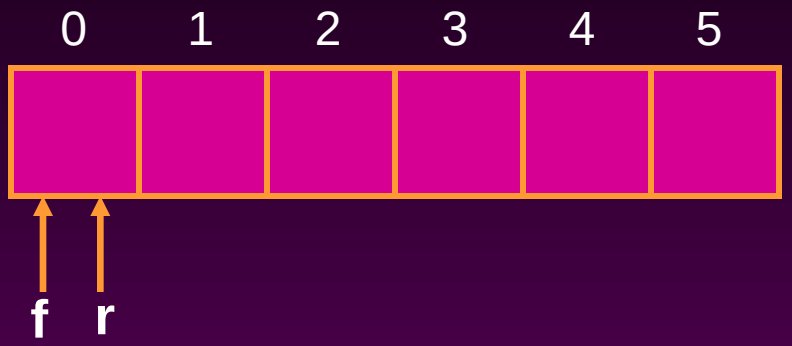
Cmps 1053

Fall 2009

Queue

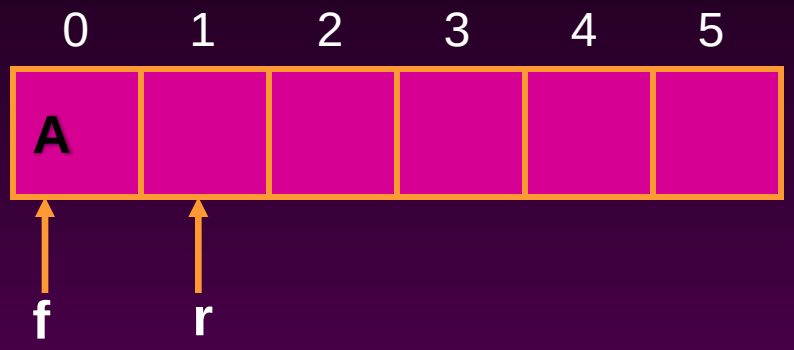
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



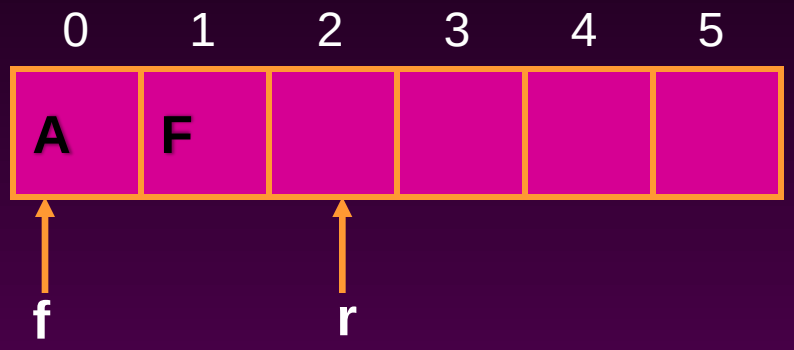
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



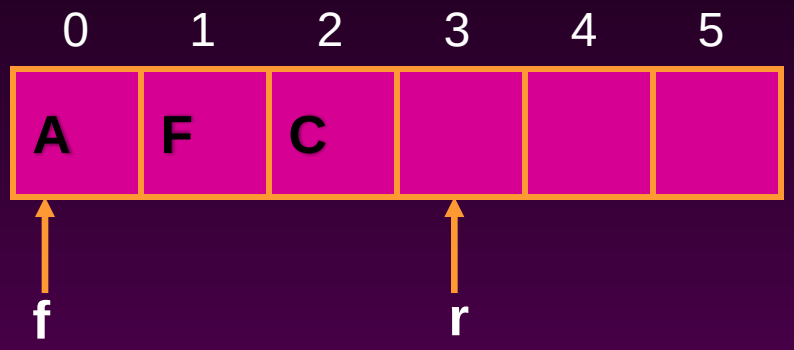
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



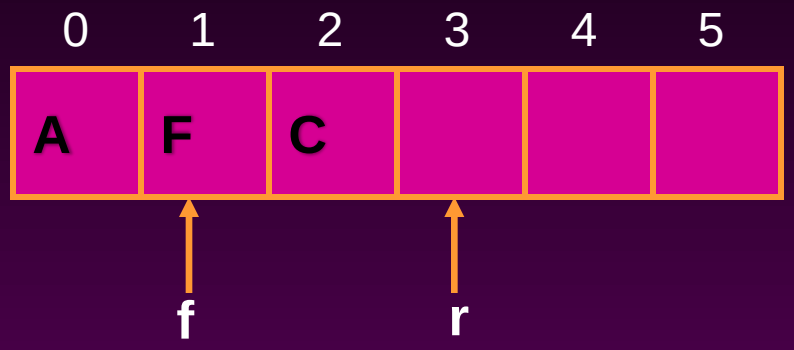
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



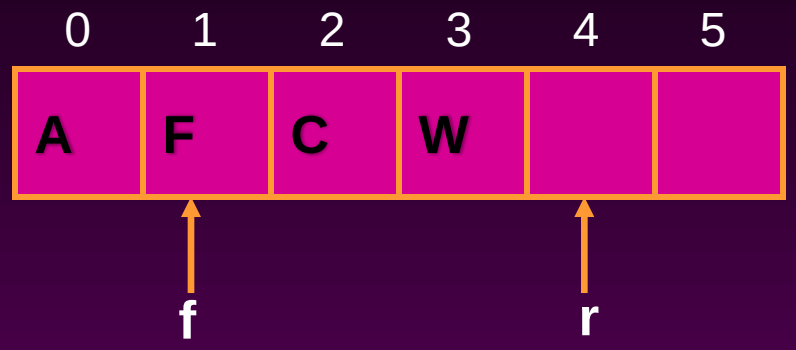
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



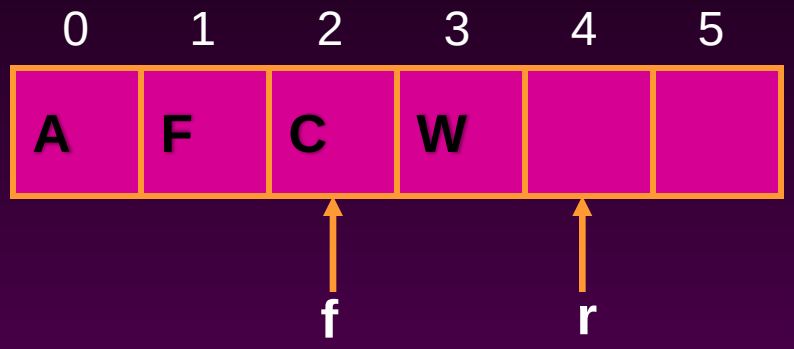
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



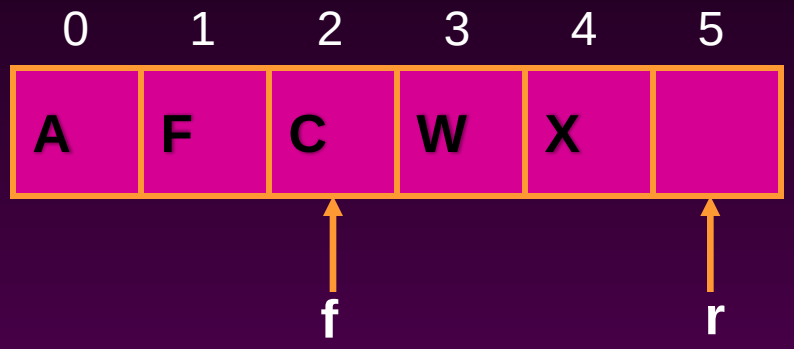
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



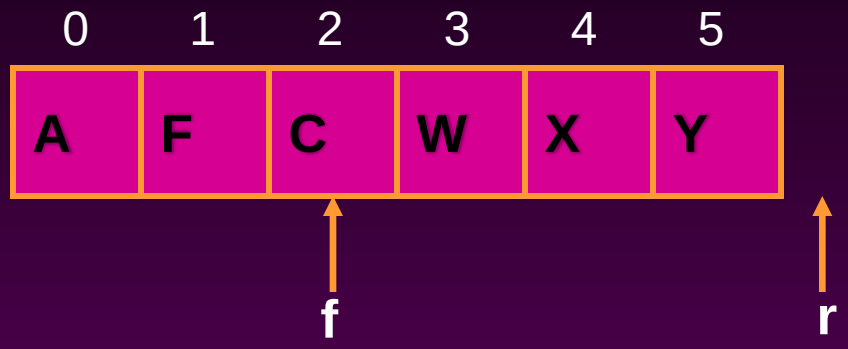
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



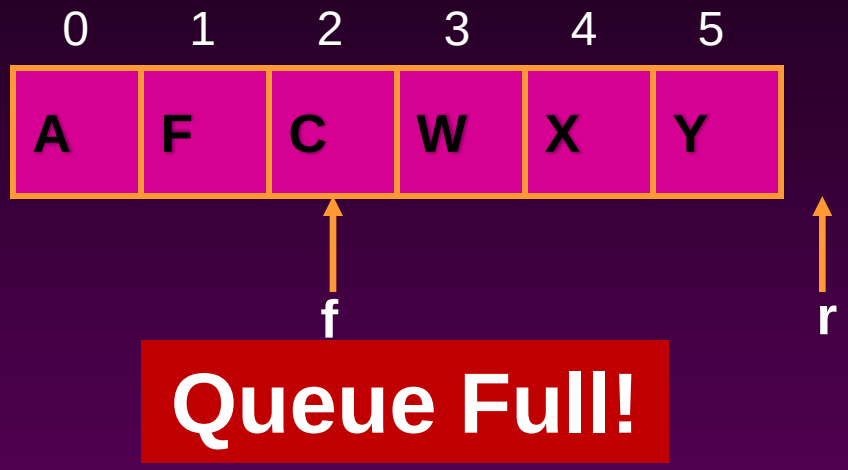
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



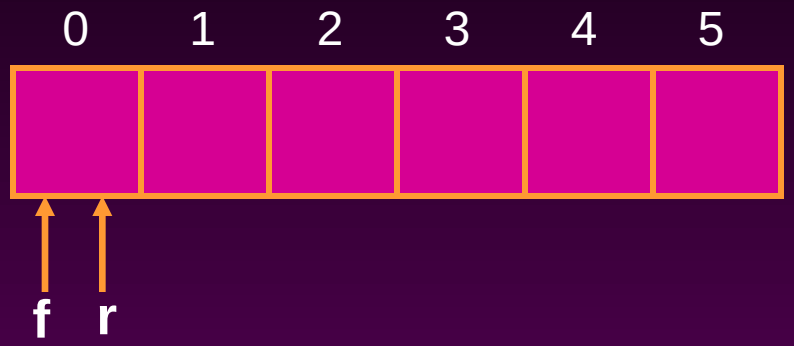
Simple Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
```



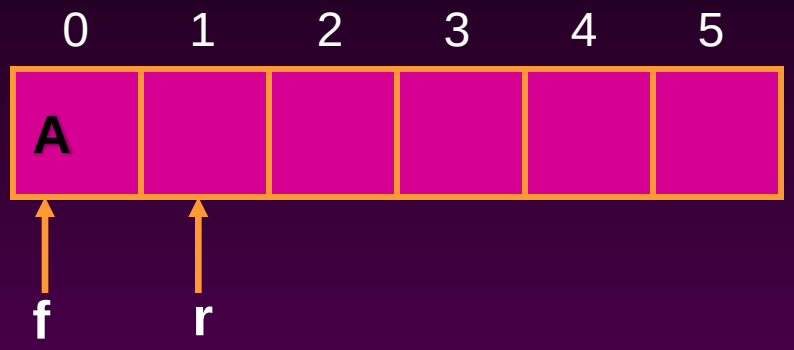
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



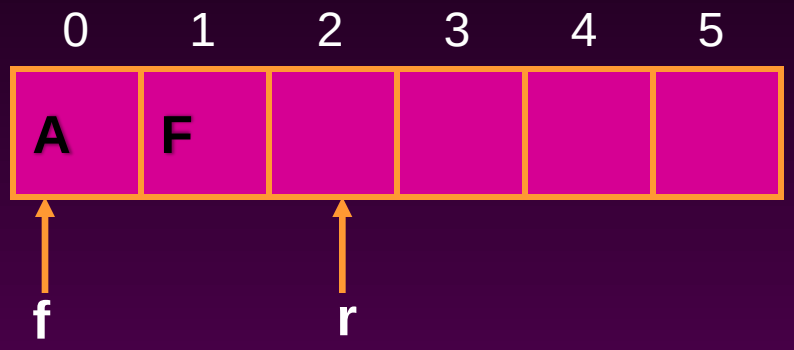
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



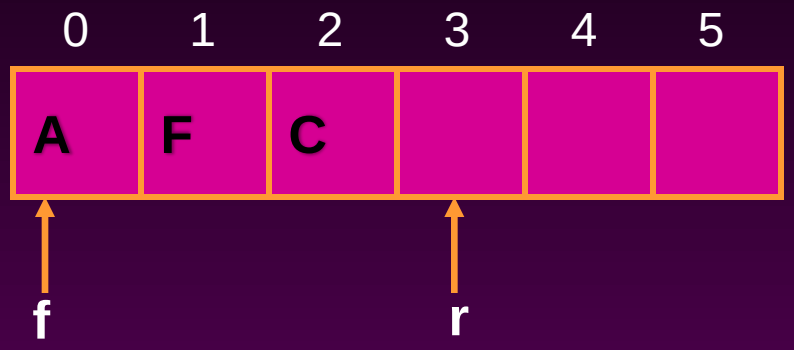
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



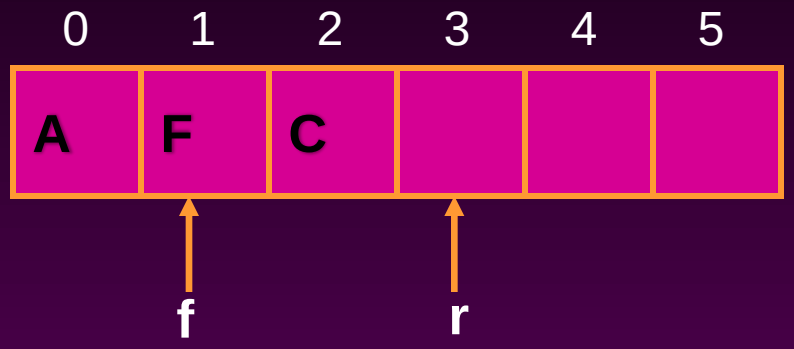
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



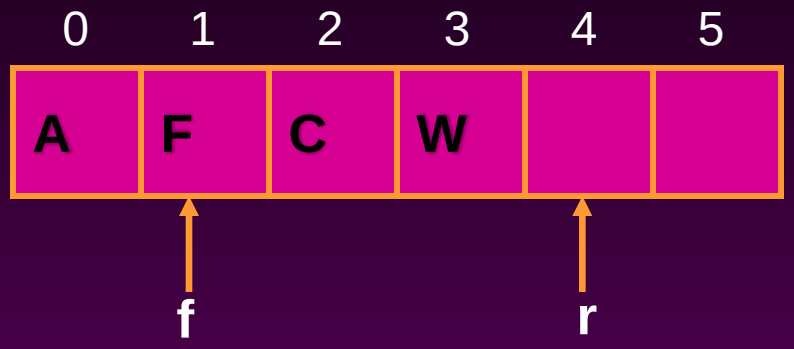
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



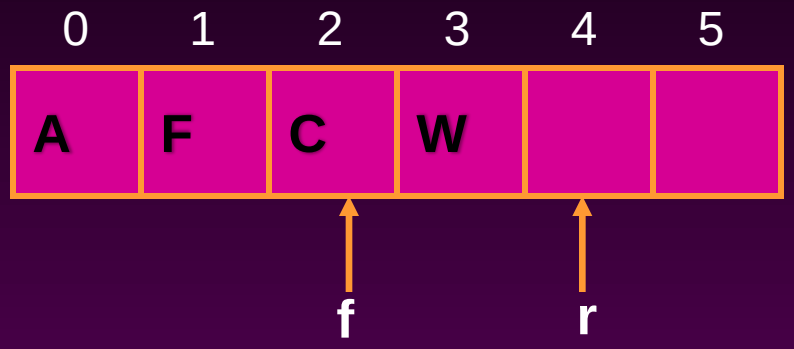
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



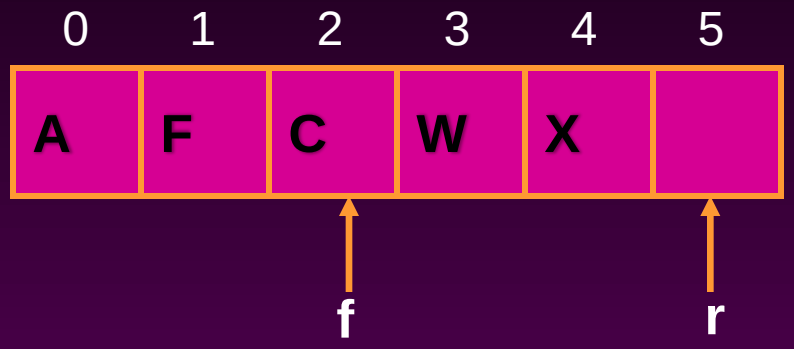
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



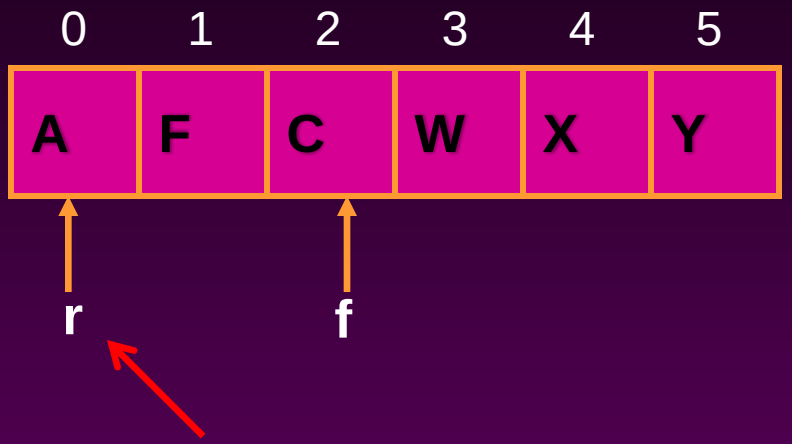
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



Circular Array Based Queue

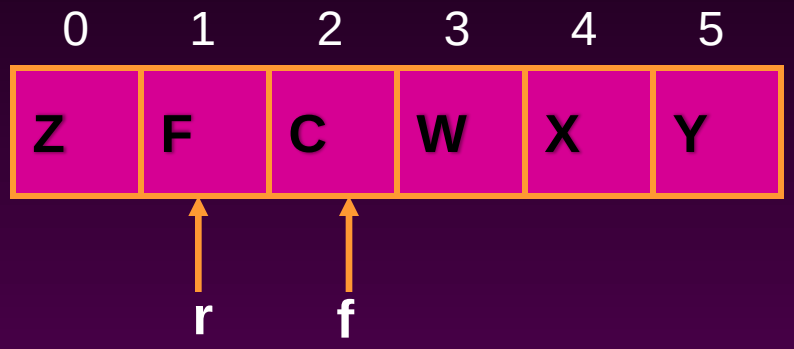
```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



