ADT
Linked List

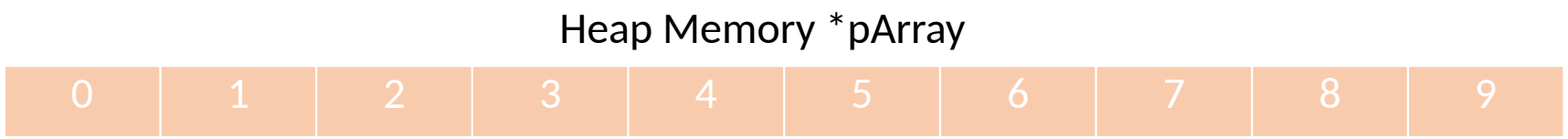
Hermawan

ADT LinkedList

LinkedList adalah Abstract Data Type (ADT) yang digunakan untuk membangun struktur Dinamis Bag-Data satu dimensi.

Alokasi memori dinamis didalam heap dapat dilakukan dengan Alokasi Array pointer,

```
int *pArray = new int[?]
```

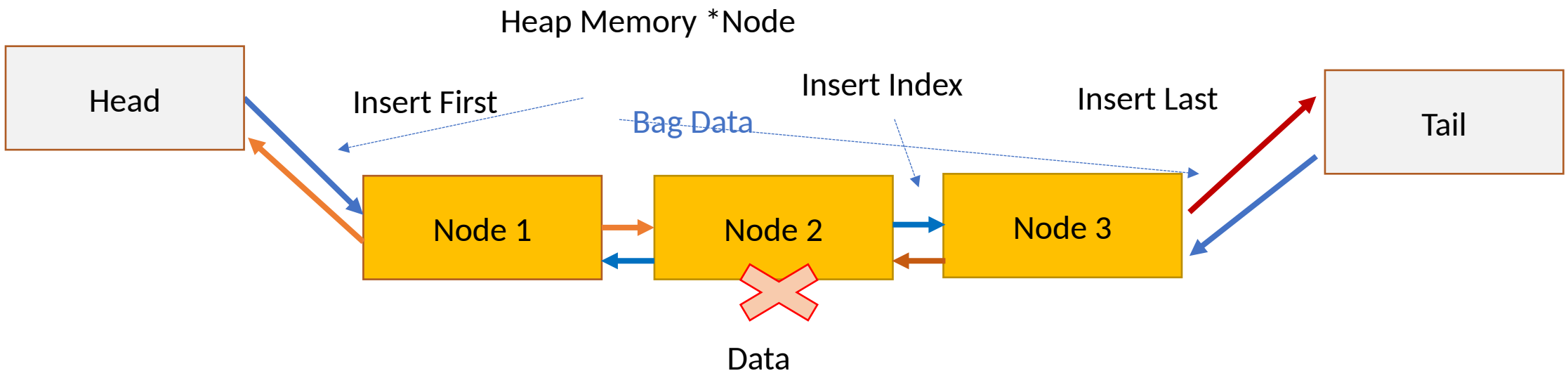


Tetapi struktur ini hanya bisa berubah ukuran tanpa bias melakukan operasi Bag-Data untuk memasukkan, menghapus secara Dinamis.

ADT LinkedList

LinkedList adalah Abstract Data Type (ADT) yang digunakan untuk membangun struktur Dinamis Bag-Data satu dimensi.

