Universität Bremen

Praktische Informatik 2
Grundlegende Datenstrukturen

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für
Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Abstrakte Datentypen (In Java: Klassen)

- Ein abstrakter Datentyp besteht aus den Daten selbst (z.B. Objektzustand) und den auf den Daten auszuführenden Operationen (z.B. Methoden)
- Interne Abläufe werden versteckt (Geheimnisprinzip)
- Beispiele
  - **plus : int × int → int**
  - **null : → int**
  - **neg : int → int**

plus(a, b) = plus(b, a)
plus(a, null) = a
plus(a, plus(b, c)) = plus(plus(a, b), c)
plus(a, neg(a)) = null

- Semantik der Operationen kann formal beschrieben werden, z.B. mit Axiomen

Universität Bremen

Stapel (Stack)

- Dient zum Zwischenspeichern von Elementen
- Der Zugriff erfolgt nach dem **last-in**, **first-out** Prinzip (**lifo**), d.h das **zuletzt** abgelegte Element wird **zuerst** zurückgeliefert
- Stapel erzeugen: **new : → Stack<E>**
- Eintrag ablegen: **push : Stack<E> × E → Stack<E>**
- Kopf auslesen: **top : Stack<E> → E**
- Eintrag entnehmen (und zurückliefern): **pop : Stack<E> → Stack<E>** (**× E**)
- Auf Leere testen: **empty : Stack<E> → boolean**

top(push(s, e)) = e
pop(push(s, e)) = s
empty(new)
¬ empty(push(s, e))

Beschränkter Stapel: Beispiel

Stack<Integer>										
size	3									
values	17	5	8	0	0	0	0	0	0	0

top() → 8

Stack<Integer>										
size	4									
values	17	5	8	**6**	0	0	0	0	0	0

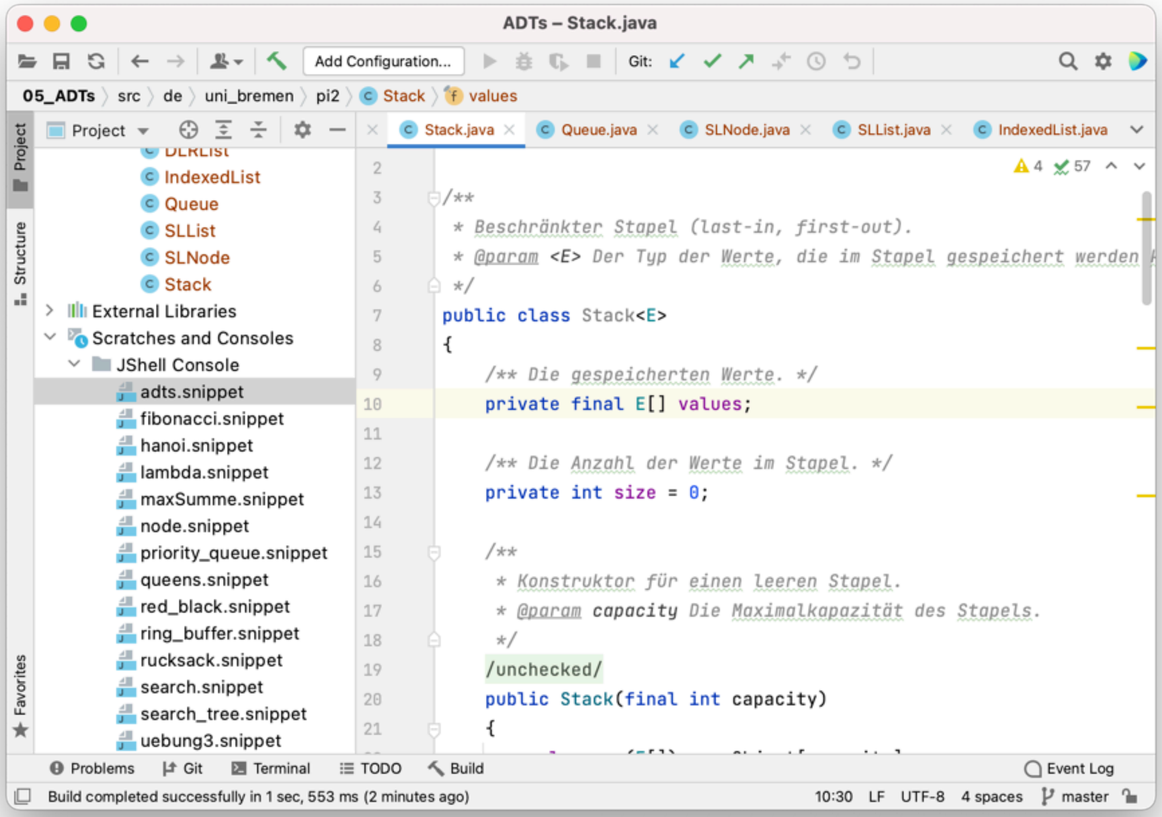
push(6)

Stack<Integer>										
size	3									
values	17	5	8	**6**	0	0	0	0	0	0

pop() → 6

Universität Bremen

Beschränkter Stapel: Demo



Universität Bremen

Warteschlange (Queue)

- Dient zum Zwischenspeichern von Daten
- Zugriff erfolgt nach dem **first-in**, **first-out** Prinzip (**fifo**), d.h. das zuerst abgelegte Element wird zuerst zurückgeliefert
- Warteschlange erzeugen: **new : → Queue<E>**
- Eintrag ablegen: **push : Queue<E> × E → Queue<E>**
- Kopf auslesen: **top : Queue<E> → E**
- Eintrag entnehmen (und zurückliefern): **pop : Queue<E> → Queue<E> (× E)**
- Auf Leere testen: **empty : Queue<E> → boolean**

top(push(new, e)) = e
pop(push(new, e)) = new
empty(new)
¬ empty(push(q, e))
top(push(q, e)) = top(q), q ≠ new
pop(push(q, e)) =
push(pop(q), e), q ≠ new

Beschränkte Warteschlange: Beispiel

Queue<Integer>										
first	2									
size	8									
values	17	5	8	0	7	4	42	19	11	0

top() → 8

Queue<Integer>										
first	2									
size	9									
values	**6**	5	8	0	7	4	42	19	11	0

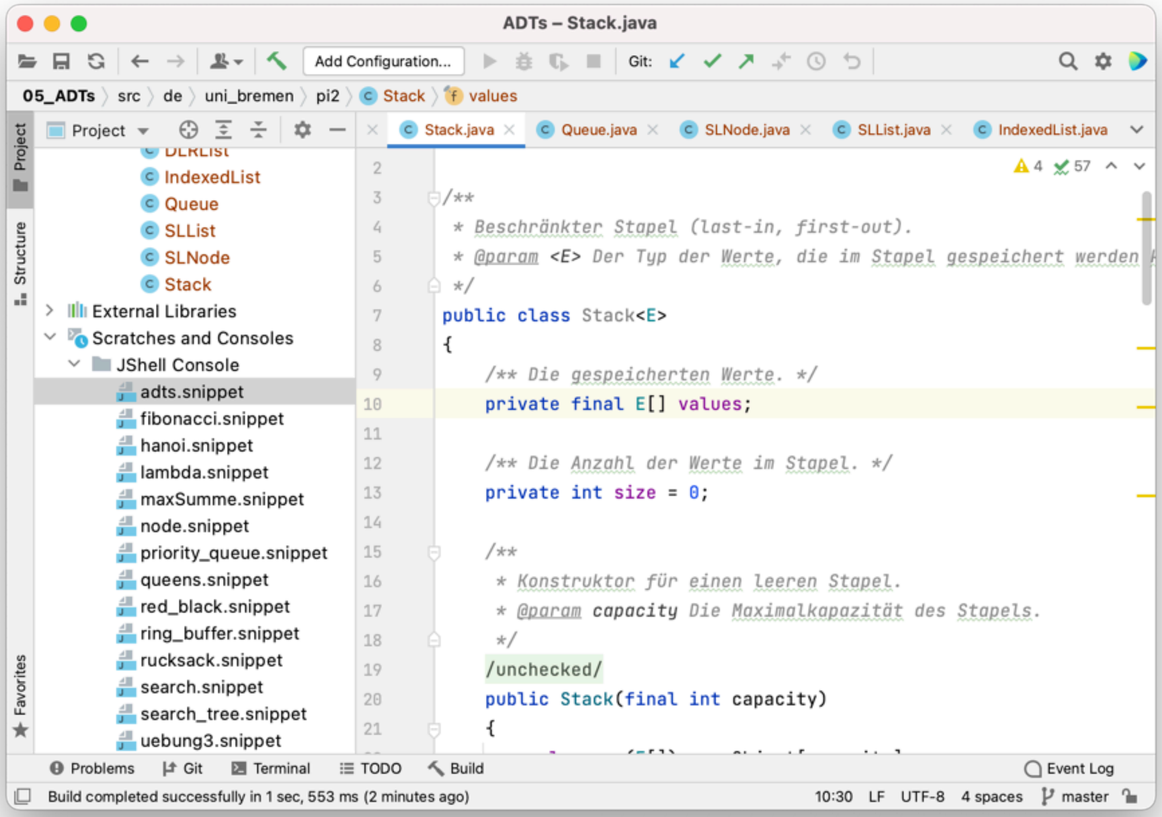
push(6)

Queue<Integer>										
first	3									
size	8									
values	6	5	**8**	0	7	4	42	19	11	0

pop() → 8

Universität Bremen

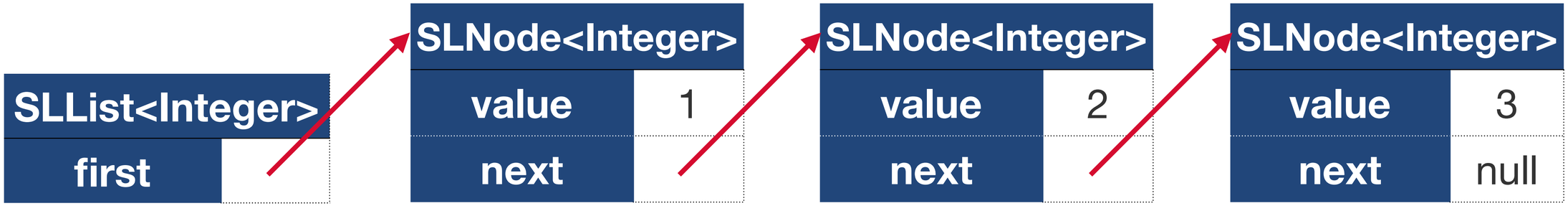
Beschränkte Warteschlange: Demo



Universität Bremen

Einfach verkettete Liste

- Jedes Element enthält einen Zeiger auf seinen Nachfolger
- Kann nur in einer Richtung durchlaufen werden
- Einfügen immer am Anfang der Liste oder hinter einem bekannten Listenelement
- Ein Element kann nur gelöscht werden, wenn sein Vorgänger bekannt ist
- Einfügen und Löschen sind in konstanter Zeit möglich
- Speicherplatzverbrauch höher als bei Arrays

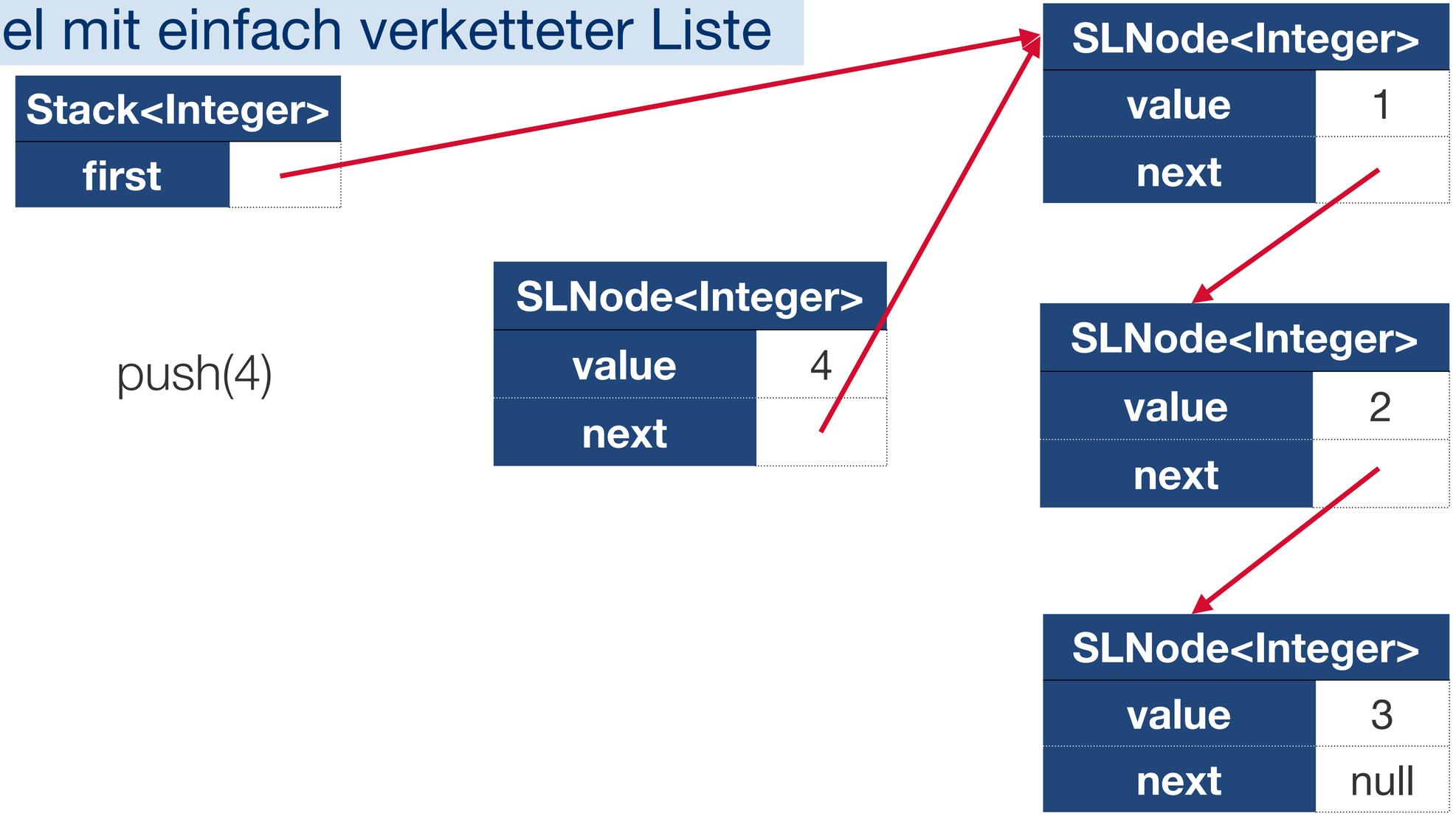


Stapel mit einfach verketteter Liste

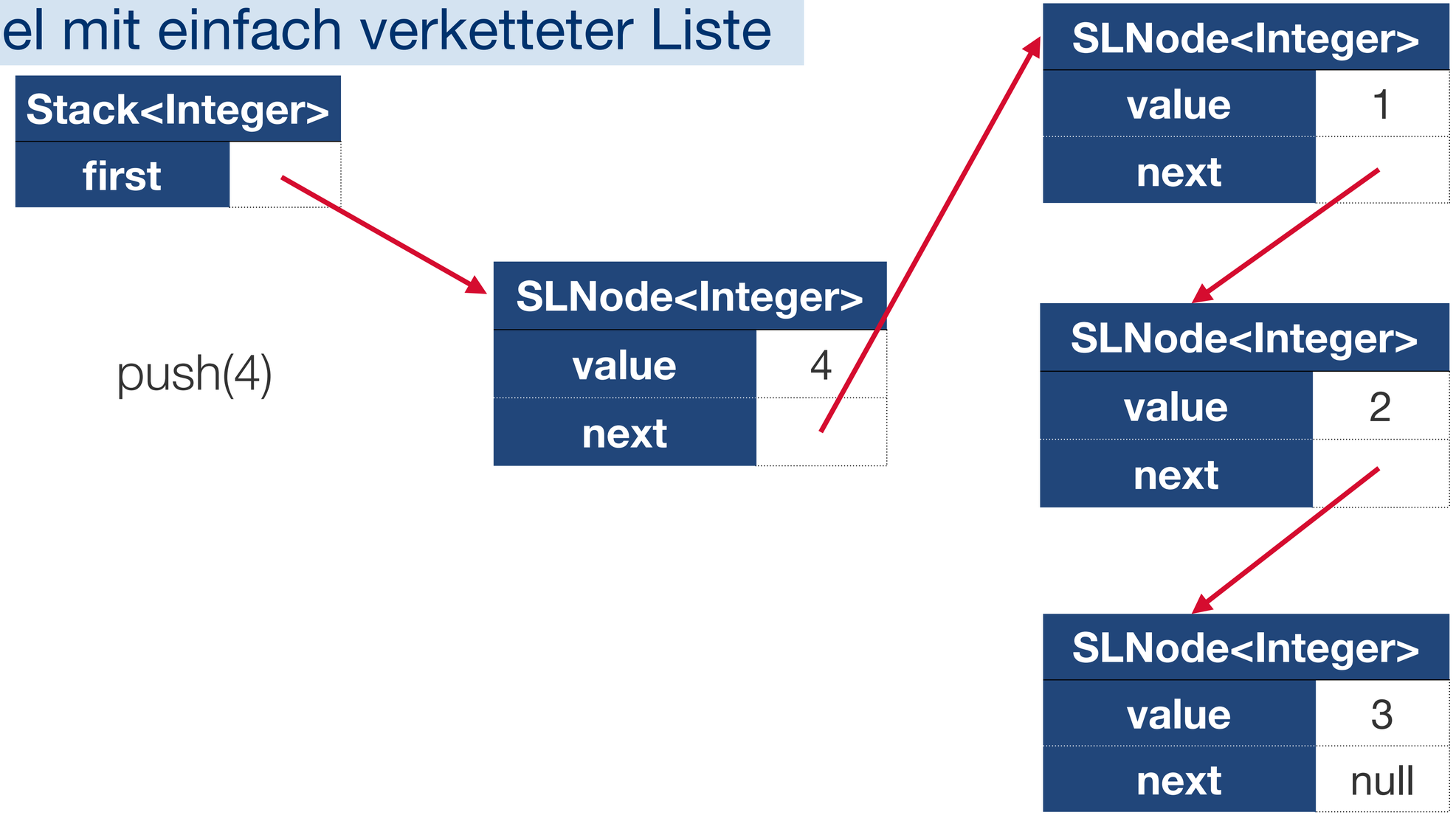
push(4)

