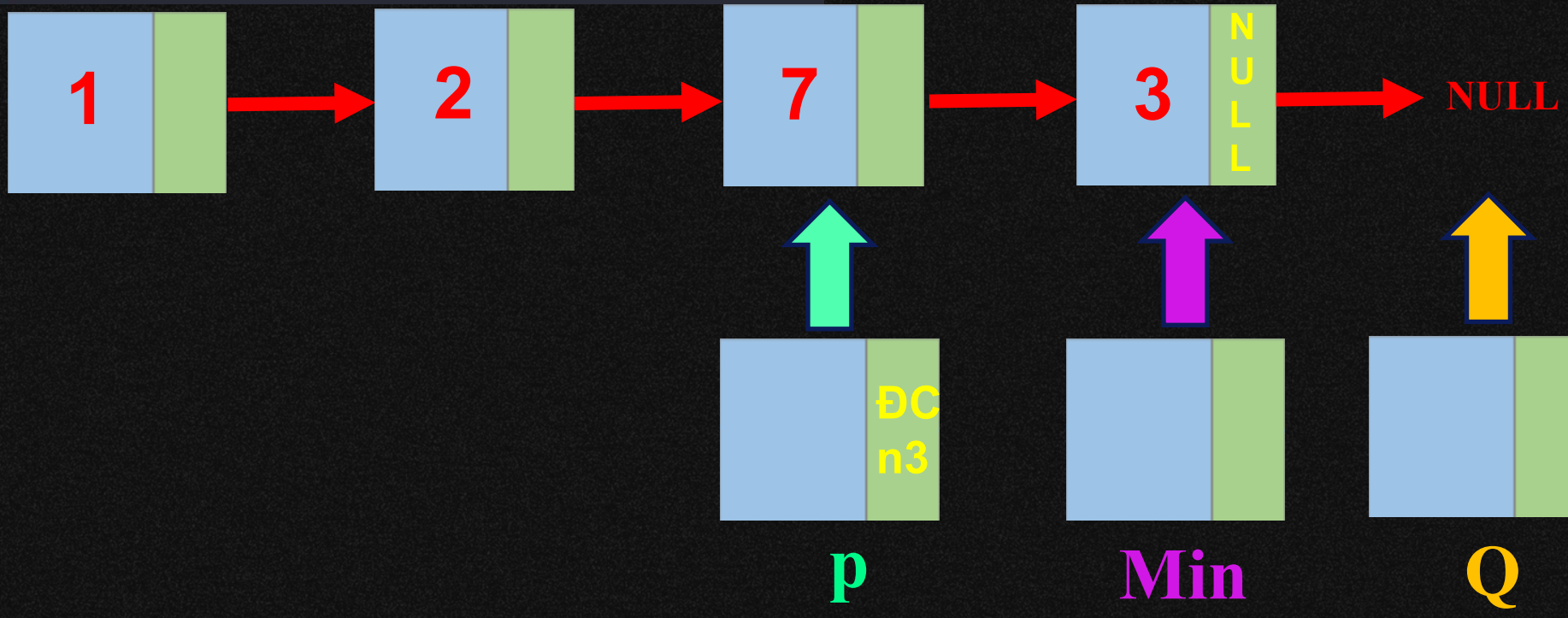
```
while (P != L.pTail)
{
    MIN = P;
    Q = P->pNext;
    while (Q != NULL)
    {
        if (Q->Info < MIN->Info)
            MIN = Q;
        Q = Q->pNext;
    }
```



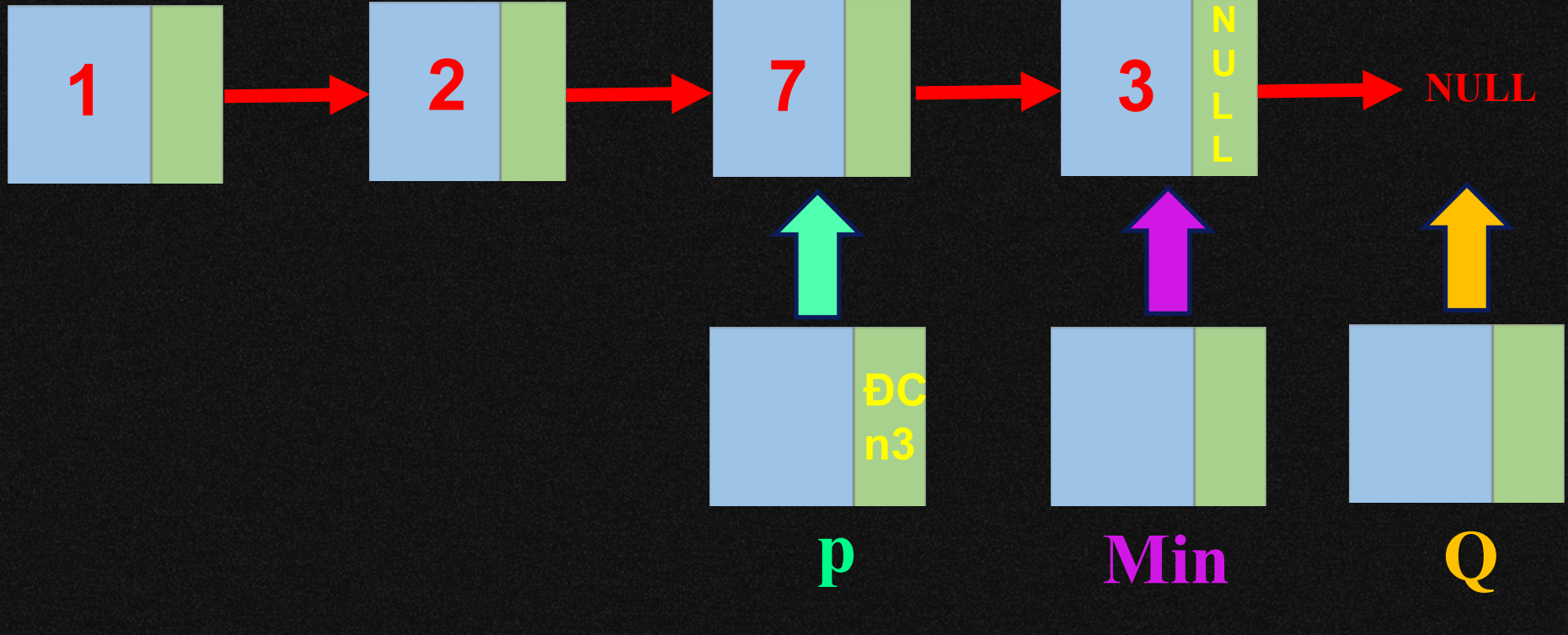
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info)
        MIN = Q;
