

# Introduktion til datastrukturer

---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

# Introduktion til datastrukturer

---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

# Datastrukturer

---

- **Datastruktur.** Metode til at organisere data så det kan søges i/tilgås/manipuleres effektivt.
- **Mål.**
  - Hurtig
  - Kompakt
- **Terminologi.**
  - **Abstrakt** vs. **konkret** datastruktur.
  - **Dynamisk** vs. **statisk** datastruktur.

# Introduktion til datastrukturer

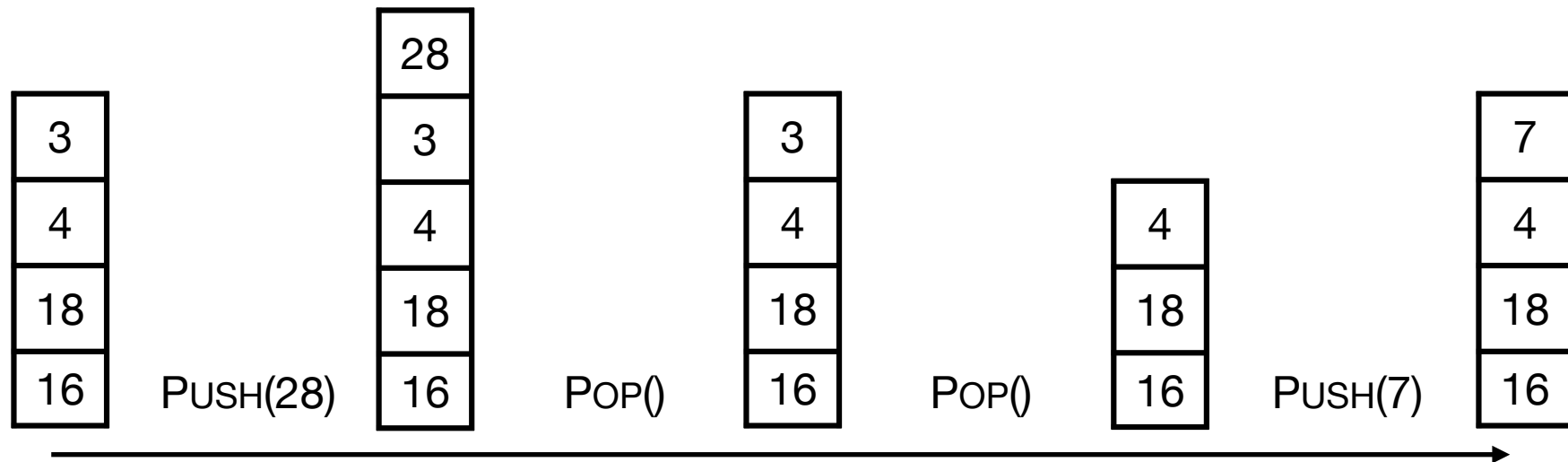
---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

# Stak

---

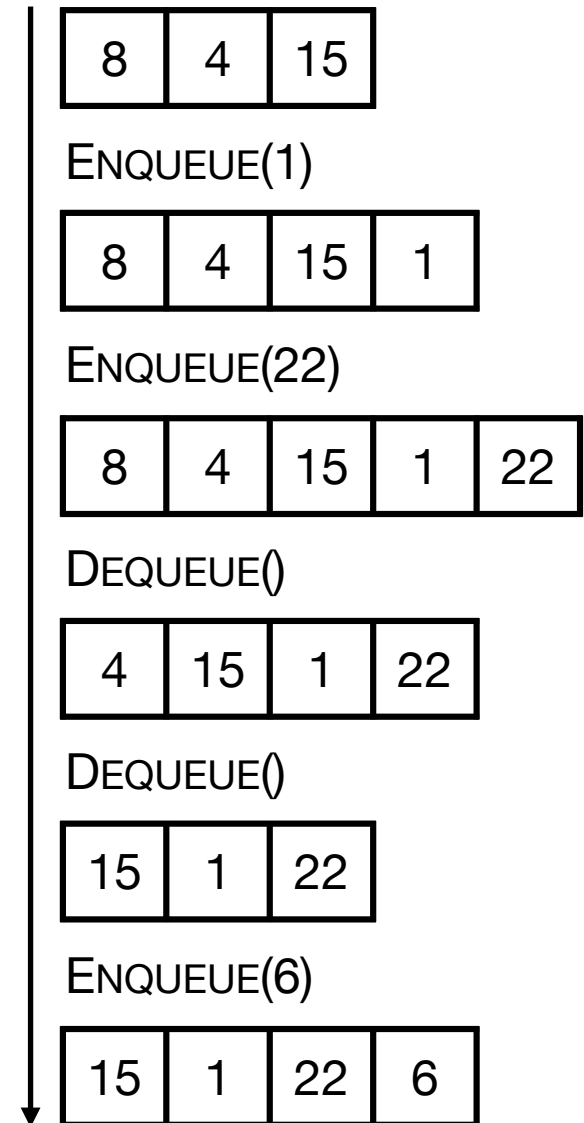
- **Stak.** Vedligehold en dynamisk sekvens (stakken) S af elementer under følgende operationer.
  - PUSH(x): tilføj et nyt element x til S.
  - POP(): fjern og returner det **seneste** tilføjede element i S.
  - IEMPTY(): returner sand hvis S ikke indeholder nogle elementer.



