Speicherorganisation: der Heap
(am Beispiel von Arrays)

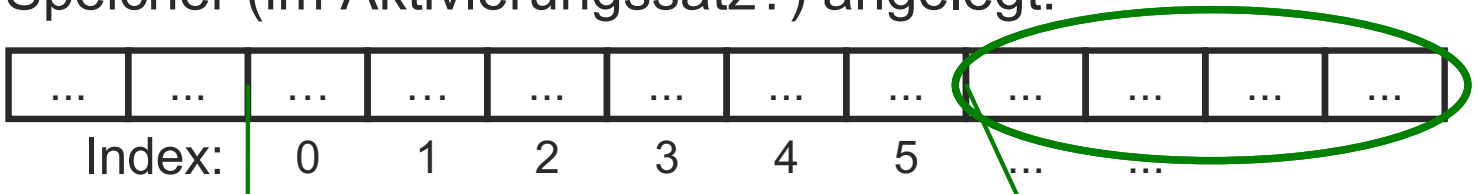
- Array-Verwaltung im Speicher
- Illustration
  - Der Stack
  - Der Heap
- Zuweisung von Arrays
- Speicherbereinigung
- Aliasierung
- Erläuterung

Motivation

- Betrachten wir folgende Array-Vereinbarung:

  int[] numbers = new int[6];

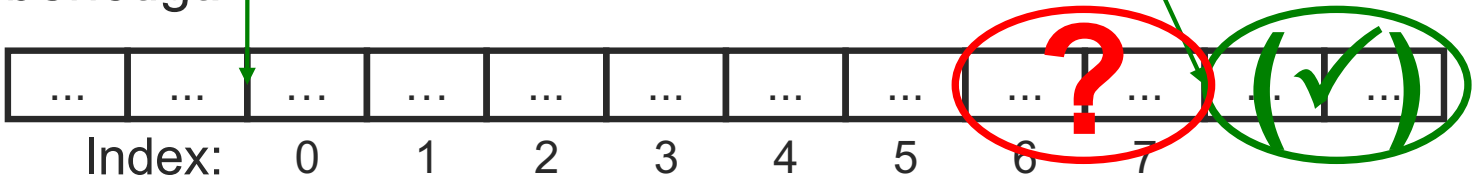
- Jedes Array wird als zusammenhängender Bereich im Speicher (im Aktivierungssatz?) angelegt:



- Neue Zuweisungen an die Array-Variable sind jederzeit möglich:

  numbers = new int[8]; // neues Array anlegen

- Dafür wird jetzt aber mehr Speicher (im Aktivierungssatz?) benötigt:

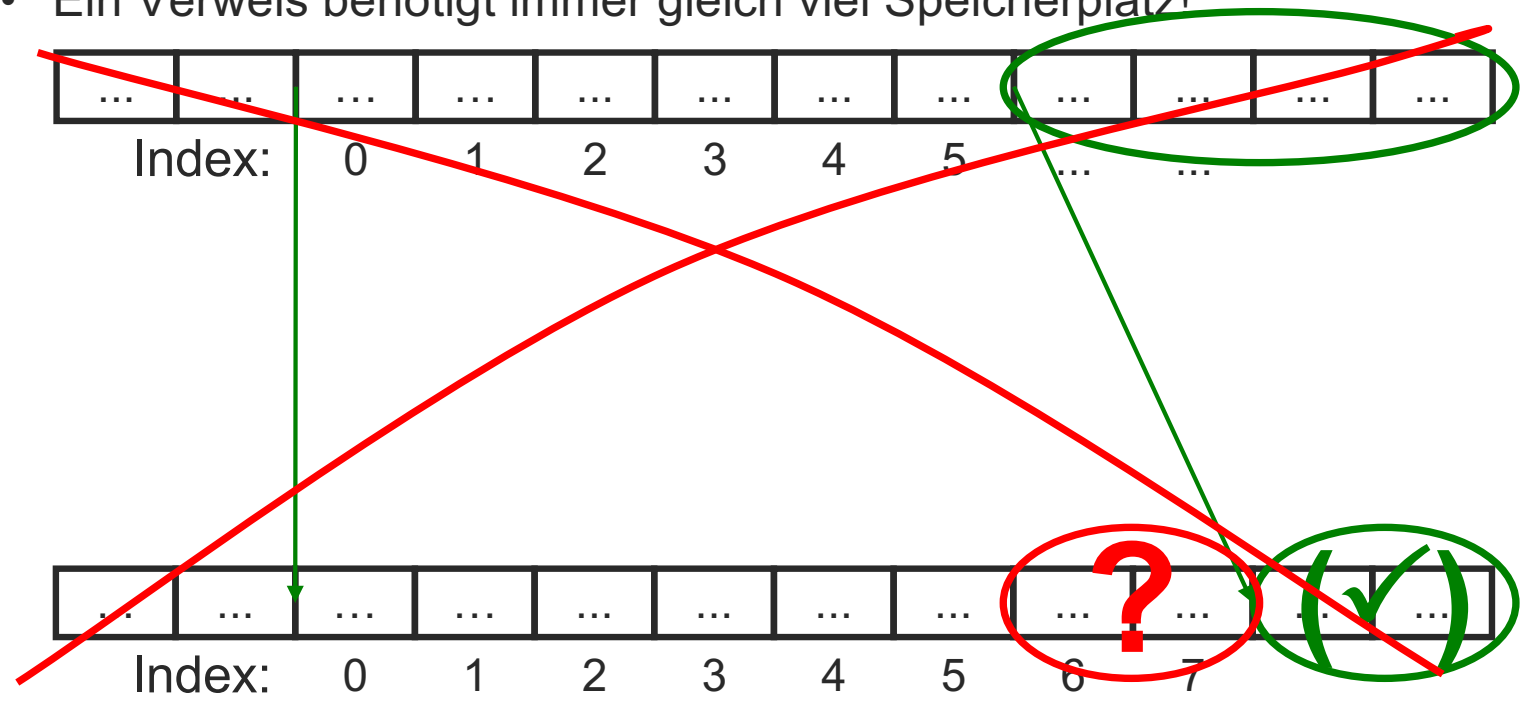


- Frage: Was passiert mit Variablen, die (im Aktivierungssatz) nach dem *numbers*-Array abgelegt sind?

Motivation

Bessere Organisation im Speicher:

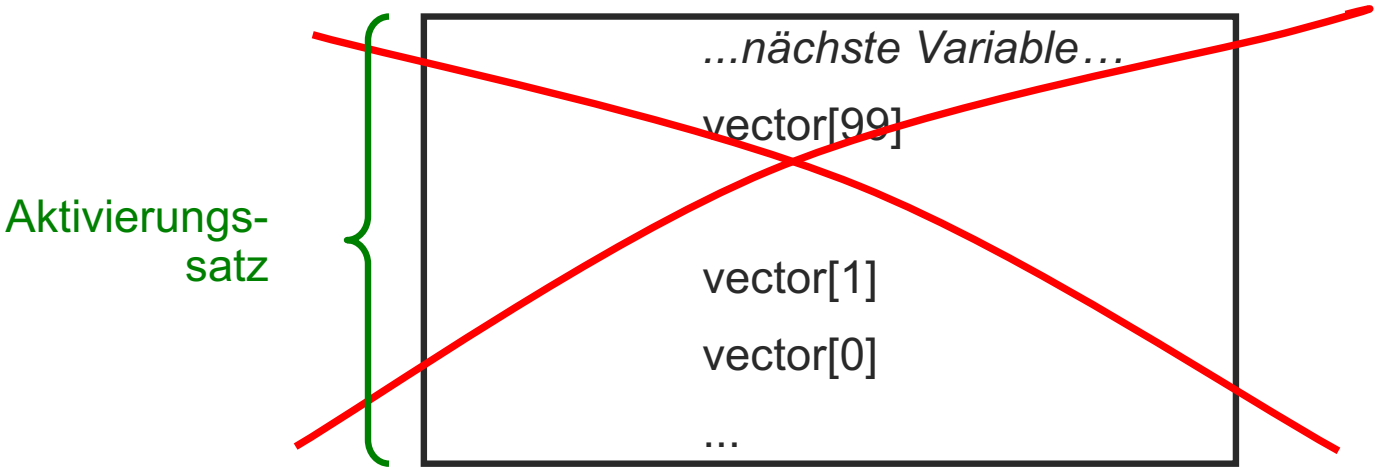
- Das Array wird „an einer anderen Stelle" abgelegt
- An der Stelle der Variablen steht nur ein *Verweis* auf den tatsächlichen Speicherort
- Ein Verweis benötigt immer gleich viel Speicherplatz!



Gleicher Sachverhalt, andere Darstellung

Beispiel:

```
int [ ]  vector = new int[100];
```



So haben wir das bisher aufgezeichnet, so kann es aber nicht funktionieren

Im Aktivierungssatz bei den "normalen" Variablen wird ein sogenannter **Array-Deskriptor** abgelegt

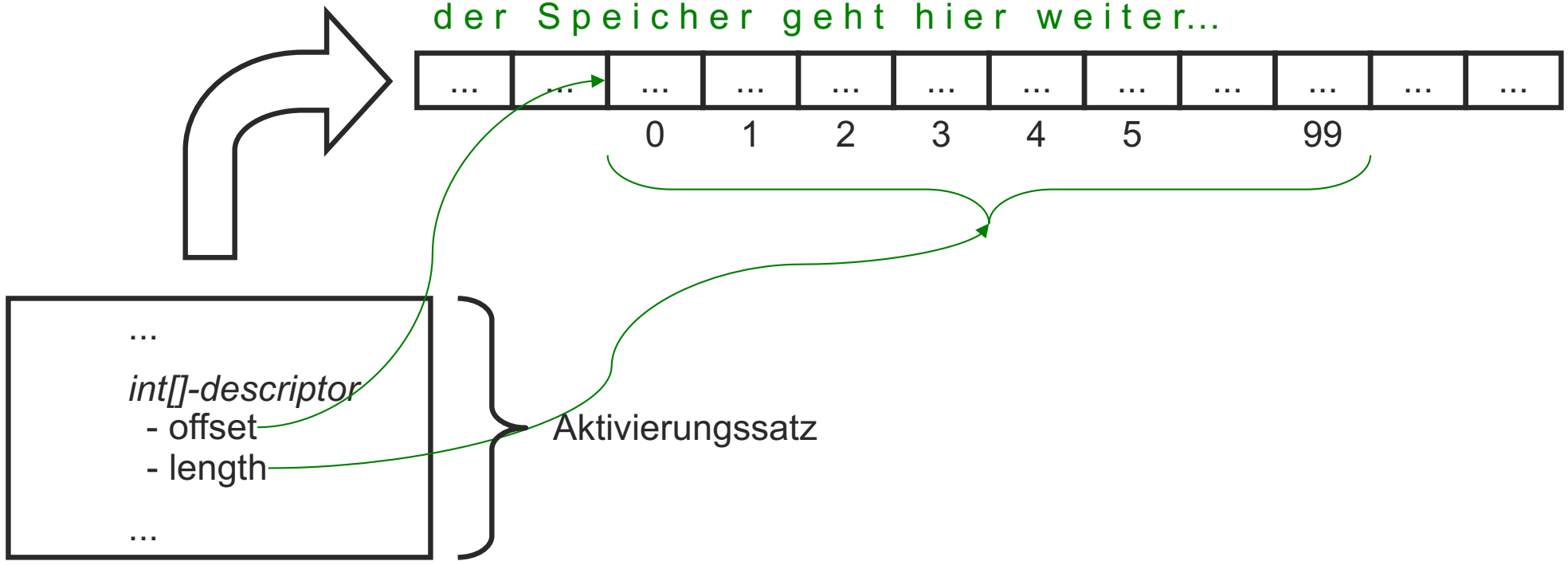
Das eigentliche Array wird an anderer Stelle im Speicher abgelegt

Was gehört in einen Deskriptor?

Deskriptor-Verwendung

Zu einem Array-Deskriptor gehören

- die Startadresse (englisch: offset) des eigentlichen Arrays im Speicher und
- die Länge (englisch: length) des Arrays



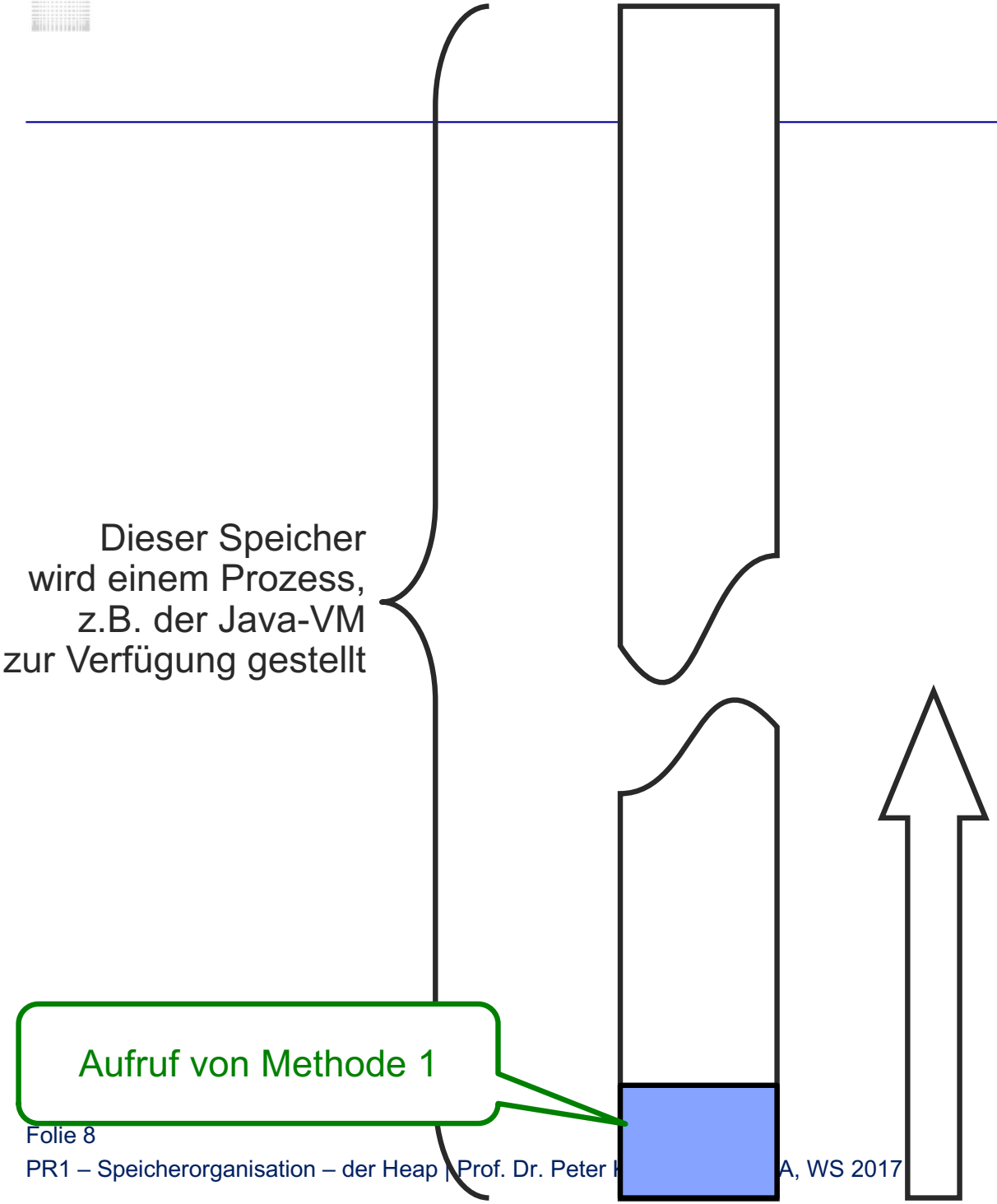
Zur Erinnerung

- Von "Unten" nach "Oben" im Speicher "wächst" der ***Stack***
- Das ist der Bereich, in dem der Speicher für die *Aktivierungssätze* von Methoden (für Variablen, Parameter und Rückgabewerte) reserviert und wieder freigegeben wird

