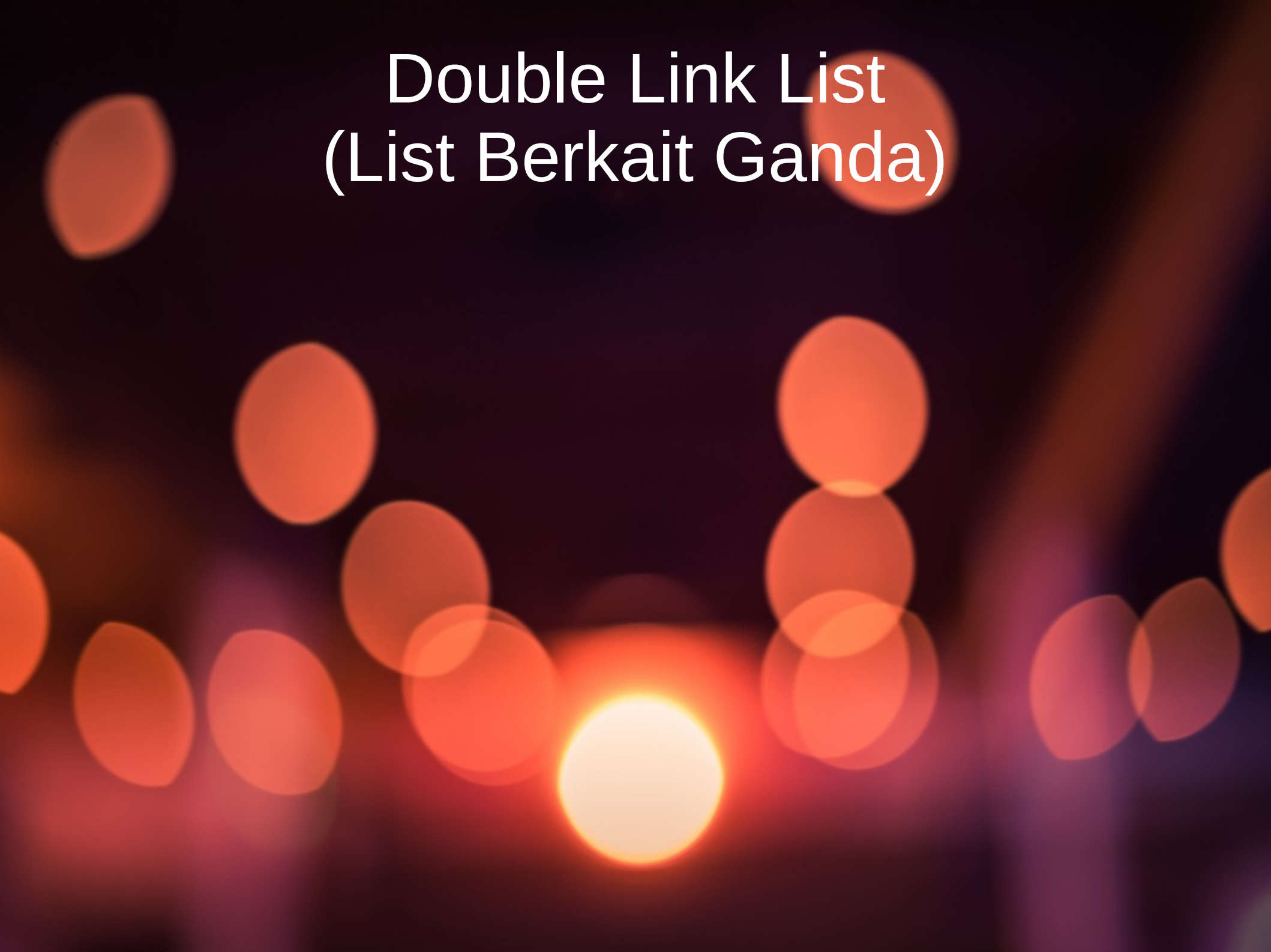

Double Link List
(List Berkait Ganda)