# Kø

---

- **Kø**. Vedligehold en dynamisk sekvens (køen) K af elementer under følgende operationer.
  - ENQUEUE(x): tilføj et nyt element x til K
  - DEQUEUE(): fjern og returner det **tidligst** tilføjede element i K.
  - ISEMPTY(): returner sand hvis K ikke indeholder nogle elementer.



# Anvendelser

---

- Stakke.

- Virtuelle maskiner
- Parsing
- Funktionskald
- Backtracking

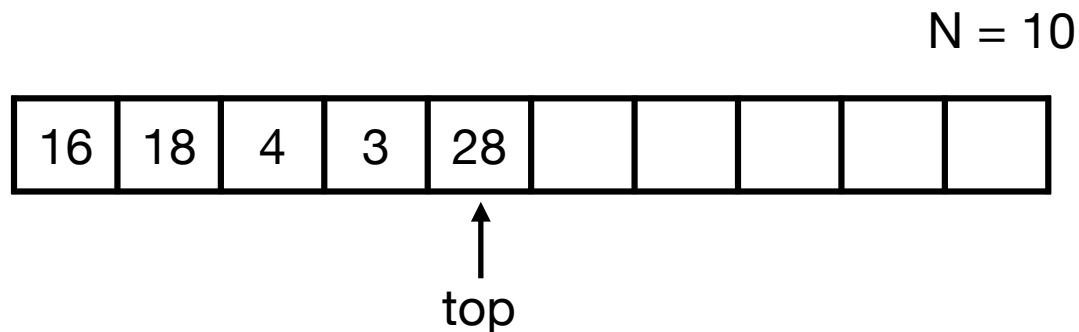
- Køer.

- Skedulering af processer
- Buffering
- Breddeførst søgning

# Implementation af stak med tabel

---

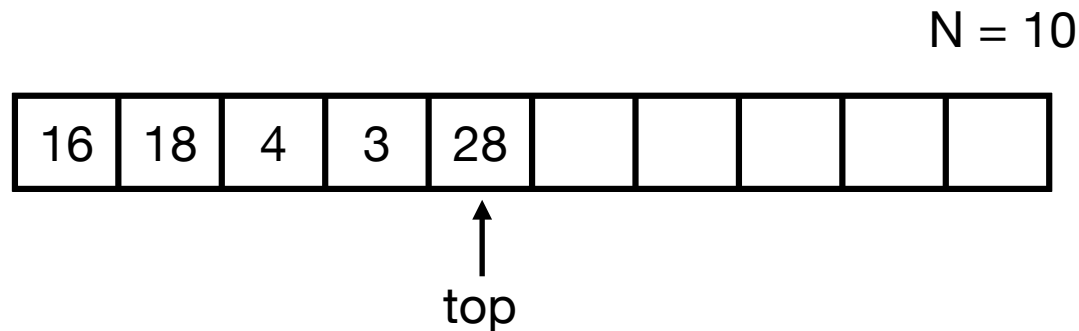
- **Stak.** Stak med **kapacitet** N vha. tabel.
- **Datastruktur.**
  - Tabel  $S[0..N-1]$
  - Index  $top$  i S.
- **Operationer.**
  - **PUSH(x):** Tilføj x på  $S[top+1]$ , sæt  $top = top + 1$
  - **POP():** returner  $S[top]$ , sæt  $top = top - 1$
  - **ISEMPTY():** returner sand hvis og kun hvis  $top = -1$ .
  - Tjek for overløb og underløb i **PUSH** og **POP**.