Rear pointer wraps around!

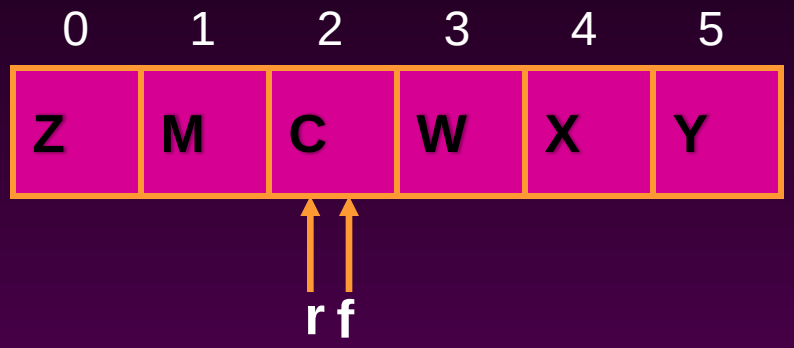
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



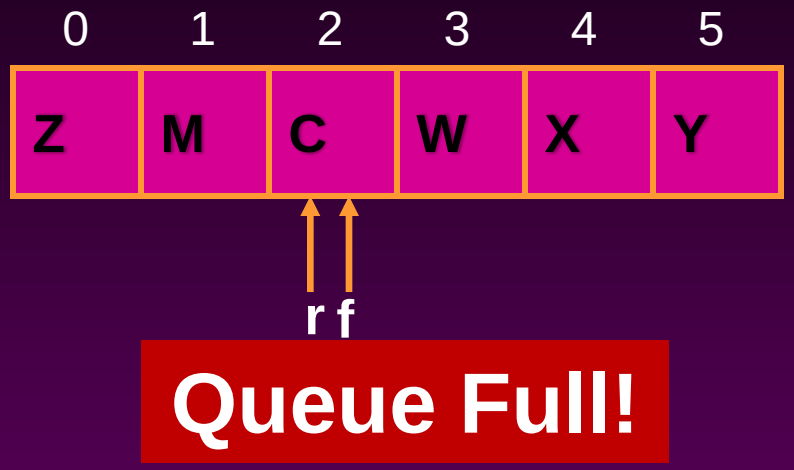
Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



Circular Array Based Queue

```
enqueue('A');
enqueue('F');
enqueue('C');
dequeue();
enqueue('W');
dequeue();
enqueue('X');
enqueue('Y');
enqueue('Z');
enqueue('M');
enqueue('N');
```



List Based Structure (not a queue … yet)

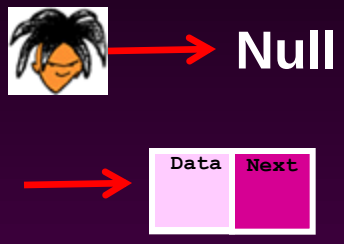
```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
```

List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
```

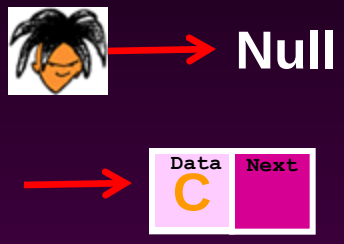
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
```



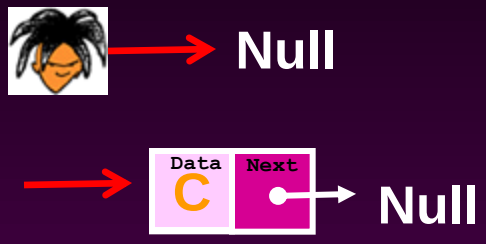
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
```



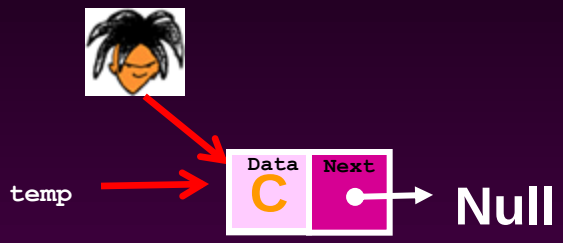
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
```



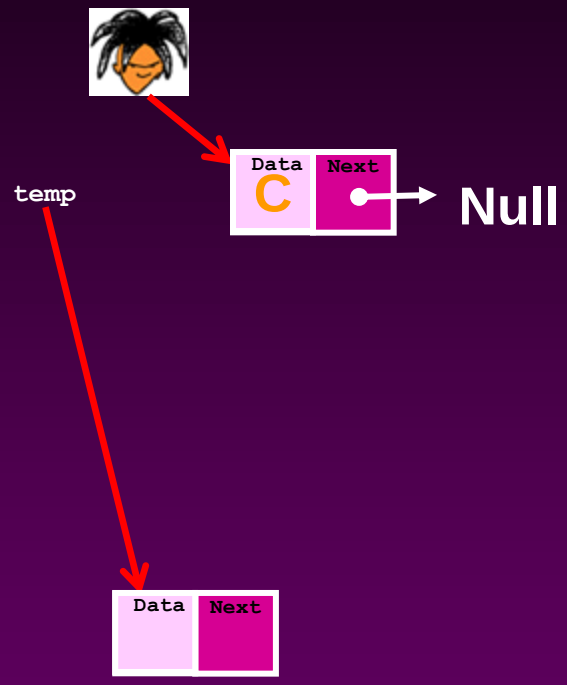
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
```



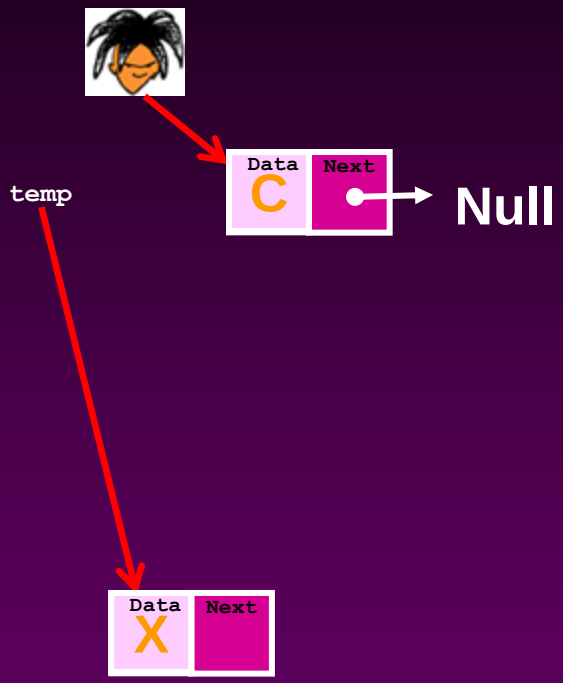
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
```



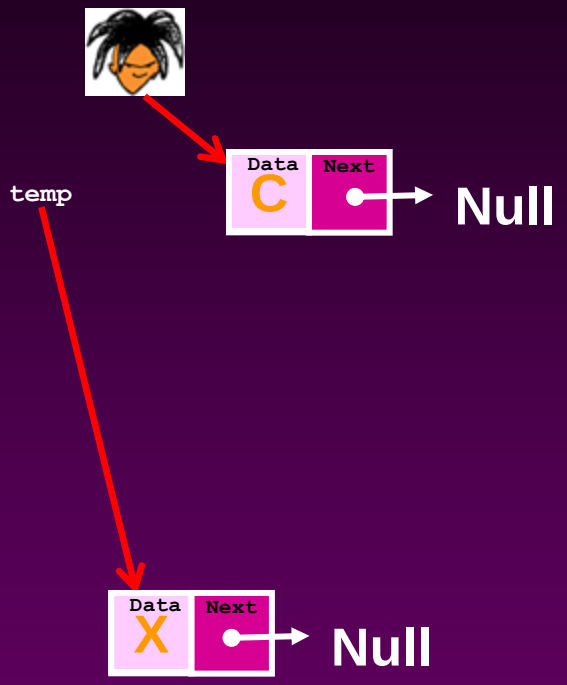
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
```



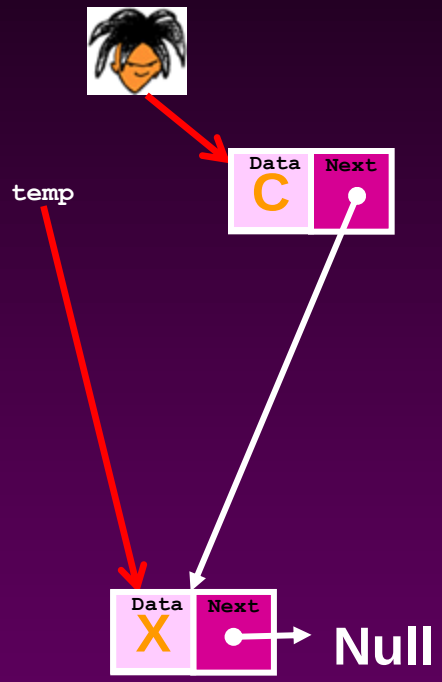
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
```



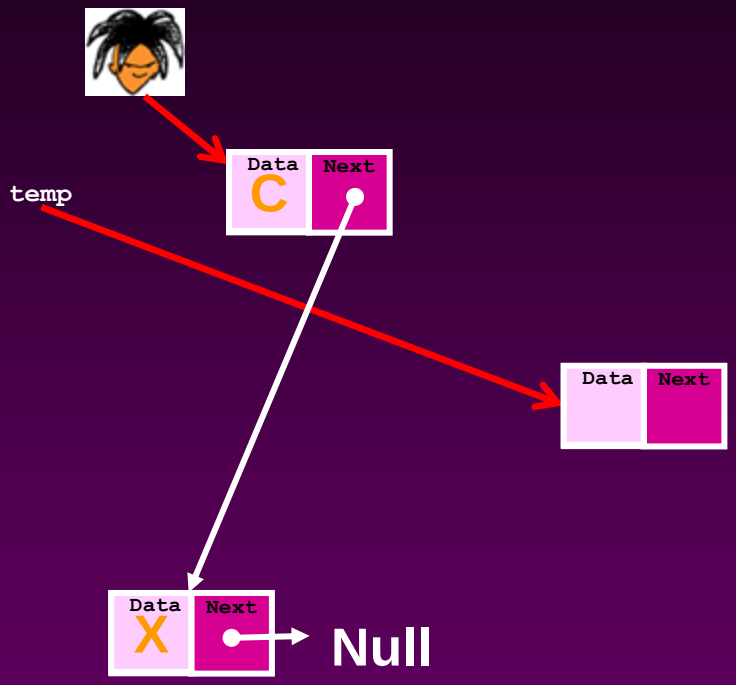
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
```



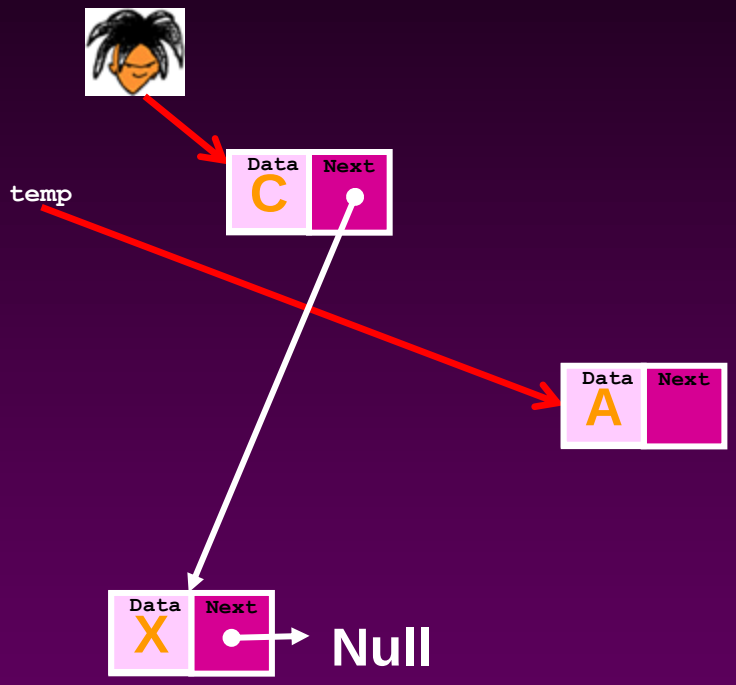
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
```



