



DOUBLE ENDED QUEUE (DEQUE/DEQUEUE)

Seminar – Data Structure in C

Erik Lorenz
20.07.2020



GLIEDERUNG

1. Grundlagen – *Was ist eine Deque?*
2. Umsetzung – *Welche Möglichkeiten gibt es?*
 1. *Double-Linked-List (C)*
 2. *Felder mit Hilfsindex (C)*
 3. *Ringpuffer (C)*
 4. *Vektorähnliche Chunks (C++)*
3. Benchmark-Test – *Welche Schreib- und Lesegeschwindigkeit haben diese?*
4. Zusammenfassung

Quellen



1. GRUNDLAGEN

Was ist eine Deque?



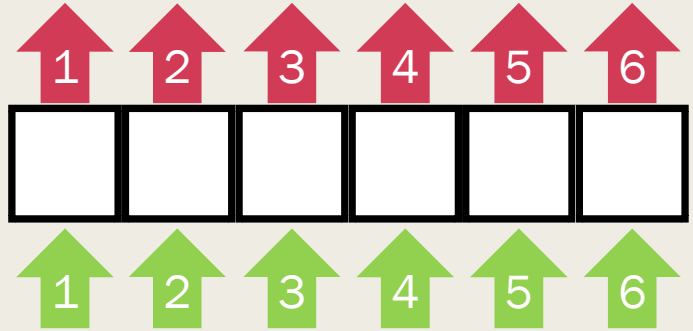
Double Ended Queues(DEQUEUE)

= Datenstruktur, eindimensionale Liste mit zwei Endpunkten

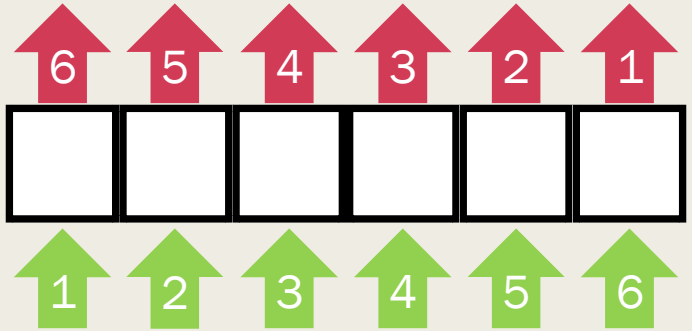
→ Ideen von FIFO(First In First Out) und LIFO (Last In First Out) bzw. Queue und Stack verbindet

[1],[2]

FIFO:



LIFO:

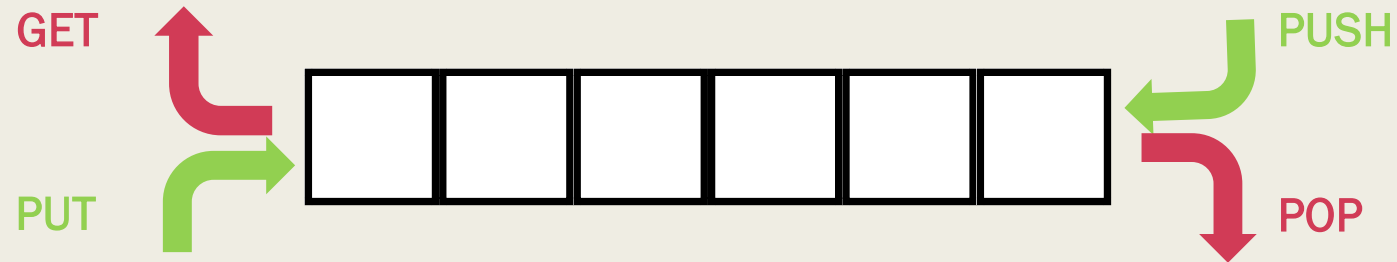


Deque (Double Ended Queues)

= Datenstruktur, eindimensionale Liste mit zwei Endpunkten

→ Ideen von FIFO (First In First Out) und LIFO (Last In First Out) bzw. Queue und Stack verbindet

[1],[2]



[2]

Methoden

```
deque * init_deque ();
void free_deque (deque * q);
//HINTERES ENDE
// einfügen
int push(deque * q,int val);
//entnehmen und entfernen
int pop (deque * q);
//nur auslesen
int last (deque * q);
//VORDERES ENDE
// einfügen
int put(deque * q,int val);
//entnehmen und entfernen
int get (deque * q);
//nur auslesen
int first (deque * q);
```

•Eingangs-restricted Deque

- Löschen an beiden enden
- Einfügen nur an einer

•Ausgangs-restricted Deque

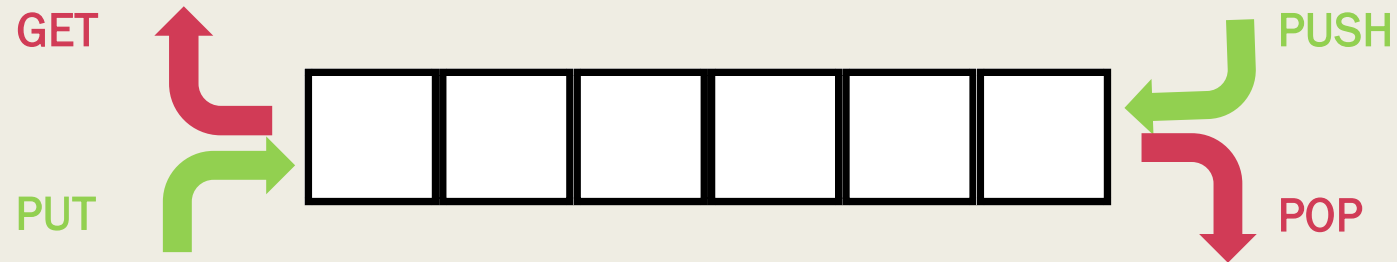
- Einfügen an beiden enden
- Löschen nur an einer [1]

Deque (Double Ended Queues)

= Datenstruktur, eindimensionale Liste mit zwei Endpunkten

→ Ideen von FIFO (First In First Out) und LIFO (Last In First Out) bzw. Queue und Stack verbindet

[1],[2]



[2]

Methoden

```
deque * init_deque ();
void free_deque (deque * q);
//HINTERES ENDE
// einfügen
int push(deque * q,int val);
//entnehmen und entfernen
int pop (deque * q);
//nur auslesen
int last (deque * q);
//VORDERES ENDE
// einfügen
int put(deque * q,int val);
//entnehmen und entfernen
int get (deque * q);
//nur auslesen
int first (deque * q);
```

•Eingangs-restricted Deque

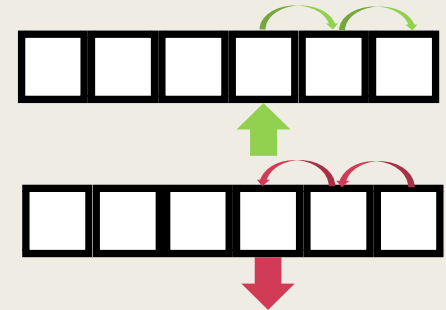
- Löschen an beiden enden
- Einfügen nur an einer

•Ausgangs-restricted Deque

- Einfügen an beiden enden
- Löschen nur an einer [1]

Zusätzliche Methoden:

- clear und all
- Isfull und isempty
- insert und erase



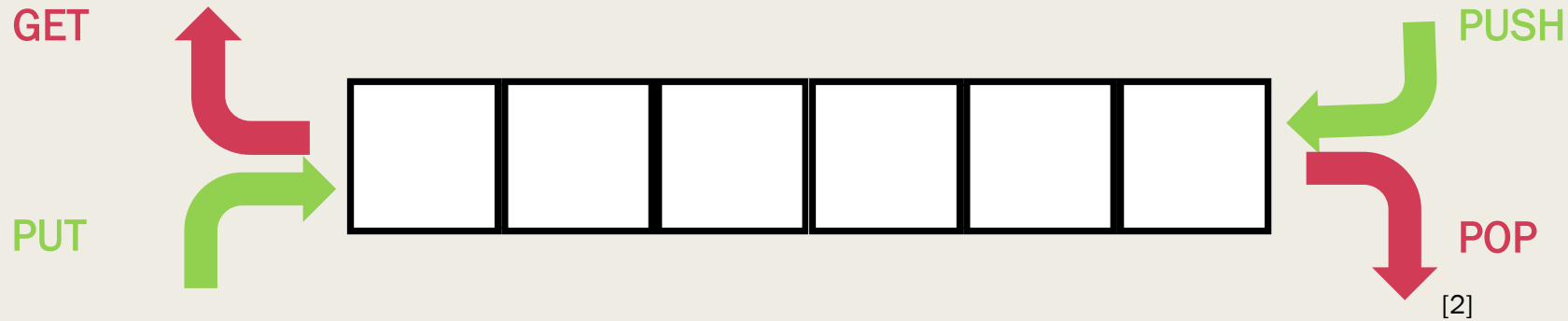
[3]

Deque (Double Ended Queues)

= Datenstruktur, eindimensionale Liste mit zwei Endpunkten

→ Ideen von FIFO(First In First Out) und LIFO (Last In First Out) bzw. Queue und Stack verbindet

[1],[2]



Methoden

```
deque * init_deque ();
void free_deque (deque * q);
//HINTERES ENDE
// einfügen
int push(deque * q,int val);
//entnehmen und entfernen
int pop (deque * q);
//nur auslesen
int last (deque * q);
//VORDERES ENDE
// einfügen
int put(deque * q,int val);
//entnehmen und entfernen
int get (deque * q);
//nur auslesen
int first (deque * q);
```

•Eingangs-restricted Deque

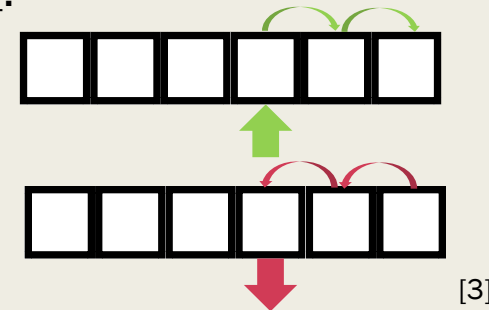
- Löschen an beiden enden
- Einfügen nur an einer

•Ausgangs-restricted Deque

- Einfügen an beiden enden
- Löschen nur an einer [1]

Zusätzliche Methoden:

- clear und all
- Isfull und isempty
- insert und erase



Vorteil

- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$

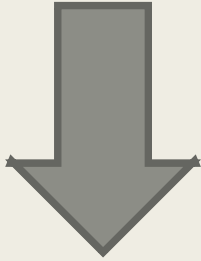
Nachteil

- Entnehmen aus der Mitte nicht leicht möglich

[1],[2],[17]

Anwendungen

- Genutzt als Stapel- und Warteschlange
- Rotationen im und gegen Uhrzeigersinn
- Werte an beiden Seiten entfernen bzw. anfügen



[1],[4],[7]

Beispiele:

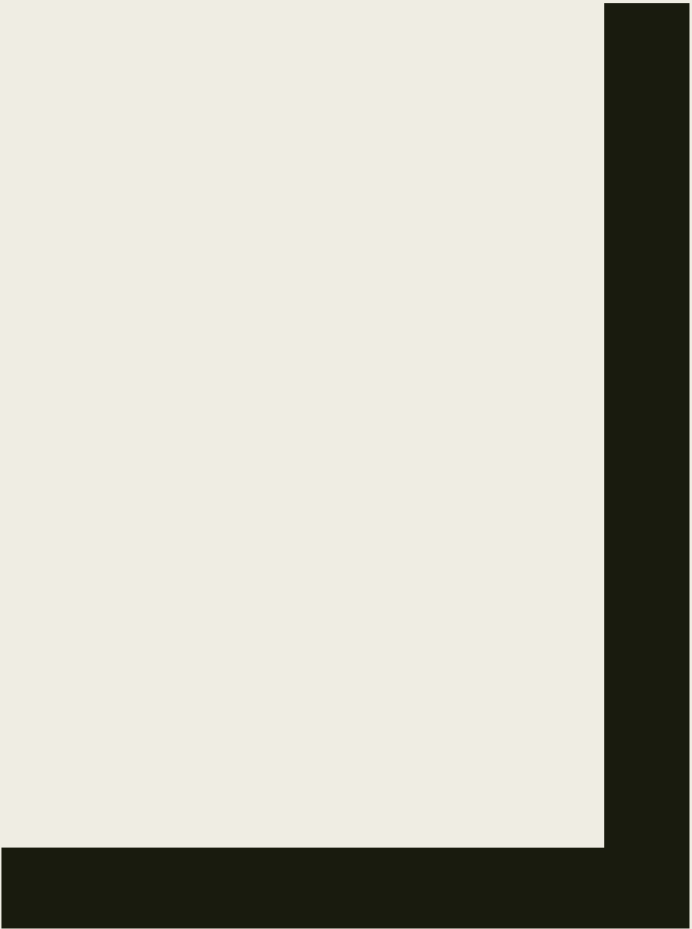
- Verteiltes Computing
- Networking
- Nichtdeterministischer endlicher Automat (NEA)
- Kürzester Weg in binär gewichteten Graphen^[5]
- Finden der ersten Rundtour^[6]

[1],[4],[7]

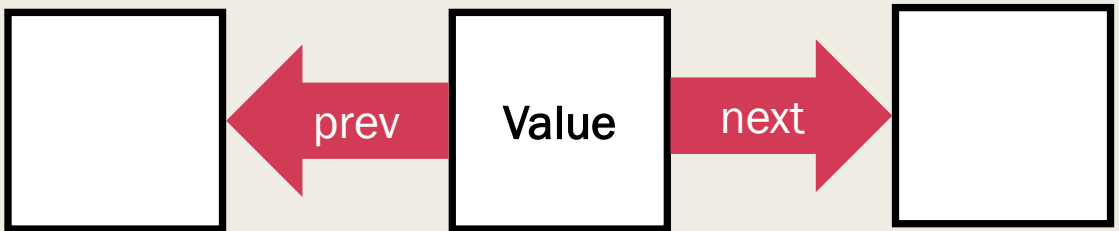


2. UMSETZUNG

Welche Möglichkeiten gibt es?

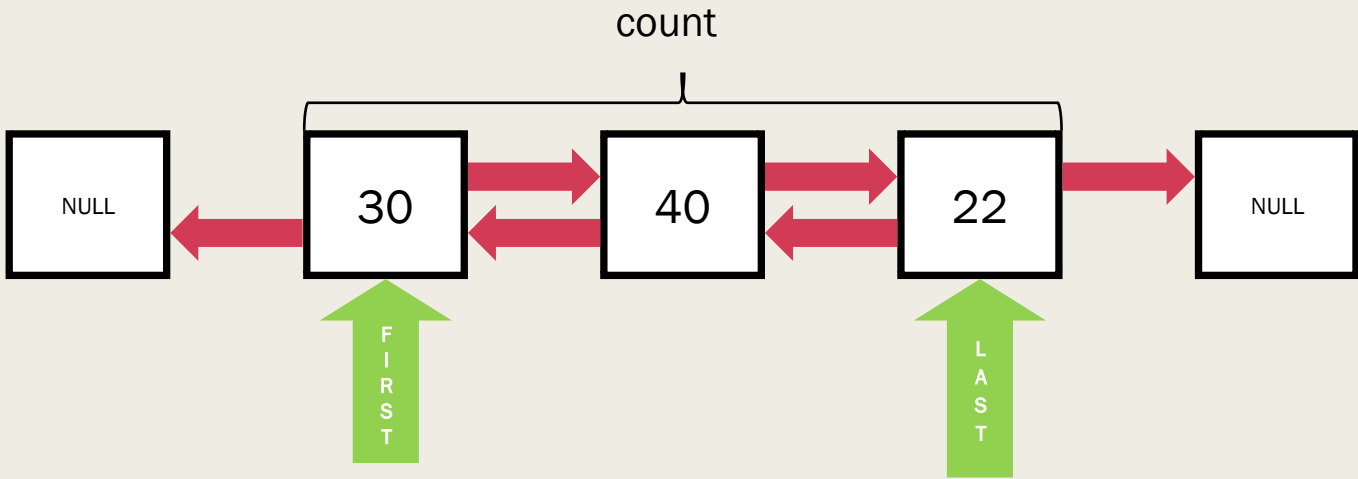


Double-Linked-List (C)



```
typedef struct node
{
    unsigned int value;
    struct node * next;
    struct node * prev;
}node;
```

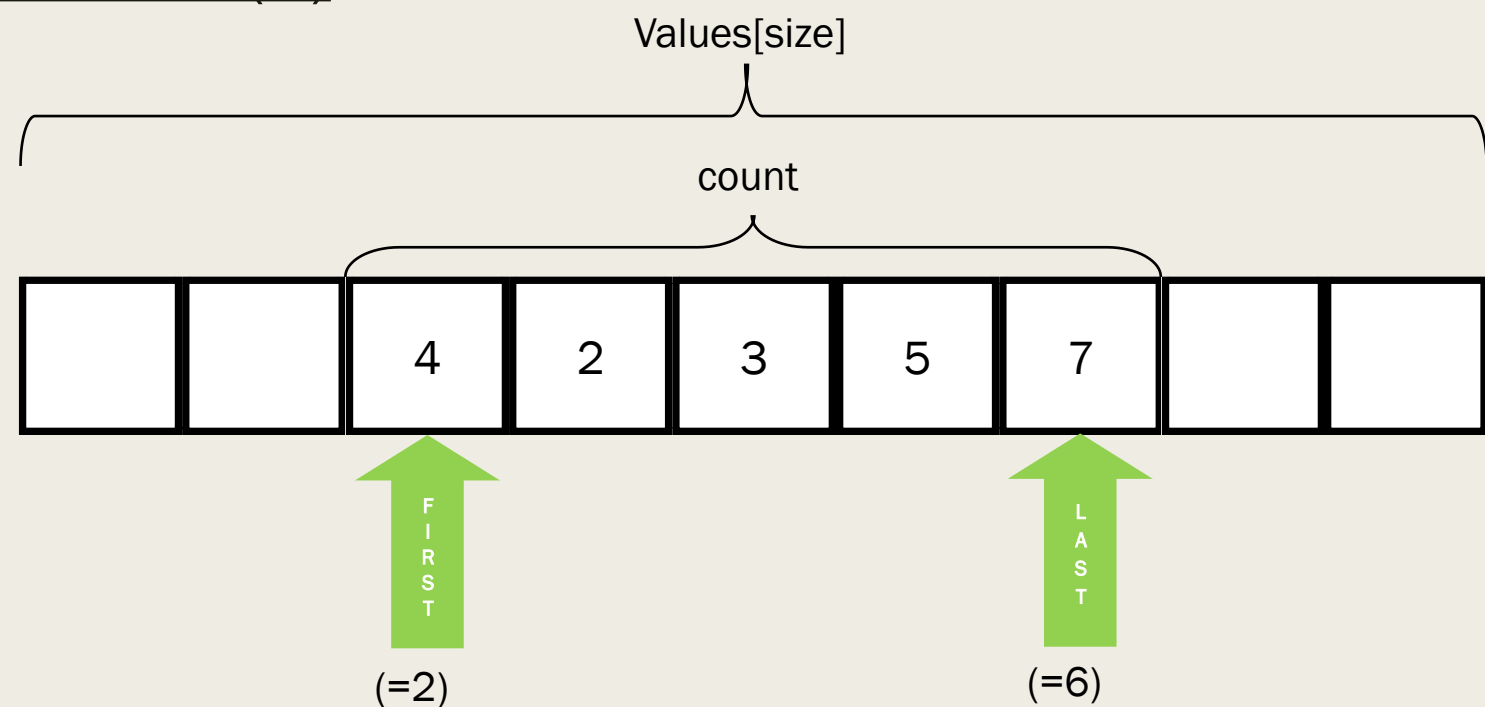
```
typedef struct deque
{
    struct node * first;
    struct node * last;
    int count;
} deque;
```



- Beliebige Größe der Deque (unbounded)

- Felder quer im Speicher verteilt → langsam

Felder mit Hilfsindex (C)



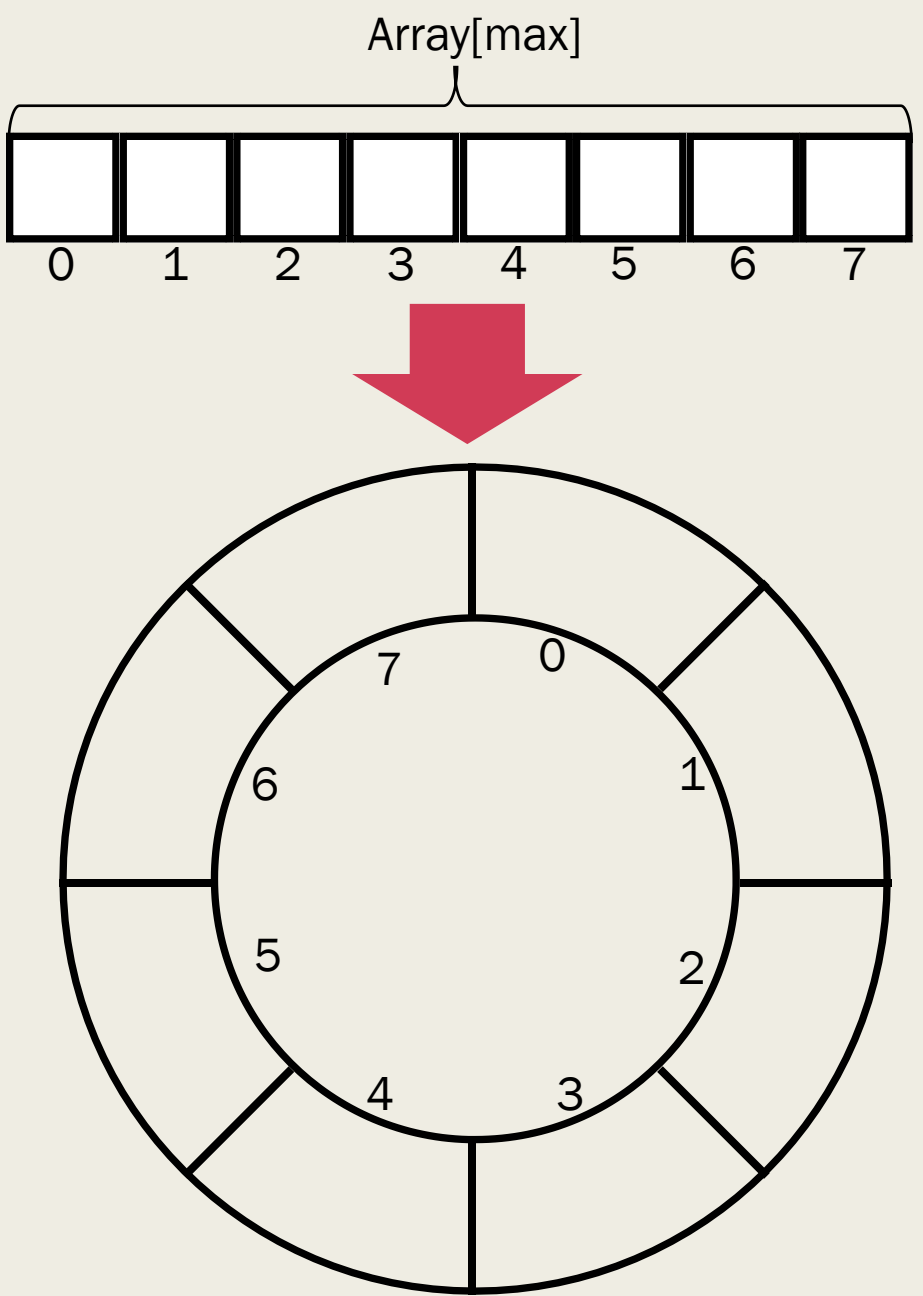
```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```



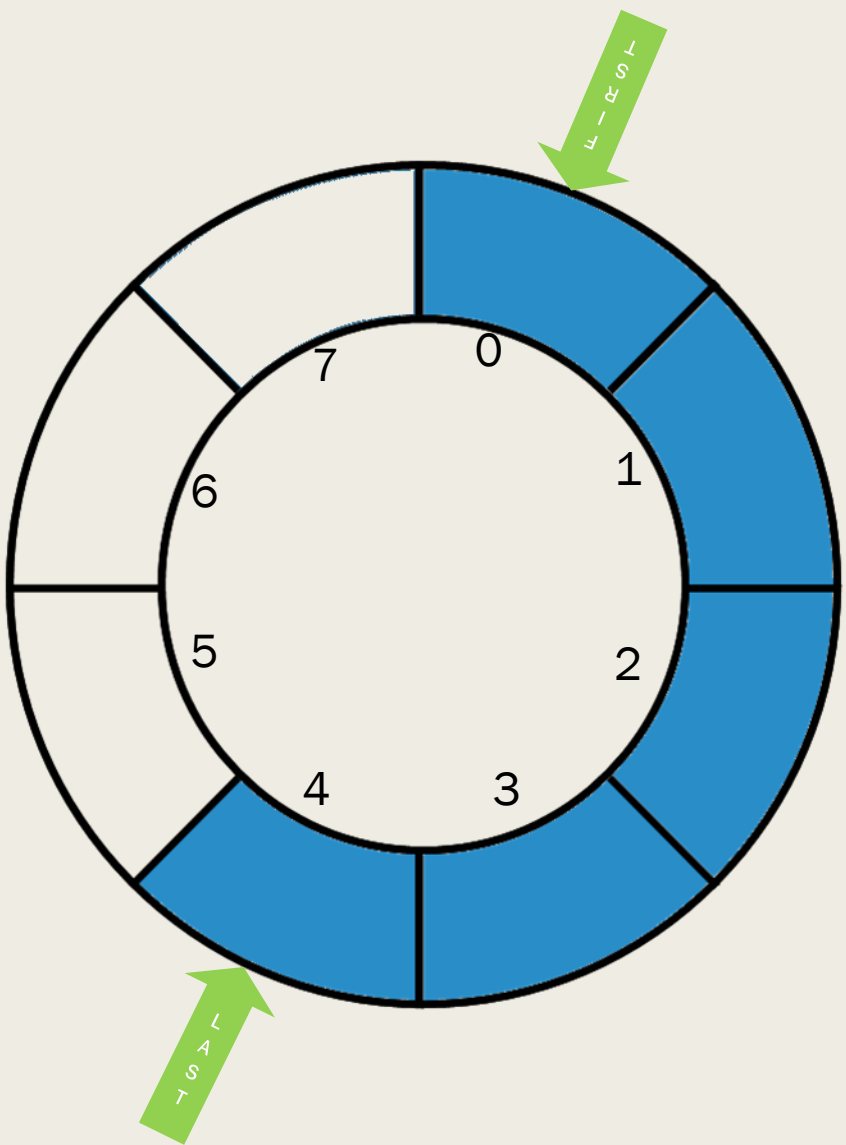
- Schnelle Speichernutzung



- Vorbelegung des Speichers
- Begrenzte Größe (bounded)
- Nur $n/2$ Elemente an jeder Seite

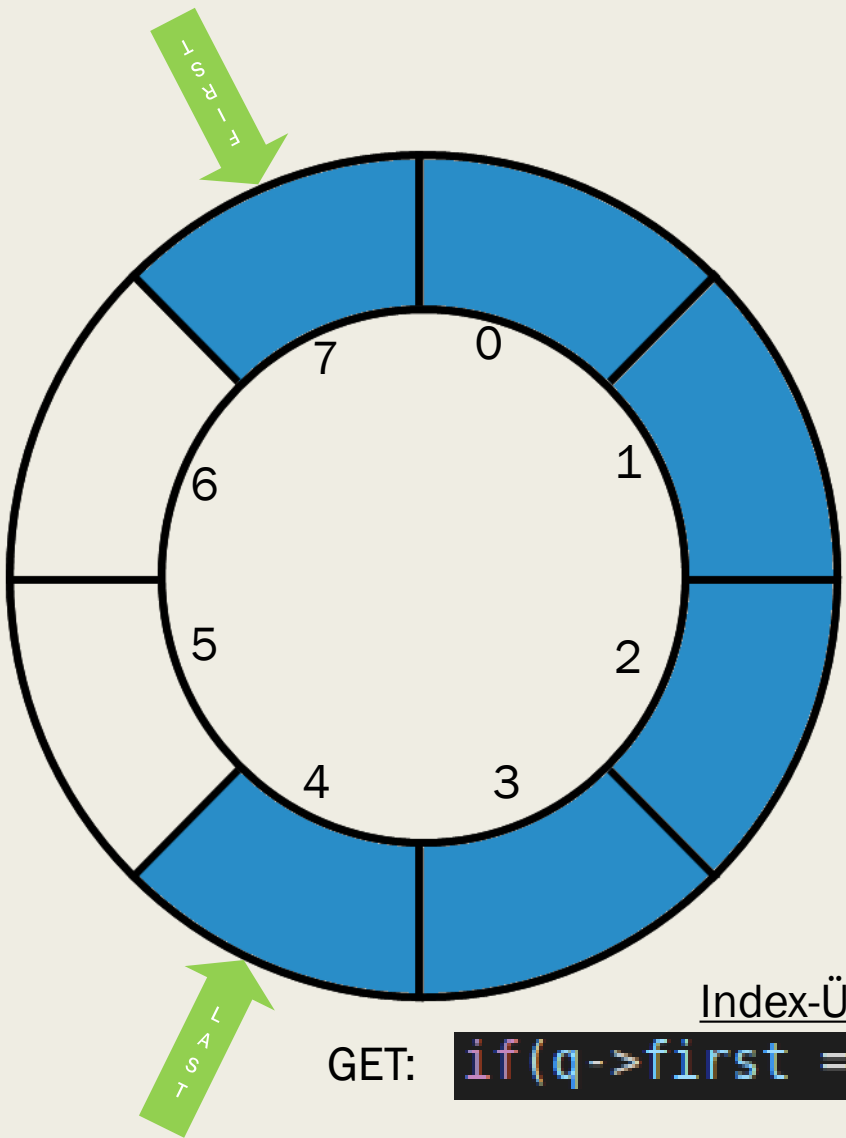


Ringpuffer (C)



```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```

Ringpuffer (C)



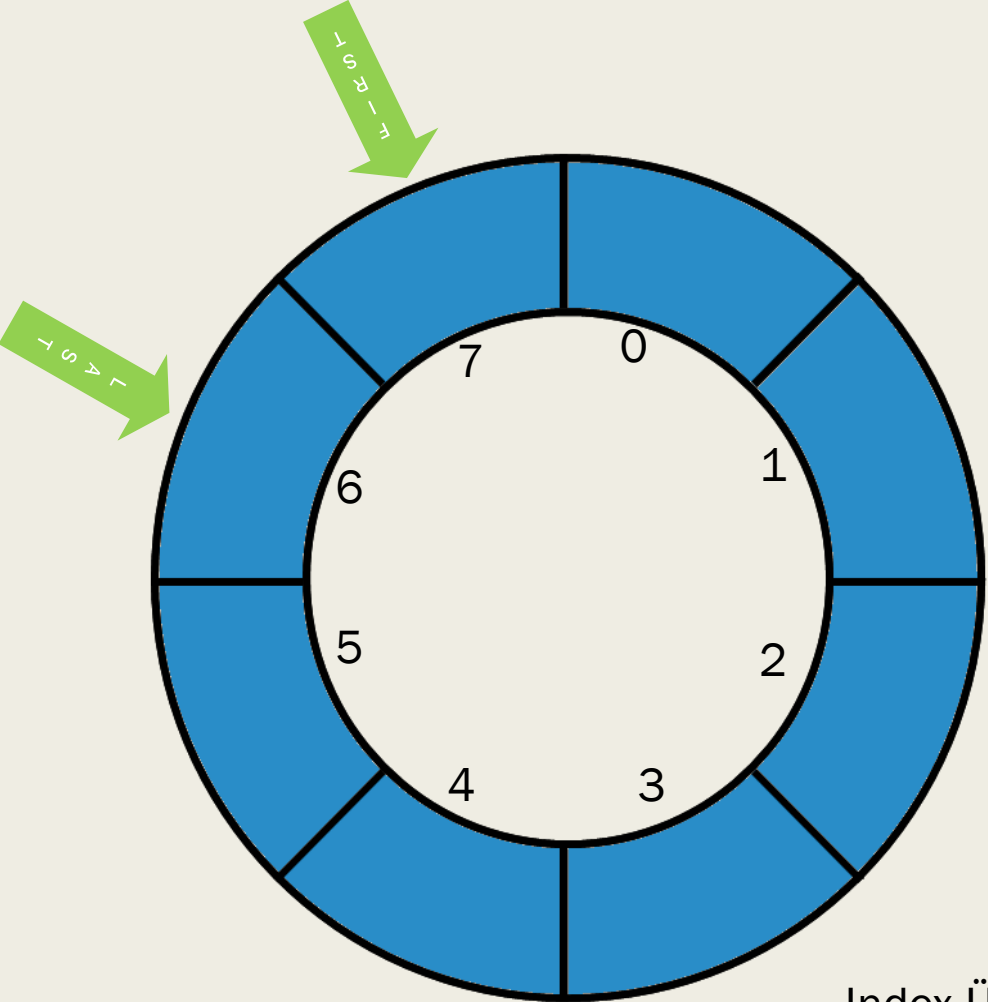
```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```

Index-Überlauf:

GET: `if(q->first == DEQUE_SIZE-1) { q->first = 0;}`

PUT: `if (q->first == 0) {q->first = DEQUE_SIZE-1;}`

Ringpuffer (C)



```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```

Puffer voll:

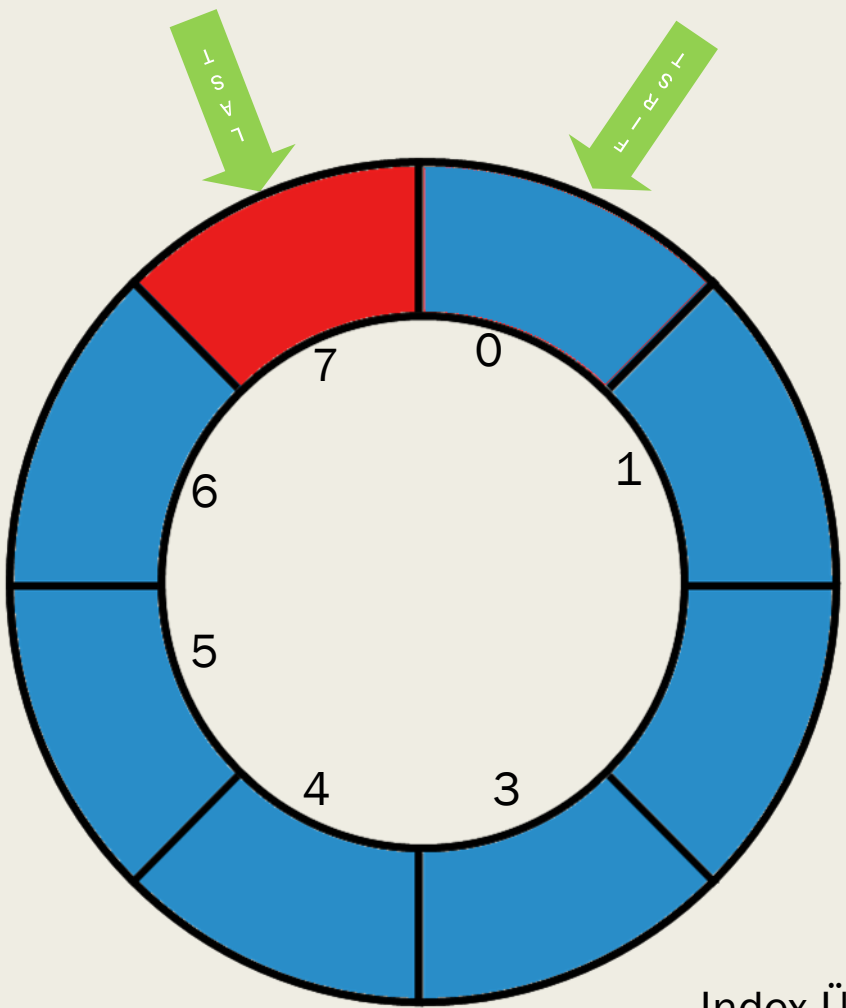
```
PUSH
if (q->last == DEQUE_SIZE-1) q->last = 0;
else q->last++;
q->values[q->last] = val;
if(q->count == DEQUE_SIZE)
{
    if(q->first == DEQUE_SIZE-1)
    {q->first = 0;}
    else q->first++;
}
else q->count++;
```

Index-Überlauf:

```
GET: if(q->first == DEQUE_SIZE-1) { q->first = 0;}
```

```
PUT: if (q->first == 0) {q->first = DEQUE_SIZE-1;}
```

Ringpuffer (C)



```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```

Puffer voll:

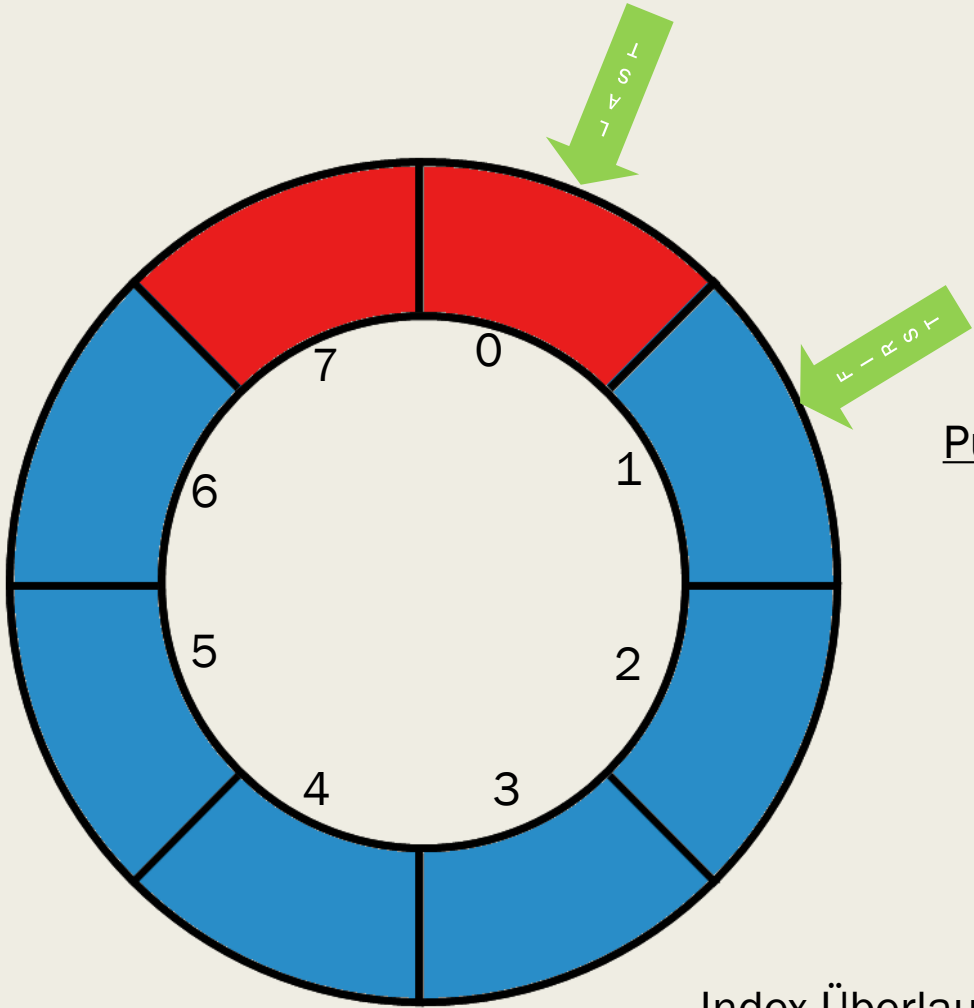
```
PUSH
if (q->last == DEQUE_SIZE-1) q->last = 0;
else q->last++;
q->values[q->last] = val;
if(q->count == DEQUE_SIZE)
{
    if(q->first == DEQUE_SIZE-1)
    {q->first = 0;}
    else q->first++;
}
else q->count++;
```

Index-Überlauf:

```
GET: if(q->first == DEQUE_SIZE-1) { q->first = 0;}
```

```
PUT: if (q->first == 0) {q->first = DEQUE_SIZE-1;}
```


Ringpuffer (C)



```
typedef struct deque
{
    int first;
    int last;
    int count;
    int values [DEQUE_SIZE];
} deque;
```

Puffer voll:

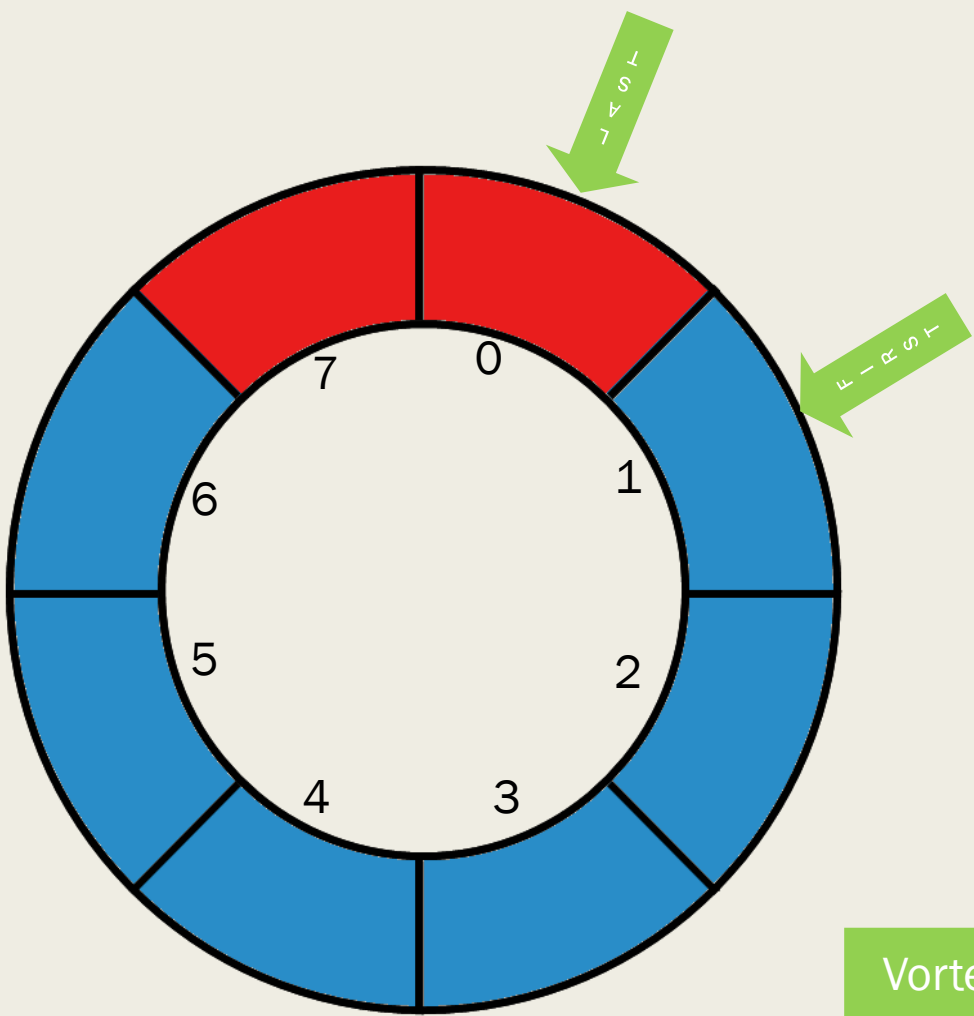
```
PUSH
if (q->last == DEQUE_SIZE-1) q->last = 0;
else q->last++;
q->values[q->last] = val;
if(q->count == DEQUE_SIZE)
{
    if(q->first == DEQUE_SIZE-1)
    {q->first = 0;}
    else q->first++;
}
else q->count++;
```

Index-Überlauf:

```
GET: if(q->first == DEQUE_SIZE-1) { q->first = 0;}
```

```
PUT: if (q->first == 0) {q->first = DEQUE_SIZE-1;}
```

Ringpuffer (C)



Vorteile

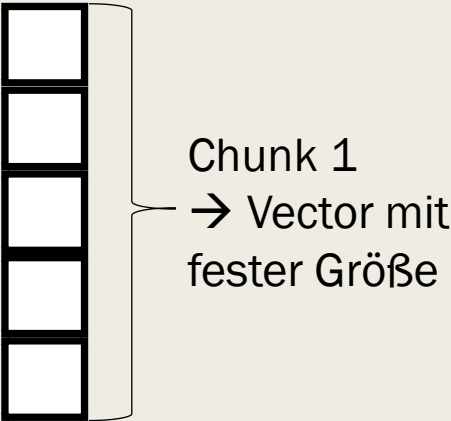
- Hinzufügen immer möglich
- Keine Anfügebegrenzung an den Seiten

Nachteile

- Felder werden überschrieben bzw.
- wieder begrenzt

Vektorähnliche Chunks

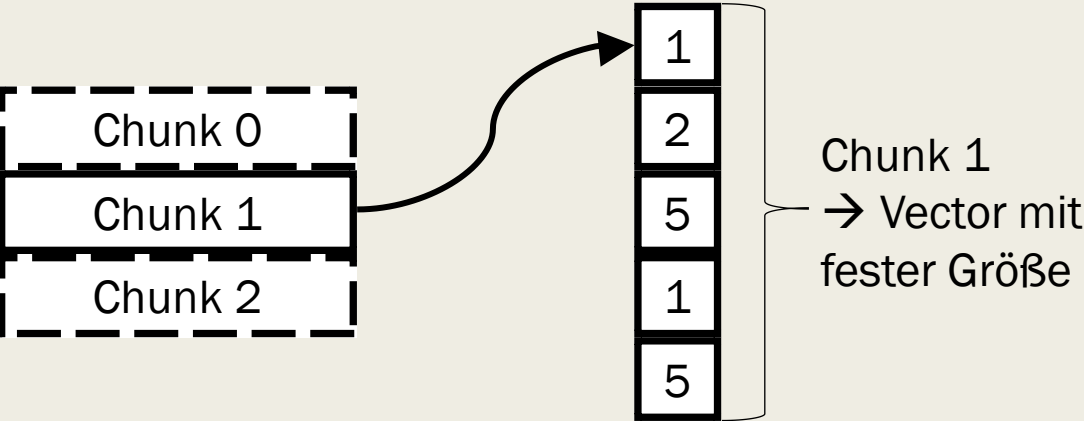
C++ → STL vorhanden



Vektorähnliche Chunks

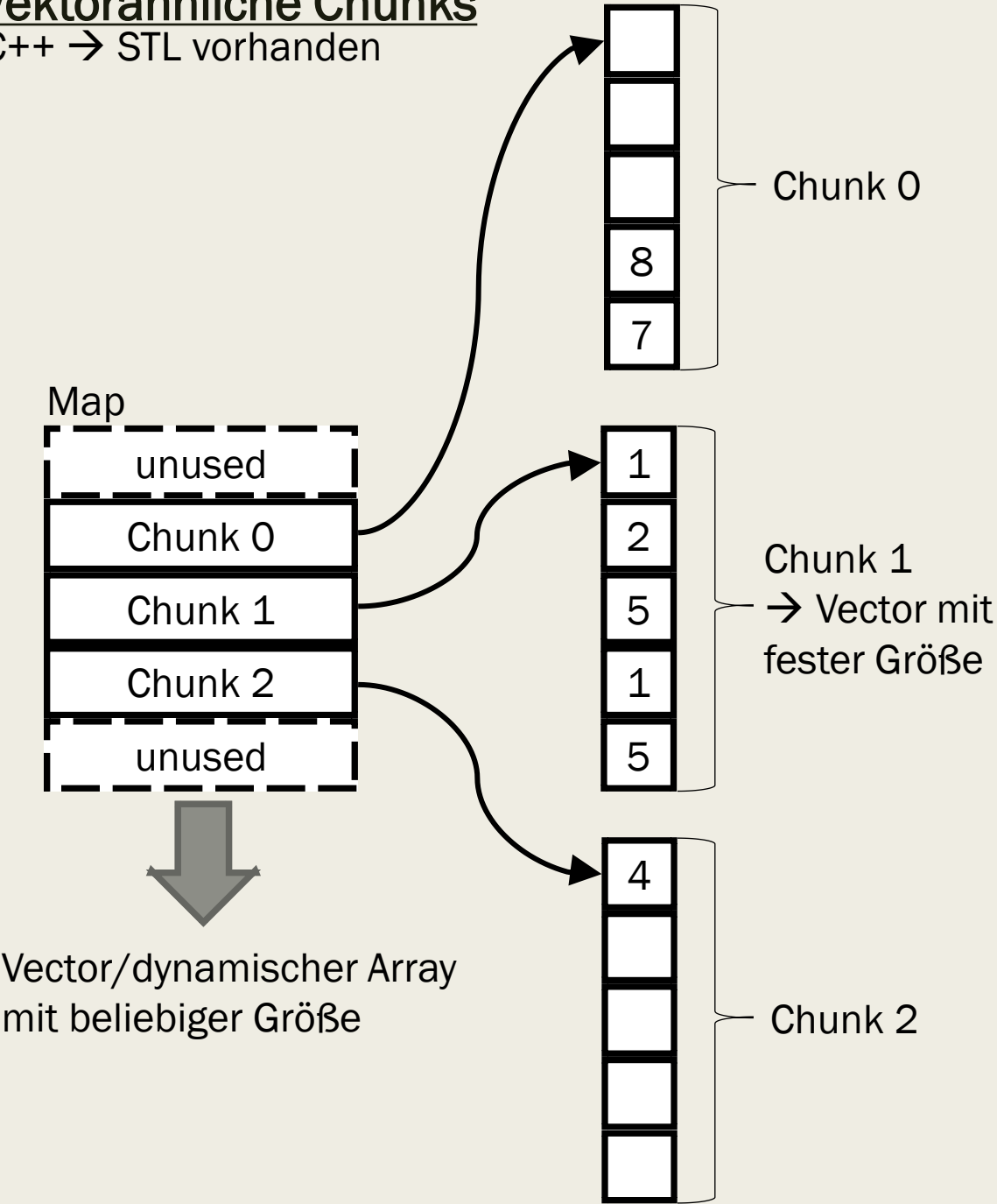
C++ → STL vorhanden

Map



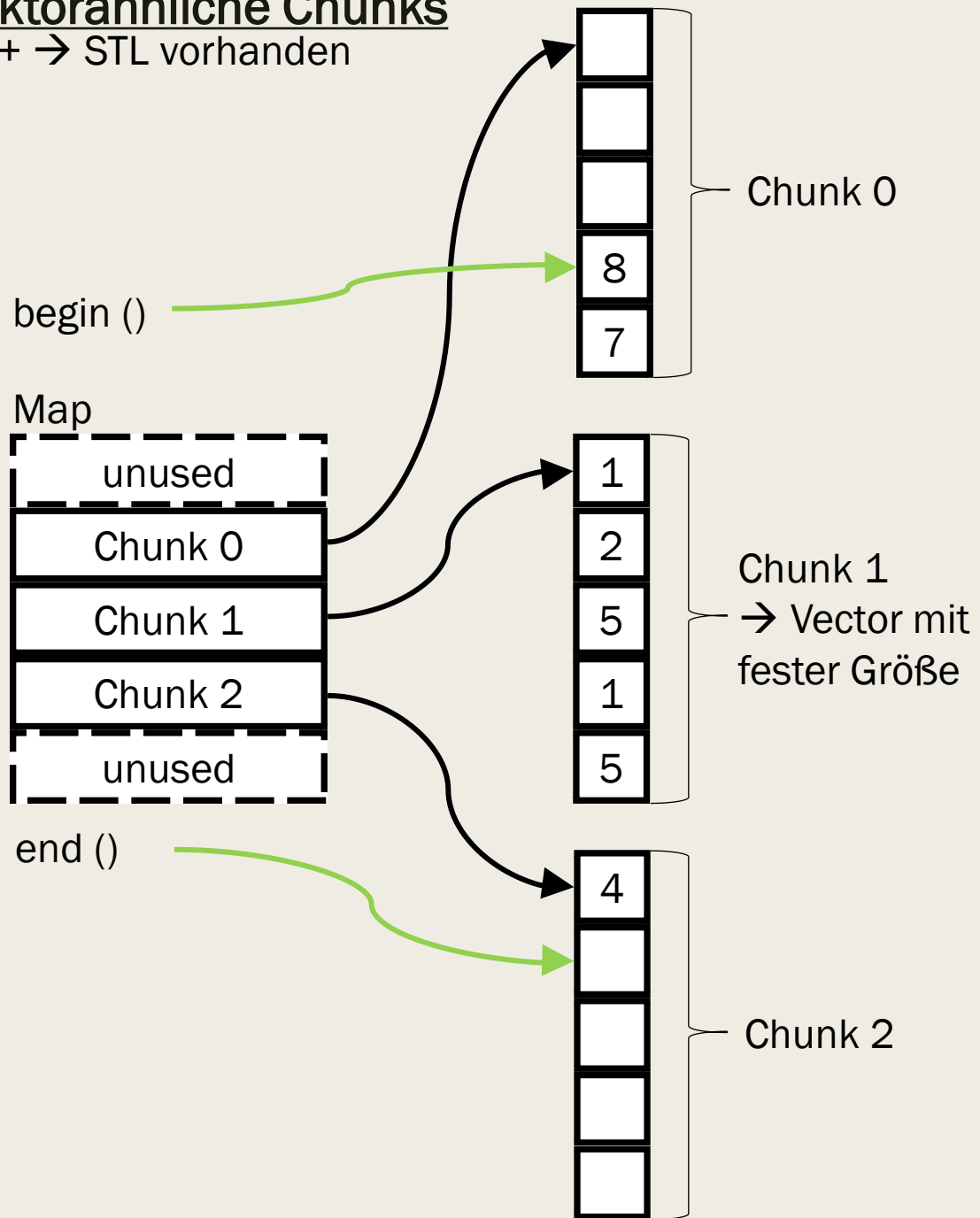
Vektorähnliche Chunks

C++ → STL vorhanden



Vektorähnliche Chunks

C++ → STL vorhanden



Vorteile

- unbounded
- Schnelle und effiziente Speichernutzung

Nachteile

- Dynamische Arrays bzw Vektoren werden benötigt



3. BENCHMARK TEST

Welche Schreib- und Lesegeschwindigkeit haben diese?



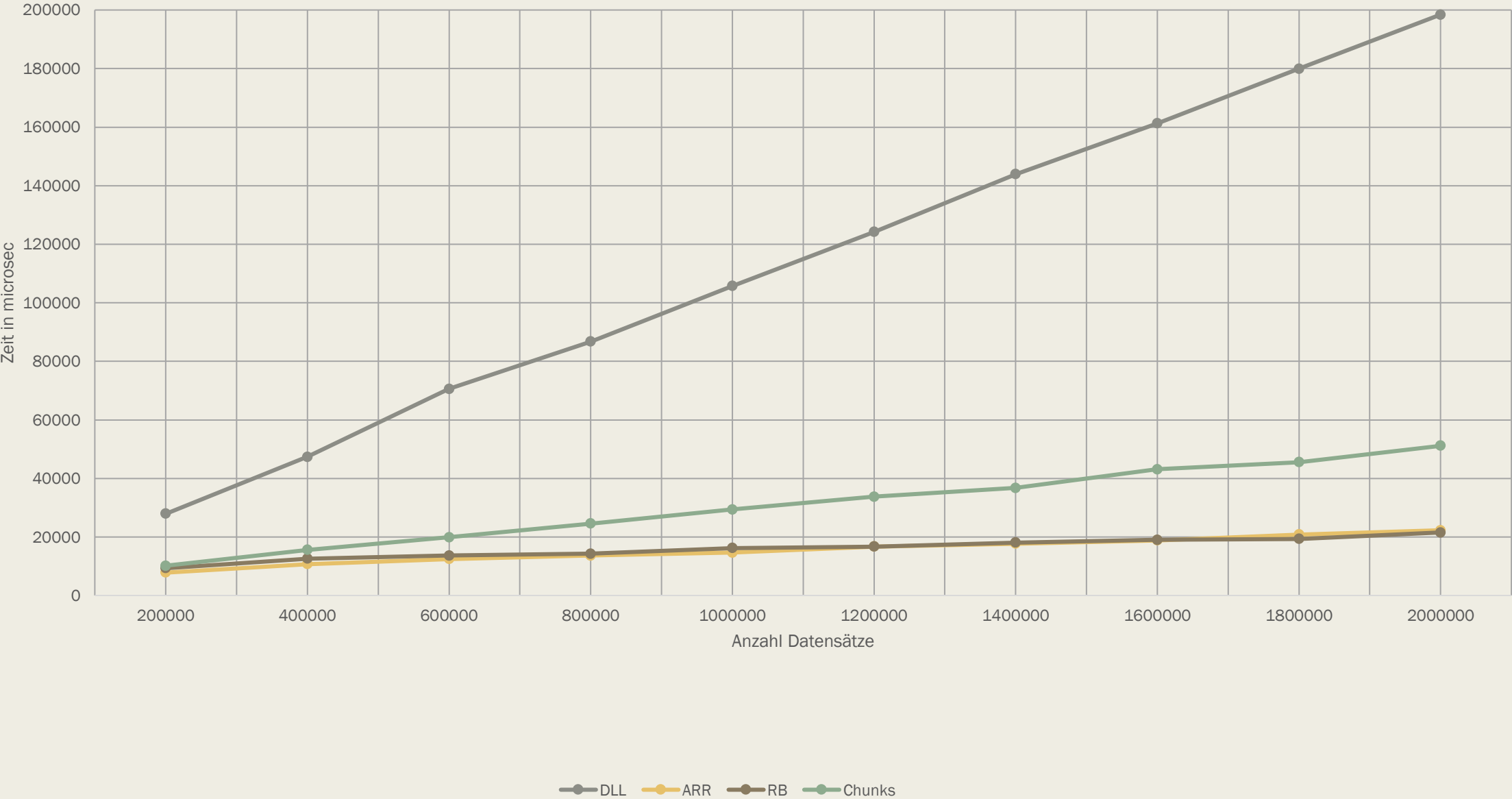
Vorgehen

Test 1	Test 2
Init_deque(), an jeder Seite abwechselnd anfügen	An jeder Seite abwechselnd entnehmen, free_deque()
Anzahl Datensätze gesamt: in 200000er Schritten bis 2000000	
Jeweils 10-mal durchgeführt und Mittelwert berechnet	
Felder mit Hilfsindex: Deque_Size = Anzahl Datensätze Ringpuffer: Deque_Size = 100000 Anfügen nur an einer Seite	Felder mit Hilfsindex = Ringpuffer

Bench Mark Test

Diagramm – Test 1

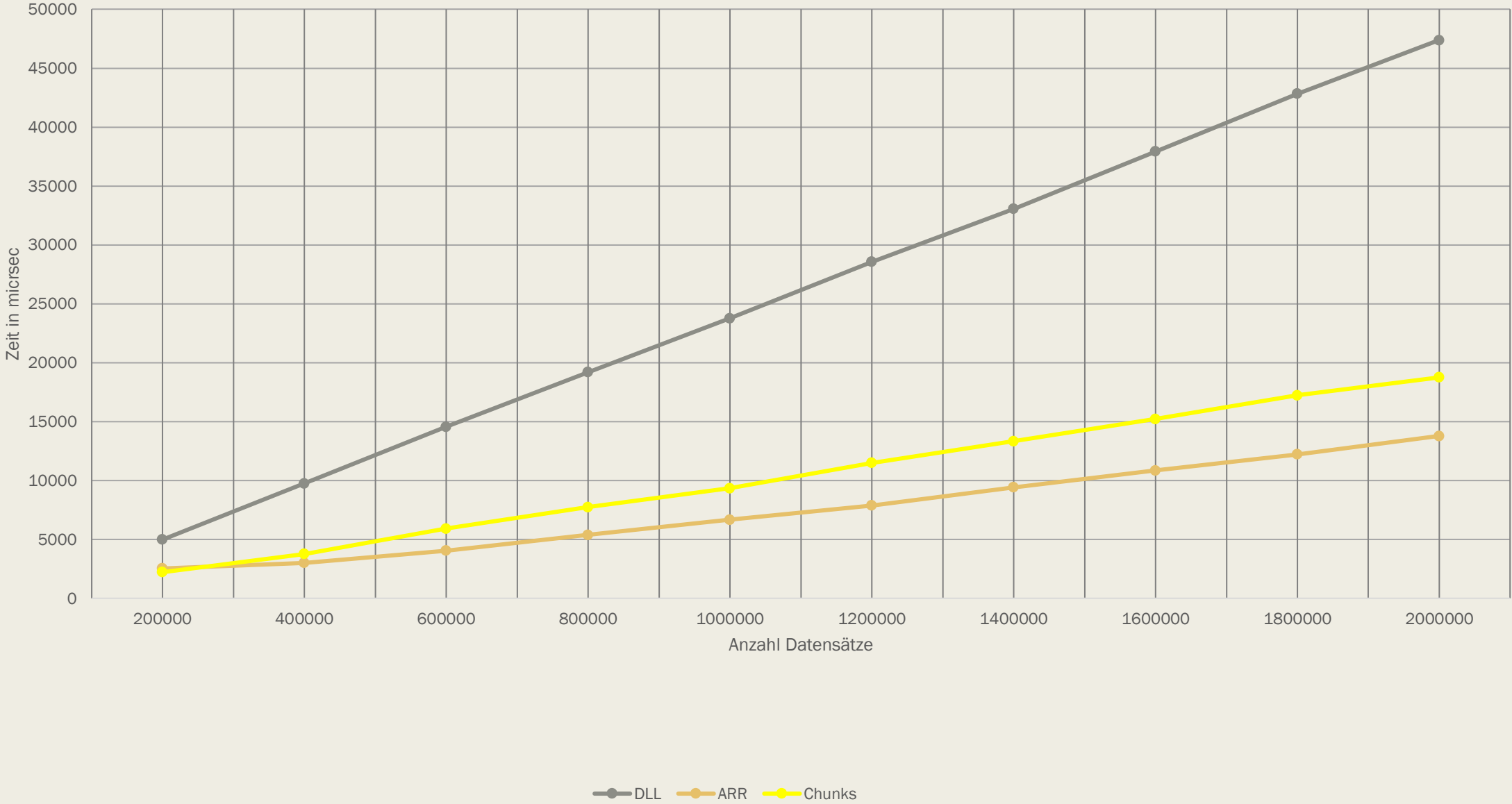
→ Big O Notation : $O(n)$



Bench Mark Test

Diagramm – Test 2

→ Big O Notation : $O(n)$

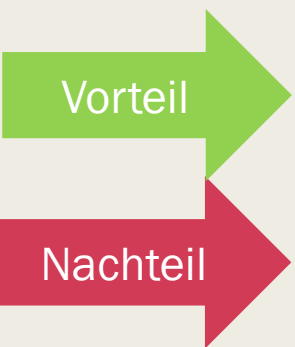
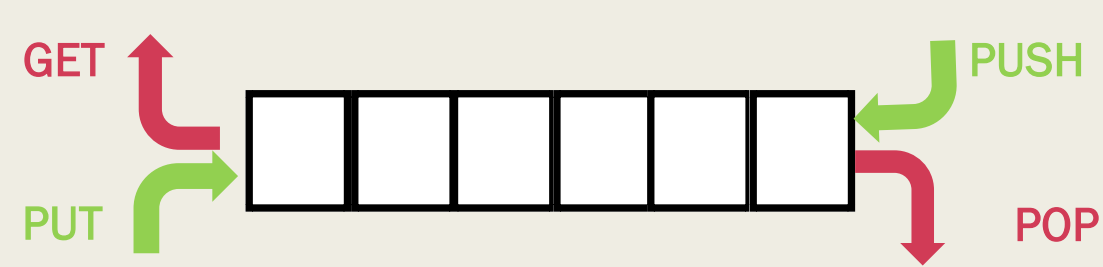




4. ZUSAMMENFASSUNG

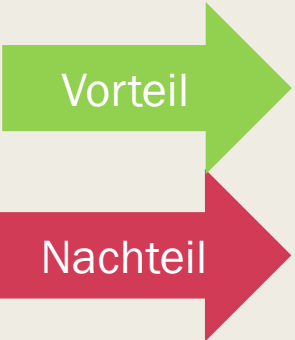
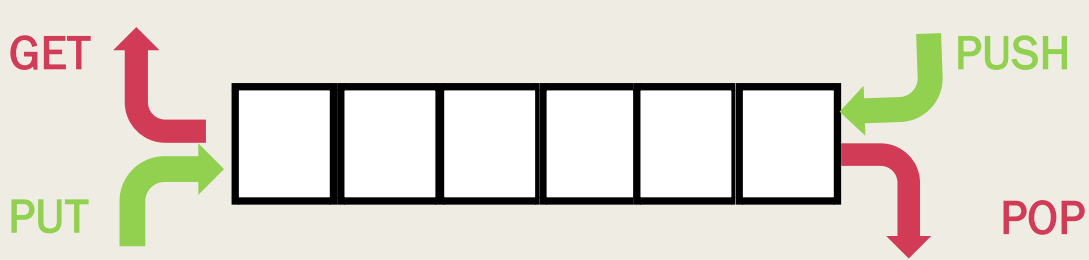


DEQUE(Double Ended Queue) = Datenstruktur, eindimensionale Liste mit zwei Endpunkten



- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$
- Entnehmen aus der Mitte nicht leicht möglich

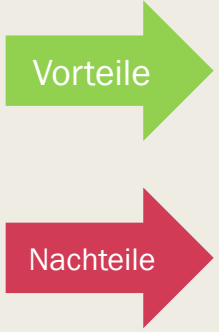
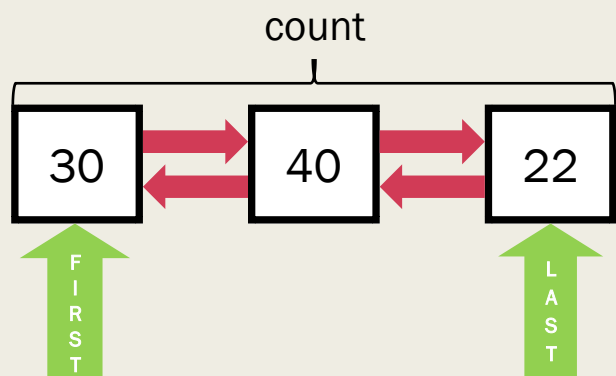
DEQUEUE(Double Ended Queue) = Datenstruktur, eindimensionale Liste mit zwei Endpunkten



- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$
- Entnehmen aus der Mitte nicht leicht möglich

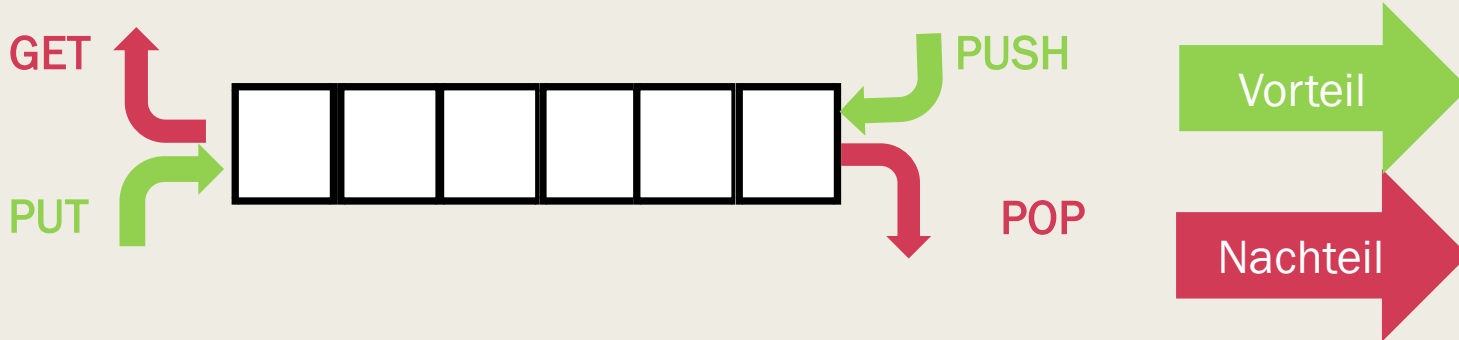
Vorgehen:

- Double-Linked-List



- Beliebige Größe der Deque (unbounded)
- Felder quer im Speicher verteilt → langsam

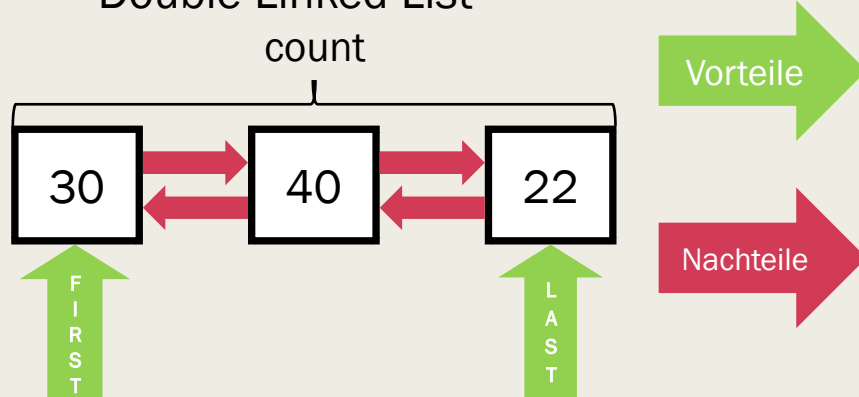
DEQUEUE(Double Ended Queue) = Datenstruktur, eindimensionale Liste mit zwei Endpunkten



- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$
- Entnehmen aus der Mitte nicht leicht möglich

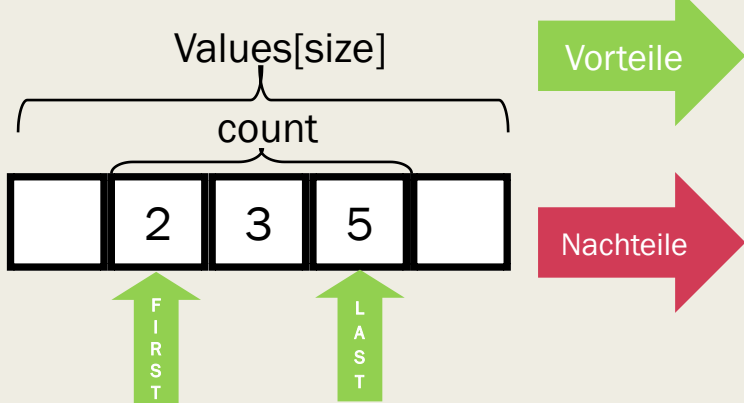
Vorgehen:

- Double-Linked-List



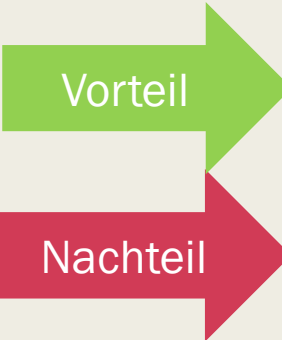
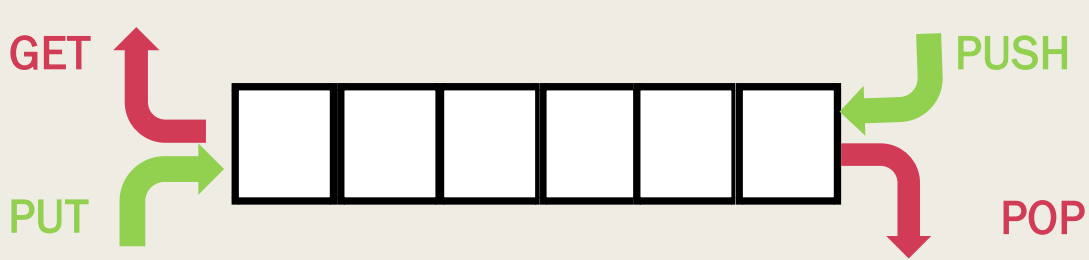
- Beliebige Größe der Deque (unbounded)
- Felder quer im Speicher verteilt → langsam

- Feld mit Hilfsindex



- Schnelle Speichernutzung
- Vorbelegung des Speichers
- Begrenzte Größe (bounded)
- Nur $n/2$ Elemente an jeder Seite

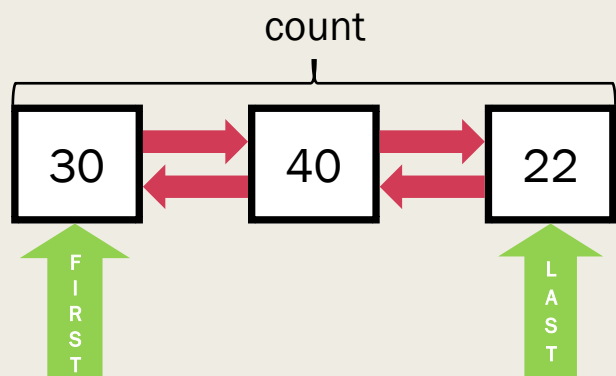
DEQUE(Double Ended Queue) = Datenstruktur, eindimensionale Liste mit zwei Endpunkten



- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$
- Entnehmen aus der Mitte nicht leicht möglich

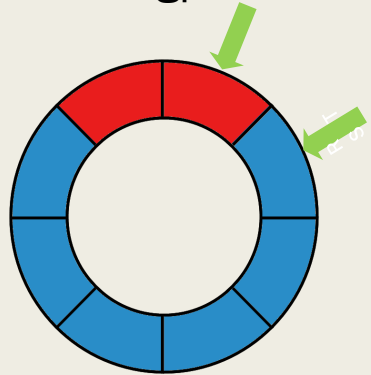
Vorgehen:

- Double-Linked-List



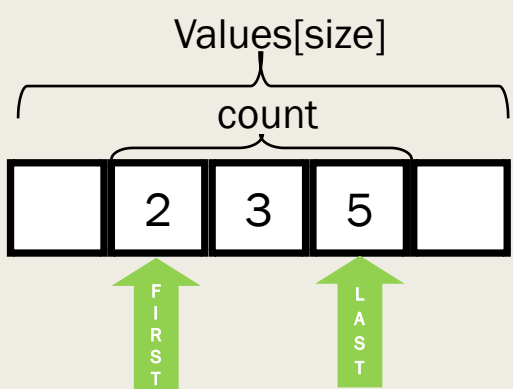
- Beliebige Größe der Deque (unbounded)
- Felder quer im Speicher verteilt → langsam

- Ringpuffer



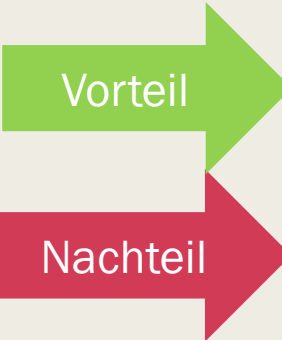
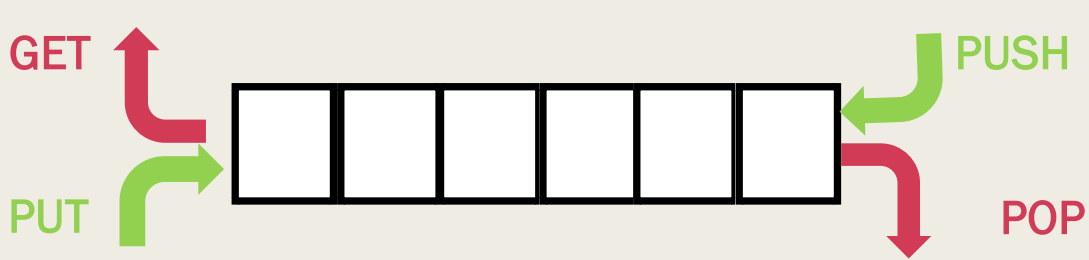
- Hinzufügen immer möglich
- Keine Anfügebegrenzung an den Seiten
- Felder werden überschrieben bzw. wieder begrenzt

- Feld mit Hilfsindex



- Schnelle Speichernutzung
- Vorbelegung des Speichers
- Begrenzte Größe (bounded)
- Nur $n/2$ Elemente an jeder Seite

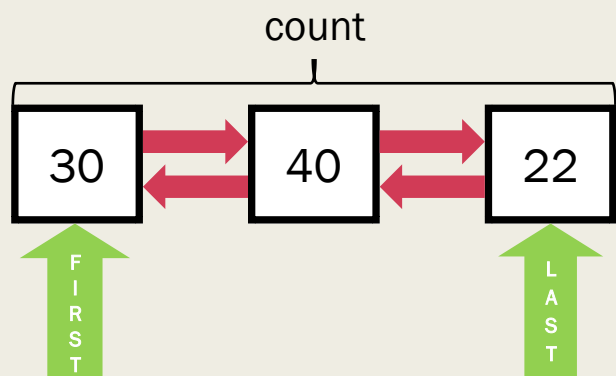
DEQUE(Double Ended Queue) = Datenstruktur, eindimensionale Liste mit zwei Endpunkten



- Anfügen/Entnehmen simpel und Flexibel
- Zeitaufwand für jede Funktion → $O(1)$
- Entnehmen aus der Mitte nicht leicht möglich

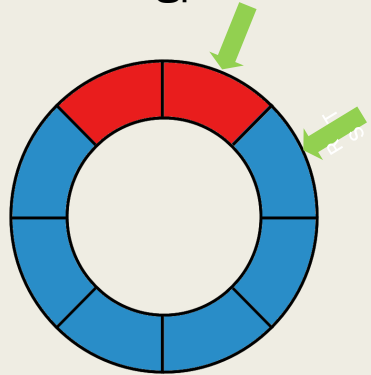
Vorgehen:

- Double-Linked-List



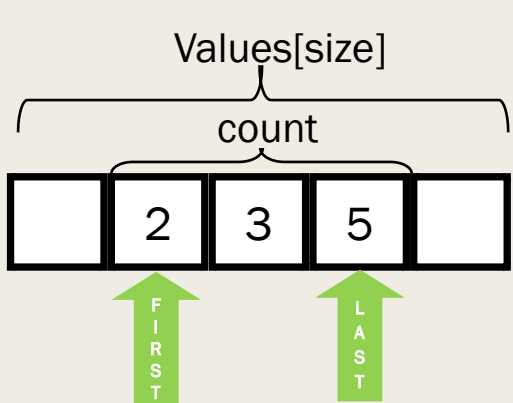
- Beliebige Größe der Deque (unbounded)
- Felder quer im Speicher verteilt → langsam

- Ringpuffer



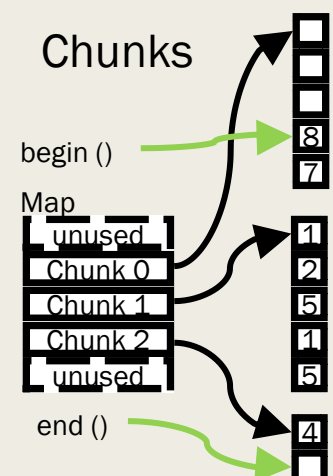
- Hinzufügen immer möglich
- Keine Anfügebegrenzung an den Seiten
- Felder werden überschrieben bzw. wieder begrenzt

- Feld mit Hilfsindex



- Schnelle Speichernutzung
- Vorbelegung des Speichers
- Begrenzte Größe (bounded)
- Nur $n/2$ Elemente an jeder Seite

- Chunks



- unbounded
- Schnelle und effiziente Speichernutzung
- Dynamische Arrays bzw Vektoren werden benötigt

Quellen

- [1] Wikipedia contributors, 2020, Double-ended queue, In: Wikipedia, The Free Encyclopedia, 15.06.2020, [Zugriff am : 08.07.2020], Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Double-ended_queue&oldid=962716369
- [2] Wikipedia-Autoren, 2020, Deque, In: Wikipedia, Die freie Enzyklopädie, 11.04.2019, [Zugriff am : 08.07.2020], Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Deque&oldid=187480707>
- [3] U. Breymann, 2002, Deque, Chemnitz: Technische Universität Chemnitz, [Zugriff am: 12.06.2020] Verfügbar unter https://www.tu-chemnitz.de/informatik/friz/Grundl-Inf/Aufgaben/stdbibl/hilf_html/p14.html
- [4] o.V. , 2019, Deque | Set 1 : (Introduction and Applications), Noida: GeeksforGeeks, [Zugriff am : 13.07.2020], Verfügbar unter: <https://www.geeksforgeeks.org/deque-set-1-introduction-applications/>
- [5] o.V., 2019, 0-1 BFS: (Shortest Path in a Binary Weight Graph), Noida: GeeksforGeeks, [Zugriff am: 13.07.2020], Verfügbar unter: <https://www.geeksforgeeks.org/0-1-bfs-shortest-path-binary-graph/>
- [6] o.V., 2019, Find the first circular tour that visits all petrol pumps, Noida: GeeksforGeeks, [Zugriff am: 13.07.2020], Verfügbar unter: <https://www.geeksforgeeks.org/find-a-tour-that-visits-all-stations/>
- [7] Chumbley, Alex, o.J, Double Ended Queues, San Francisco: Brilliant DMCA Claims, [Zugriff am: 13.07.2020], Verfügbar unter: <https://brilliant.org/wiki/double-ended-queues/#deque-questions>
- [8] o.V., 2020, Doubly Linked List | Set 1 (Introduction and Insertion), Noida: GeeksforGeeks, [Zugriff am: 13.07.2020] , Verfügbar unter: <https://www.geeksforgeeks.org/doubly-linked-list/>
- [9] mycodeschool, 2020, Doubly Linked List implementation in C, San Francisco: GitHub, [Zugriff am : 10.06.2020], Verfügbar unter: <https://gist.github.com/mycodeschool/7429492>
- [10] o.V, 2020, Double Ended Queue, Ghaziabad: Studytonight, [Zugriff am: 12.06.2020], Verfügbar unter: <https://www.studytonight.com/data-structures/double-ended-queue>
- [11] o.V., 2018, Implementation of Deque using circular array, Noida: GeeksforGeeks, [Zugriff am: 13.06.2020], Zugriff auf: <https://www.geeksforgeeks.org/implementation-deque-using-circular-array/>
- [12] o.V., 2016, Wie implementiere ich eine zirkuläre Liste (Ringpuffer) in C?, it-swarm.dev, [Zugriff am : 20.06.2020], Verfügbar unter: <https://www.it-swarm.dev/de/c/wie-implementiere-ich-eine-zirkulaere-liste-ringpuffer-c/958422889/>
- [13] Chandrasekaran, Siddharth, 2014, Implementing Circular Buffer in C, *Embed Journal*, [Zugriff am: 13.07.2020] Verfügbar unter: <https://embedjournal.com/implementing-circular-buffer-embedded-c/>
- [14] T. Ottmann/ P. Widmayer, Algorithmen und Datenstrukturen, Heidelberg/Berlin, 4. Auflage, Spektrum Akademischer Verlag GmbH, S.33 ff.
- [15] Nitron, 2003, An In-Depth Study of the STL Deque Container, Code Project, [Zugriff am: 20.07.2020], Verfügbar unter: <https://www.codeproject.com/Articles/5425/An-In-Depth-Stud>
- [16] Martin Broadhurst, 2016, Deque in C, Cambridge: Martin Broadhurst, <http://www.martinbroadhurst.com/deque.html>
- [17] Richard Li, 2019, What is Big O Notation, Medium, [Zugriff am 13.07.2020], Verfügbar unter: <https://medium.com/@richardli125/what-is-big-o-notation-d72f602370ea>