Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder

Holger Dell
Folien adaptiert von Philip Bille

Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder

Datenstrukturen

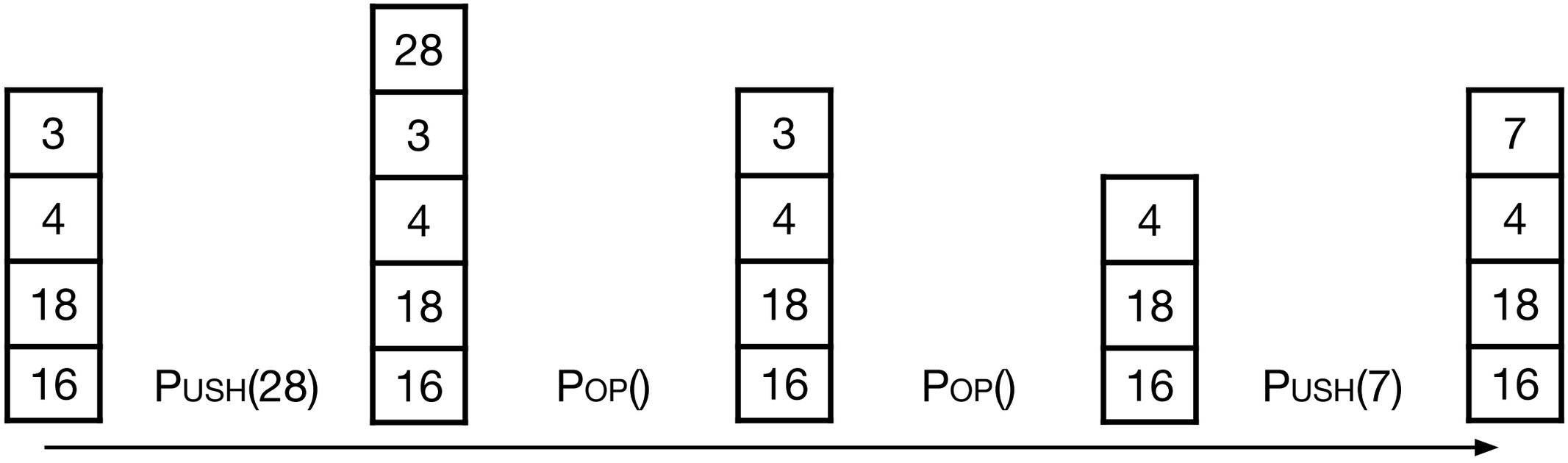
- Datenstruktur. Methode, um Daten so zu organisieren, dass Operationen wie Zugriff, Suche, Manipulation, usw. effizient möglich sind.
- Ziele.
  - Schnell.
  - Kompakt.
- Terminologie.
  - Abstrakte vs. konkrete Datenstruktur.
  - Dynamische vs. statische Datenstruktur.

Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder

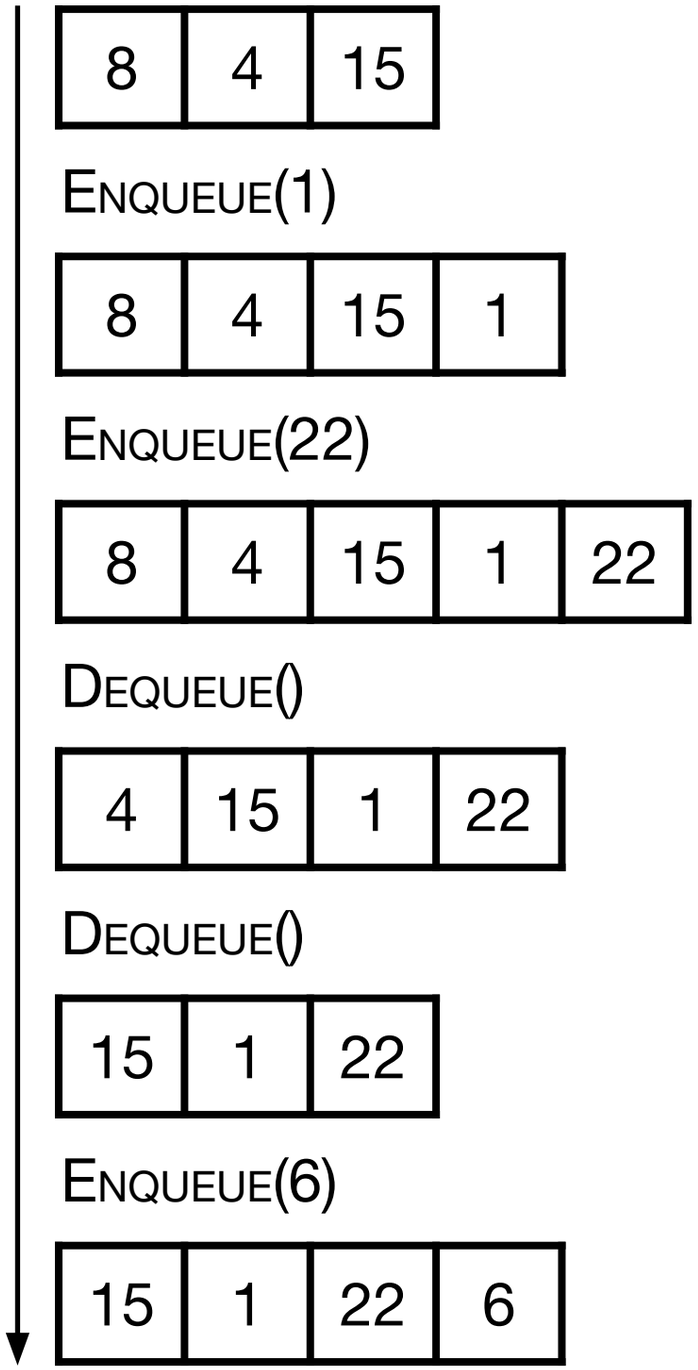
Stapel

- Stapel. Verwalte einen dynamischen Stapel S, der folgende Operationen unterstützt:
  - PUSH(x): lege x auf den Stapel S drauf.
  - POP(): entferne und liefere oberstes (=zuletzt eingefügtes) Element von S zurück.
  - ISEMPTY(): liefere true zurück, wenn S leer ist.



Warteschlange

- Warteschlange. Verwalte eine dynamische Warteschlange Q, die folgende Operationen unterstützt:
  - ENQUEUE(x): füge x zu Q hinzu.
  - DEQUEUE(): entferne und liefere das am frühesten eingefügte Element von Q zurück.
  - ISEMPTY(): liefere true zurück, wenn Q leer ist.