- Von "Oben" nach "Unten" im Speicher "wächst" der ***Heap***
- Das ist der Bereich, in dem unter anderem *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

- Dieser Sachverhalt wird nun illustriert, zunächst der Stack

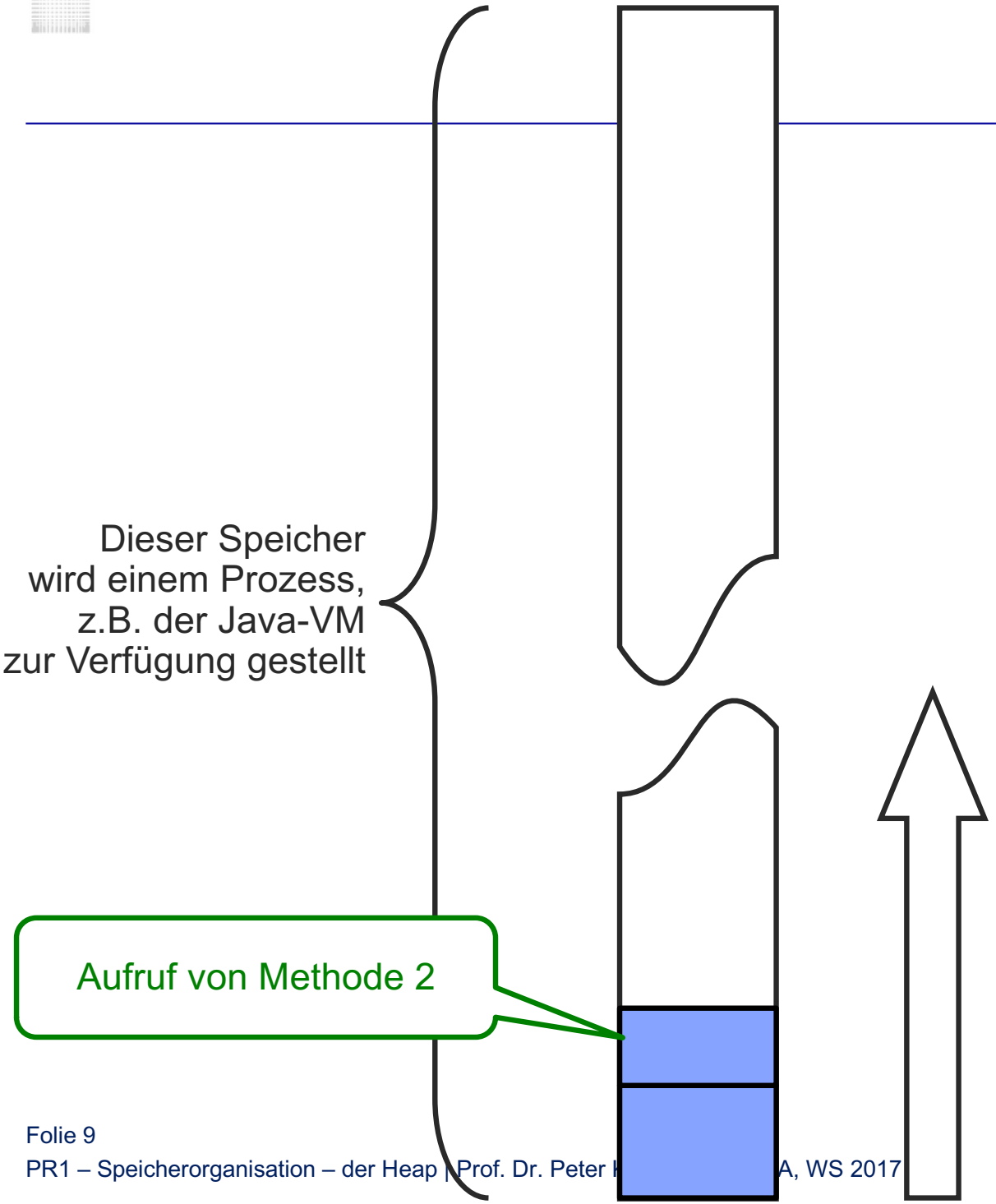
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

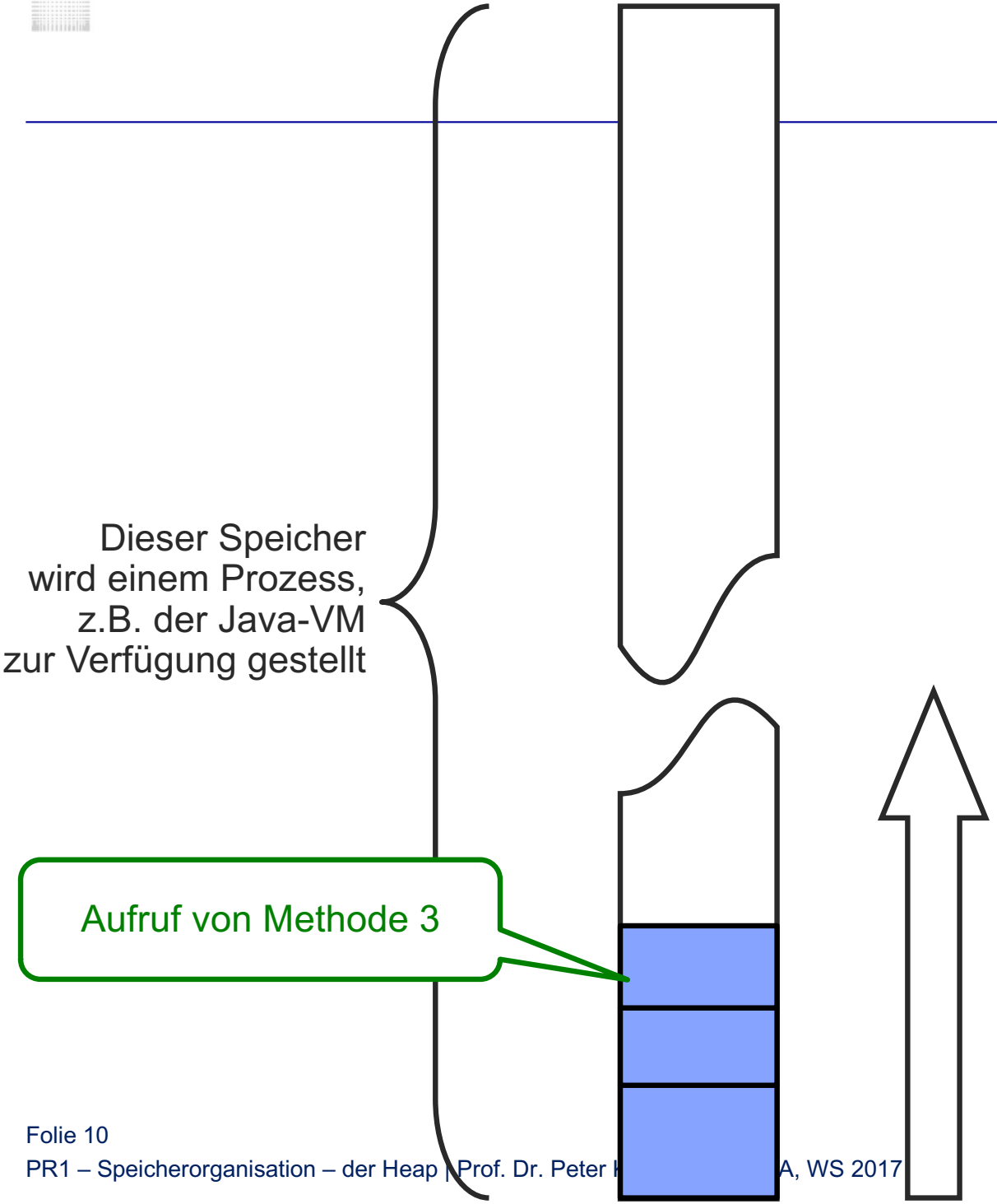
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

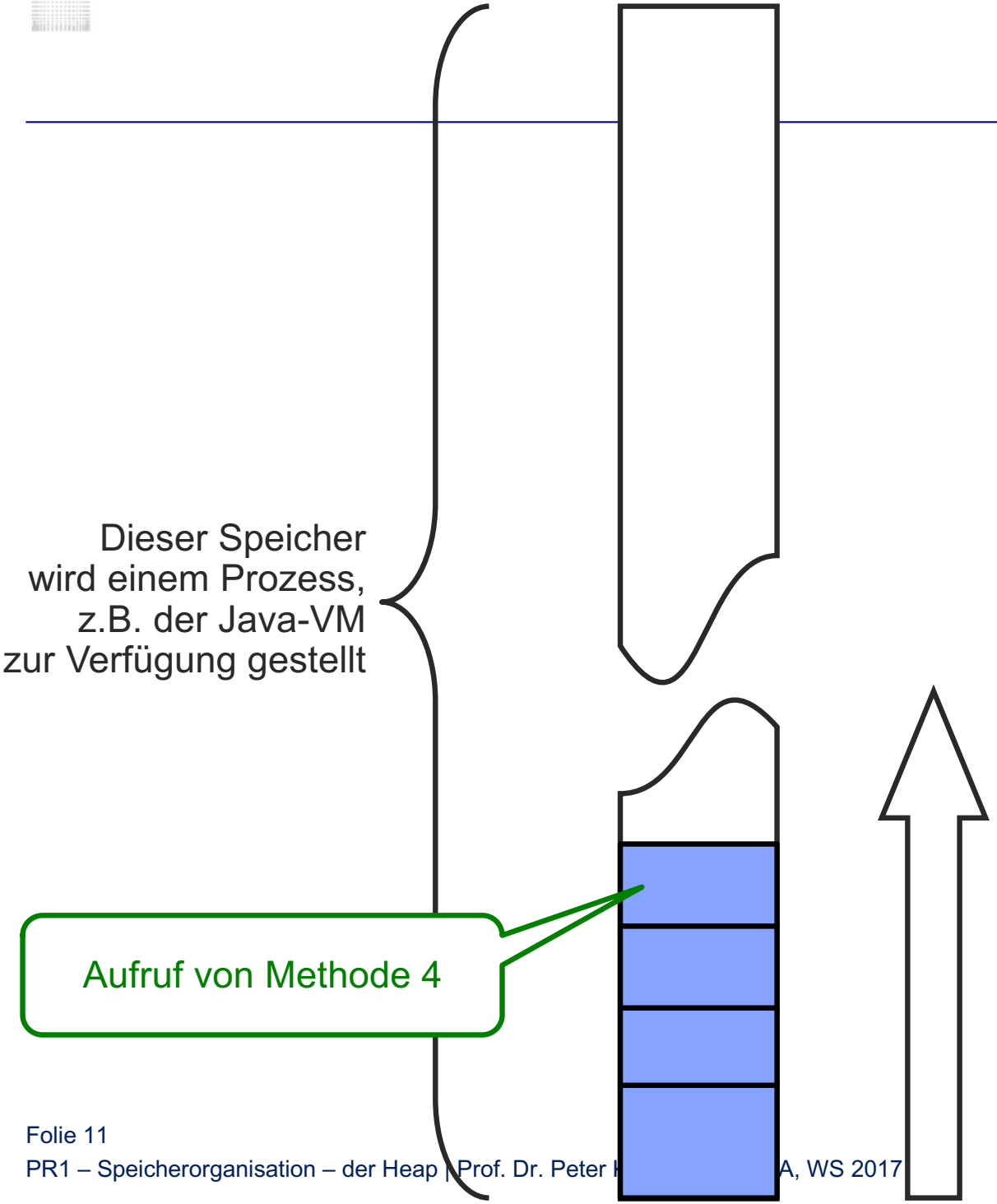
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

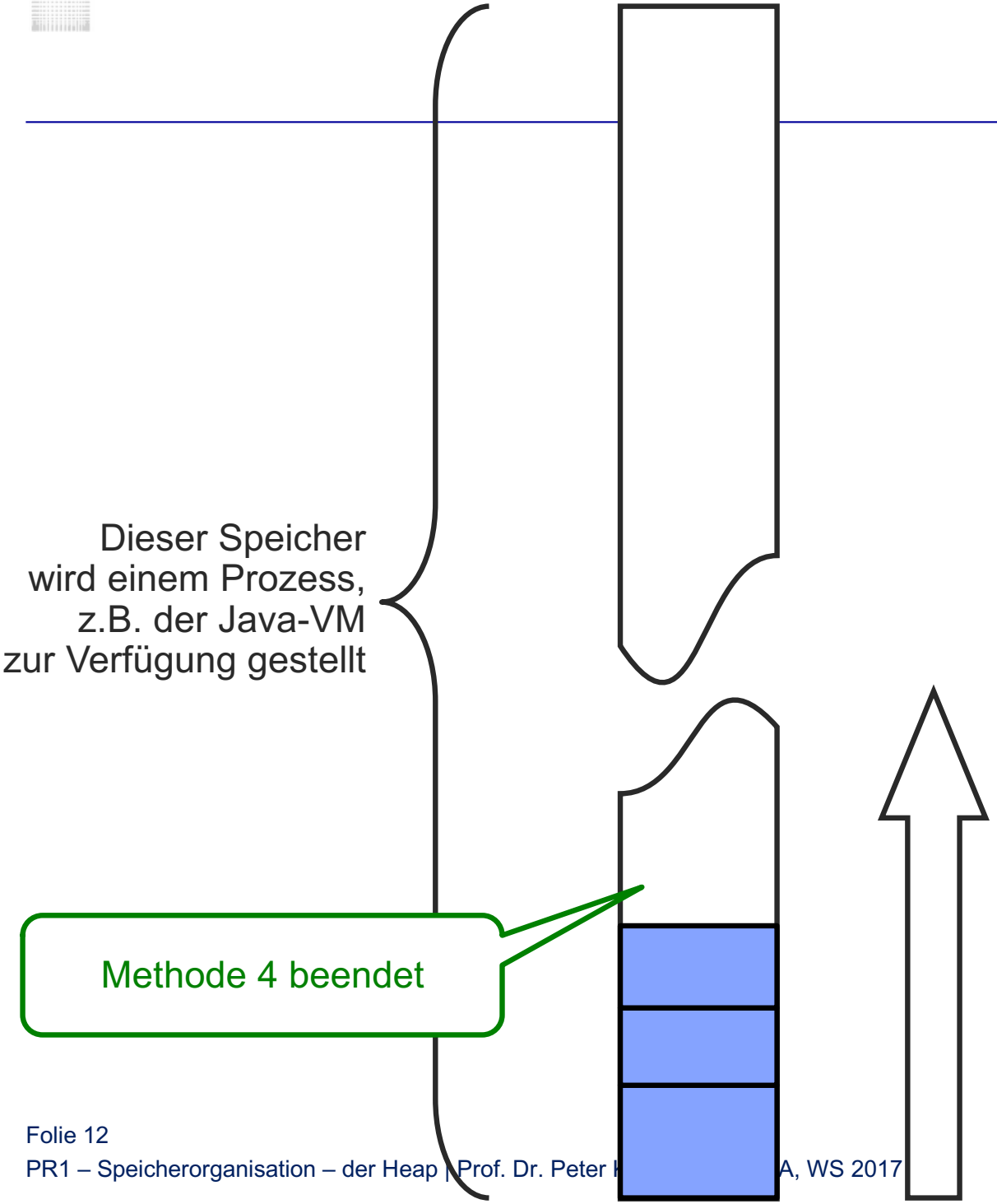
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

