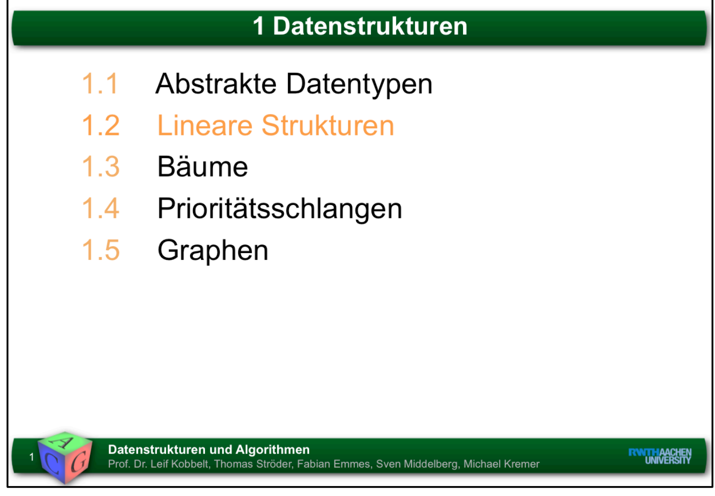
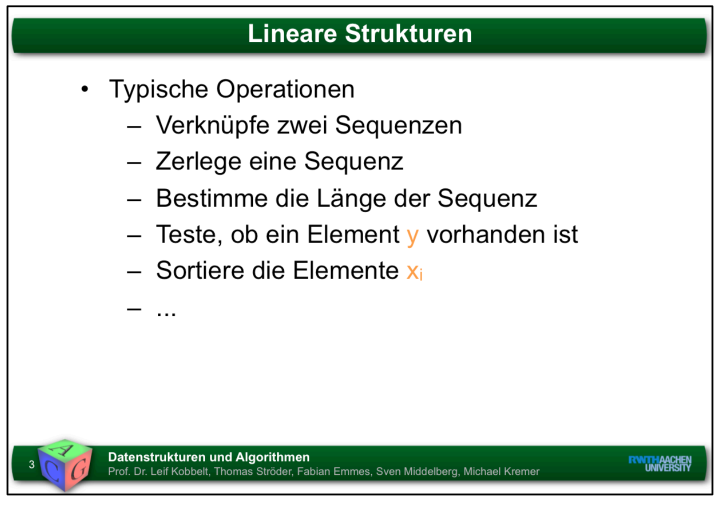


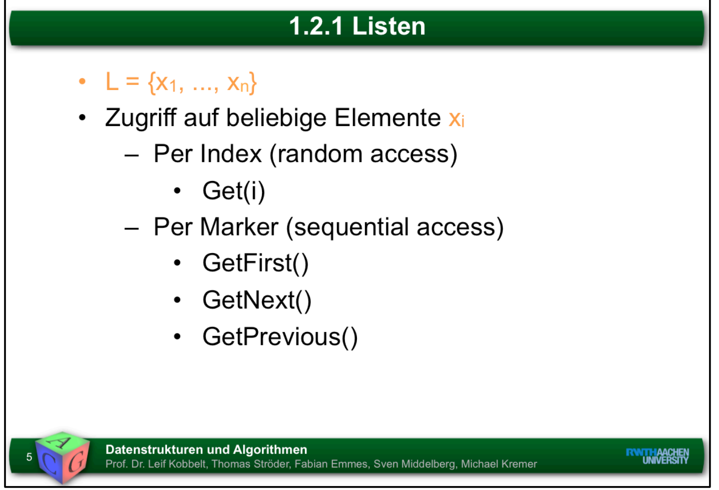
Lineare Strukturen

- Sequenz $\{x_1, ..., x_n\}$ von beliebigen Datenobjekten x_i
- Typische Operationen
  - Füge y am Anfang / am Ende / hinter x_i ein
  - Ersetze x_i durch y
  - Entferne x_i
  - Lese das i-te Element



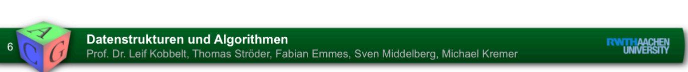
Wie kann man diese Operationen in Form von Axiomen festlegen?

Die (linearen) Datenstrukturen, die wir in dieser Vorlesung betrachten werden.

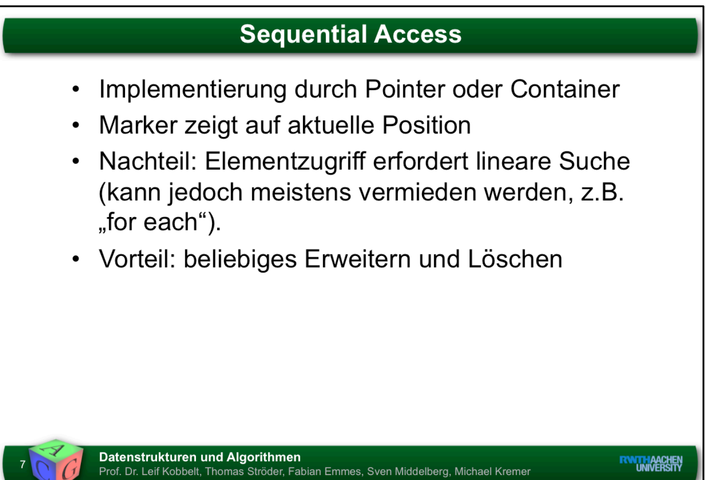


Random Access

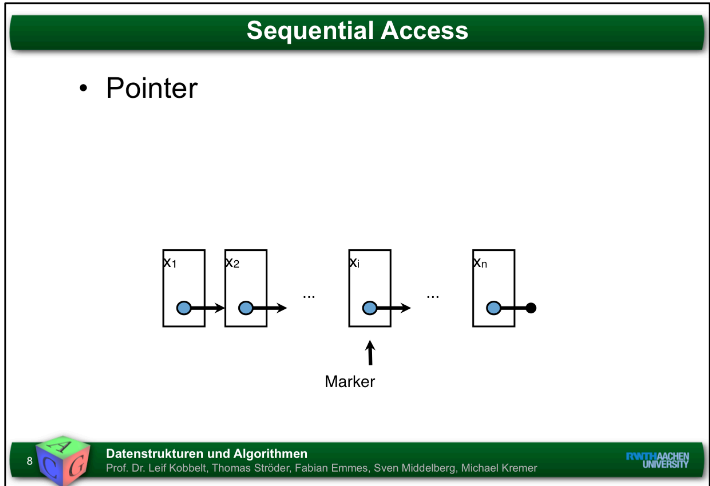
- Implementierung durch Arrays L[ ]
- Get(i) = L[i]
- Nachteile
  - Elemente löschen erzeugt Lücken oder alle Elemente mit höherem Index müssen verschoben werden (garbage collection)
  - Statische Obergrenze für Listenlänge



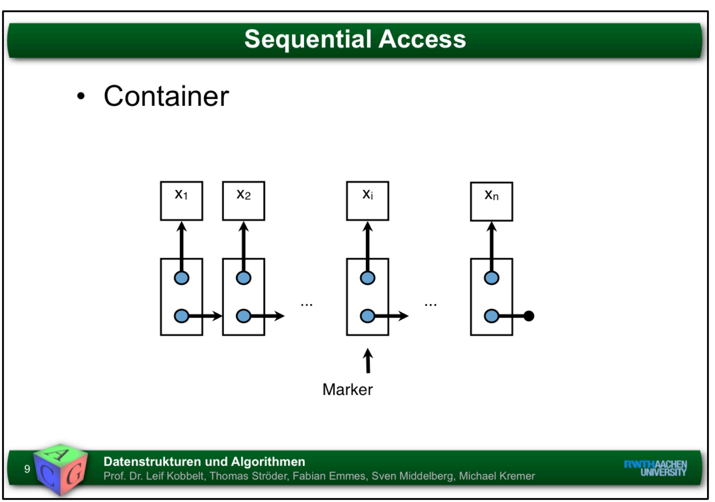
Was ist, wenn Daten nicht mehr in Speicher passen? Mit der Frage werden wir uns später auch beschäftigen. Random Access bedeutet, dass man auf jedes beliebige Element zu jeder Zeit zugreifen kann.



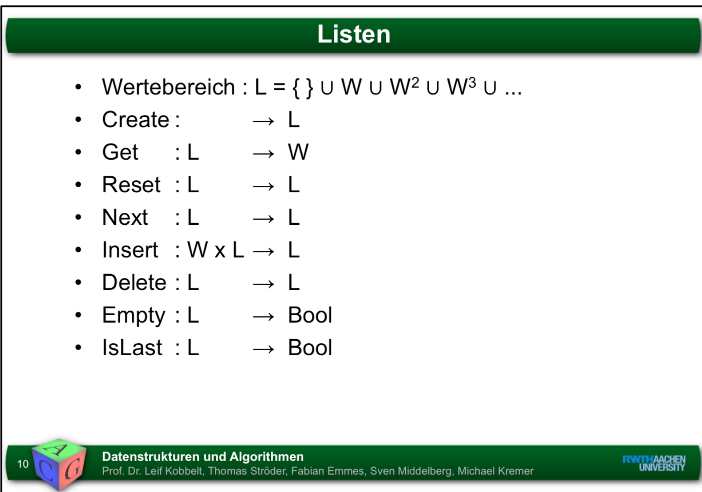
Sequentieller Zugriff bedeutet, dass man nicht auf jedes Element zu jeder Zeit, sondern auf die Daten in linearer Reihenfolge zugreifen kann.

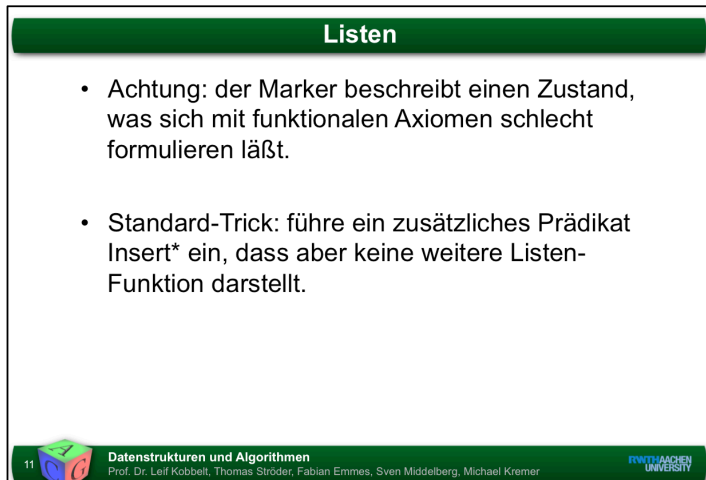


Beispiel einer einfach verketteten Liste. Die blauen Punkte repräsentieren Pointer, die auf einen Speicherbereich, an dem Daten liegen, zeigen.
Entweder werden direkt Werte gespeichert...

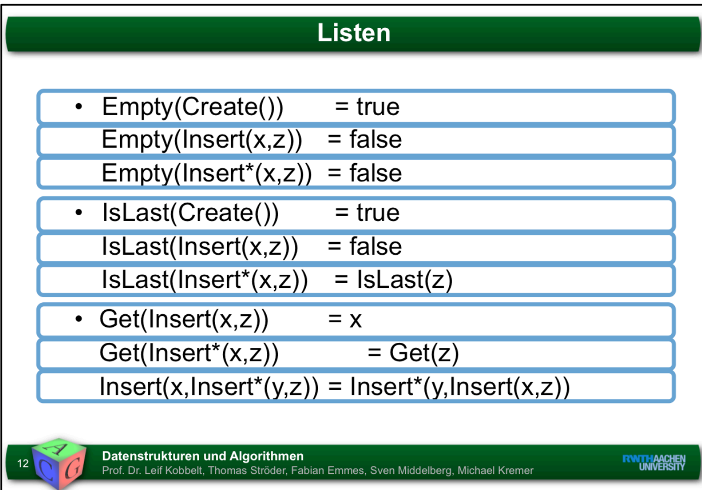


...oder Container-Klassen zum Kapseln von Daten benutzt.

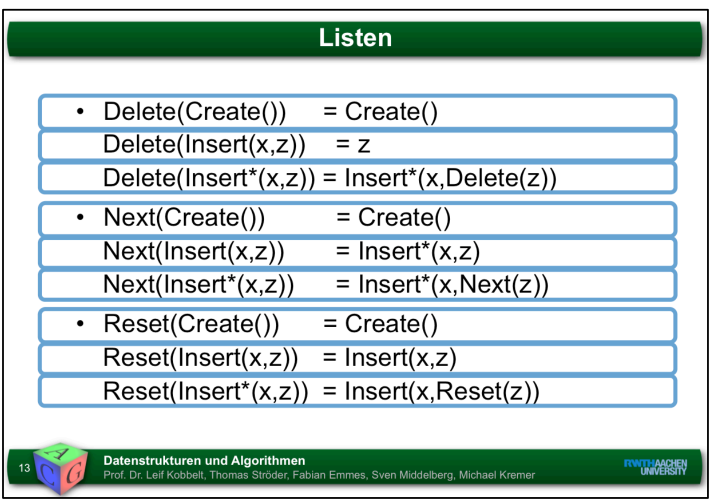




Sternchen dient als „Marker".



Erst Inserts mit Stern, dann Inserts ohne Stern. IsLast(Insert*(x,z)) = IsLast(z) wird solange rekursiv aufgerufen, bis Insert ohne Stern erreicht wurde (Markerposition). Der Marker gibt die aktuelle „Leseposition" an. Das letzte Axiom wird benutzt, um neue Elemente einzufügen (und die alte Markerposition beizubehalten).



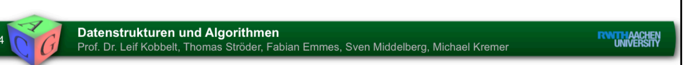
Reset() setzt Markerposition wieder zurück auf Ausgangsposition (äußerstes Element).