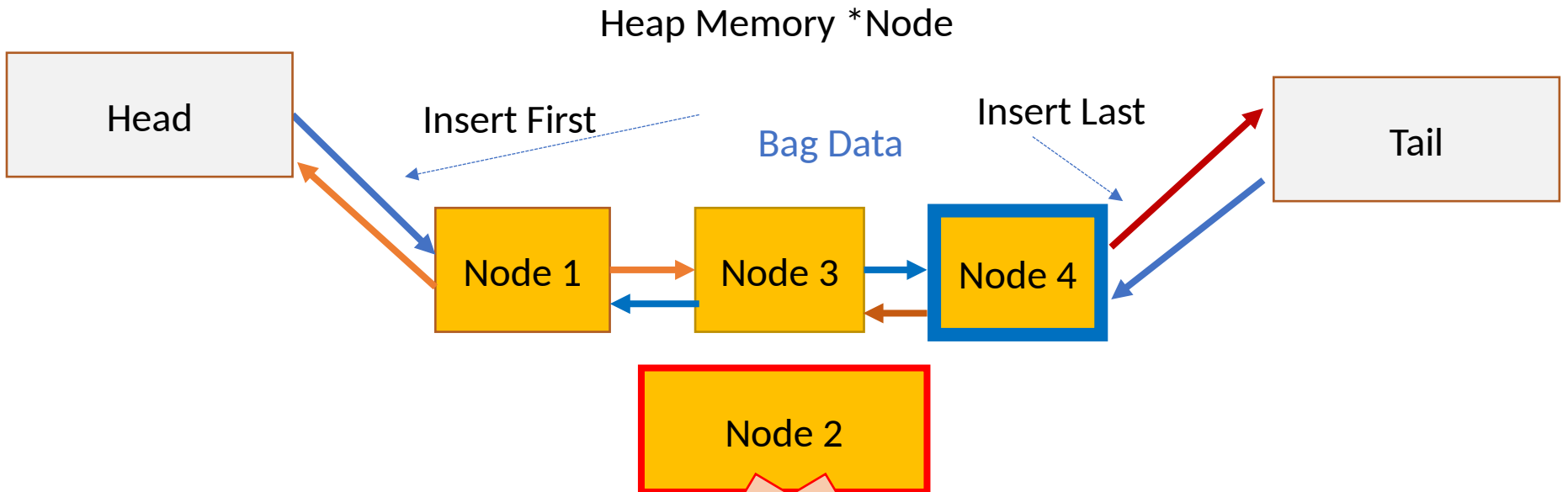
Struktur Dinamis Bag-Data Linked



ADT LinkedList

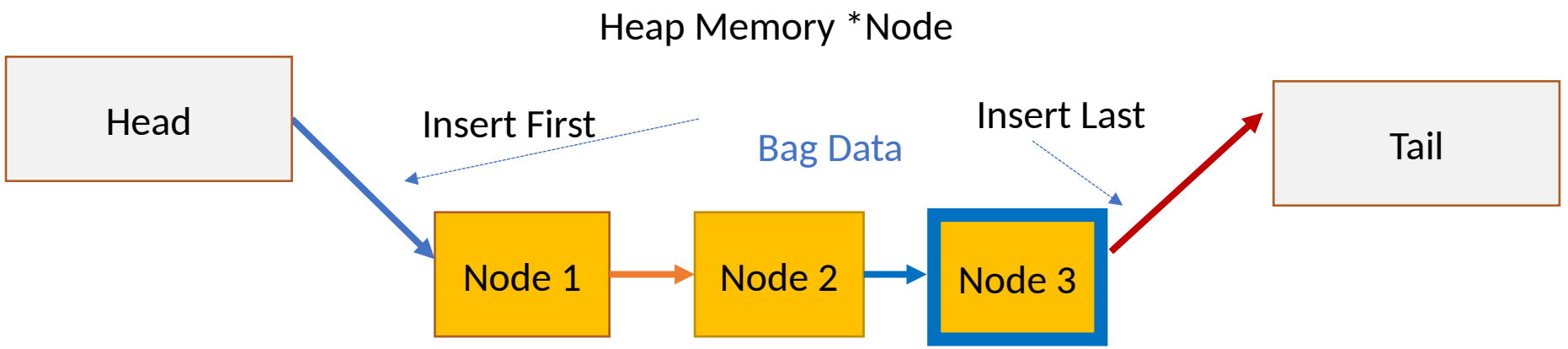
LinkedList adalah Abstract Data Type (ADT) yang digunakan untuk membangun struktur Dinamis Bag-Data satu dimensi.

Struktur Dinamis Bag-Data Linked



ADT LinkedList
Single List

LinkedList yang hanya memiliki satu jalur traversal.



ADT LinkedList
Single List

Struktur ADT,

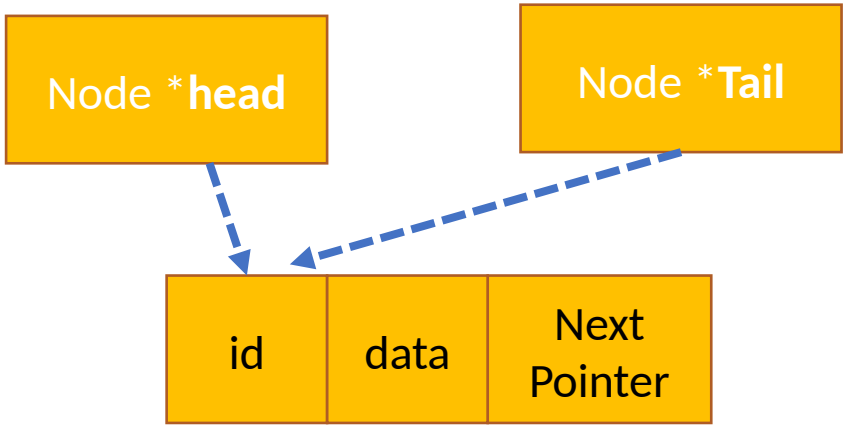
1. Node


```
struct Node {
    int id;
    char* data;
    struct Node* next;
};
```

ADT LinkedList
Single List

Struktur ADT,

2. Bag Frame Head-Tail/Start-End

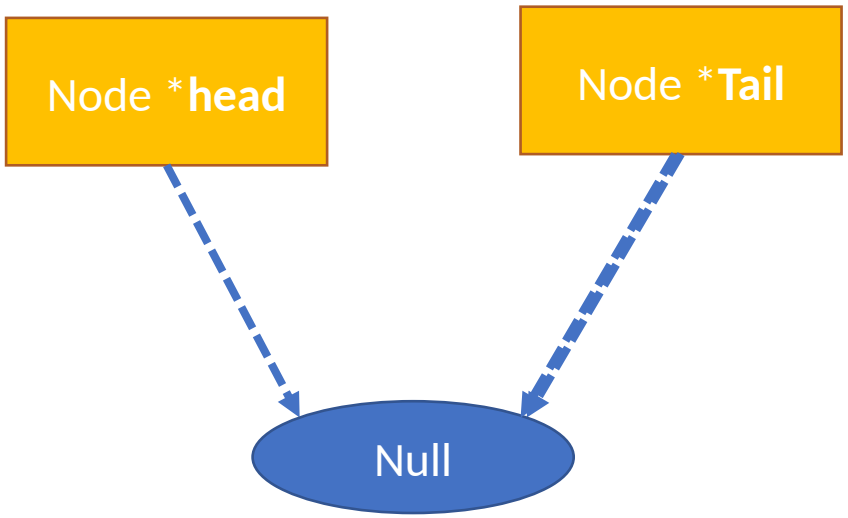



```
class linkedList{
    Node *head;
    Node *tail;
...
}
```