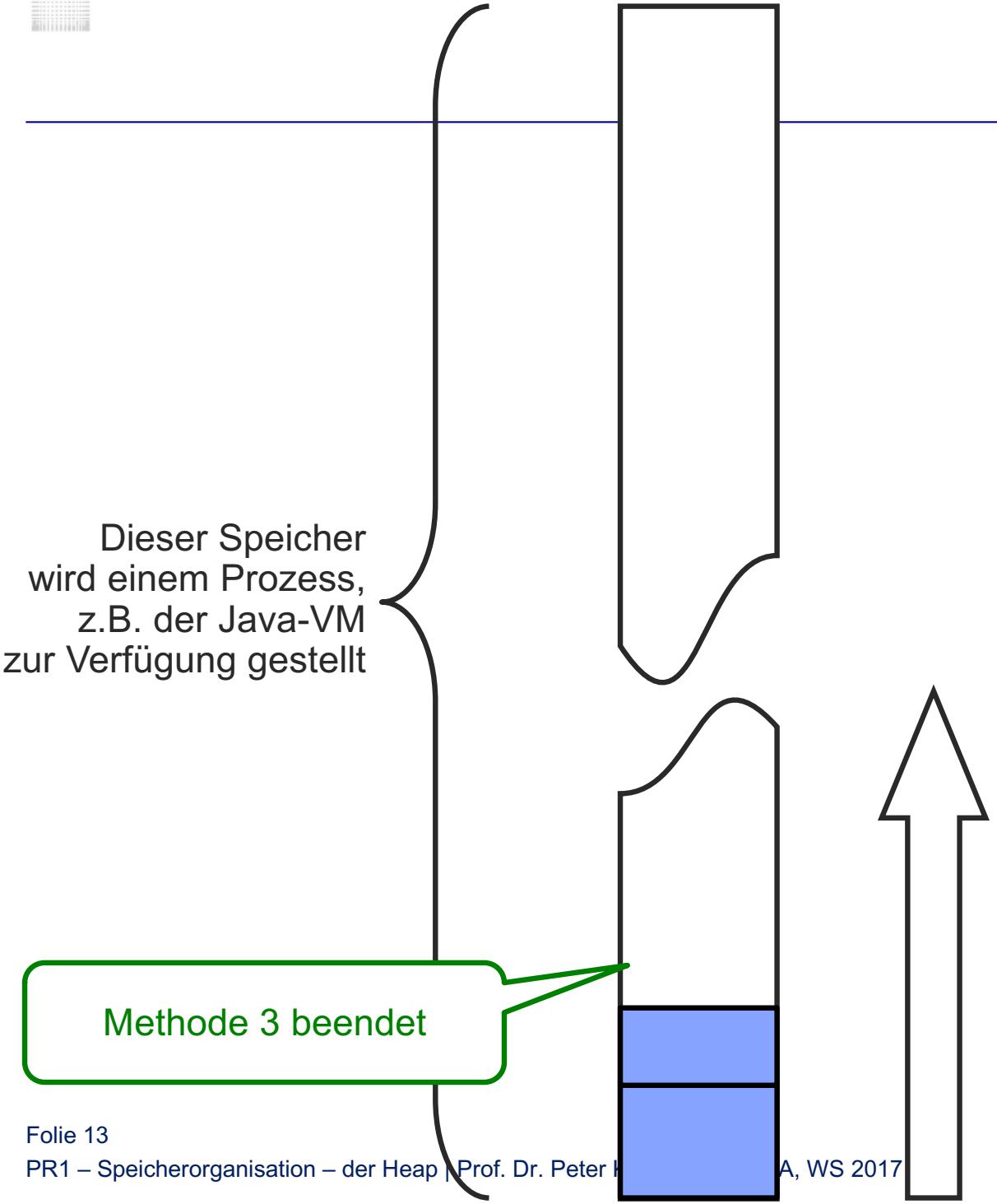
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

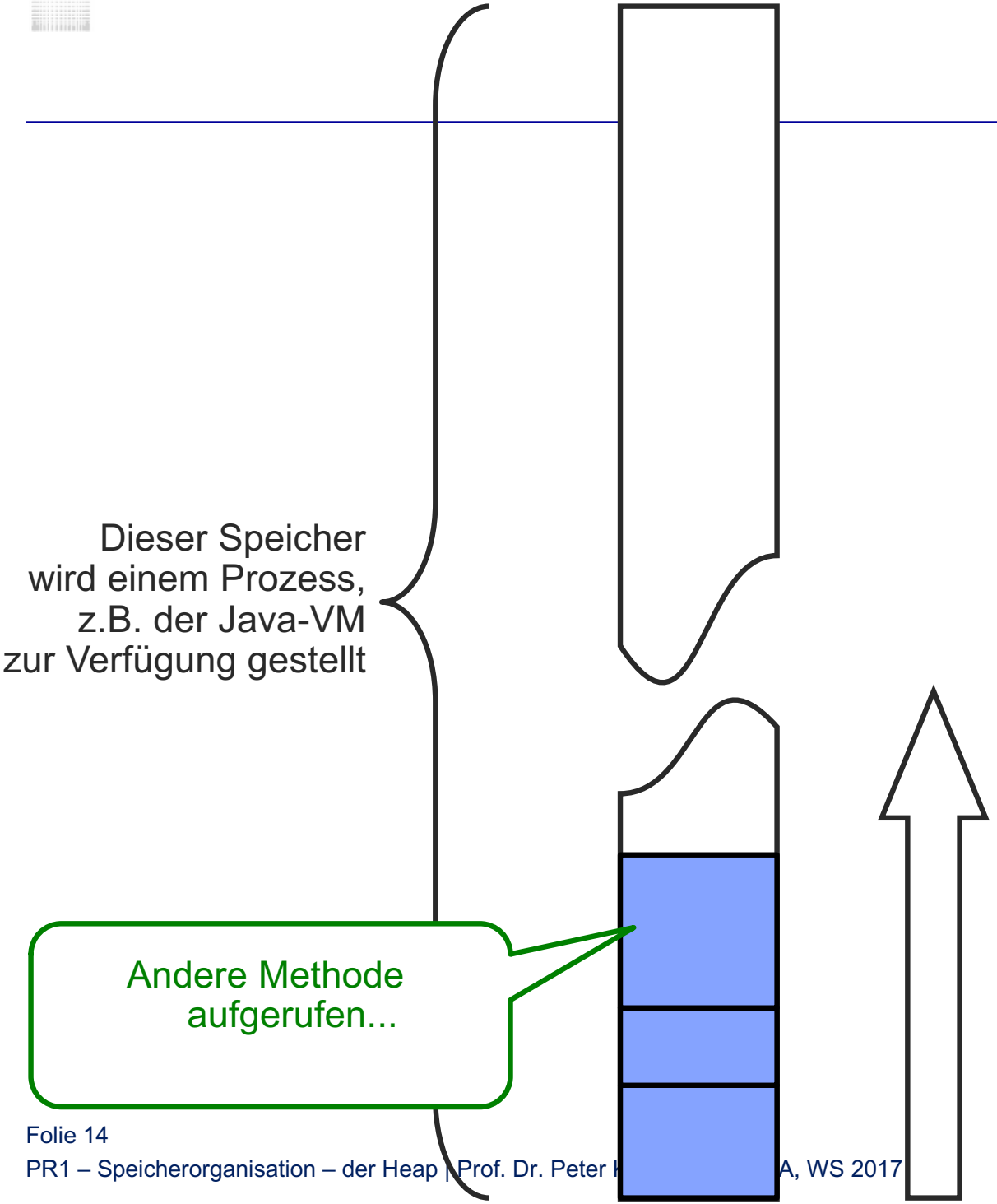
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

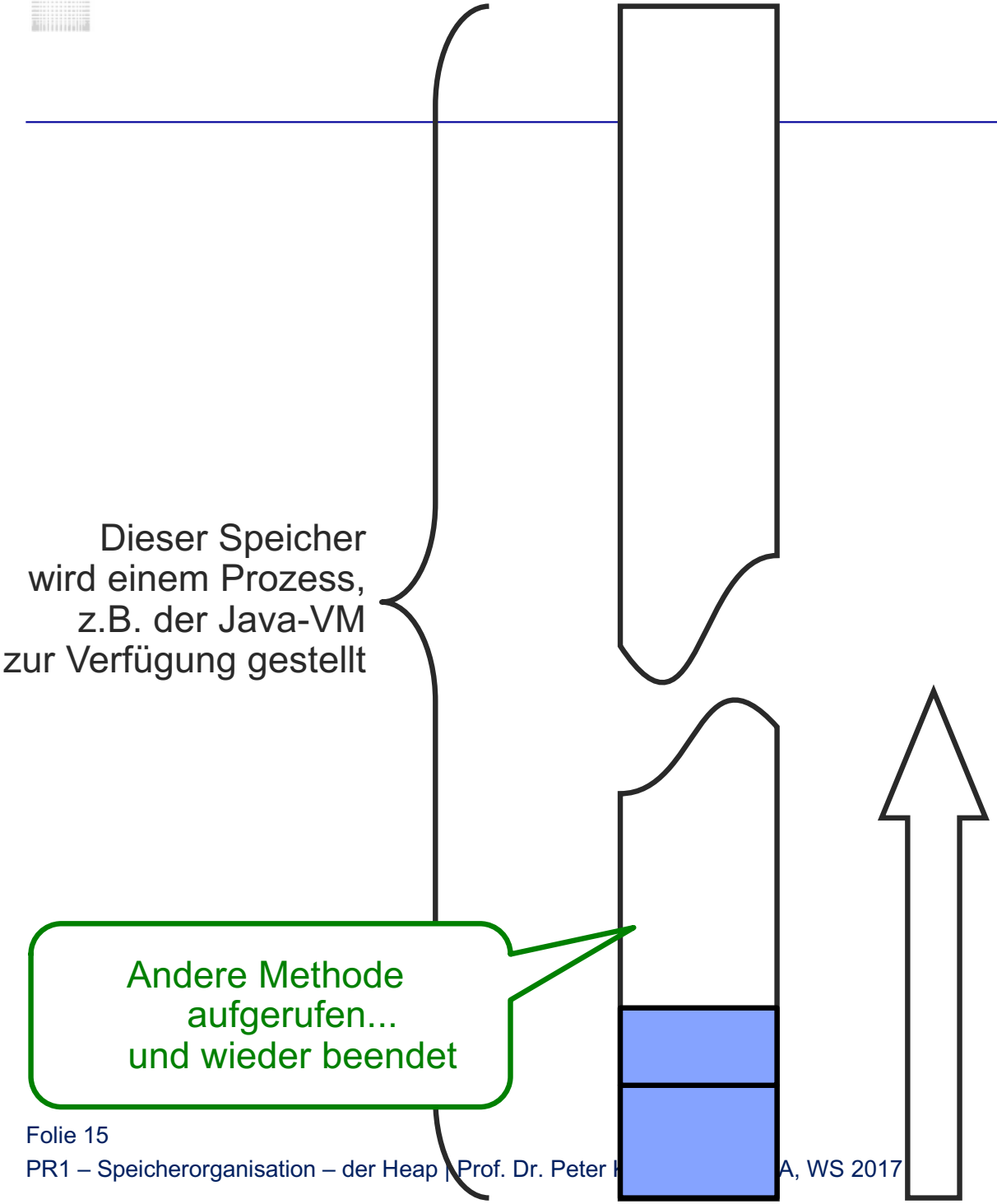
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