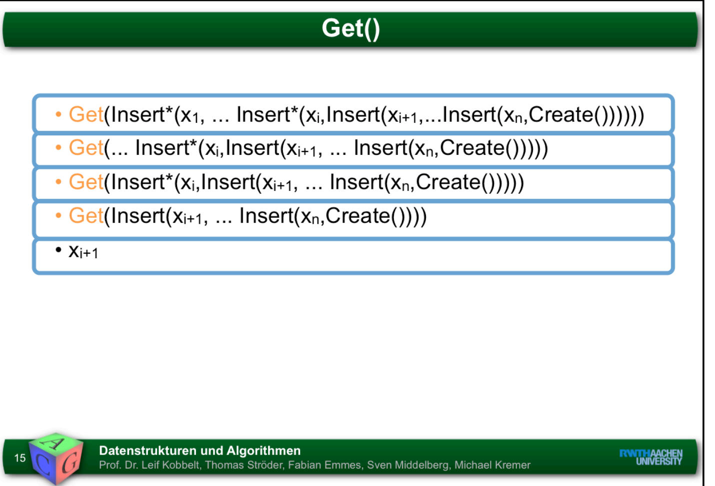
Listen

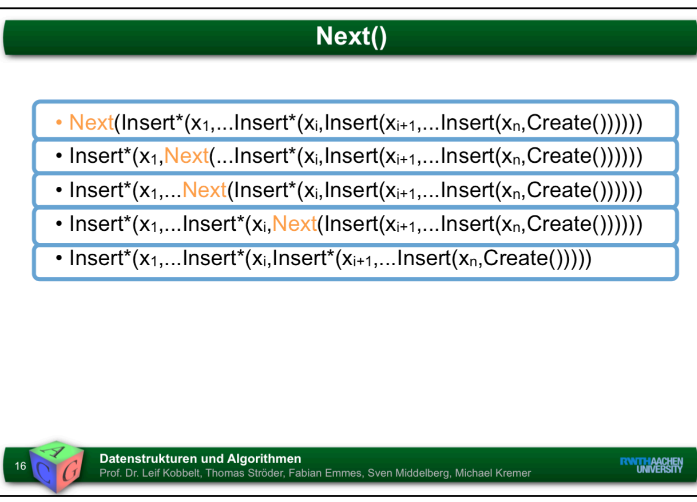
- Satz: Jede Liste hat die Form

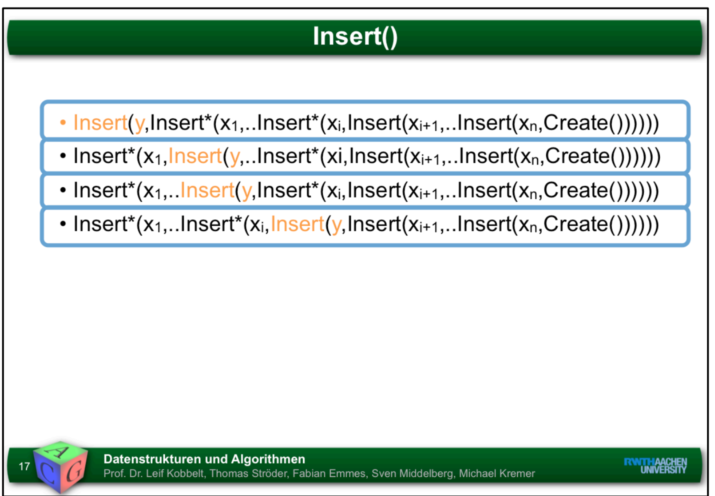
  $Insert^*(x_1,..Insert^*(x_i,Insert(x_{i+1},..Insert(x_n,Create()))))$

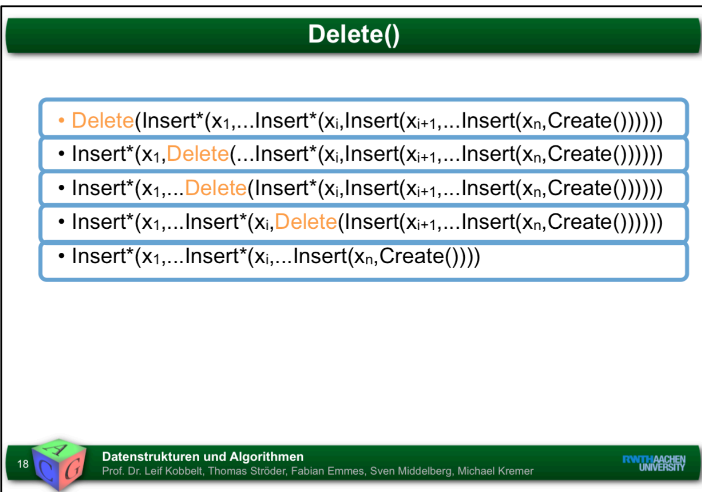
- Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
  - Create() ... n=0
  - Jede andere Operation erhält die Form



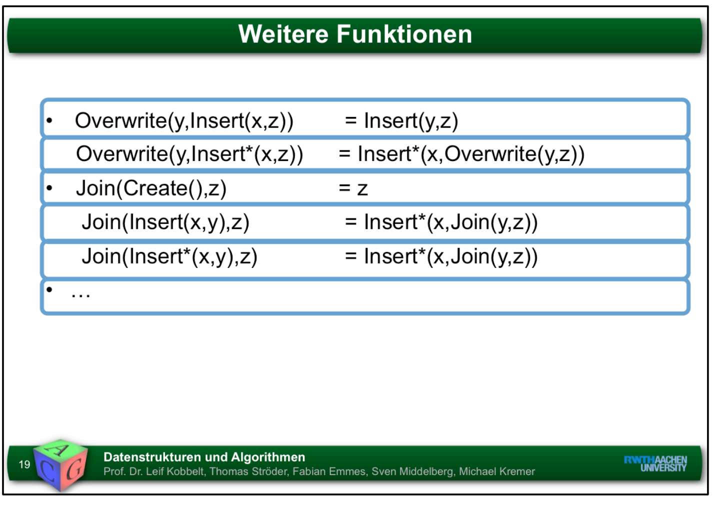

Get()
• Get(Insert*(x1, … Insert*(xi,Insert(xi+1,...Insert(xn,Create()))))))
• Get(… Insert*(xi,Insert(xi+1, … Insert(xn,Create())))))
• Get(Insert*(xi,Insert(xi+1, … Insert(xn,Create())))))
• Get(Insert(xi+1, … Insert(xn,Create())))
• xi+1
15
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


Next()
• Next(Insert*(x1,...Insert*(xi,Insert(xi+1,...Insert(xn,Create()))))))
• Insert*(x1,Next(...Insert*(xi,Insert(xi+1,...Insert(xn,Create()))))))
• Insert*(x1,...Next(Insert*(xi,Insert(xi+1,...Insert(xn,Create()))))))
• Insert*(x1,...Insert*(xi,Next(Insert(xi+1,...Insert(xn,Create()))))))
• Insert*(x1,...Insert*(xi,Insert*(xi+1,...Insert(xn,Create())))))
16
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY



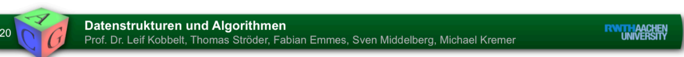


Hier wird ein neues Element bei bestehender Markerposition eingefügt.



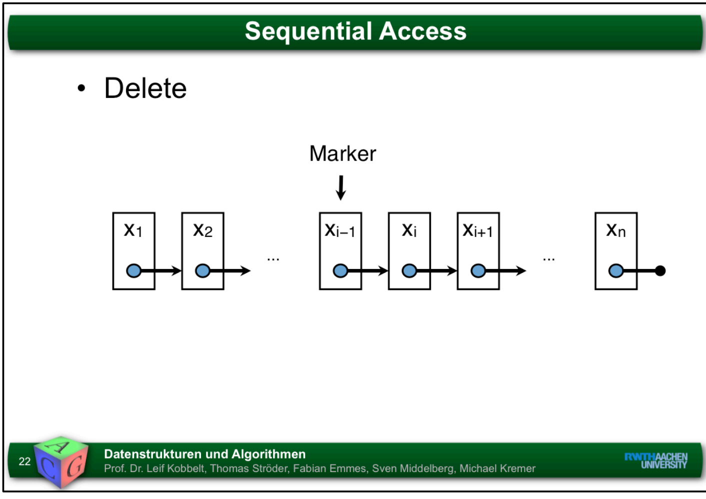
Implementierung

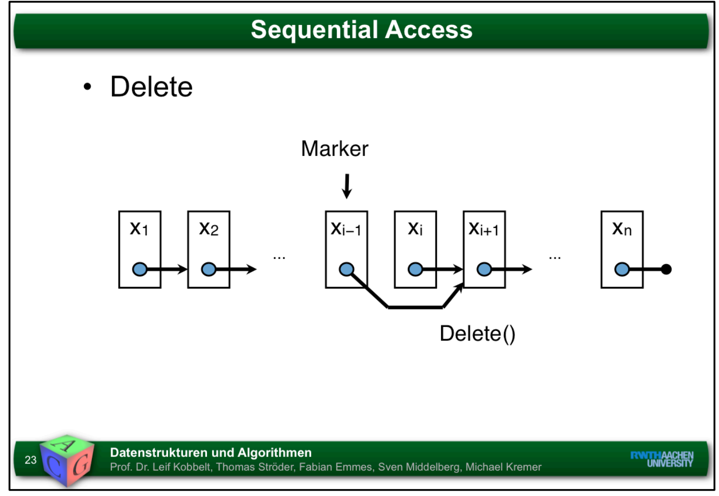
- Finden, Lesen, Überschreiben
- Löschen (Sonderfälle bei leerer Liste)
- Einfügen (Sonderfälle an den Enden)
- Anchor, Sentinel
- Einfach / Doppelt verkettete Listen

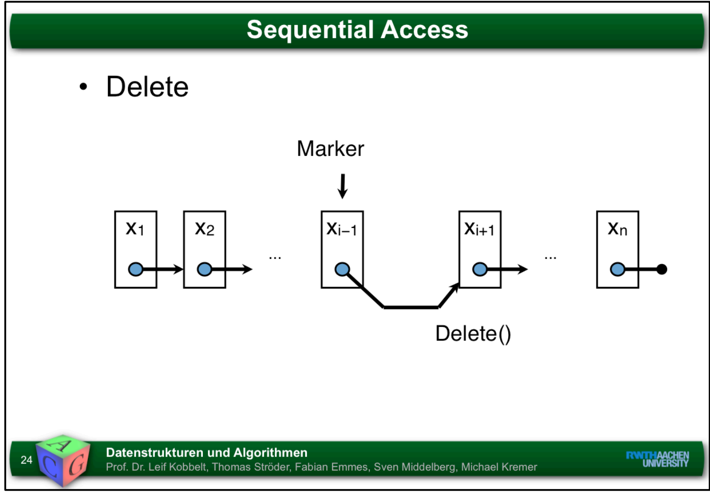


Einfach verkettete Liste

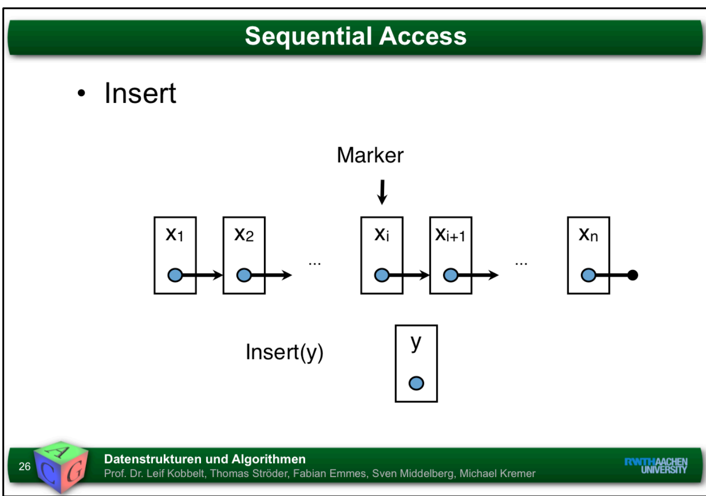
```
• class Element {
    Datentyp   X
    Element    next
  }
```

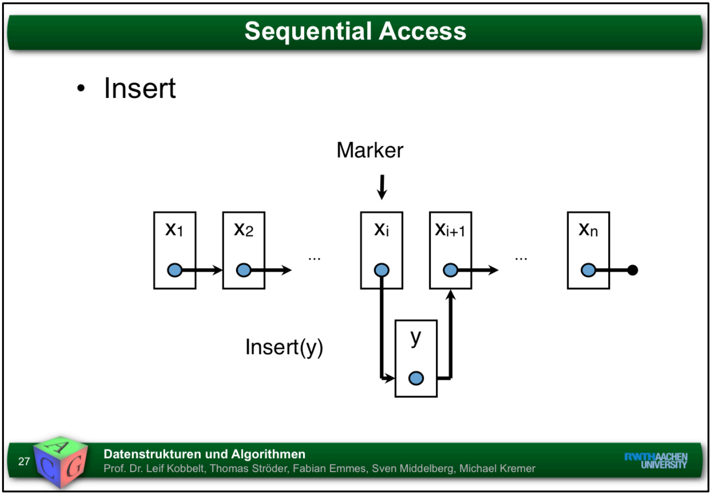

Sequential Access
• Delete
Marker
x1
x2
...
xi−1
xi
xi+1
...
xn
Delete()
23
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


Sequential Access
• Delete
Marker
x1
x2
...
xi−1
xi+1
...
xn
Delete()
24
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