Stapel mit einfach verketteter Liste

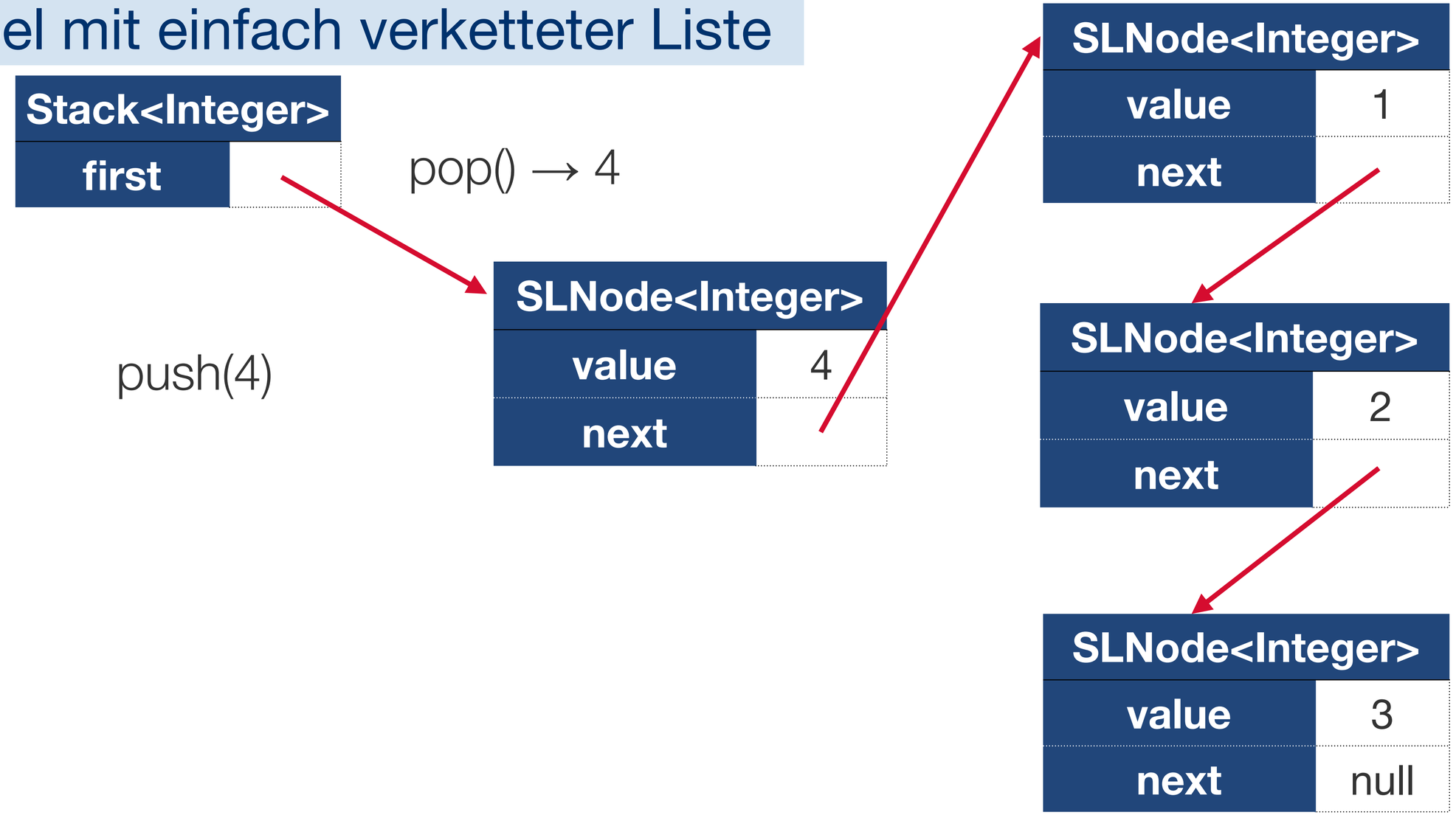
Stapel mit einfach verketteter Liste



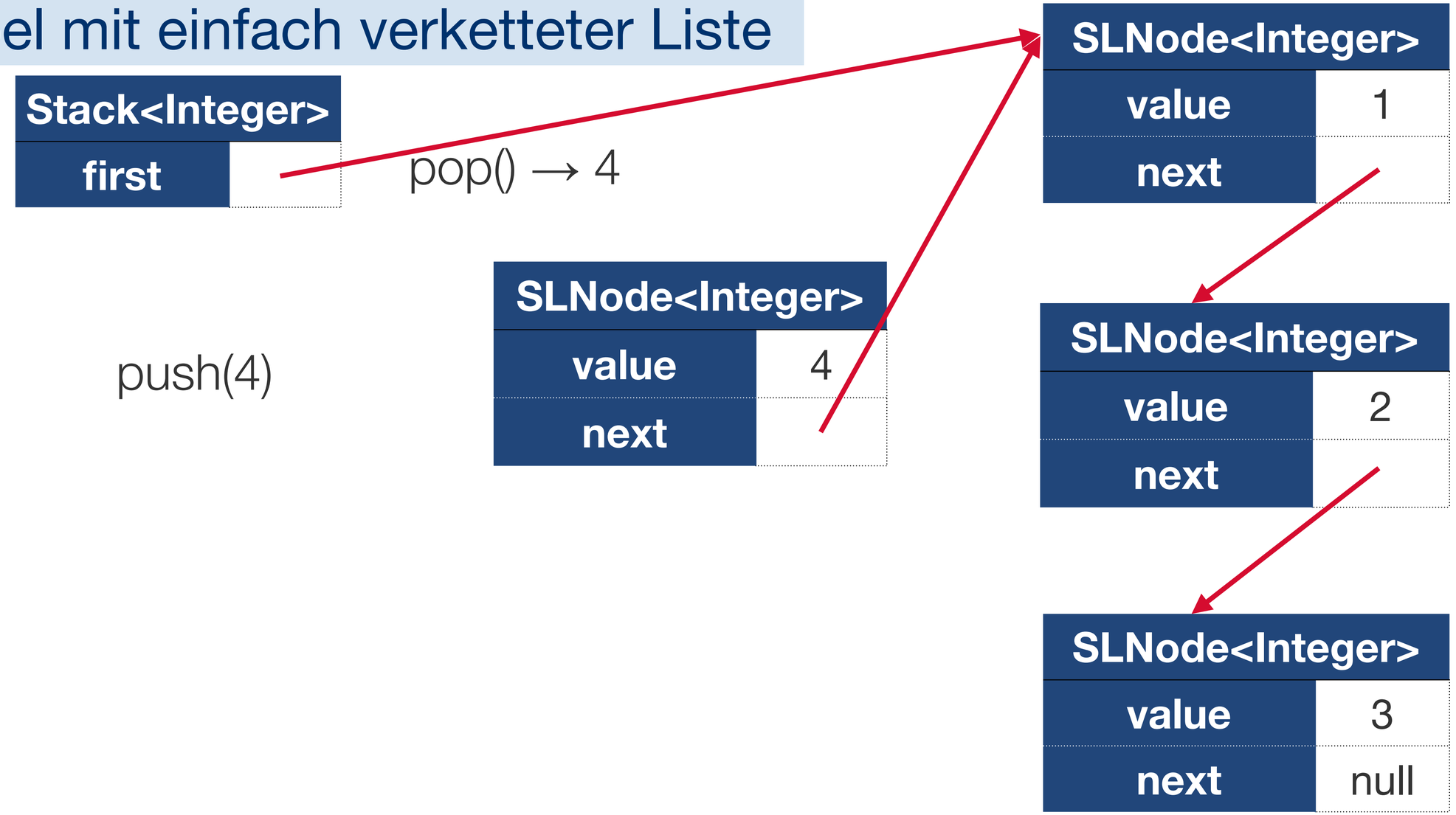
Stapel mit einfach verketteter Liste



Stapel mit einfach verketteter Liste



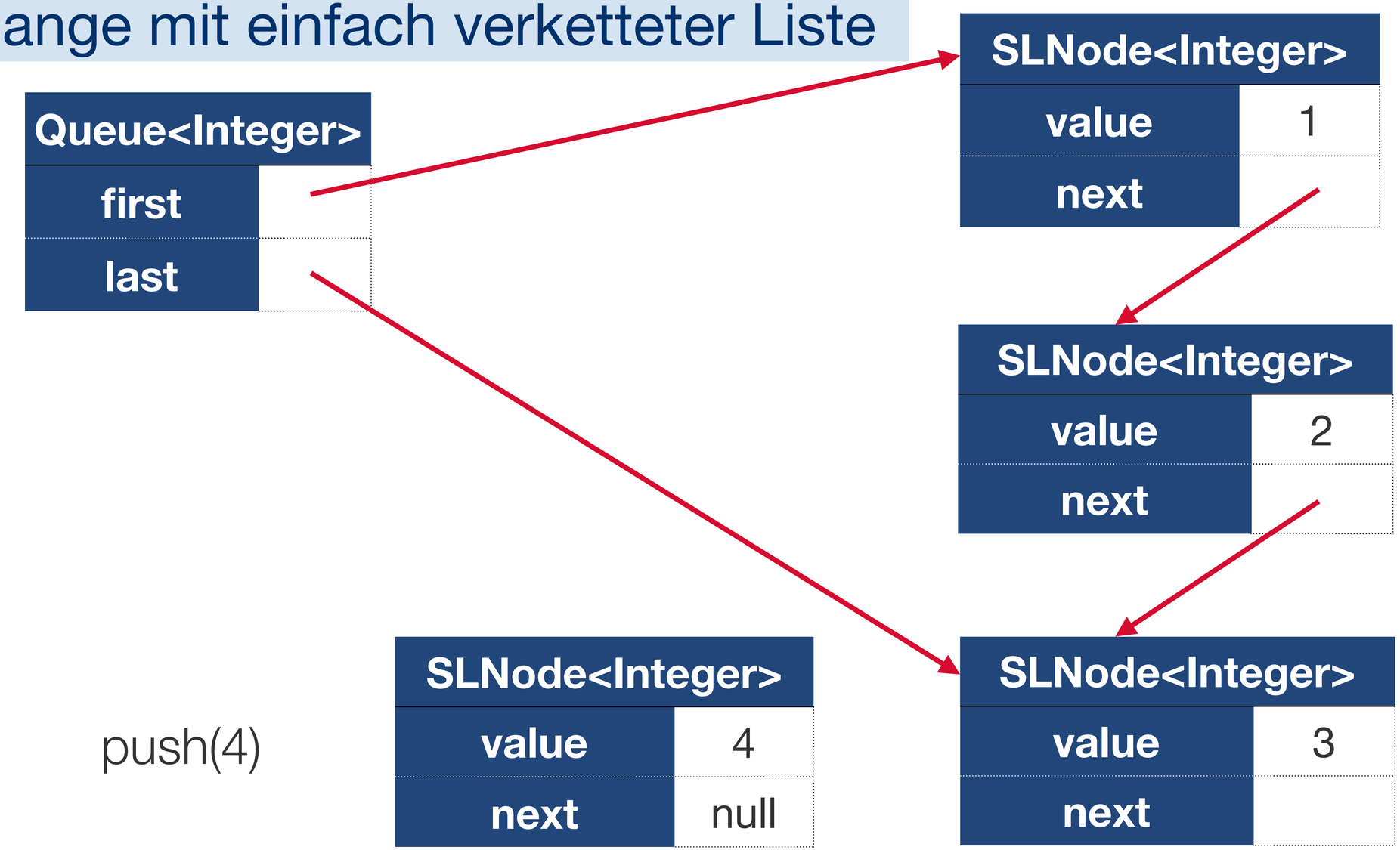
Stapel mit einfach verketteter Liste



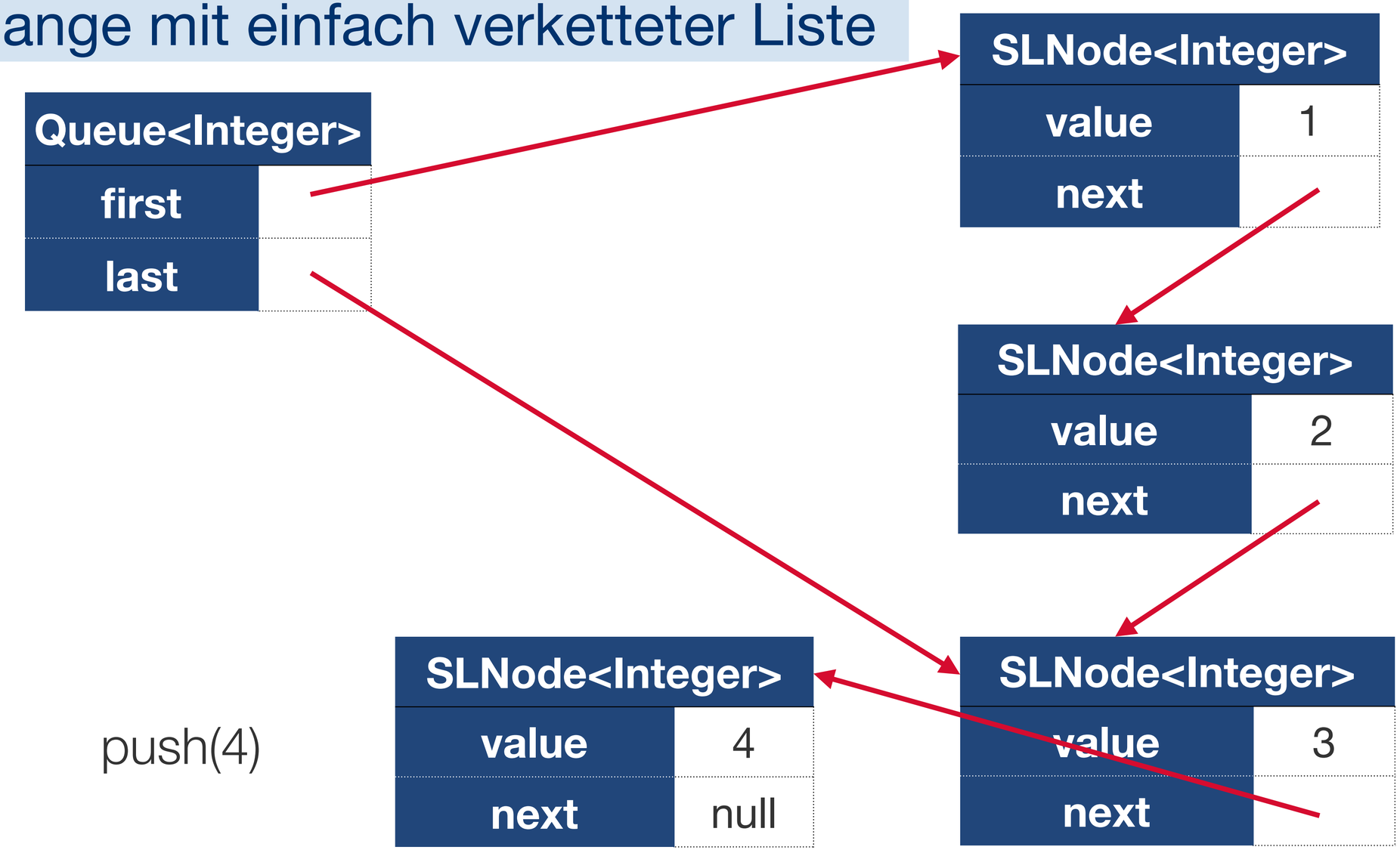
Warteschlange mit einfach verketteter Liste

push(4)

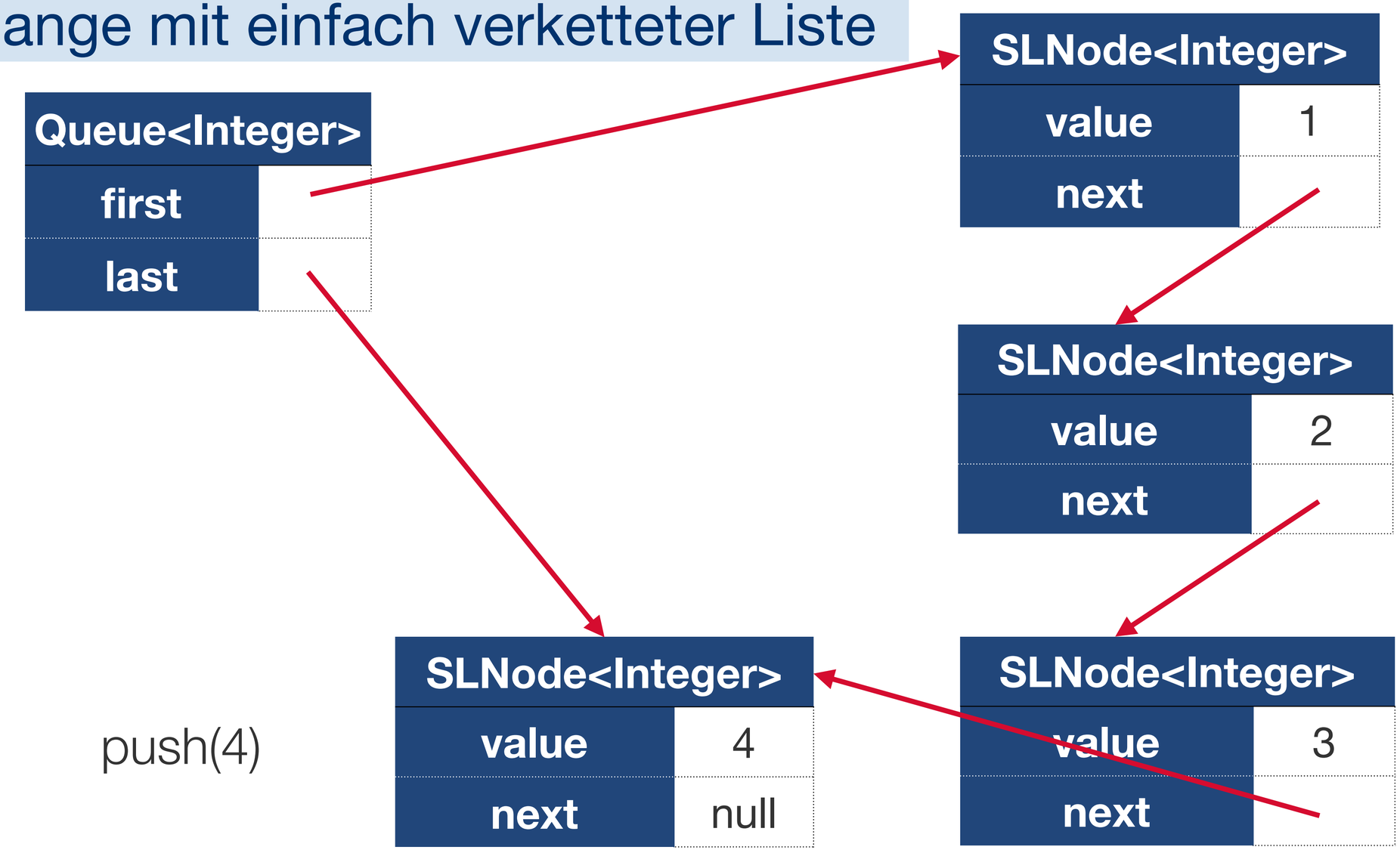
Warteschlange mit einfach verketteter Liste



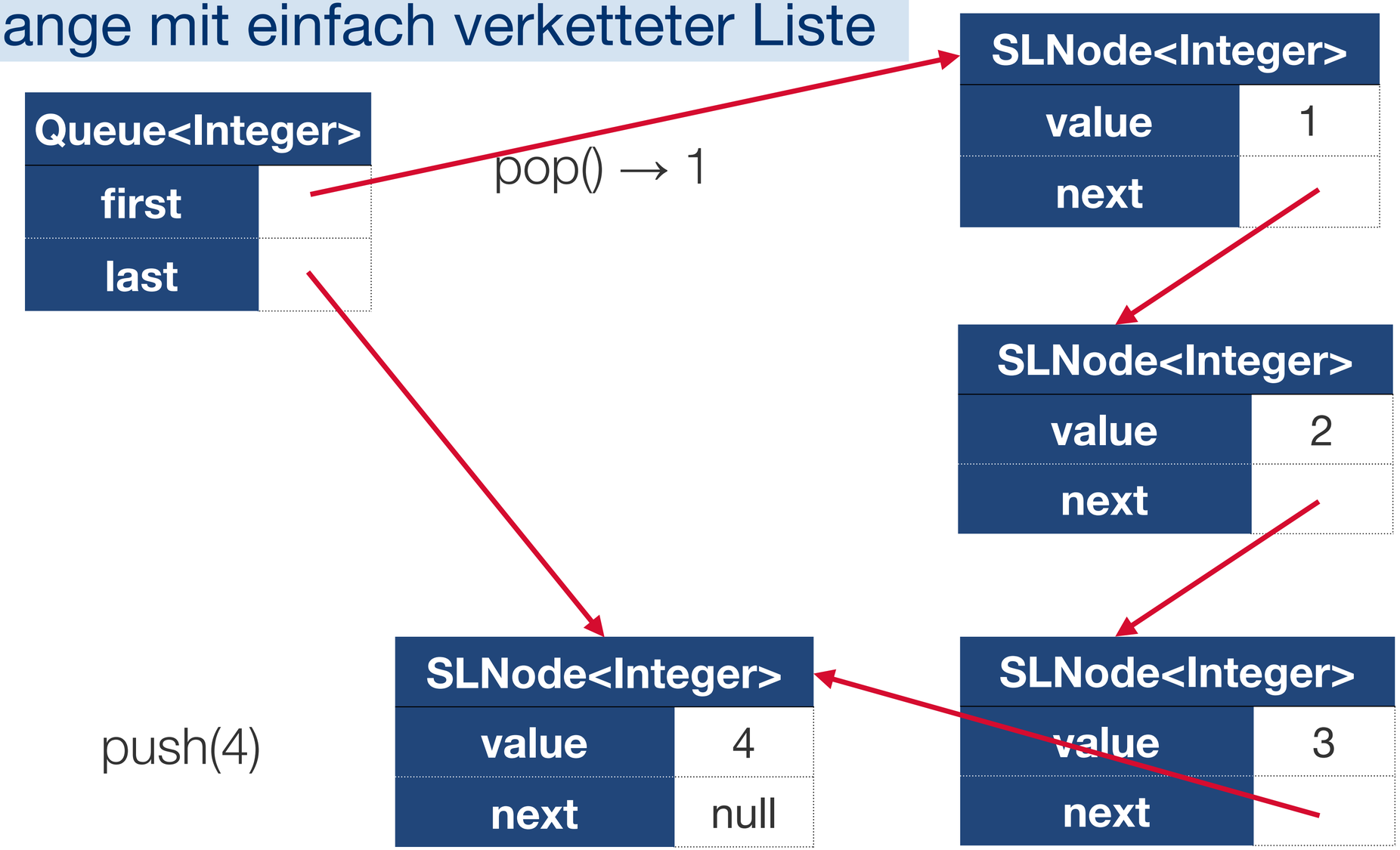
Warteschlange mit einfach verketteter Liste