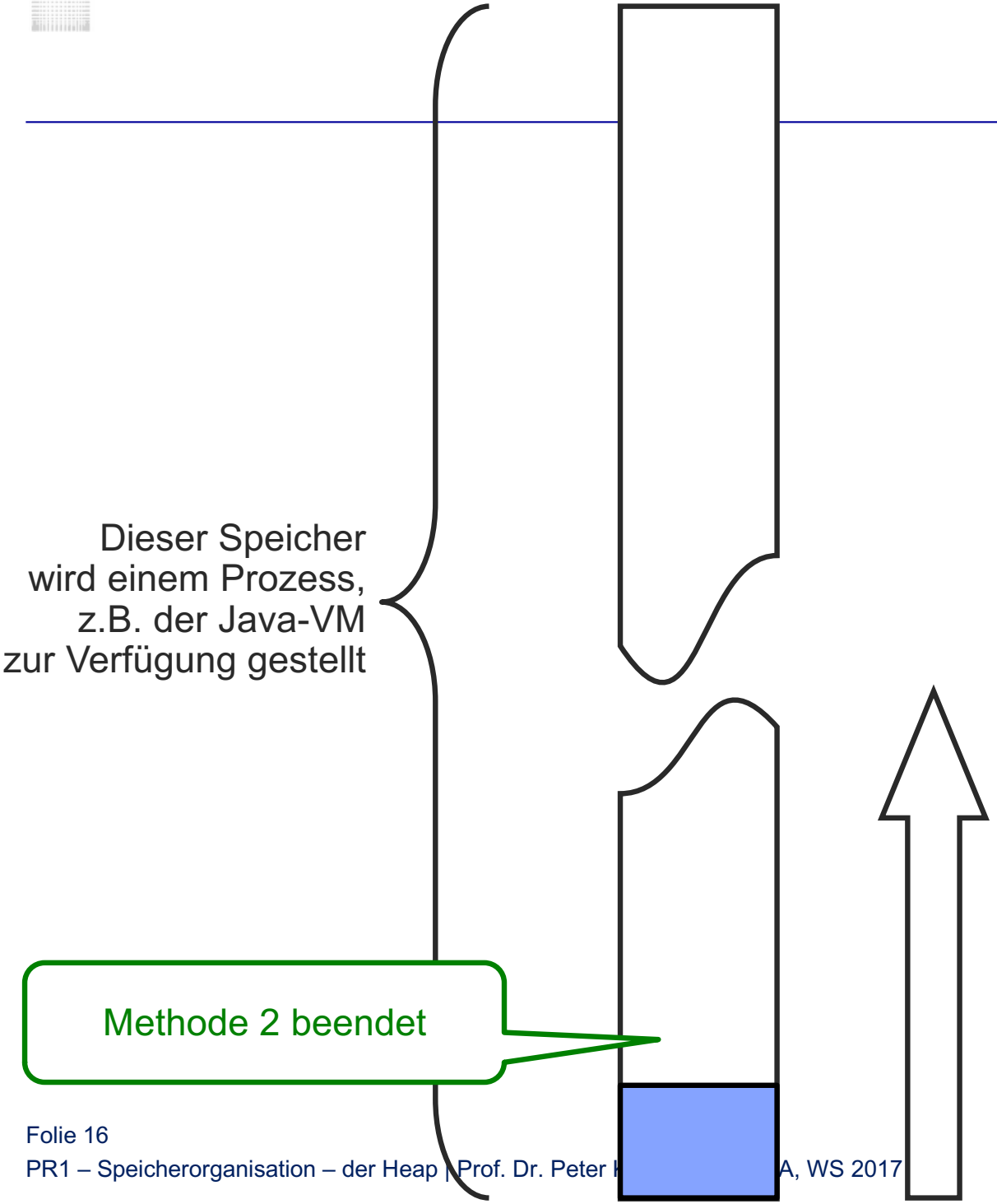
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

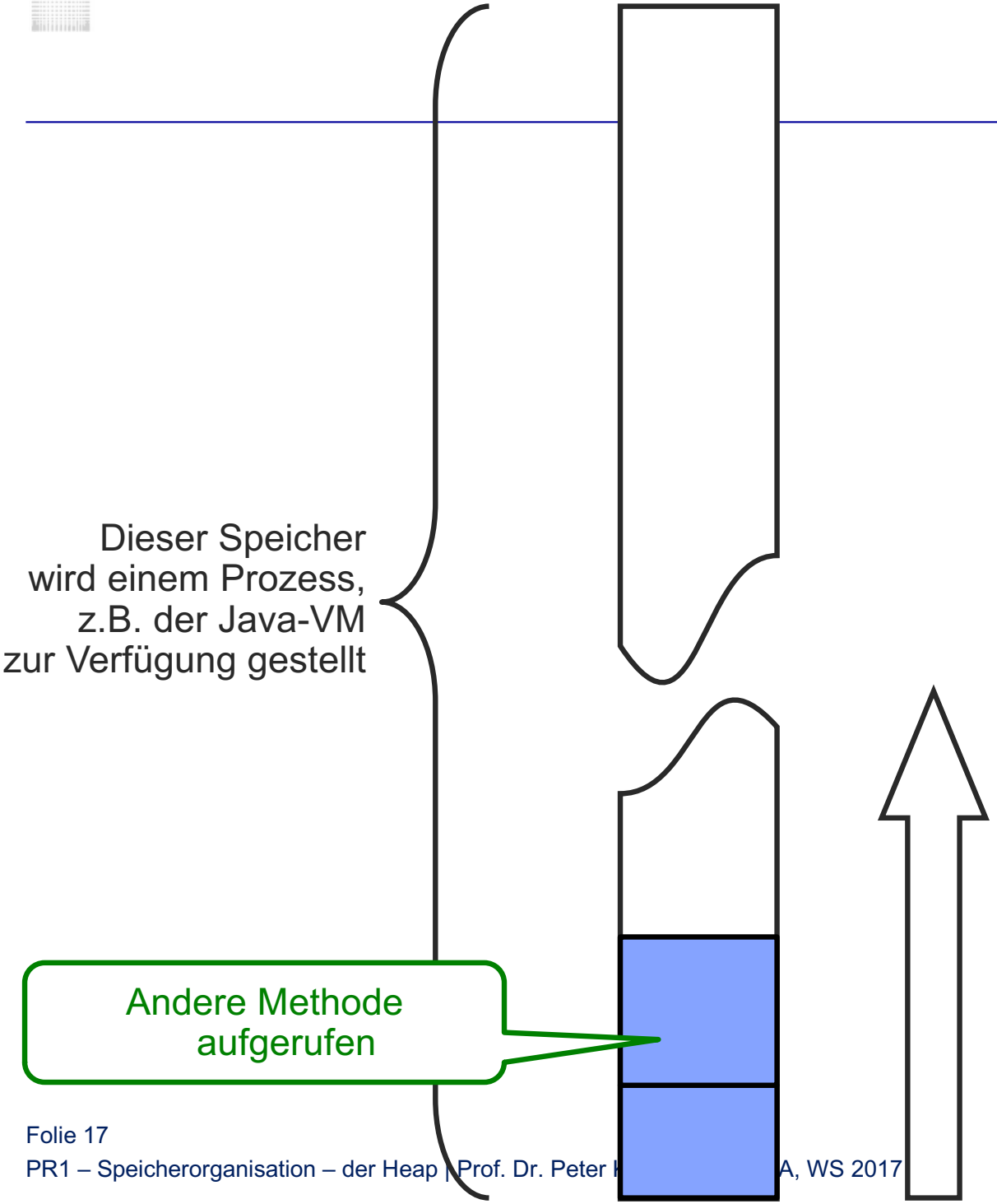
Illustration: Der Stack



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

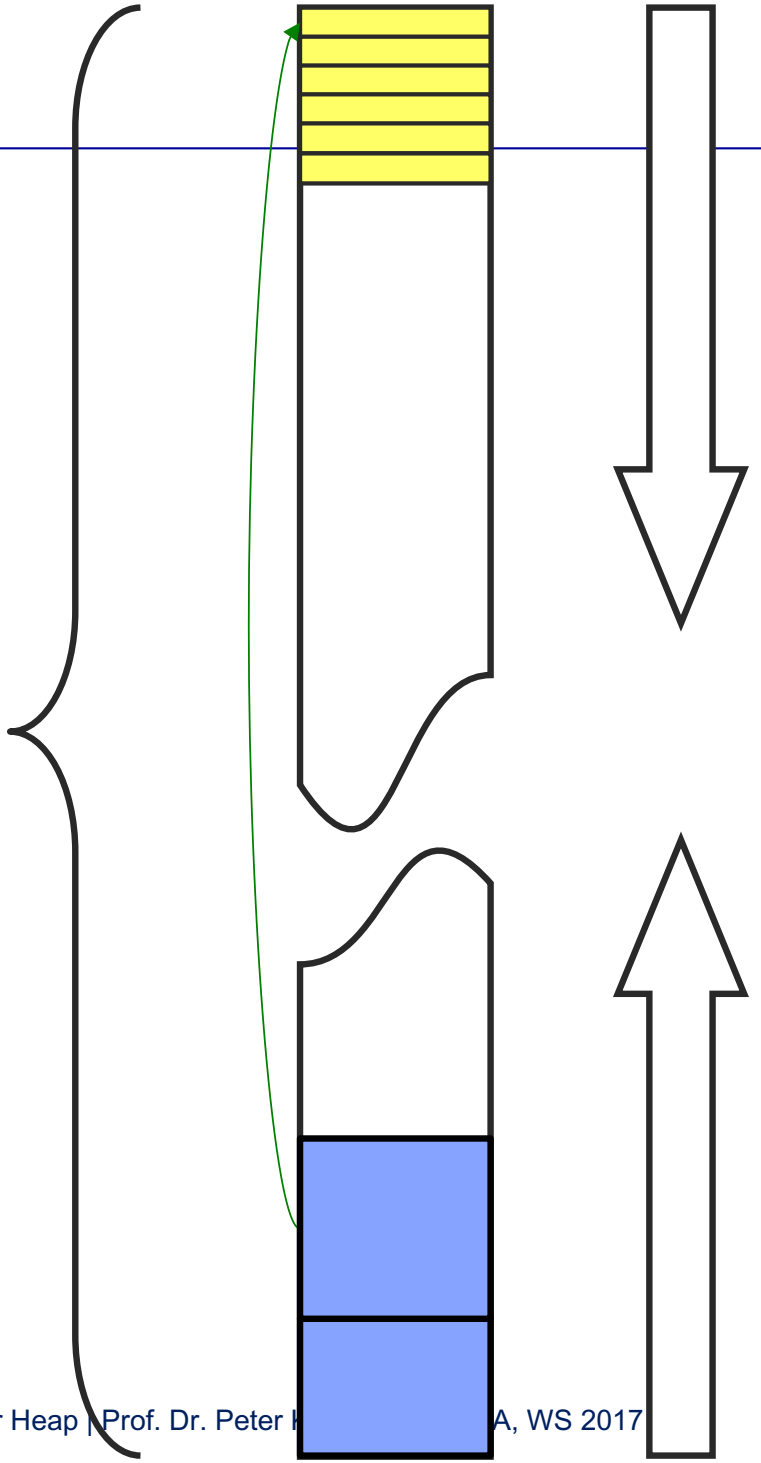
Illustration: Der Heap



Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Von "Oben" nach "Unten" im Speicher wächst der *Heap*

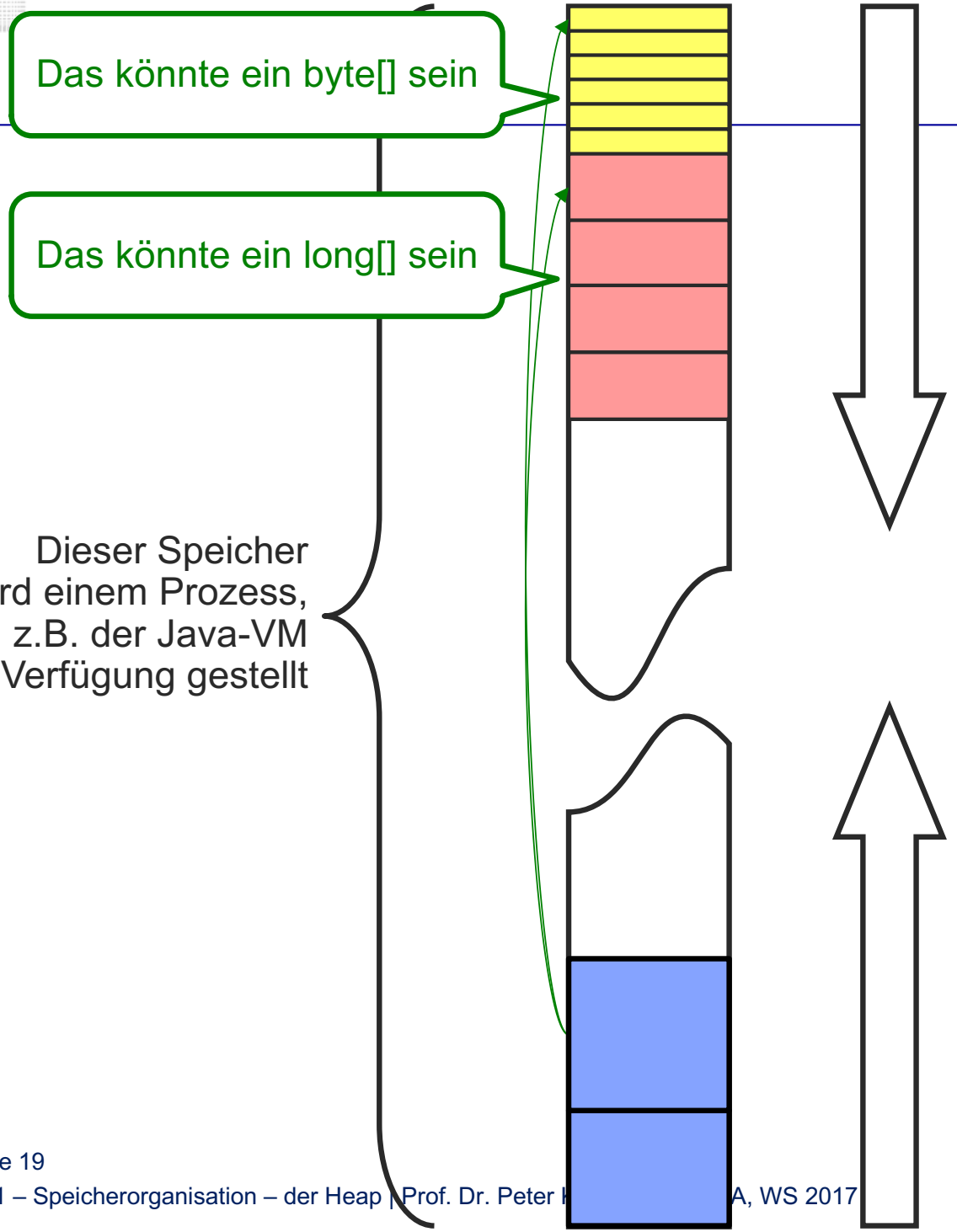
Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Von "Oben" nach "Unten" im Speicher wächst der *Heap*