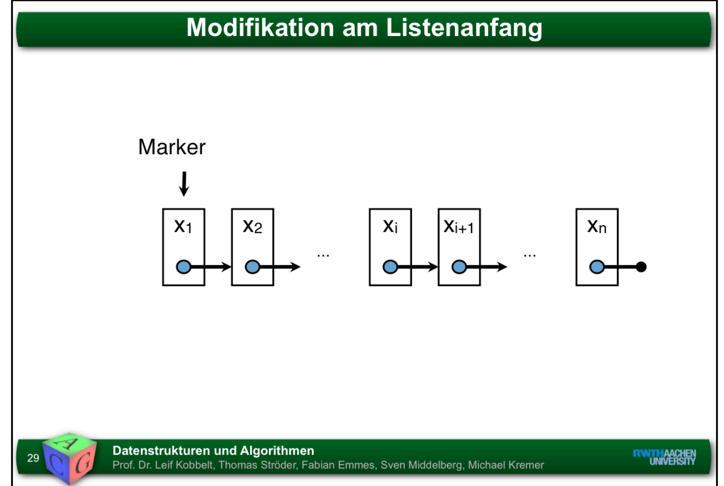
Einfach verkettete Listen
• Delete()
Marker.next ← Marker.next.next
25
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

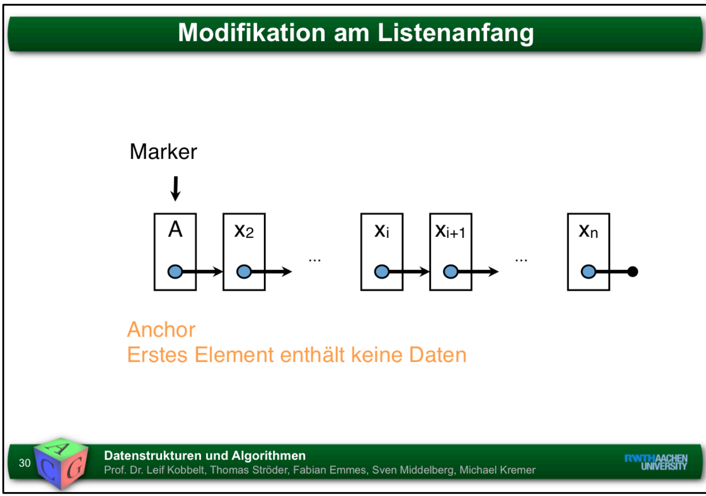


Sequential Access
• Insert
Marker
x1
x2
...
xi
xi+1
...
xn
Insert(y)
y
26
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Sequential Access
• Insert
Marker
x1
x2
...
xi
xi+1
...
xn
Insert(y)
y
27
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

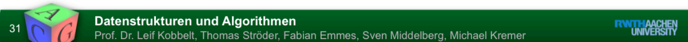
Einfach verkettete Listen
• Insert(Y)
Y.next ← Marker.next
Marker.next ← Y
28
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Modifikation am Listenanfang
Marker
x1
x2
...
xi
xi+1
...
xn
29
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


Modifikation am Listenanfang
Marker
A
x2
...
xi
xi+1
...
xn
Anchor
Erstes Element enthält keine Daten
30
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

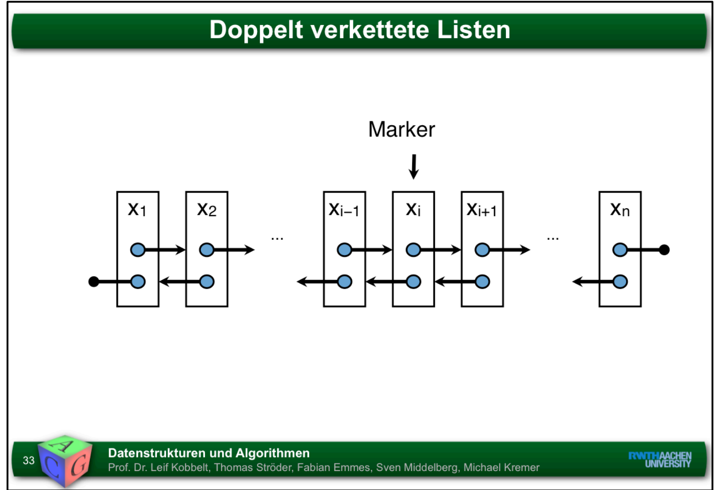
Doppelt verkettete Listen

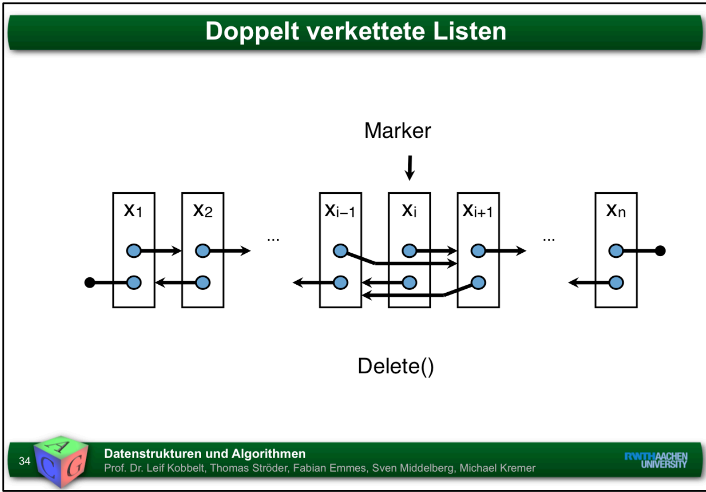
- Flexiblerer Zugriff (upstream/downstream)
  - Next(), Prev()
- Bisher: Einfügen/Löschen nach dem Marker (wg. Pointer update)
- Jetzt: Marker zeigt auf aktuelles Element

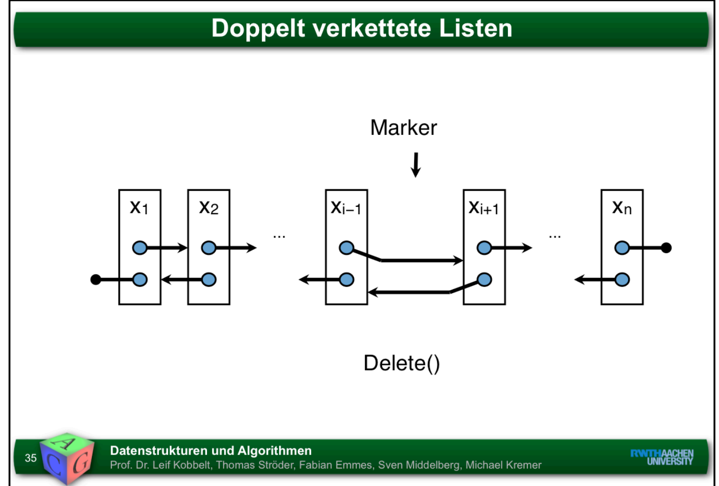


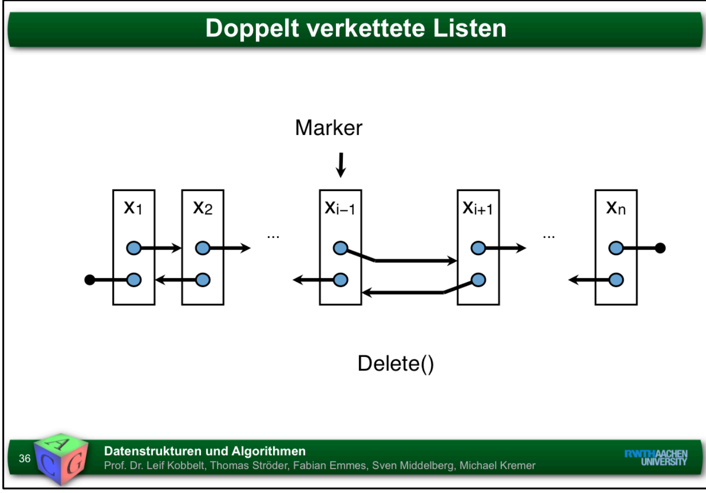
Doppelt verkettete Liste

- class Element {
  Datentyp X
  Element prev, next
  }

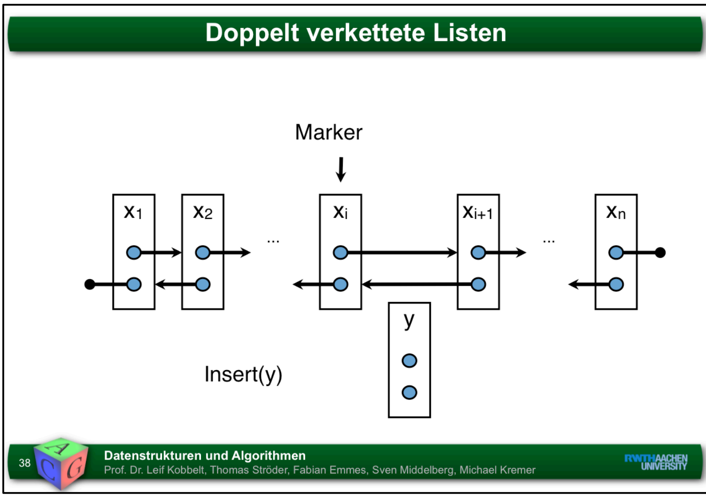

Doppelt verkettete Listen
Marker
x1
x2
...
xi−1
xi
xi+1
...
xn
33
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

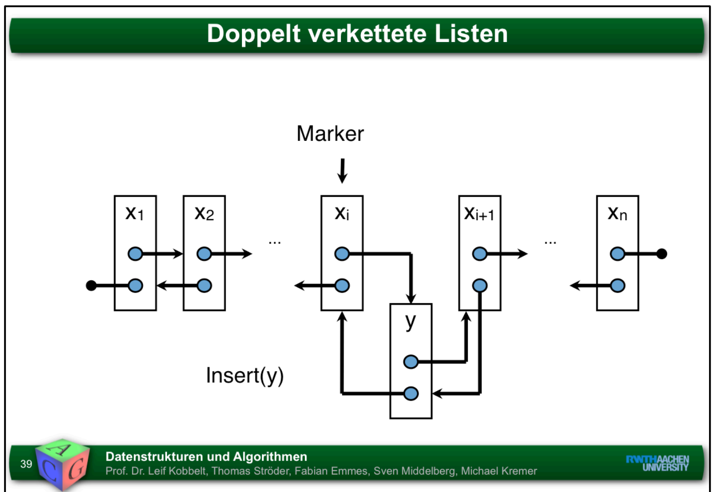

Doppelt verkettete Listen
Marker
x1
x2
...
xi−1
xi
xi+1
...
xn
Delete()
34
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

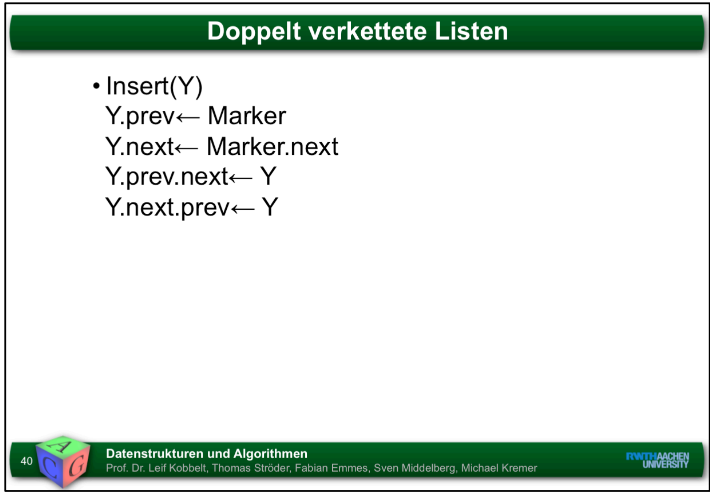

Doppelt verkettete Listen
Marker
x1
x2
...
xi−1
xi+1
...
xn
Delete()
35
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Doppelt verkettete Listen
Marker
x1
x2
...
xi−1
xi+1
...
xn
Delete()
36
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

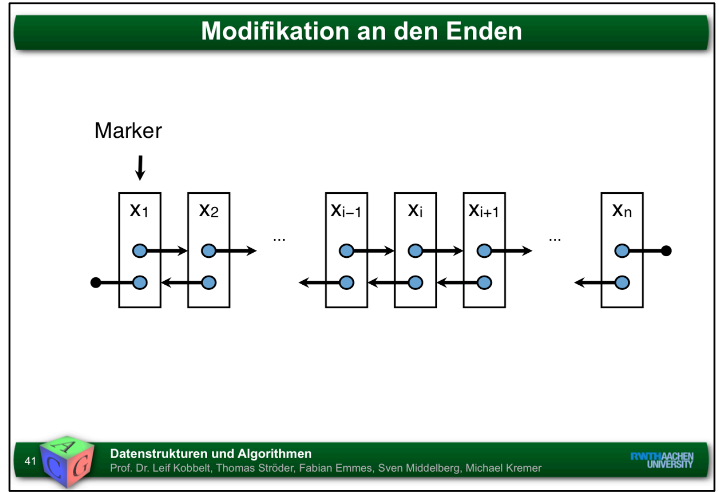
Doppelt verkettete Listen
• Delete()
Marker.prev.next← Marker.next
Marker.next.prev← Marker.prev
Marker← Marker.prev
37
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

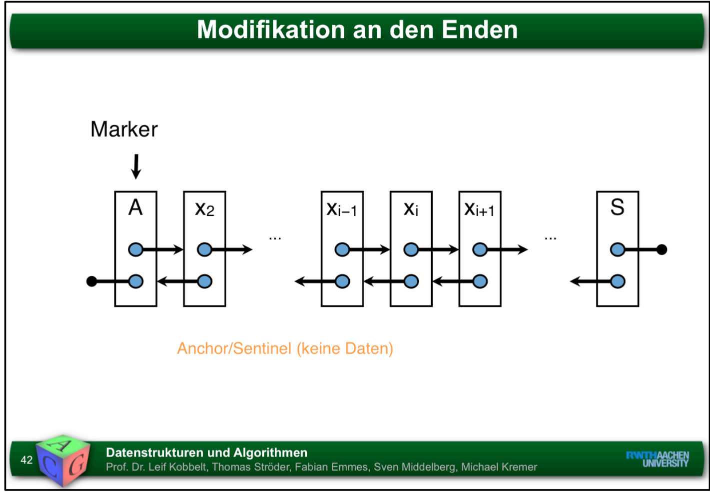

Doppelt verkettete Listen
Marker
x1
x2
...
xi
xi+1
...
xn
y
Insert(y)
38
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY





Fazit: Listen klingen erst mal unhandlich. Aber alle Operationen haben IMMER konstante Anzahl an Pointer-Operationen, unabhängig von der Datengröße.