# Implementation af stak med tabel

---

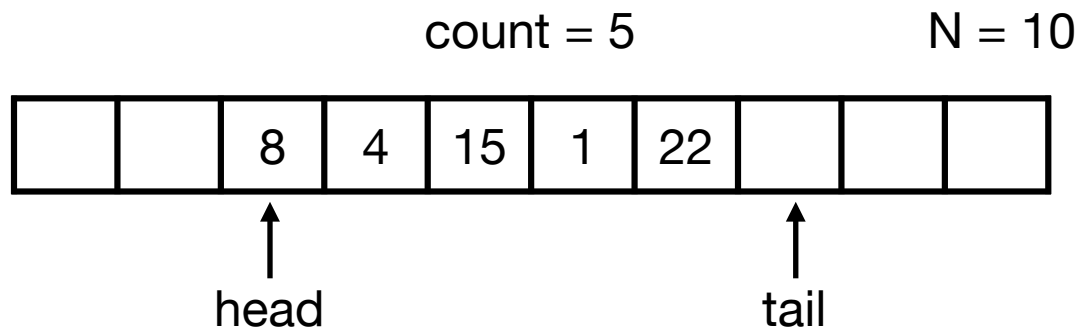


- Tid.
  - PUSH i  $\Theta(1)$  tid.
  - POP i  $\Theta(1)$  tid.
  - ISEEMPTY i  $\Theta(1)$  tid.
- Plads.
  - $\Theta(N)$  plads.
- Mangler.
  - Vi skal kende kapacitet N fra start.
  - Vi spilder plads når antal elementer er  $\ll N$ .

# Implementation af kø med tabel

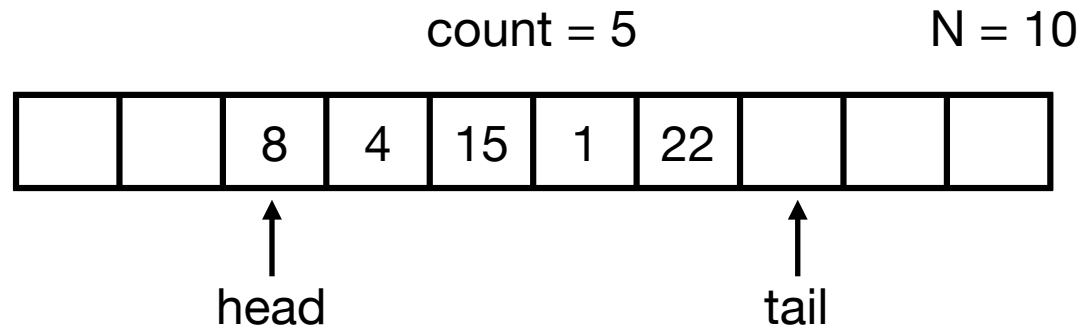
---

- **Kø.** Kø med **kapacitet**  $N$  vha. tabel.
- **Datastruktur.**
  - Tabel  $S[0..N-1]$
  - Indeks  $head$  (tidligst indsatte element) og  $tail$  (næste ledige element) i  $S$  og tæller  $count$  (antal elementer i kø).
- **Operationer.**
  - $ENQUEUE(x)$ : Tilføj  $x$  på  $S[tail]$ , opdater  $count$  og  $tail$  **cyklisk**.
  - $DEQUEUE()$ : returner  $S[head]$ , opdater  $count$  og  $head$  **cyklisk**.
  - $ISEMPTY()$ : returner sand hvis og kun hvis  $count = 0$ .
  - Tjek for overløb og underløb i  $DEQUEUE$  og  $ENQUEUE$ .



# Implementation af kø med tabel

---

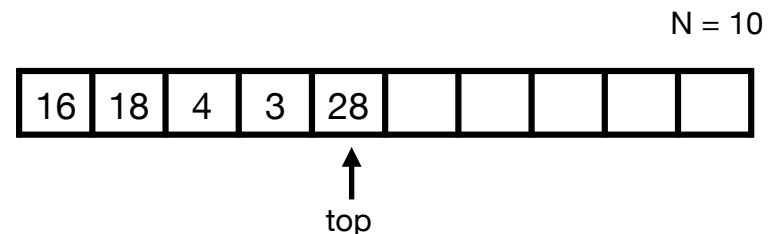


- Tid.
  - ENQUEUE i  $\Theta(1)$  tid.
  - DEQUEUE i  $\Theta(1)$  tid.
  - ISEEMPTY i  $\Theta(1)$  tid.
- Plads.
  - $\Theta(N)$  plads.
- Mangler.
  - Vi skal kende kapacitet N fra start.
  - Vi spilder plads når antal elementer er  $\ll N$ .

# Stakke og køer

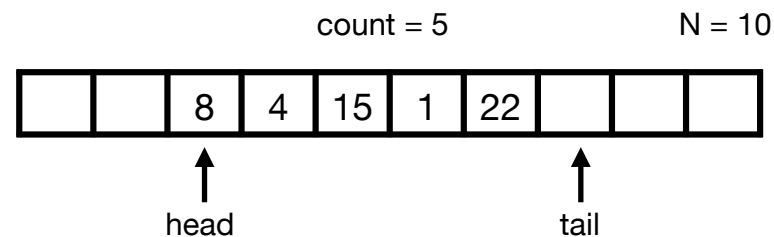
- **Stak.**

- **Tid.** PUSH, POP, ISEMPTY i  $\Theta(1)$  tid.
- **Plads.**  $\Theta(N)$



- **Kø.**

- **Tid.** ENQUEUE, Dequeue, ISEMPTY i  $\Theta(1)$  tid.
- **Plads.**  $\Theta(N)$



- **Udfordring.** Kan vi komme ned på lineær plads med samme tid?