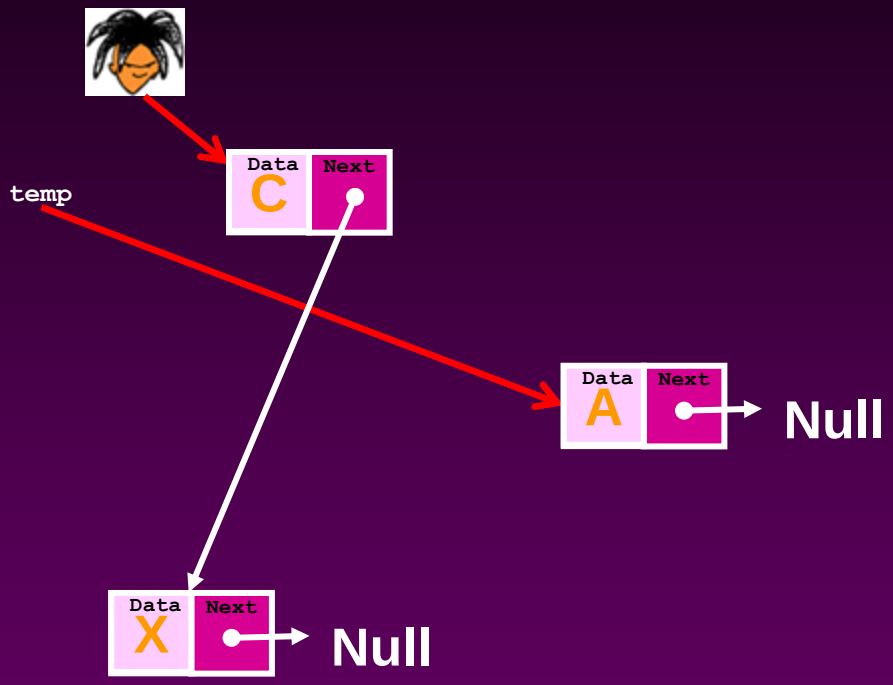
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
```



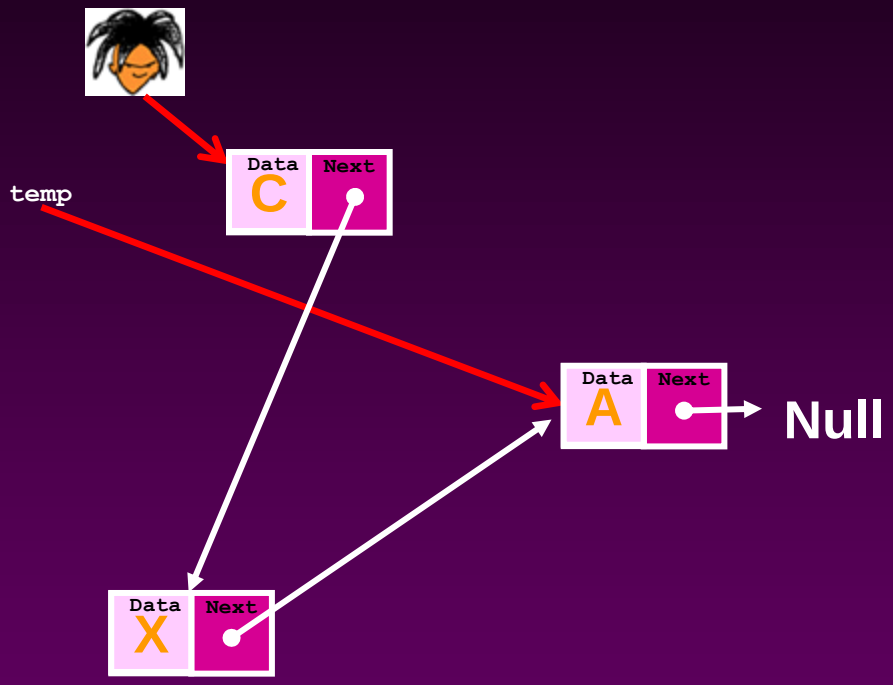
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
```



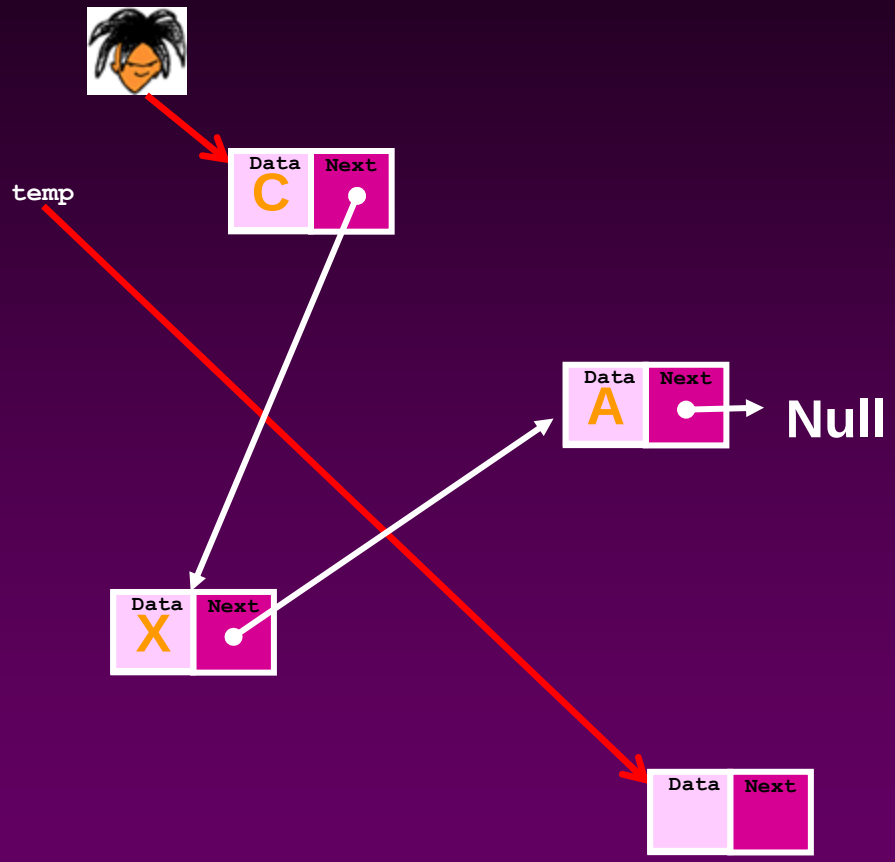
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
```



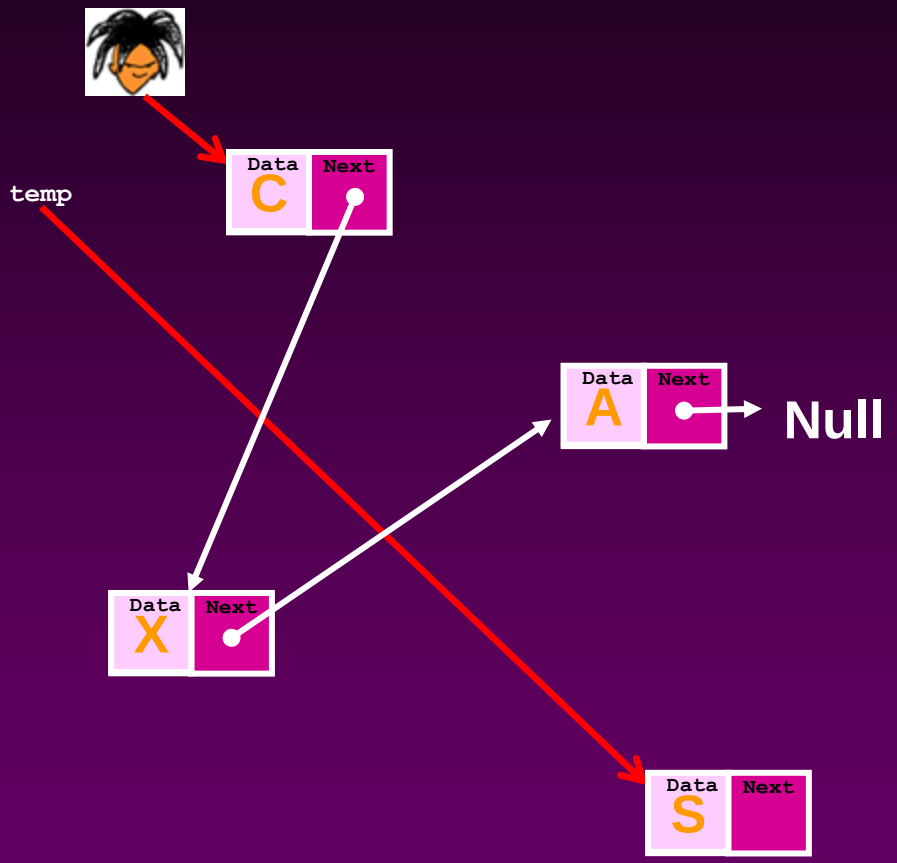
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
temp= new Node;
```



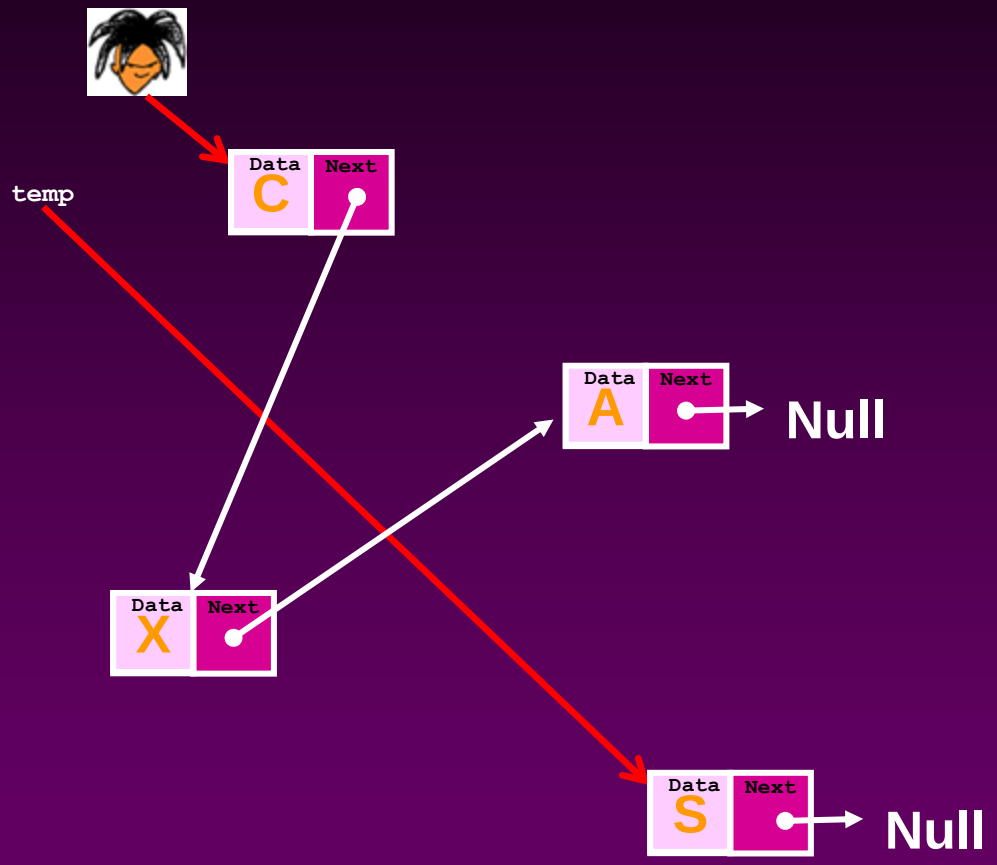
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
temp= new Node;
temp->Data = 'S';
```



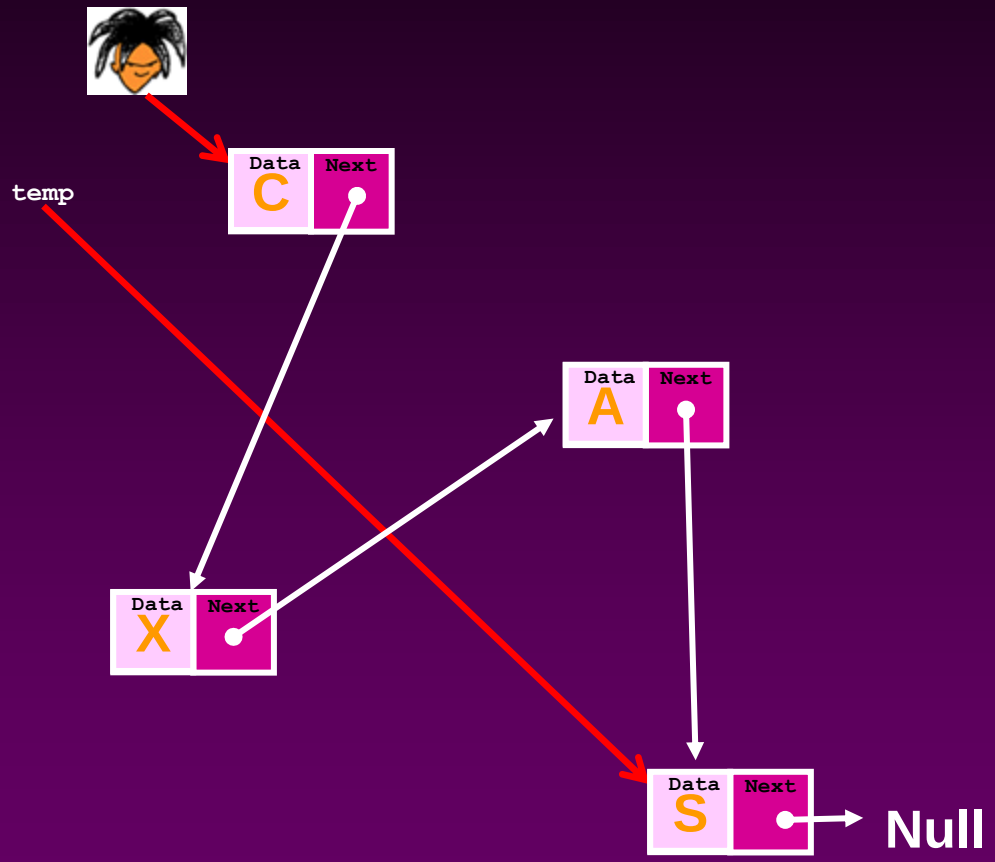
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
temp= new Node;
temp->Data = 'S';
temp=>next = NULL;
```