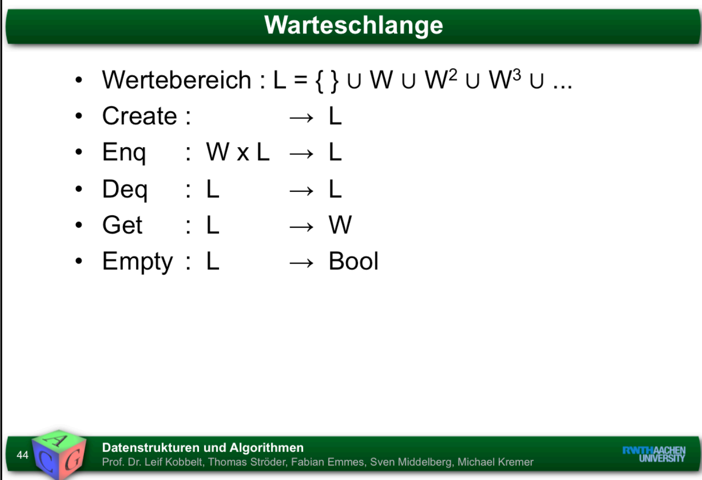

Modifikation an den Enden
Marker
x1
x2
...
xi−1
xi
xi+1
...
xn
41
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Modifikation an den Enden
Marker
A
x2
...
xi−1
xi
xi+1
...
S
Anchor/Sentinel (keine Daten)
42
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

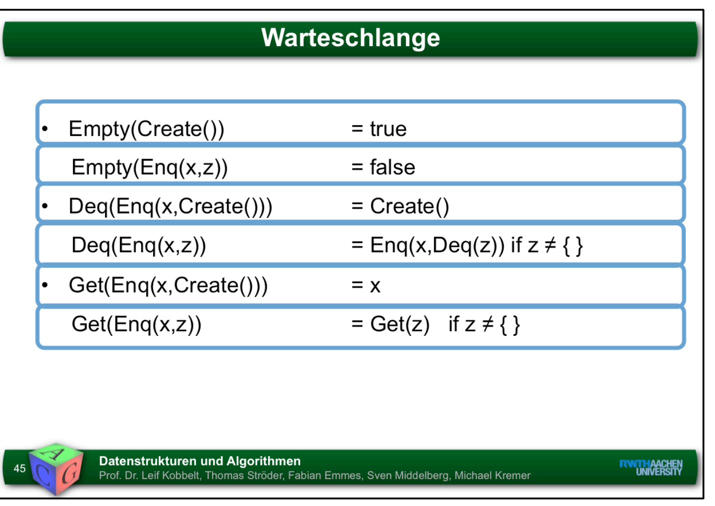
1.2.2 Warteschlange

- Liste mit eingeschränkter Funktionalität
- Einfügen nur am Ende
- Auslesen/Entfernen nur am Anfang
- „First in, first out“ (FIFO)

43 Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer



Enq = Enqueue (einfügen)
Deq = Dequeue (ausfügen)

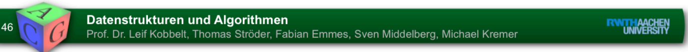


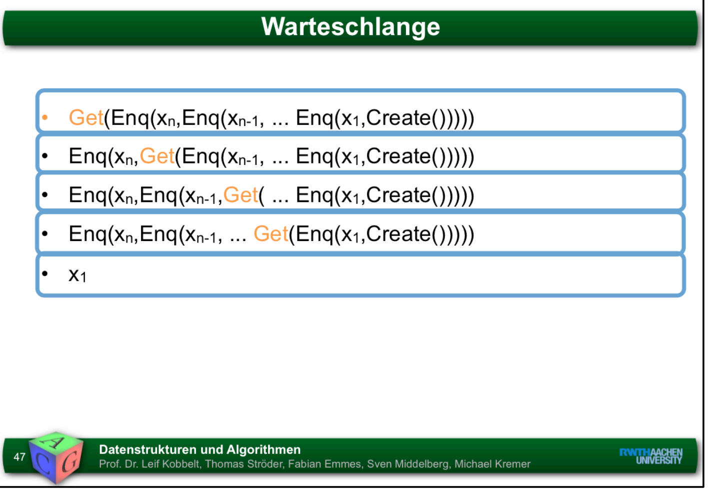
Warteschlange

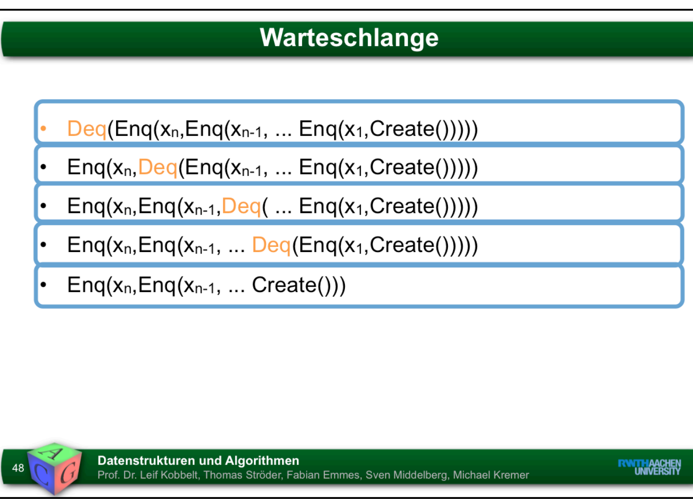
- Satz: Jede Warteschlange hat die Form

  $Enq(x_n, Enq(x_{n-1}, \ldots Enq(x_1, Create())))$

- Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
  - Create() ... n=0
  - Jede andere Operation erhält die Form




Warteschlange
• Get(Enq(xn,Enq(xn-1, ... Enq(x1,Create())))
• Enq(xn,Get(Enq(xn-1, ... Enq(x1,Create())))
• Enq(xn,Enq(xn-1,Get(... Enq(x1,Create())))
• Enq(xn,Enq(xn-1, ... Get(Enq(x1,Create())))
• x1
47
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

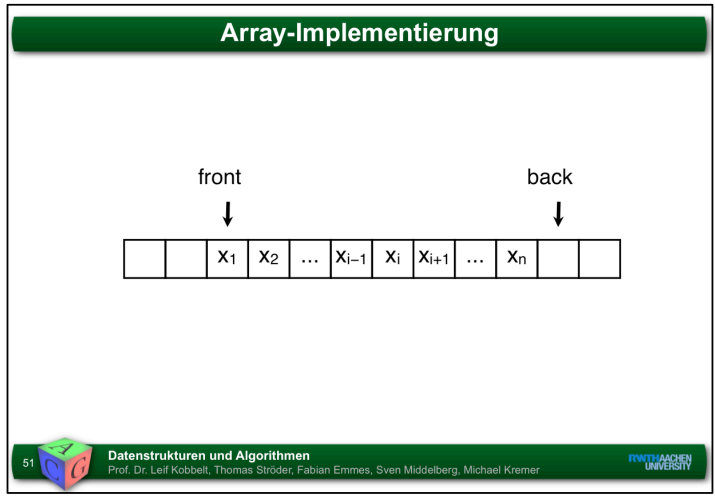

Warteschlange
• Deq(Enq(xn,Enq(xn-1, ... Enq(x1,Create())))
• Enq(xn,Deq(Enq(xn-1, ... Enq(x1,Create())))
• Enq(xn,Enq(xn-1,Deq(... Enq(x1,Create())))
• Enq(xn,Enq(xn-1, ... Deq(Enq(x1,Create())))
• Enq(xn,Enq(xn-1, ... Create()))
48
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

Array-Implementierung

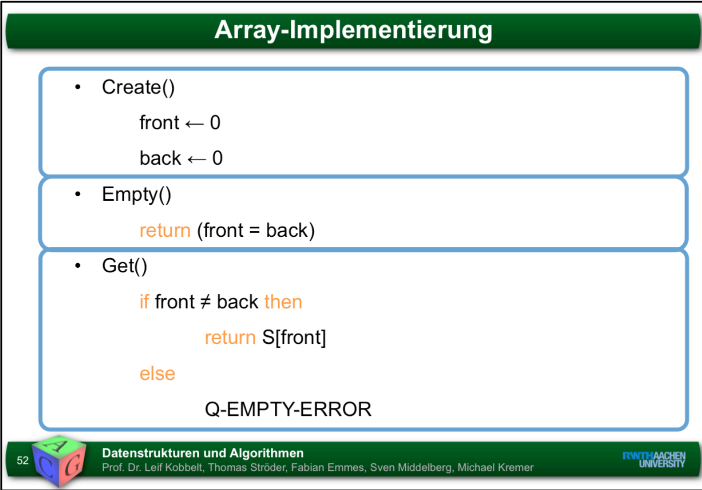
- Da nur am Anfang eingefügt und am Ende gelöscht wird, ist keine garbage collection notwendig.
- Maximale Länge wird als bekannt vorausgesetzt.

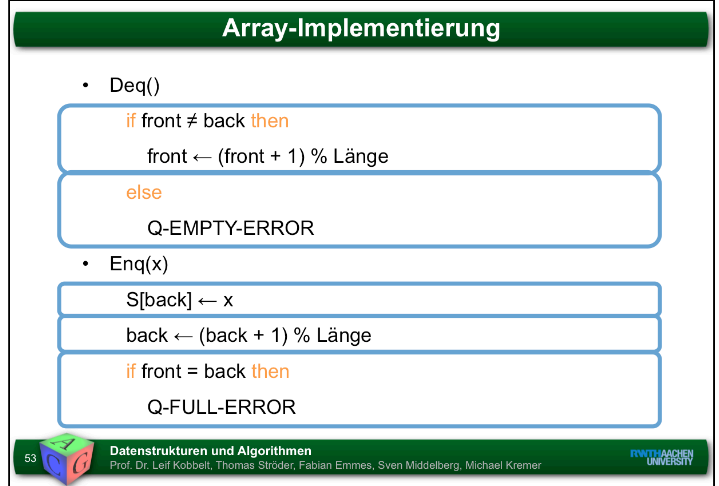
Array-Implementierung

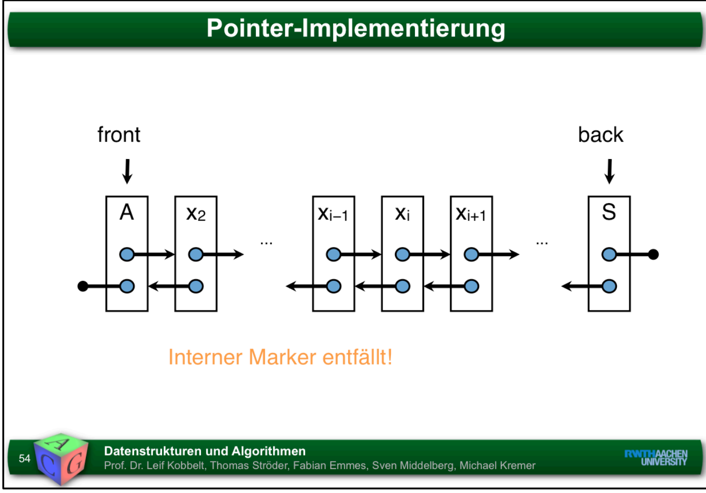
- class Schlange {
    Datentyp S[Länge]
    int front, back
  }

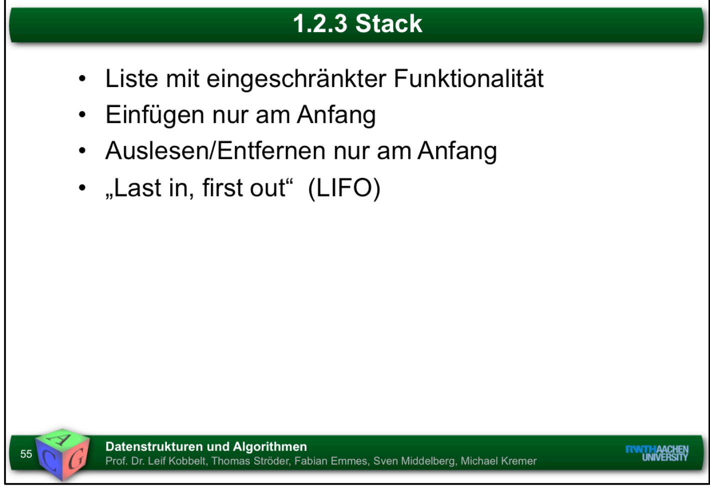


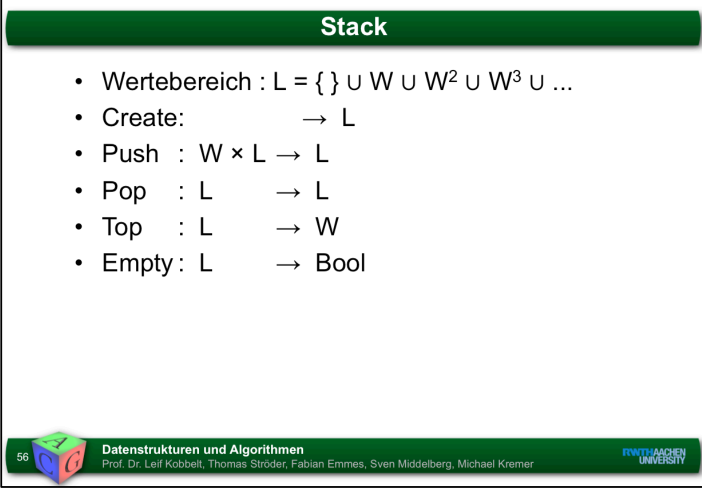
Front- und Back-Pointer werden benutzt, um sich zu jeder Zeit die Position des ersten und letzten Elements zu „merken“. Diese müssen bei den Operationen auf dem Array natürlich auch angepasst werden (zusätzlich zu den Daten).

