# Single-dopple Linked List

EINFACHE-DOPPELTE VERKETTETE LISTE

# History

- Herbert A. Simon (1916-2001)
- Nobelpreisträger und Gewinner des Turing Awards.
- Erfinder der verketteten Liste (im Rahmen der IPL Sprache).

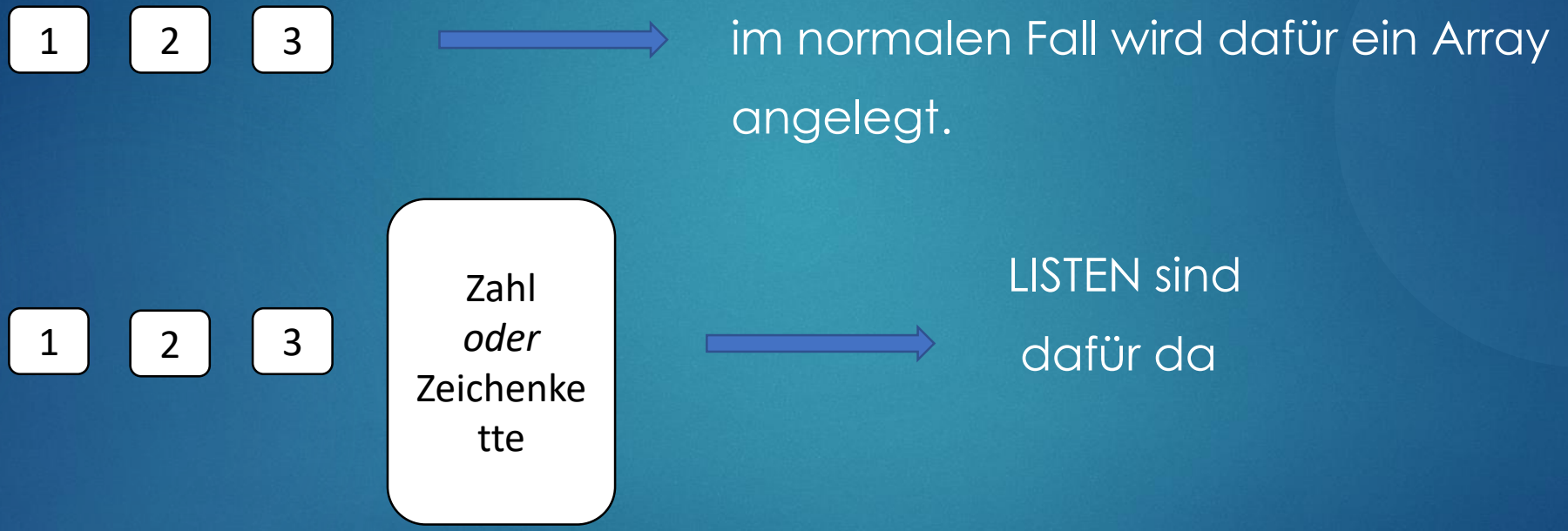
# Themen

- Verkettete Liste
- Liste – Stack
- Bauteile einer Liste
  - Knoten und Liste
  - Knoten und Liste erstellen
- Operationen
  - Elemente Hinzufügen(Rechts)
  - Liste löschen
  - Letztes Element löschen
  - Suchen
  - Das Letzte Element in single linked list finden