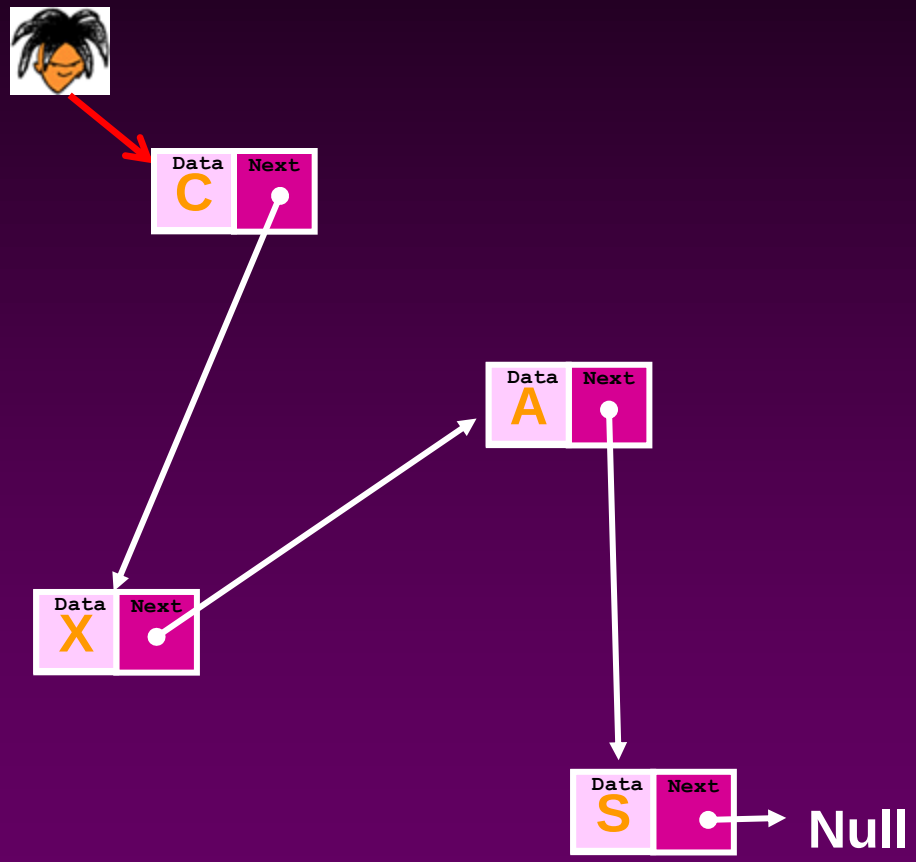
List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
temp= new Node;
temp->Data = 'S';
temp->next = NULL;
Head->next->next->next = temp;
```



List Based Structure (not a queue … yet)

```
//Create the node Type
struct Node{
     char Data;
     Node * next;
}
//Create some pointers
Node * Head;
Head = NULL;
Node *temp = new Node;
temp->Data = 'C';
temp->next = NULL;
Head = temp;
temp = new Node;
temp->Data = 'X';
temp->next = NULL;
Head->Next = temp;
temp = new Node;
temp->Data = 'A';
temp->next = NULL;
Head->next->next = temp;
temp= new Node;
temp->Data = 'S';
temp->next = NULL;
Head->next->next->next = temp;
temp = NULL;
```



List Based Queue

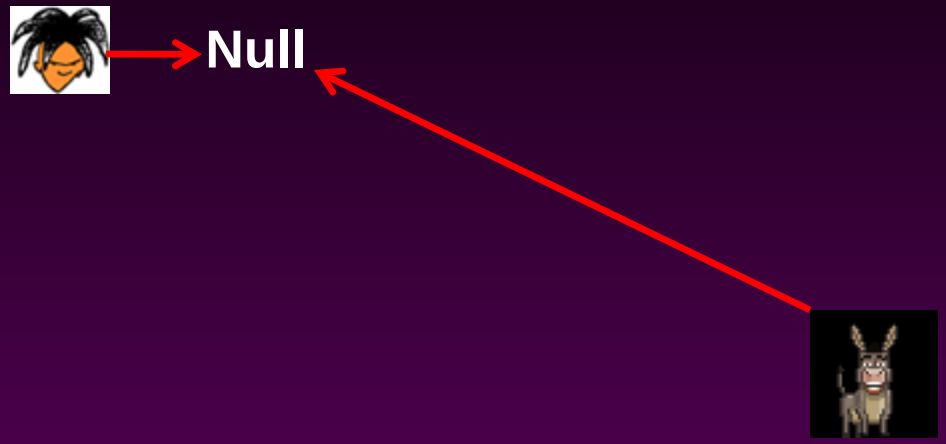
```
struct Node {
        char Data;
        Node* next;
};
```

```
class Queue {
    private:
        Node* head;
        Node* front;
        Node* rear;
        int numItems;
    public:
        void Enqueue(char val);
        void Dequeue(char &val);
        void PrintQueue(void);
        bool IsEmpty(void);
};
```

List Based Queue

Main Function:

```
Queue MyQ;
```



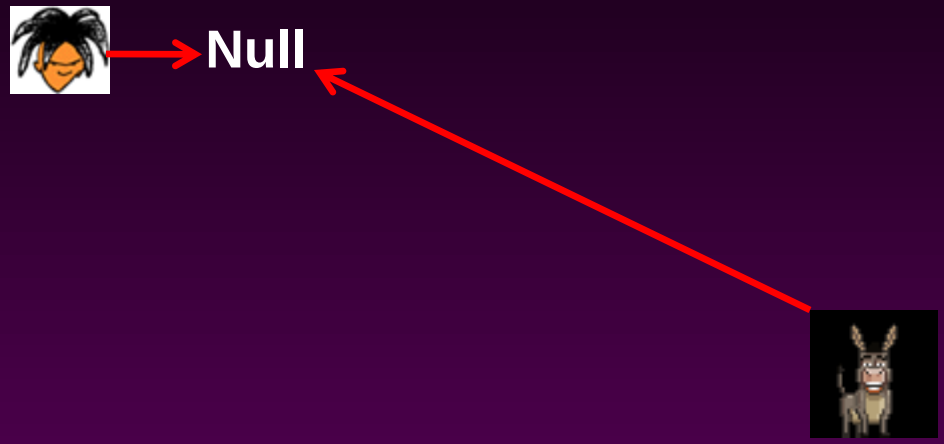
Code Executed:

```
void Queue(){
    front =  rear = NULL;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
```



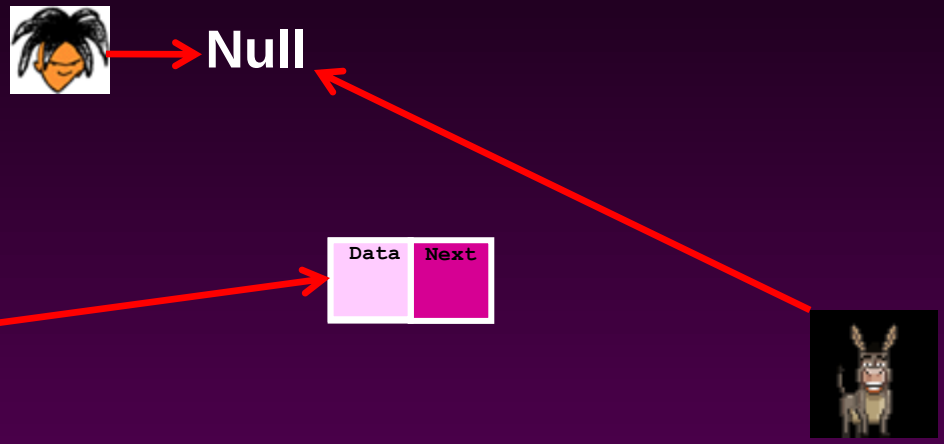
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
```



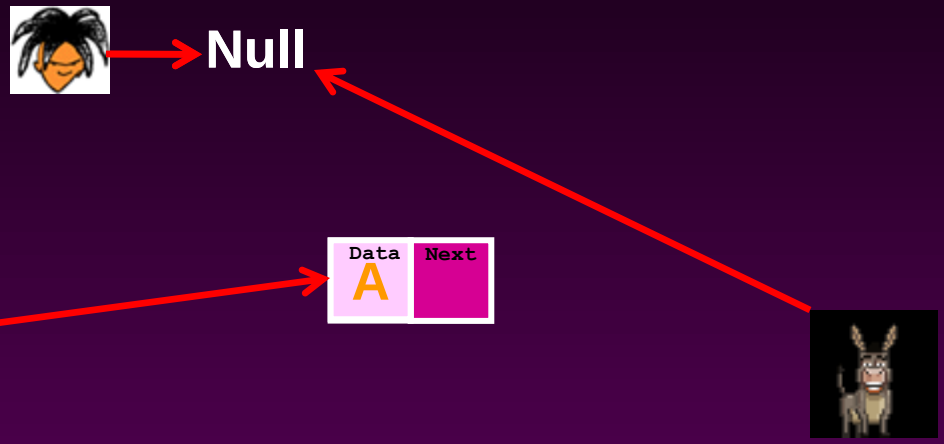
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
```



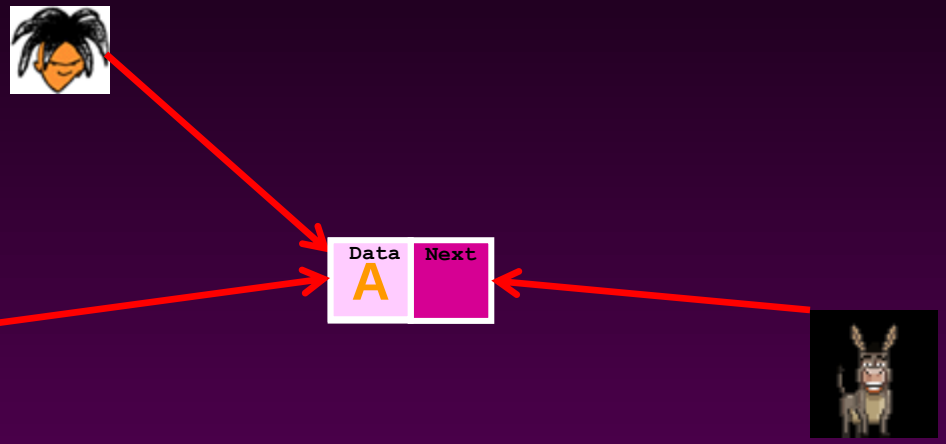
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
```



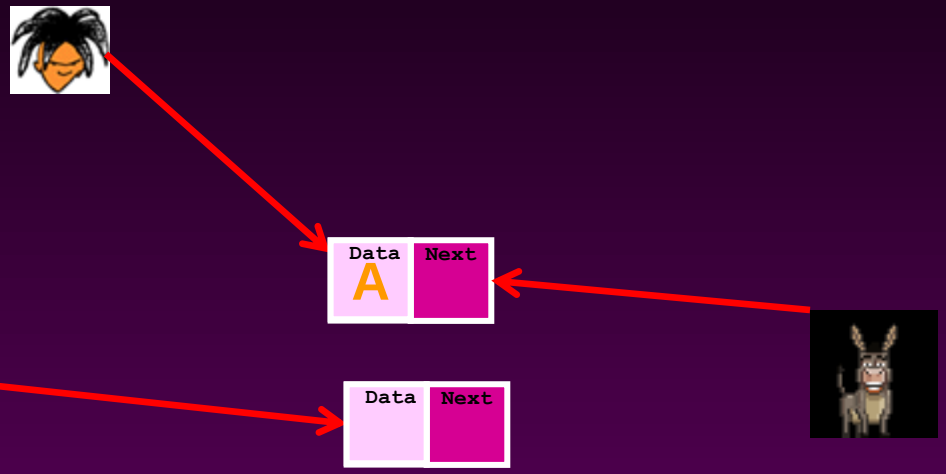
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
```



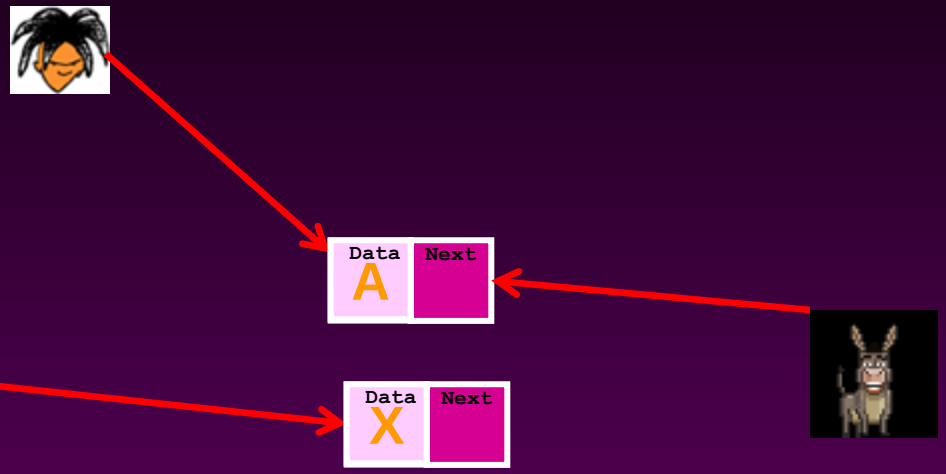
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
```



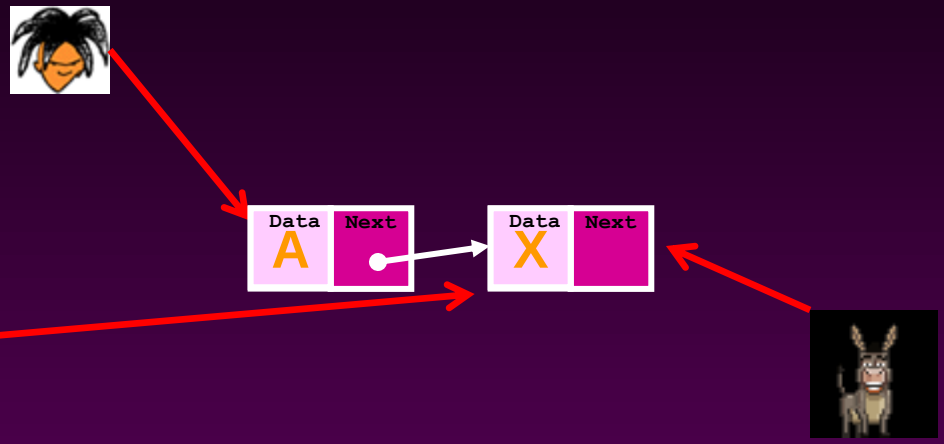
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
```



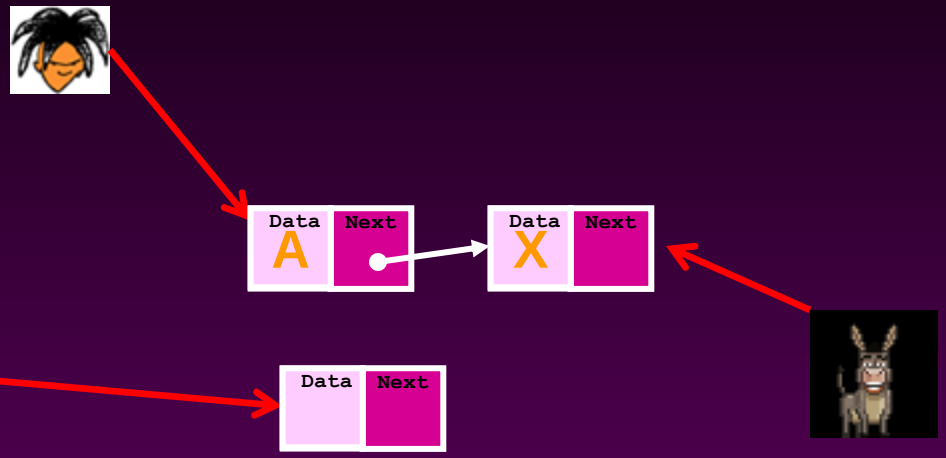
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
```