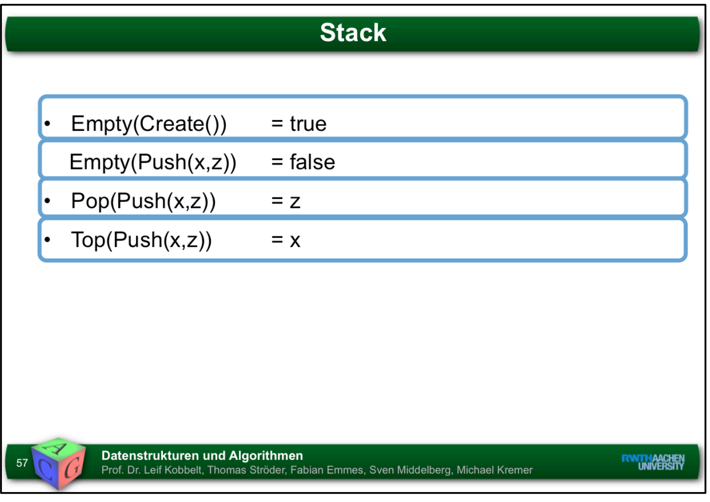


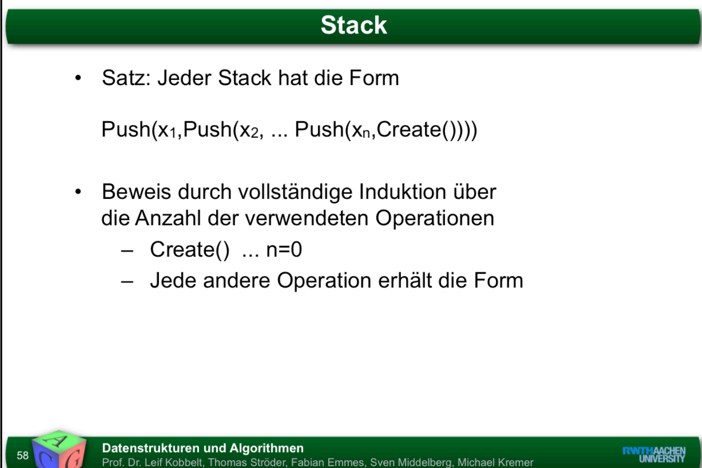

Array-Implementierung
• Deq()
if front ≠ back then
front ← (front + 1) % Länge
else
Q-EMPTY-ERROR
• Enq(x)
S[back] ← x
back ← (back + 1) % Länge
if front = back then
Q-FULL-ERROR
53
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Pointer-Implementierung
front
back
A
x2
...
xi−1
xi
xi+1
S
Interner Marker entfällt!
54
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


1.2.3 Stack
• Liste mit eingeschränkter Funktionalität
• Einfügen nur am Anfang
• Auslesen/Entfernen nur am Anfang
• „Last in, first out“ (LIFO)
55
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

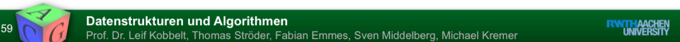

Stack
• Wertebereich : L = { } ∪ W ∪ W² ∪ W³ ∪ ...
• Create: → L
• Push : W × L → L
• Pop : L → L
• Top : L → W
• Empty : L → Bool
56
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Stack
Empty(Create()) = true
Empty(Push(x,z)) = false
Pop(Push(x,z)) = z
Top(Push(x,z)) = x
57
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Stack
Satz: Jeder Stack hat die Form
Push(x1,Push(x2, ... Push(xn,Create())))
Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
Create() ... n=0
Jede andere Operation erhält die Form
58
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

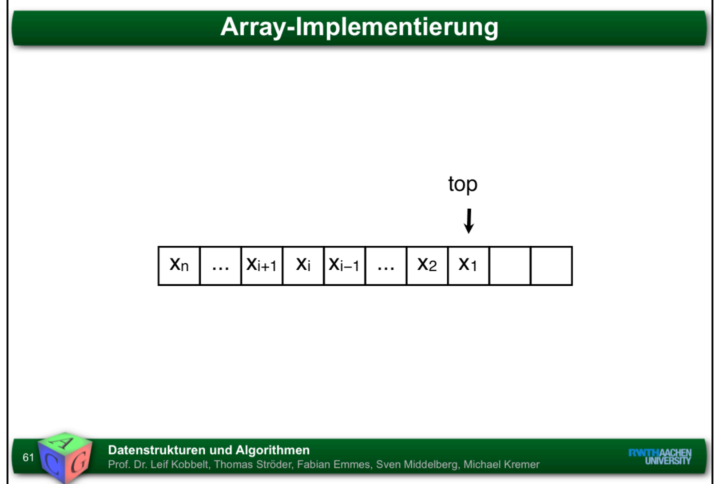
Array-Implementierung

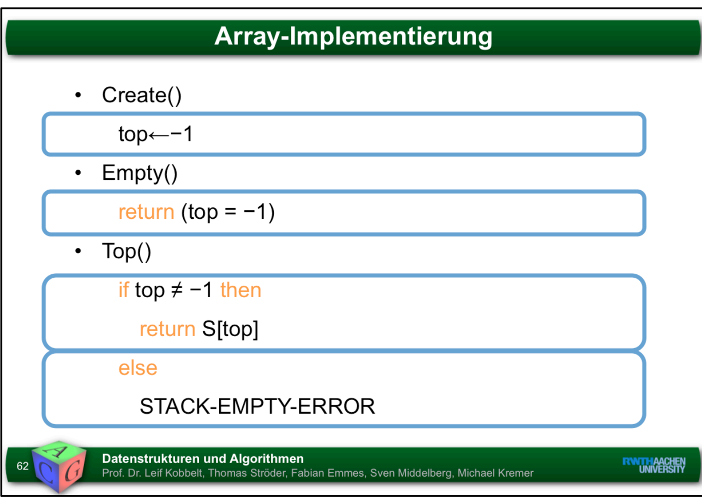
- Da nur am Anfang eingefügt und gelöscht wird, ist keine garbage collection notwendig.
- Maximale Größe wird als bekannt vorausgesetzt.

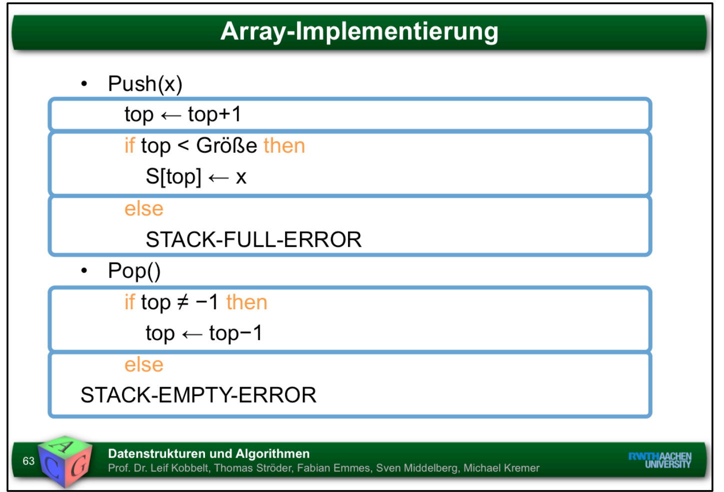


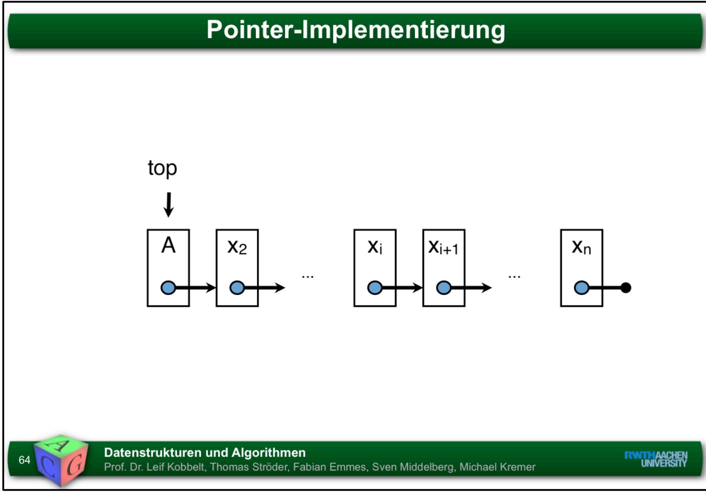
Array-Implementierung

- class Stack {
  Datentyp S[Größe]
  int top
  }


Array-Implementierung
top
xn … xi+1 xi xi−1 … x2 x1
61
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


Array-Implementierung
• Create()
top←−1
• Empty()
return (top = −1)
• Top()
if top ≠ −1 then
return S[top]
else
STACK-EMPTY-ERROR
62
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


Array-Implementierung
Push(x)
top ← top+1
if top < Größe then
S[top] ← x
else
STACK-FULL-ERROR
Pop()
if top ≠ −1 then
top ← top−1
else
STACK-EMPTY-ERROR
63
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Pointer-Implementierung
top
A
x2
...
xi
xi+1
...
xn
64
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

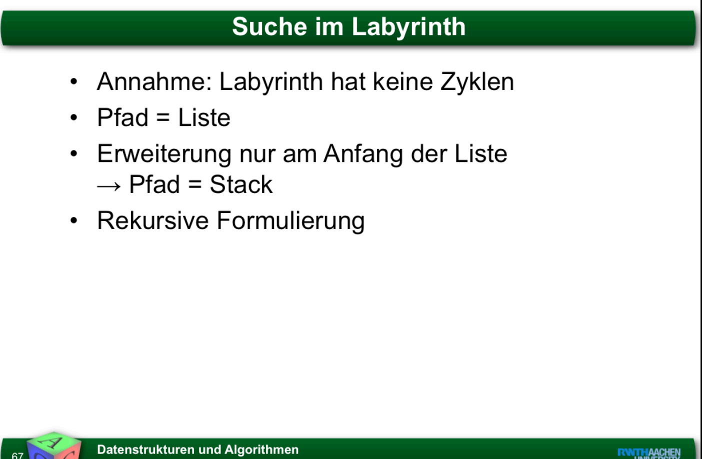
Suche im Labyrinth

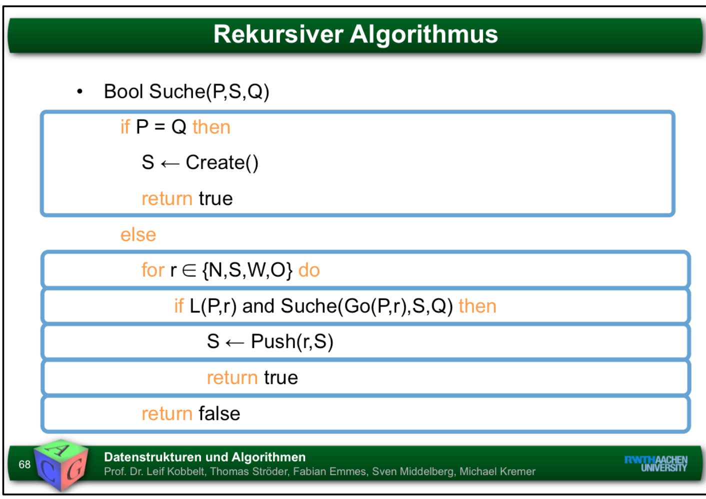
- Geg:
  - Spielfläche $W = [0..n] \times [0..m]$
  - mögliche Positionen $P = (x,y) \in W$
  - Bewegungsrichtungen: $R = \{N,S,W,O\}$
  - Labyrinth ... $L : W \times R \rightarrow Bool$
  - Startposition P_{begin}
  - Zielposition P_{end}
- Ges: Pfad $S \in R^k$ von P_{begin} nach P_{end}

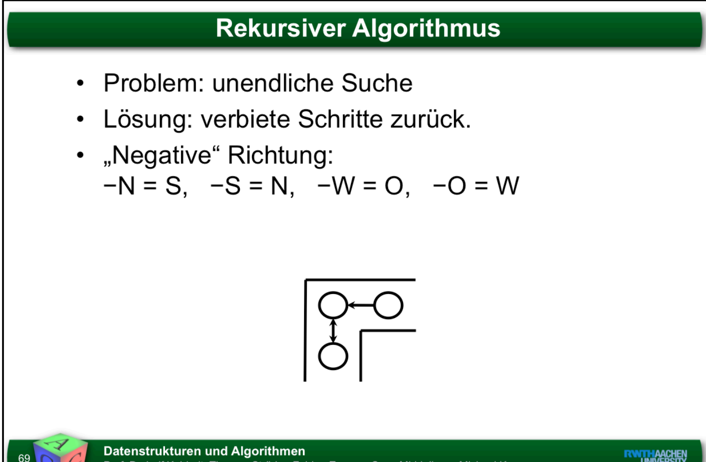
Suche im Labyrinth

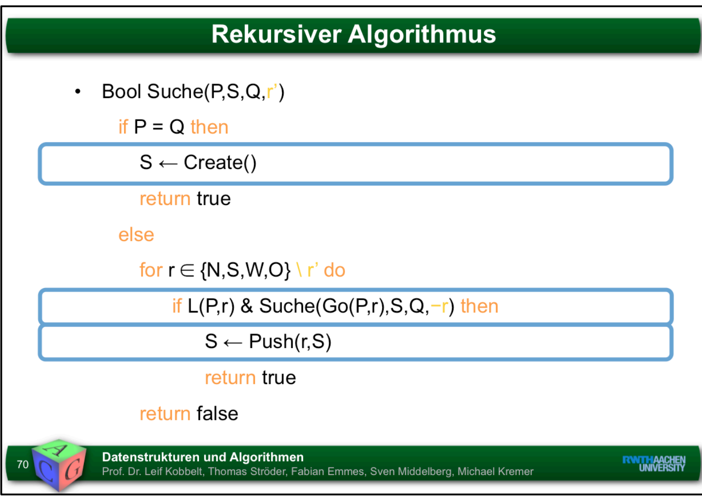
- Funktion $Go: W \times R \rightarrow W$ beschreibt einen Schritt $Go(P,r)=Q$ von P in die Richtung r nach Q.