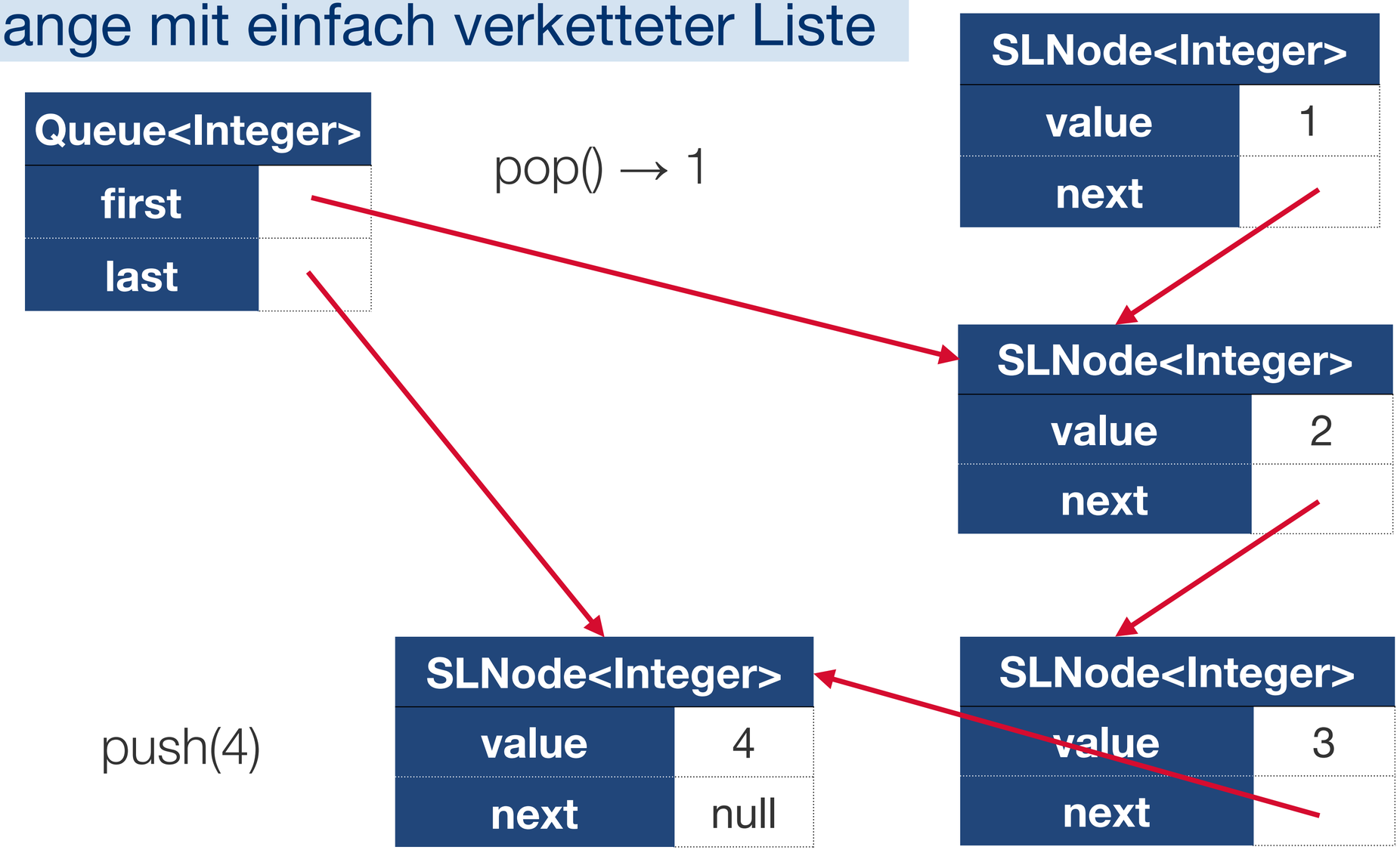
Warteschlange mit einfach verketteter Liste



Warteschlange mit einfach verketteter Liste



Warteschlange mit einfach verketteter Liste



Operationen auf Listen

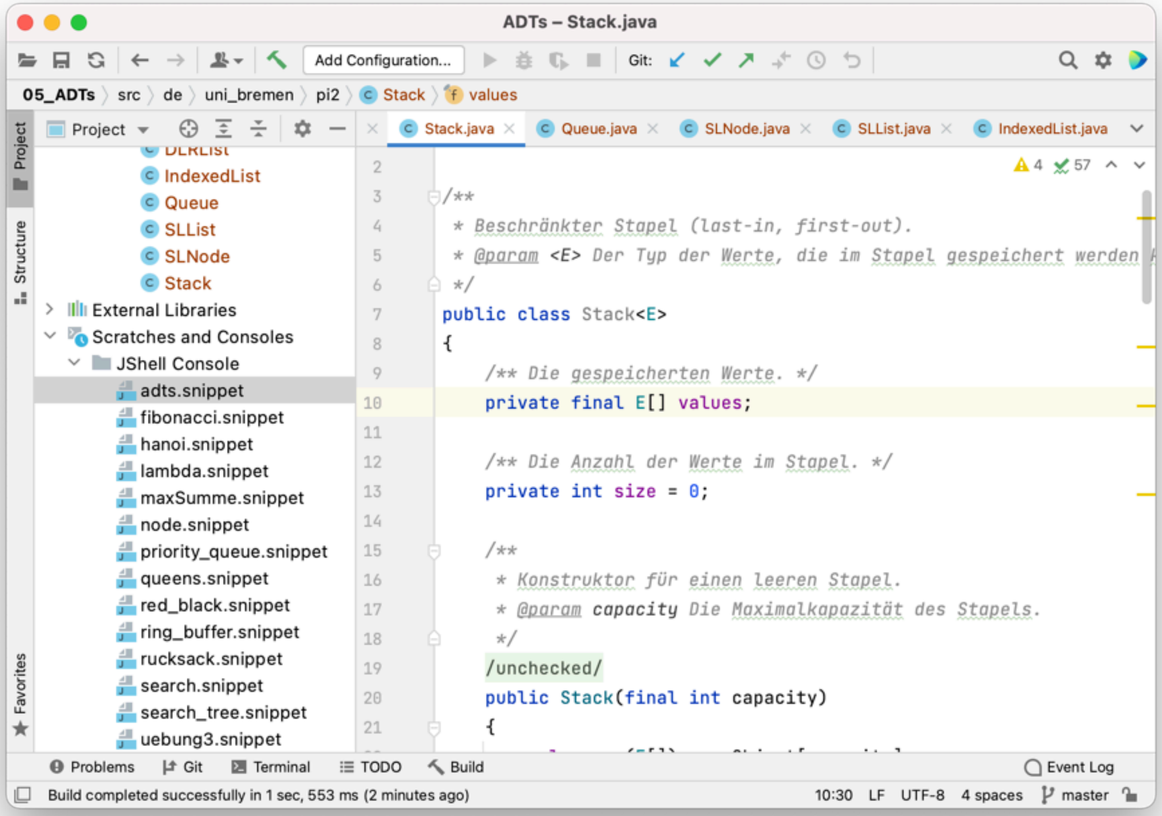
- Liste erzeugen: **new : → SLList<E>**
- Ersten Knoten liefern: **first : SLList<E> → SLNode<E>**
- Eintrag einfügen: **insert : SLList<E> × E × SLNode<E> → SLList<E>**
- Knoten löschen: **remove : SLList<E> × SLNode<E> → SLList<E>**
- Auf Leere testen: **empty : SLList<E> → boolean**
- Knoten erzeugen: **new : E × SLNode<E> → SLNode<E>**
- Nächsten Knoten liefern: **next : SLNode<E> → SLNode<E>**
- Wert liefern: **value : SLNode<E> → E**

SLList<E>

SLNode<E>

Universität Bremen

Einfach verkettete Liste: Demo



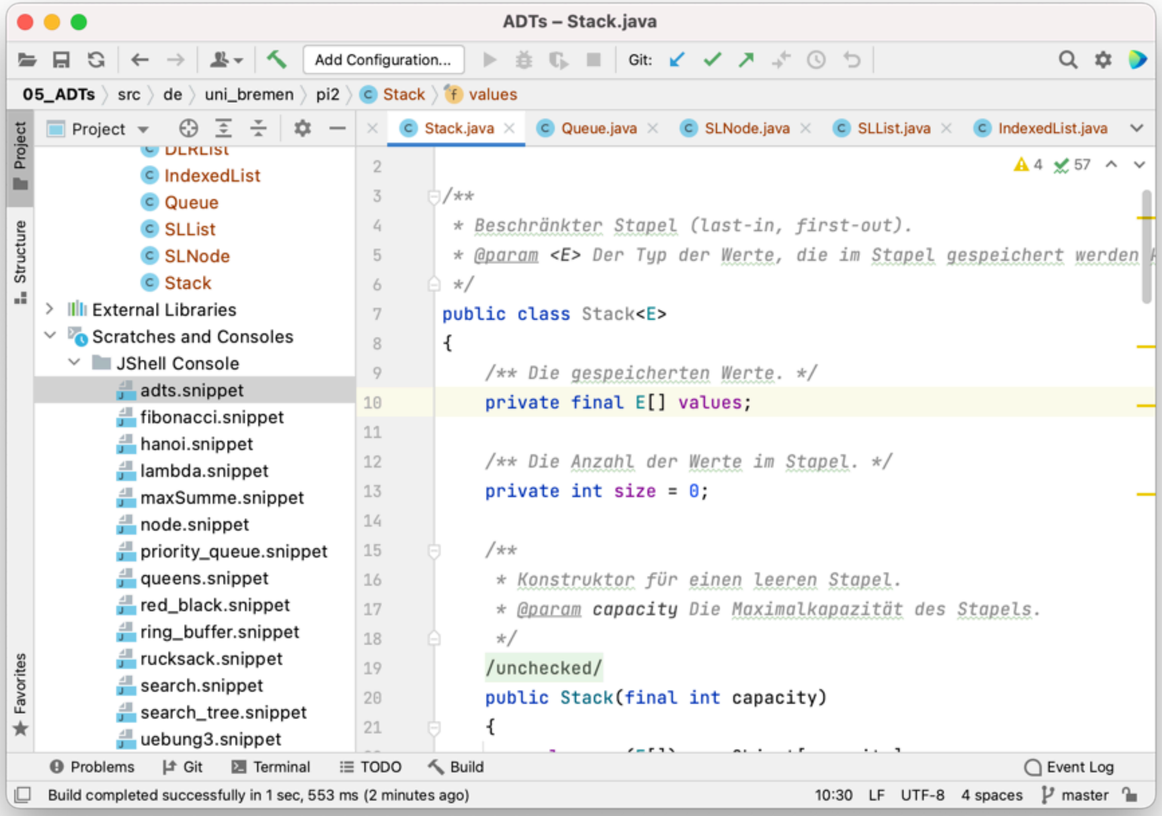
Universität Bremen

Einfach verkettete Liste mit Index

- Bildet ein Array dynamischer Größe mithilfe einer Liste nach, wodurch Einfügen und Löschen ohne Umkopieren möglich sind
- Liste erzeugen: **new : → IndexedList<E>**
- Eintrag schreiben: **set : IndexedList<E> × int × E → IndexedList<E>**
- Eintrag auslesen: **get : IndexedList<E> × int → E**
- Einfügen: **insert : IndexedList<E> × int × E → IndexedList<E>**
- Löschen: **remove : IndexedList<E> × int → IndexedList<E>**
- Größe: **size : IndexedList<E> → int**

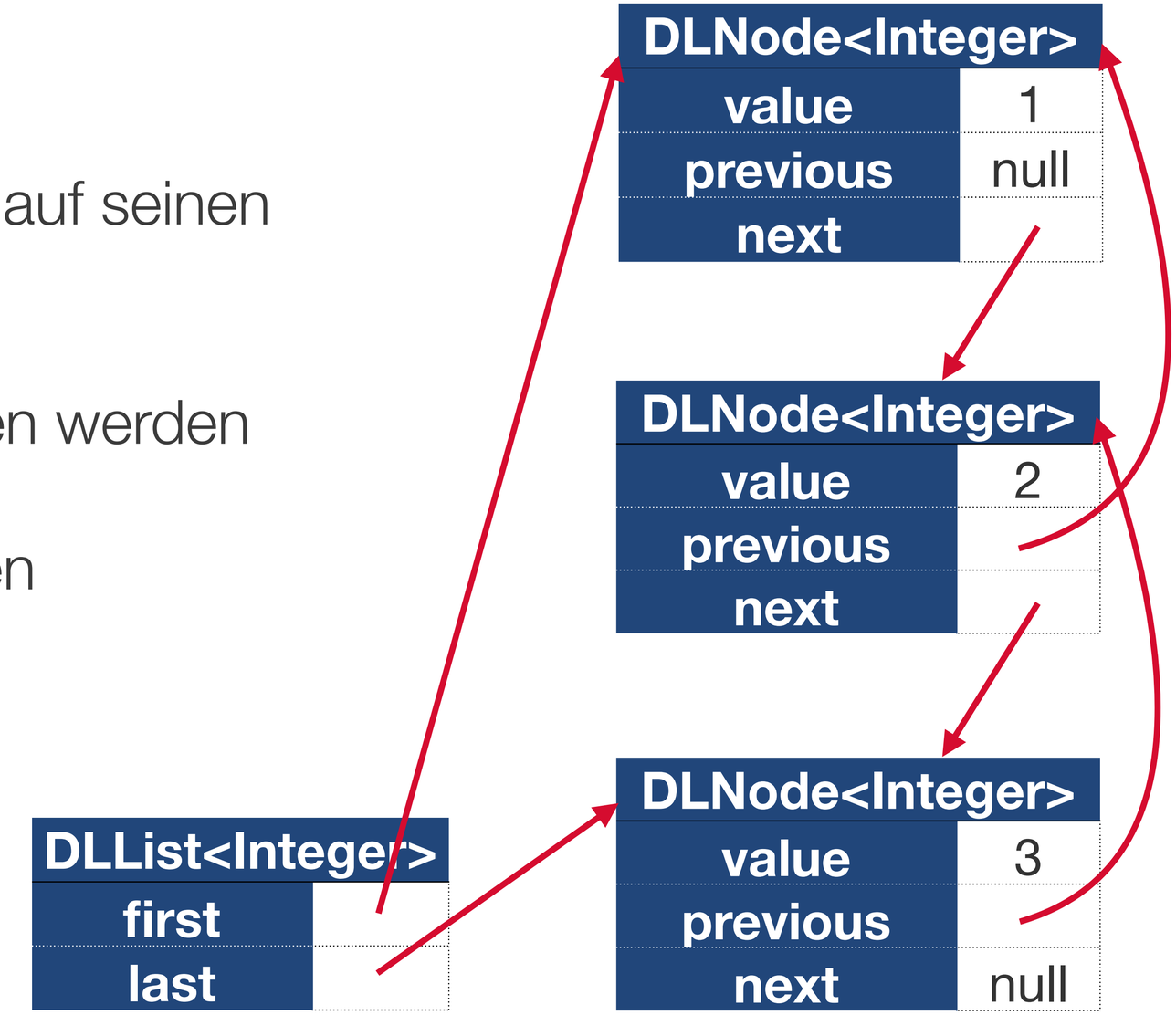
Universität Bremen

Einfach verkettete Liste mit Index: Demo

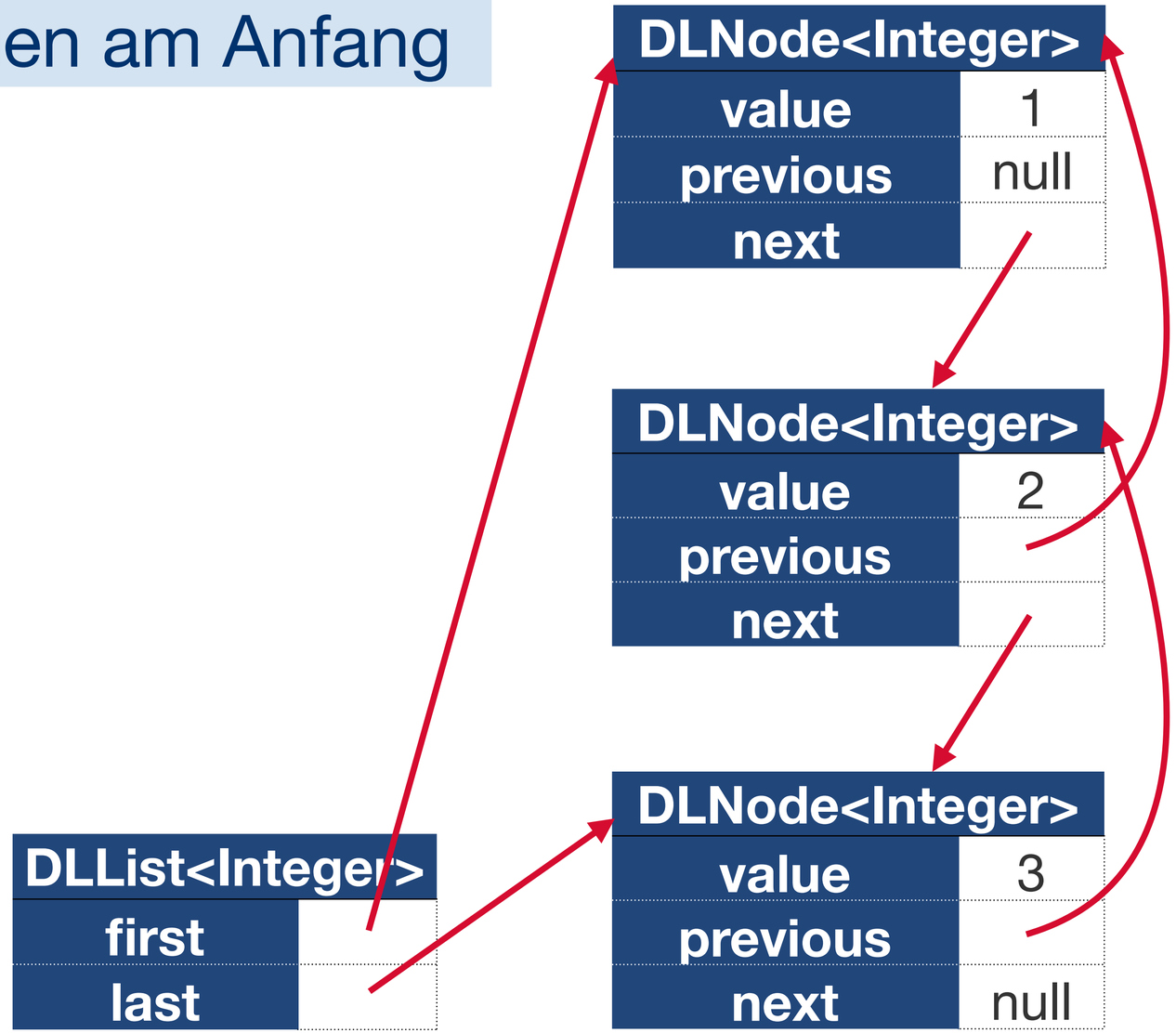


Doppelt verkettete Liste

- Jedes Element enthält einen Zeiger auf seinen Vorgänger und seinen Nachfolger
- Kann vor- und rückwärts durchlaufen werden
- Einfügen vor und hinter existierenden Listenelementen
- Löschen von Elementen problemlos möglich