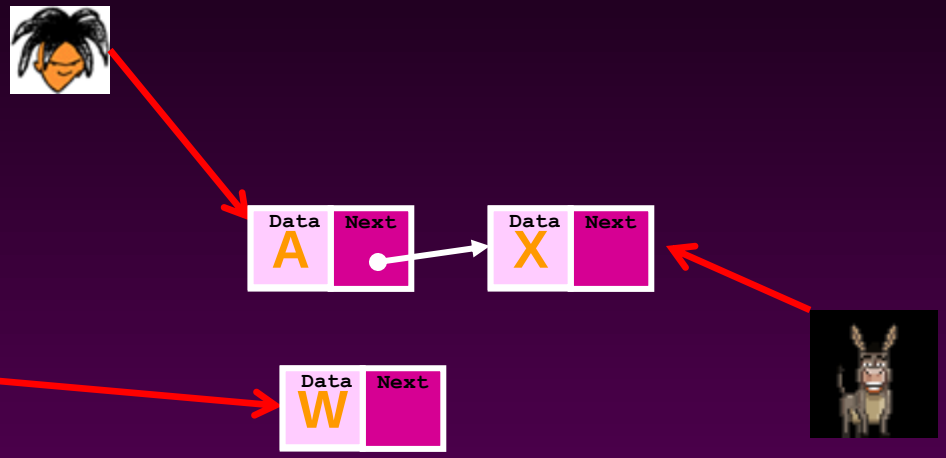
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
```



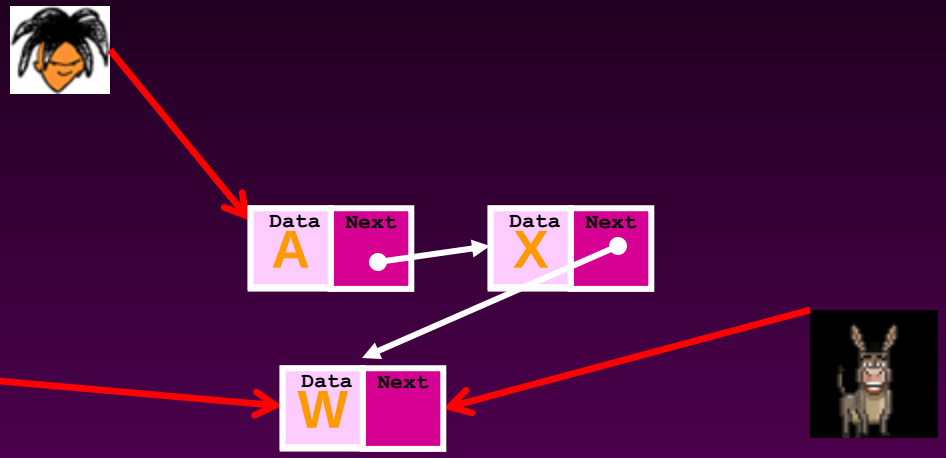
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
```



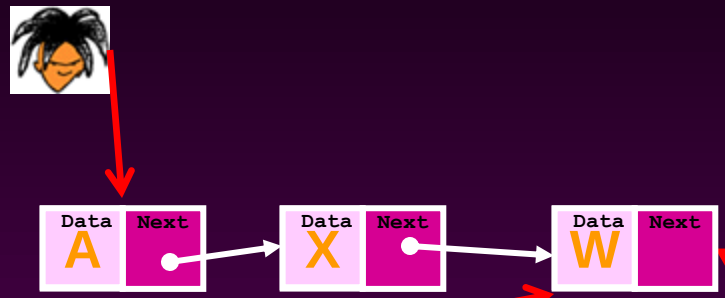
Code Executed:

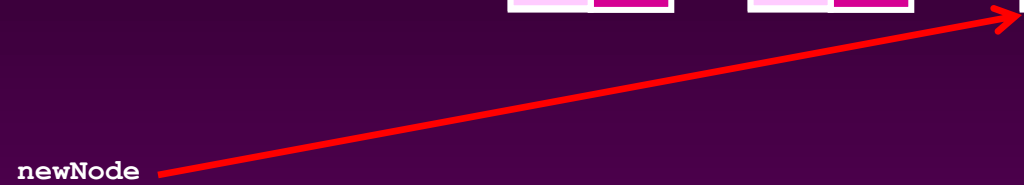
```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
```





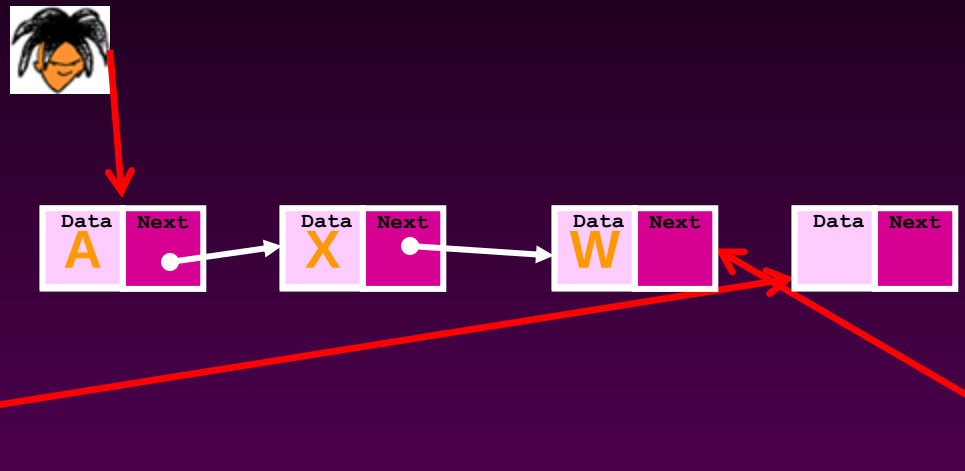
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
```



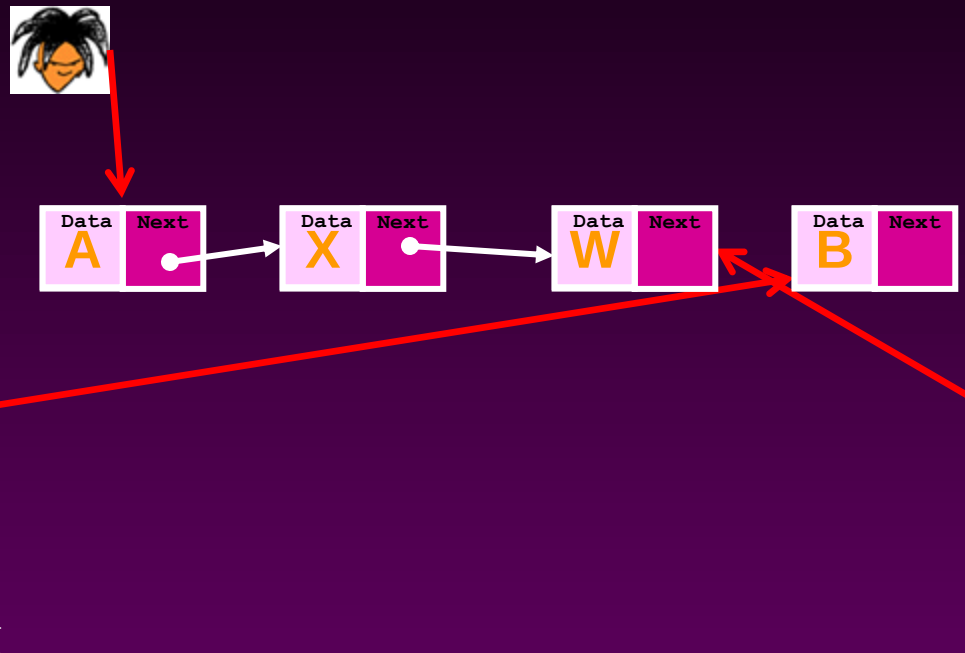
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
```



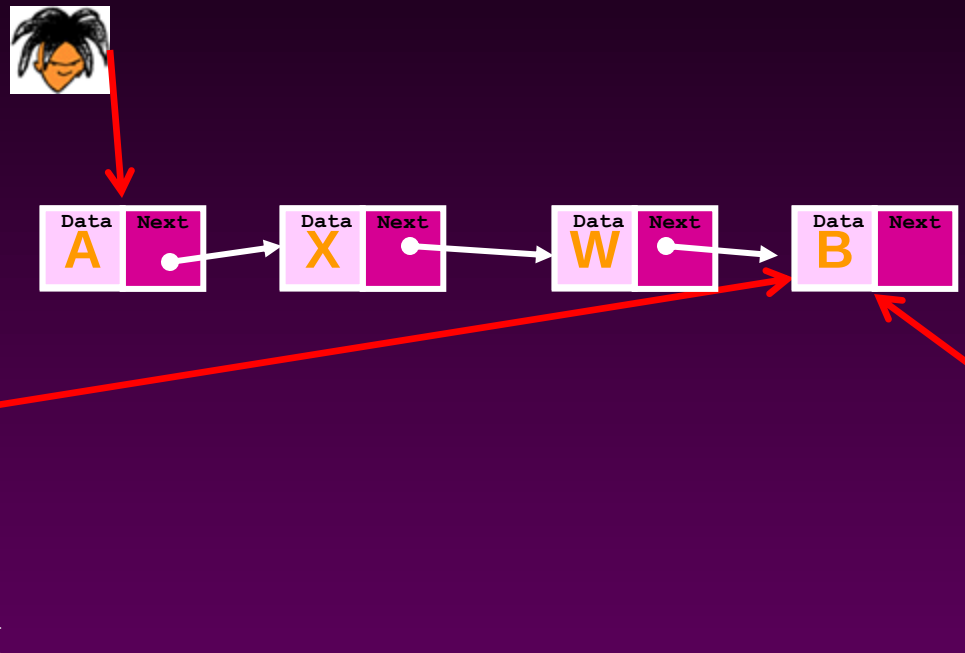
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
```



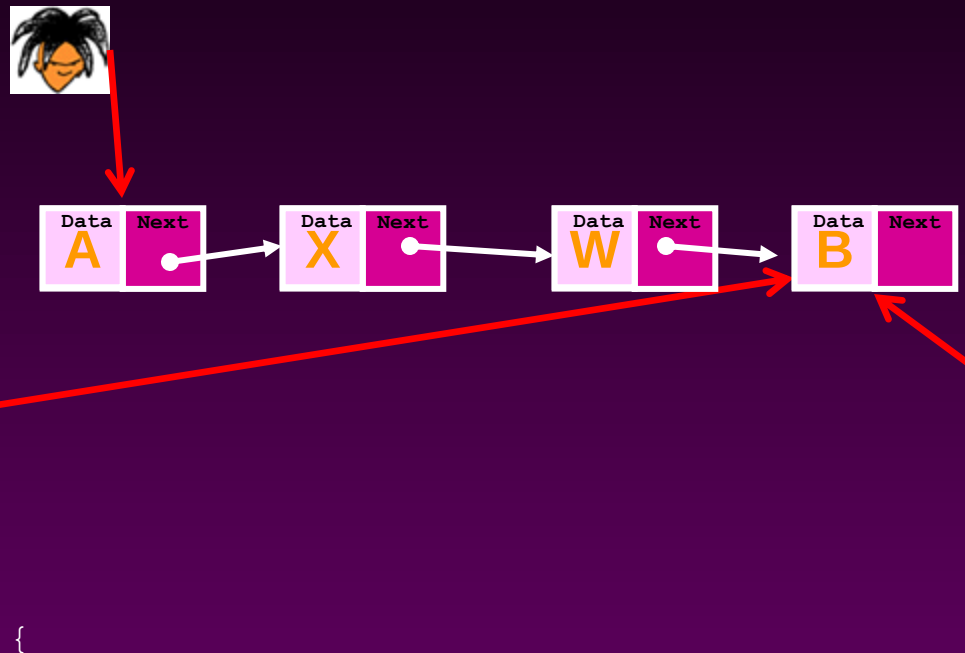
Code Executed:

```
void Enqueue(char x;){
        Node* newNode = new Node;
        newNode->Data = x;
        if ( isEmpty ( ) )
                front = newNode;
                rear = newNode;
        else
                rear->next = newNode;
                rear = newNode;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
MyQ.Dequeue(N);
```



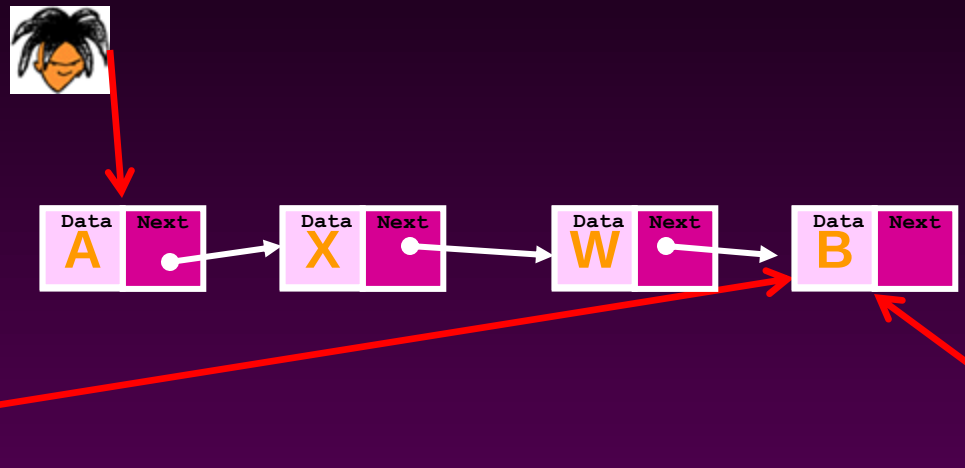
Code Executed:

```
Void Dequeue(char &x;){
        x = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
MyQ.Dequeue(N);
```



N

A

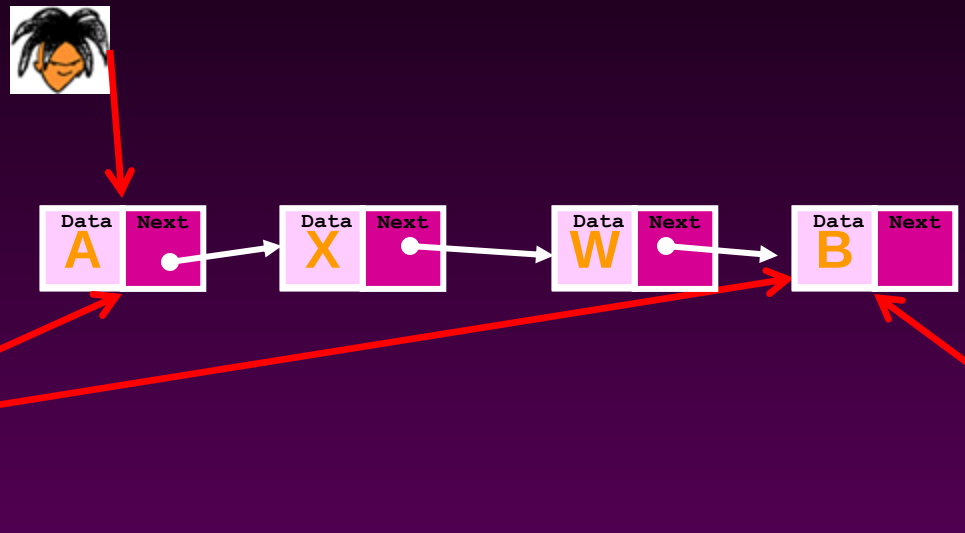
Code Executed:

```
Void Dequeue(char &x;){
        x = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
MyQ.Dequeue(N);
```



N

A

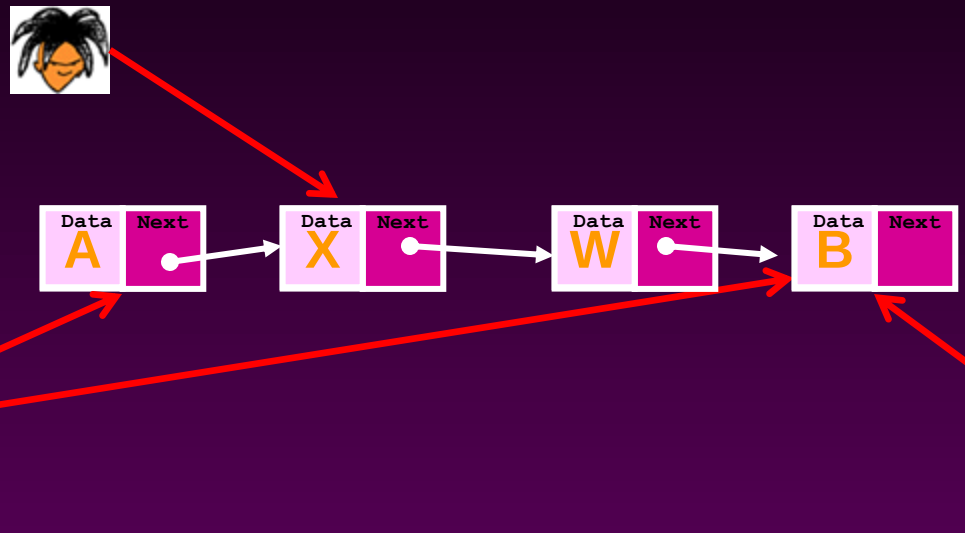
Code Executed:

```
Void Dequeue(char &x;){
        x = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
MyQ.Dequeue(N);
```



N

A

Code Executed:

```
Void Dequeue(char &x;){
        x = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
}
```

List Based Queue

Main Function:

```
Queue MyQ;
MyQ.Enqueue('A');
MyQ.Enqueue('X');
MyQ.Enqueue('W');
MyQ.Enqueue('B');
MyQ.Dequeue(N);
```

N

A

newNode

temp

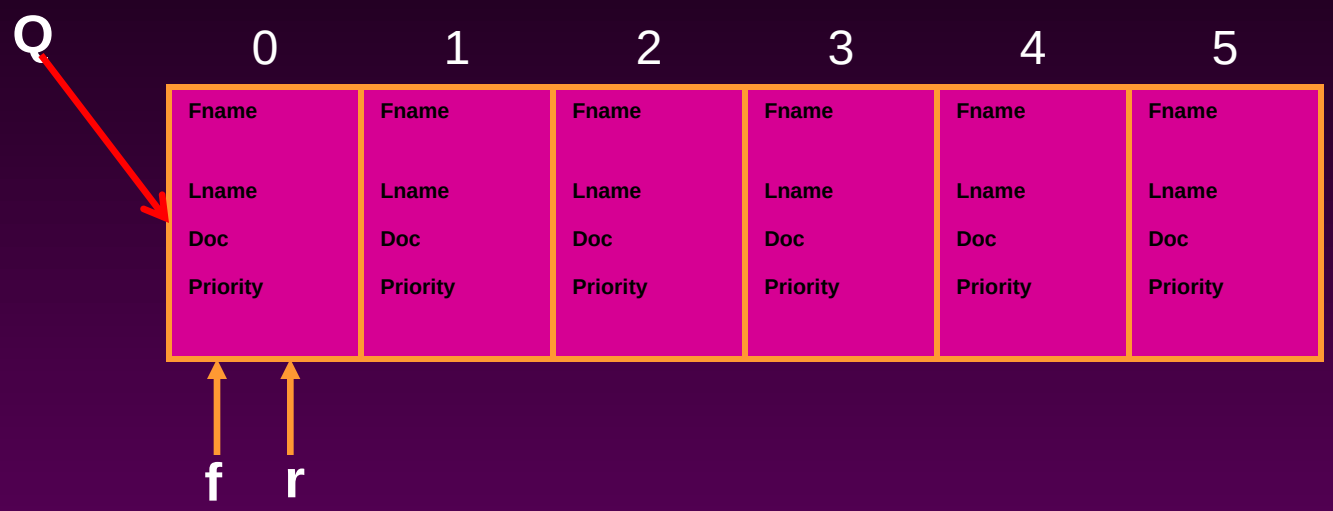
Code Executed:

```
Void Dequeue(char &x;){
        x = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
}
```

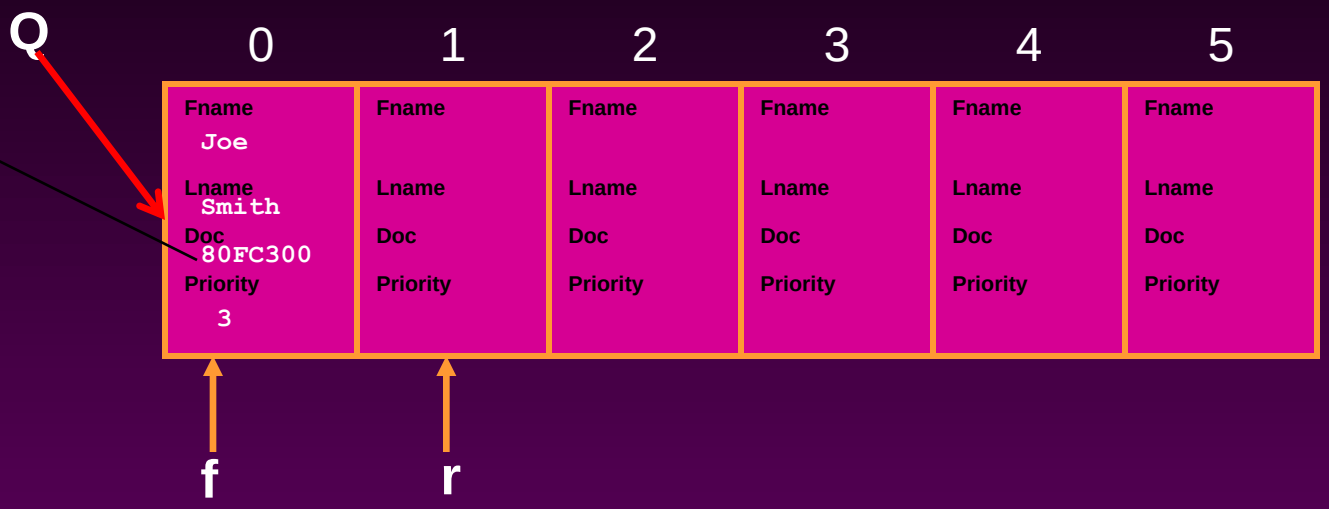
Priority Based Queue (array implementation)

```
struct QueData{
    string Fname;
    string Lname;
    binary *doc;
    int Priority;
};

 QueData Q[6];
 front = 0;
 Rear = 0;
```

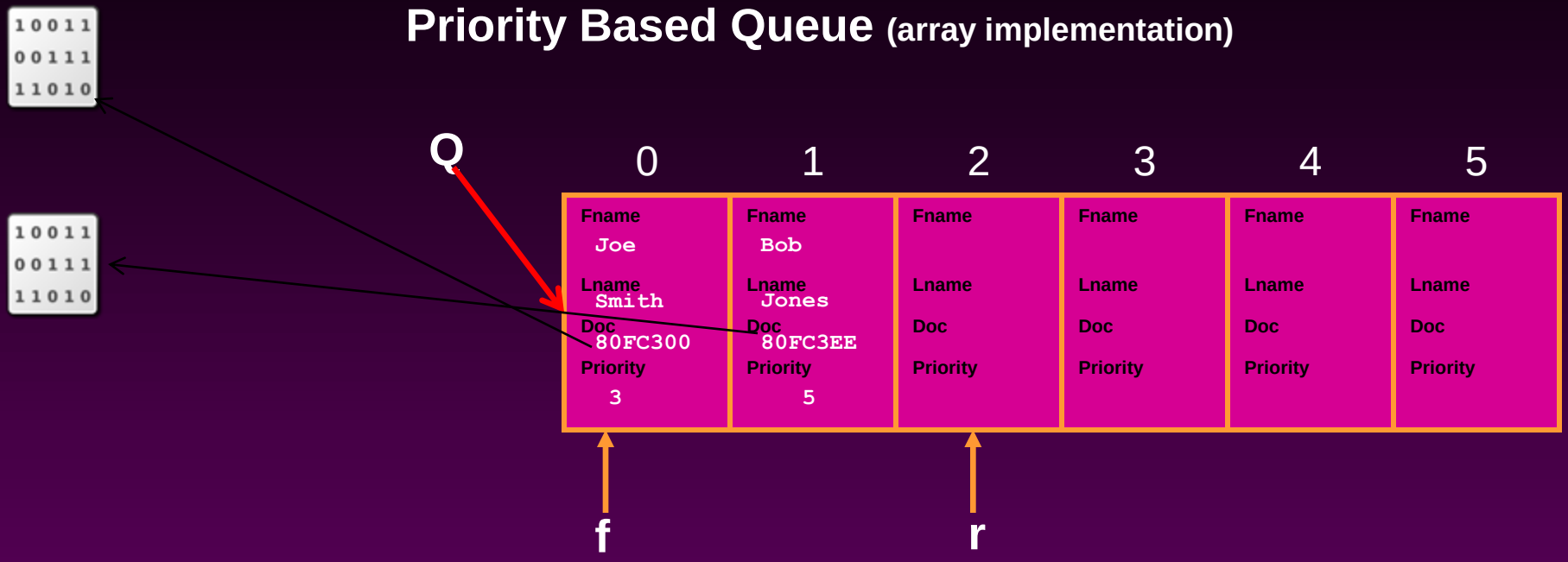


Priority Based Queue (array implementation)



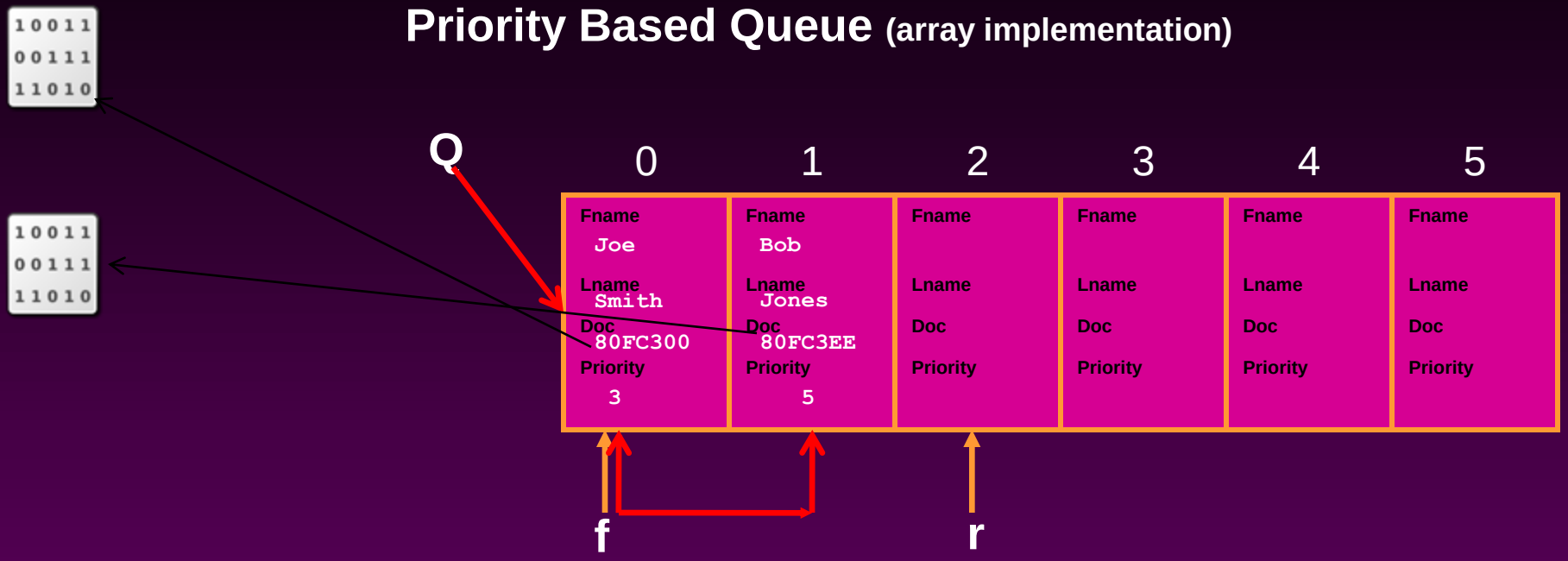

```
Enqueue("Joe","Smith","homework.doc",3);
```

Priority Based Queue (array implementation)



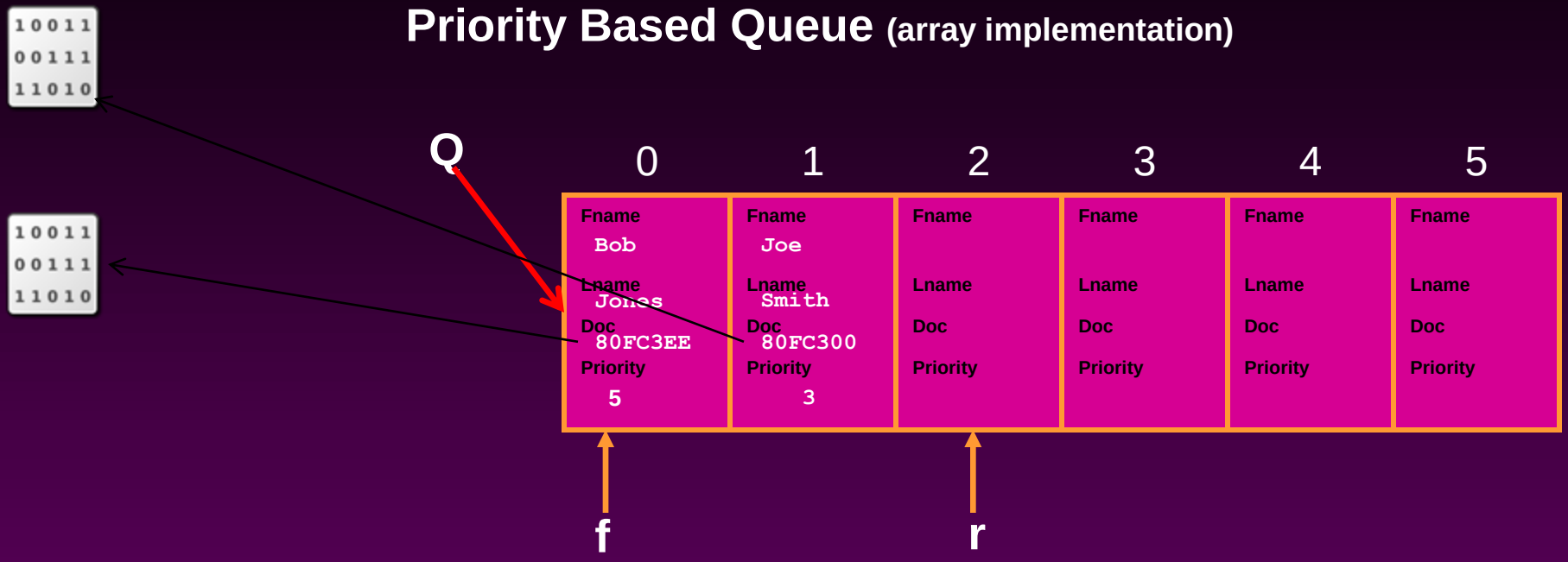

