Linked Lists

Hitesh Boinpally
Summer 2023

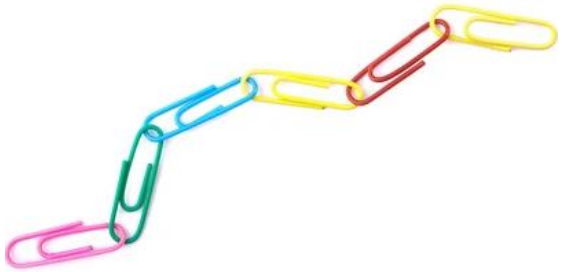
Agenda

- Linked Nodes review
- Linked Lists
- Practice
- Reminders

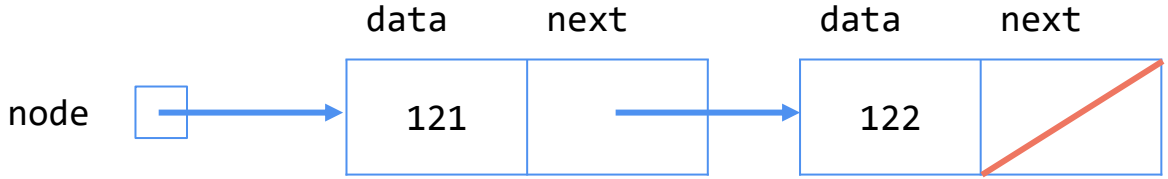
Agenda

- Linked Nodes review 
- Linked Lists
- Practice
- Reminders

Linked Nodes So Far



- Sequences of nodes connected together
- Using the `ListNode` class
- Traversals over these sequences with `while` loops



Agenda

- Linked Nodes review
- Linked Lists 
- Practice
- Reminders

Introducing the `LinkedIntList`!

- New collection named `LinkedIntList`

Introducing the `LinkedIntList`!

- New collection named `LinkedIntList`
- Same kinds of methods as the `ArrayIntList`
  - `add, add, get, indexOf, remove, size, toString`

Introducing the `LinkedIntList`!

- New collection named `LinkedIntList`
- Same kinds of methods as the `ArrayIntList`
  - `add, add, get, indexOf, remove, size, toString`
- Implemented with chain of linked nodes

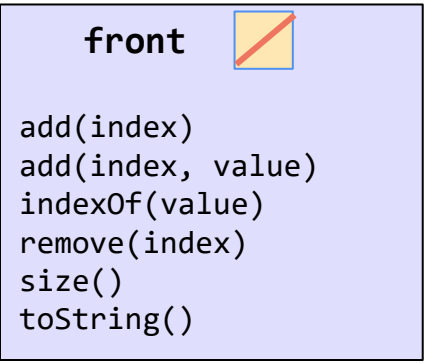
Introducing the `LinkedIntList`!

- New collection named `LinkedIntList`
- Same kinds of methods as the `ArrayIntList`
  - `add, add, get, indexOf, remove, size, toString`
- Implemented with chain of linked nodes
  - Keeps reference to its `front` as a field
  - `null` is the end of the list
  - If `front` is `null`, list is empty

Introducing the `LinkedIntList`!

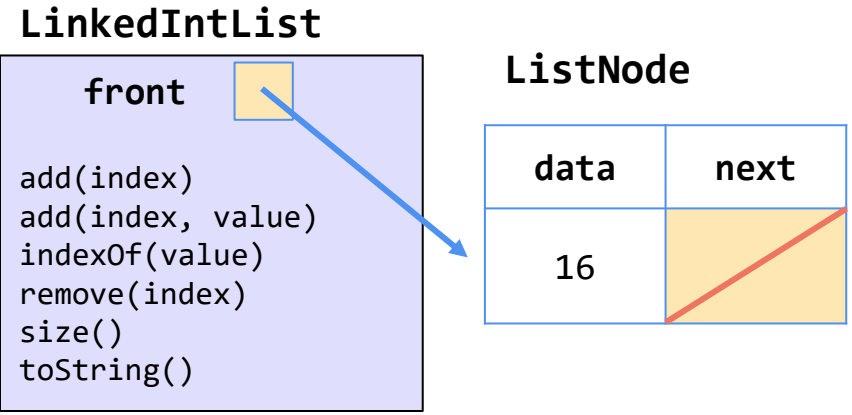
- Implemented with chain of linked nodes
  - Keeps reference to its `front` as a field
  - `null` is the end of the list; if `front` is `null`, list is empty