# Introduktion til datastrukturer

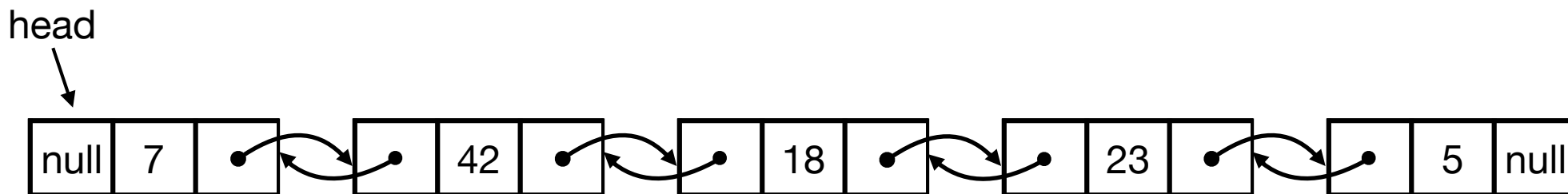
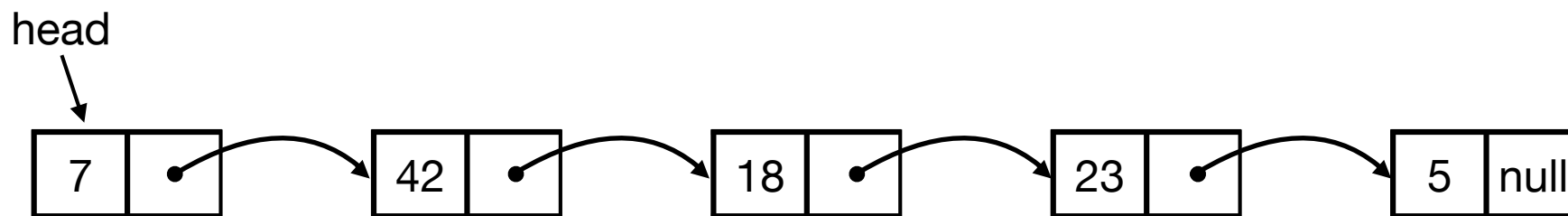
---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

# Hægtede lister

- Hægtede lister.

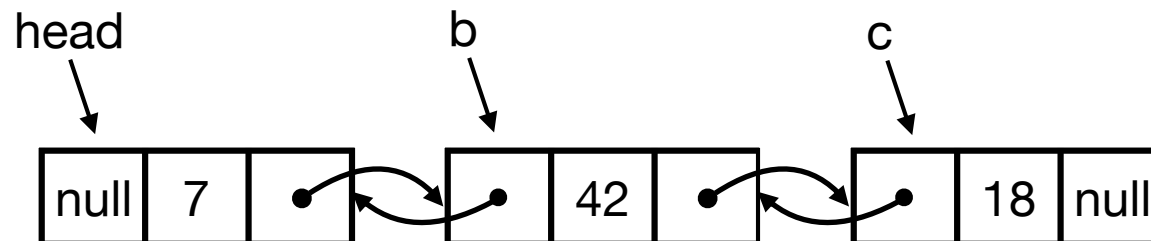
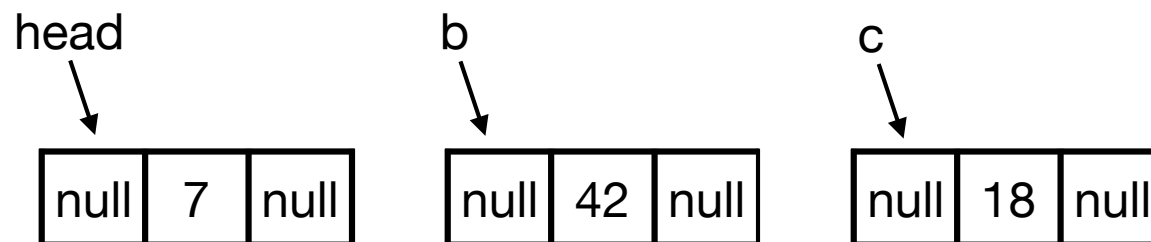
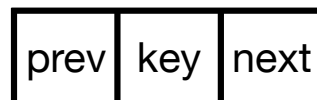
- Datastruktur til at vedligeholde en **dynamisk** sekvens af elementer i lineær plads.
- Rækkefølge af elementer bestemt af referencer/pegere kaldet **hægter**.
- Effektivt at indsætte og fjerne elementer eller sammenhængende dele af elementer.
- **Dobbelt-hægtede** vs **enkelt-hægtede**.