# Verkettete Liste

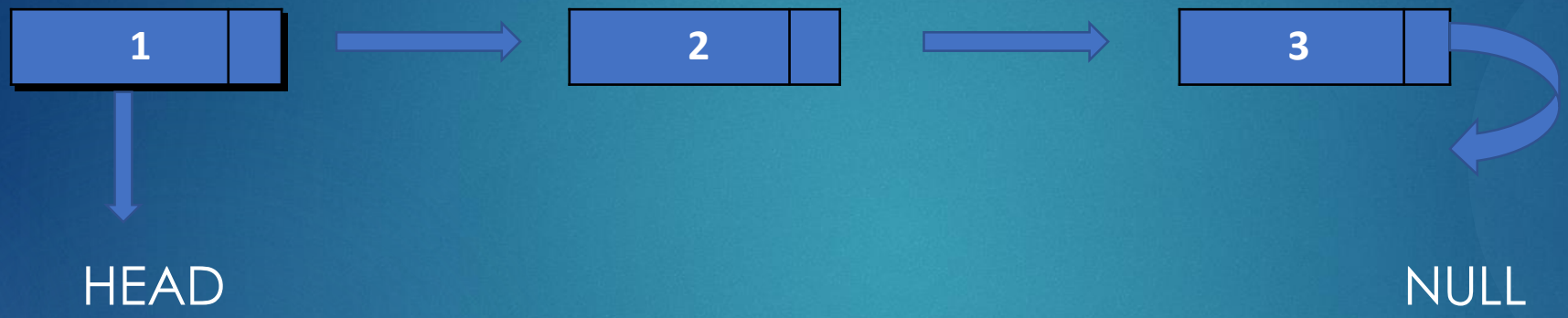
- Verkettete liste ist dynamische Datenstruktur.
- Vorteil:
  - ✓ Es wird nur der benötigte Speicherplatz reserviert.
  - ✓ Man kann den Speicher wieder freigeben.
  - ✓ Man ist in der Lage, Objekte als Pointer anzulegen

# Verkettete Liste

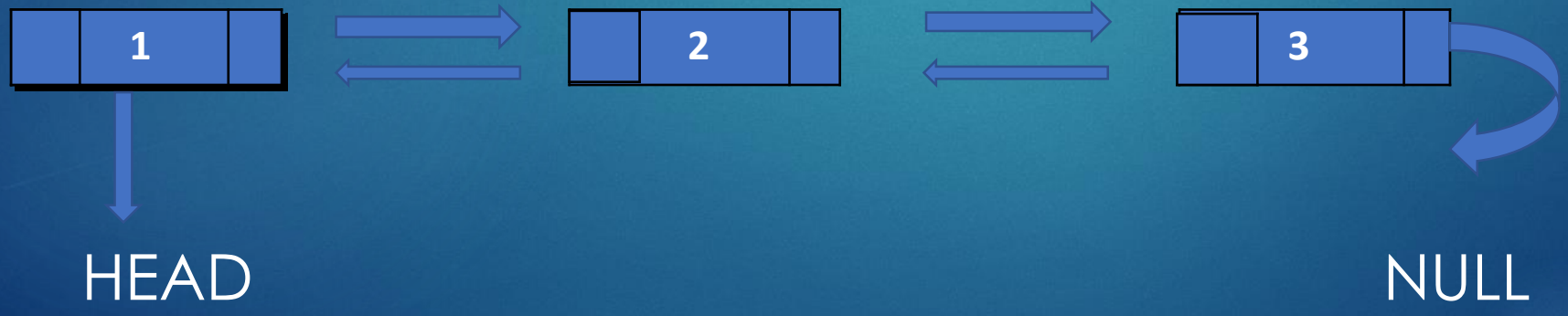


# Verkettete Listen

- Verkettete Listen unterscheiden sich in Zwei Arten
  - Einfache verkettete Listen (single linked list)




  - Doppelte verkettete Listen (double linked list)



# Einfach verkettete Liste

- Vorteil
  - Elemente können sehr schnell am Anfang der Liste eingefügt werden
  - Im Gegensatz zu einem Array ist das Einfügen problemlos möglich.

- Nachteil
  - Es ist aufwändig nach Daten zu suchen, Knoten einzufügen, zu löschen und die Liste zu sortieren.