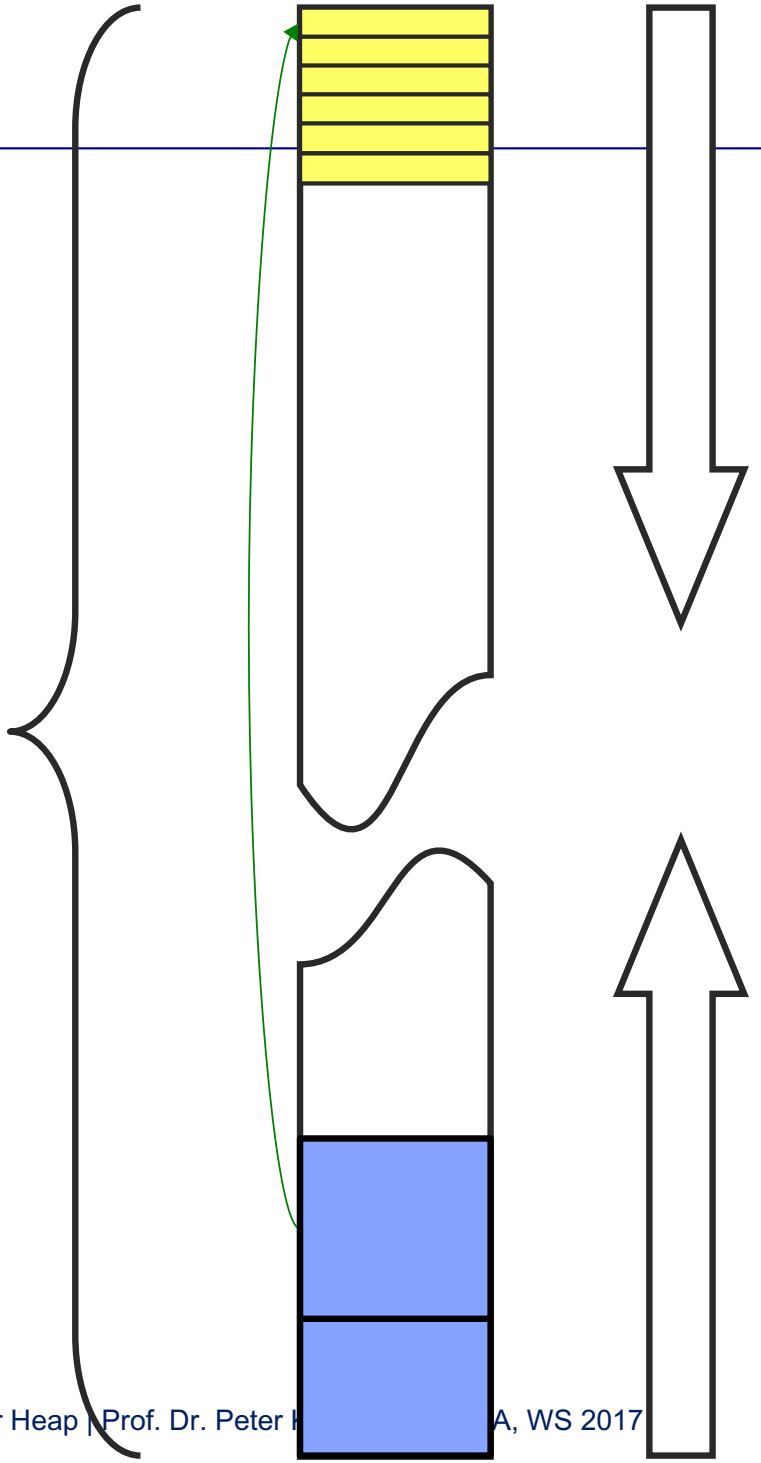
Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Von "Oben" nach "Unten" im Speicher wächst der *Heap*

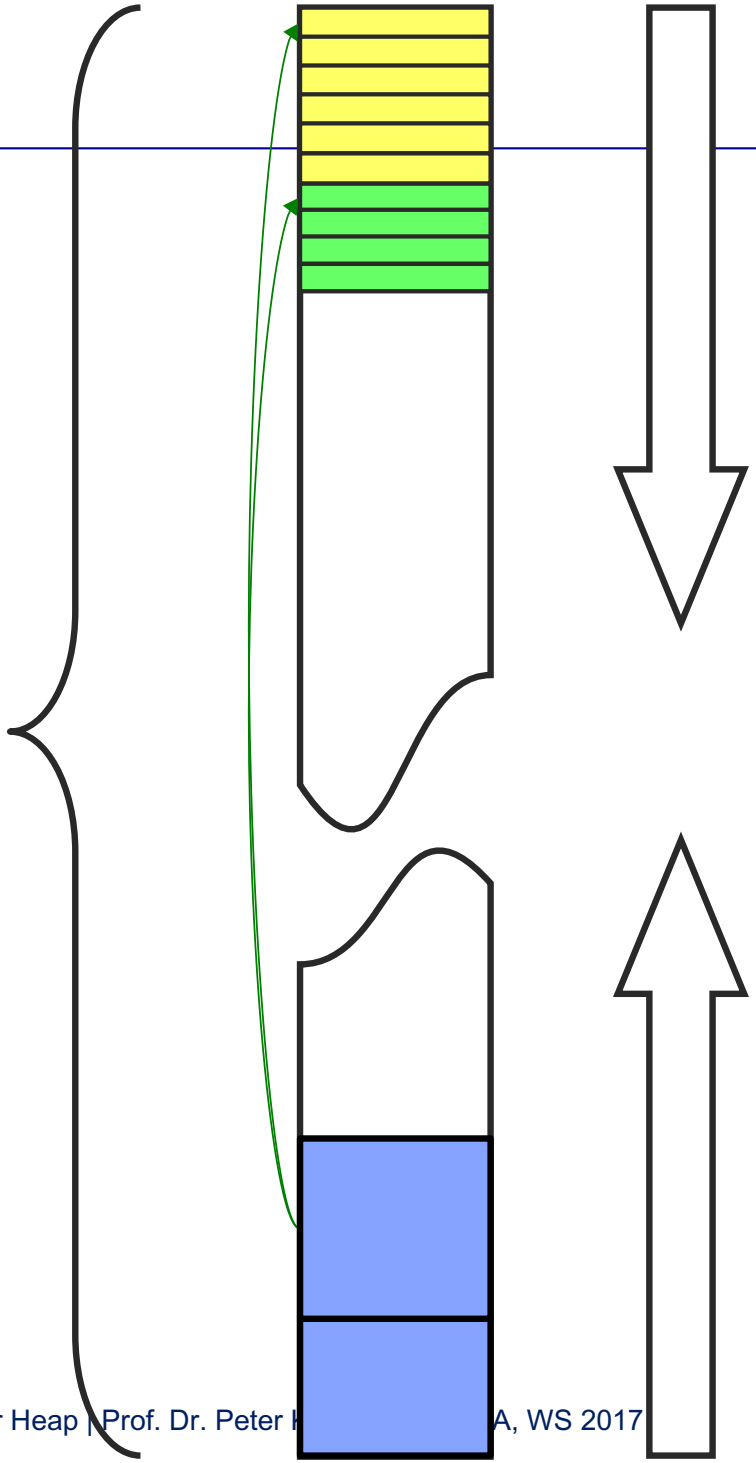
Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Von "Oben" nach "Unten" im Speicher wächst der *Heap*

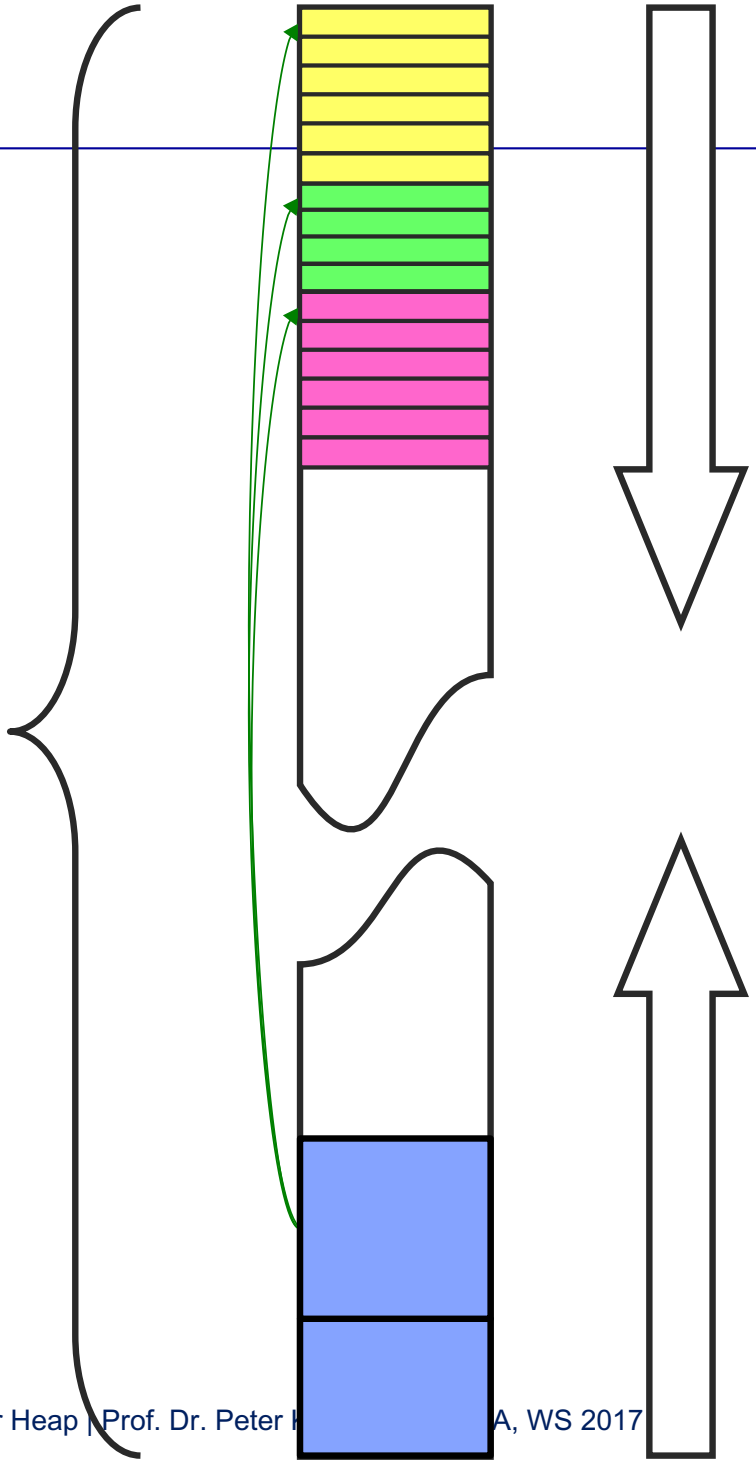
Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

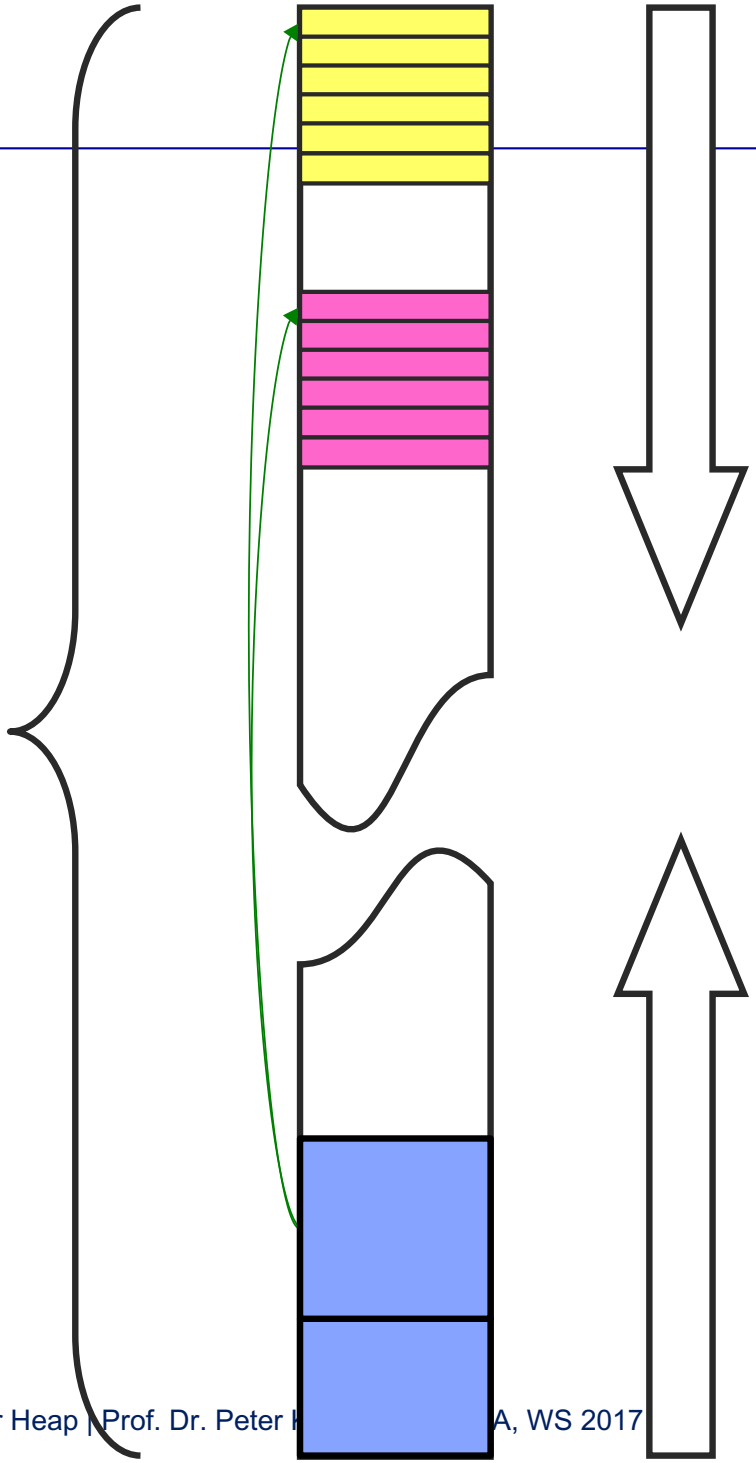
Von "Oben" nach "Unten" im Speicher wächst der *Heap*

Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

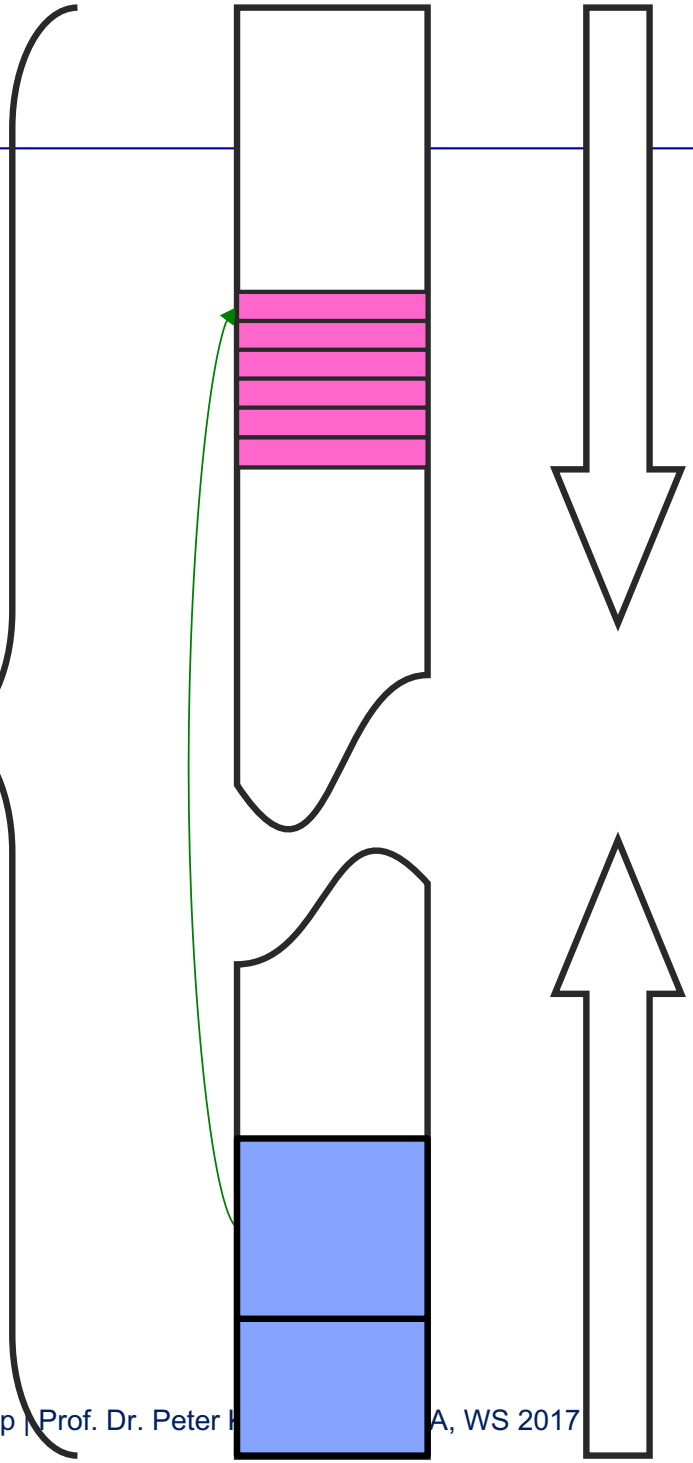
Von "Oben" nach "Unten" im Speicher wächst der *Heap*

Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

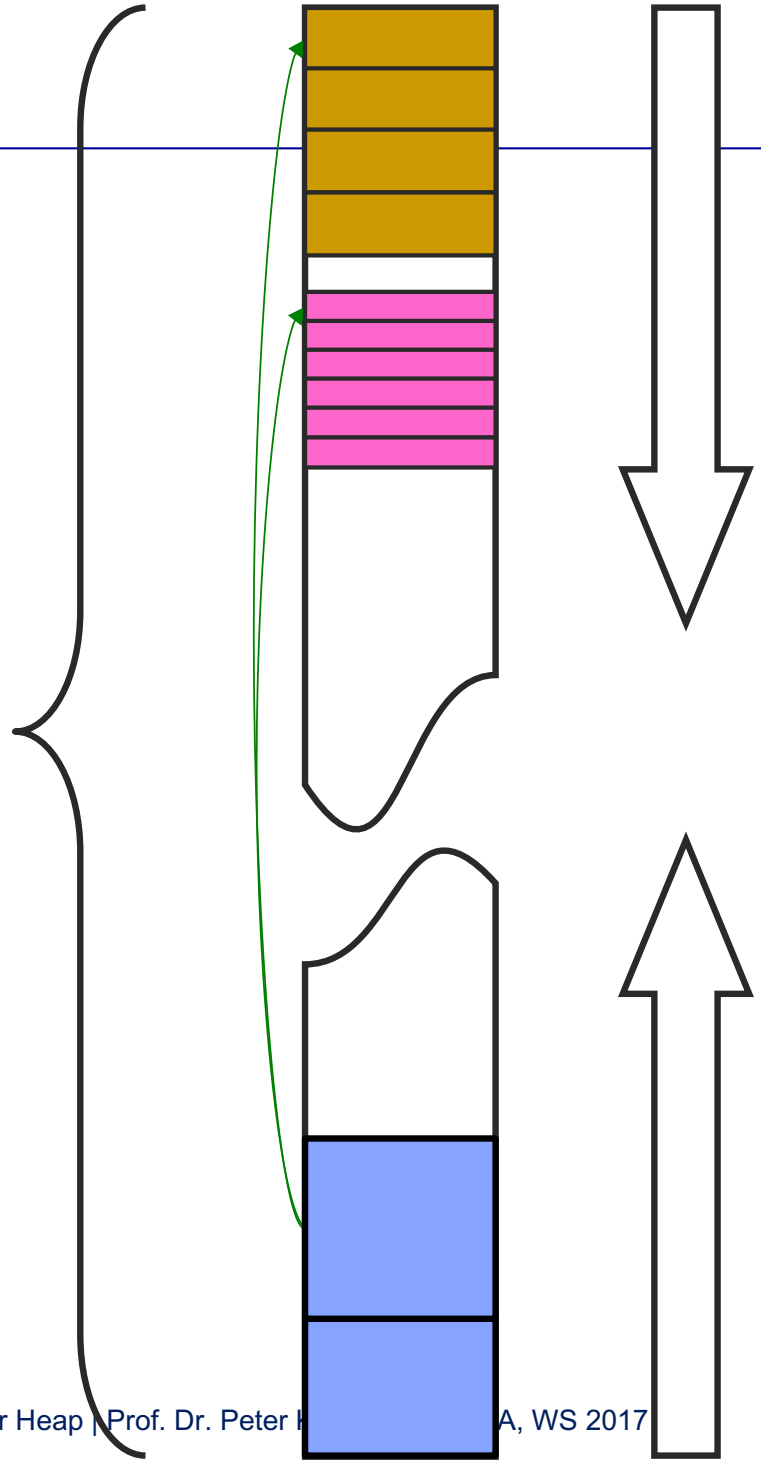
Von "Oben" nach "Unten" im Speicher wächst der *Heap*

Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

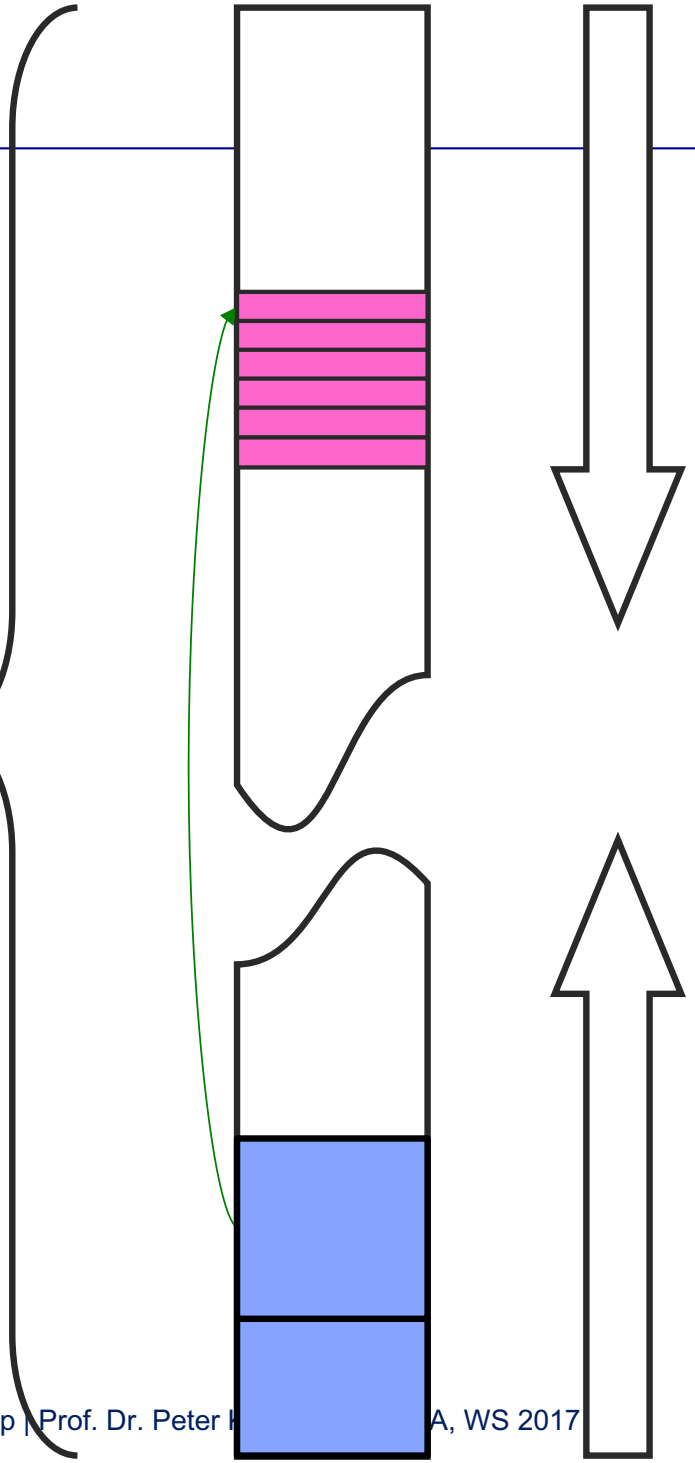
Von "Oben" nach "Unten" im Speicher wächst der *Heap*

Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

Illustration: Der Heap



Von "Oben" nach "Unten" im Speicher wächst der *Heap*

Das ist der Bereich, in dem z.B. *Speicherplatz für Arrays* belegt (und wieder freigegeben) wird

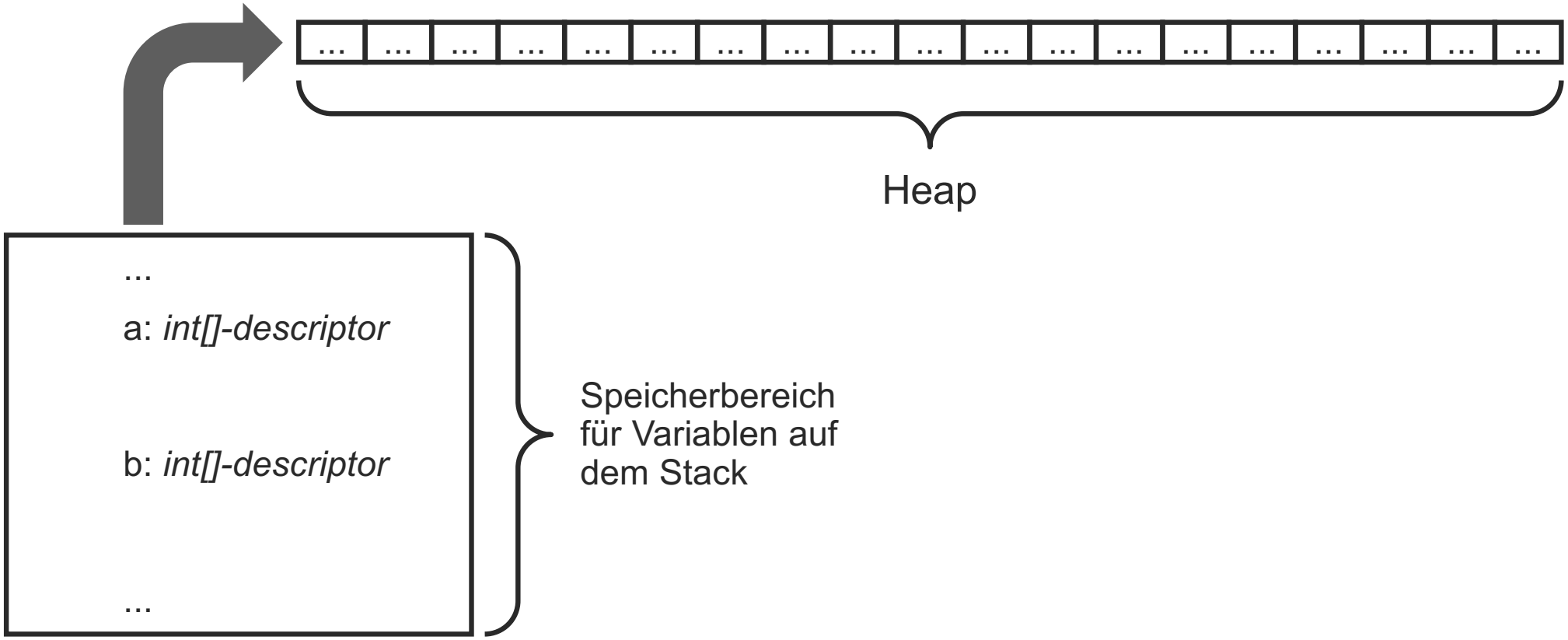
Dieser Speicher wird einem Prozess, z.B. der Java-VM zur Verfügung gestellt

Von "Unten" nach "Oben" im Speicher wächst der *Stack*

Das ist der Bereich, in dem Speicher für die Aktivierungssätze von Methoden reserviert und wieder freigegeben wird

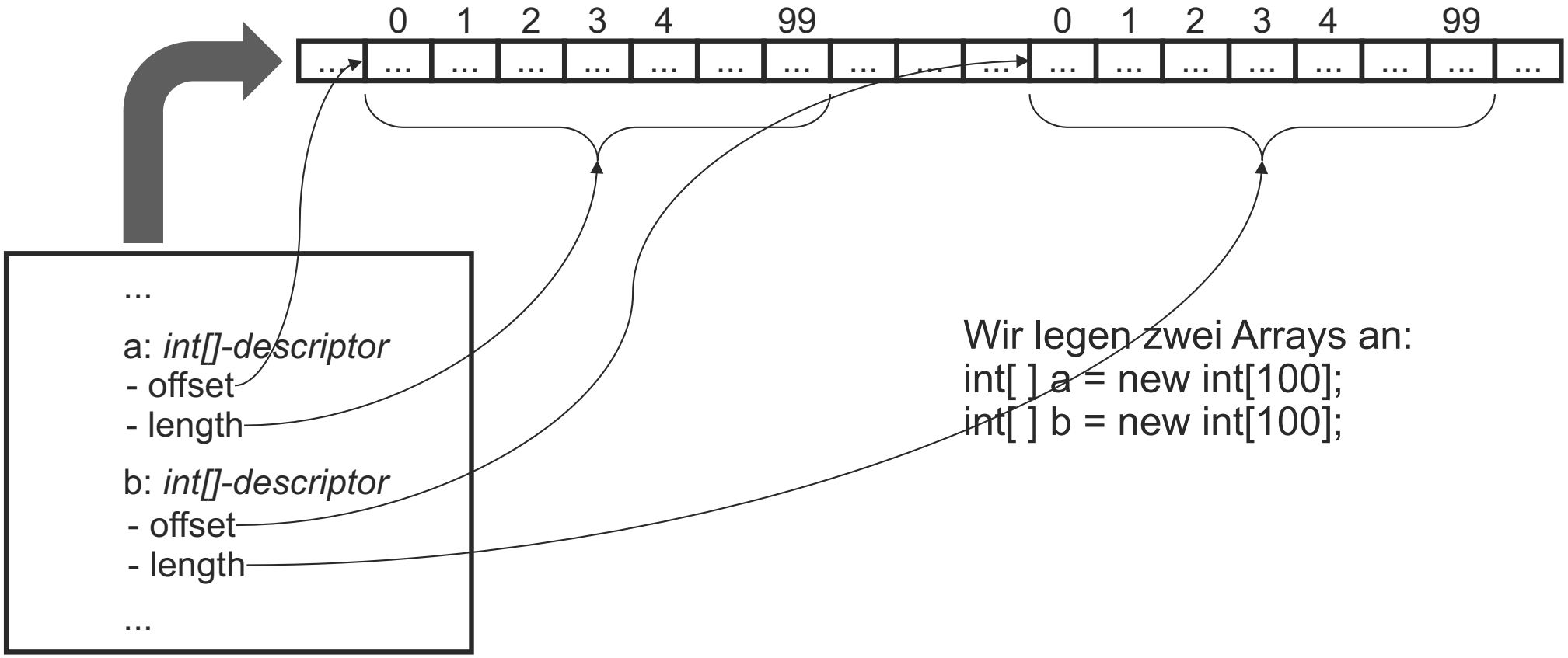
Zuweisung von (ganzen) Arrays

- Bei der Zuweisung eines Arrays wird *nicht* der Array*inhalt*, *sondern* nur sein *Deskriptor* zugewiesen
- Im Speicher sieht das wie folgt aus