```
Enqueue("Bob","Jones","presentation.ppt",5);
```

Priority Based Queue (array implementation)



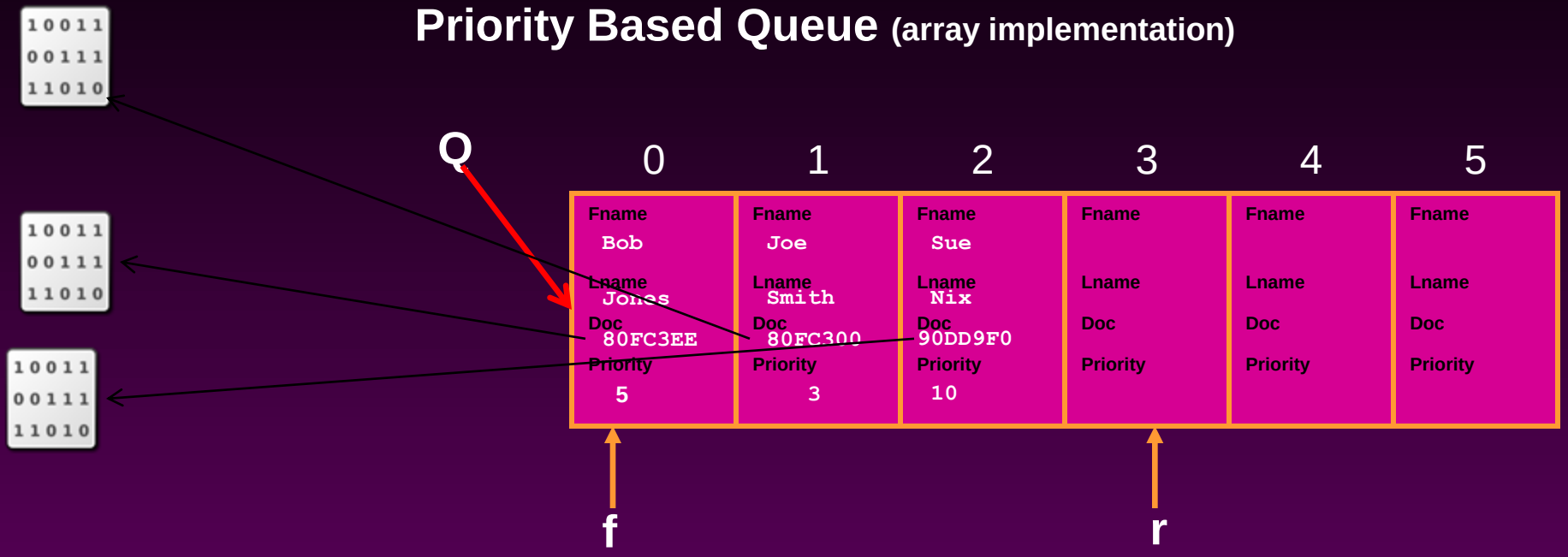
Compare the priorities…

Priority Based Queue (array implementation)



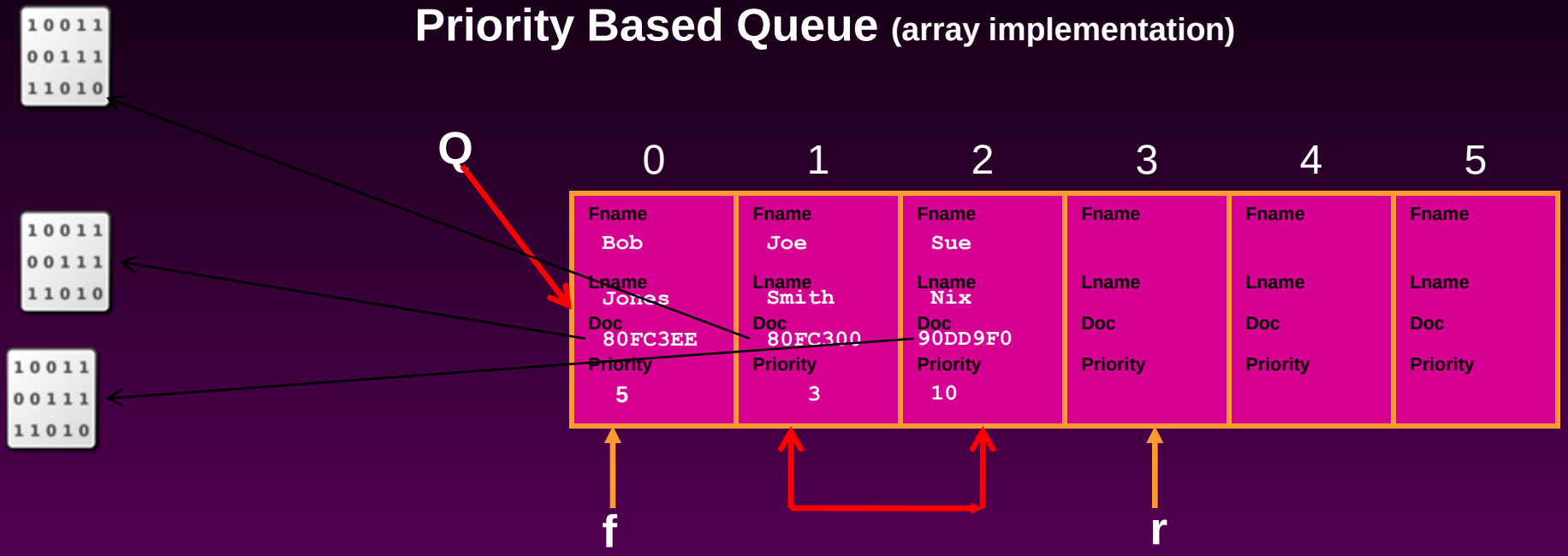
5 is larger (higher) so we swap…

Priority Based Queue (array implementation)



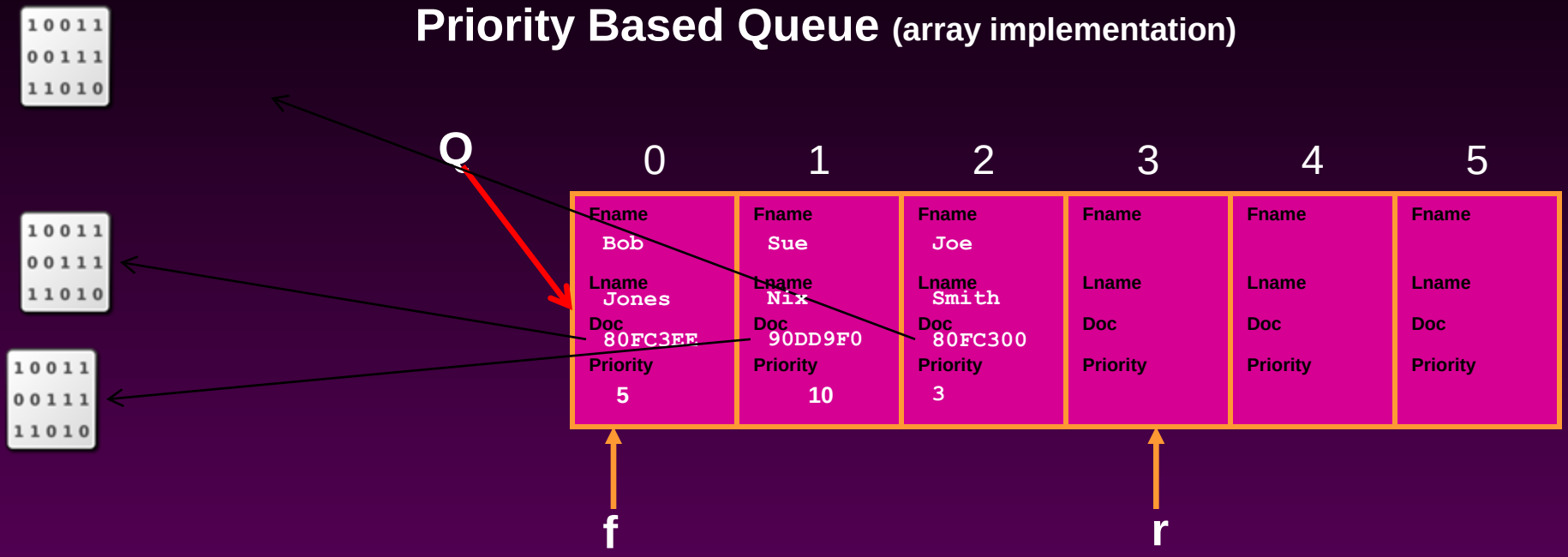

```
Enqueue("Sue","Nix","companypay.form",10);
```

Priority Based Queue (array implementation)



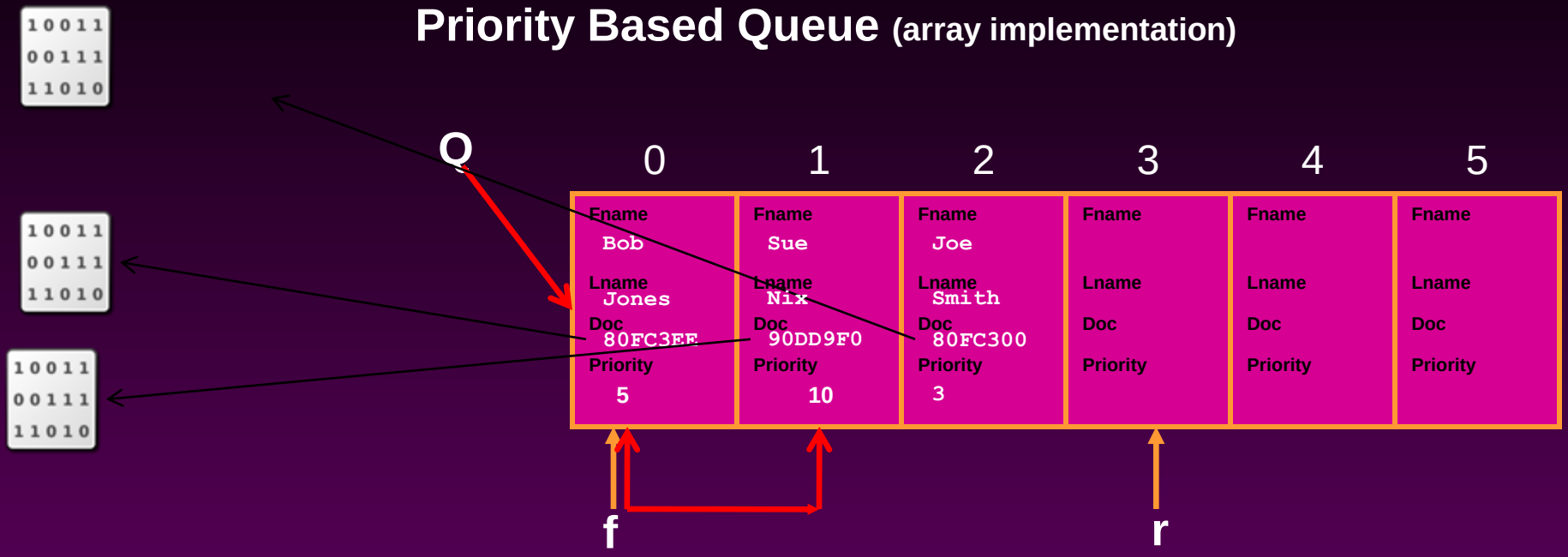
Compare the priorities…

Priority Based Queue (array implementation)



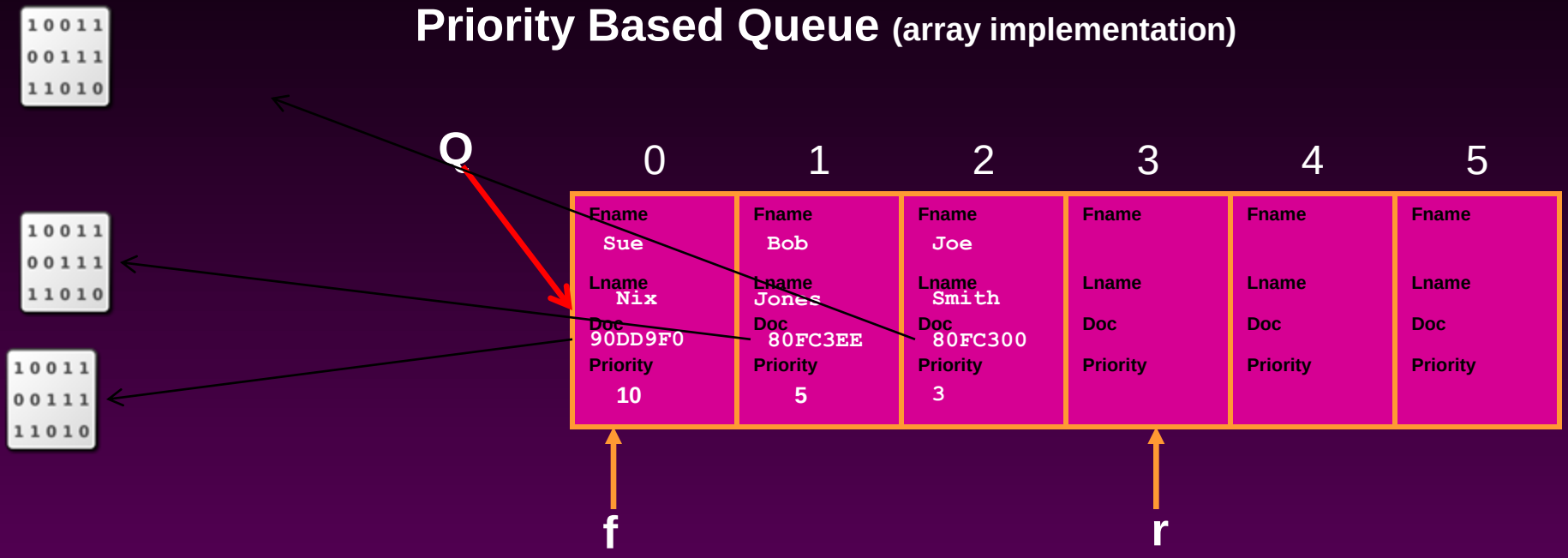
10 is larger so we swap…

Priority Based Queue (array implementation)



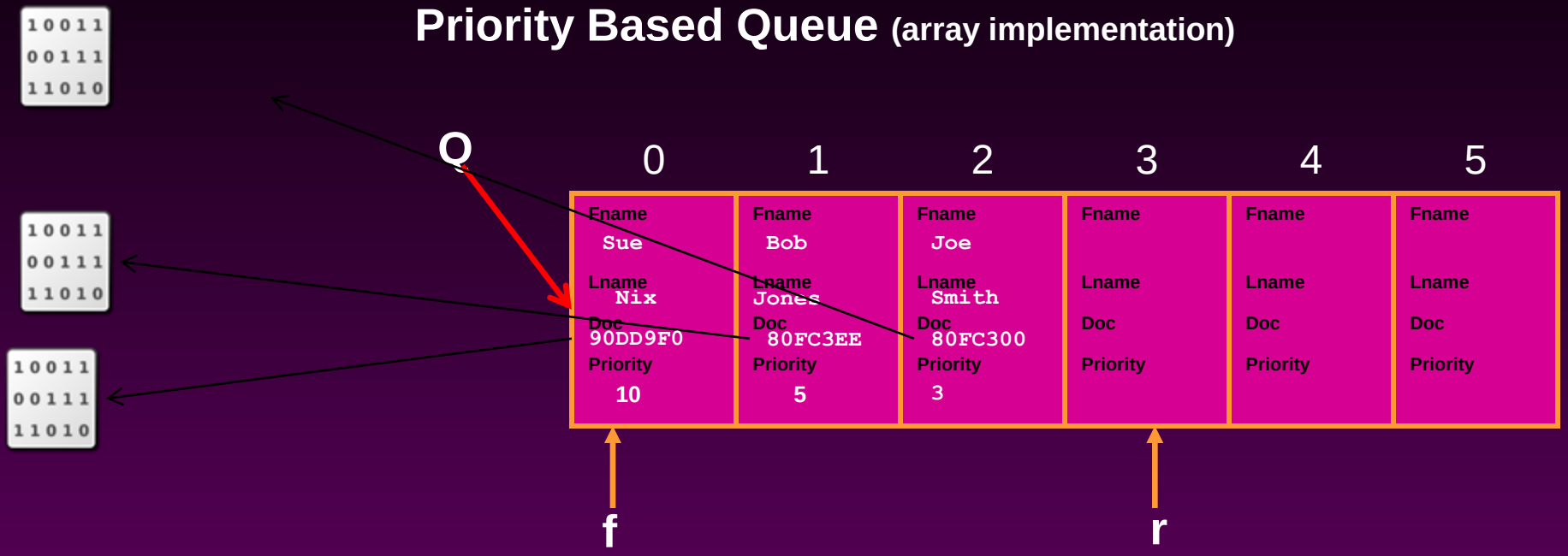
Compare again…

Priority Based Queue (array implementation)

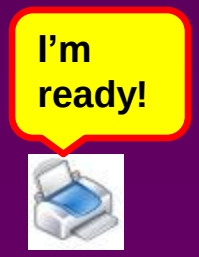


10 is still larger, so we swap again…

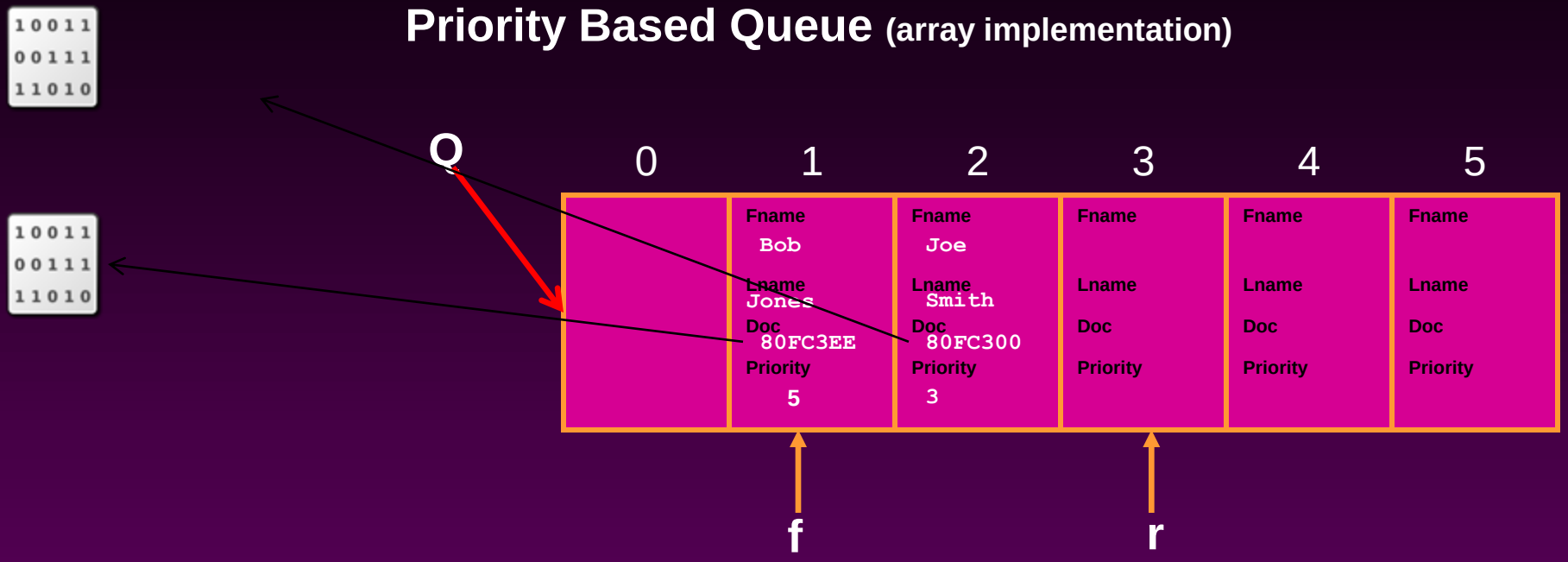
Priority Based Queue (array implementation)