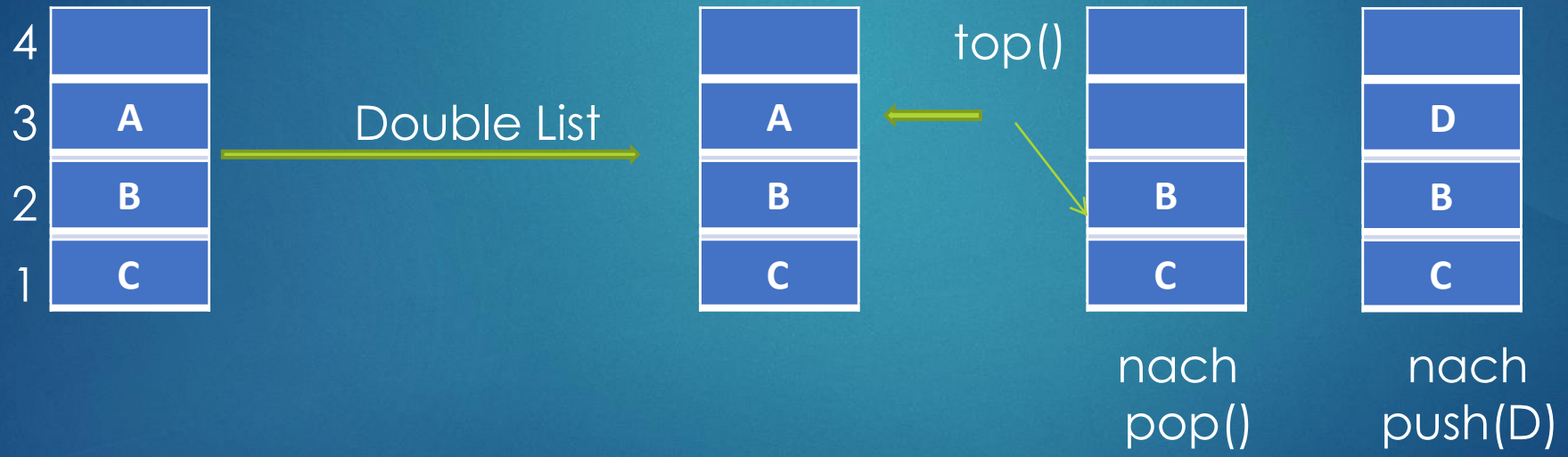
# Doppelt verkettete Liste

- **Vorteile**
  - Die Elemente in der zweiten Hälfte einer sortierten Liste sind schneller aufzufinden.
  - Schnelles Löschen und Einfügen von Elementen.
  - Das macht zum Beispiel das effiziente Sortieren der Liste möglich.
  - Über die Liste kann von hinten nach vorne iteriert werden.
- **Nachteile**
  - Höherer Speicherbedarf für die zusätzlichen Zeiger.

# Listen - Stacks

- Ein Stack ist ein LIFO-Speicher. Die Operationen eines Stacks sind top() , pop(), push(Element).

Angenommen:



# Bauteile einer Liste

# Knoten und Liste

- Knoten(Einfache verkettete Listen) besteht aus:
  - ✓ Zeiger : der auf das nächste Element verweist.
  - ✓ Zeiger: der daten trägt.
- Knoten(Doppelte verkettete Listen) besteht aus:
  - ✓ Zeiger : der auf das nächste Element verweist.
  - ✓ Zeiger : der auf das Letzte Element verweist.
  - ✓ Zeiger: der daten trägt.
- Liste besteht aus:
  - ✓ Head
  - ✓ Tail

# Knoten und Liste

```c
typedef struct single_node
{
    struct single_node* next;
    void* val;
} single_node_t;
```

```c
typedef struct double_node
{
    struct double_node* next;
    struct double_node* prev;
    void* val;
} double_node_t;
```

```c
typedef struct single_list
{
    struct single_node* head;
    struct single_node* tail;
    unsigned int len;
} single_list_t;
```

```c
typedef struct double_list
{
    struct double_node* head;
    struct double_node* tail;
    unsigned int len;
} double_list_t;
```

# Liste und Knoten erstellen

- Mit helfe der *malloc()* Funktion entwirft man die Liste und den Knoten.

```
single_list_t* create_list_S()
{
    single_list_t* new_list = (single_list_t*) malloc(LENGTH*sizeof(single_list_t));
    //checking if list contains a elements(node)
    if(!new_list)
        return NULL;

    new_list->head = NULL;
    new_list->tail = NULL;
    new_list->len = 0;

    return new_list;
}
```

