Data Structure and Object Oriented-Programming

Hafara Firdausi, M.Kom.

Departemen Teknologi Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

02. Advanced Linked List

Linked List Lanjutan

Circular Linked List

Circular Linked List

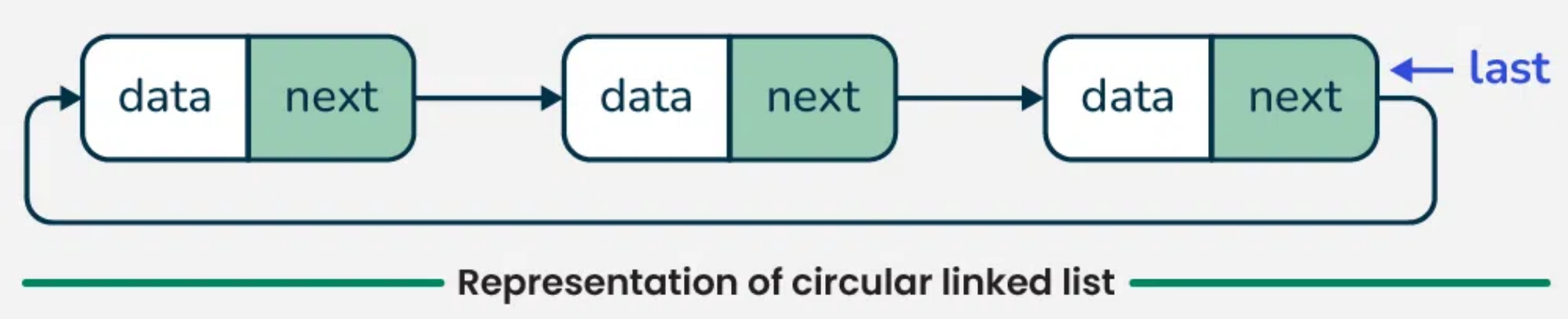
- Circular Linked List adalah sekumpulan node atau simpul yang saling terhubung **tanpa adanya nilai NULL pada salah satu nodenya** (membentuk circle)

- Perbedaannya terletak pada node terakhir:
  - Single Linked List node terakhir menunjuk ke NULL
  - Circular Single Linked List, **node terakhir akan menunjuk kembali ke node pertama,** membentuk struktur melingkar

- Jenis Circular Linked List:
  - ***Circular Singly Linked List***
  - ***Circular Doubly Linked List***

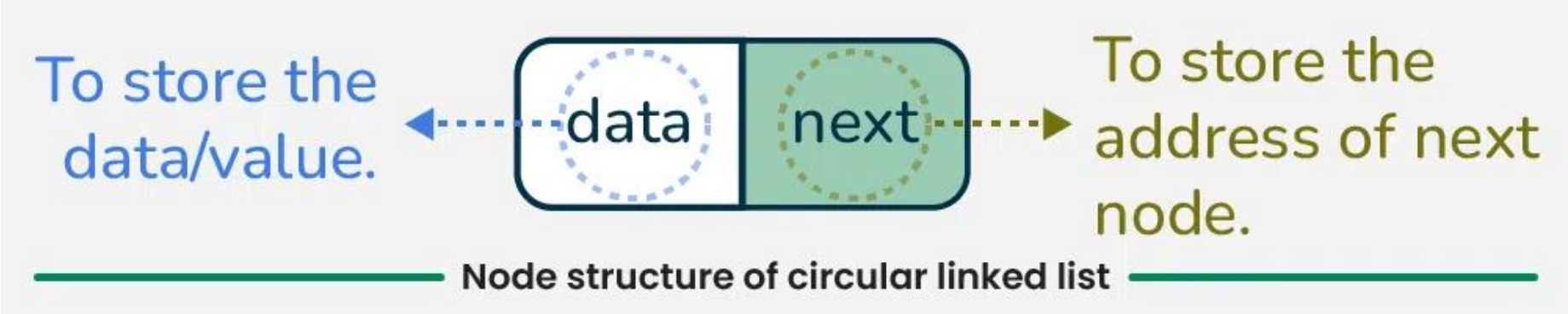
Circular Singly Linked List

Circular Singly Linked List



- Hanya punya 1 next pointer
- Next pointer dari Last / Tail Node menunjuk ke First / Head Node
- Hanya 1 arah

Circular Singly Linked List



- Struktur node pada Circular Single Linked List serupa dengan Single Linked List, yaitu terdiri dari **data** dan **pointer next**

Implementasi

Buat 2 Class:

1. **Node** → Menyimpan data dan pointer ke node berikutnya

2. **CircularSinglyLinkedList** → Mengelola operasi pada linked list (insert, display, dll)

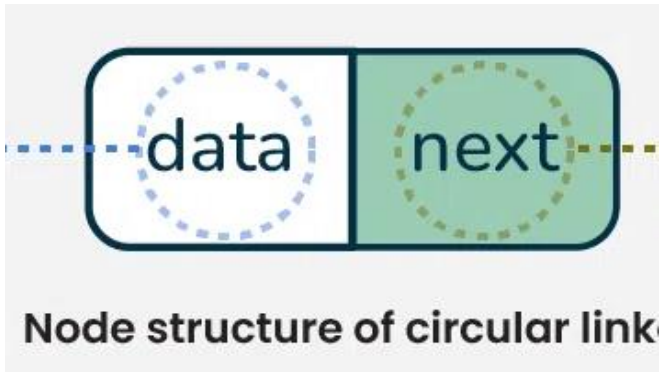
Implementasi

Class **Node**

- Property
  - **Data** – Untuk menyimpan data
  - **Next** – Untuk menyimpan pointer node selanjutnya. Tipe datanya objek kelas Node

- Method
  - **Node(int data)** – Constructor untuk instansiasi kelas dengan parameter data

```
class Node {
public:
   int data;
   Node* next;

   Node(int data) {
      this->data = data;
      this->next = nullptr;
   }
};
```



Node structure of circular link

Implementasi

Class **CircularSinglyLinkedList**

- Property
  - **Head** – Untuk menyimpan head / first node. **Tipe datanya objek kelas Node**

- Method
  - **CircularSinglyLinkedList()**– Default constructor untuk instansiasi kelas dengan memberikan default value NULL ke head

```
class CircularSinglyLinkedList {
private:
   Node* head;

public:
   CircularSinglyLinkedList() {
      head = nullptr;
   }

   void insert(int data) {
      ...
   }

        void delete(int data) {
      ...
   }
```

Implementasi

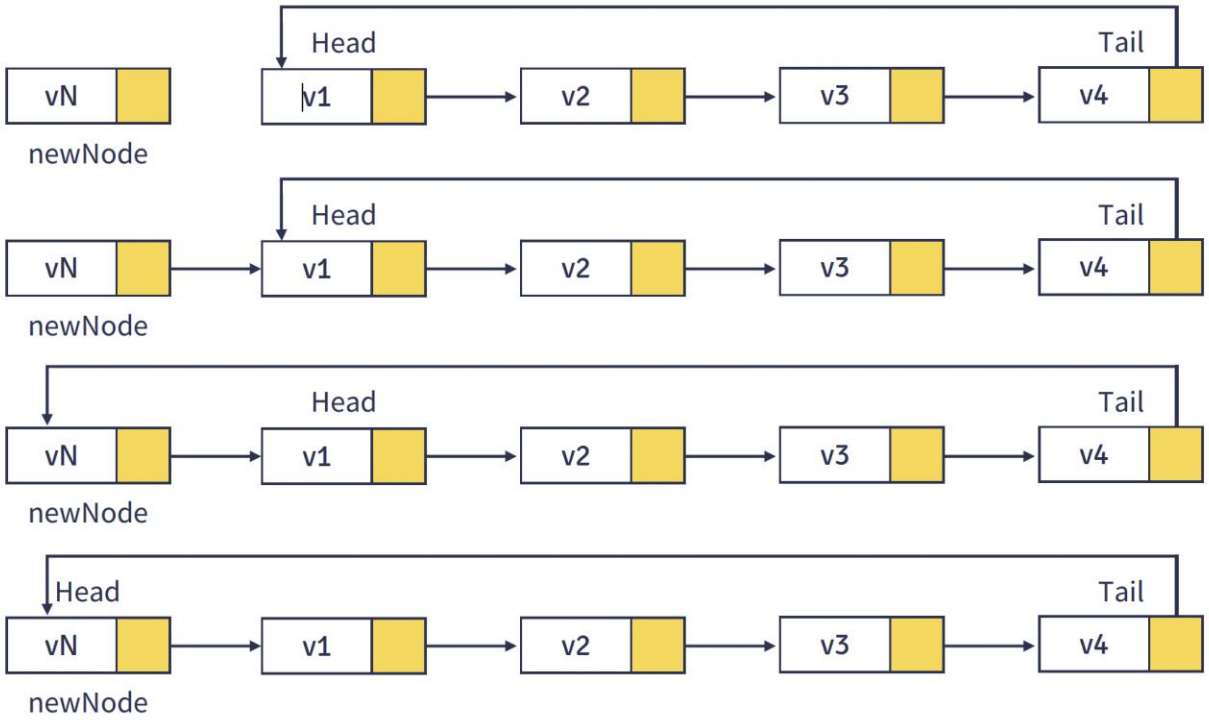
Class **CircularSinglyLinkedList**

- Method
  - **void insert(int data)** – Menambah node (ada 3 jenis):
    - Menambah di awal
    - Menambah di tengah
    - Menambah di akhir
  - **void delete(int data)** – Menghapus node (ada 3 jenis):
    - Menghapus di awal
    - Menghapus di tengah
    - Menghapus di akhir

```
class CircularSinglyLinkedList {
private:
   Node* head;

public:
   CircularSinglyLinkedList() {
      head = nullptr;
   }

   void insert(int data) {
      ...
   }

        void delete(int data) {
      ...
   }
```