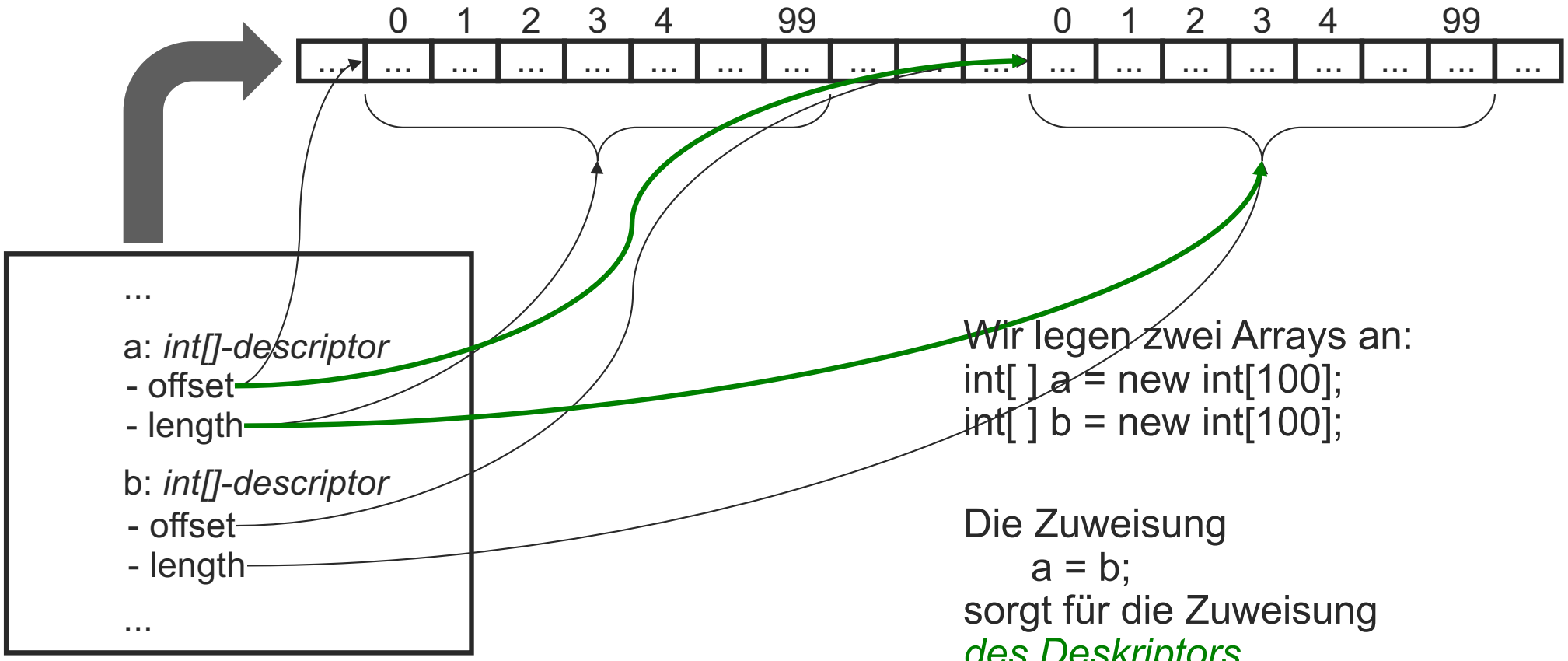
Zuweisung von (ganzen) Arrays

- Bei der Zuweisung eines Arrays wird *nicht* der Array*inhalt*, *sondern* nur sein *Deskriptor* zugewiesen
- Im Speicher sieht das wie folgt aus



Zuweisung von (ganzen) Arrays

- Bei der Zuweisung eines Arrays wird *nicht* der Array*inhalt*, *sondern* nur sein *Deskriptor* zugewiesen
- Im Speicher sieht das wie folgt aus

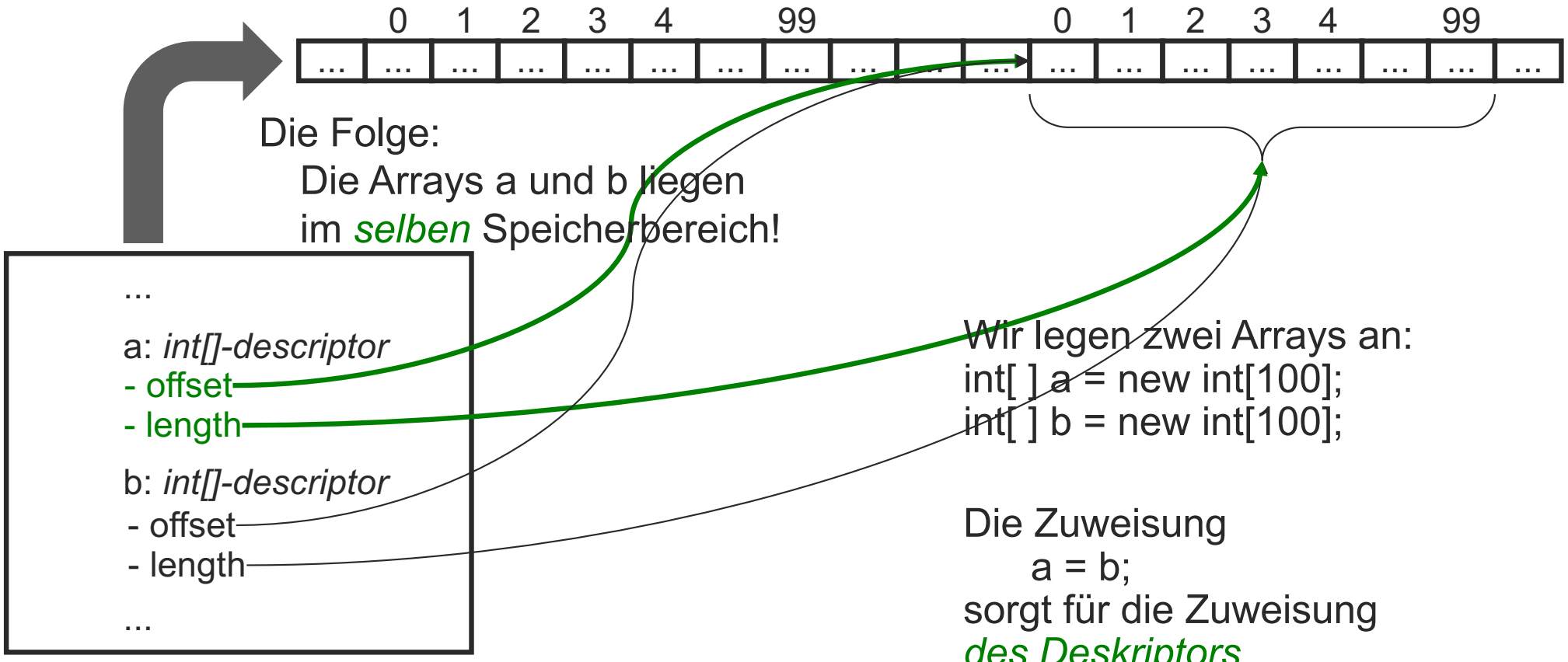


Wir legen zwei Arrays an:
int[ ] a = new int[100];
int[ ] b = new int[100];

Die Zuweisung
a = b;
sorgt für die Zuweisung
des Deskriptors

Zuweisung von (ganzen) Arrays

- Bei der Zuweisung eines Arrays wird *nicht* der Array*inhalt*, *sondern* nur sein *Deskriptor* zugewiesen
- Im Speicher sieht das wie folgt aus



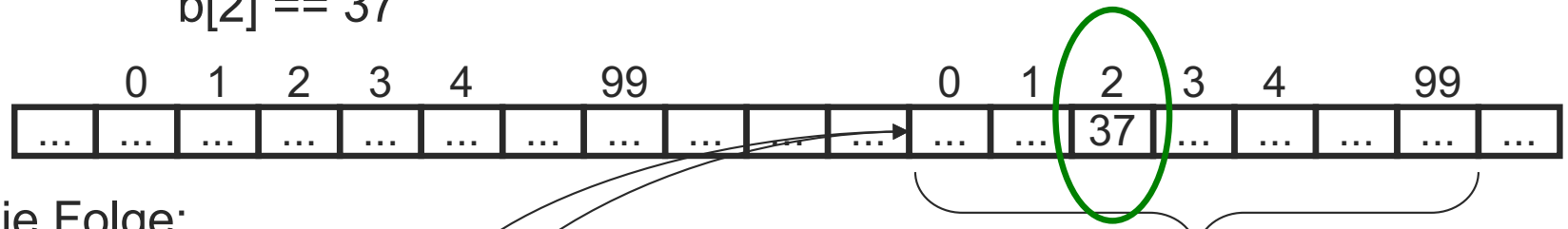
Zuweisung von (ganzen) Arrays

Beispiel: Die Zuweisung

a[2] = 37;

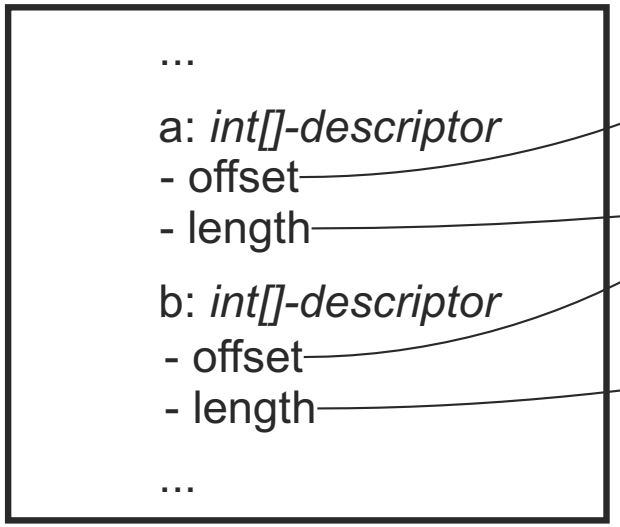
bewirkt

b[2] == 37



Die Folge:

Die Arrays a und b liegen im *selben* Speicherbereich!



Wir legen zwei Arrays an:

```
int[ ] a = new int[100];
int[ ] b = new int[100];
```