ADT LinkedList
Single List

Struktur ADT Operation

1. Init

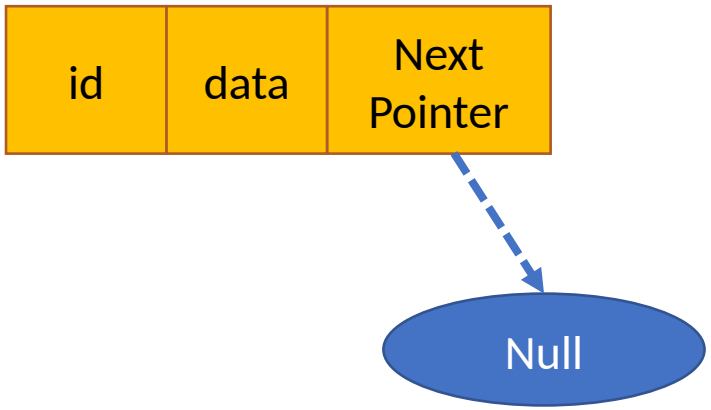



```
linkedList::linkedList(){
    this->setHead(NULL);
    this->setTail(NULL);
}
```

ADT LinkedList
Single List

Struktur ADT Operation

2. Create Node

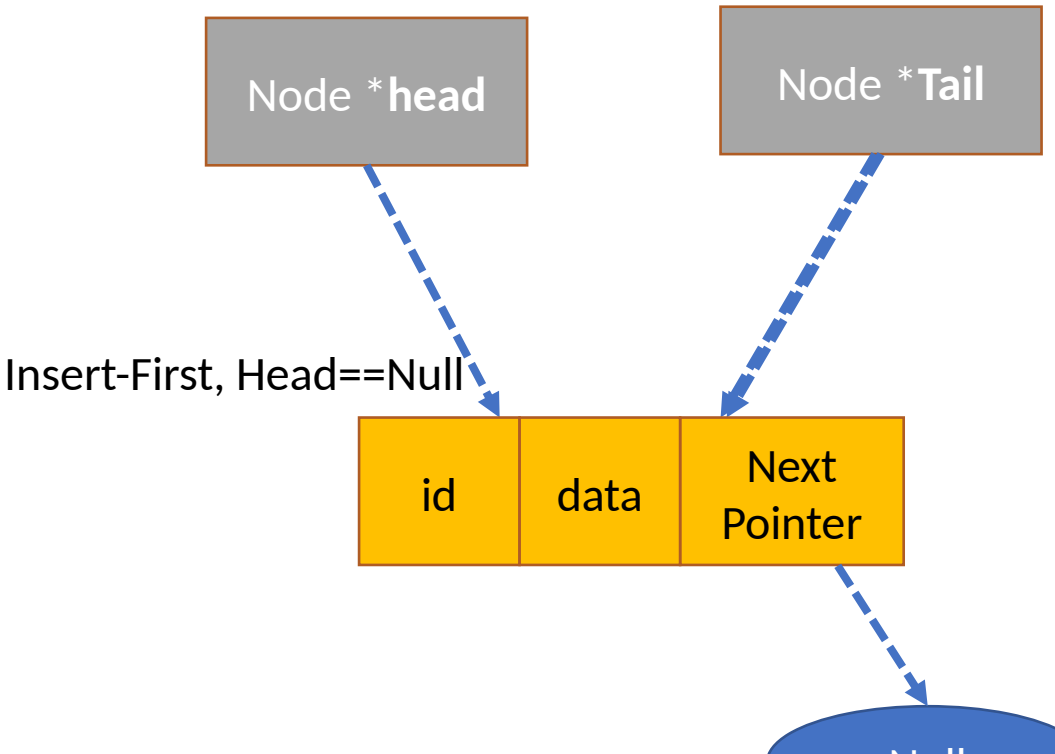



```
Node* createNode(int id, char* data)
{
    Node* temp = new Node;
    temp->id = id;
    temp->data= data;
    temp->next = NULL;
    return temp;
}
```