    Q = Q->pNext;
}
```



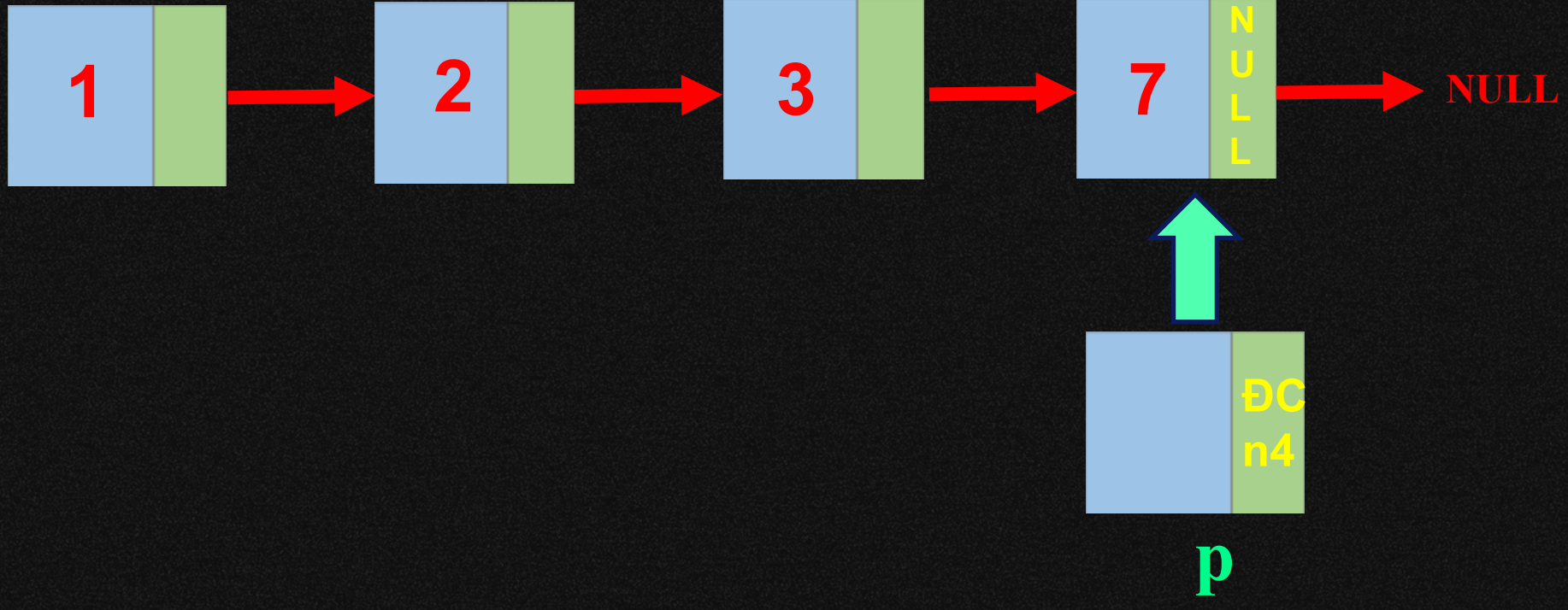
```
while (Q != NULL)
{
    if (Q->Info < MIN->Info
        MIN = Q;
    Q = Q->pNext;
}
```



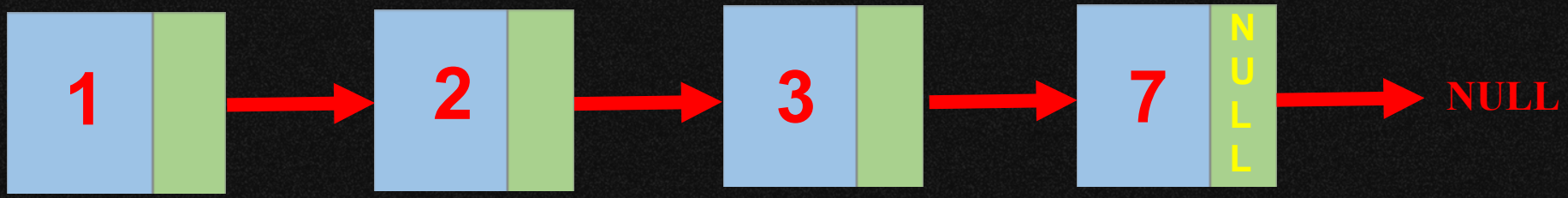
```
swap(MIN->Info, P->Info);
P = P->pNext;
```



```
swap(MIN->Info, P->Info);
P = P->pNext;
```



```
swap(MIN->Info, P->Info);
P = P->pNext;
```


1
2
3
7
NULL
NULL

MỘT SỐ BÀI TẬP ÁP DỤNG CTDL - DSLK ĐƠN

- *- Viết chương trình thực hiện sắp xếp 1 danh sách liên kết bao gồm các phần tử là số nguyên*

- *- Viết chương trình tạo 1 danh sách liên kết đơn để sắp xếp các phân số với nhau*
- *Di chuyển phần tử đầu tiên xuống cuối cùng*