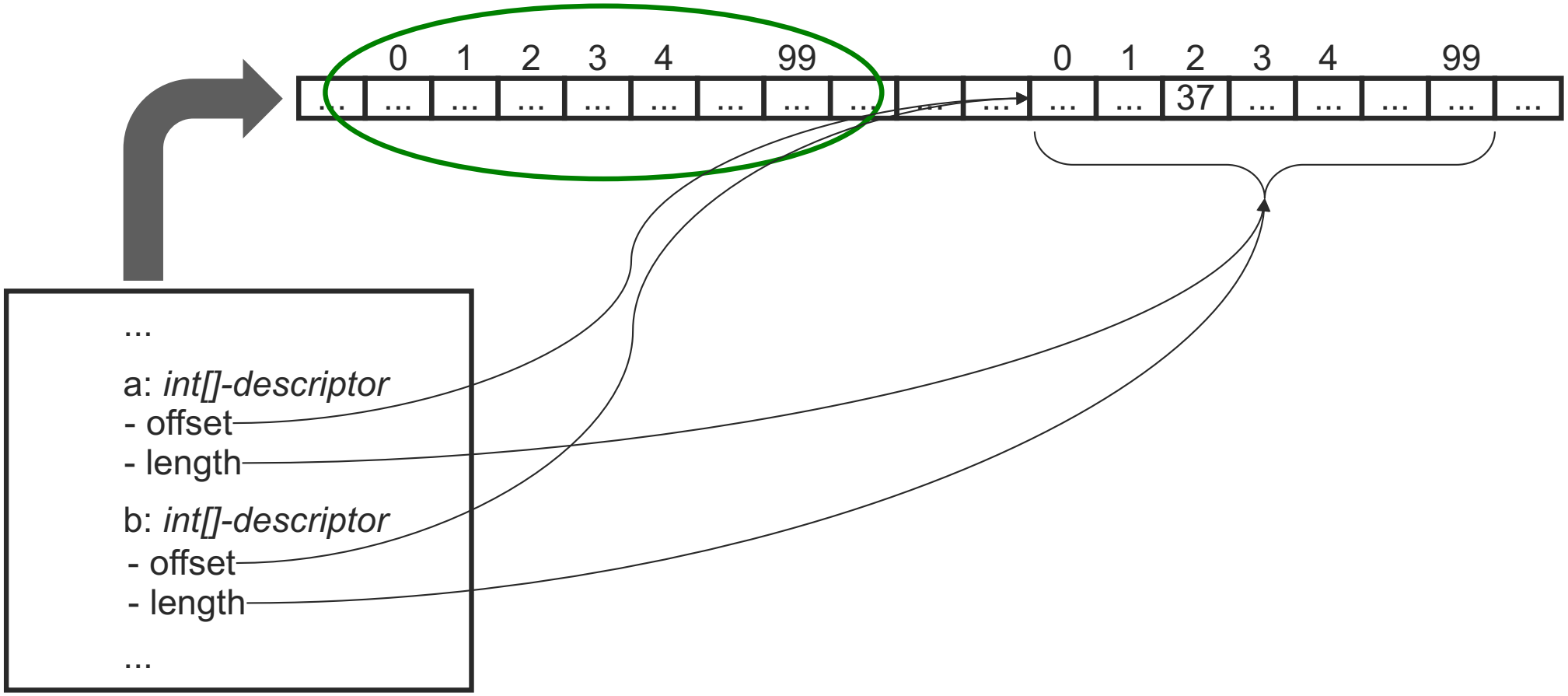
Die Zuweisung

```
a = b;
```

sorgt für die Zuweisung des Deskriptors

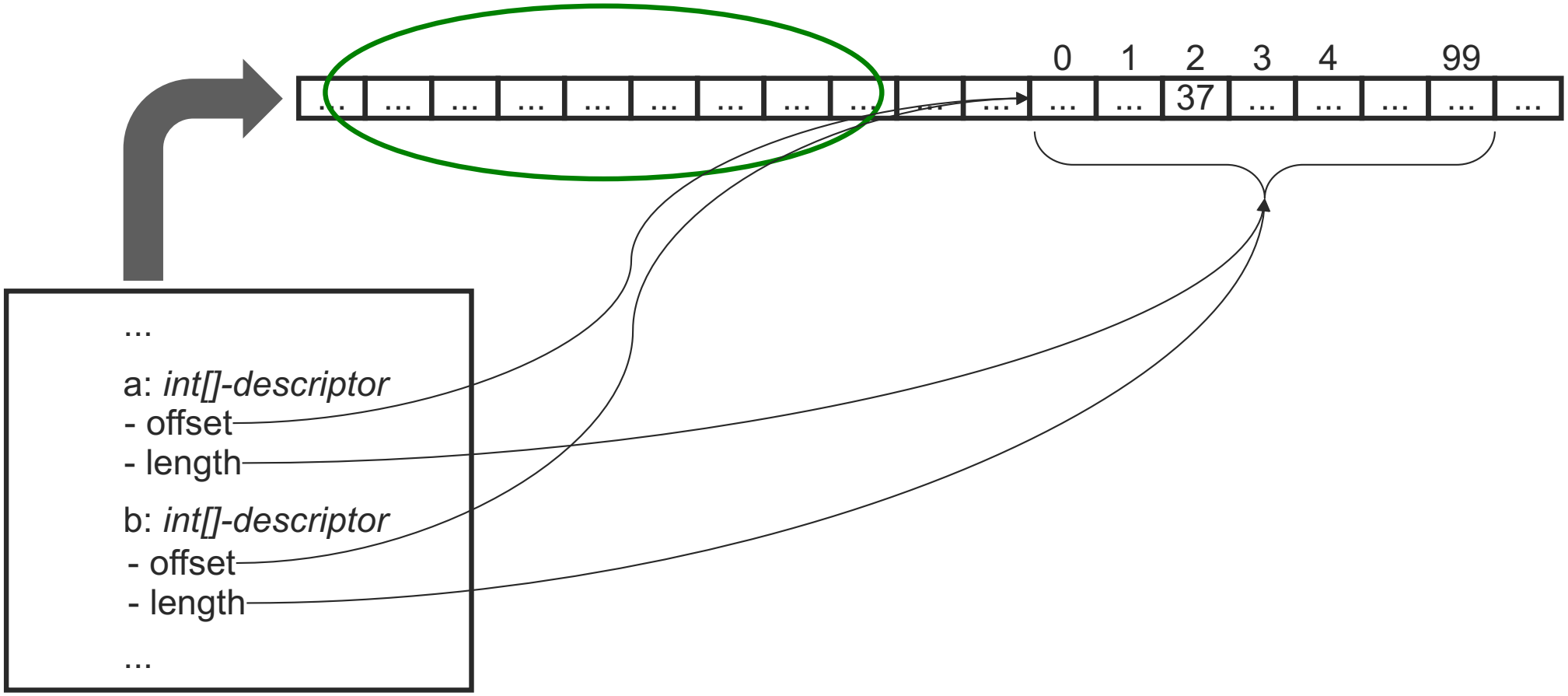
Freigeben von Speicher

Speicherbereiche, die nicht länger benötigt werden, werden *automatisch* wieder freigegeben



Freigeben von Speicher

Speicherbereiche, die nicht länger benötigt werden, werden *automatisch* wieder freigegeben

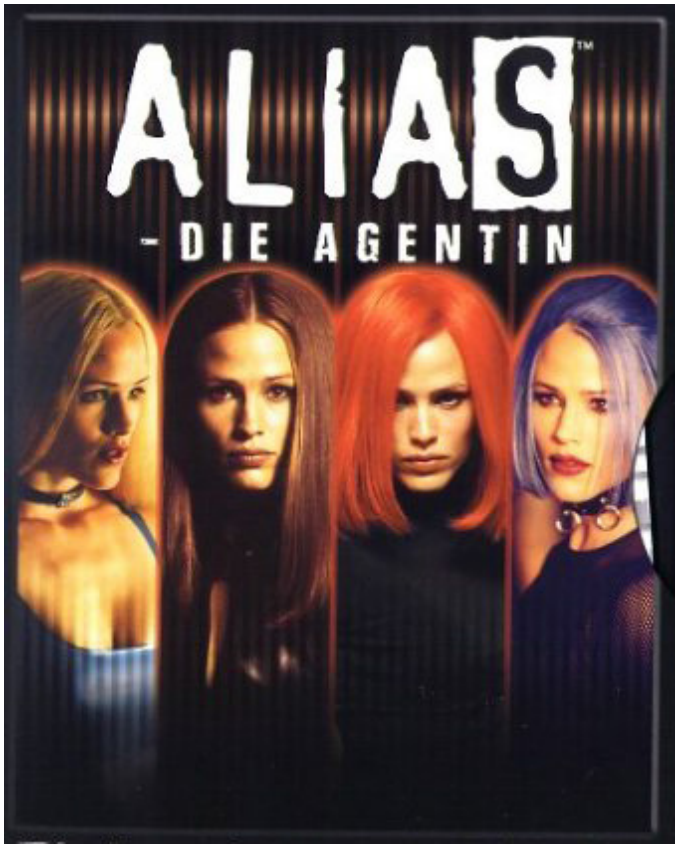


Zusammenfassung: Zuweisung von (ganzen) Arrays

- Bei der Zuweisung *ganzer Arrays* werden nicht die Array-Inhalte, sondern nur die Deskriptoren zugewiesen

  Im Gegensatz dazu gilt in Java *für alle elementaren Datentypen*, dass die *Werte* der jeweiligen Variablen zugewiesen werden
  - byte, short, int, long, float, double, char, boolean (nicht: String!)
  - Hält man sich an die Konvention, sieht man das den Typnamen an: (nur) die Namen aller elementaren Datentypen beginnen mit einem Kleinbuchstaben

- Referenzieren zwei (oder mehr) (Array-)Variablen den gleichen Speicherbereich, spricht man von ***Aliasierung***



Aliasierung

- Als Folge einer Arrayzuweisung zeigen beide Arrays auf den selben Speicherbereich;
  Änderungen in einem vom mehreren aliasierten Arrays wirken sich unmittelbar und identisch auf das andere aus
- Beispiel

```
int [] a = new int[100], b = new int[100];


...
a[12] = 1;
b[12] = 0;

a = b;
println( a[12] );
                                → ?

a[12] = 2;
println( b[12] );
                                → ?
```

Arrays ungleicher Länge

- Als Nebeneffekt ist es möglich, zwei Arrays ungleicher Länge einander zuzuweisen
- Das Array auf der linken Seite der Zuweisung übernimmt u.a. die Länge des Arrays auf der rechten Seite der Zuweisung
- Beispiel

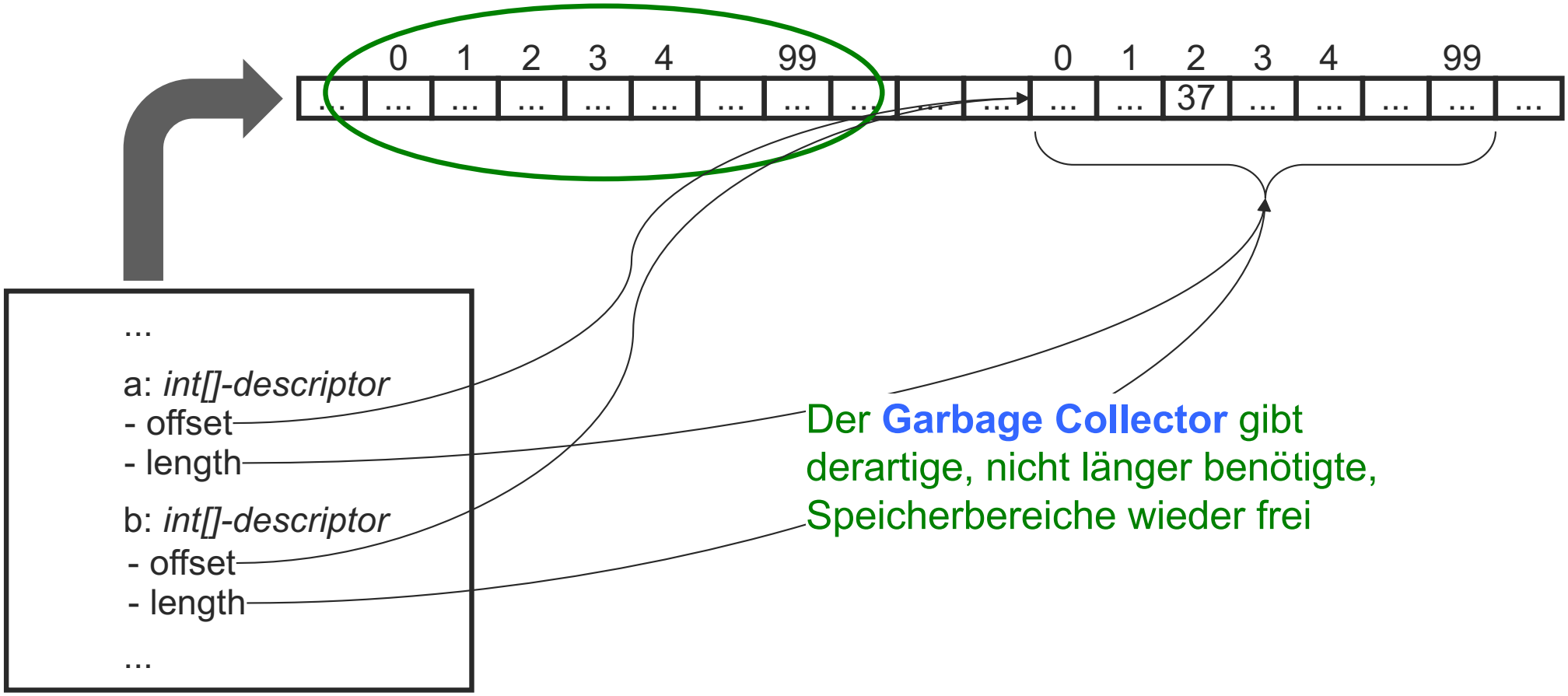
```
int [] a = new int[100], b = new int[10];
println( a.length );
                                        → ?
a = b;
println( a.length );
                                        → ?
```

Was passiert mit frei gewordenen Speicherbereichen?

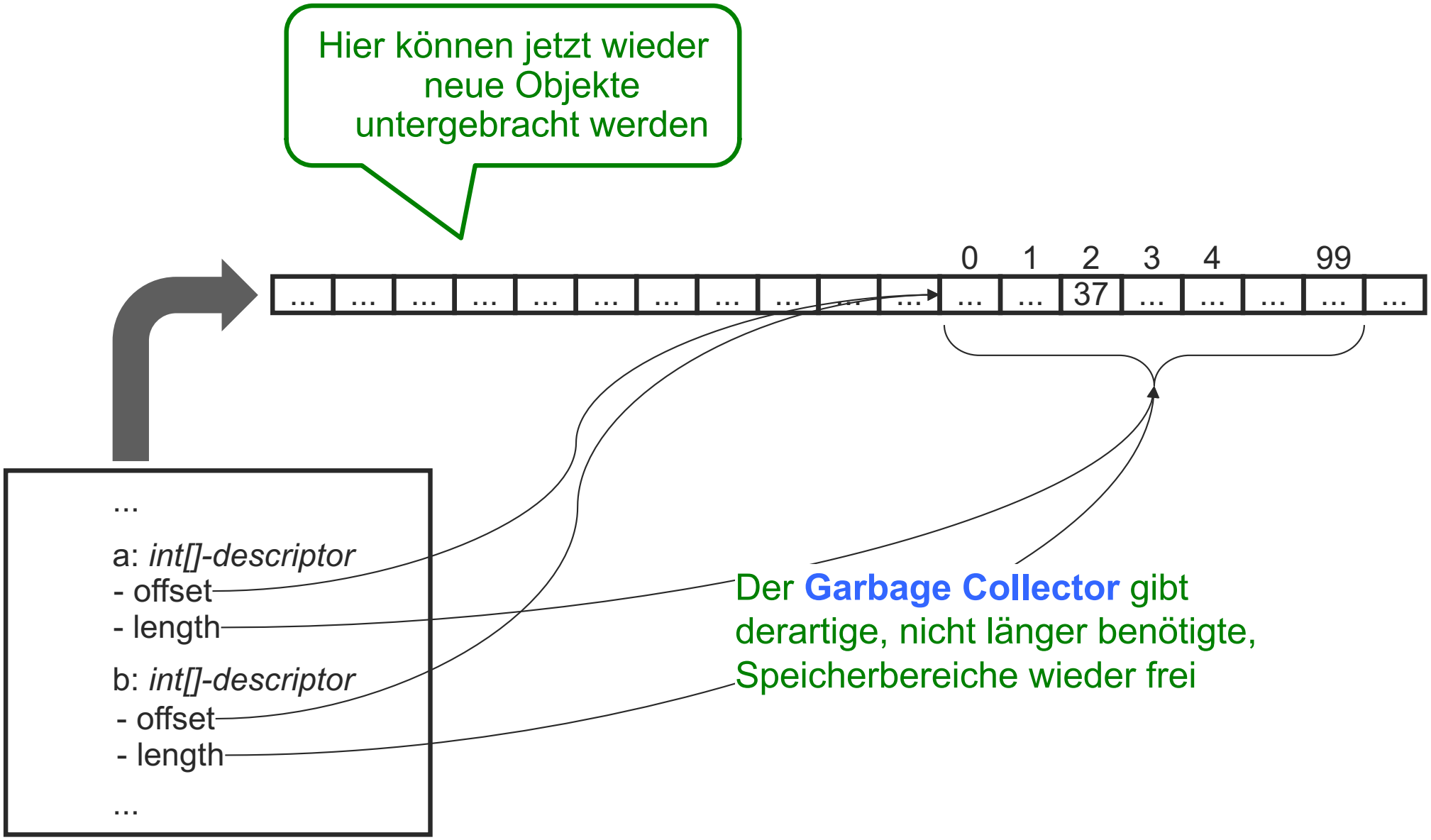
- Wir haben gesehen, was bei der Zuweisung von Objekten im Speicher passiert:
  Der Offset-Zeiger im Deskriptor eines Objekts zeigt nach einer Zuweisung an eine andere Stelle
- Es stellt sich die Frage, ob / wie der zuvor belegte und nun nicht länger benötigte Speicherbereiche freigegeben werden kann, damit er durch andere Objekte erneut belegt werden kann
- Der Sachverhalt wird auf der folgenden Folie illustriert

Illustration

Was passiert mit dem Speicherbereich, den die Variable *a* nun nicht mehr „braucht“?



Illustration



Was darf freigegeben werden?

- Welche Speicherbereiche dürfen freigegeben werden, welche nicht?

- Wie kann man das feststellen?

- In anderen Sprachen gibt es die Möglichkeit / die Notwendigkeit, diese Aufgabe manuell durchzuführen
  Das ist zum Beispiel bei Programmen in C++ notwendig

Vor- und Nachteile

Vorteile der automatischen Speicherverwaltung

Die nicht-manuelle Beseitigung ist… :

Nachteil der automatischen Speicherverwaltung

Die manuelle Beseitigung ist… :