ADT LinkedList
Single List

Struktur ADT Operation

3. Insert Node



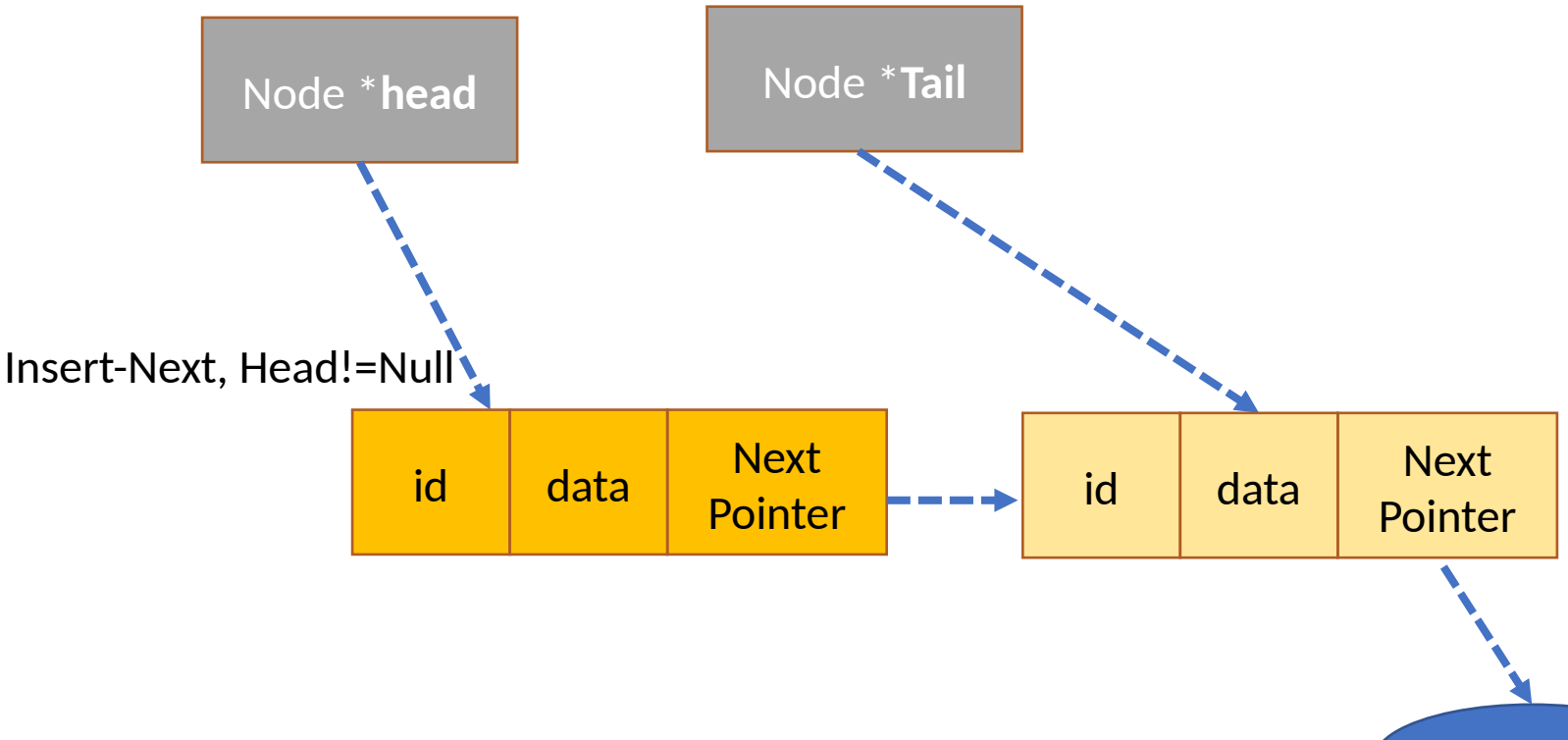

```
void linkedList::insertNode(Node*
node){
if(head==NULL){
        head=node;
        tail=node;
    }
    else{
        tail->next=node;
        tail=node;
    }
}
```

ADT LinkedList Single List

Struktur ADT Operation

3. Insert Node

```
void linkedList::insertNode(Node*
node){
if(head==NULL){
      head=node;
      tail=node;
   }
   else{
      tail->next=node;
      tail=node;
   }
}
```