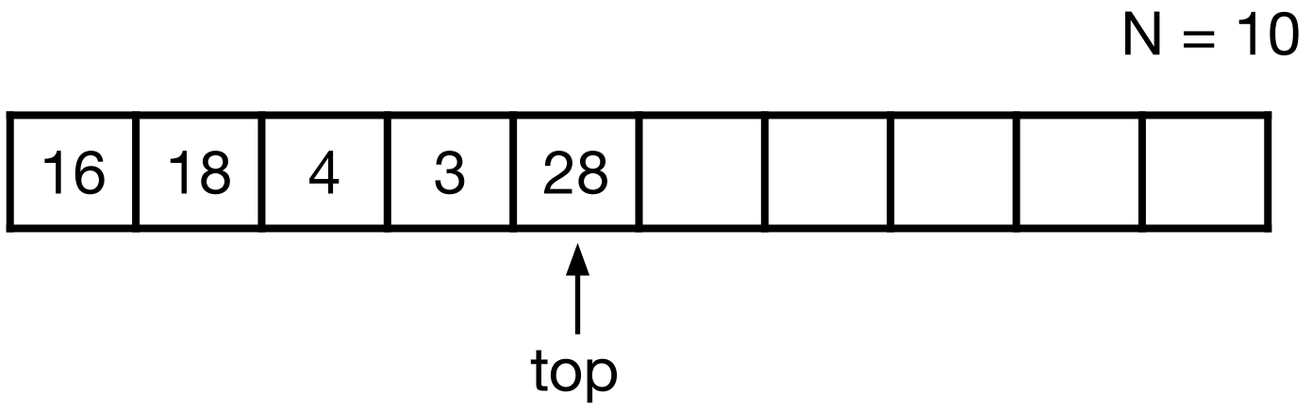


Anwendungen

- Stapel.
  - Virtuelle Maschinen
  - Parser
  - Funktionsaufrufe
  - Backtracking
  - Tiefensuche
- Warteschlangen.
  - Scheduling
  - Buffering
  - Breitensuche

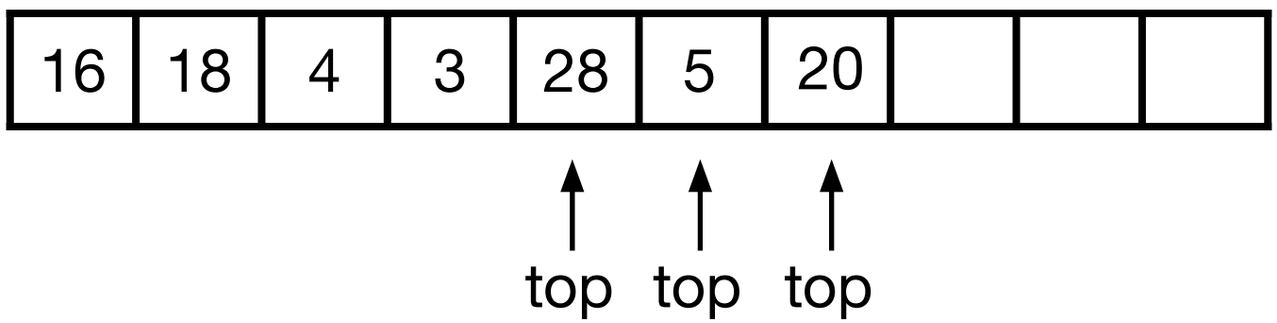
Implementierung von Stapeln als festes Feld

- Stapel. Stapel mit Kapazität N
- Datenstruktur.
  - Feld S[0..N-1]
  - Index top. Anfangs ist top = -1
- Operationen.
  - PUSH(x): Füge x in S[top+1] ein und setze top = top + 1
  - POP(): liefere S[top] zurück und setze top = top - 1
  - ISEMPTY(): liefere true zurück, wenn top = -1 gilt.
  - Teste in PUSH and POP, dass kein Überlauf oder Unterlauf stattfindet.

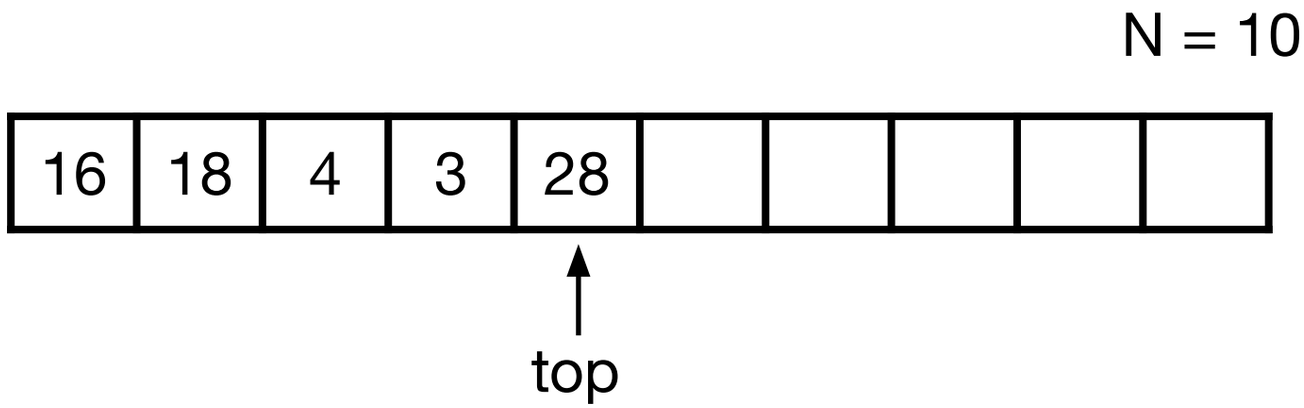


Push 20

Pop



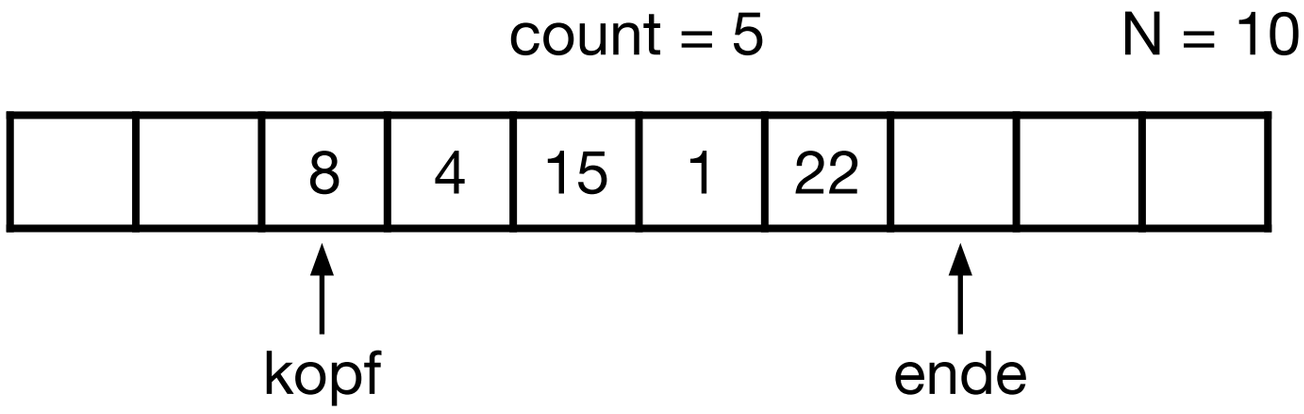
Implementierung von Stapeln als festes Feld