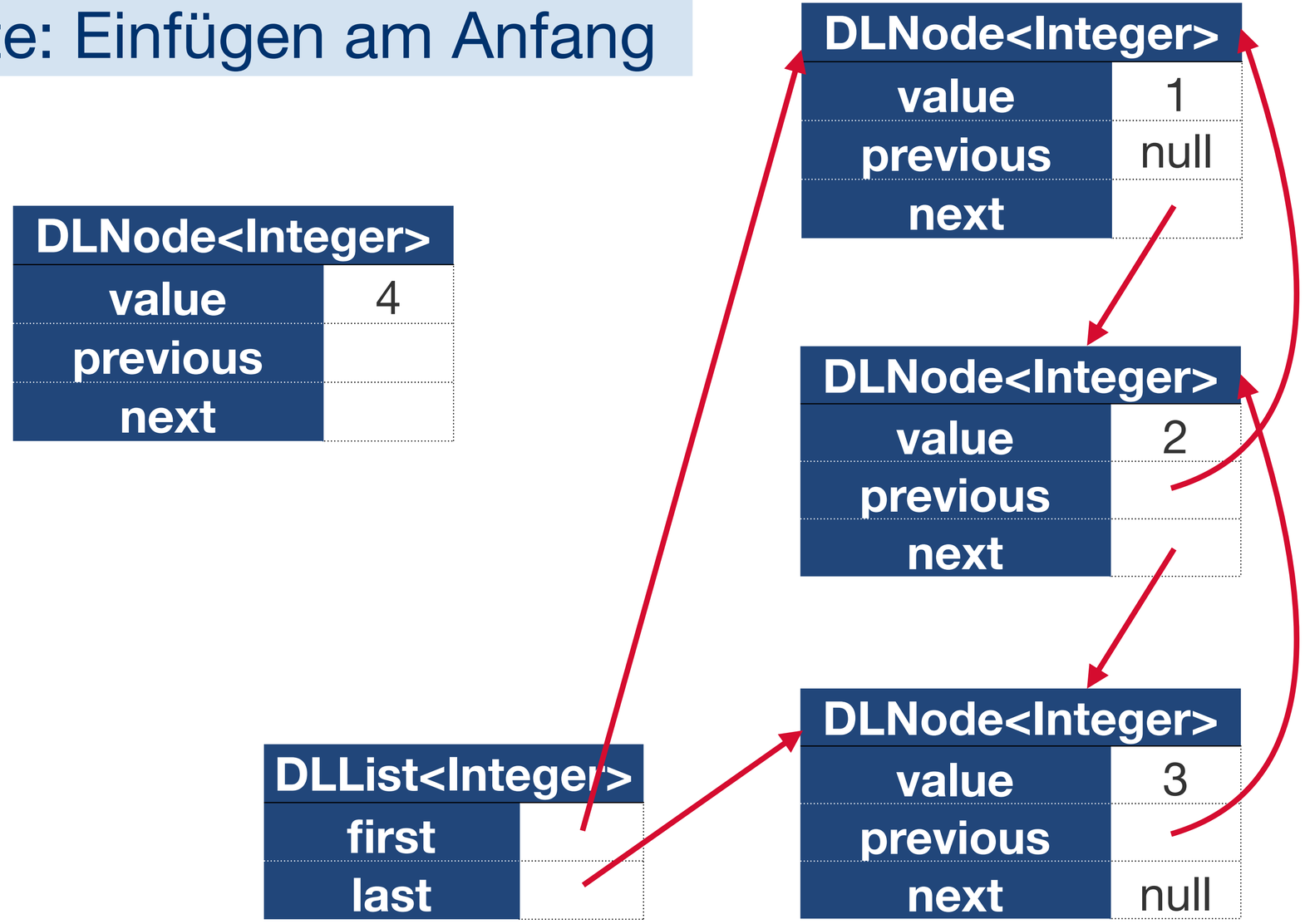


Doppelt verkettete Liste: Einfügen am Anfang



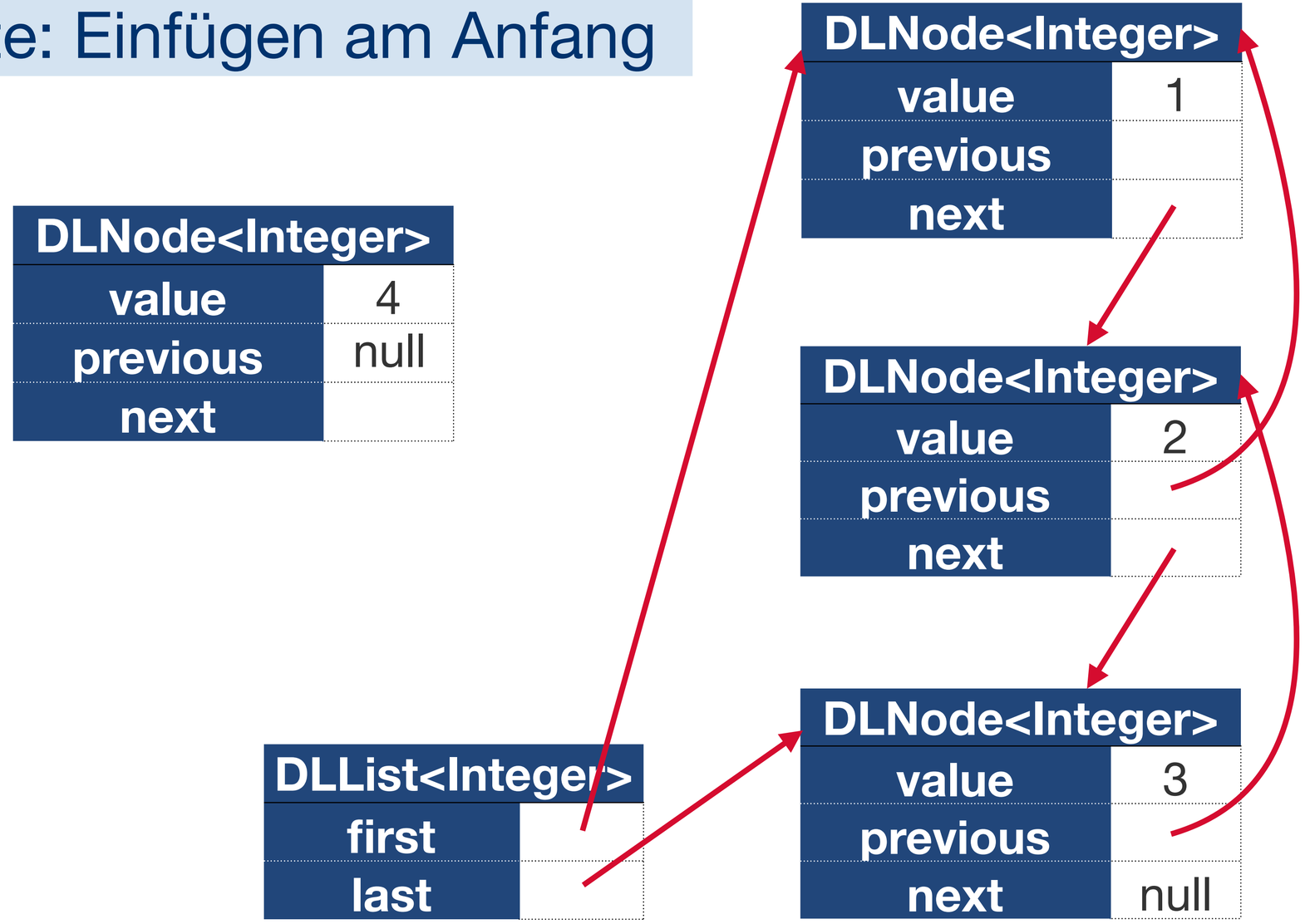
Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)



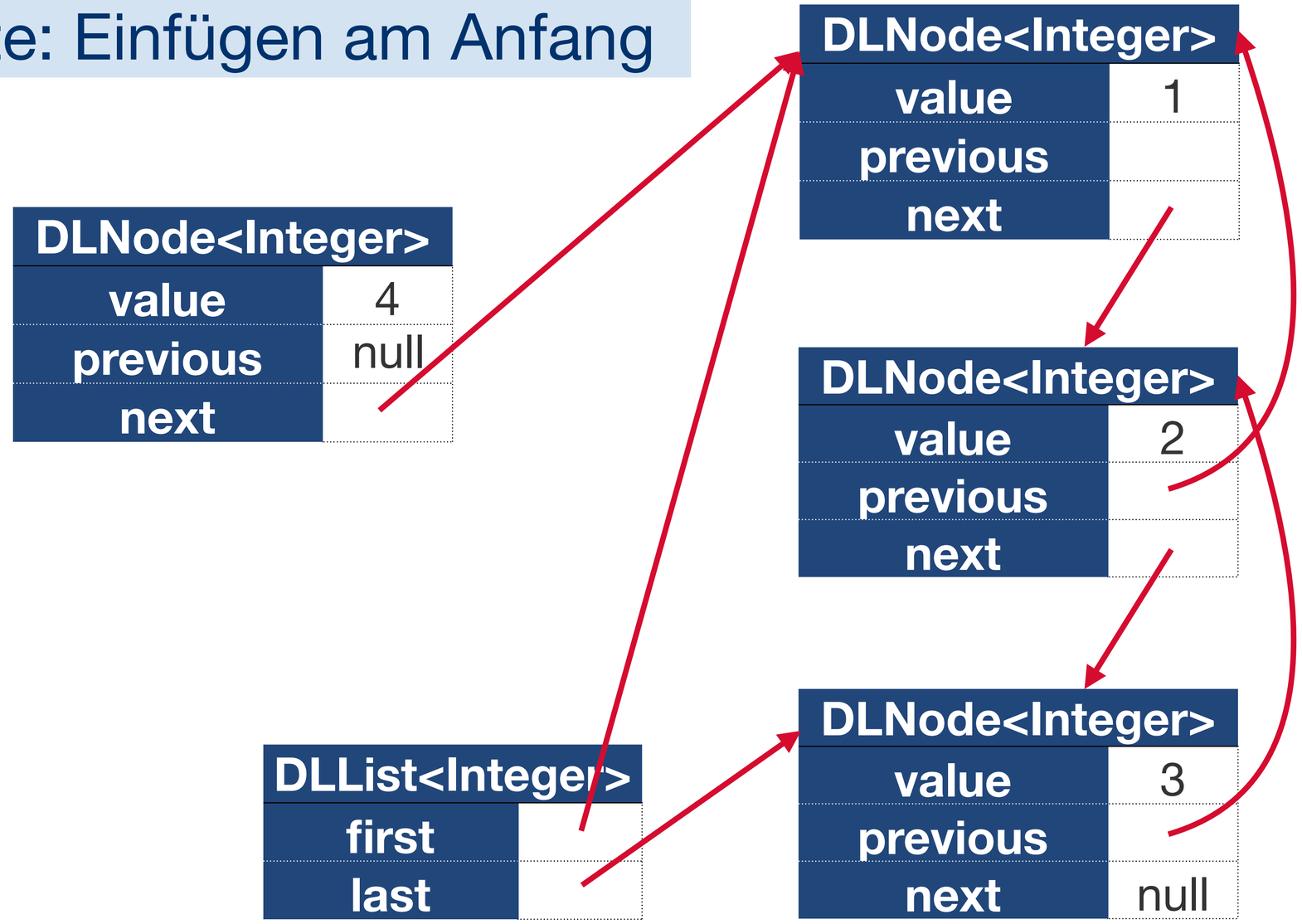
Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)



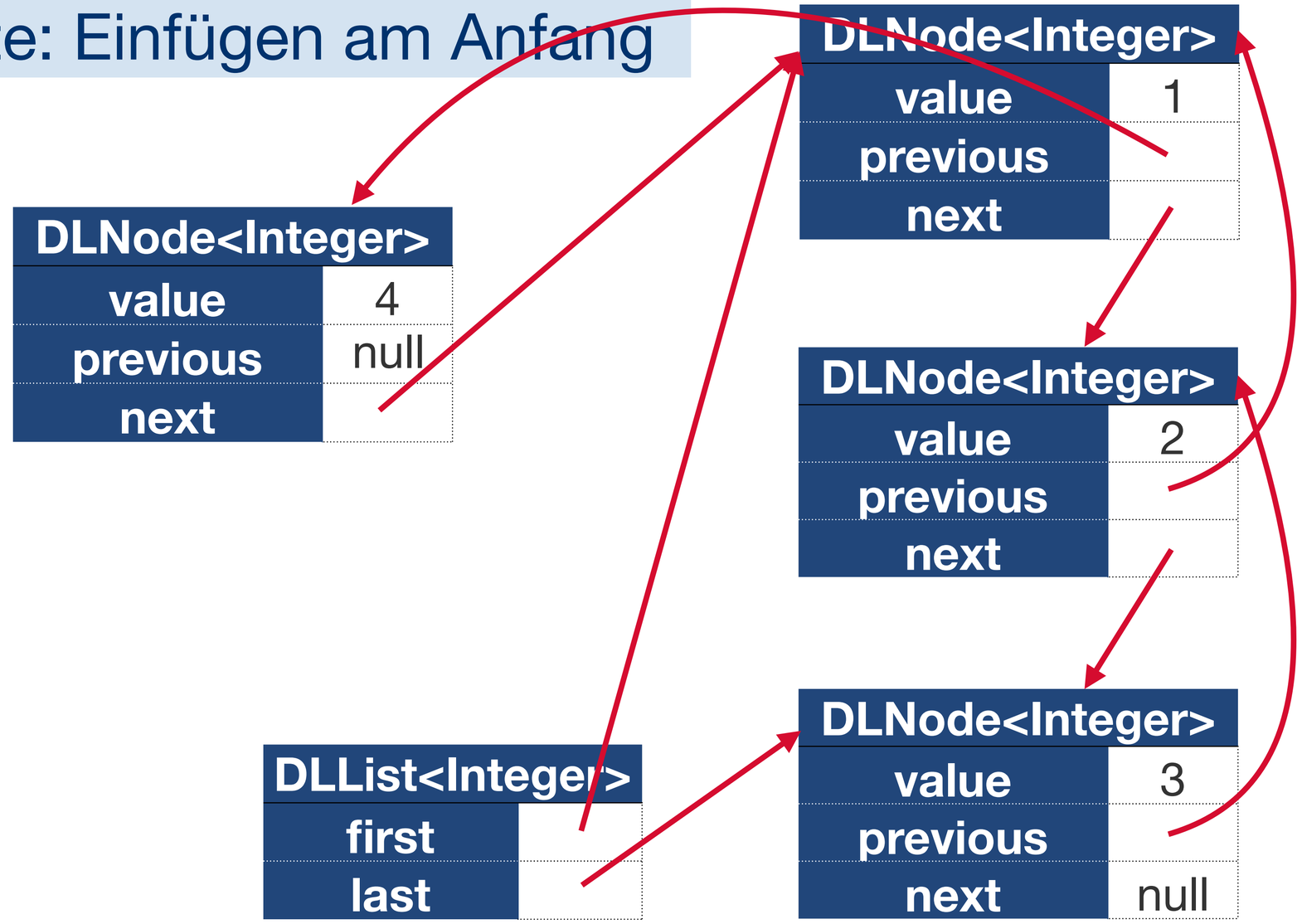
Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)



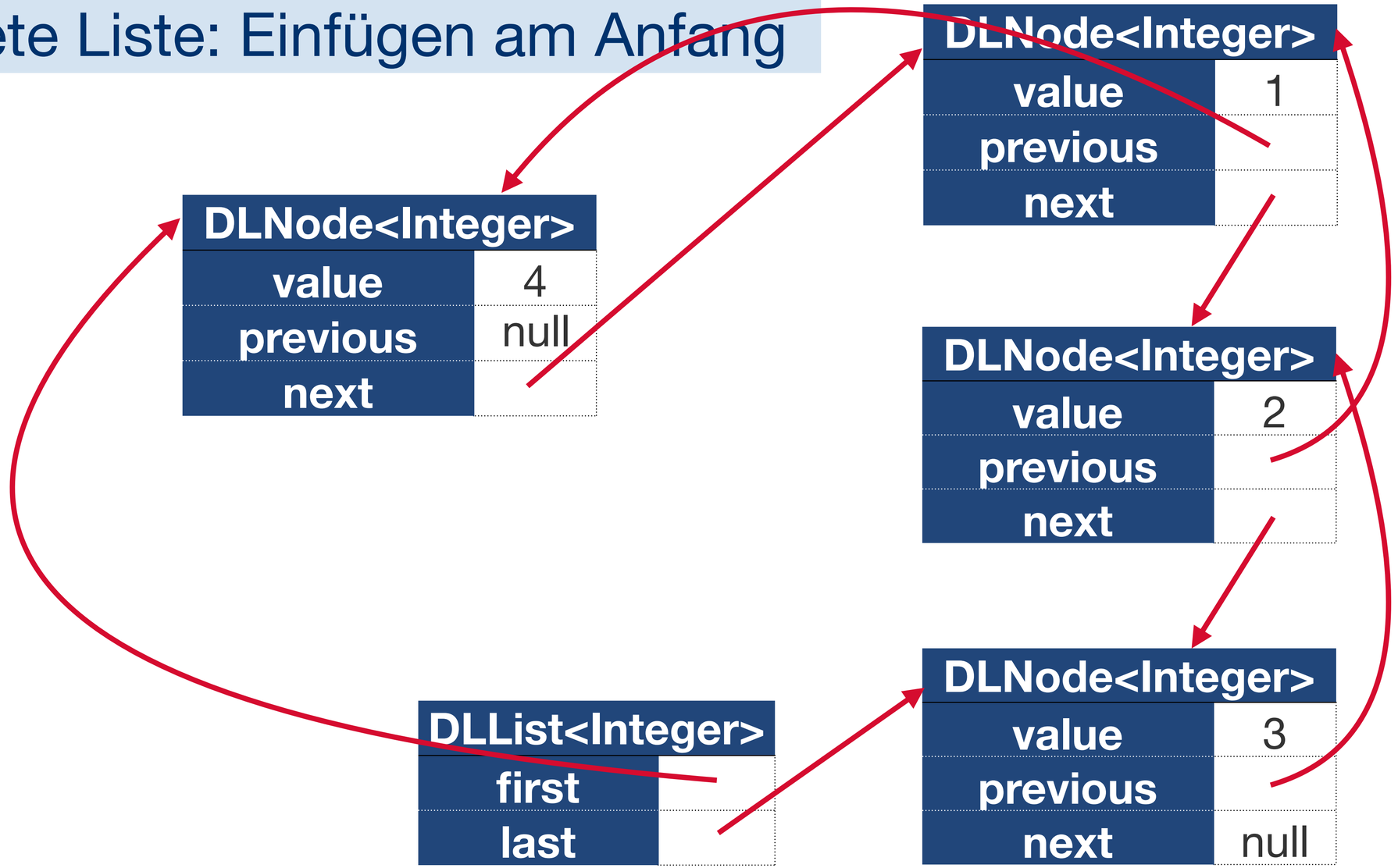
Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

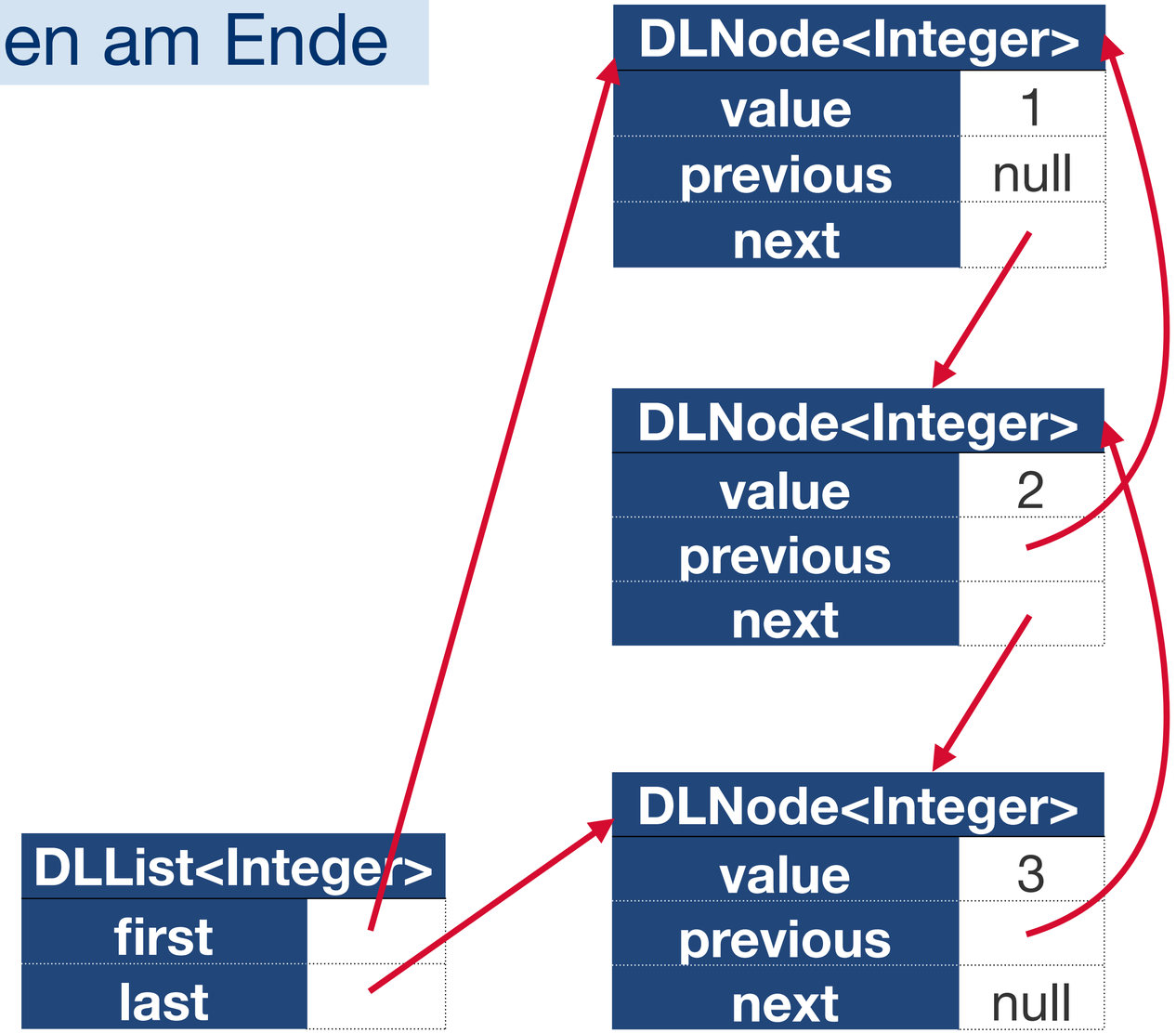


Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

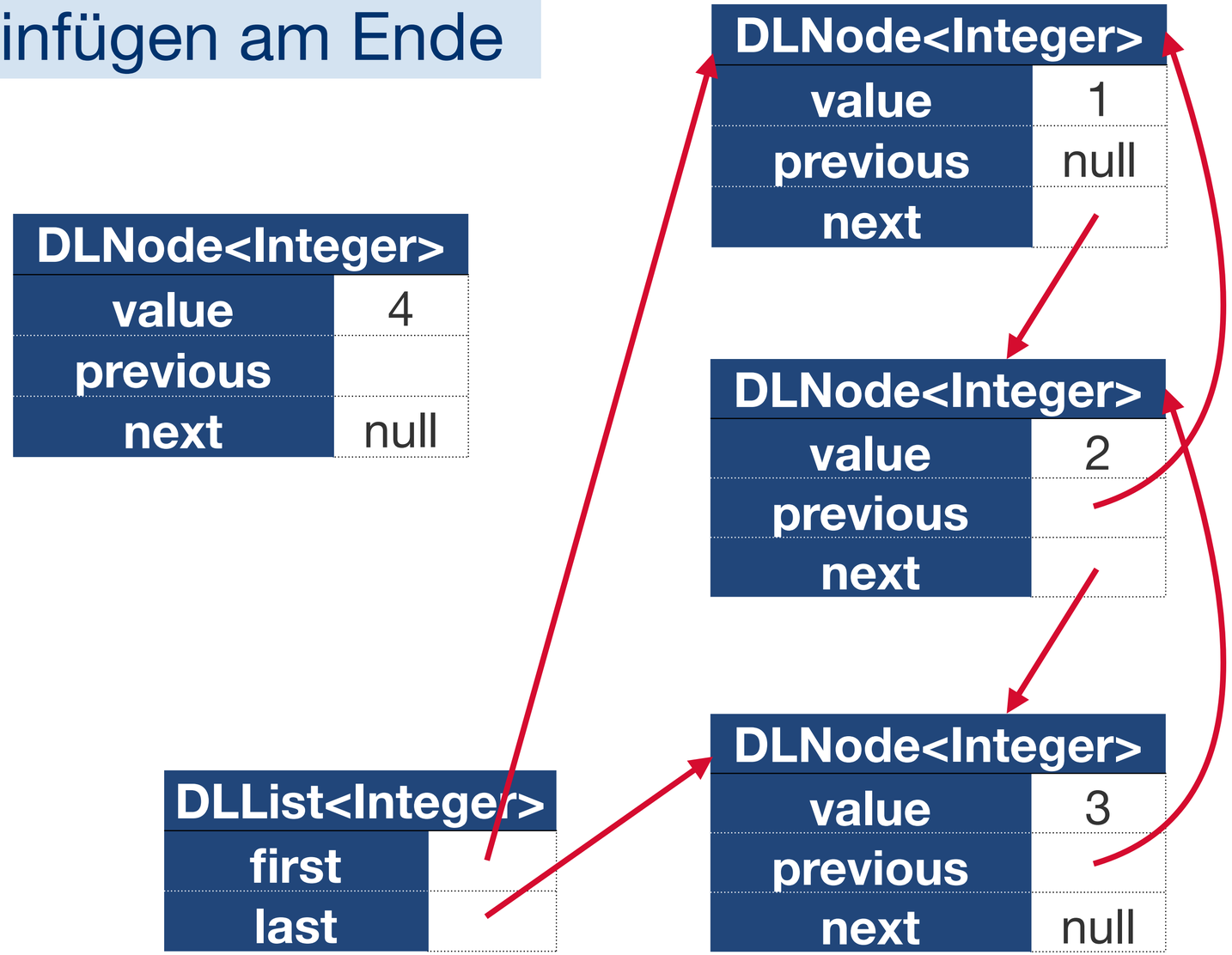


Doppelt verkettete Liste: Einfügen am Ende



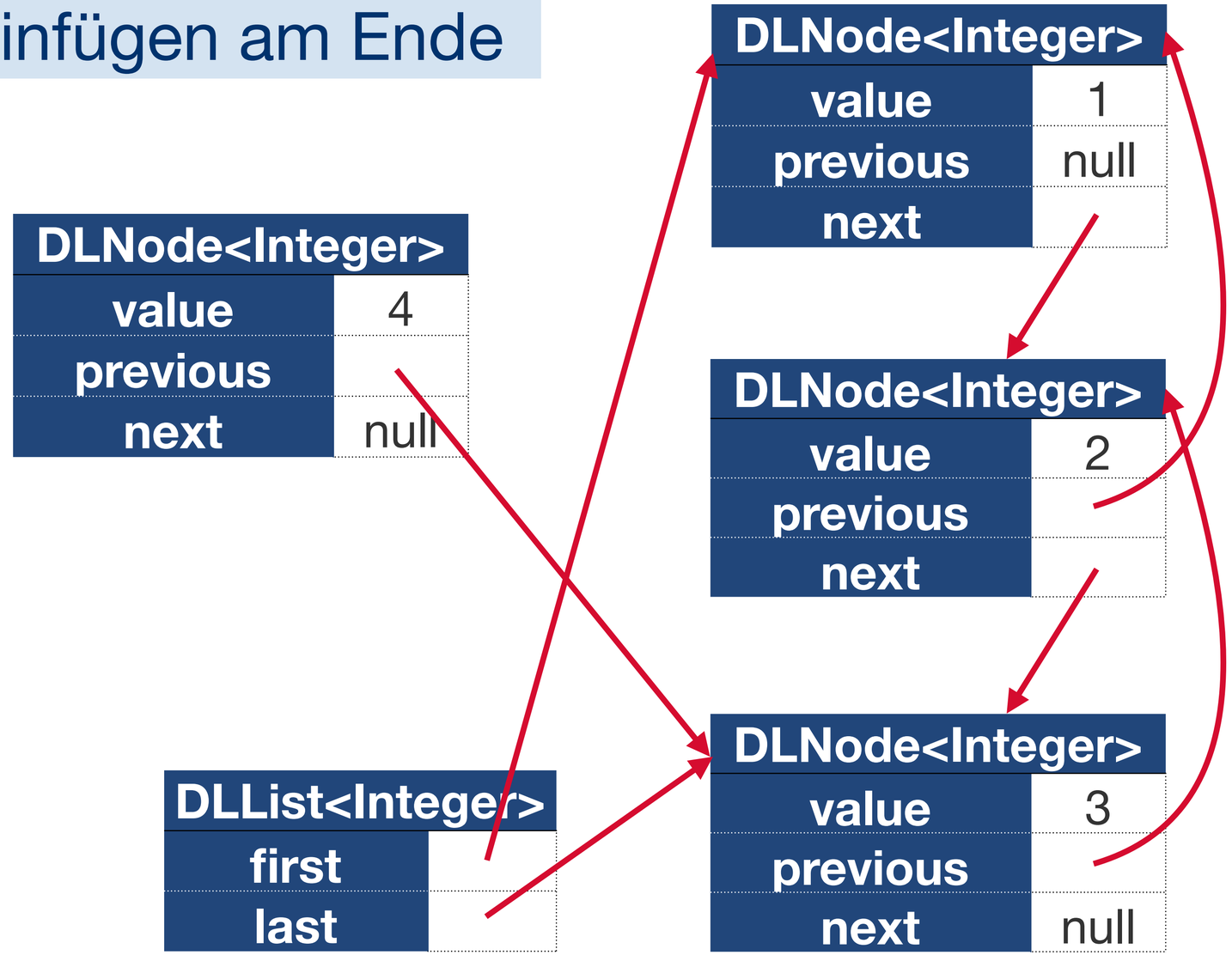
Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)



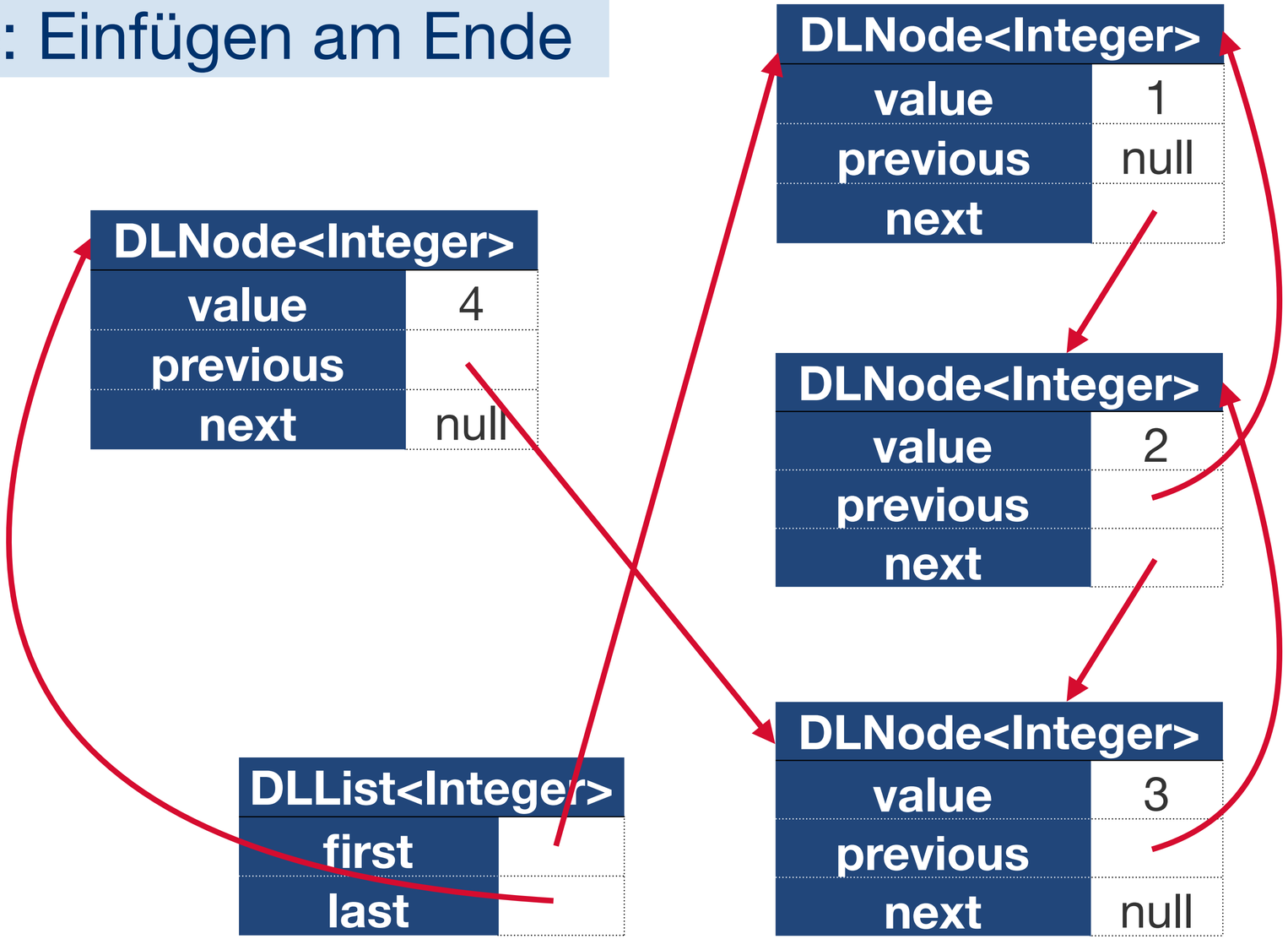
Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)



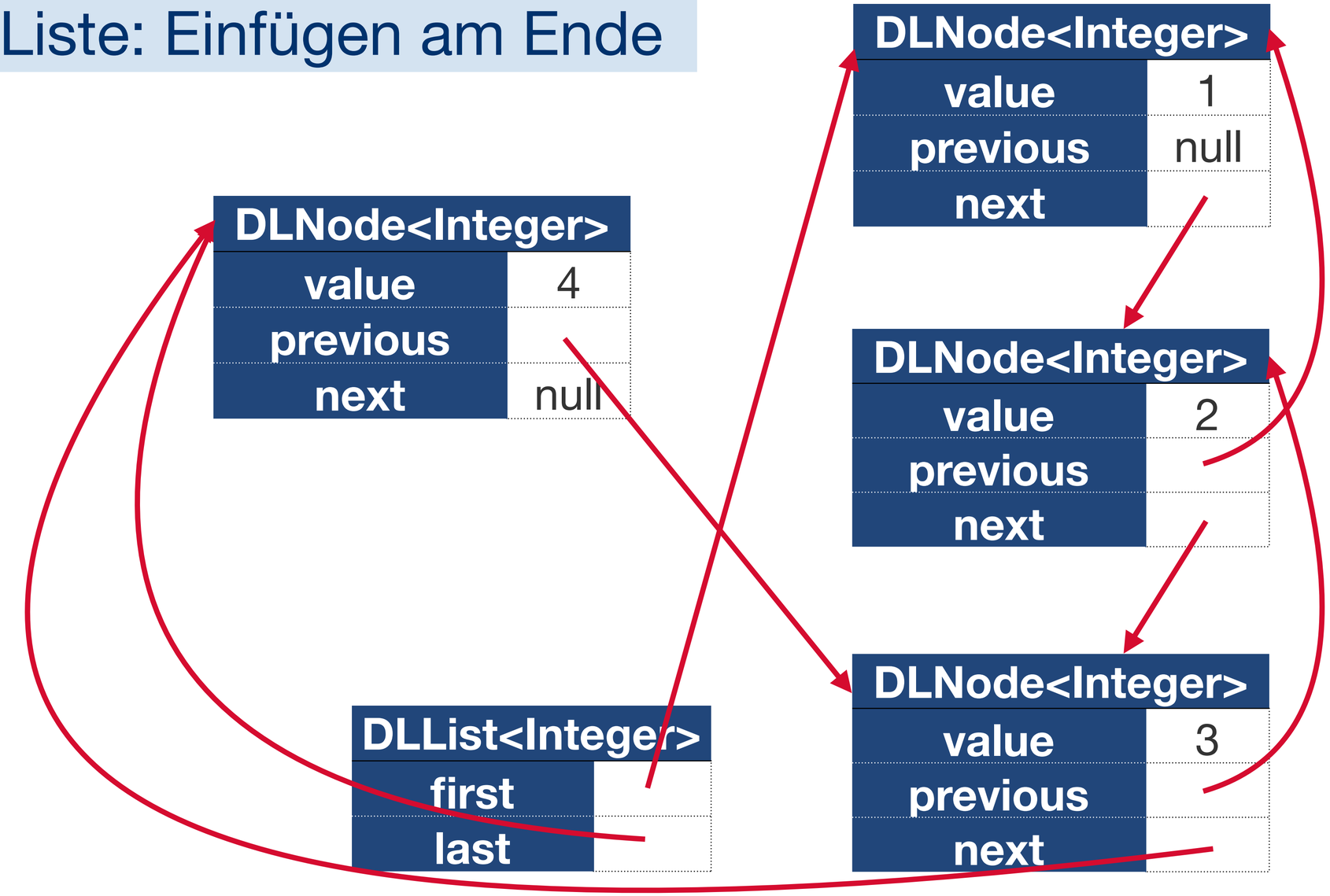
Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)



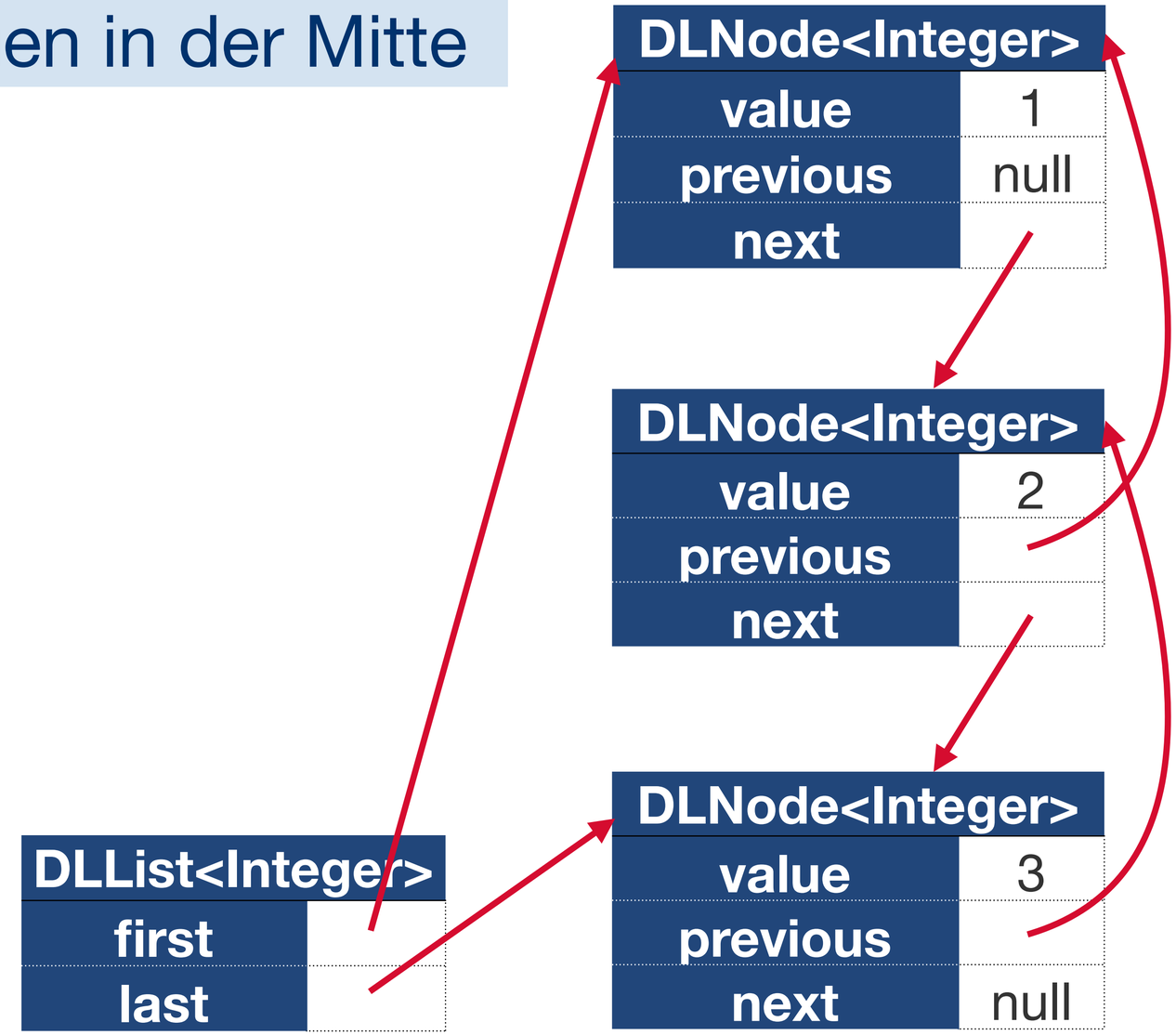
Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)



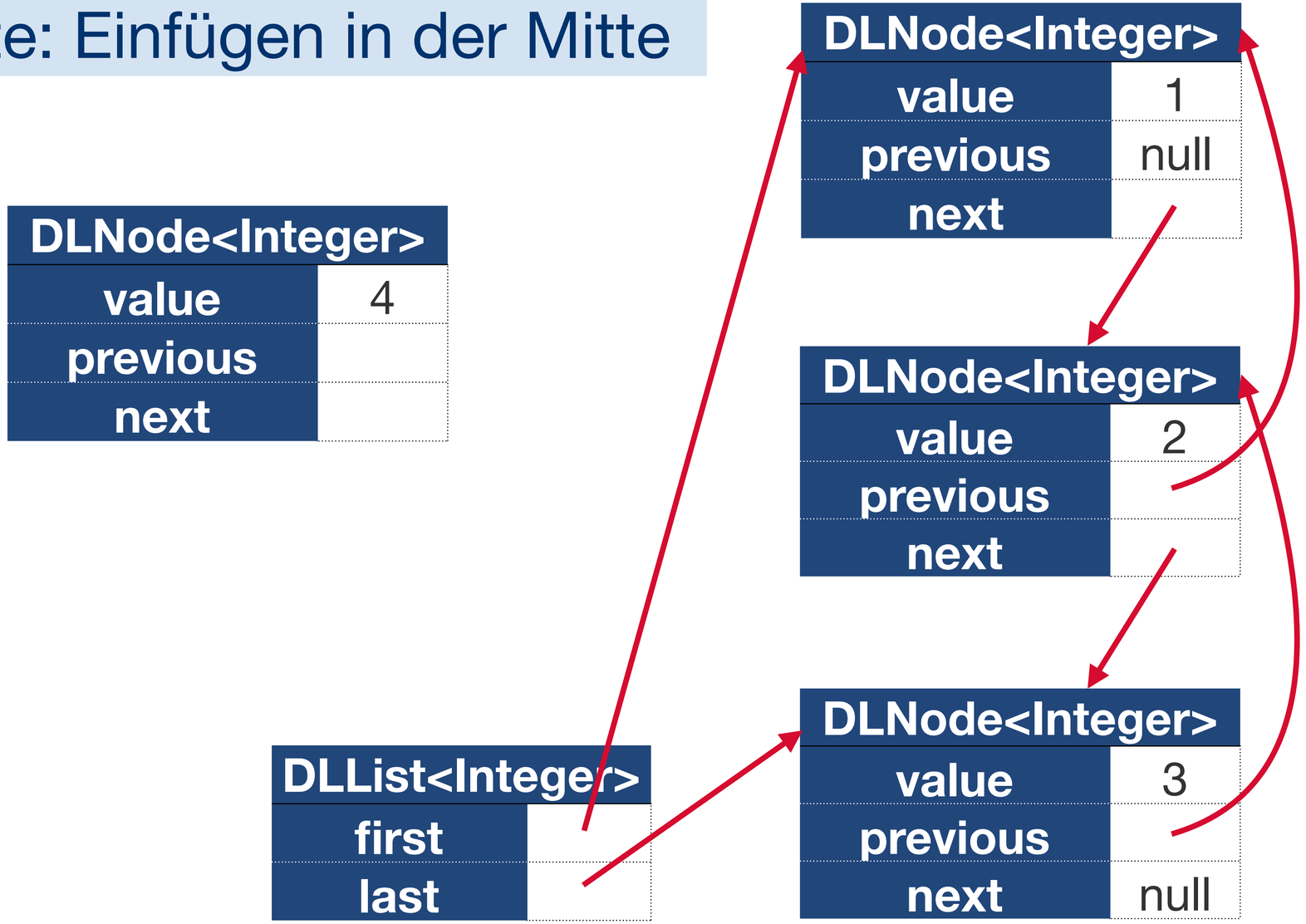
Universität Bremen

Doppelt verkettete Liste: Einfügen in der Mitte



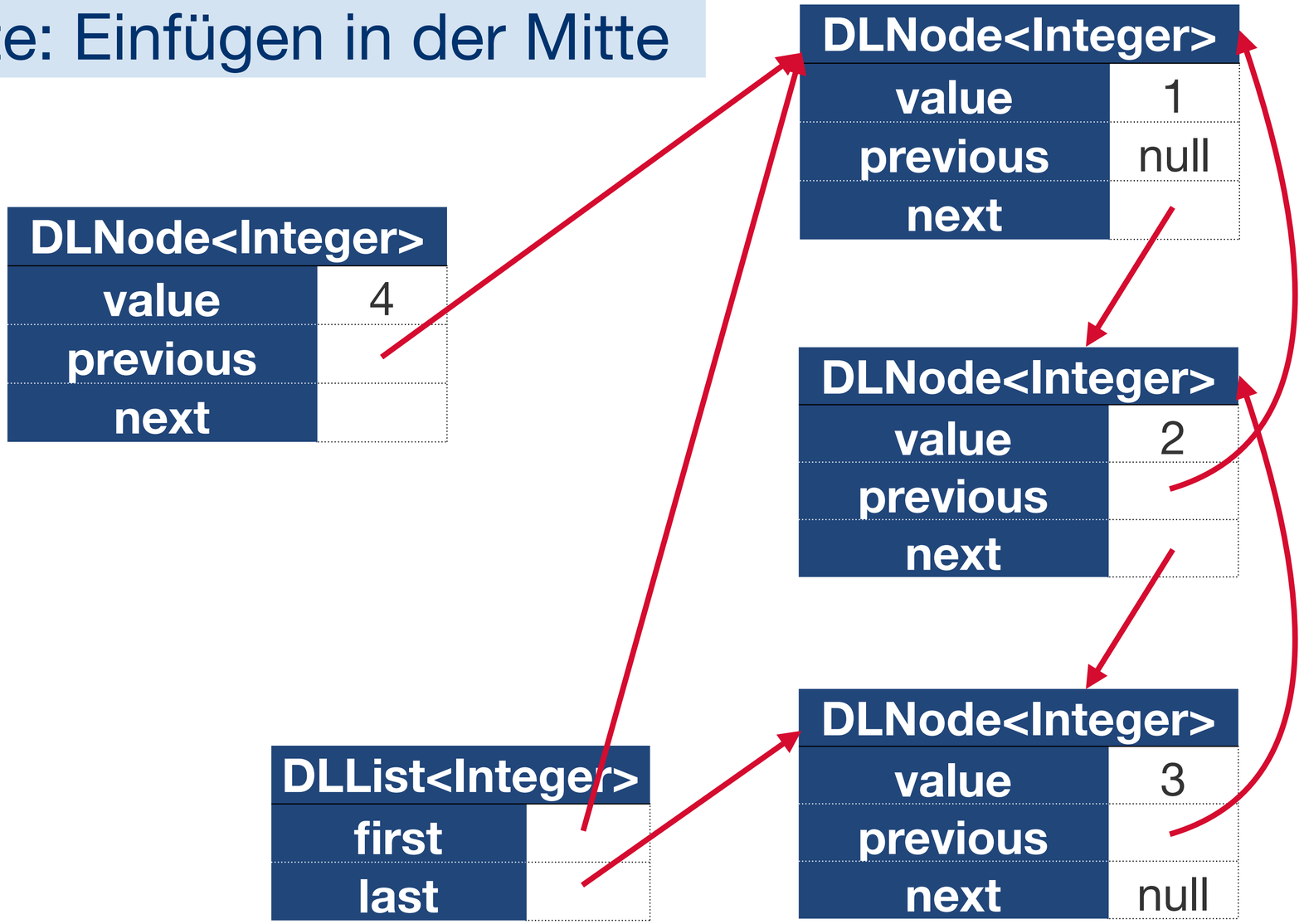
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



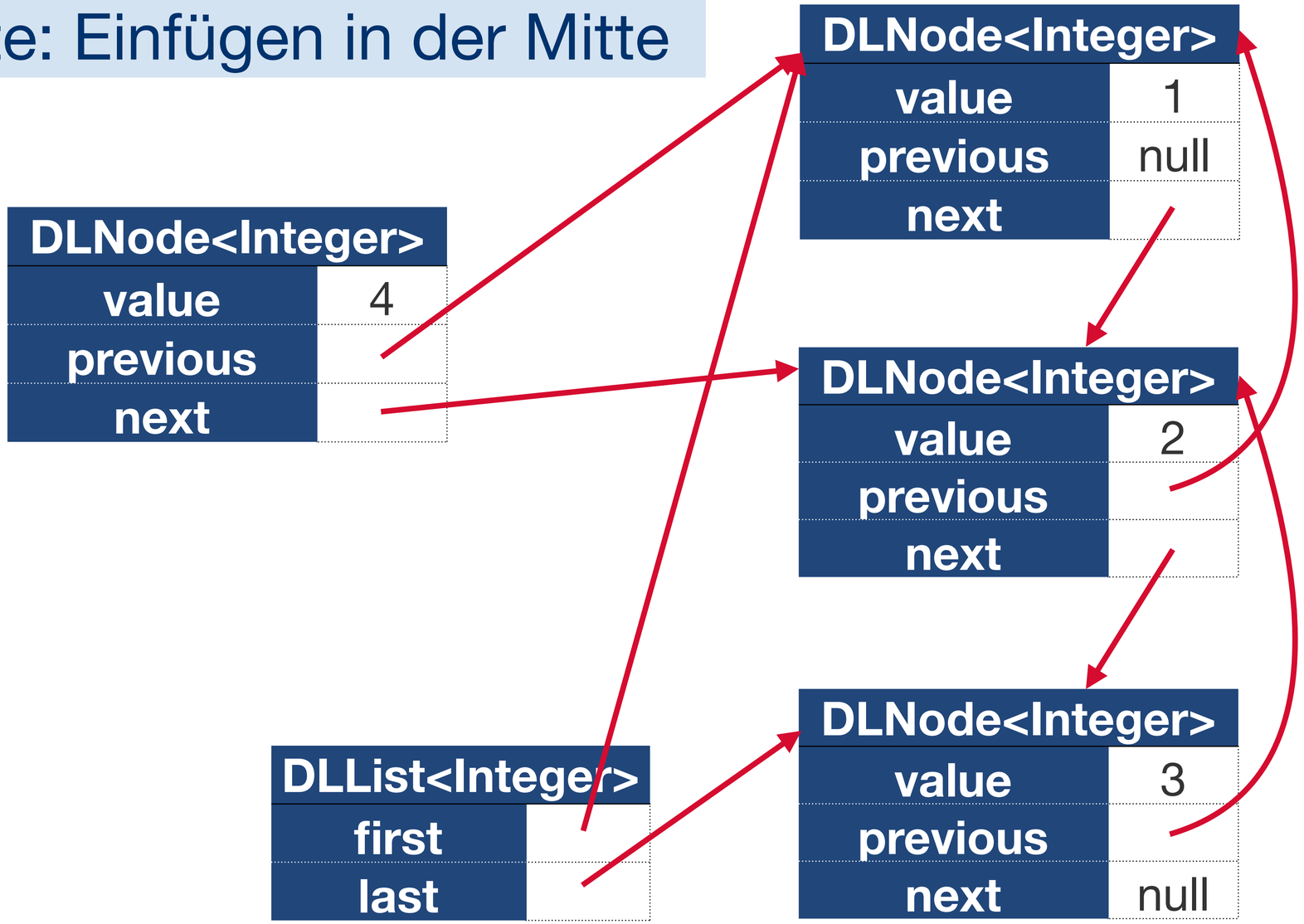
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



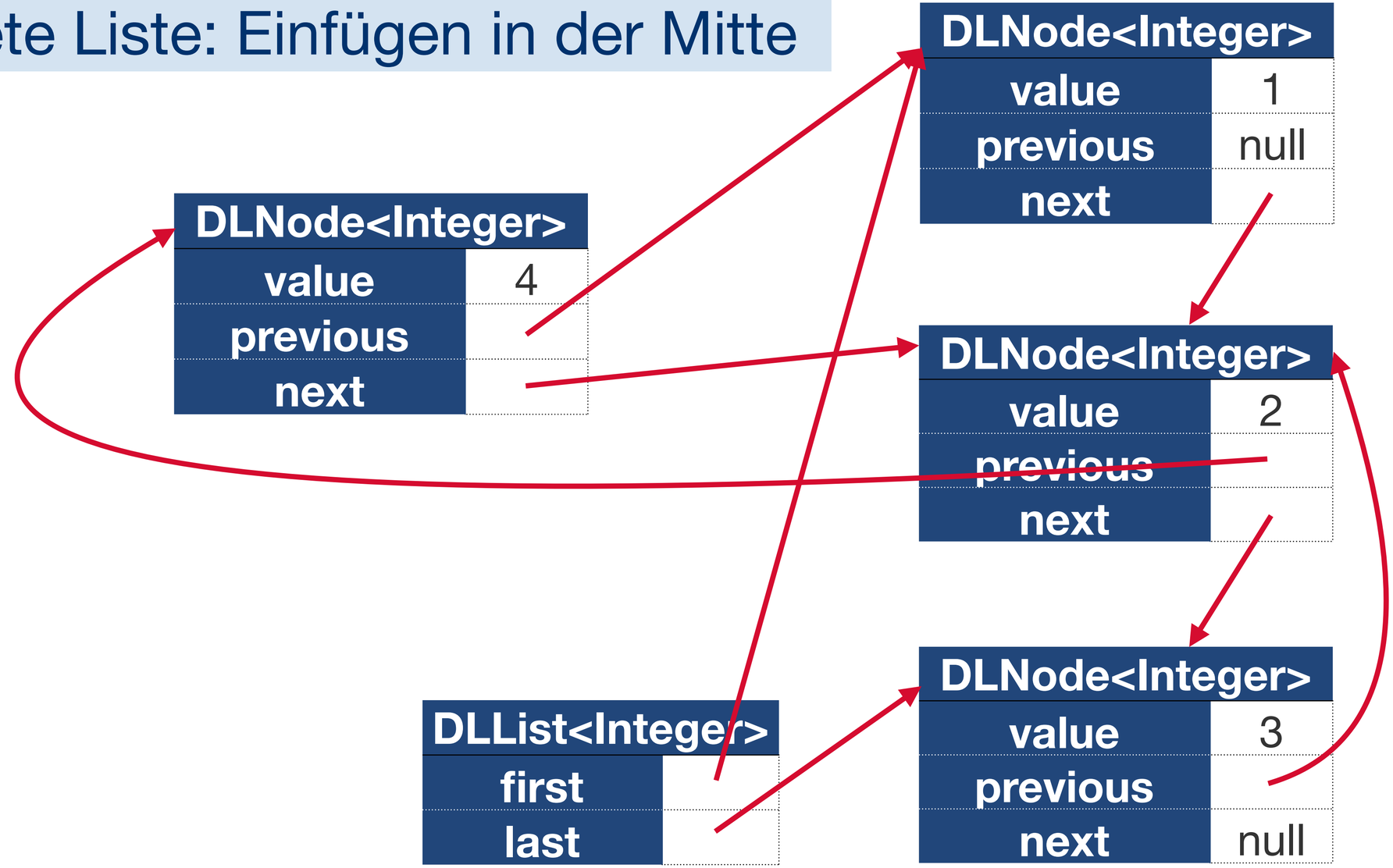
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



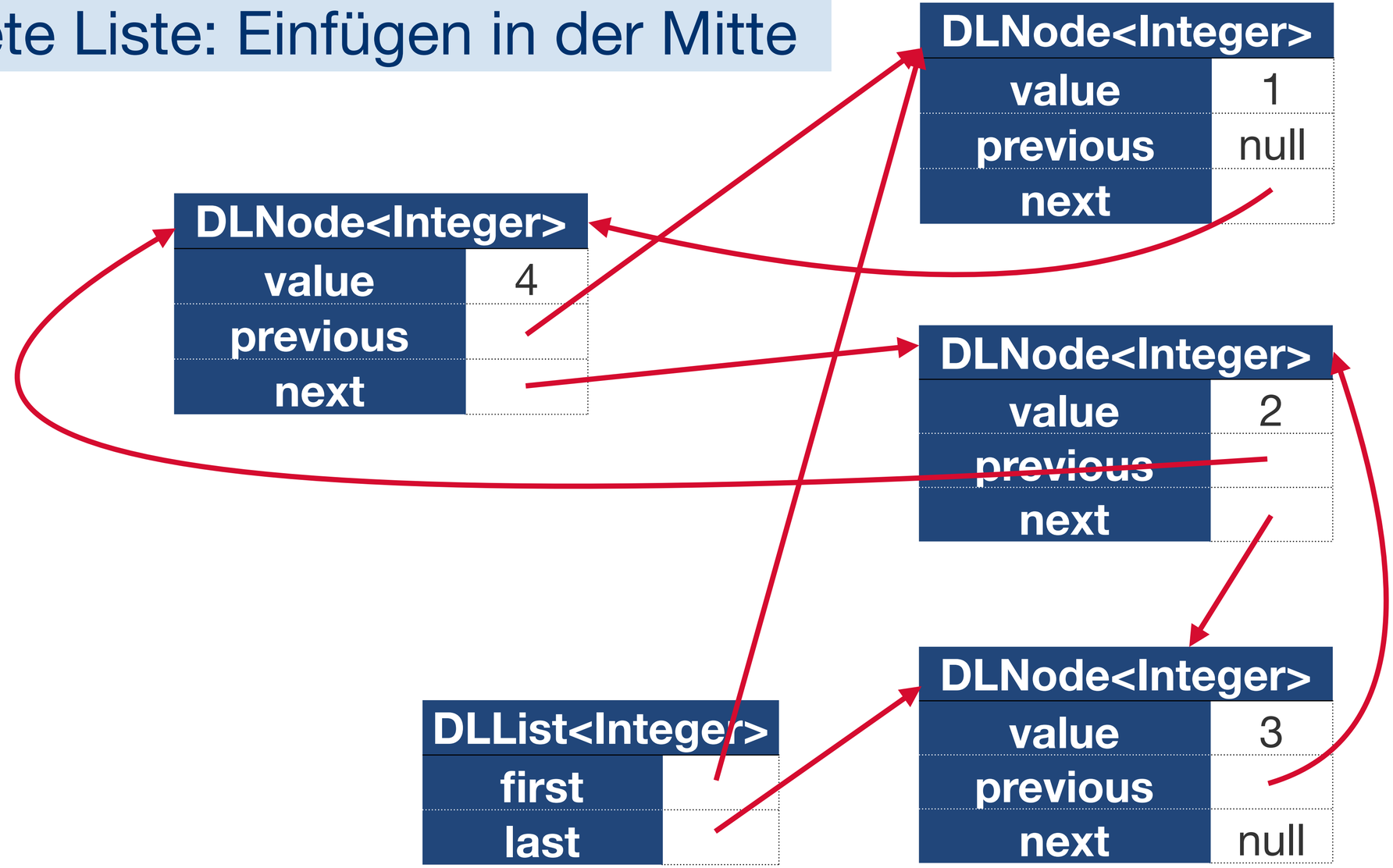
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



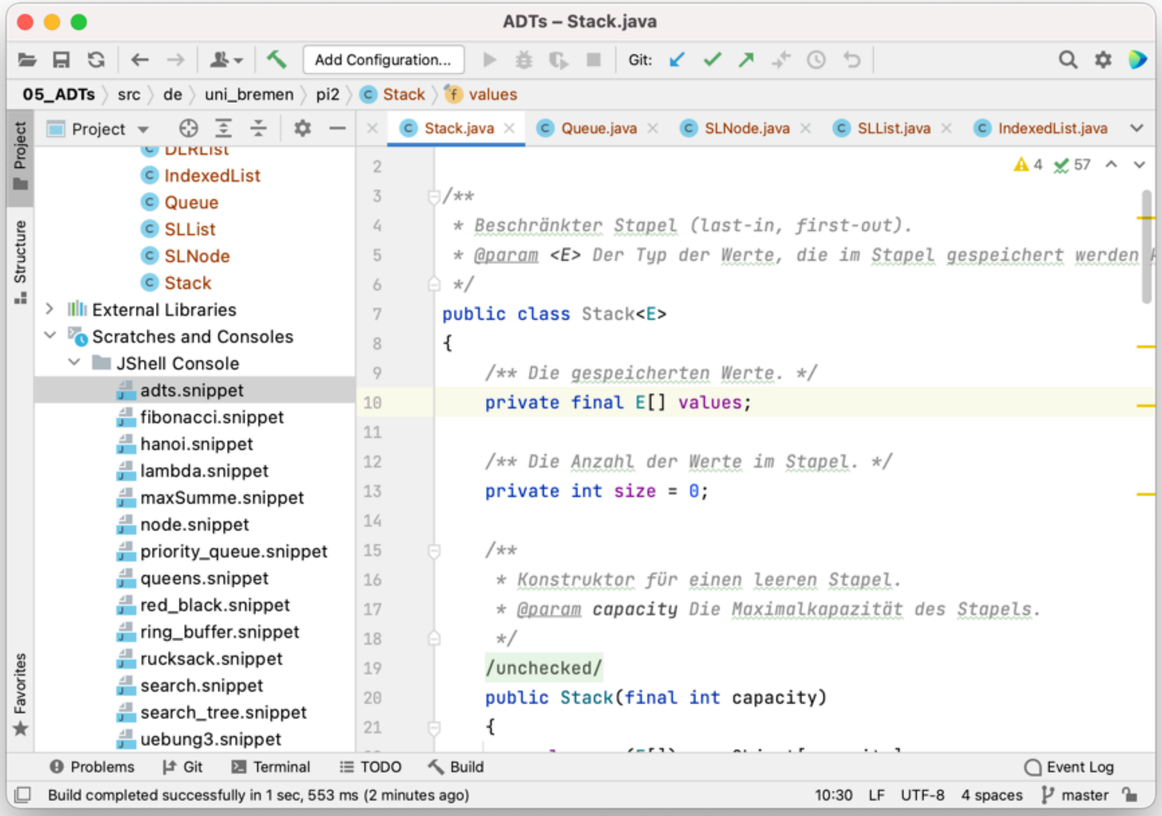
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



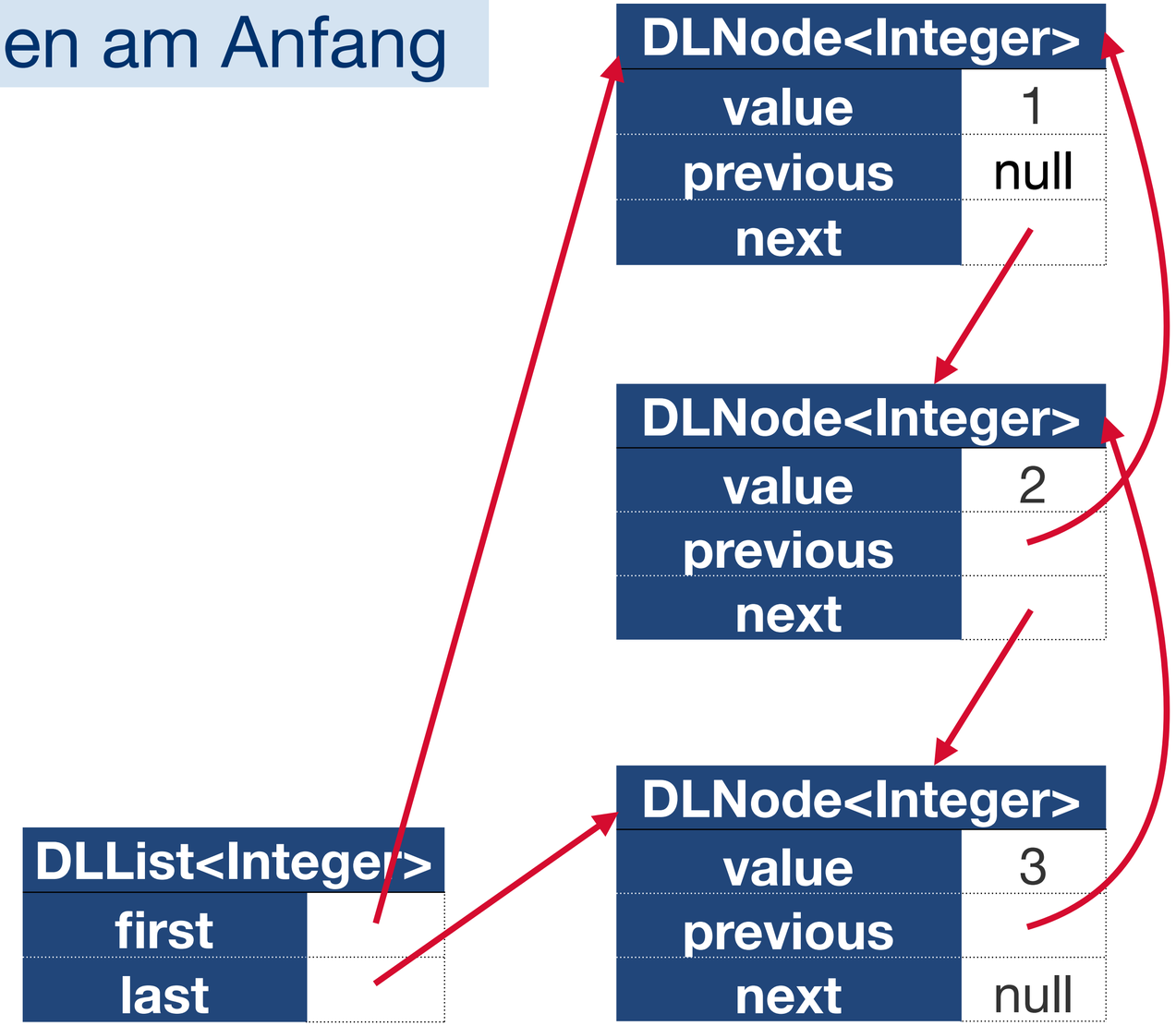
Universität Bremen

Einfügen in doppelt verkettete Liste: Demo



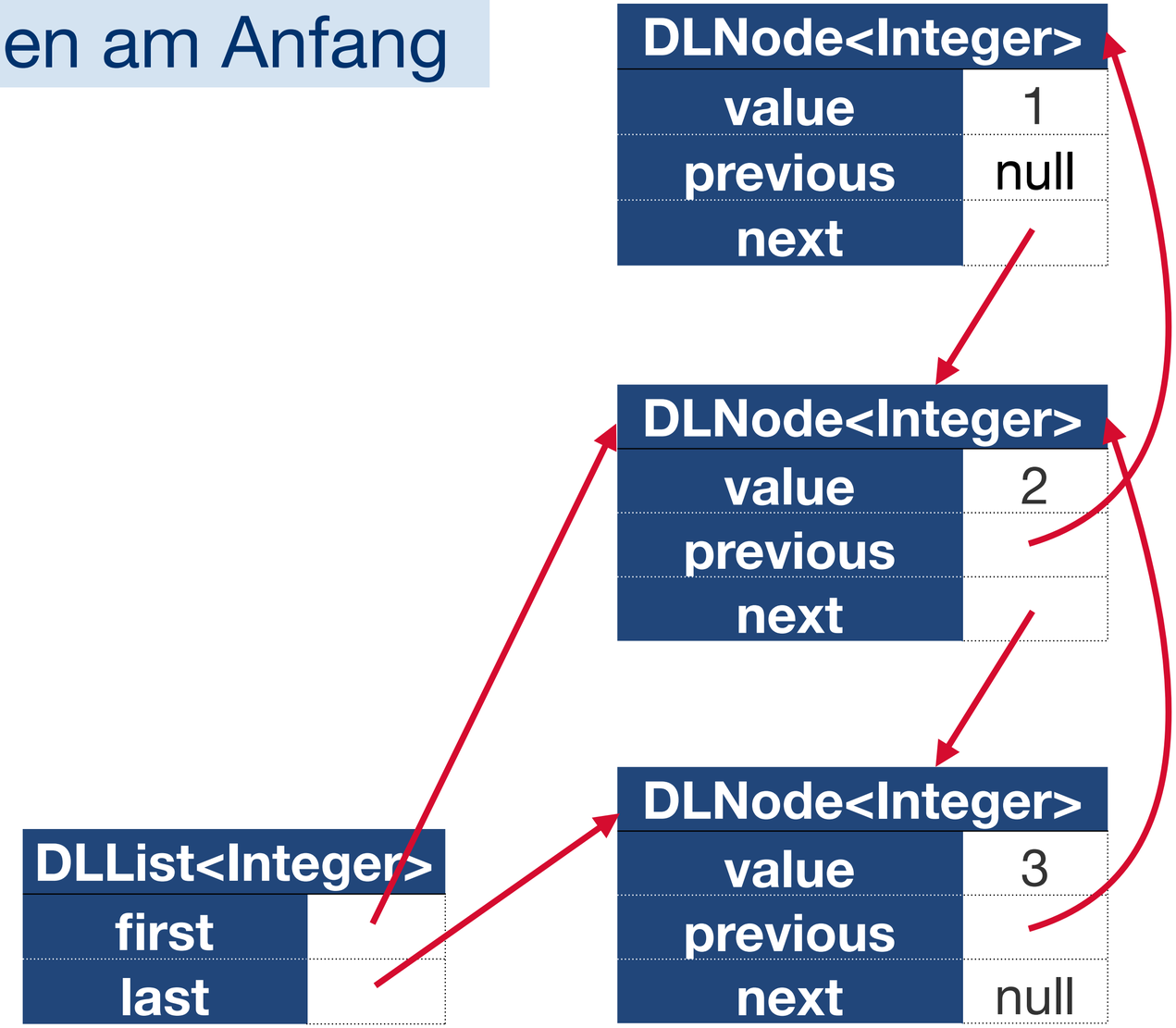
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



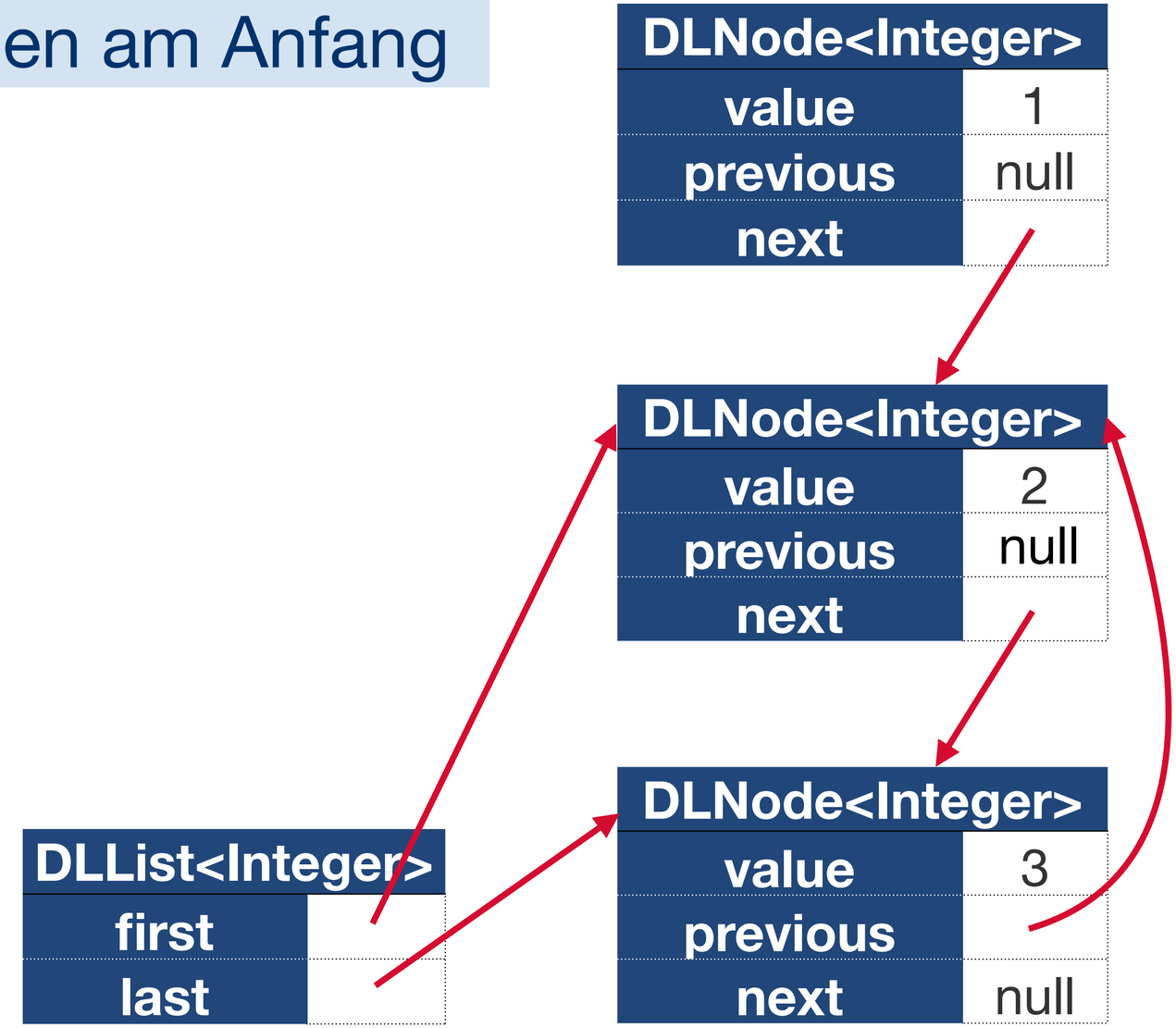
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



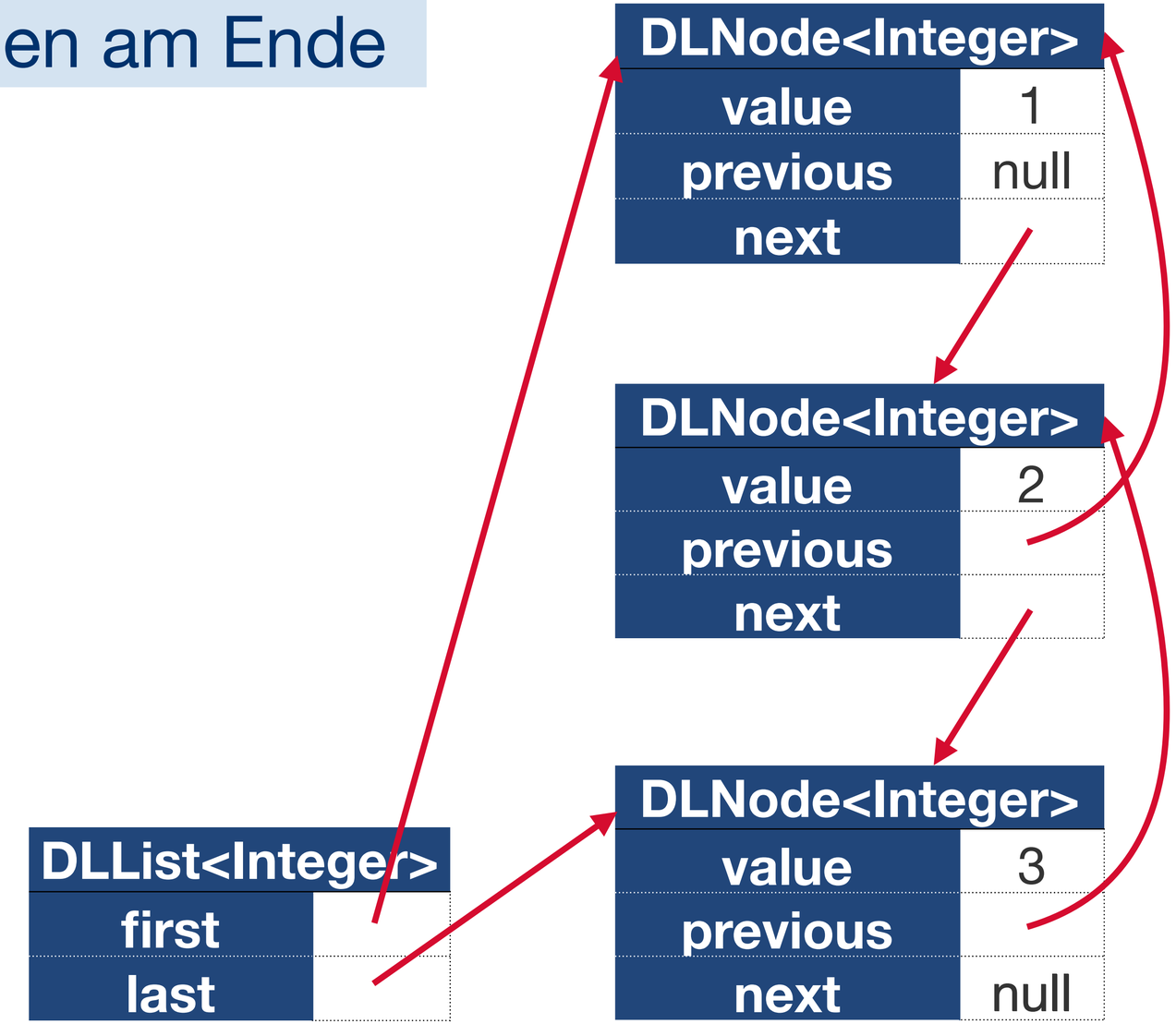
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



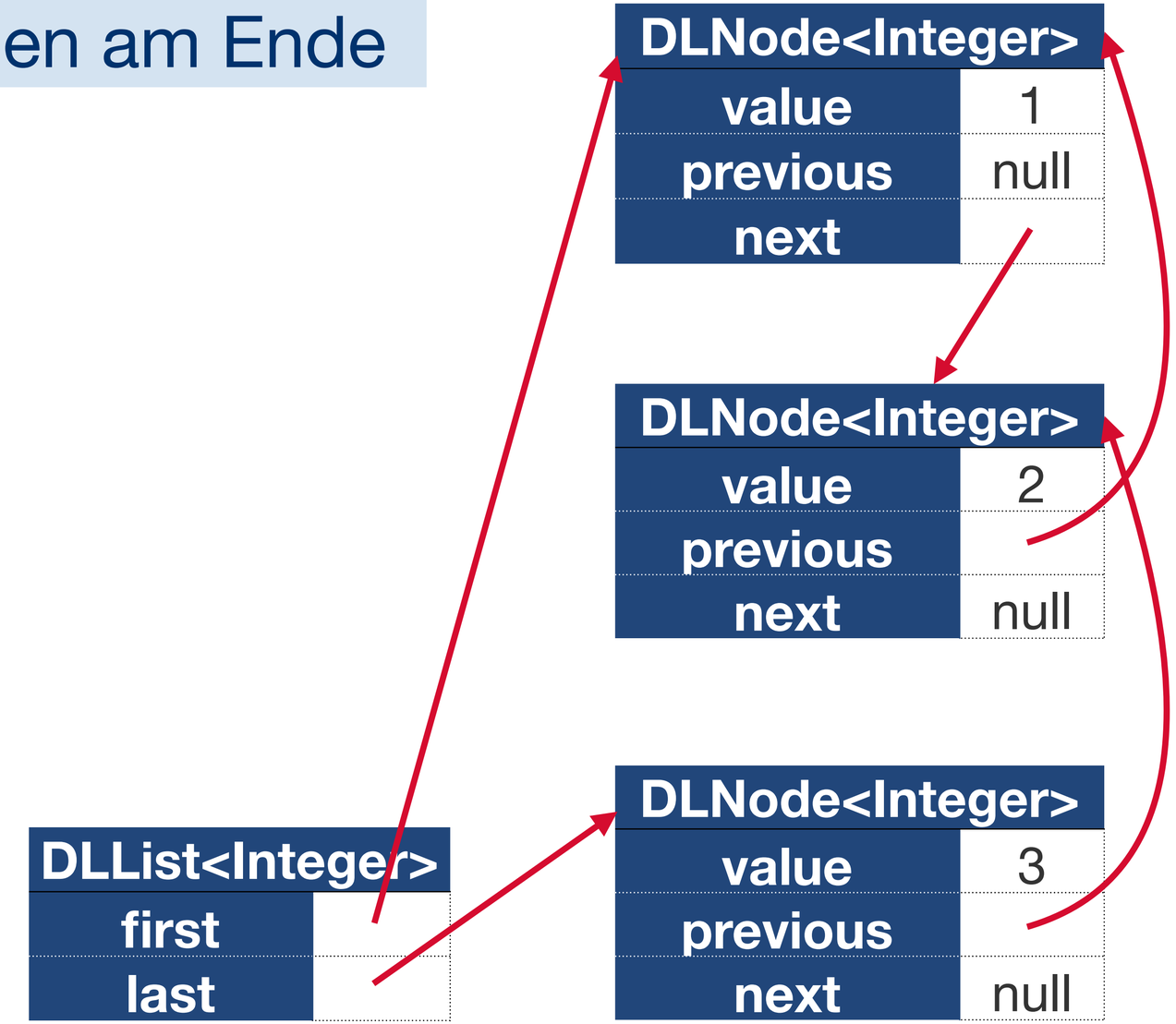
Doppelt verkettete Liste: Löschen am Ende

remove(last)



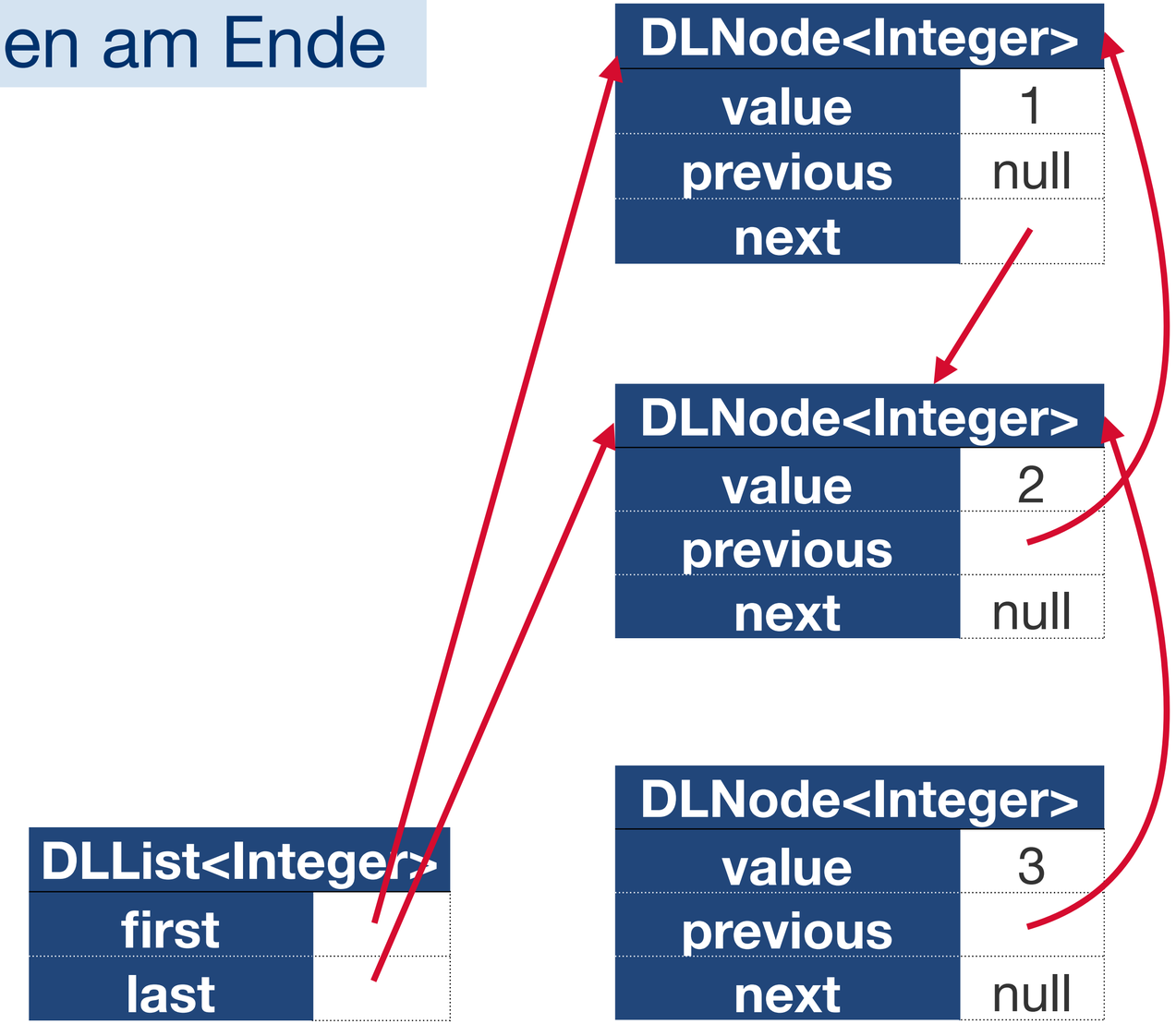
Doppelt verkettete Liste: Löschen am Ende

remove(last)