# Liste und Knoten erstellen

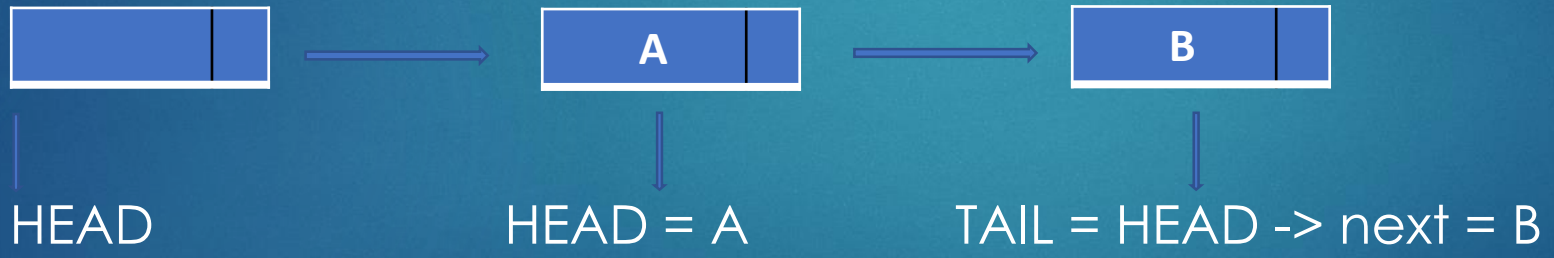
```
double_node_t* create_node_D(void* val)
{
    double_node_t* new_node = (double_node_t*) malloc(Length*sizeof(double_node_t));
    if(!new_node)
        return NULL;

    new_node->next = NULL;
    new_node->prev = NULL;
    new_node->val = val;

    return new_node;
}
```