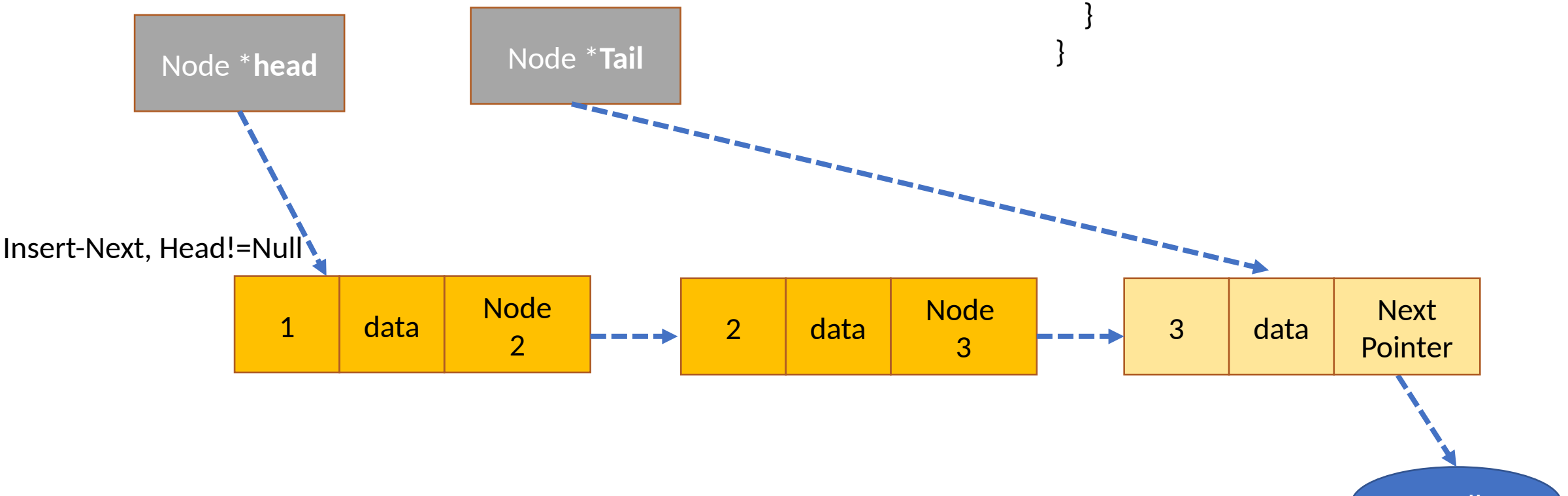


ADT LinkedList Single List

Struktur ADT Operation

3. Insert Node

```
void linkedList::insertNode(Node*
node){
if(head==NULL){
      head=node;
      tail=node;
   }
   else{
      tail->next=node;
      tail=node;
   }
}
```

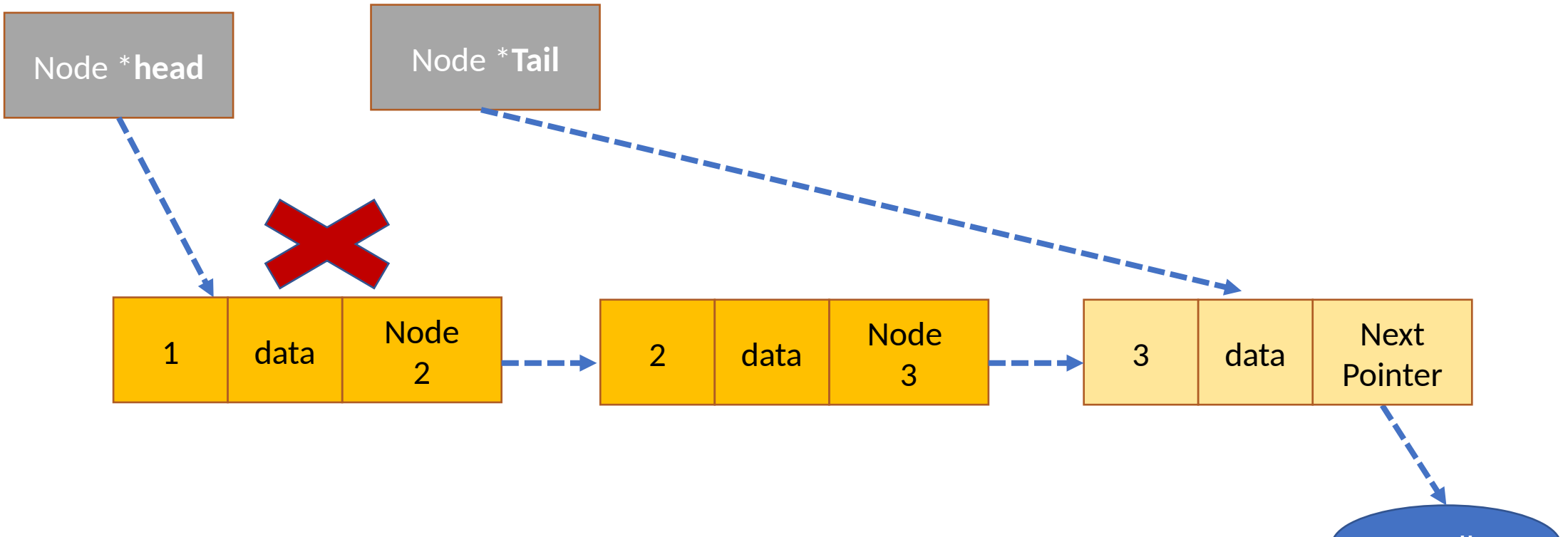


ADT LinkedList Single List

Struktur ADT Operation

4. Remove HeadNode (Stack Operation)

```
void linkedList::RemoveHead(Node*
node){
   if(head!=NULL){
      Node *temp;
      temp=head;
      head=head->next;
      delete(temp);
   }
```