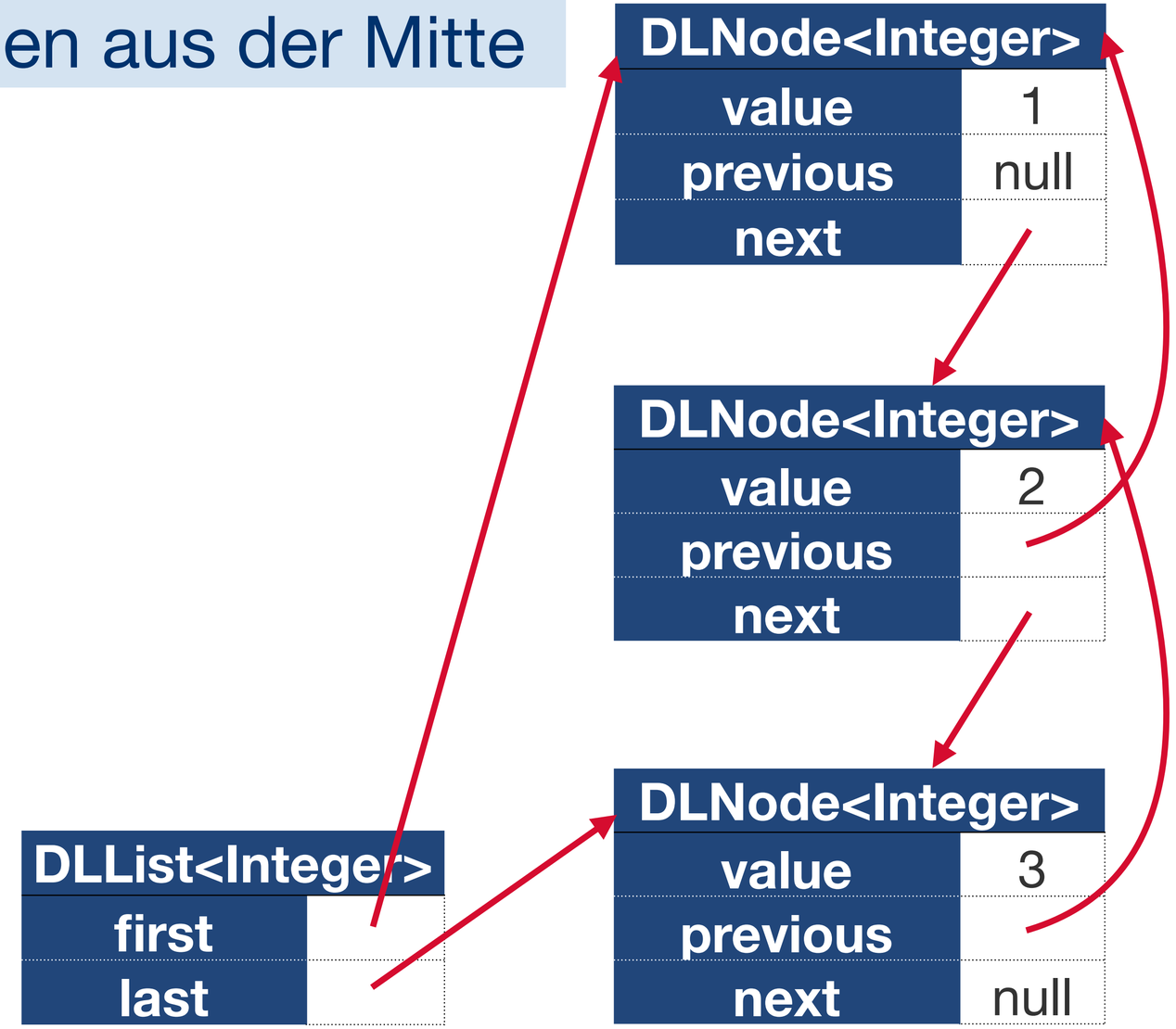
Doppelt verkettete Liste: Löschen am Ende

remove(last)



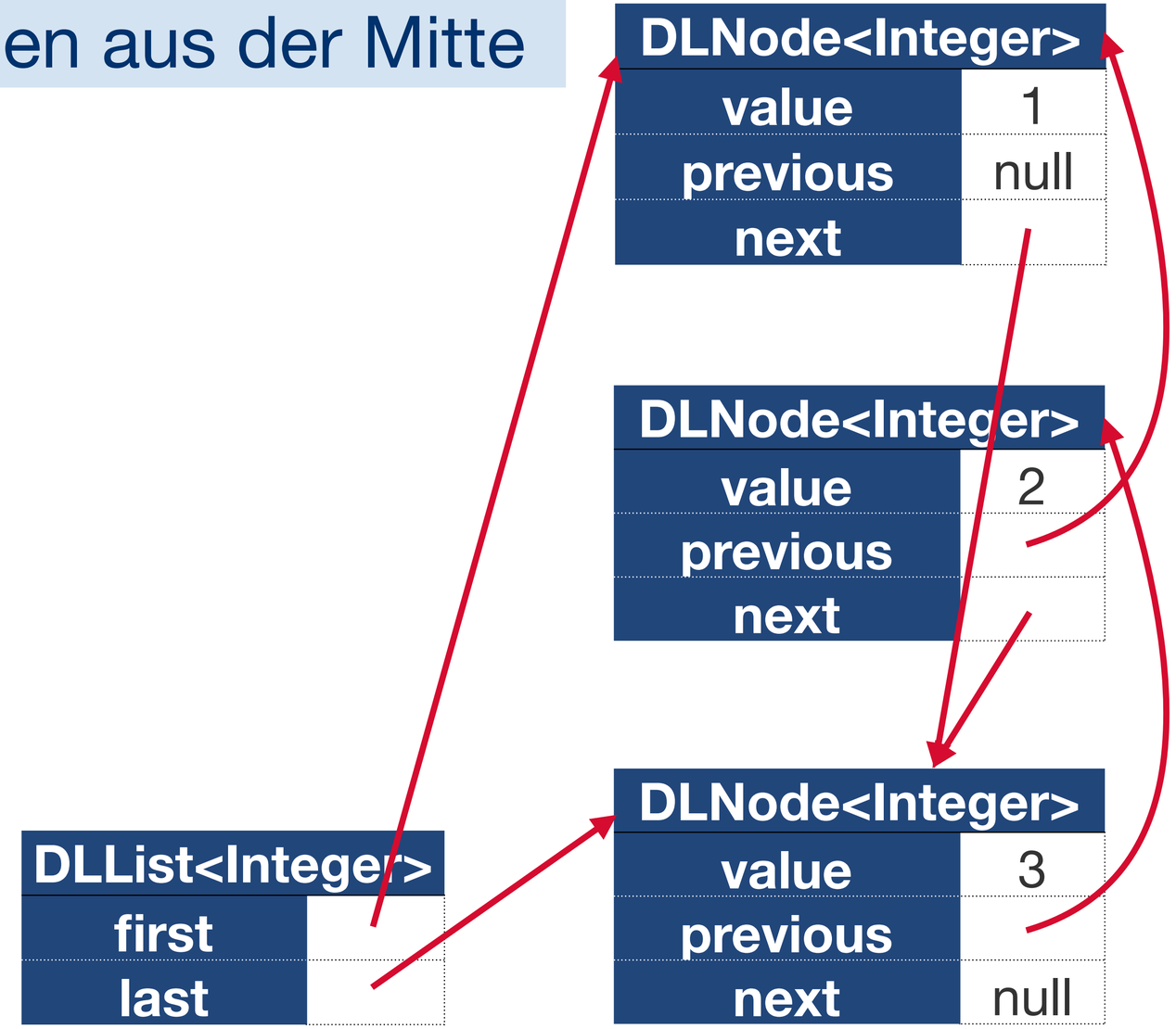
Doppelt verkettete Liste: Löschen aus der Mitte

remove(first.next)



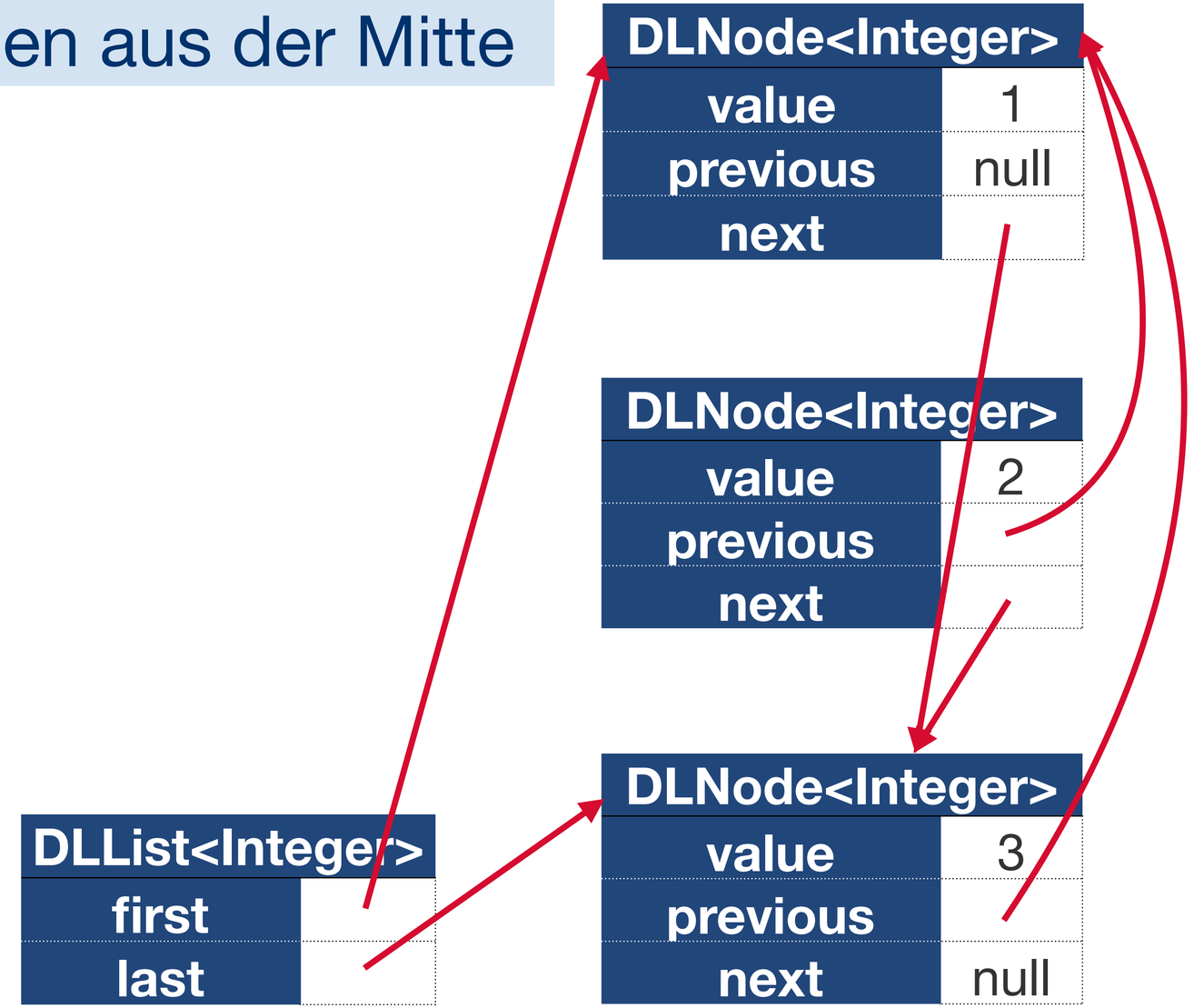
Doppelt verkettete Liste: Löschen aus der Mitte

remove(first.next)



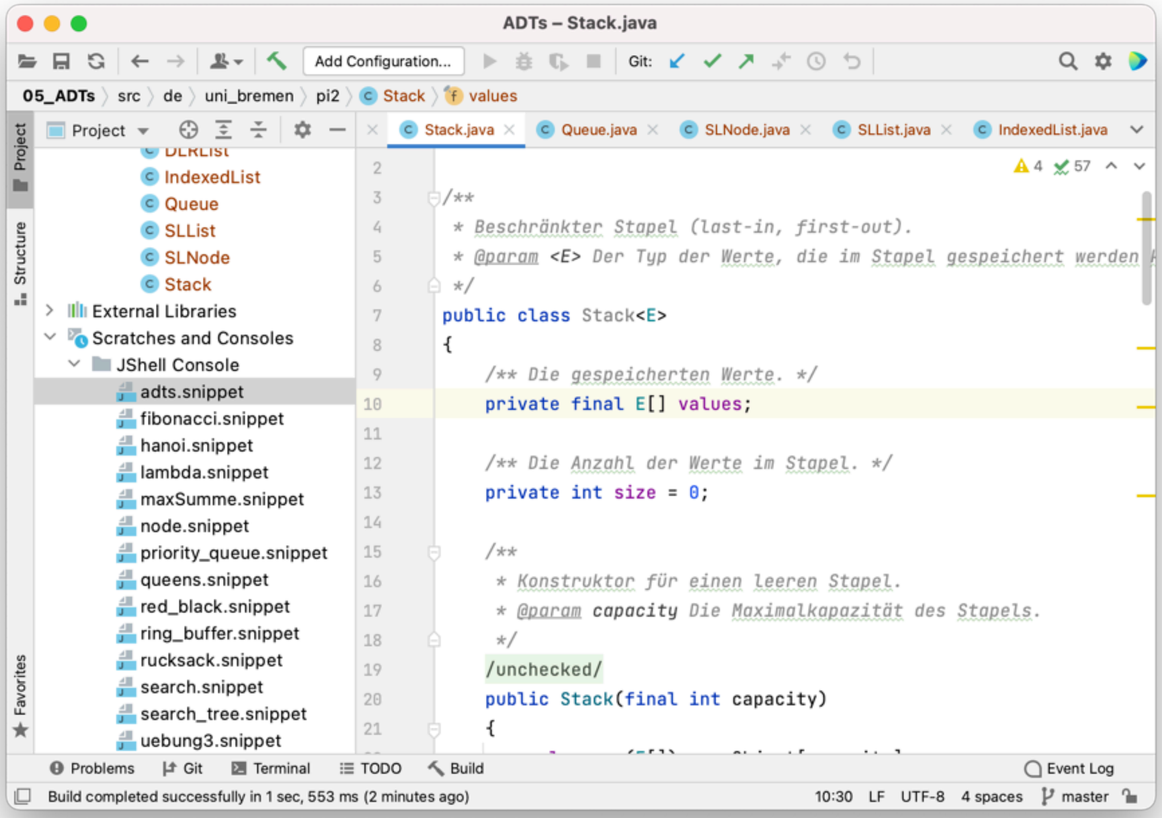
Doppelt verkettete Liste: Löschen aus der Mitte

remove(first.next)



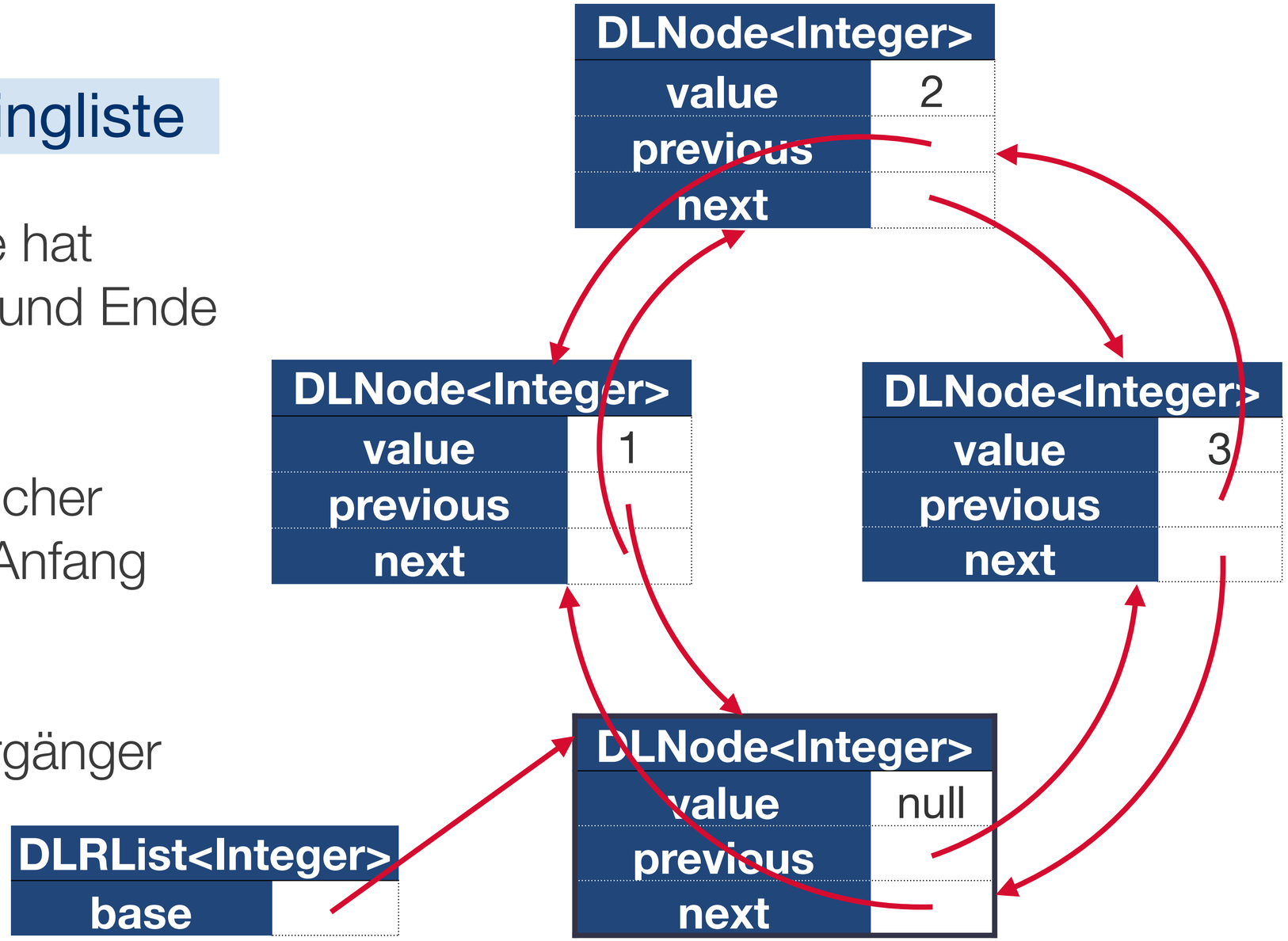
Universität Bremen

Löschen aus doppelt verketteter Liste: Demo



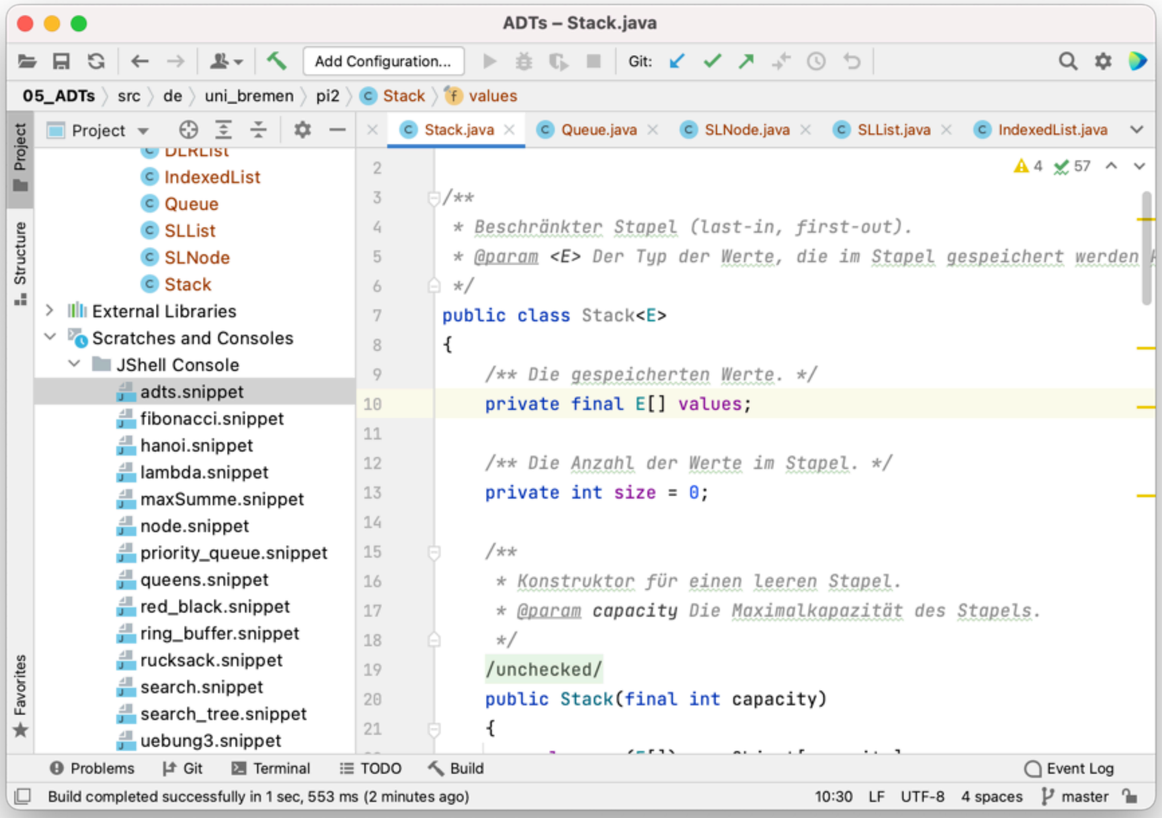
Doppelt verkettete Ringliste

- Doppelt verkettete Liste hat Sonderfälle am Anfang und Ende der Liste
- Ringliste: leerer, zusätzlicher Knoten gleichzeitig als Anfang und Ende
  - Jeder Knoten hat Vorgänger und Nachfolger



Universität Bremen

Doppelt verkettete Ringliste: Demo



Zusammenfassung der Konzepte

- **Abstrakter Datentyp**
- **Stapel** und **Warteschlange** (**beschränkt**/**unbeschränkt**)
- **Einfach** und **doppelt verkettete (Ring-)Liste**
- **Liste mit Index**