`LinkedIntList`



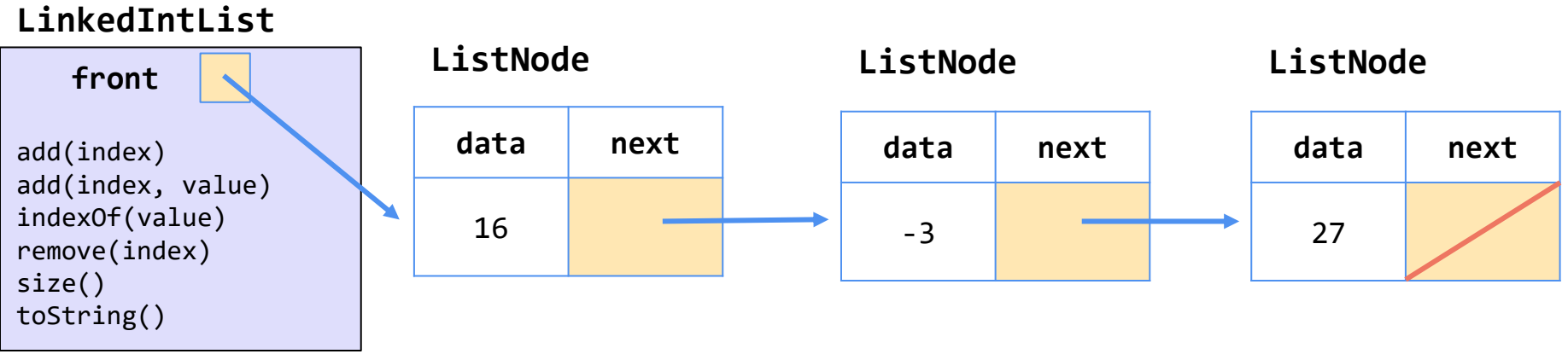
Introducing the `LinkedIntList`!

- Implemented with chain of linked nodes
  - Keeps reference to its `front` as a field
  - `null` is the end of the list; if `front` is `null`, list is empty



Introducing the `LinkedIntList`!

- Implemented with chain of linked nodes
  - Keeps reference to its `front` as a field
  - `null` is the end of the list; if `front` is `null`, list is empty