Implementasi – Operasi

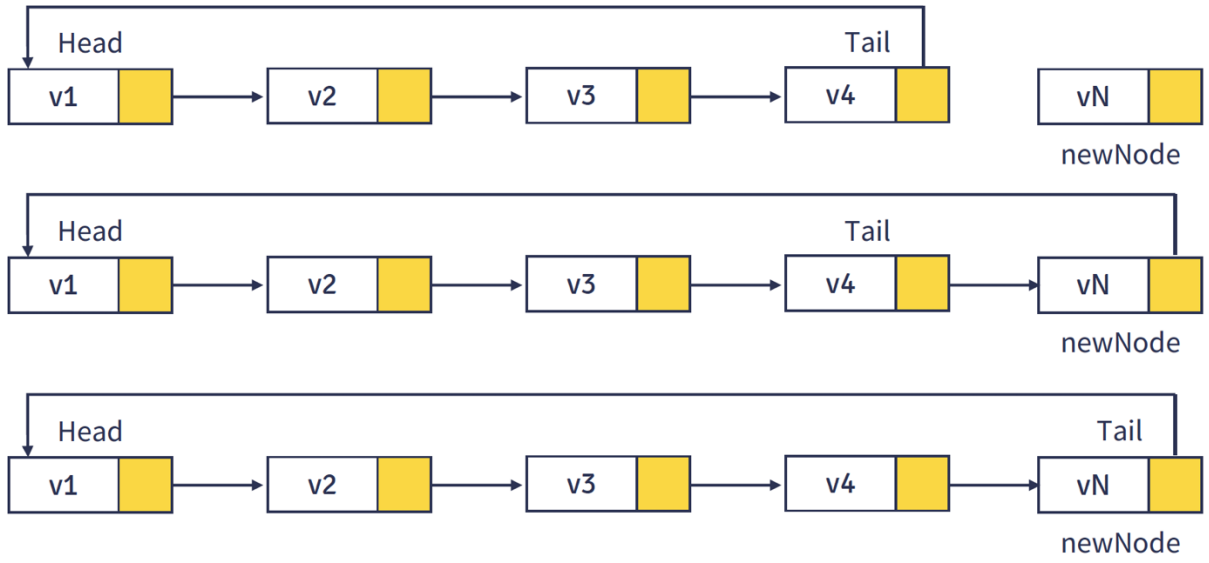
1. Added at the Beginning Node




```
void insertAtFirst(int data) {
    Node* newNode = new Node(data);
    if (!head) {
        head = newNode;
        head->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        newNode->next = head;
        temp->next = newNode;
        head = newNode;
    }
}
```

Implementasi – Operasi

2. Added at the Last Node

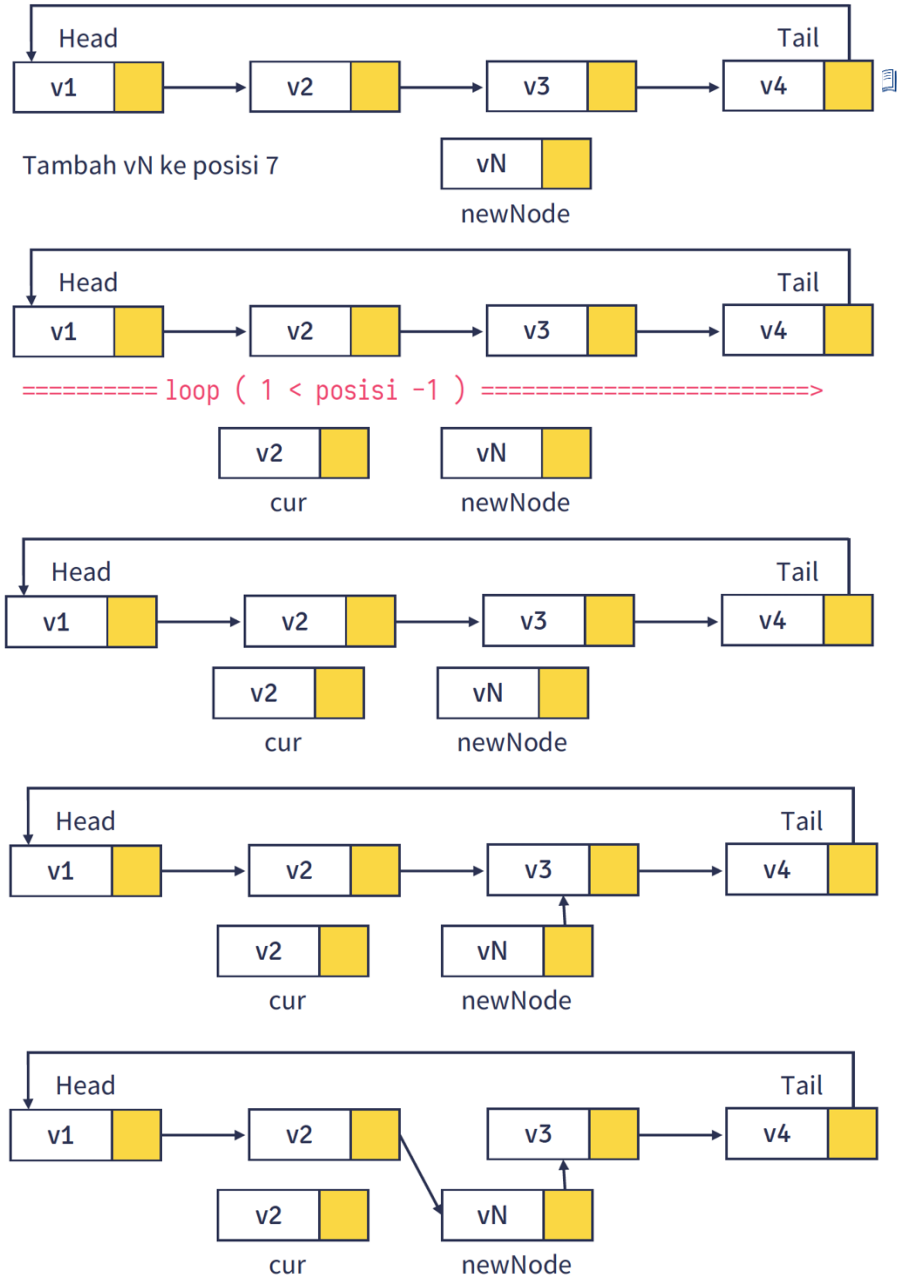



```
void insertAtLast(int data) {
    Node* newNode = new Node(data);
    if (!head) {
        head = newNode;
        head->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
```

Implementasi – Operasi

3. Added at the Middle Node

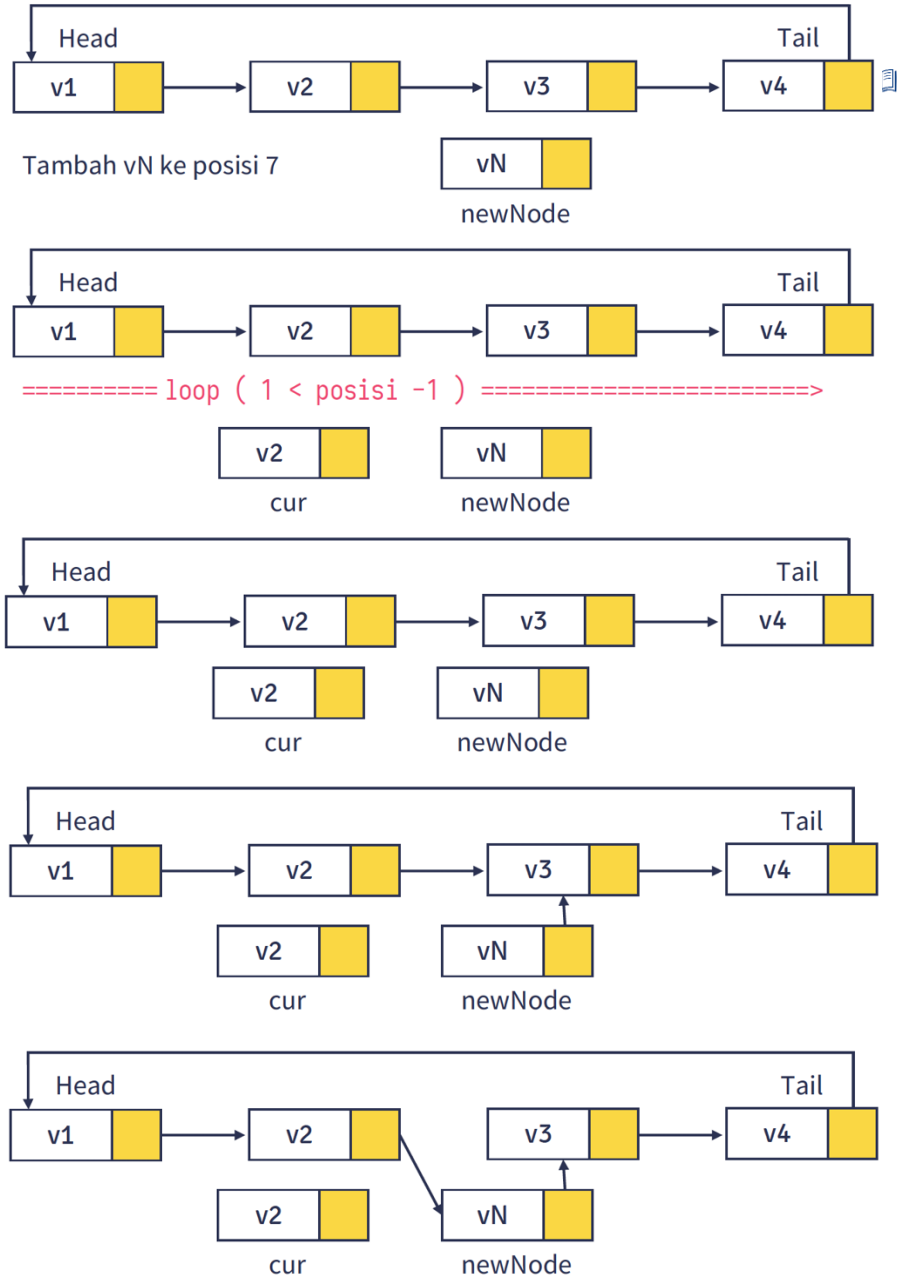
```
void insertAtMiddle(int data, int position) {
    if (!head || position == 1) {
        insertAtFirst(data);
        return;
    }

    Node *temp = head;
    int count = 1;
    while (count < position - 1 && temp->next != head) {
        temp = temp->next;
        count++;
    }
```



Implementasi – Operasi

3. Added at the Middle Node

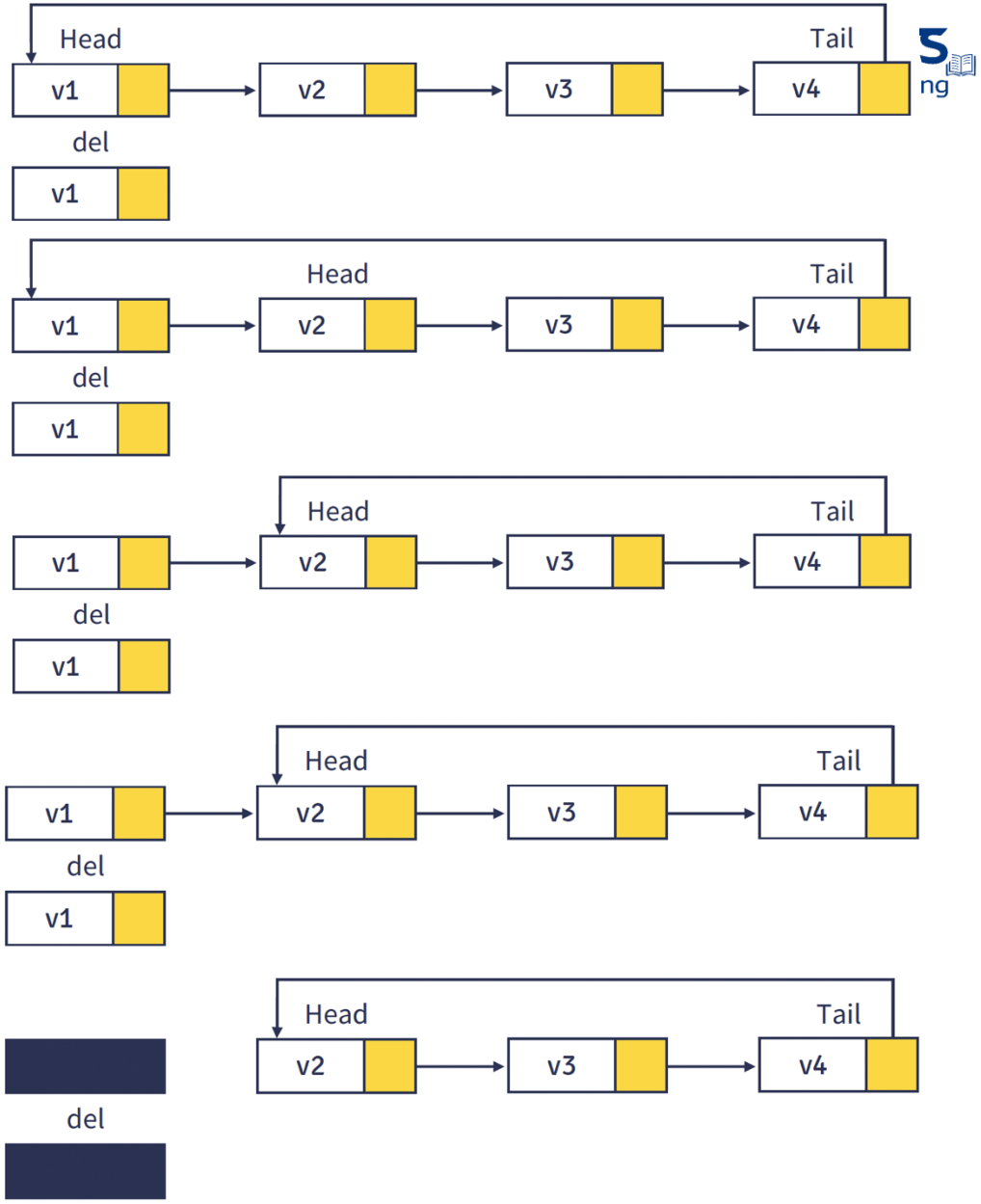
```
    if (count != position - 1 && temp->next == head) {
        cout << "Posisi melebihi jumlah node, menambahkan di akhir." << endl;
        insertAtLast(data);
        return;
    }

    Node *newNode = new Node(data);
    newNode->next = temp->next;
    temp->next = newNode;
}
```



Implementasi – Operasi

4. Delete the First Node

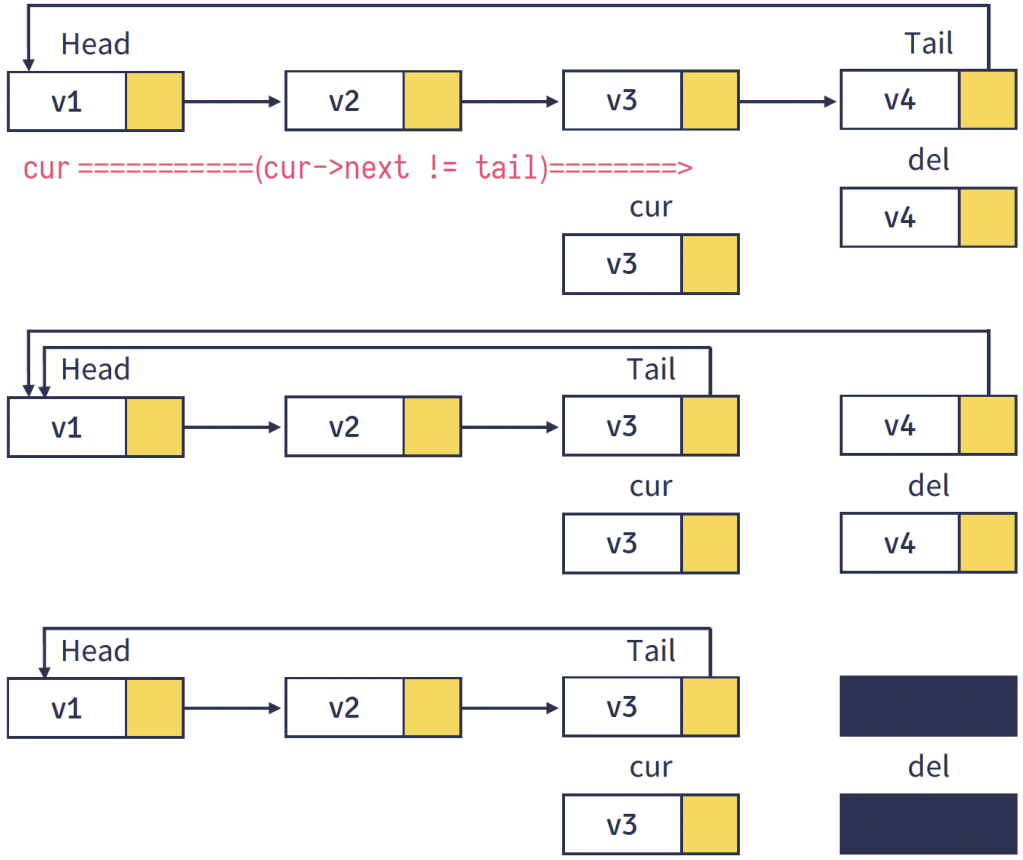
```
void deleteFirst() {
    if (!head) {
        cout << "List kosong." << endl;
        return;
    }

    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        Node* temp = head;
        Node* last = head;
        while (last->next != head) {
            last = last->next;
        }
        head = head->next;
        last->next = head;
        delete temp;
    }
}
```



Implementasi – Operasi

5. Delete the Last Node

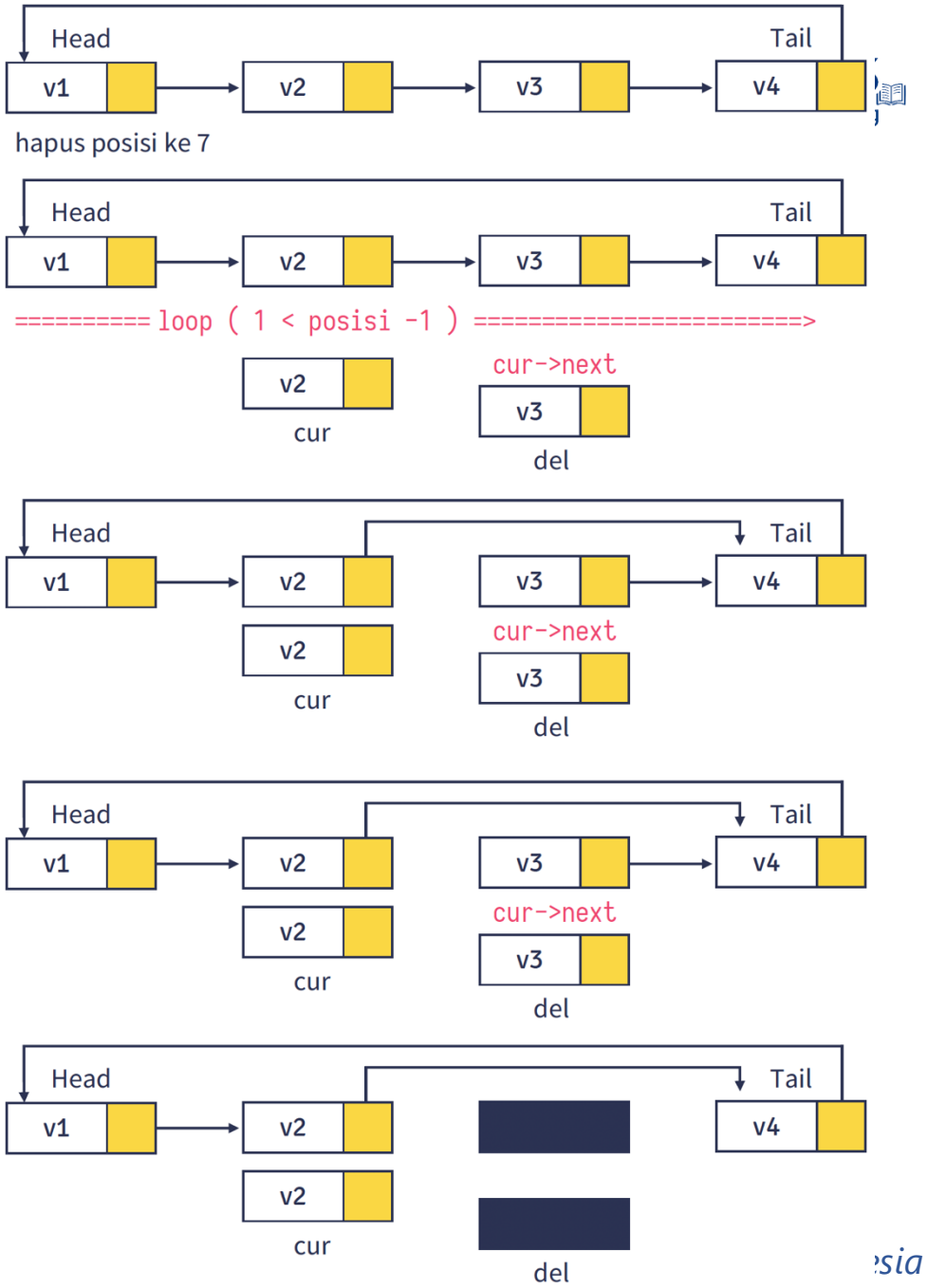
```
void deleteLast() {
    if (!head) {
        cout << "List kosong." << endl;
        return;
    }

    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        Node *temp = head;
        Node *prev = nullptr;
        while (temp->next != head) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = head;
        delete temp;
    }
}
```



Implementasi – Operasi

6. Delete the Middle Node

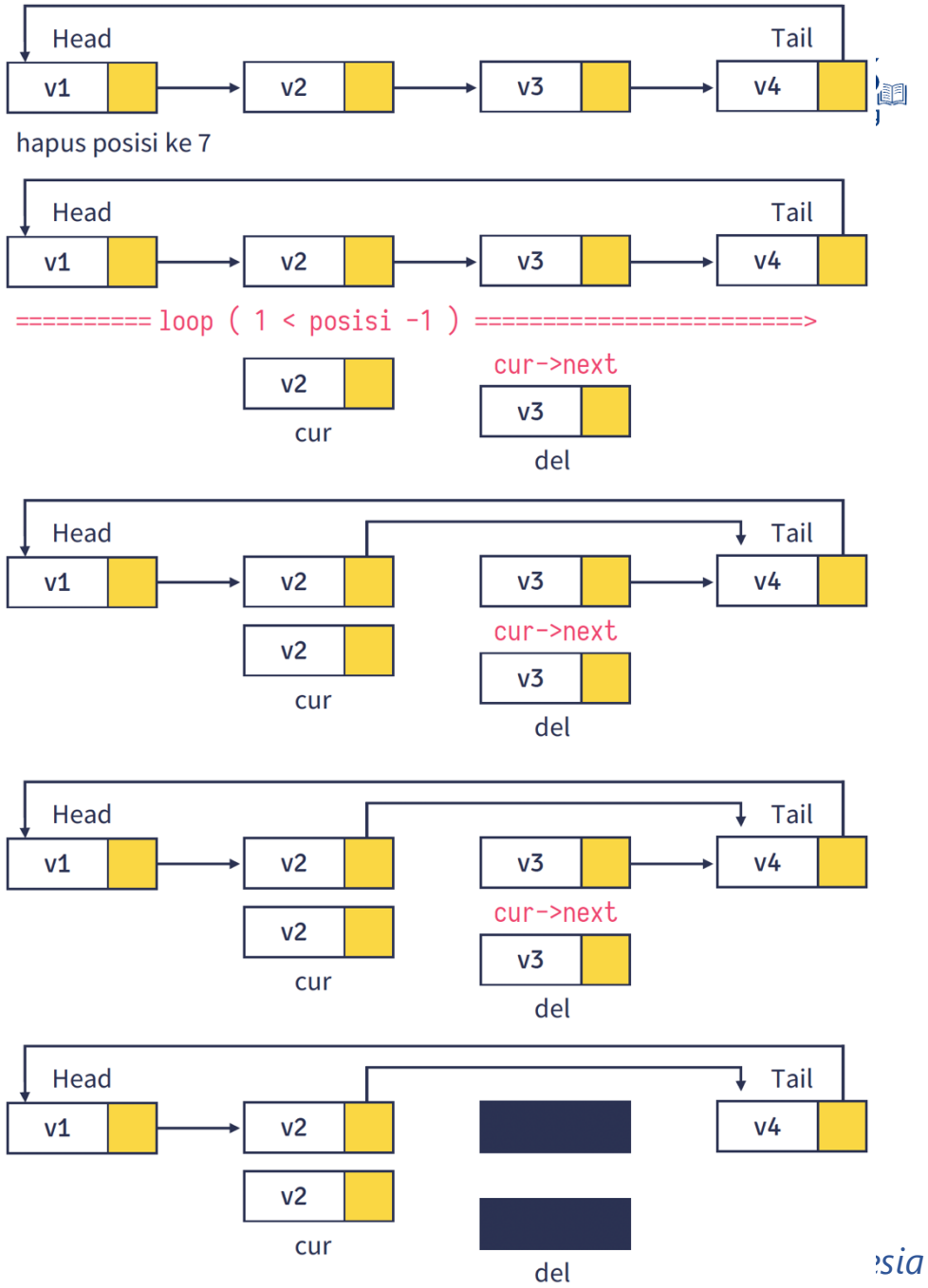
```
void deleteMiddle(int position) {
    if (!head) {
        cout << "List kosong." << endl;
        return;
    }

    if (position == 1) {
        deleteFirst();
        return;
    }

    Node *temp = head;
    Node *prev = nullptr;
    int count = 1;
    while (count < position && temp->next != head) {
        prev = temp;
        temp = temp->next;
        count++;
    }
```



Implementasi – Operasi

6. Delete the Middle Node

```
    if (count != position) {
        cout << "Posisi melebihi jumlah node dalam list." << endl;
        return;
    }

    prev->next = temp->next;
    delete temp;
}
```



Implementasi

Class **CircularSinglyLinkedList**

- Method
  - **void display()** – Menampilkan linked list dengan cara terus tracing temp node->next, hingga temp->next adalah head node
  - **~CircularSinglyLinkedList() –** Destructor.
    - Wajib ada yaa, untuk menghapus semua node saat objek dihapus
    - Menghindari kebocoran memori

```
void display() {
    if (!head) {
        cout << "List kosong." << endl;
        return;
    }
    Node *temp = head;
    do {
        cout << temp->data << " -> ";
        temp = temp->next;
    } while (temp != head);
    cout << "(kembali ke head)" << endl;
}

~CircularSinglyLinkedList() {
    if (!head) return;
    Node *temp = head;
    Node *nextNode;
    do {
        nextNode = temp->next;
        delete temp;
        temp = nextNode;
    } while (temp != head);
    head = nullptr;
}
```

Implementasi

Cara instansiasi kelas
CircularSinglyLinkedList

```
└─$ ./csll
10 -> 20 -> (kembali ke head)

Insert at Last
10 -> 20 -> 30 -> (kembali ke head)

Insert at First
40 -> 10 -> 20 -> 30 -> (kembali ke head)

Insert at Middle (posisi 3)
40 -> 10 -> 50 -> 20 -> 30 -> (kembali ke head)

Delete First
10 -> 50 -> 20 -> 30 -> (kembali ke head)

Delete Last
10 -> 50 -> 20 -> (kembali ke head)

Delete Middle (posisi 2)
10 -> 20 -> (kembali ke head)
```

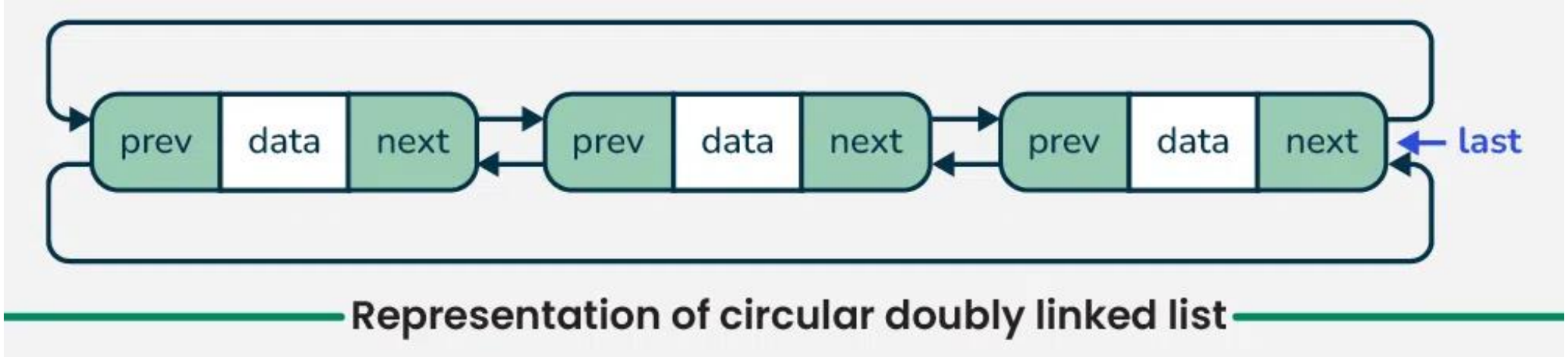
```
int main() {
    CircularSinglyLinkedList csll;
    csll.insertAtLast(10);
    csll.insertAtLast(20);
    csll.display();

    cout << "\nInsert at Last" << endl;
    csll.insertAtLast(30);
    csll.display();

    cout << "\nInsert at First" << endl;
    csll.insertAtFirst(40);
    csll.display();

    cout << "\nInsert at Middle (posisi 3)" << endl;
    csll.insertAtMiddle(50, 3);
    csll.display();

    cout << "\nDelete First" << endl;
    csll.deleteFirst();
    csll.display();

    cout << "\nDelete Last" << endl;
    csll.deleteLast();
    csll.display();

    cout << "\nDelete Middle (posisi 2)" << endl;
    csll.deleteMiddle(2);
    csll.display();

    return 0;
}
```

Kompleksitas

Operasi	Kompleksitas Waktu
Tambah di awal (insertAtFirst)	**O(1)**
Tambah di akhir (insertAtLast)	**O(n)** (mencari node terakhir)
Tambah di tengah (insertAtMiddle)	**O(n)**
Hapus dari awal (deleteFirst)	**O(1)**
Hapus dari akhir (deleteLast)	**O(n)** (mencari node sebelum terakhir)
Hapus di tengah (deleteMiddle)	**O(n)**
Pencarian Elemen (search)	**O(n)**
Traversal (Menampilkan data)	**O(n)**

Circular Doubly Linked List

Circular Doubly Linked List



Representation of circular doubly linked list

- Punya 2 pointer: Next dan Prev
- Next pointer dari Last / Tail Node menunjuk ke First / Head Node
- Prev pointer dari First / Head Node menunjuk ke Last / Tail Node
- 2 arah

Implementasi

Buat 2 Class:

1. **Node** → Menyimpan data dan pointer ke node berikutnya

2. **CircularDoublyLinkedList** → Mengelola operasi pada linked list (insert, display, dll)

Implementasi

Class **Node**

- Property
  - **Data** – Untuk menyimpan data
  - **Next** – Untuk menyimpan pointer node selanjutnya. Tipe datanya objek kelas Node
  - **Prev** – Untuk menyimpan pointer node sebelumnya. Tipe datanya objek kelas Node

```
// Kelas Node
class Node {
public:
   int data;
   Node* next;
   Node* prev;

   Node(int data) {
      this->data = data;
      this->next = this->prev = nullptr;
   }
};
```

Implementasi

Class **Node**

- Method
  - **Node(int data)** – Constructor untuk instansiasi kelas dengan parameter data

```
// Kelas Node
class Node {
public:
   int data;
   Node* next;
   Node* prev;

   Node(int data) {
      this->data = data;
      this->next = this->prev = nullptr;
   }
};
```

Implementasi

Class **CircularDoublyLinkedList**

- Property
  - **Head** – Untuk menyimpan head / first node. Tipe datanya objek kelas Node
- Method
  - **CircularDoublyLinkedList()**– Default constructor untuk instansiasi kelas dengan memberikan default value NULL ke head

```
// Kelas Circular Doubly Linked List
class CircularDoublyLinkedList {
private:
    Node* head;

public:
    CircularDoublyLinkedList() {
        head = nullptr;
    }

    void insert(int data) {
        ...
    }

        void delete(int data) {
        ...
    }
```

Implementasi

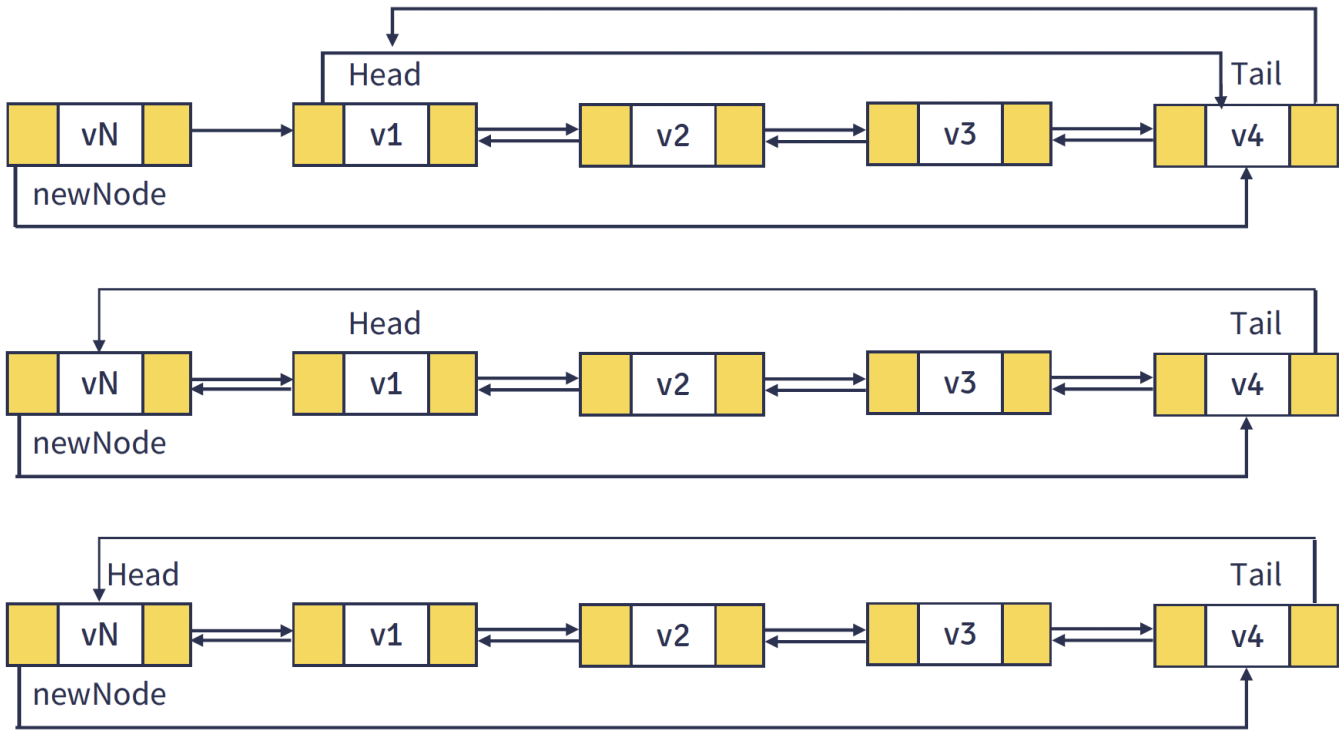
Class **CircularDoublyLinkedList**

- Method
  - **void insert(int data)** – Menambah node (ada 3 jenis):
    - Menambah di awal
    - Menambah di tengah
    - Menambah di akhir
  - **void delete(int data)** – Menghapus node (ada 3 jenis):
    - Menghapus di awal
    - Menghapus di tengah
    - Menghapus di akhir

```
// Kelas Circular Doubly Linked List
class CircularDoublyLinkedList {
private:
    Node* head;

public:
    CircularDoublyLinkedList() {
        head = nullptr;
    }

    void insert(int data) {
        ...
    }

        void delete(int data) {
        ...
    }
```

Implementasi – Operasi

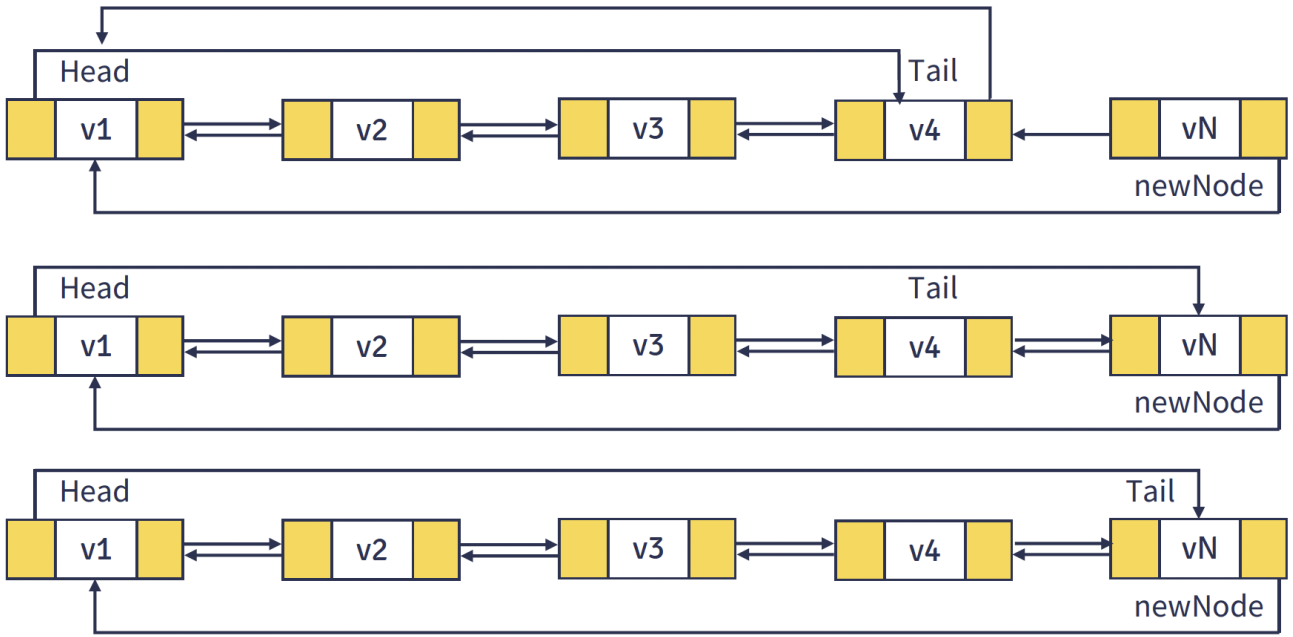
1. Added at the Beginning Node




```
void insertAtFirst(int data) {
    Node *newNode = new Node(data);
    if (!head) {
        head = newNode;
        head->next = head;
        head->prev = head;
    } else {
        Node *tail = head->prev;
        newNode->next = head;
        newNode->prev = tail;
        head->prev = newNode;
        tail->next = newNode;
        head = newNode;
    }
}
```

Implementasi – Operasi

2. Added at the Last Node

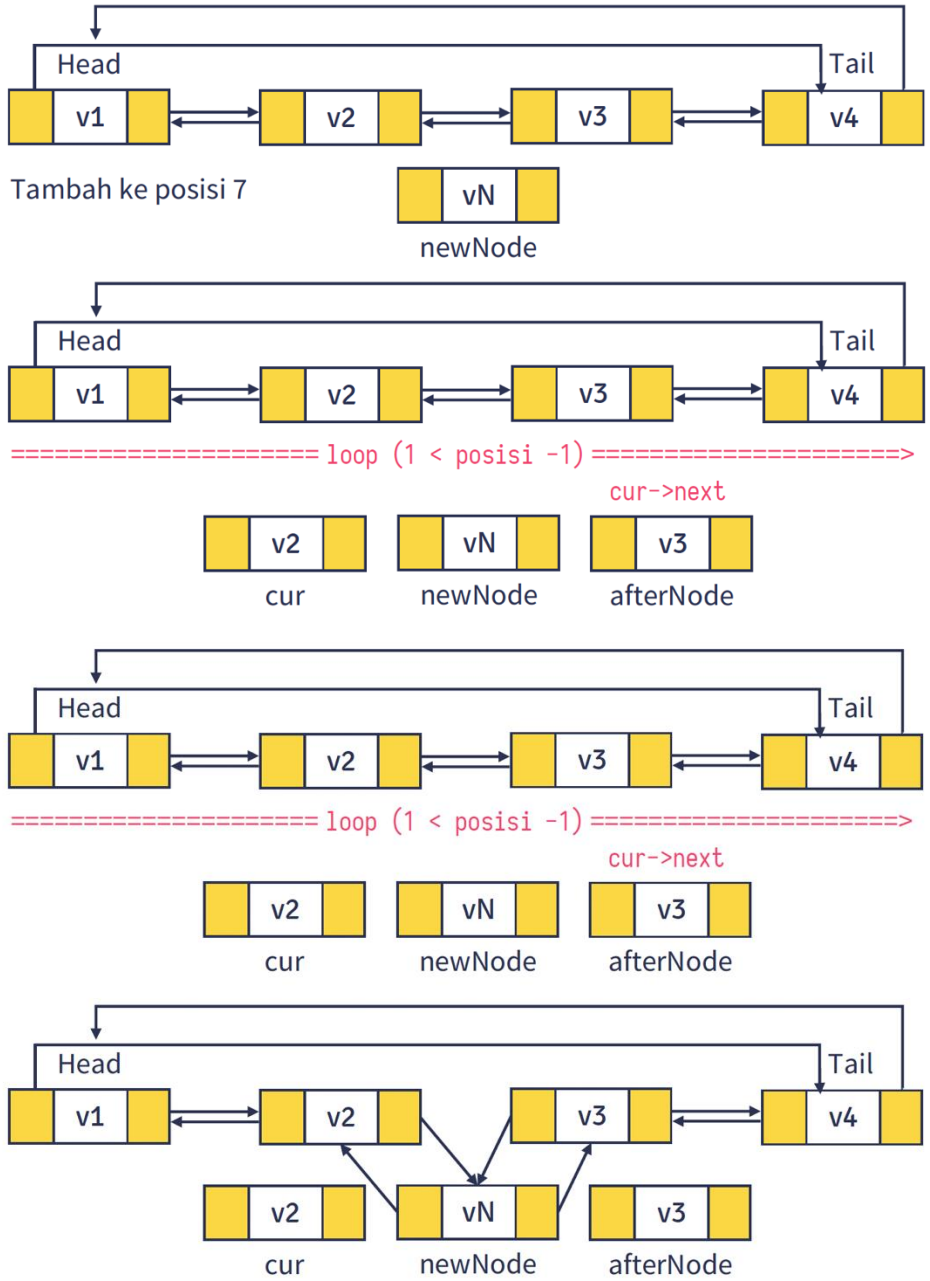



```
void insertAtLast(int data) {
    Node *newNode = new Node(data);
    if (!head) {
        head = newNode;
        head->next = head;
        head->prev = head;
    } else {
        Node *tail = head->prev;
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
    }
}
```

Implementasi – Operasi

3. Added at the Middle Node

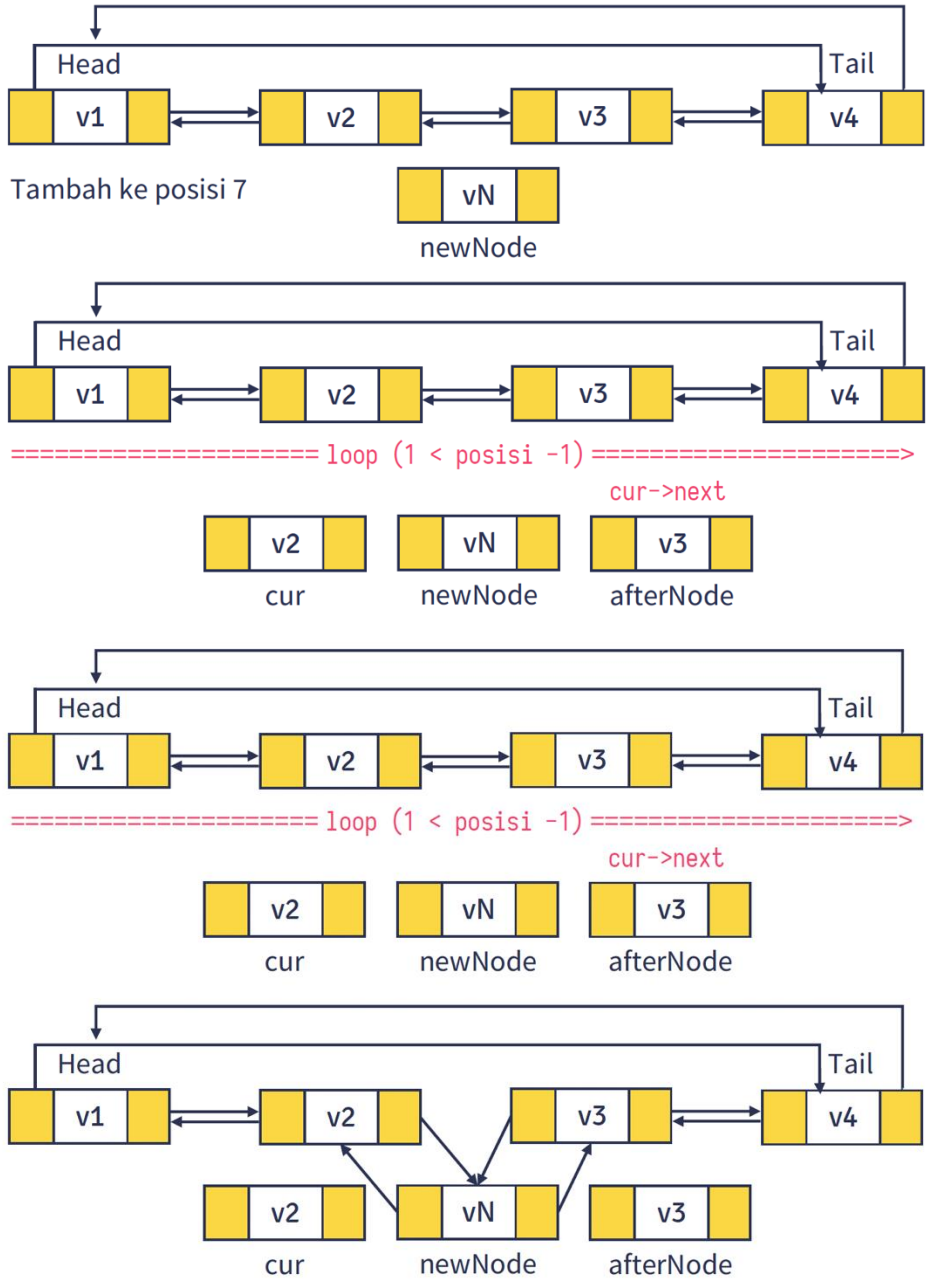
```
void insertAtMiddle(int data, int position) {
    if (!head || position == 1) {
        insertAtFirst(data);
        return;
    }

    Node *temp = head;
    int count = 1;

    while (count < position - 1 && temp->next != head) {
        temp = temp->next;
        count++;
    }
```



Implementasi – Operasi

3. Added at the Middle Node

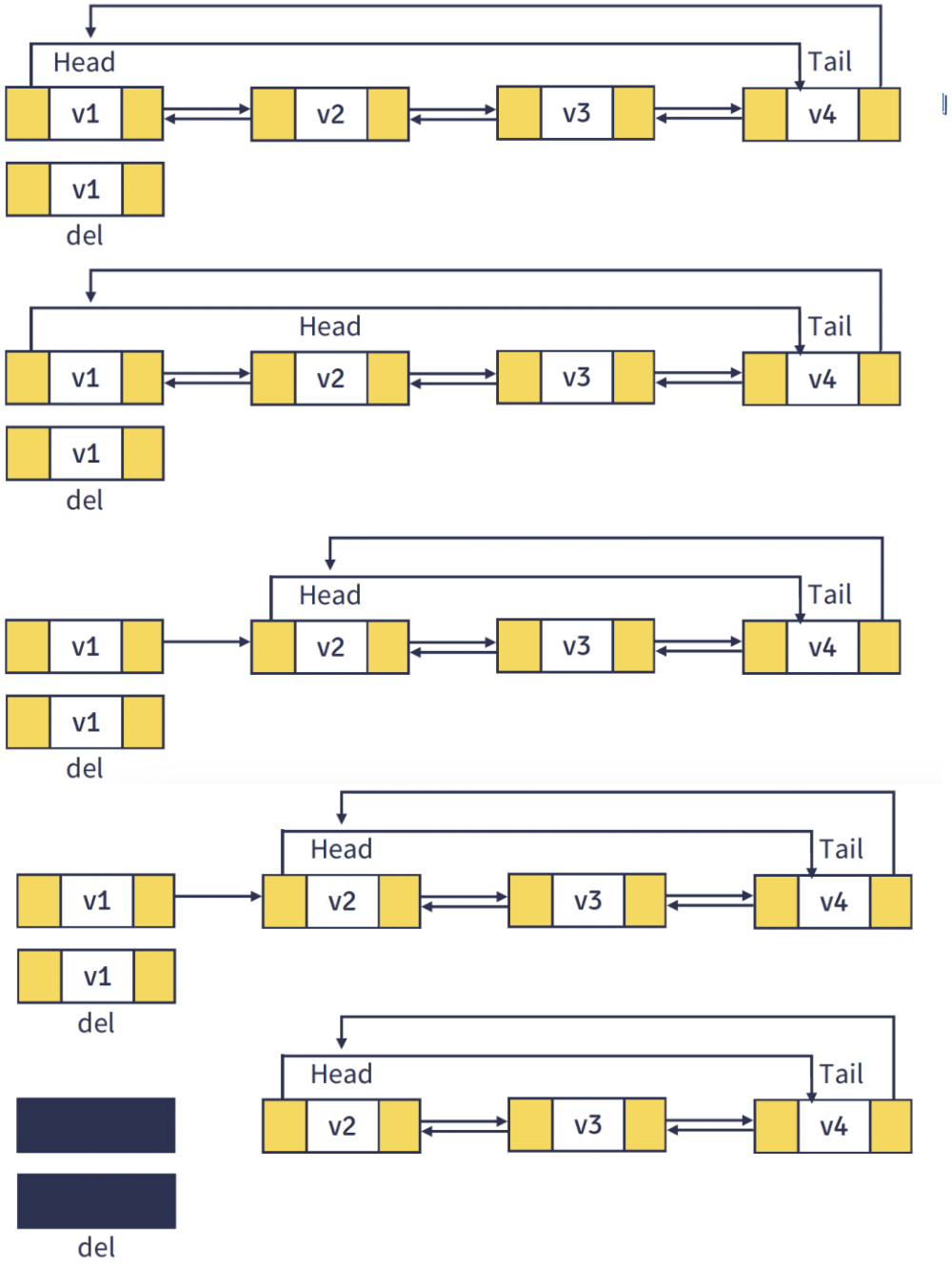
```
    if (count != position - 1 && temp->next == head) {
        cout << "Posisi melebihi jumlah node, menambahkan di akhir." << endl;
        insertAtLast(data);
        return;
    }

    Node *newNode = new Node(data);
    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
}
```



Implementasi – Operasi

4. Delete the First Node

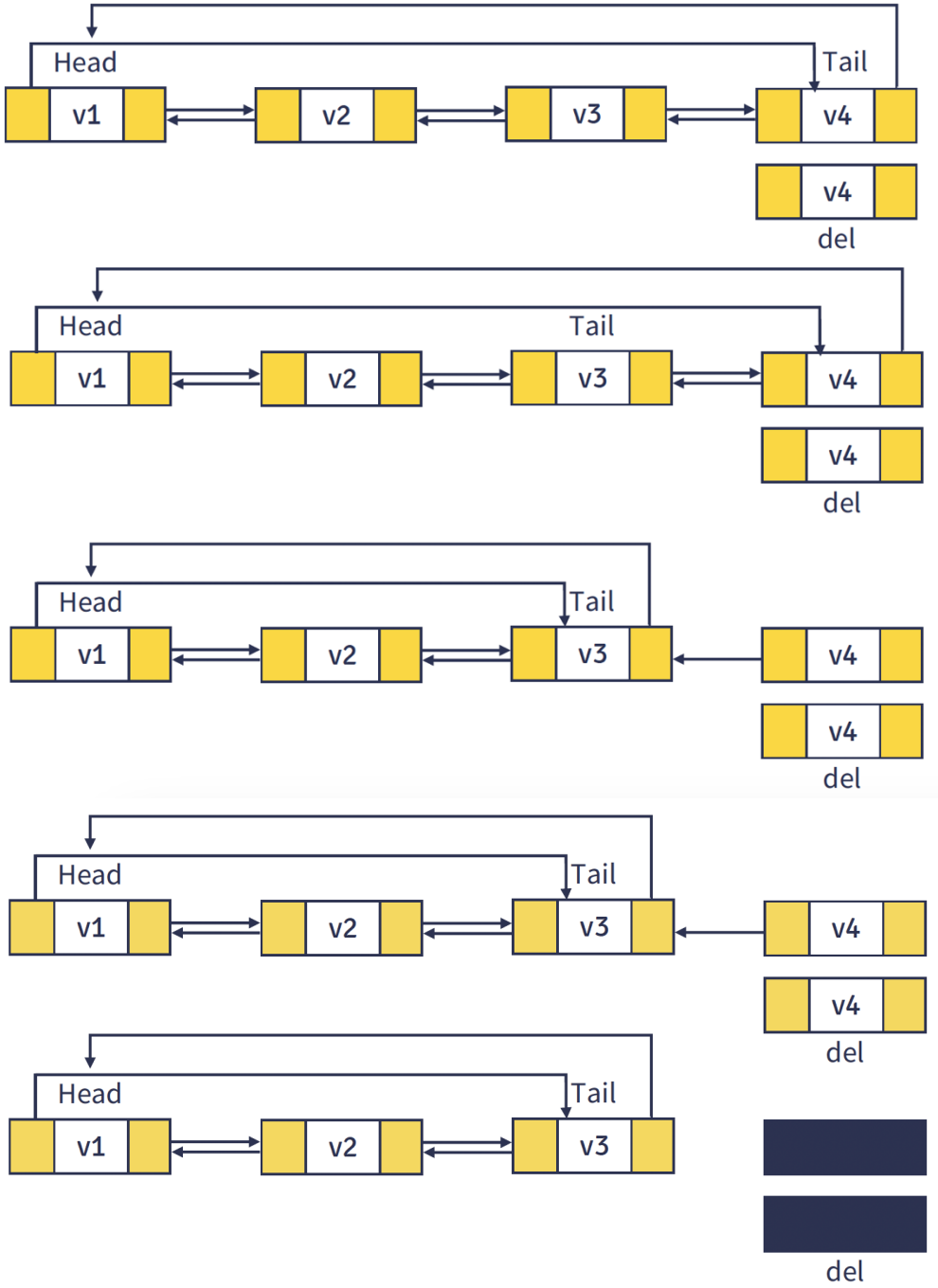
```
void deleteFirst() {
    if (!head) {
        cout << "List kosong!" << endl;
        return;
    }

    Node *tail = head->prev;
    Node *toDelete = head;

    if (head == head->next) {
        head = nullptr;
    } else {
        head = head->next;
        head->prev = tail;
        tail->next = head;
    }
    delete toDelete;
}
```



Implementasi – Operasi

5. Delete the Last Node

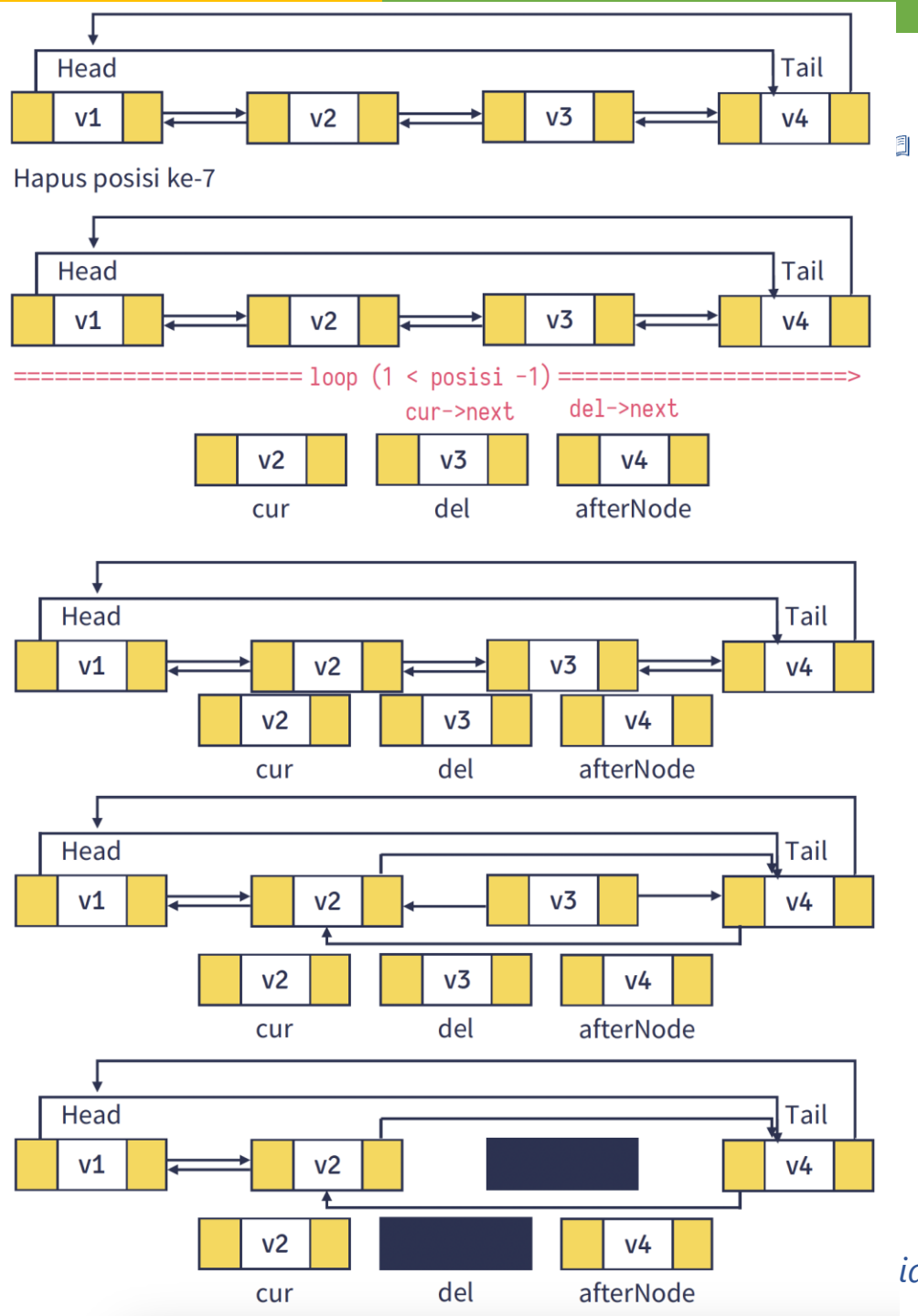
```
void deleteLast() {
    if (!head) {
        cout << "List kosong!" << endl;
        return;
    }

    Node *tail = head->prev;
    if (head == tail) {
        delete head;
        head = nullptr;
        return;
    }

    Node *newTail = tail->prev;
    newTail->next = head;
    head->prev = newTail;
    delete tail;
}
```



Implementasi – Operasi

6. Delete the Middle Node

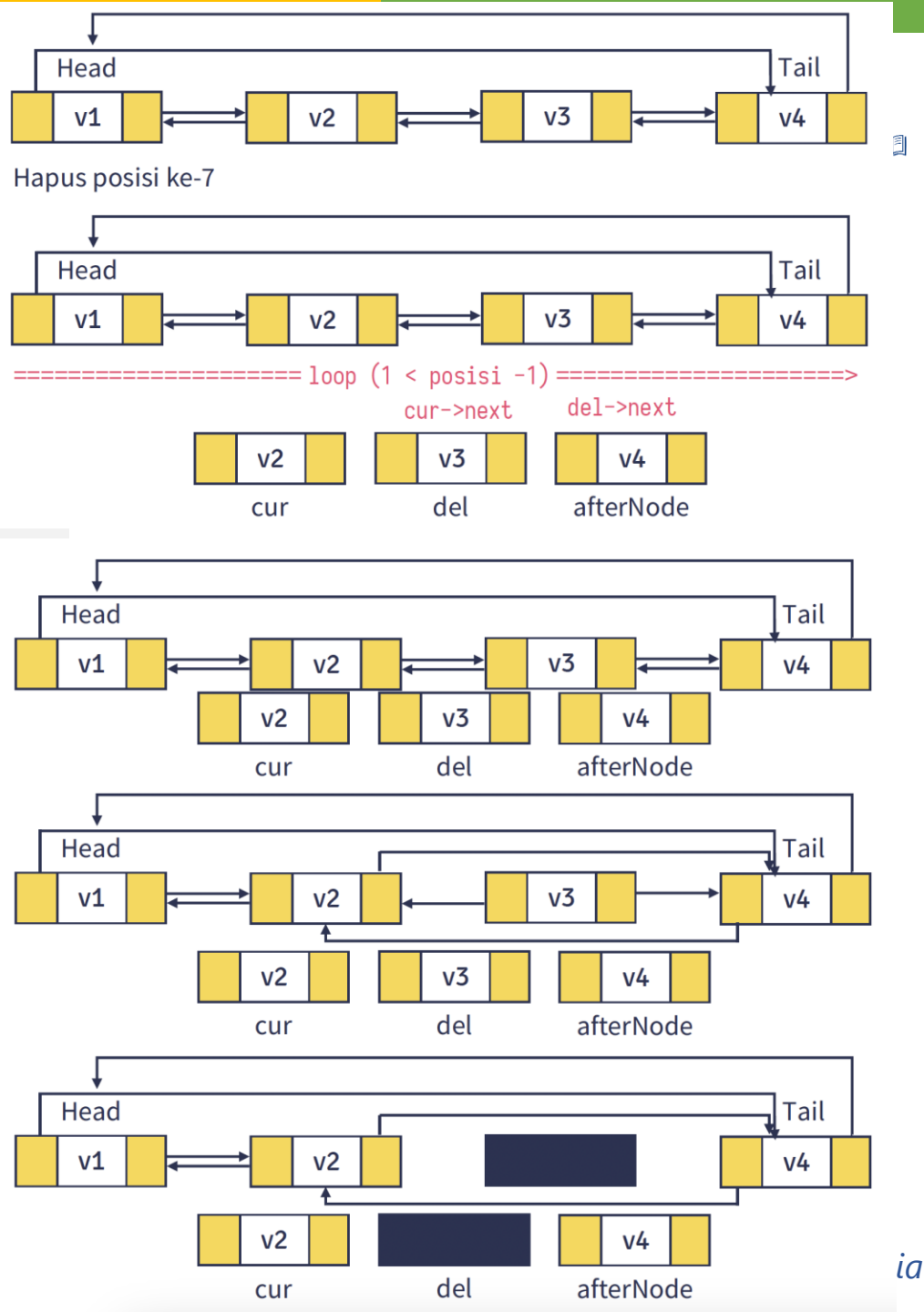
```
void deleteMiddle(int position) {
    if (!head) {
        cout << "List kosong!" << endl;
        return;
    }

    if (position == 1) {
        deleteFirst();
        return;
    }

    Node *temp = head;
    int count = 1;
```



Implementasi – Operasi

6. Delete the Middle Node

```
    while (count < position && temp->next != head) {
        temp = temp->next;
        count++;
    }

    if (count != position) {
        cout << "Posisi melebihi jumlah node dalam list!" << endl;
        return;
    }

    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;

    delete temp;
}
```



Implementasi

Class **CircularDoublyLinkedList**

- Method
  - **void displayForward()** – Menampilkan linked list dengan cara tracing dari head ke tail (depan ke belakang)
  - **void displayBackward()** – Menampilkan linked list dengan cara tracing dari tail ke head (belakang ke depan)

```
void displayForward() {
    if (!head) {
        cout << "List kosong!" << endl;
        return;
    }
    Node *temp = head;
    do {
        cout << temp->data << " <-> ";
        temp = temp->next;
    } while (temp != head);
    cout << "(kembali ke head)" << endl;
}

void displayBackward() {
    if (!head) {
        cout << "List kosong!" << endl;
        return;
    }
    Node *temp = head->prev;
    do {
        cout << temp->data << " <-> ";
        temp = temp->prev;
    } while (temp != head->prev);
    cout << "(kembali ke tail)" << endl;
}
```

Implementasi

Class **CircularDoublyLinkedList**

- Method
  - **~CircularDoublyLinkedList() –** Destructor.
    - Wajib ada yaa, untuk menghapus semua node saat objek dihapus
    - Menghindari kebocoran memori

```
~CircularDoublyLinkedList() {
    if (!head)
        return;

    Node *temp = head;
    Node *nextNode;

    do {
        nextNode = temp->next;
        delete temp;
        temp = nextNode;
    } while (temp != head);
    head = nullptr;
}
```

Implementasi

Cara instansiasi kelas
CircularDoublyLinkedList

```
└─$ ./cdll
10 <-> 20 <-> (kembali ke head)
20 <-> 10 <-> (kembali ke tail)

Insert at Last
10 <-> 20 <-> 30 <-> (kembali ke head)
30 <-> 20 <-> 10 <-> (kembali ke tail)

Insert at First
40 <-> 10 <-> 20 <-> 30 <-> (kembali ke head)
30 <-> 20 <-> 10 <-> 40 <-> (kembali ke tail)

Insert at Middle (posisi 3)
40 <-> 10 <-> 50 <-> 20 <-> 30 <-> (kembali ke head)
30 <-> 20 <-> 50 <-> 10 <-> 40 <-> (kembali ke tail)

Delete First
10 <-> 50 <-> 20 <-> 30 <-> (kembali ke head)
30 <-> 20 <-> 50 <-> 10 <-> (kembali ke tail)

Delete Last
10 <-> 50 <-> 20 <-> (kembali ke head)
20 <-> 50 <-> 10 <-> (kembali ke tail)

Delete Middle (posisi 2)
10 <-> 20 <-> (kembali ke head)
20 <-> 10 <-> (kembali ke tail)
```

```
int main() {
    CircularDoublyLinkedList cdll;

    cdll.insertAtLast(10);
    cdll.insertAtLast(20);
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nInsert at Last" << endl;
    cdll.insertAtLast(30);
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nInsert at First" << endl;
    cdll.insertAtFirst(40);
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nInsert at Middle (posisi 3)" << endl;
    cdll.insertAtMiddle(50, 3);
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nDelete First" << endl;
    cdll.deleteFirst();
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nDelete Last" << endl;
    cdll.deleteLast();
    cdll.displayForward();
    cdll.displayBackward();

    cout << "\nDelete Middle (posisi 2)" << endl;
    cdll.deleteMiddle(2);
    cdll.displayForward();
    cdll.displayBackward();

    return 0;
}
```

Kompleksitas

Operasi	Kompleksitas Waktu
Tambah di awal (insertAtBeginning)	**O(1)**
Tambah di akhir (insertAtEnd)*	**O(1)**
Tambah di tengah (insertAtMiddle)	**O(n)**
Hapus dari awal (deleteAtBeginning)	**O(1)**
Hapus dari akhir (deleteAtEnd)*	**O(1)**
Hapus di tengah (deleteAtMiddle)	**O(n)**
Pencarian Elemen (search)	**O(n)**
Traversal (Menampilkan data)	**O(n)**

**Lebih efisien dibanding Circular Singly Linked List*

Any Discussion?