```
Printer sends the Dequeue();
```

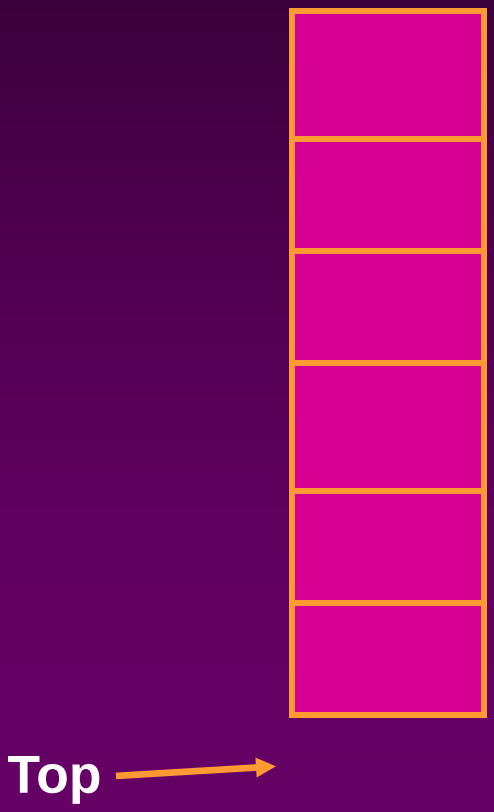


Priority Based Queue (array implementation)



Stacks

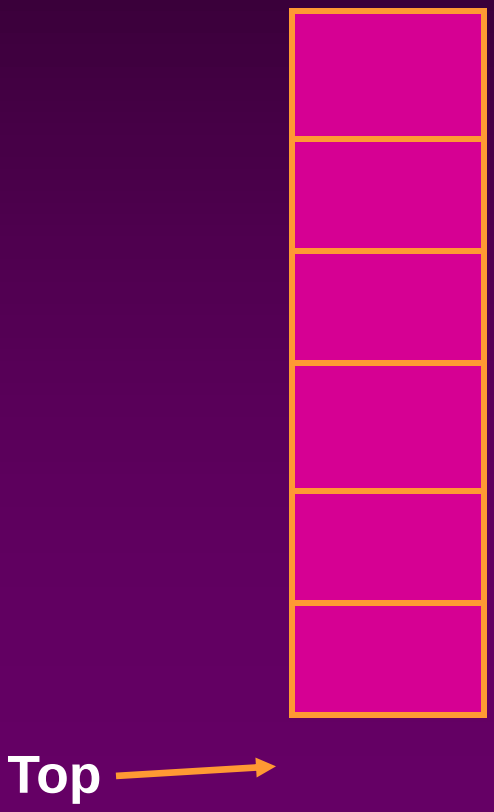
- Queue implementation use a FIFO ordering
- Stacks use a LIFO ordering
- Queue uses:
  - Enqueue
  - Dequeue
- Stack uses:
  - Push
  - Pop



Stacks

- **For example:**

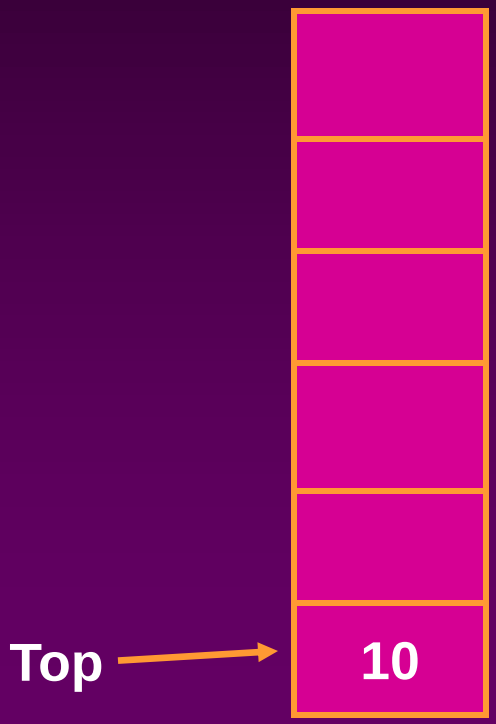
```
Push(10);
```



Stacks

- **For example:**

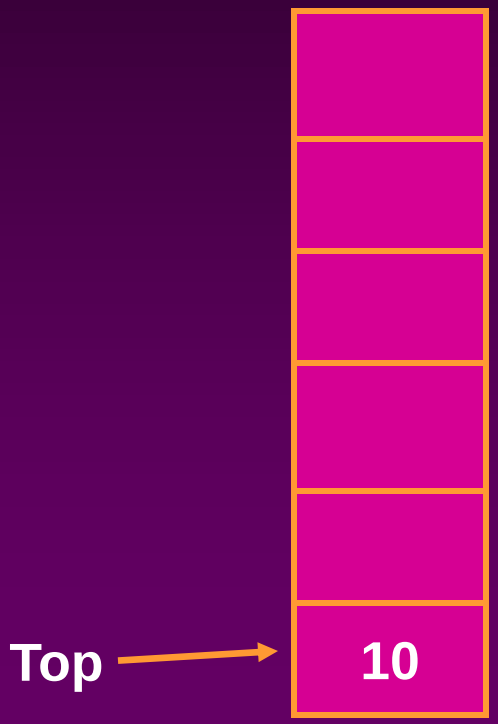
```
Push(10);
```



Stacks

- **For example:**

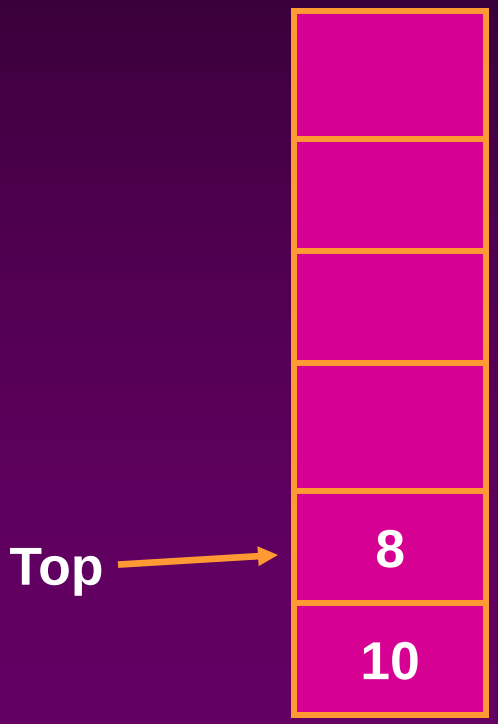
```
Push(10);
Push(8);
```



Stacks

- **For example:**

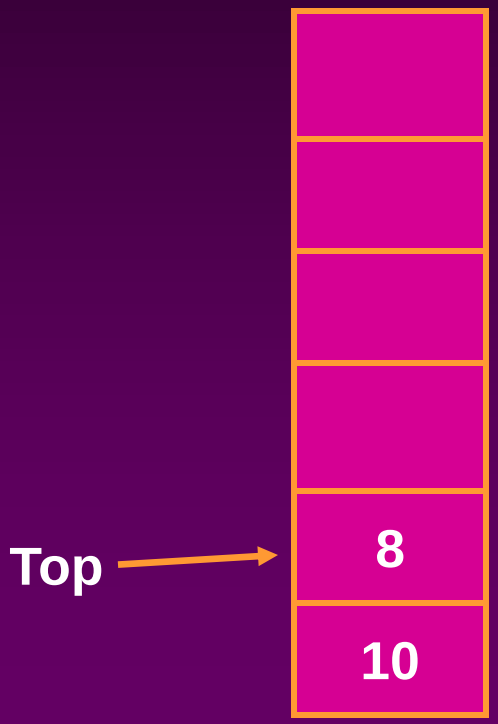
```
Push(10);
Push(8);
```



Stacks

- **For example:**

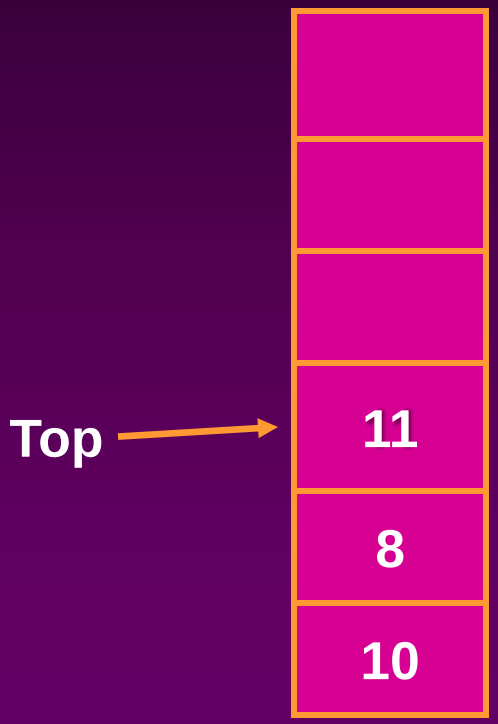
```
Push(10);
Push(8);
Push(11);
```



Stacks

- **For example:**

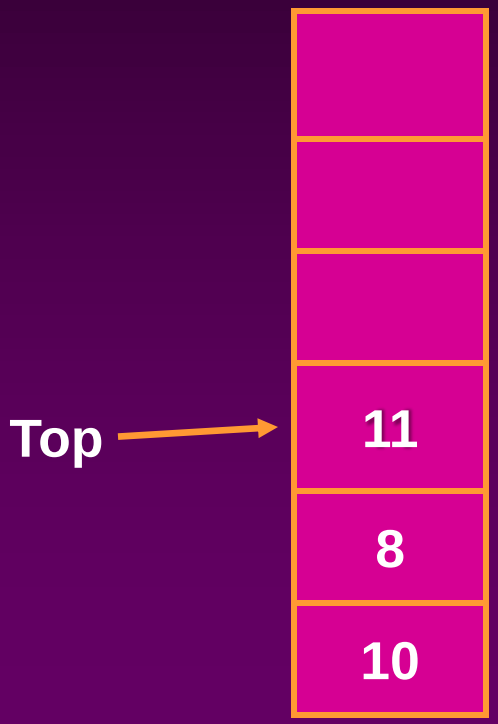
```
Push(10);
Push(8);
Push(11);
```



Stacks

- **For example:**

```
Push(10);
Push(8);
Push(11);
Pop();
```



Stacks

- **For example:**

```
Push(10);
Push(8);
Push(11);
Pop();
```