- Zeit.
  - PUSH in Zeit O(1).
  - POP in Zeit O(1).
  - ISEMPTY in Zeit O(1).
- Platz.
  - O(N) Platz.
- Limitierung.
  - Kapazität muss bekannt sein.
  - Platz wird verschwendet.

Implementierung von Warteschlangen

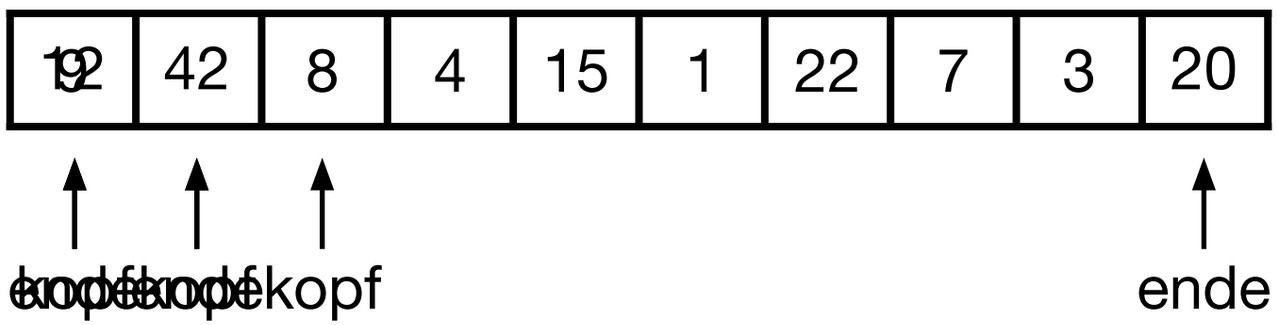
- Warteschlange. Warteschlange with Kapazität N.
- Datenstruktur.
  - Feld Q[0..N-1]
  - Indizes kopf und ende, und ein Zähler count.
- Operationen.
  - ENQUEUE(x): füge x in Q[ende] hinzu, aktualisiere count und ende zyklisch.
  - DEQUEUE(): gib Q[kopf] zurück, aktualisiere count und kopf zyklisch.
  - ISEMPTY(): gib true zurück, wenn count = 0.
  - Teste in DEQUEUE and ENQUEUE, dass kein Überlauf oder Unterlauf stattfindet.



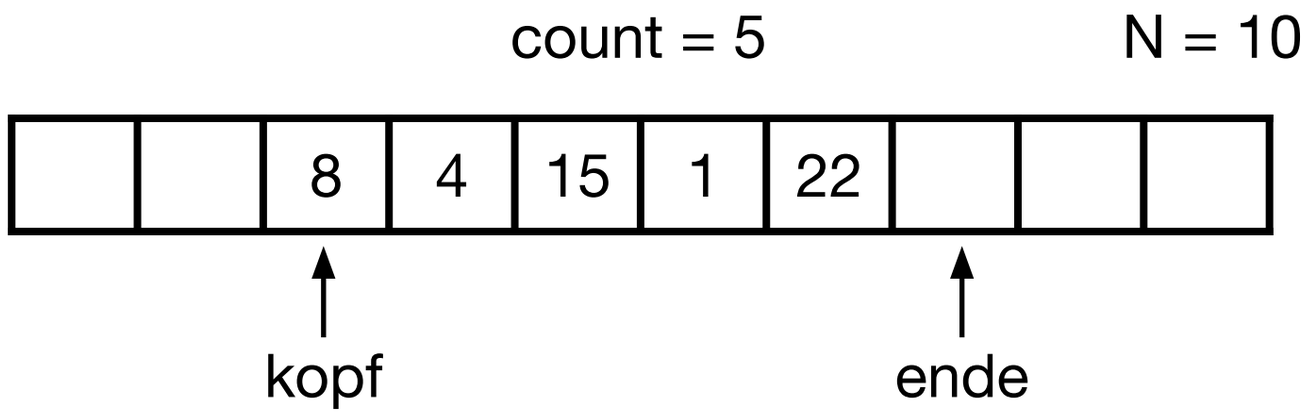
DEQUEUE()

ENQUEUE(20)

ENQUEUE(9)



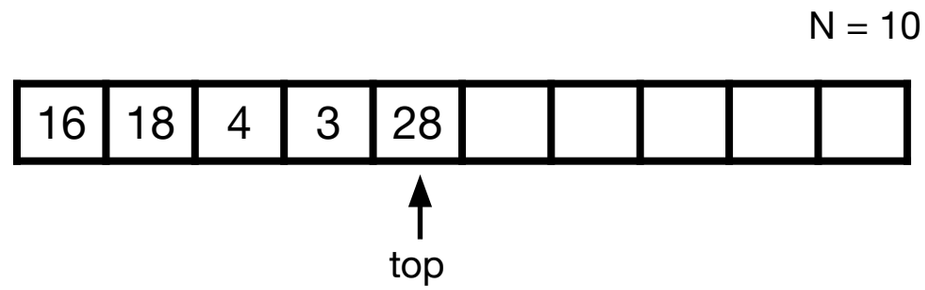
Implementierung von Warteschlangen



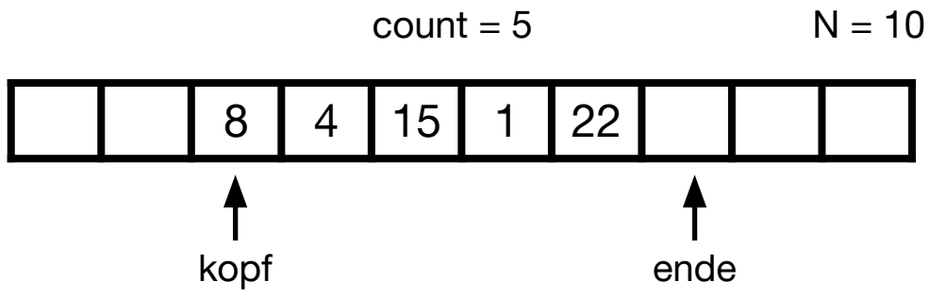
- Zeit.
  - ENQUEUE in Zeit O(1).
  - DEQUEUE in Zeit O(1).
  - ISEMPTY in Zeit O(1).
- Platz.
  - O(N) Platz.
- Limitierung.
  - Kapazität muss bekannt sein.
  - Platz wird verschwendet.

Stapel und Warteschlangen

- Stapel.
  - Zeit. PUSH, POP, ISEMPTY in Zeit O(1).
  - Platz. O(N)



- Warteschlange.
  - Zeit. ENQUEUE, Dequeue, ISEMPTY in Zeit O(1).
  - Platz. O(N)



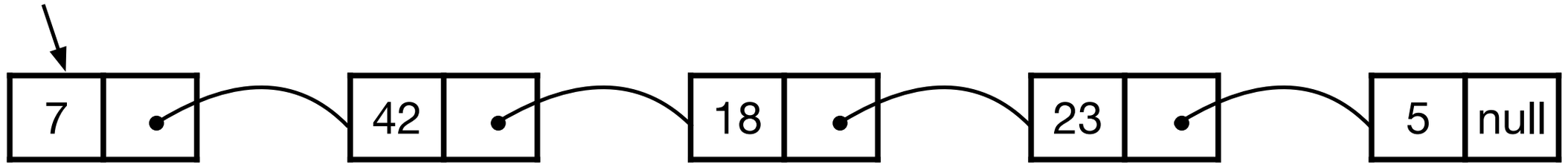
- Frage. Geht's auch in konstanter Zeit mit linearem Platz in der Zahl der Elemente?

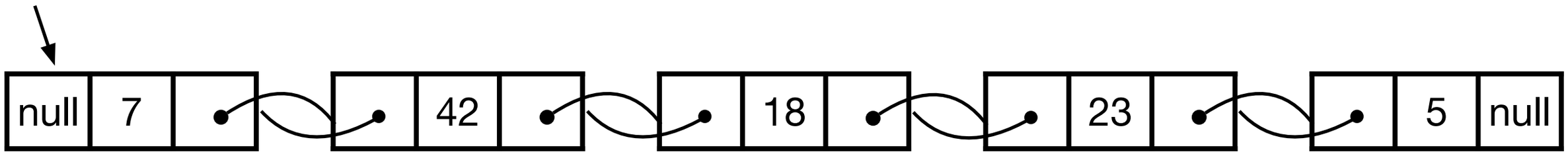
Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder

Verkettete Listen

- Datenstruktur. Verwaltet dynamische Sequenz von Elementen in linearem Platz.
- Reihenfolge wird durch Zeiger/Referenzen festgelegt, die Kettenglied heißen.
- Schnell: Elemente einfügen und löschen, oder zusammenhängende Unterliste.
- einfach verkettet vs. doppelt verkettet.

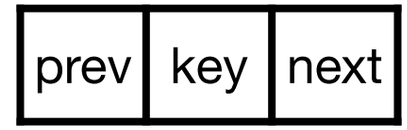


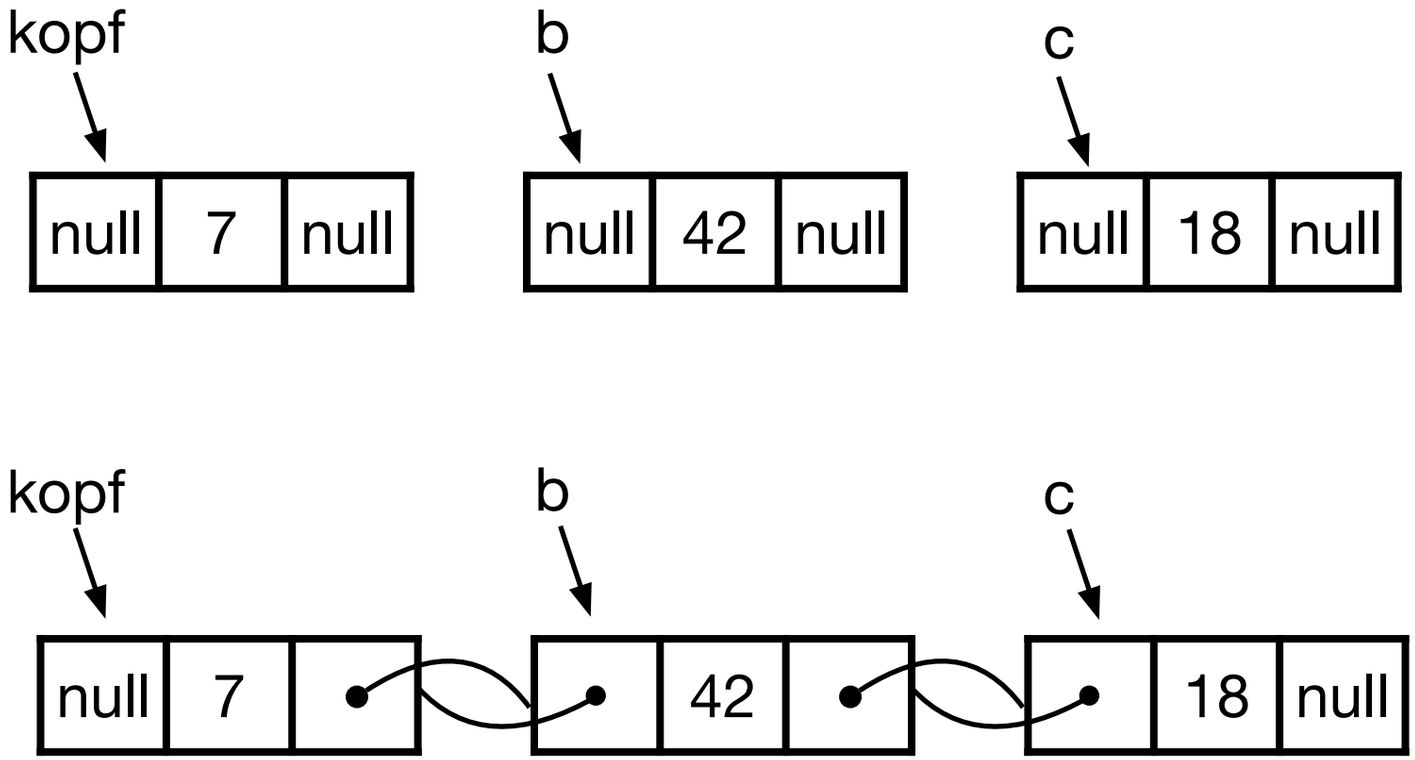


Verkettete Listen

- doppelt verkettete Listen in Java.

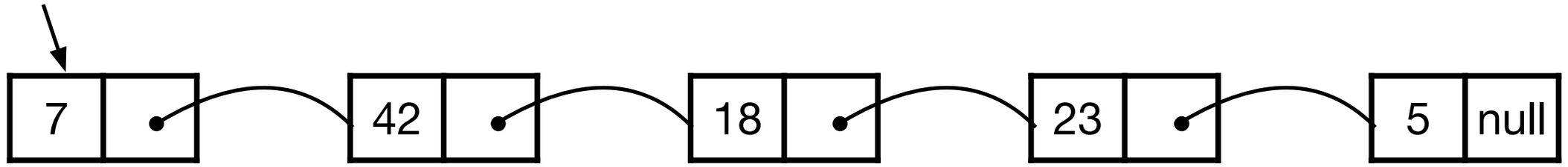
```
class Node {
   int key;
   Node next;
   Node prev;
}

Node kopf = new Node();
Node b = new Node();
Node c = new Node();
kopf.key = 7;
b.key = 42;
c.key = 18;

kopf.prev = null;
kopf.next = b;
b.prev = kopf;
b.next = c;
c.prev = b;
c.next = null;
```

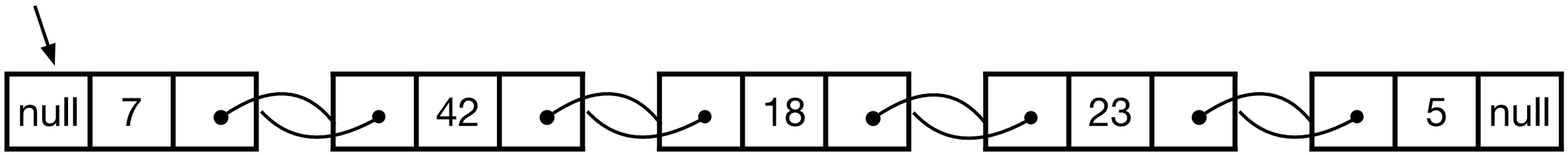




Verkettete Listen

- Einfache Operationen.
  - SEARCH(kopf, k): gib Element mit Schlüssel k zurück / null falls es nicht existiert.
  - INSERT(kopf, x): füge Element x vorne in die Liste ein, gib neuen kopf zurück.
  - DELETE(kopf, x): lösche Element x aus der Liste.





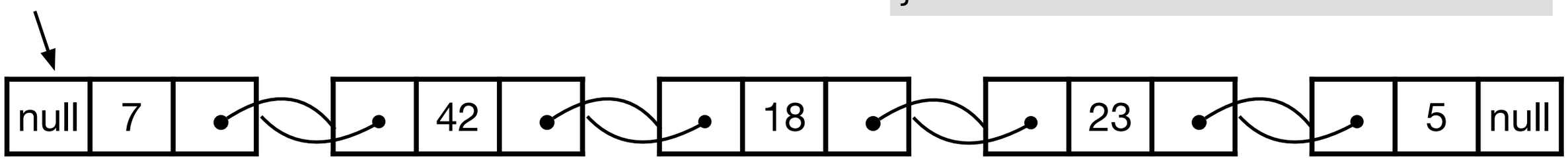
Verkettete Listen

- Operationen in Java.

```
Node Search(Node kopf, int value) {
    Node x = kopf;
    while (x != null) {
        if (x.key == value) return x;
        x = x.next;
    }
    return null;
}
```

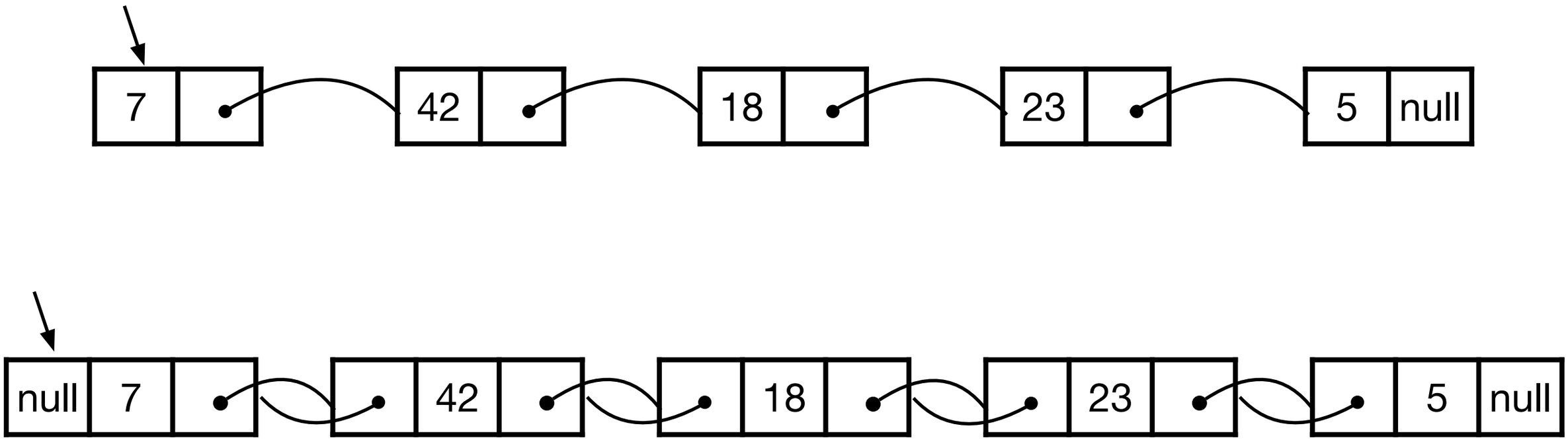
```
Node Insert(Node kopf, Node x) {
    x.prev = null;
    x.next = kopf;
    kopf.prev = x;
    return x;
}
```

```
Node Delete(Node kopf, Node x) {
    if (x.prev != null)
        x.prev.next = x.next;
    else kopf = x.next;
    if (x.next != null)
        x.next.prev = x.prev;
    return kopf;
}
```



- Beispiel. Sei p ein neues Element mit Schlüssel 10 und sei q das Element mit Schlüssel 23. Verfolge die Ausführung der folgenden Anweisungen:
  `Search(kopf,18); Insert(kopf,p); Delete(kopf,q);`

Verkettete Listen



- Zeit.
  - SEARCH in Zeit O(n).
  - INSERT and DELETE in Zeit O(1).
- Platz.
  - O(n)

Stapel und Warteschlangen durch verkettete Listen

- Frage. Wie lassen sich Stapel und Warteschlangen effizient implementieren?

- Stapel. Verwalte einen dynamischen Stapel S, der folgende Operationen unterstützt:
  - PUSH(x): lege x auf den Stapel S drauf.
  - POP(): entferne und liefere oberstes (=zuletzt eingefügtes) Element von S zurück.
  - ISEMPTY(): liefere true zurück, wenn S leer ist.

- Warteschlange. Verwalte eine dynamische Warteschlange Q, die folgende Operationen unterstützt:
  - ENQUEUE(x): füge x zu Q hinzu.
  - DEQUEUE(): entferne und liefere das am frühesten eingefügte Element von Q zurück.
  - ISEMPTY(): liefere true zurück, wenn Q leer ist.

Stapel und Warteschlange Implementation

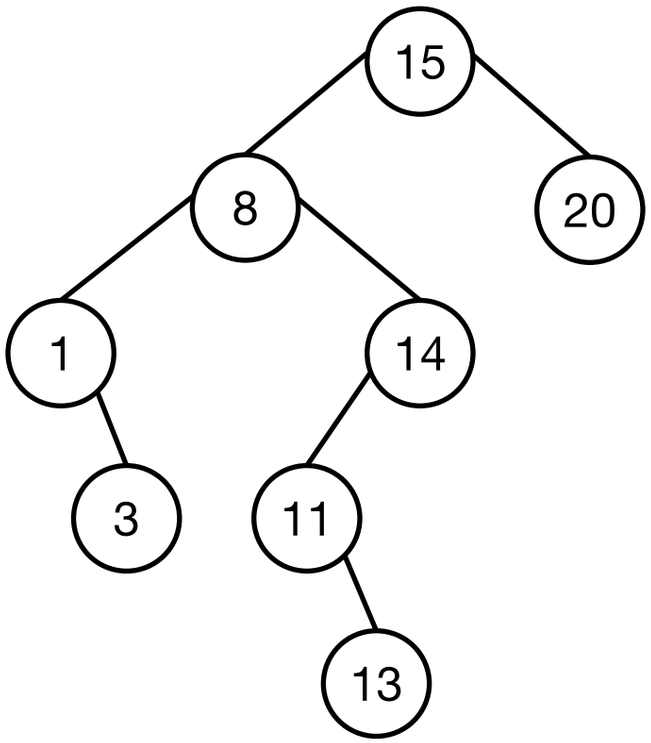
- Stapel und Warteschlangen mit verketteten listen

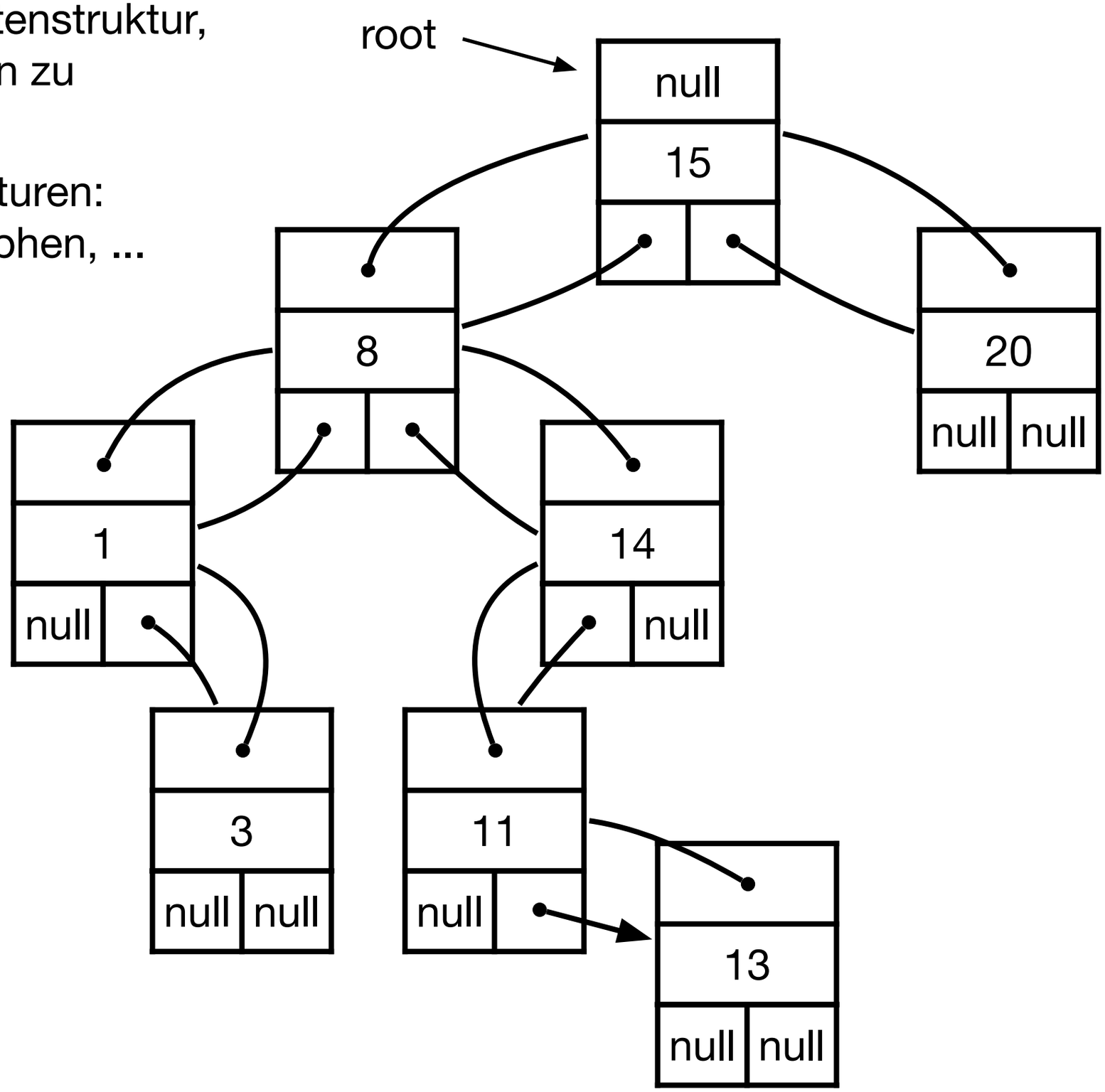
- Stapel.
  - Zeit. PUSH, POP, ISEMPTY in Zeit O(1).
  - Platz. O(n)

- Warteschlange.
  - Zeit. ENQUEUE, DEQUEUE, ISEMPTY in Zeit O(1).
  - Platz. O(n)

Verkettete Listen

- Verkettete Listen. Flexible Datenstruktur, um Sequenzen von Elementen zu verwalten.
- Andere verkettete Datenstrukturen: zyklische Listen, Bäume, Graphen, ...





Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder

Implementierung von Stapeln als Felder

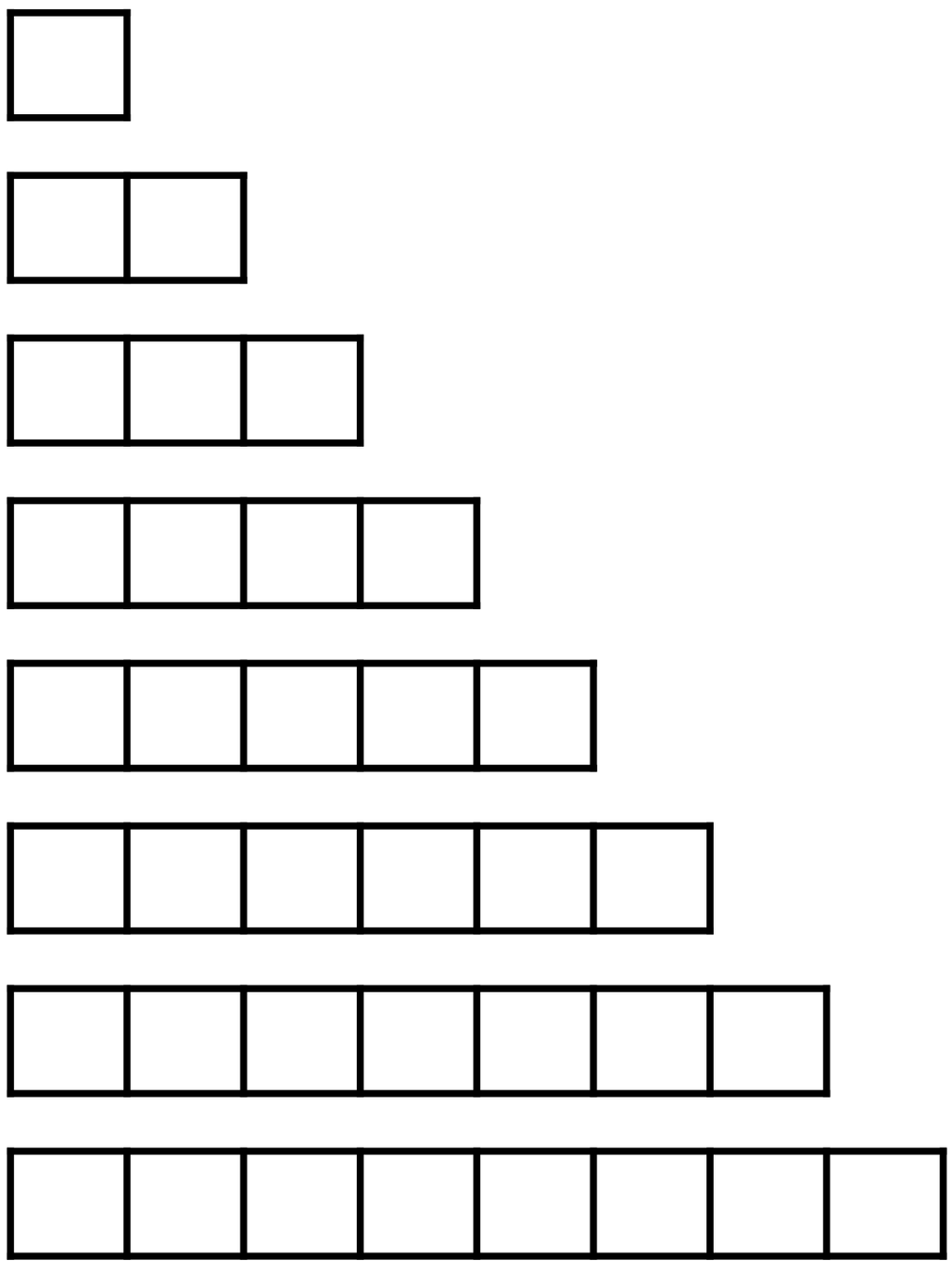
- Frage. Können wir Stapel mit Feldern effizient implementieren?
  - Brauchen wir die feste Kapazität?
  - Ist es in linearem Platz und mit konstant-Zeit Operationen möglich?

Dynamische Felder

- Ziel.
  - Implementiere einen Stapel mit n Elementen durch Feldern in O(n) Platz.
  - So schnell wie möglich.
  - Nur Push. Ignoriere Pop und isEmpty erstmal.

- Lösung 1.
  - Starte mit Feld der Größe 1.
  - Push(x):
    - Allokiere neues Feld der Größe +1.
    - Verschiebe alle Elemente ins neue Feld.
    - Lösche altes Feld.

PUSH 5 2 1 8 7 15 24 2

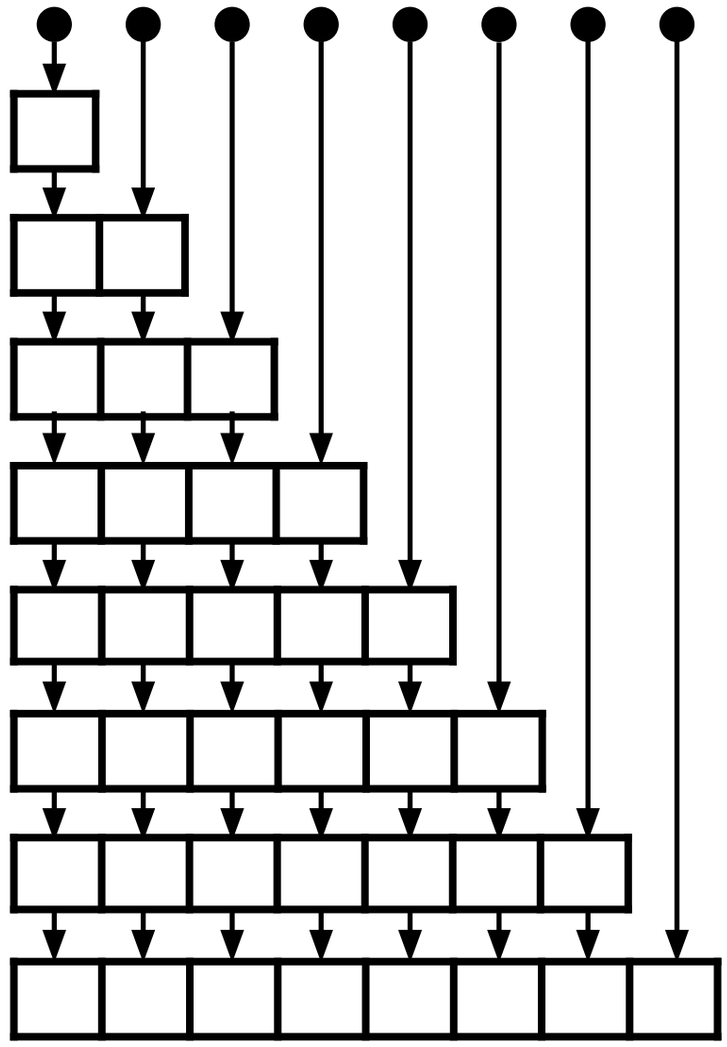


Dynamische Felder

- PUSH(x):
  - Allokiere neues Feld der Größe +1.
  - Verschiebe alle Elemente ins neue Feld.
  - Lösche altes Feld.

- Zeit. Zeit für n PUSH-Operationen?
  - i-te PUSH-Operation braucht Zeit O(i).
  - ⇒ Gesamtzeit ist $1 + 2 + 3 + 4 + \ldots + n = O(n^2)$
- Platz. O(n)

- Frage. Geht das besser?

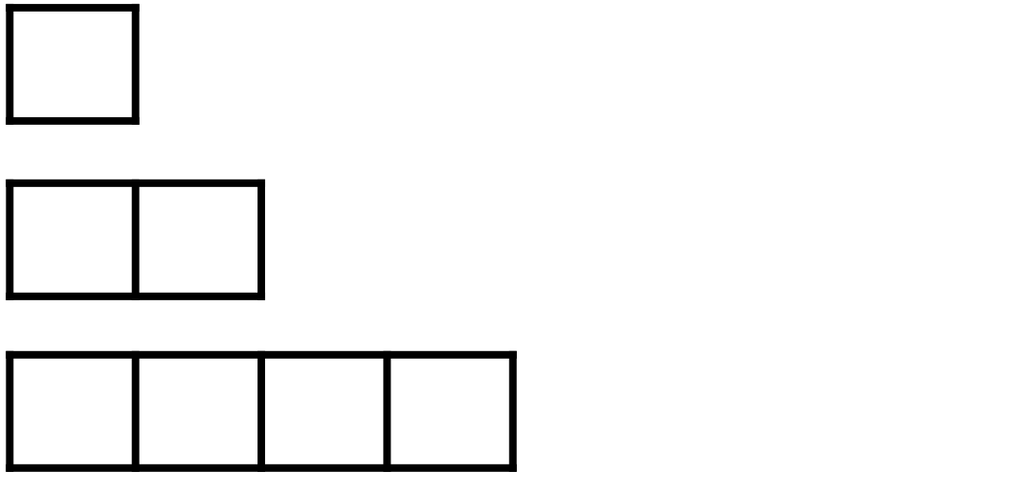


Dynamische Felder

- Idee. Vergrößere das Feld nur manchmal.

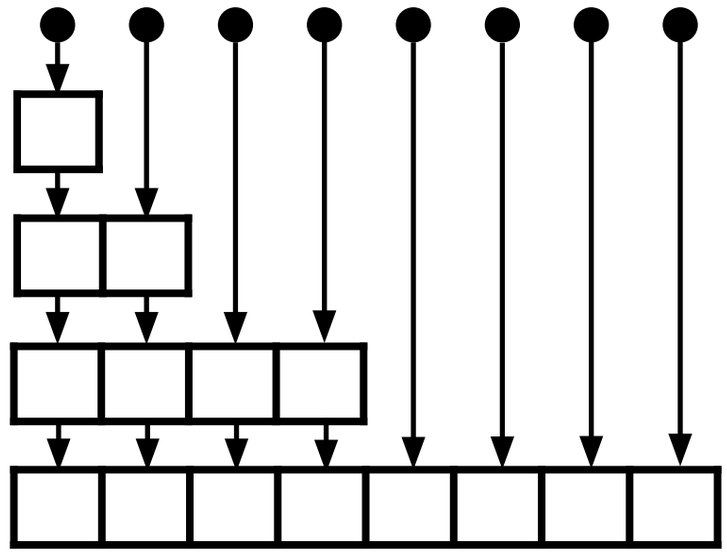
- Lösung 2.
  - Starte mit Feld der Größe 1.
  - PUSH(x):
    - Wenn das Feld voll ist:
      - Allokiere neues Feld der doppelten Größe.
      - Verschiebe alle Elemente ins neue Feld.
      - Lösche altes Feld.

PUSH 5 2 1 8 7 15 24 2



Dynamische Felder

- PUSH(x):
  - Wenn das Feld voll ist:
    - Allokiere neues Feld der doppelten Größe.
    - Verschiebe alle Elemente ins neue Feld.
    - Lösche altes Feld.



- Zeit. Zeit für n PUSH-Operationen?
  - 2^k-tes PUSH braucht Zeit $O(2^k)$.
  - Alle anderen PUSH-Operationen brauchen Zeit O(1).
  - ⇒ Gesamtzeit $< 1 + 2 + 4 + 8 + 16 + \ldots + 2^{\log n} + n = O(n)$
- Platz. O(n)

Dynamische Felder

- Stapel durch dynamische Felder.
  - n PUSH-Operationen in O(n) Zeit und Platz.
  - Erweiterbar auf n PUSH, POP und ISEMPTY Operationen in Zeit O(n).
- Wir sagen: Die amortisierte Laufzeit ist O(1) pro Operation.
- (Mit cleveren Tricks lässt sich dieser Ansatz deamortisieren zu O(1) worst-case Zeit pro Operation.)

- Warteschlange mit dynamischem Feld.
  - Ähnlich zu Stapeln.

Stapel und Warteschlangen

Datenstruktur	PUSH	POP	ISEMPTY	Space
Feld mit Kapazität N	O(1)	O(1)	O(1)	O(N)
Verkettete Listen	O(1)	O(1)	O(1)	O(n)
Dynamisches Feld 1	O(n)	O(1)†	O(1)	O(n)
Dynamisches Feld 2	O(1)†	O(1)†	O(1)	O(n)
Dynamisches Feld 3	O(1)	O(1)	O(1)	O(n)

† = amortisiert

Elementare Datenstrukturen

- Datenstrukturen
- Stapel und Warteschlangen
- Verkettete Listen
- Dynamische Felder