# Hægtede lister

- Dobbelt-hægtede lister i Java.

```
class Node {  
    int key;  
    Node next;  
    Node prev;  
}  
  
Node head = new Node();  
Node b = new Node();  
Node c = new Node();  
head.key = 7;  
b.key = 42;  
c.key = 18;  
  
head.prev = null;  
head.next = b;  
b.prev = head;  
b.next = c;  
c.prev = b;  
c.next = null;
```

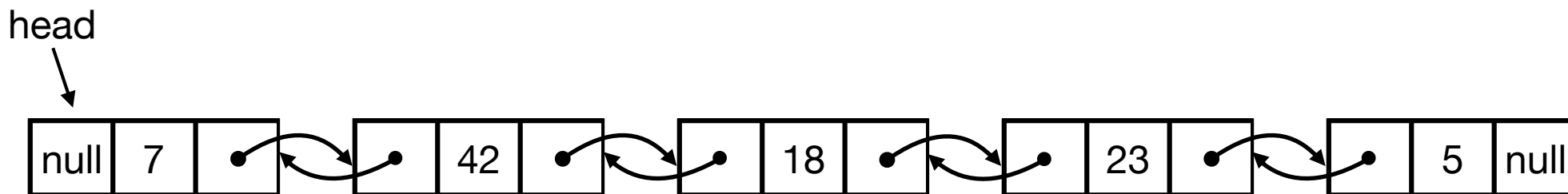
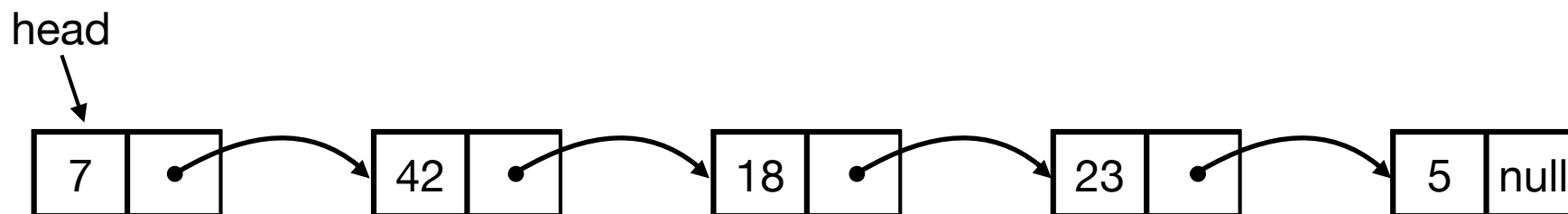


# Hægtede lister

---

- Simple operationer.

- SEARCH(head, k): returner knude med værdi k i listen. Returner null hvis den ikke findes.
- INSERT(head, x): indsæt knude x i starten af listen. Returner ny head.
- DELETE(head, x): fjern knude x i listen.





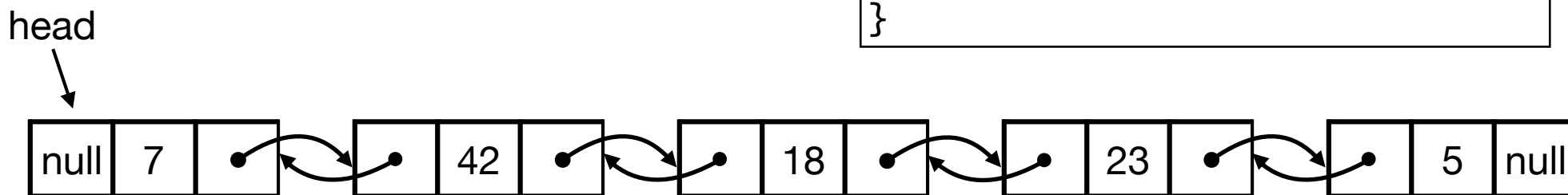
# Hægtede lister

- Operationer på dobbelthægtet liste i Java.