# Operationen

# Elemente Hinzufügen(Rechts)

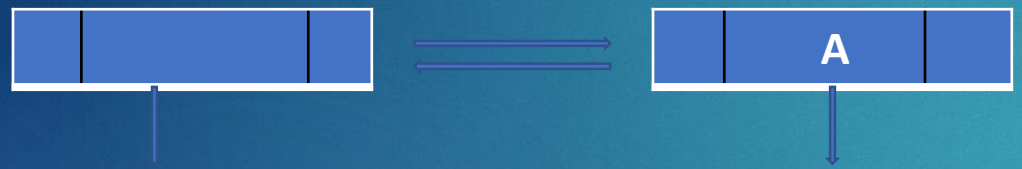
//A zu der Liste hinzufügen


//Weiter die Liste Elemente Hinzufügen

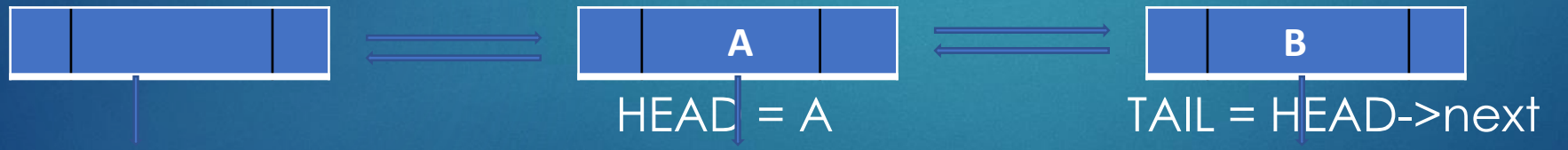


# Elemente Hinzufügen(Rechts)

// A zu der Liste hinzufügen




//Weiter die Liste Elemente Hinzufügen



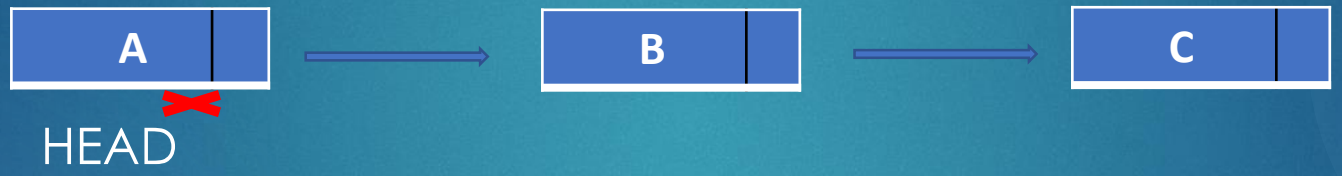

b->brev = list-> TAIL

# Liste löschen

//wir haben

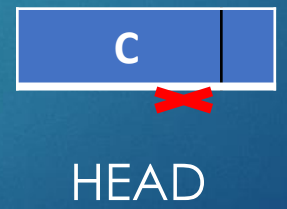

beim Löschen entfernt man Z.B den Kopf der Liste:

->




->

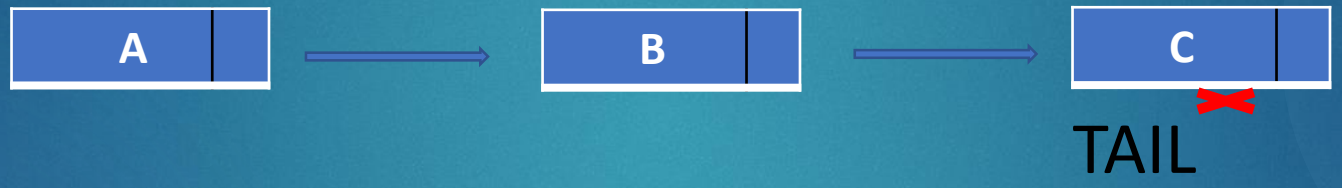

->



# Letztes Element löschen

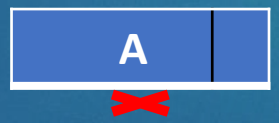
//wirhaben


->




->


->



# Suchen

- Man kann über Z.B die Wert nach einem Knoten suchen

```
double_node_t* finde_node_D(double_list_t* list, void* val)
{
    if(!list)
        return NULL;

    double_node_t* node = list->head;

    while(node != NULL)
    {
        if (node->val == val)
        {
            return node;
        }

        node = node->next;
    }

    return NULL;
}
```

```
single_node_t* finde_node_S(single_list_t* list, void* val)
{
    if(!list)
        return NULL;

    single_node_t* node = list->head;

    while(node != NULL)
    {
        if (node->val == val)
        {
            return node;
        }

        node = node->next;
    }
    //ConsoleS(list);
    return NULL;
}
```

# Suchen


// wir haben

// wert1 = A, wert2 = B, wert3 = C

// unser Hilfsnode

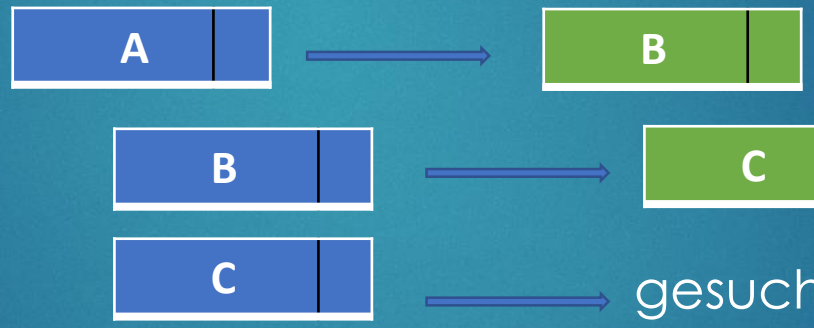
//erster Zyklus A != C

// Zweiter Zyklus B != C

// letzter Zyklus C == C

Wir suchen nach dem Knoten, der C trägt


enthält in sich die Wert A;



# Das Letzte Element in single linked list finden

- Man kann das Letzte Element eines Knotens durch eine kleine Implementierung finden.

```
single_node_t* find_prev_node_S(single_list_t* list, single_node_t* node)
{
    if(!list || !node)
        return NULL;

    single_node_t* curr = list->head;
    while(curr->next != node)
    {
        curr = curr->next;
    }
    //ConsoleS(list);
    return curr;
```

# Zeit vergleichen