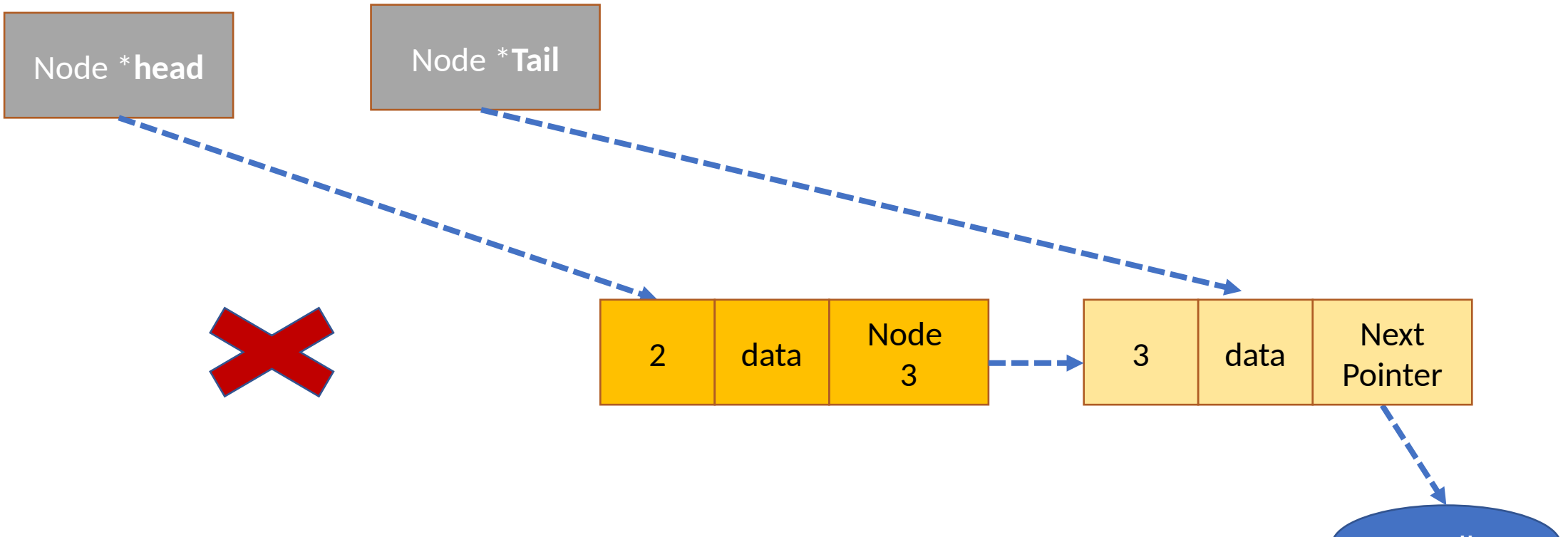


ADT LinkedList Single List

Struktur ADT Operation

4. Remove HeadNode (Stack Operation)

```
void linkedList::RemoveHead(Node*
node){
if(head!=NULL){
      Node *temp;
      temp=head;
      head=head->next;
      delete(temp);
   }
```

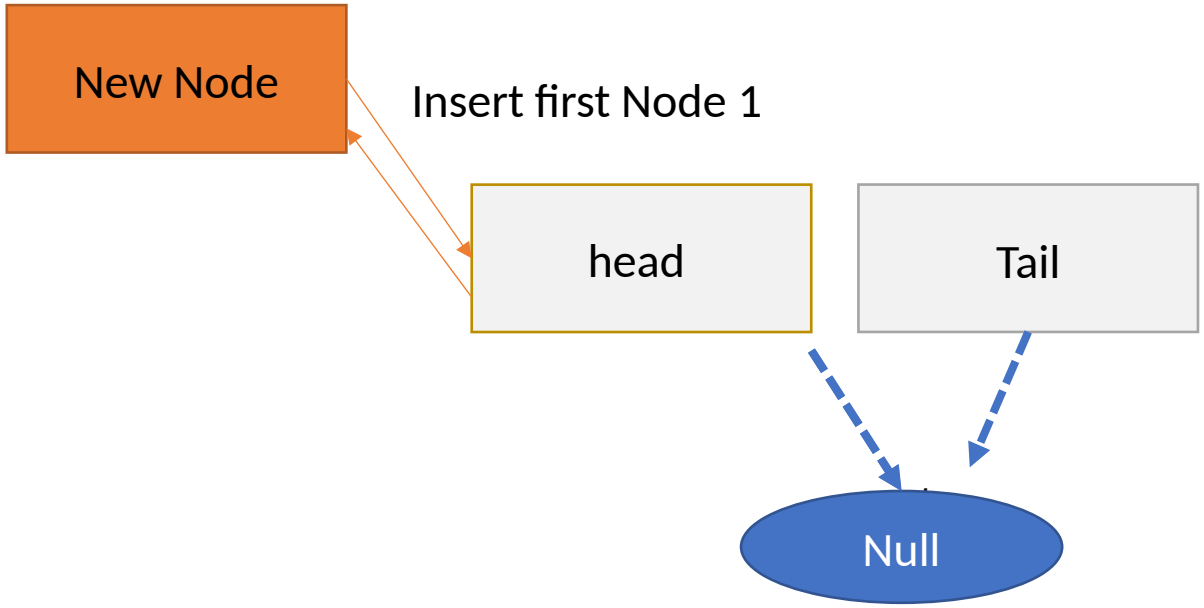


ADT LinkedList
Single List

Quiz 1:

1. Bagaimana cara menghapus menggunakan index (id)
2. Bagaimana cara menghapus menggunakan index counter ke-?
3. Bagaimana cara menghapus (Queue Operation) Tail