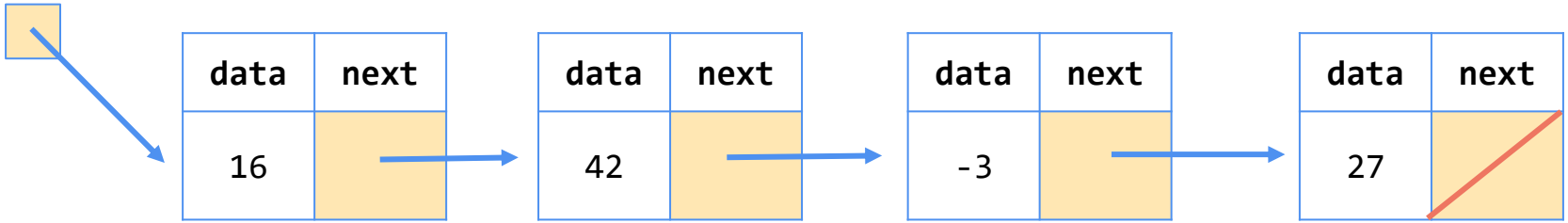


`remove(int value)` Visualization

⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

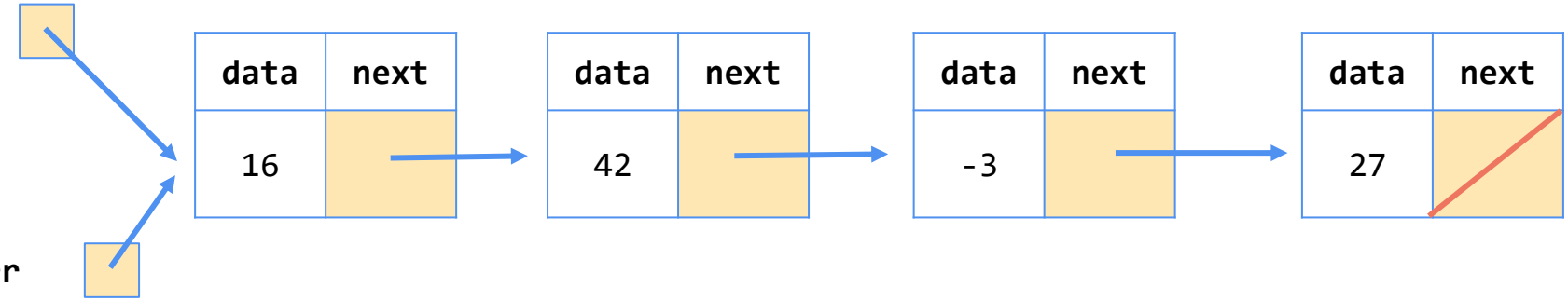
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

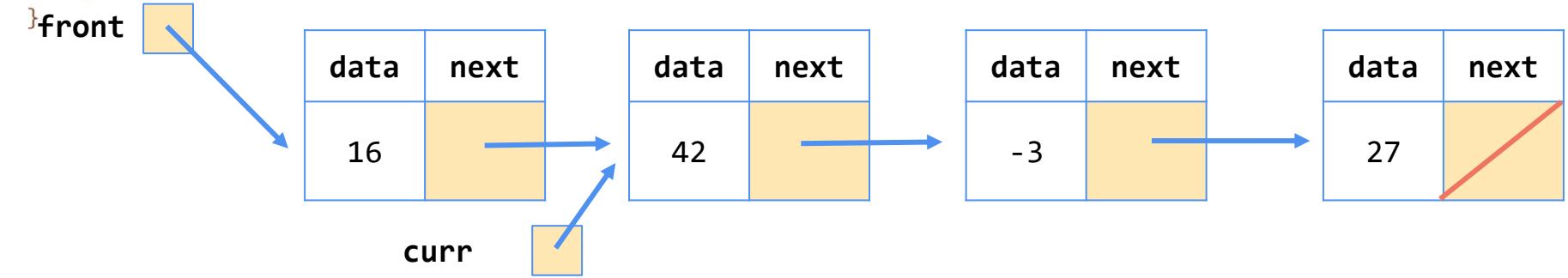
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

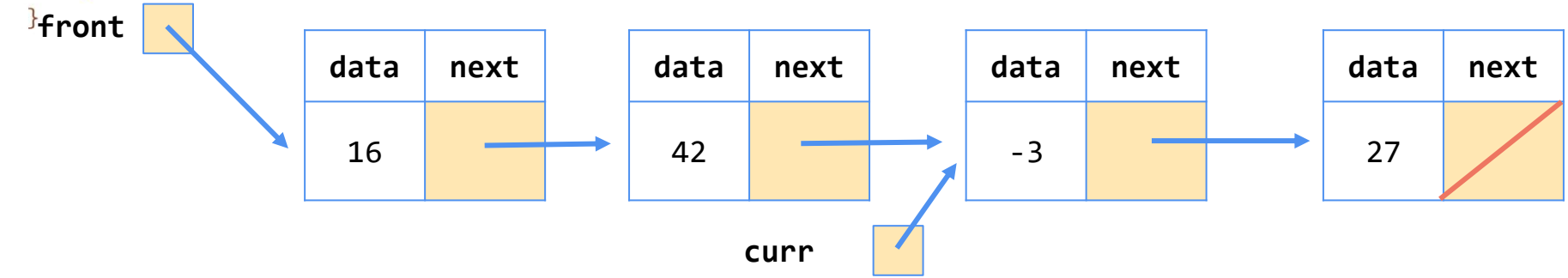
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