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

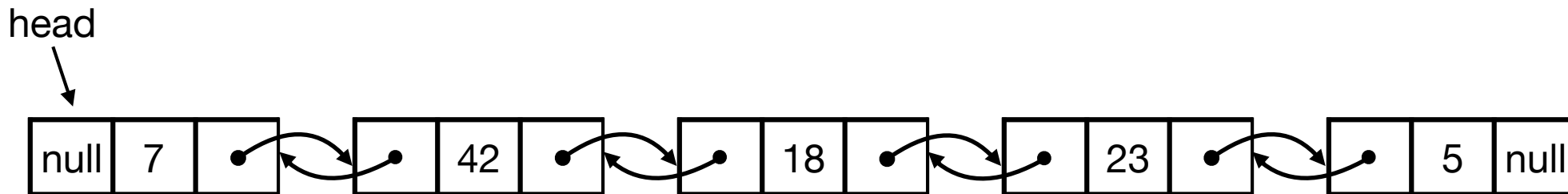
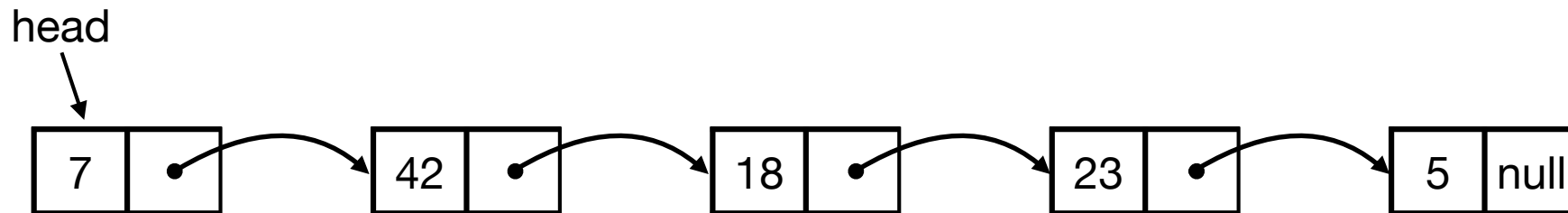
```
Node Insert(Node head, Node x) {  
    x.prev = null;  
    x.next = head;  
    head.prev = x;  
    return x;  
}
```

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



- **Opgave.** Lad p være en ny knude med værdien 10 og lad q være knuden i listen med værdi 23. Håndkør Search(head, 18), Insert(head, p) og Delete(head, q).

# Hægtede lister



- **Tid.** Hvor hurtigt kører operationerne?
  - SEARCH i  $\Theta(n)$  tid
  - INSERT og DELETE i  $\Theta(1)$  tid.
- **Plads.**
  - $\Theta(n)$

# Implementation af stak og kø med hængte liste

---

- **Opgave.** Overvej hvordan man kan implementere stakke og køer med hængte liste effektivt.
- **Stak.** Vedligehold en dynamisk sekvens (stakken) S af elementer under følgende operationer.
  - PUSH(x): tilføj et nyt element x til S.
  - POP(): fjern og returner det **seneste** tilføjede element i S.
  - ISEMPTY(): returner sand hvis S ikke indeholder nogle elementer.
- **Kø.** Vedligehold en dynamisk sekvens (køen) K af elementer under følgende operationer.
  - ENQUEUE(x): tilføj et nyt element x til K
  - DEQUEUE(): fjern og returner det **tidligst** tilføjede element i K.
  - ISEMPTY(): returner sand hvis K ikke indeholder nogle elementer.