- Wenn $L(P_{begin},r) = true$ und $Go(P_{begin},r) = Q$ und es existiert ein Pfad $r_1,..,r_n$ von Q nach P_{end} dann ist $r,r_1,..,r_n$ ein Pfad von P_{begin} nach P_{end}.


Suche im Labyrinth
• Annahme: Labyrinth hat keine Zyklen
• Pfad = Liste
• Erweiterung nur am Anfang der Liste
→ Pfad = Stack
• Rekursive Formulierung
67
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Rekursiver Algorithmus
• Bool Suche(P,S,Q)
if P = Q then
S ← Create()
return true
else
for r ∈ {N,S,W,O} do
if L(P,r) and Suche(Go(P,r),S,Q) then
S ← Push(r,S)
return true
return false
68
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


Rekursiver Algorithmus
Problem: unendliche Suche
Lösung: verbiete Schritte zurück.
„Negative“ Richtung:
−N = S, −S = N, −W = O, −O = W
69
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

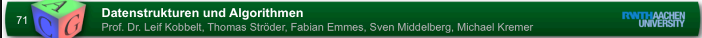

Rekursiver Algorithmus
Bool Suche(P,S,Q,r')
if P = Q then
S ← Create()
return true
else
for r ∈ {N,S,W,O} \ r' do
if L(P,r) & Suche(Go(P,r),S,Q,−r) then
S ← Push(r,S)
return true
return false
70
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

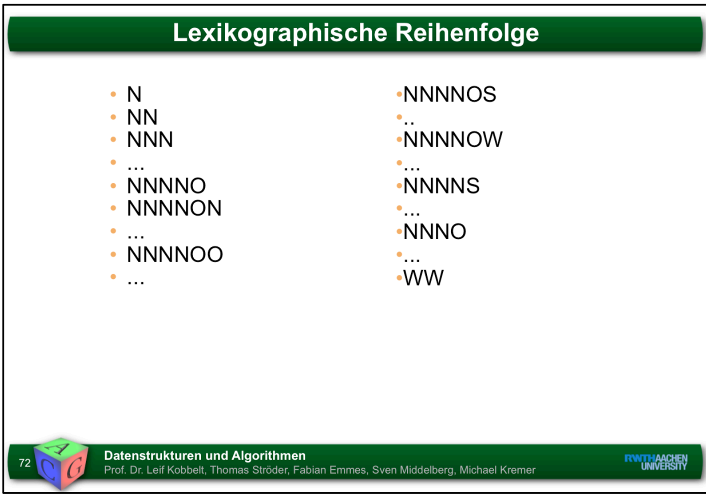
Iterativer Algorithmus

- Rekursion steuert die Suche
- Direkte Lösung unter Verwendung eines Stack
- Ordnung der Richtungen N < O < S < W

  (next(N) = O, next(O) = S, ...)

- Sortiere Pfade lexikographisch



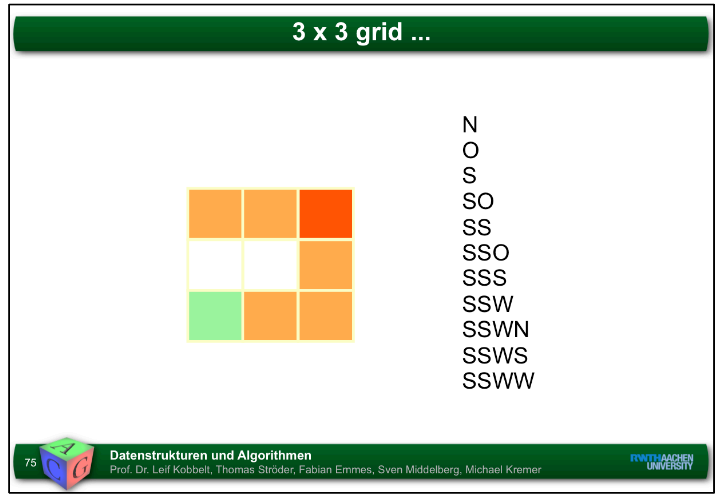


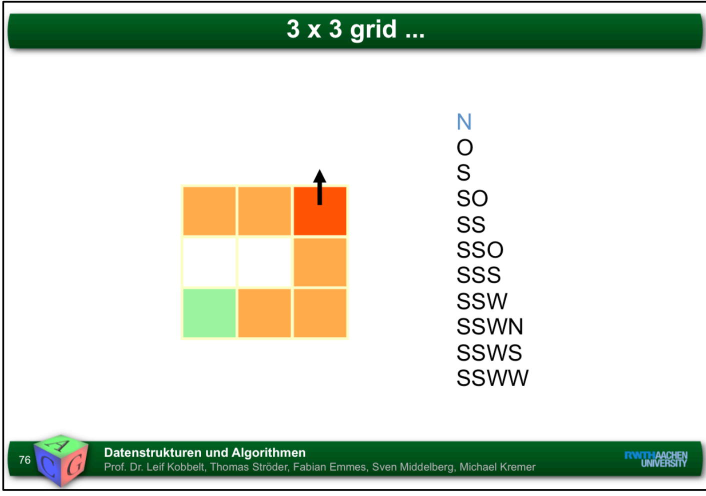
Iterativer Algorithmus

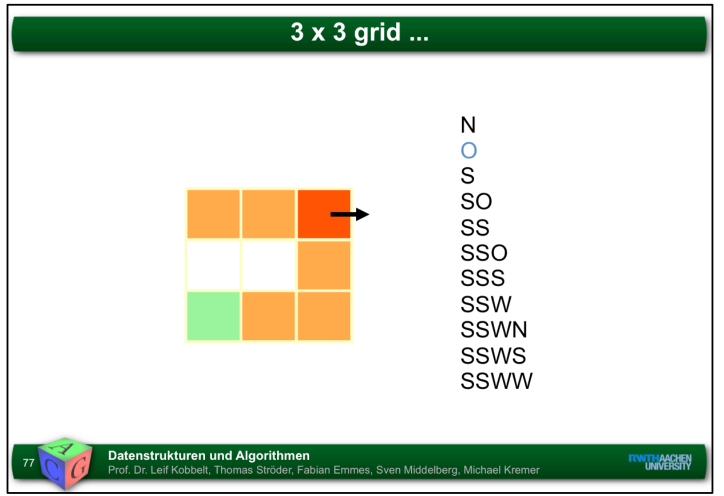
- $P \leftarrow P_{begin}$
  $S \leftarrow$ Create()
  $r \leftarrow$ ´N´
  while $P \neq P_{end}$ and (L(P,r) or r < ´W´ or not Empty(S)) do
    while $P \neq P_{end}$ and L(P,r) do
      S = Push(r,S)
      $P \leftarrow$ Go(P,r)
      $r \leftarrow$ ´N´
    if $P \neq P_{end}$ then
      while r = ´W´ and not Empty(S) do
        r = Top(S)
        $P \leftarrow$ Go(P,−r)
        $S \leftarrow$ Pop(S)
      if r < ´W´ then
        $r \leftarrow$ next(r)

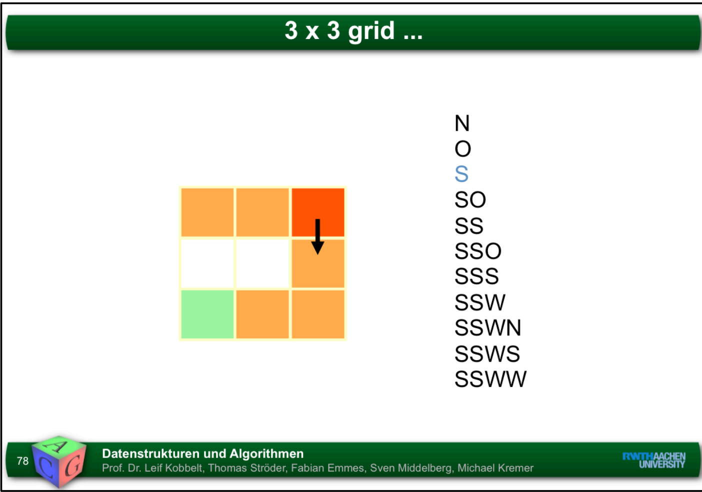
Iterativer Algorithmus

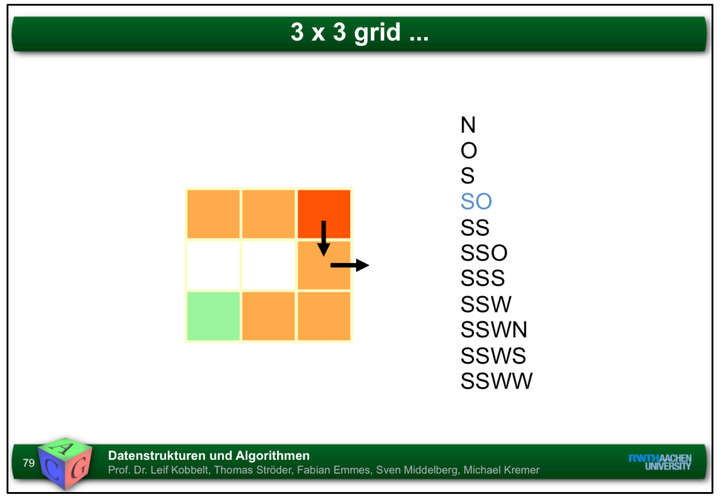
- $P \leftarrow P_{begin}$
  $S \leftarrow$ Create()
  $r \leftarrow$ ´N´
  while $P \neq P_{end}$ and (L(P,r) or r < ´W´ or not Empty(S)) do
    while $P \neq P_{end}$ and L(P,r) and $r \neq -$Top(S) do
      S = Push(r,S)
      $P \leftarrow$ Go(P,r)
      $r \leftarrow$ ´N´
    if $P \neq P_{end}$ then
      while r = ´W´ and not Empty(S) do
        r = Top(S)
        $P \leftarrow$ Go(P,−r)
        $S \leftarrow$ Pop(S)
      if r < ´W´ then
        $r \leftarrow$ next(r)

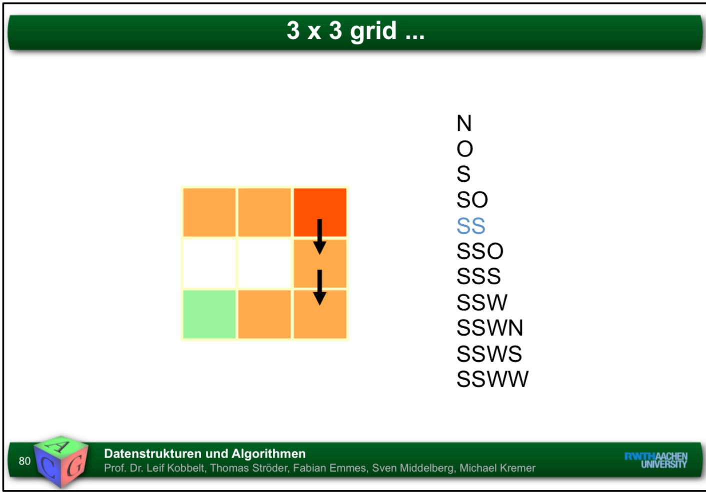

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
75
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

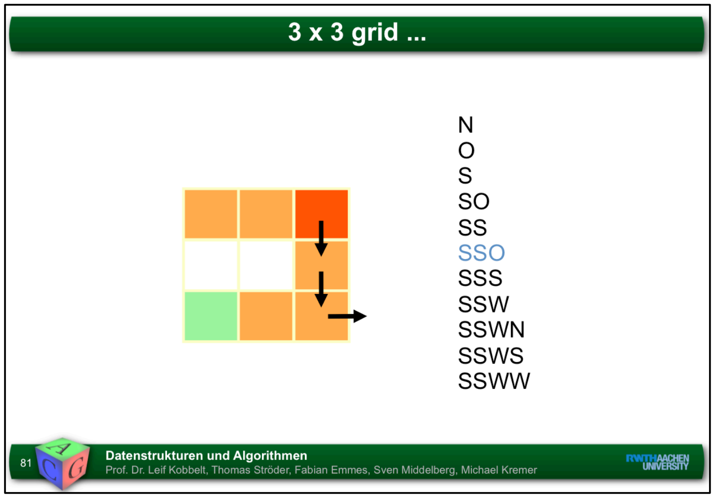

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
76
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

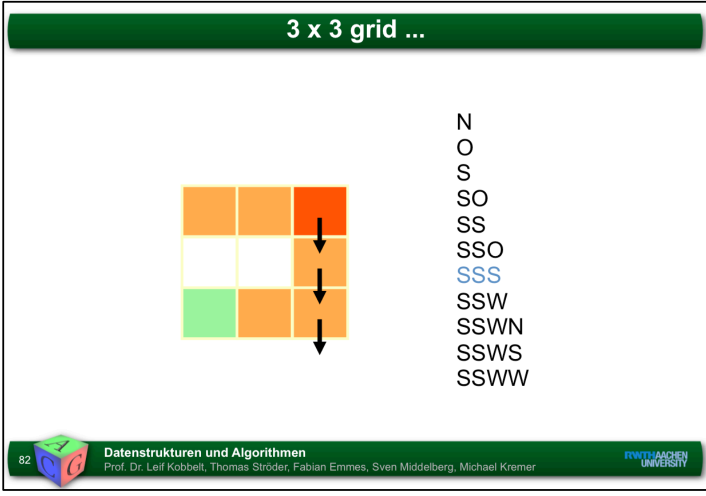

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
77
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

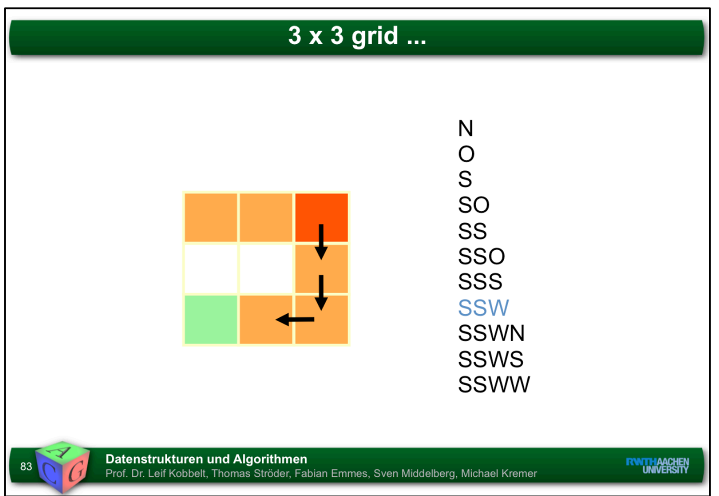

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
78
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

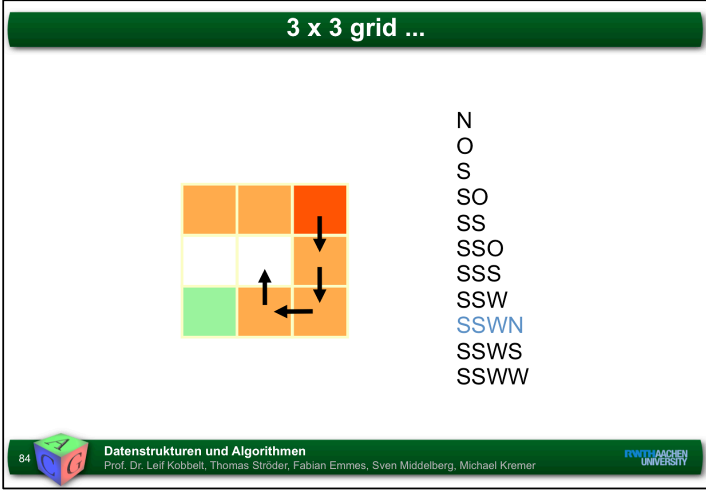

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
79
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

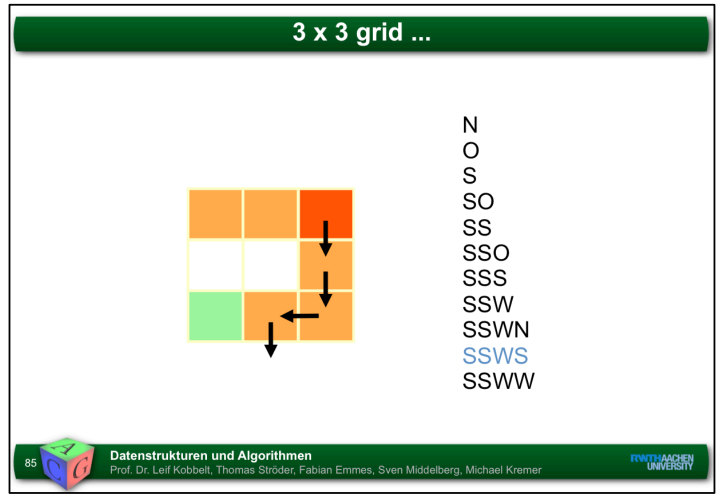

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
80
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

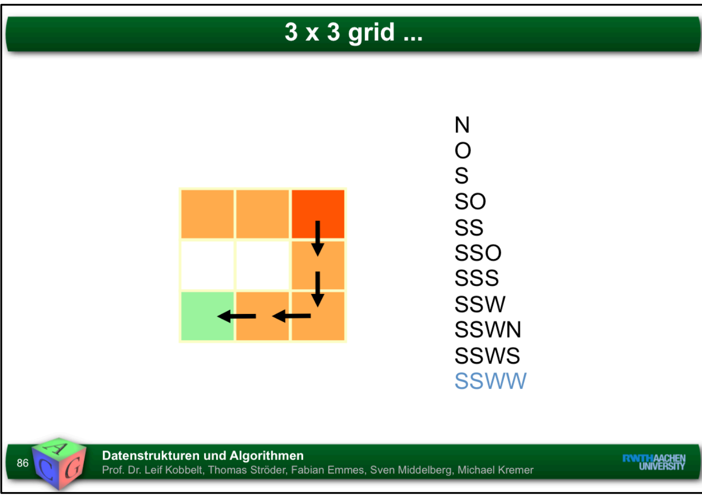

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
81
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

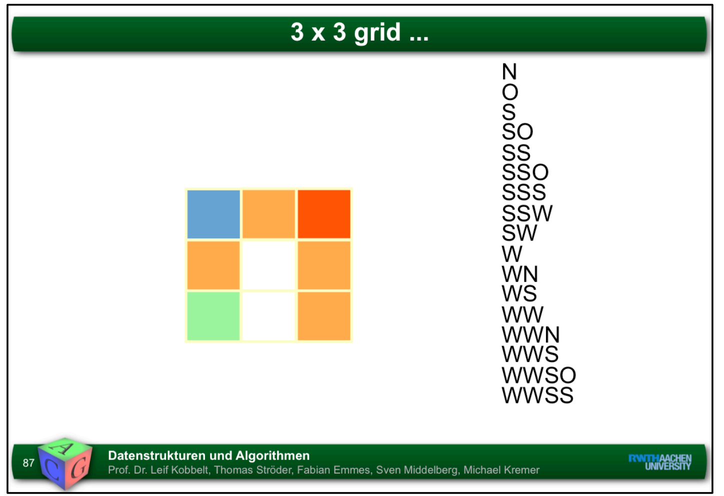

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
82
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

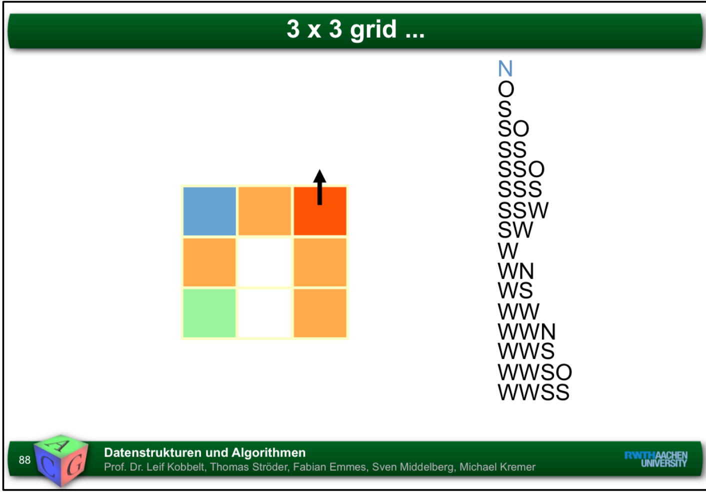

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
83
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

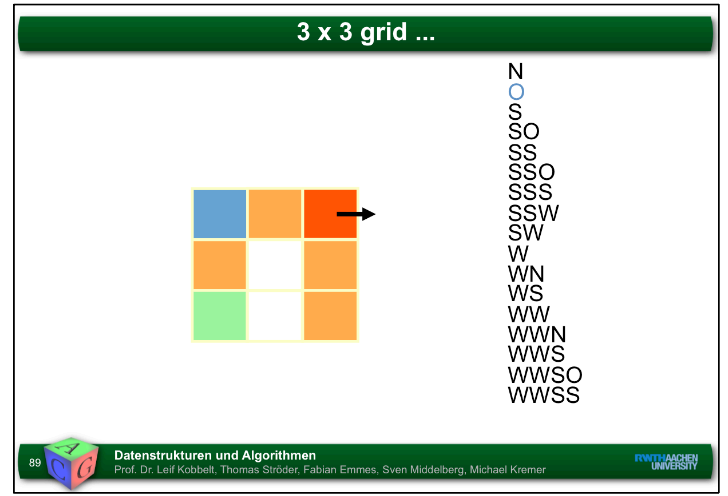

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
84
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

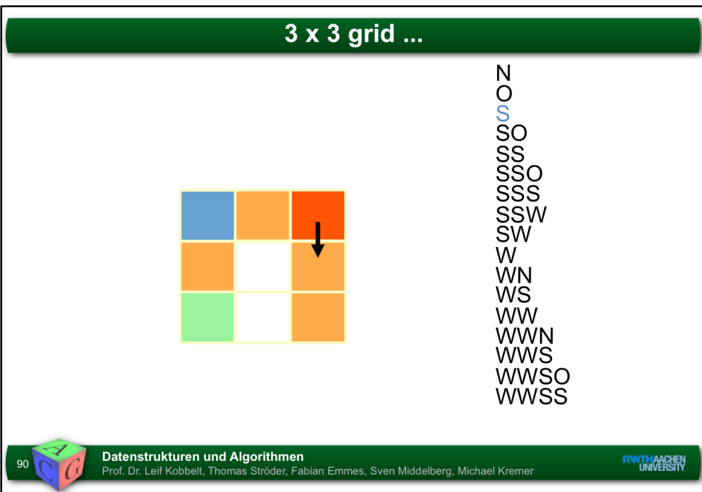

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
85
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

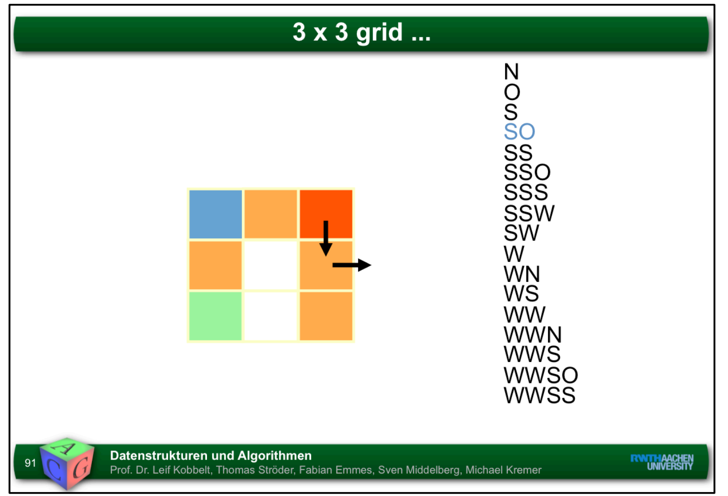

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SSWN
SSWS
SSWW
86
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

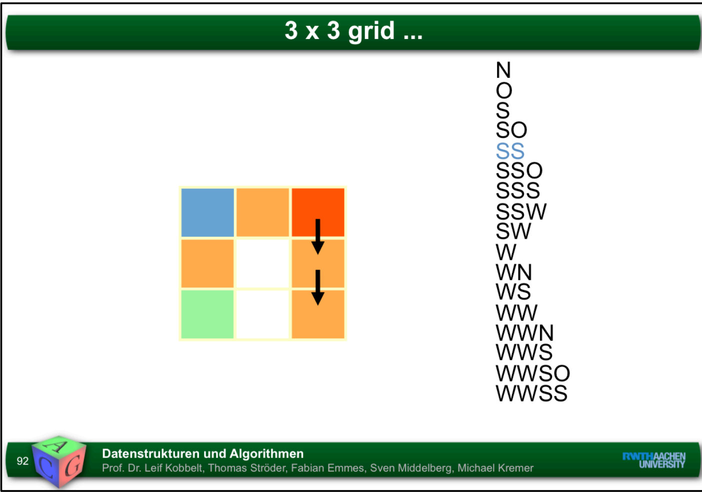

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
87
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

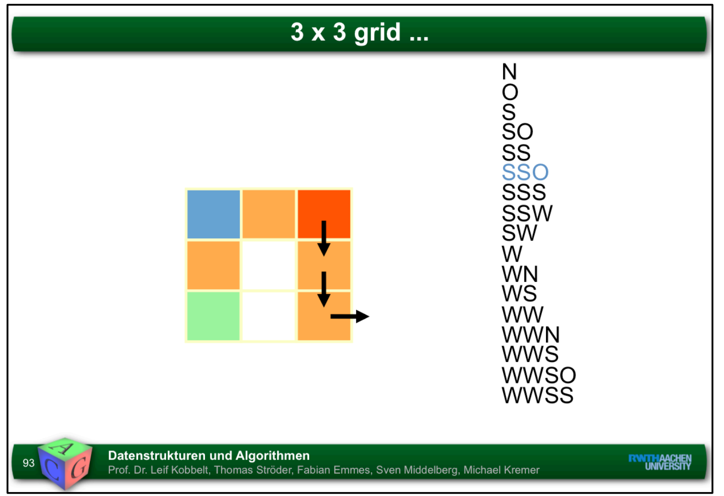

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
88
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