Linked List

- Linked list yang sudah kita pelajari bersifat linear linked list
- Semua elemen dapat ditelusuri dari awal hingga akhir
- Linked list ini memiliki satu pointer ke depan

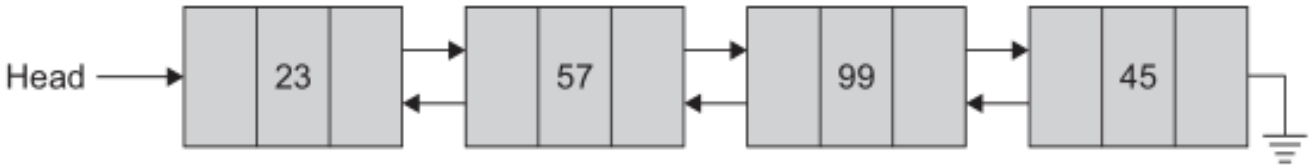
- Linked list jenis lain : Double Linked List

  memiliki dua pointer ke depan dan ke belakang

Jenis Linked List

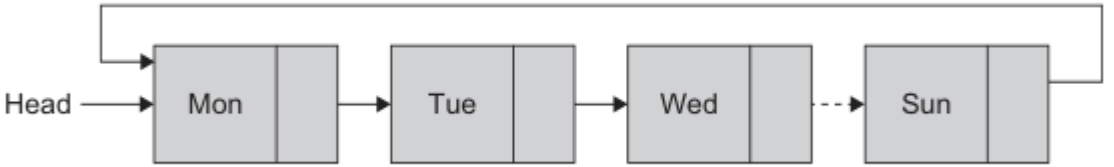
- Double Linked List

  tiap node memiliki pointer next (depan) dan prev(belakang)



- Circular linked list

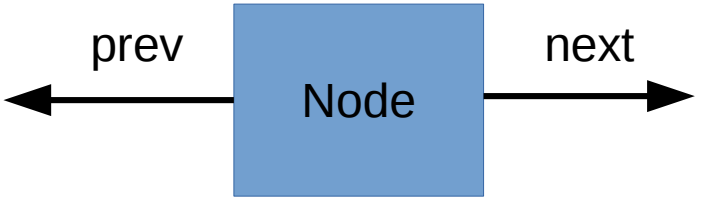
  node paling belakang menunjuk next ke head



Implementasi Double Link List

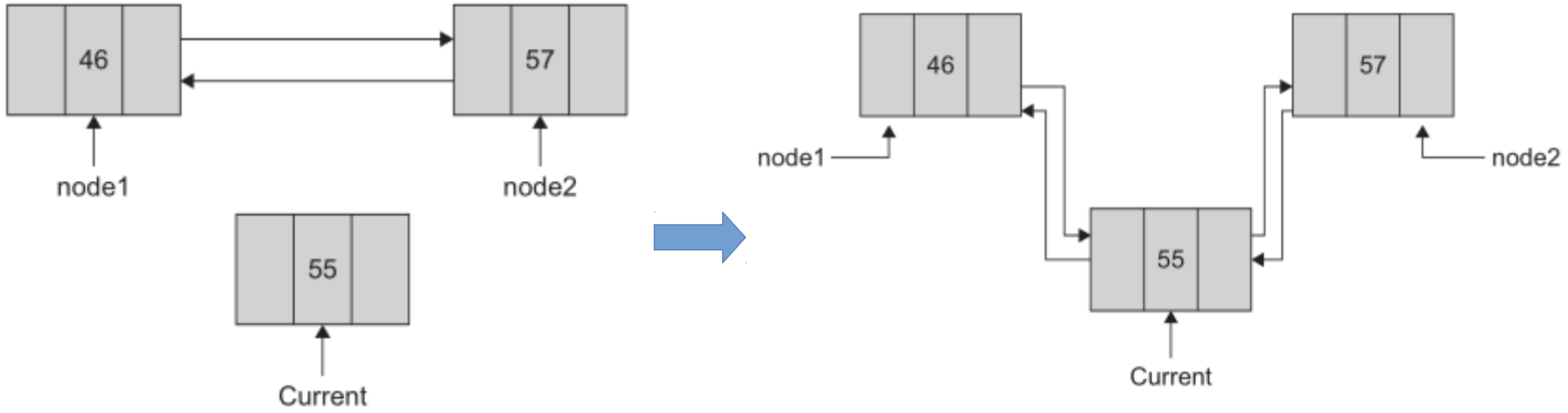
- Definisi node

```
class DLL_Node
{
public:
  int Data;  //data integer dibuat publik
  DLL_Node *prev, *next;

  DLL_Node() //konstruktur node
  {
    prev = next = 0; //tidak menunjuk kemana pun
  }
};

class DList //double link list
{
private:
  DLL_Node *Head, *Tail;  //2 pointer
public:
  DList()
  {
    Head = Tail = 0; //tidak menunjuk kemana pun
  }
  //fungsi pendukung lain dispesifikasi disini
};
```