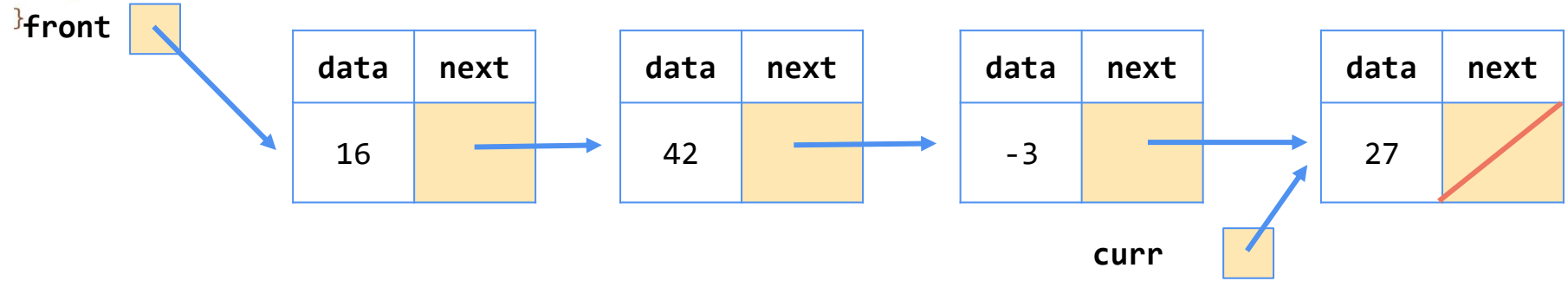
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

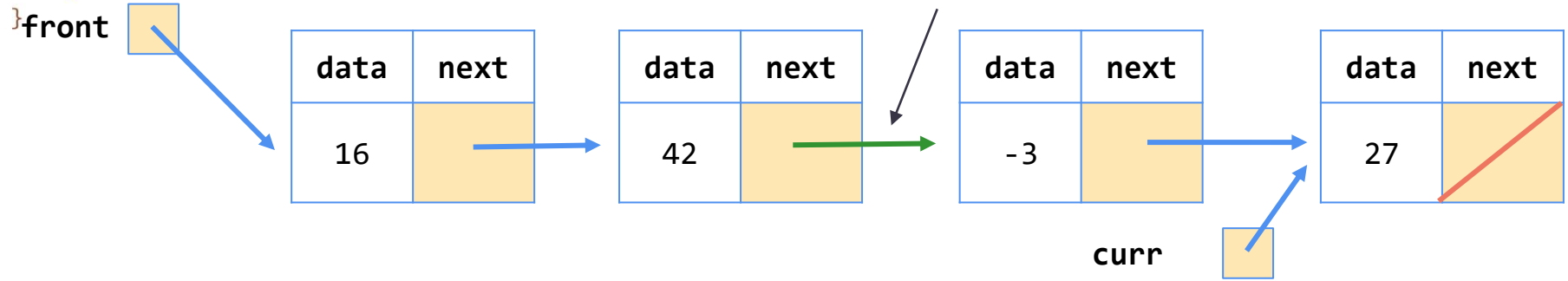
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

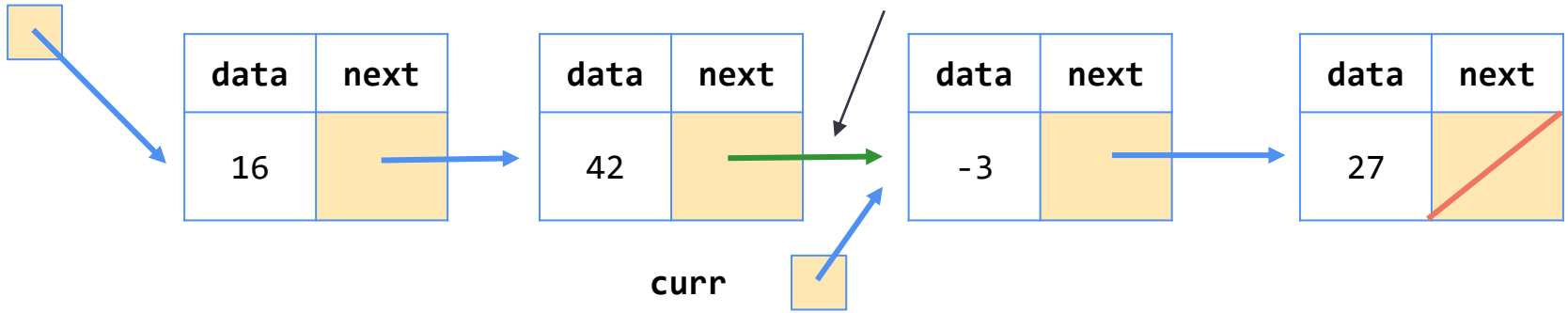
Client calls `list.remove(-3)`



⚠ Without "Stopping One Early" ⚠

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else {
            ListNode curr = front;
            while (curr != null && curr.data != value) {
                curr = curr.next;
            }
            if (curr != null) {
                curr = curr.next;
                size--;
            }
        }
    }
}
```

Client calls `list.remove(-3)`

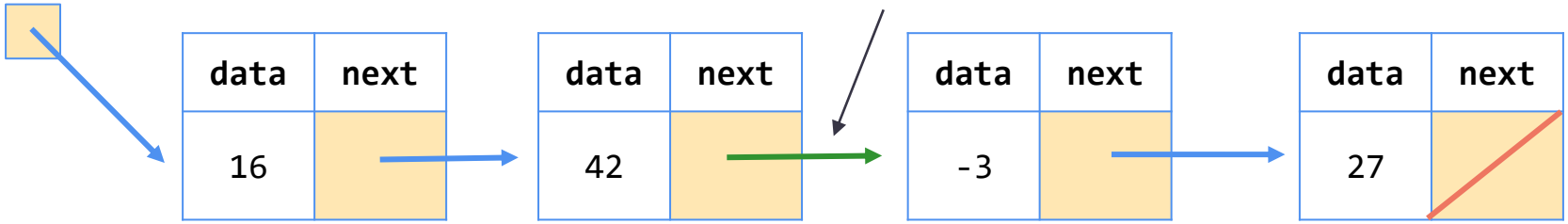


✅ With “Stopping One Early” ✅

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else
            ListNode curr = front;
            while (curr.next != null && curr.next.data != value) {
                curr = curr.next;
            }
            if (curr.next != null) {
                curr.next = curr.next.next;
                size--;
            }
        }
    }
}
```

Client calls `list.remove(-3)`

This is the reference we need to change!

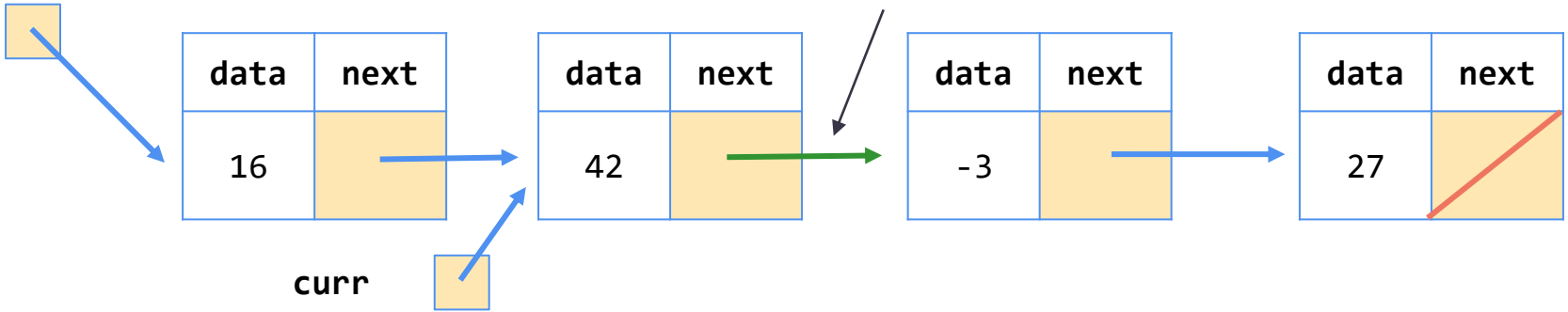


✅ With "Stopping One Early" ✅

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else
            ListNode curr = front;
            while (curr.next != null && curr.next.data != value) {
                curr = curr.next;
            }
            if (curr.next != null) {
                curr.next = curr.next.next;
                size--;
            }
        }
    }
}
```

Client calls `list.remove(-3)`

This is the reference we need to change!

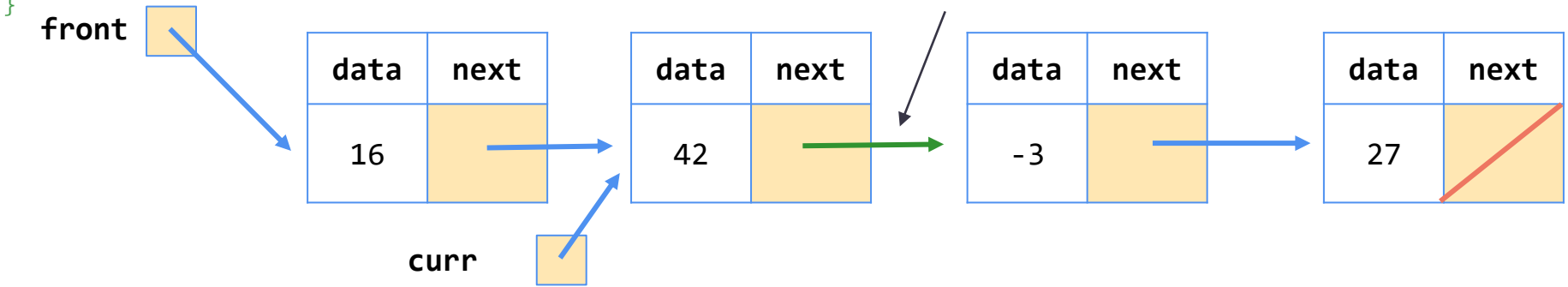


✅ With “Stopping One Early” ✅

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else
            ListNode curr = front;
            while (curr.next != null && curr.next.data != value) {
                curr = curr.next;
            }
            if (curr.next != null) {
                curr.next = curr.next.next;
                size--;
            }
        }
    }
}
```

Client calls `list.remove(-3)`

This is the reference we need to change!

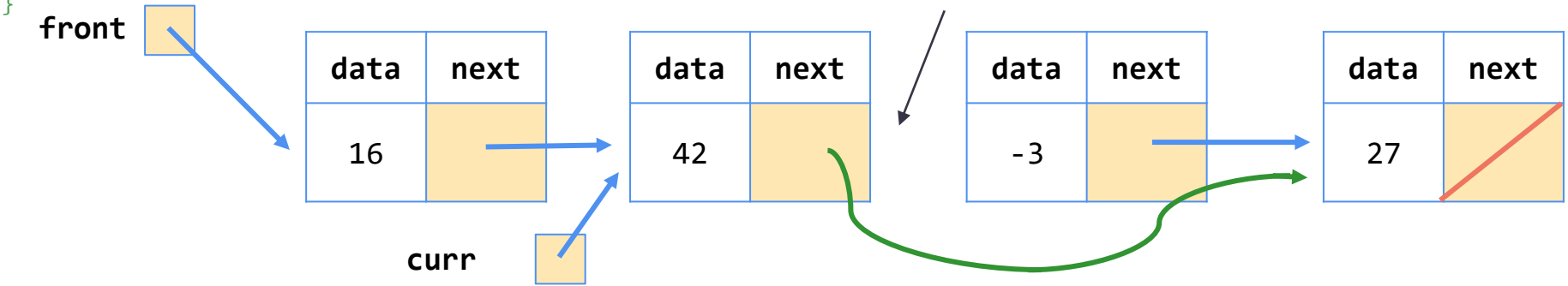


✅ With "Stopping One Early" ✅

```
public void remove(int value) {
    if (front != null) {
        // ... front case
        else
            ListNode curr = front;
            while (curr.next != null && curr.next.data != value) {
                curr = curr.next;
            }
            if (curr.next != null) {
                curr.next = curr.next.next;
                size--;
            }
        }
    }
}
```

Client calls `list.remove(-3)`

This is the reference we need to change!



Changing a list

- There are only two ways to change a linked list:
  - Change the value of `front` (modify the front of the list)
  - Change the value of `<node>.next` (modify middle or end of list to point somewhere else)
- Implications:
  - To add in the middle, need a reference to the *previous* node
  - Front is often a special case