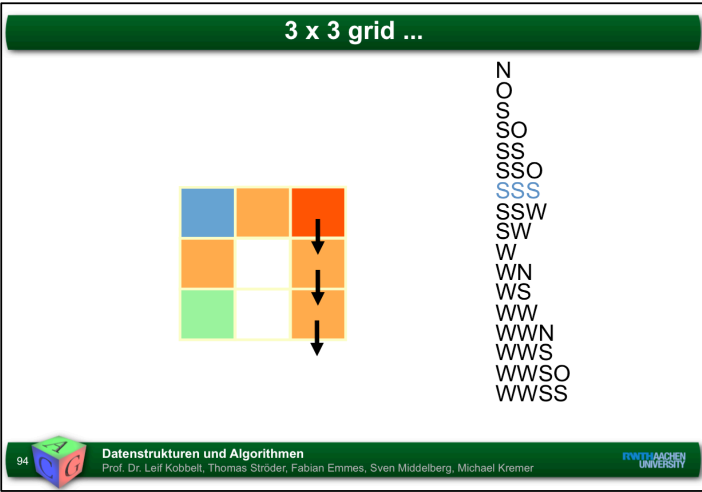

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
89
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

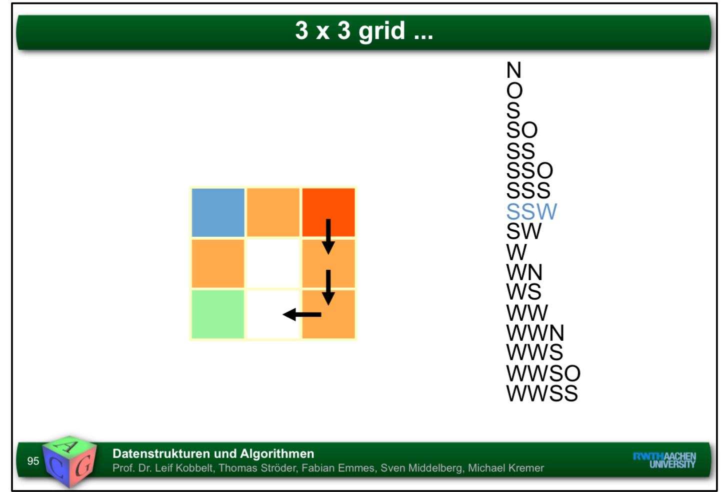

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
90
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer

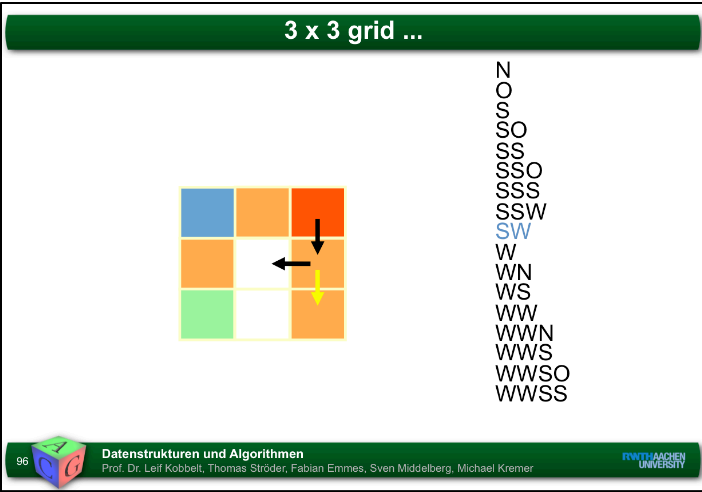

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
91
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

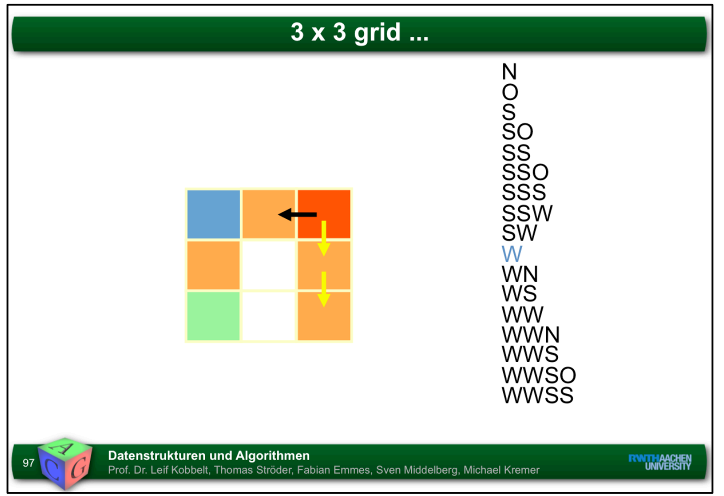

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
92
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

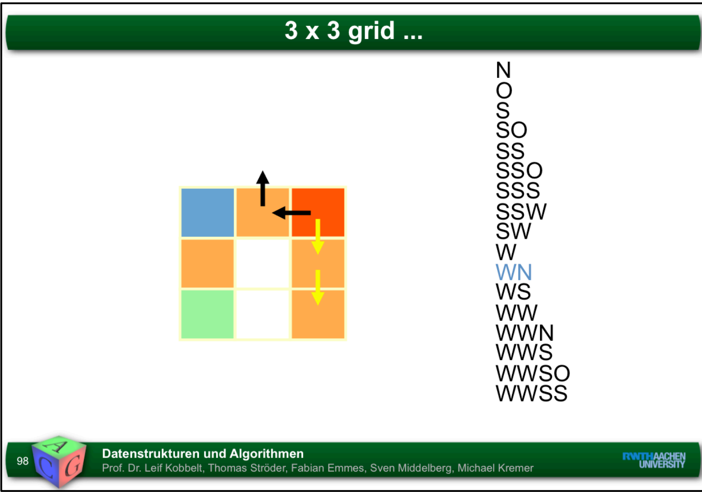

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
93
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTHAACHEN UNIVERSITY

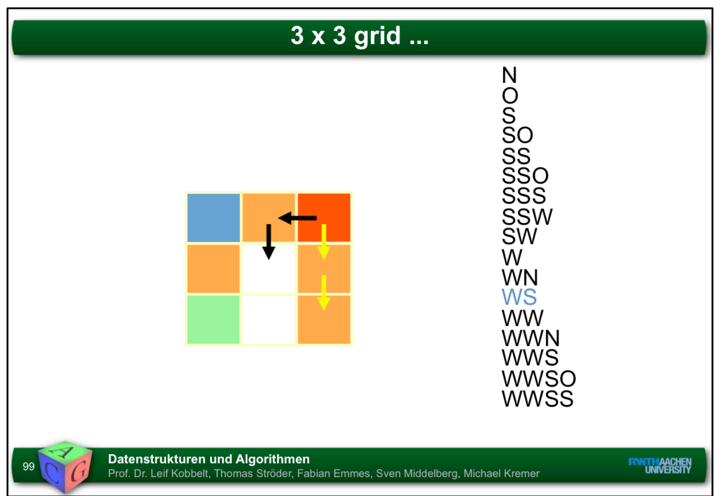

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
94
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTHAACHEN UNIVERSITY

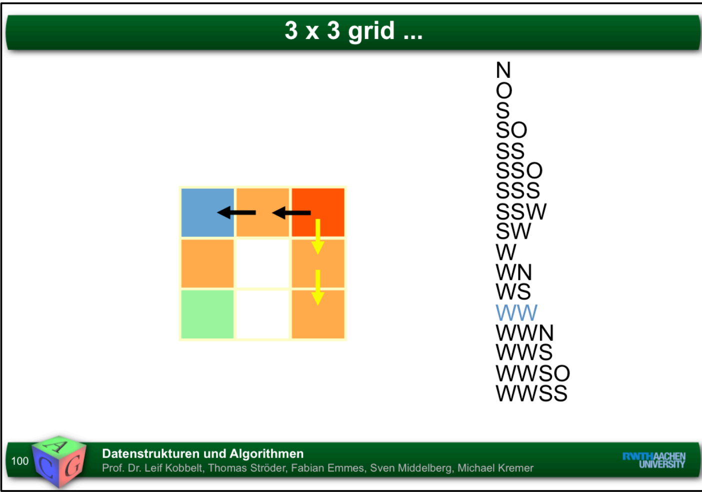

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
95
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTHAACHEN UNIVERSITY

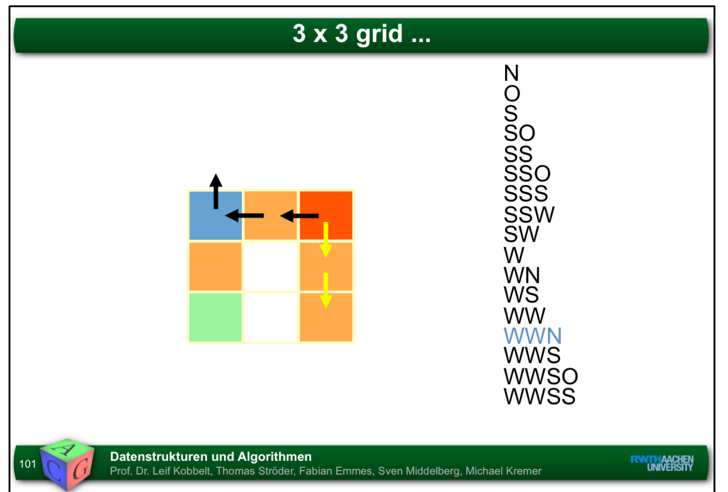

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
96
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTHAACHEN UNIVERSITY

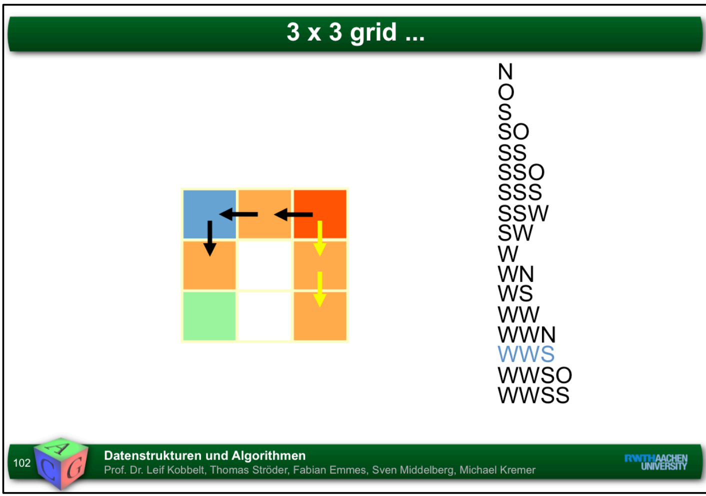

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
97
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

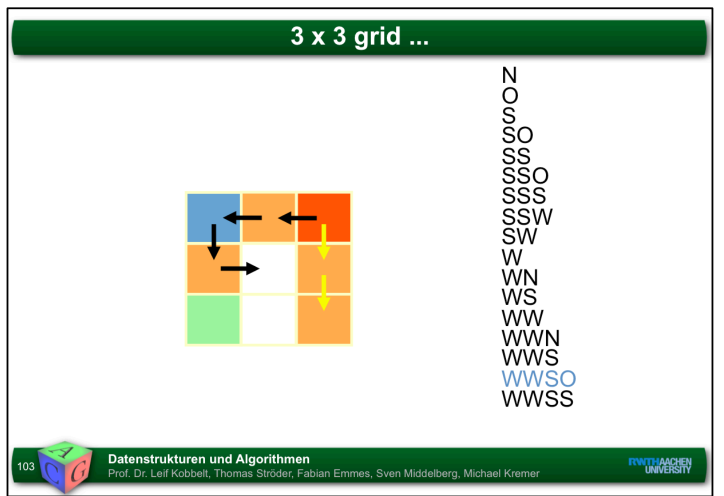

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
98
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

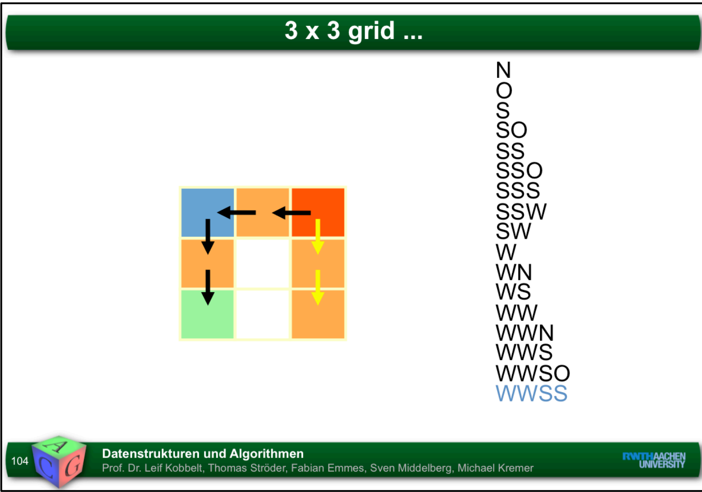

3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
99
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
100
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer


3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
101
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
102
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
103
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY


3 x 3 grid ...
N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS
104
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Ströder, Fabian Emmes, Sven Middelberg, Michael Kremer
RWTH AACHEN UNIVERSITY

Stack

- Merke: Rekursive Algorithmen lassen sich in der Regel mit einer Stack-Datenstruktur auch iterativ formulieren.
- Das LIFO-Prinzip entspricht der Abarbeitungsreihenfolge der geschachtelten Prozeduren (was zuletzt aufgerufen wird, wird als erstes bearbeitet).
- Beispiel: systematische Suche in einem Labyrinth (bei Sackgassen zurückgehen).