Insert node ke Double Link List

- Update pointer



- Pointer yang diupdate:

  node1. next, node2. prev, current.next, current.prev

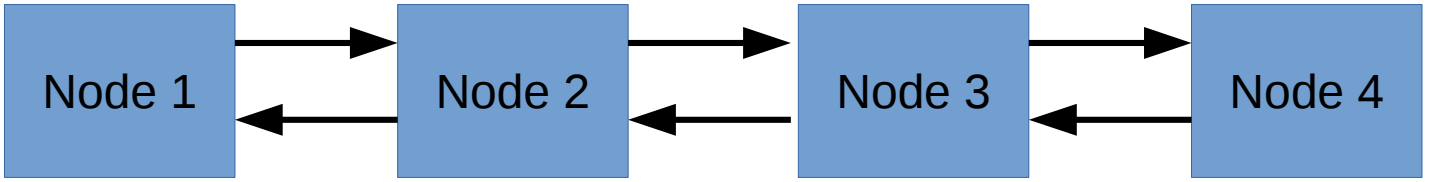
Update pointer pada operasi insert

- node1→next = current
- Node2 → prev = current
- Current → prev = node1
- Current → next = node2

```
//insert node Current di antara node1 & 2
node1->next = Current;
node2->prev = Current;
Current->next = node2;
Current->prev = node1;
```

Menyisipkan node pada posisi ke n

- Algoritma memperhatikan posisi tiap node



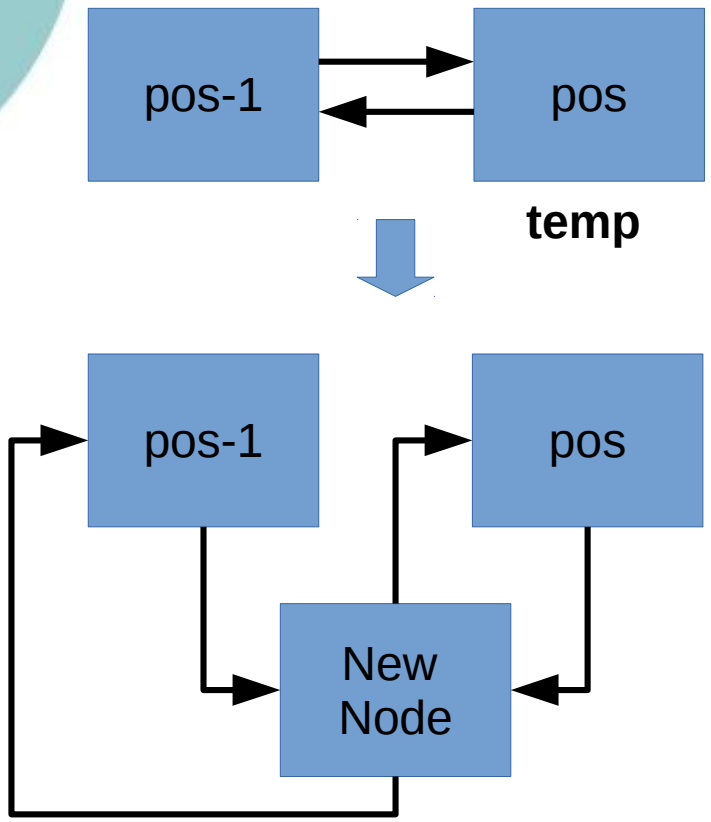
- Jika n =1, maka insert di sebelum head

```
if (pos == 1) //insert before Head
  {
    NewNode->next = Head;
    Head->prev = NewNode;
    Head = NewNode;
  }
```

- Jika list kosong: Head = NewNode = Tail;

Insert pada posisi n

- Jika n > 1 maka loop (iterasi) sampai pada posisi yang tepat




```
DLL_Node *temp=Head;//penunjuk node, mulai dari Head
int count=1;          //pencacah
while( count!=pos )
  {
    temp = temp->next; //maju satu node ke depan
    if( temp != 0 ) //null
      count++;
    else
      break;          //sampai di ujung list (Null)
  }
if ( count==pos )    //posisi ditemukan
  {
    NewNode->prev = temp->prev; //arahkan prev
    (temp->prev)->next = NewNode; //next dari prev
    temp->prev = NewNode;
    //tebak operasi apa yang kurang?
  }
```

Hapus node pada posisi tertentu

- Ada 3 kondisi
  - Hapus node pada posisi head
  - Hapus pada posisi tail (ujung)
  - Hapus pada posisi tengah
- Hapus pada posisi head:

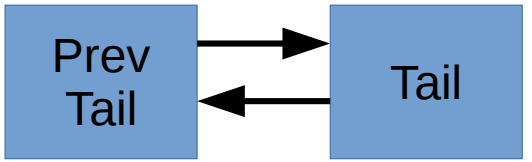
variabel pos menunjuk posisi node yang akan dihapus

```
DLL_Node *temp = Head;
if (pos==1)
  {
    Head = Head->next;
    Head->prev = 0;     //null
    delete temp;
  }
```

Hapus node pada posisi tertentu

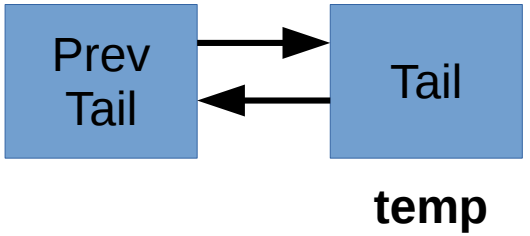
- Hapus pada posisi tail (ekor)

```
DLL_Node *temp = Head;
int count = 1;
while (count != pos)   //loop mencari posisi
  {
    temp = temp->next;
    if( temp!= 0 )
      count++;
    else
      break;
  }
if( count == pos )
  {
    if( temp==Tail ) //delete tail
      //kode delete
    else
      //delete posisi tengah
  }
```



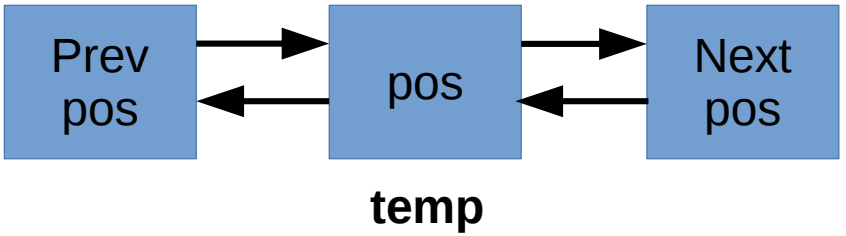
- Hapus pada posisi tengah

- Hapus pada tail




```
Tail = temp->prev;
(temp->prev)->next = 0;   //null
delete temp;
```

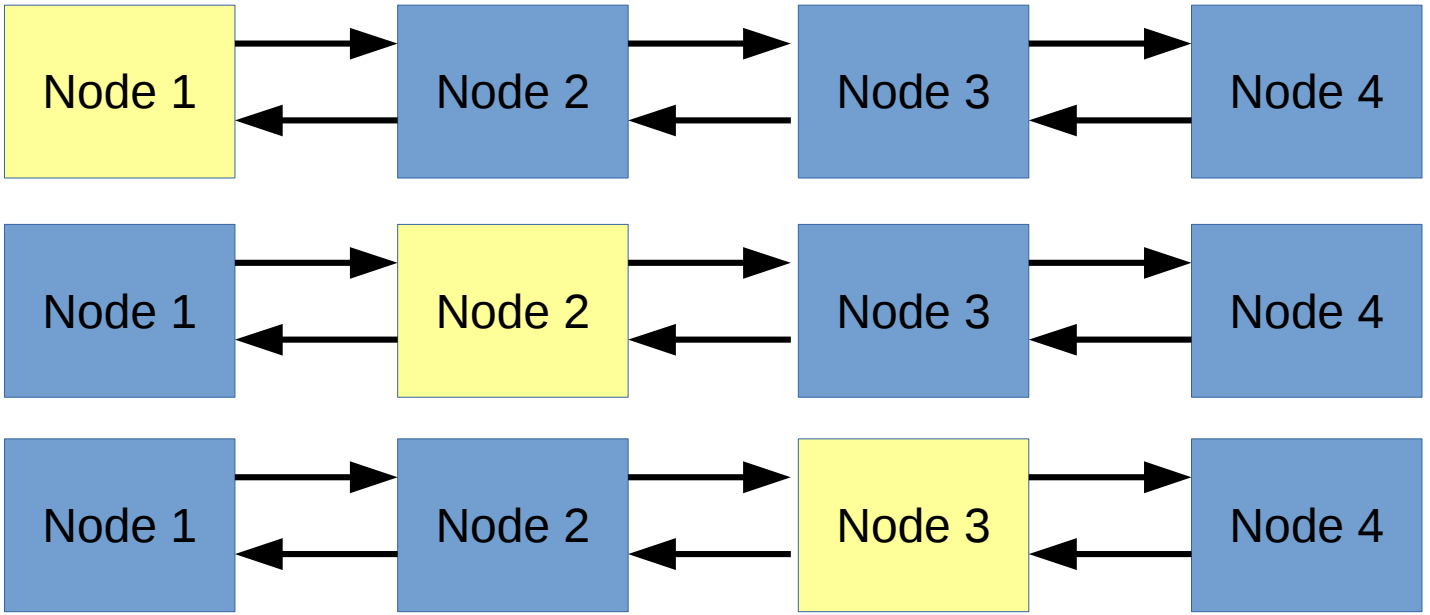
- Hapus posisi tengah




```
(temp->prev)->next = temp->next;
(temp->next)->prev = temp->prev;
delete temp;
```

Traverse (penelusuran)

- Penelusuran pada list berkait ganda tidak jauh berbeda dengan list berkait tunggal
- Hanya pada Double linked list, kita bisa bergerak dua arah yaitu depan dan belakang karena adanya dua pointer



Thank You
Obrigado
Merci
Asante
Multumesc
Spasibo
Maake
Grazie
Arigato
Matondo
Dank Je
Vinaka
Kiitos
Welalin
Juspaxar
Kia Ora
Mamana
Mochchakkeram
Nirringrazzjak
Ua Tsaug Rau Koj
Cam on ban
Chokrane
Salamat
Matur Nuwun
Terma Kasih
Raibh Maith Agat
Dankon