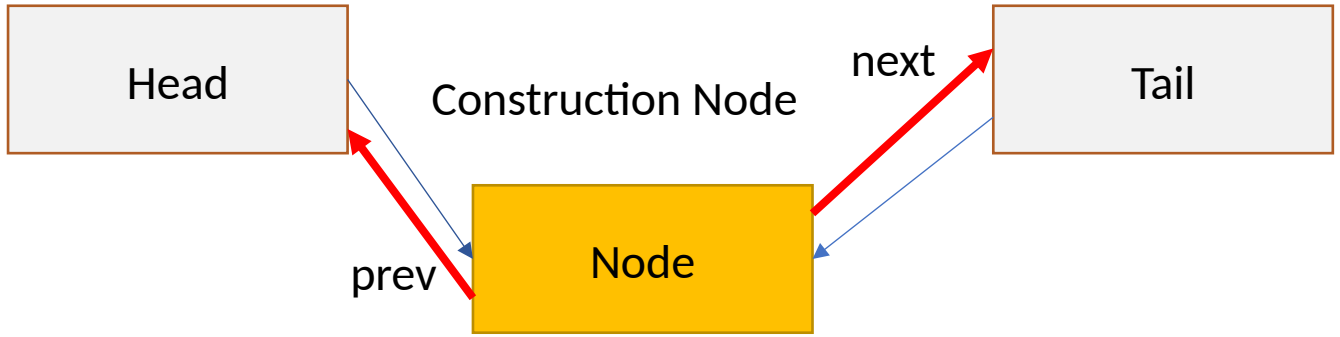
ADT LinkedList
Double List



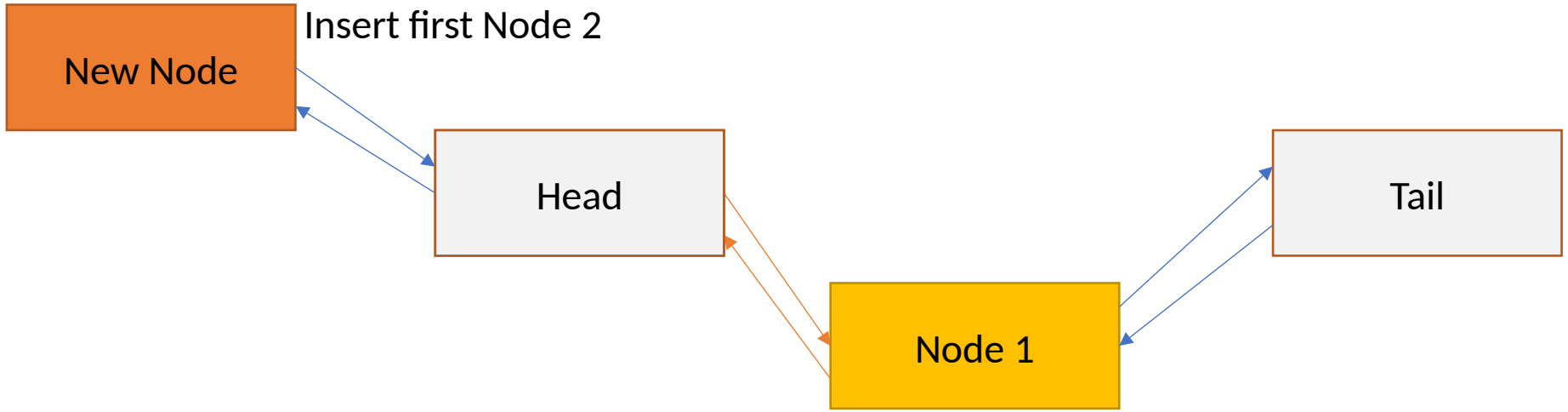

```
struct Node {
    int id;
    char* data;
    struct Node* next;
    struct Node* prev;
};
```

ADT LinkedList
Double List

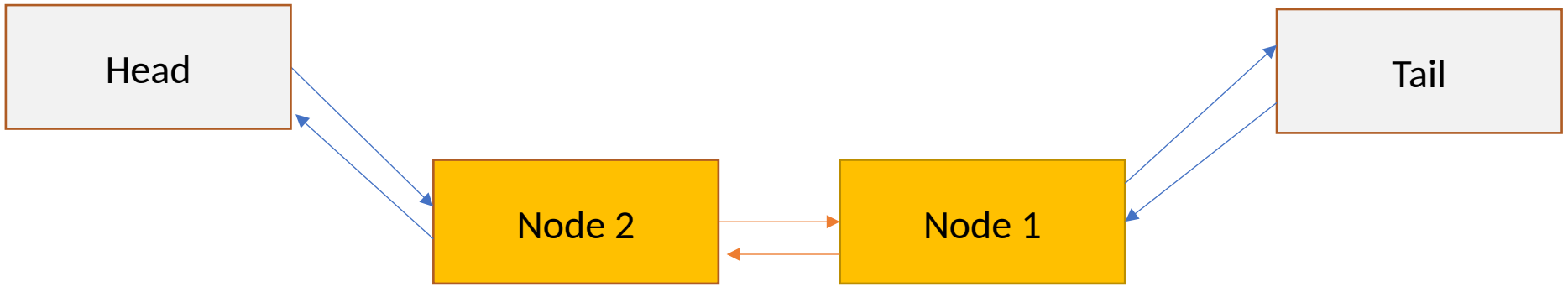
```
struct Node {
    int id;
    char* data;
    struct Node* next;
    struct Node* prev;
};
```