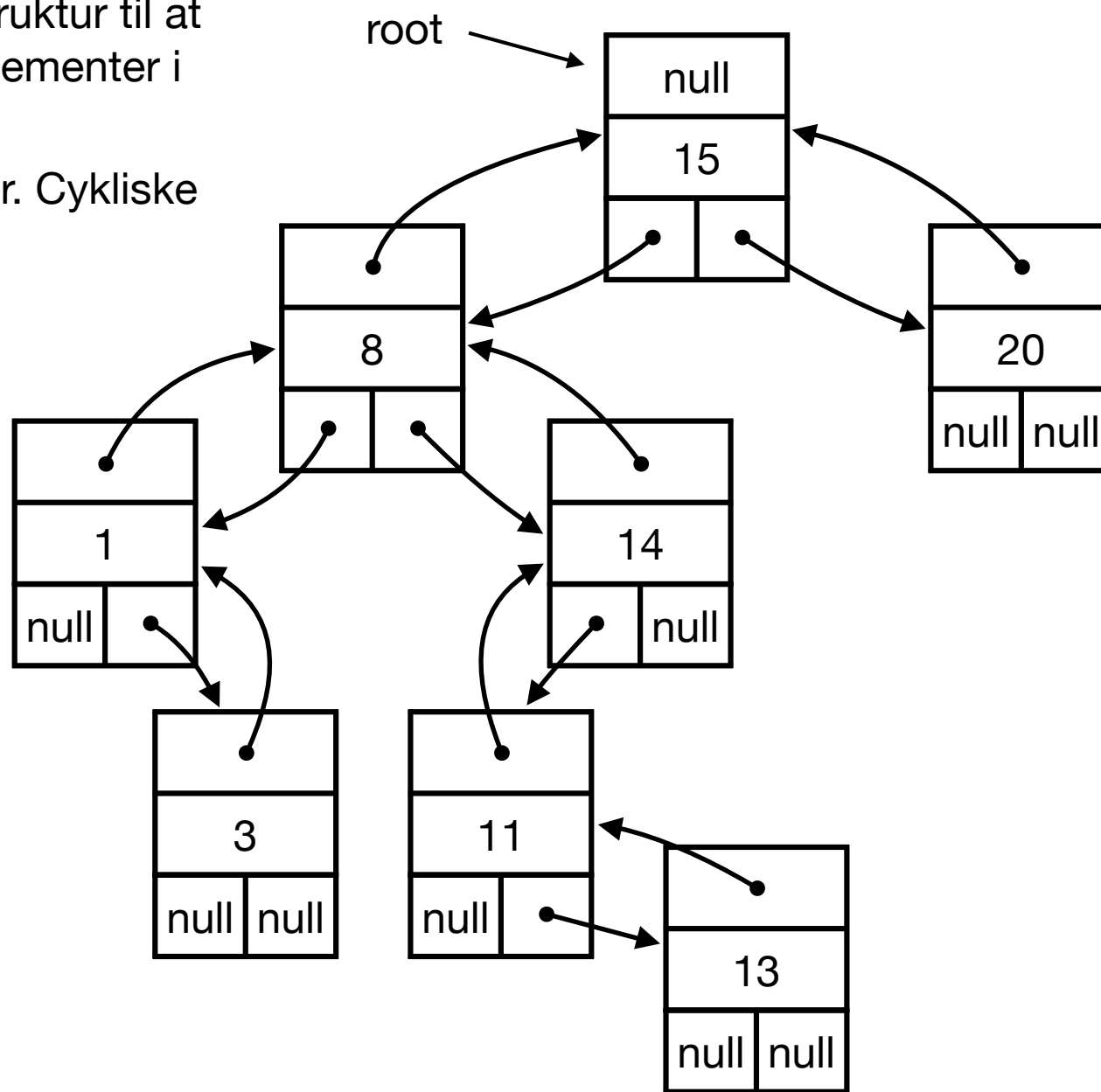
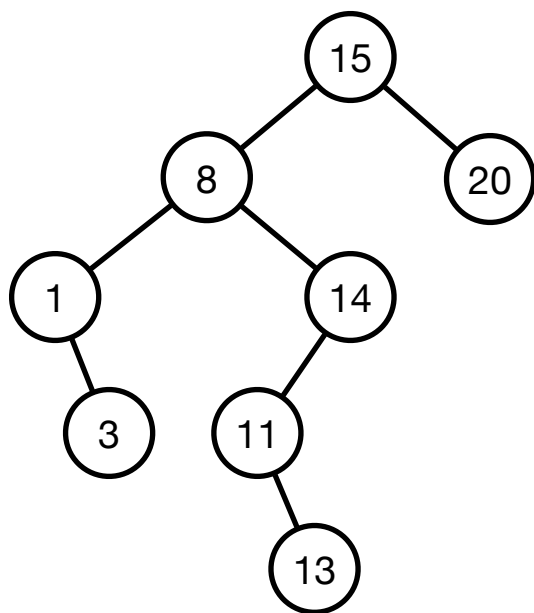
# Stakke og køer

---

- Stak og kø implementeret med hængtet liste
- Stak.
  - Tid. PUSH, POP, ISEEMPTY i  $\Theta(1)$  tid.
  - Plads.  $\Theta(n)$
- Kø.
  - Tid. ENQUEUE, Dequeue, ISEEMPTY i  $\Theta(1)$  tid.
  - Plads.  $\Theta(n)$

# Hægtede lister

- **Hægtet liste.** Fleksibel datastruktur til at vedligeholde en sekvens af elementer i lineær plads.
- Andre hægtede datastrukturer. Cykliske lister, træer, grafer, ...



# Introduktion til datastrukturer

---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

# Stak med dynamisk tabel

---

- **Udfordring.** Kan vi implementere en stak effektivt med tabel(ler)?
  - Behøver vi fastsætte en øvre grænse på antallet af elementer?
  - Kan vi komme ned på lineær plads og konstant tid?