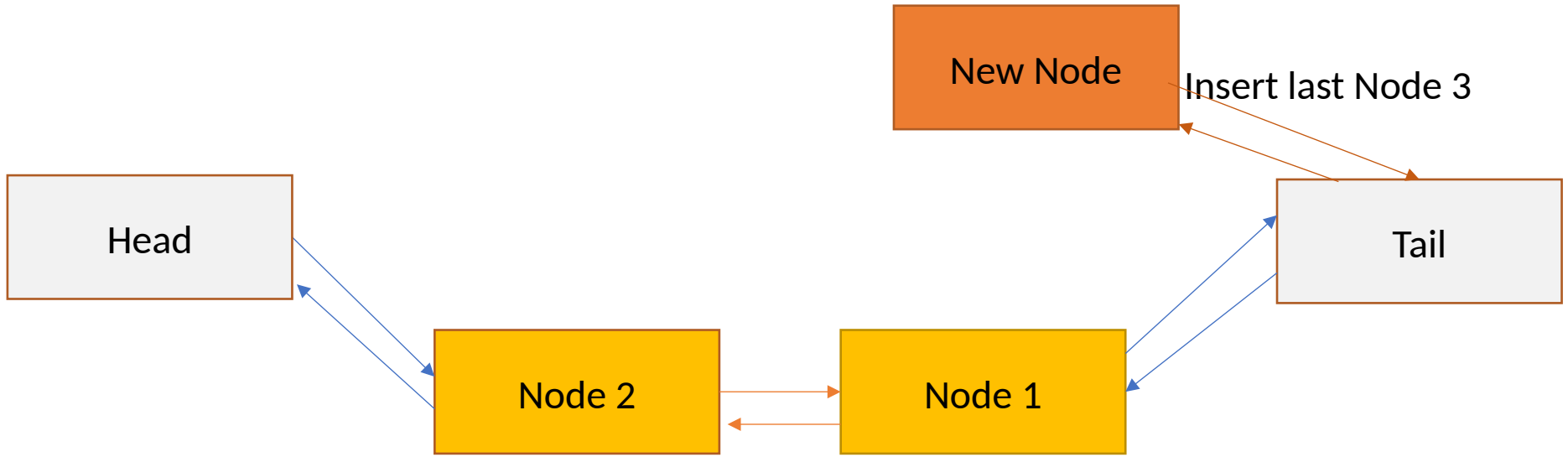
ADT LinkedList
Double List



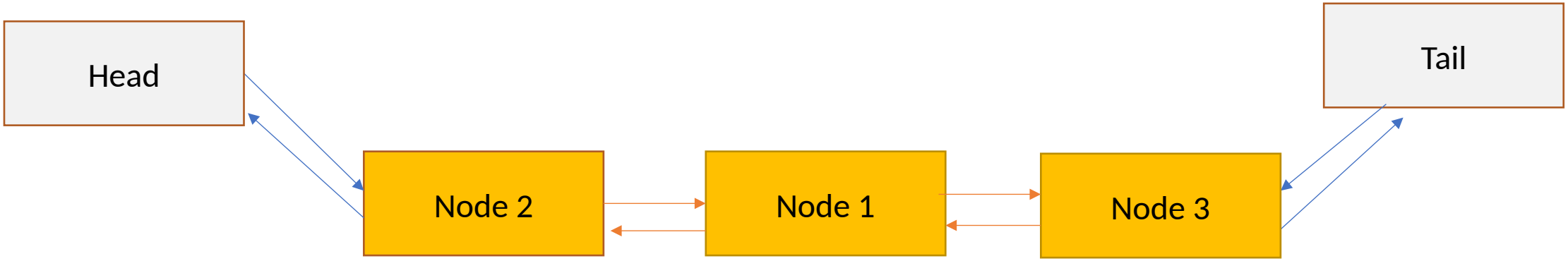
ADT LinkedList
Double List



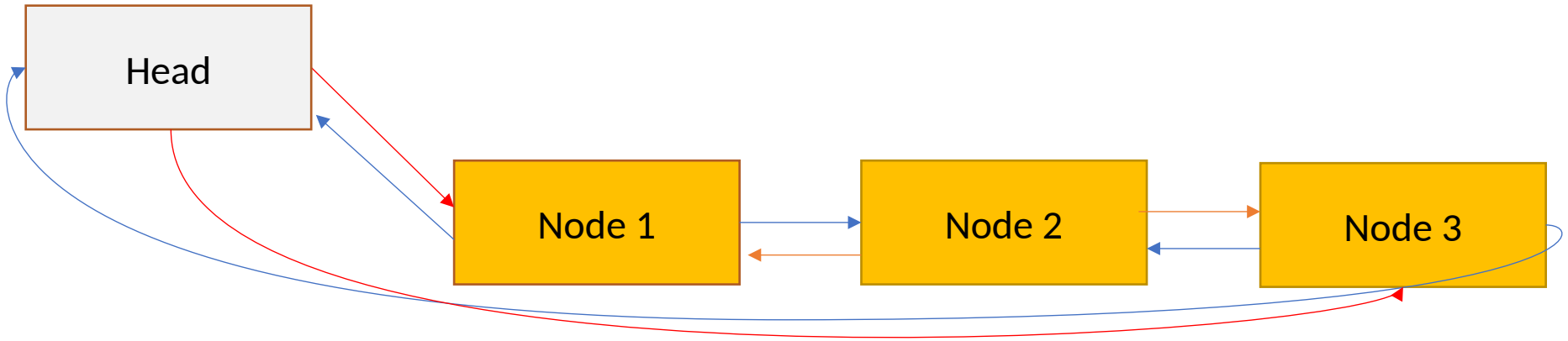
ADT LinkedList
Double List



ADT LinkedList
Double List



ADT LinkedList
Circular List



ADT LinkedList
Single List

Quiz 2:

1. Bagaimana cara menginputkan dan menghapus Stack Head
2. Bagaimana cara menginputkan dan menghapus Queue Head
3. Bagaimana cara inisialisasi dan operasional Circular Linked-List