# Dynamiske tabeller

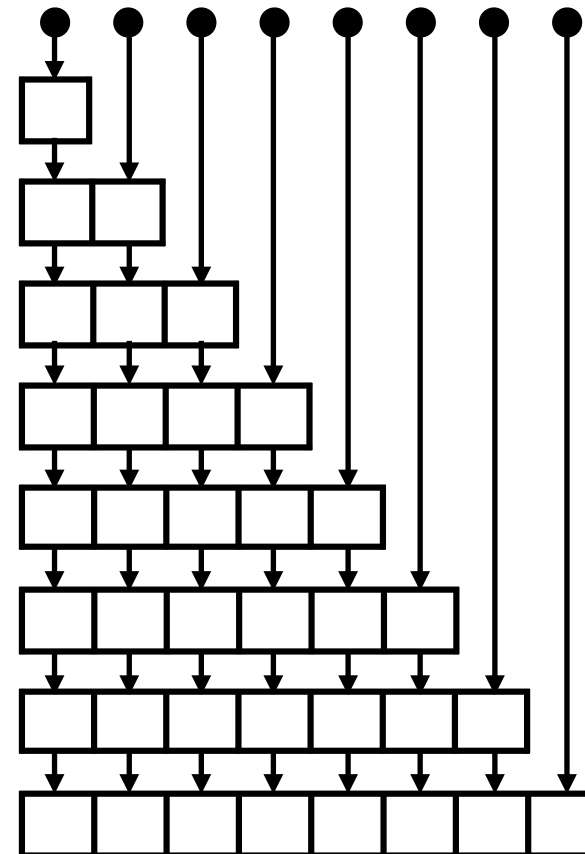
---

- **Mål.**
  - Implementer en stak vha. tabel(ler) i  $\Theta(n)$  plads for  $n$  elementer.
  - Så hurtige operationer som muligt.
  - Kun fokus på PUSH. Ignorerer POP og ISEMPTY indtil videre.
- **Løsning 1.**
  - Start med tabel af størrelse 1.
- PUSH(x):
  - Opret ny tabel af størrelse +1.
  - Flyt alle elementer til ny tabel.
  - Slet gammel tabel.



# Dynamiske tabeller

- PUSH(x):
  - Opret ny tabel af størrelse +1.
  - Flyt alle elementer til ny tabel.
  - Slet gammel tabel.
- Tid. Hvor meget tid tager n PUSH operationer?
  - $\Theta(1)$  for at indsætte element i tabel første gang.
  - $\Theta(i)$  tid for at bygge tabel af størrelse i og flytte i-1 elementer.
- Samlet tid.  $n + 2 + 3 + 4 + \dots + n = \Theta(n^2)$
- Plads.  $\Theta(n)$
- Udfordring. Kan vi gøre noget smartere?



# Dynamiske tabeller

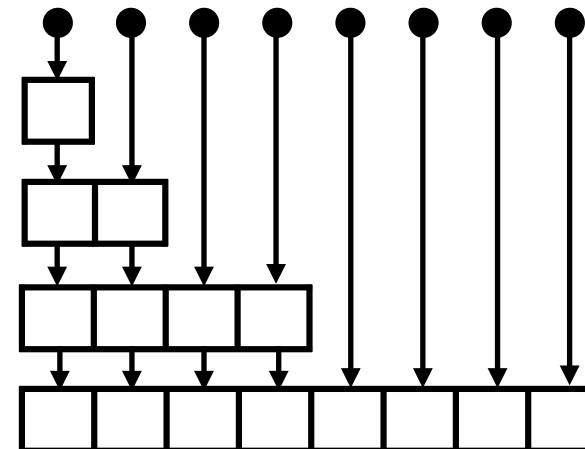
---

- **Ide.** Kopier kun elementer en gang i mellem.
- **Løsning 2.**
  - Start med tabel af størrelse 1.
- **PUSH(x):**
  - Hvis tabel er **fuld** (antallet af elementer i stak er lig tabellens størrelse).
    - Opret ny tabel af **dobbelt** størrelse.
    - Flyt elementer over i ny tabel.
    - Slet gammel tabel.

# Dynamiske tabeller

---

- PUSH(x):
  - Hvis tabel er **fuld** (antallet af elementer i stak er lig tabellens størrelse).
    - Opret ny tabel af **dobbelt** størrelse.
    - Flyt elementer over i ny tabel.
    - Slet gammel tabel.
- **Tid.** Hvor meget tid tager n PUSH operationer?
  - $\Theta(1)$  for at indsætte element i tabel **første** gang.
  - $\Theta(i)$  tid for at bygge tabel af størrelse i og flytte i-1 elementer.
- **Samlet tid.**  $n + 2 + 4 + 8 + 16 \dots + n = \Theta(n)$
- **Plads.**  $\Theta(n)$



# Dynamiske tabeller

---

- **Stak med dynamisk tabel.**
  - $n$  PUSH operationer i  $\Theta(n)$  tid og plads.
  - Kan udvides til  $n$  PUSH, POP og ISEEMPTY operationer i  $\Theta(n)$  tid.
- Køretiden er **amortiseret**  $\Theta(1)$  per operation (een operation kan tage lang tid, men tiden for **enhver** sekvens af  $k$  operationer er  $\Theta(k)$ )
- Med snedige tricks kan man **deamortisere** løsning til  $\Theta(1)$  værstefaldskøretid per operation.
  
- **Kø med dynamisk tabel.**
  - Samme resultater som stak.
- **Global genopbygning.**
  - Dynamisk tabel er eksempel på **global genopbygning** (*global rebuilding*).
  - General teknik til at gøre statiske datastrukturer dynamiske.

# Stakke og køer

Datastruktur	PUSH	POP	ISEMPTY	Plads
Tabel med kapacitet N	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(N)$
Hægtet liste	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
Dynamisk tabel med udvidelse	$\Theta(n)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)$	$\Theta(n)$
Dynamisk tabel med fordobling	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)$	$\Theta(n)$
Dynamisk tabel med deamortiseret fordobling	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$

† = amortiseret

# Introduktion til datastrukturer

---

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller