

UNTERSCHIEDLICHE

---

**ABSTRAKTE DATENTYPEN**

# ABSTRAKTER DATENTYPEN (ADT) - WAS SOLL DAS SEIN?

- ▶ ein ADT ist (allein) über Operationen definiert, die man mit seinen Objekten ausführen kann
- ▶ interne Struktur der Objekte ist nach außen hin unsichtbar  
→ restriktive (eingeschränkte) Form einer Klasse, in der das Geheimprinzip in besonders strenger Weise realisiert wird
- ▶ während in einer „normalen“ Klasse über „setAttribut“ und „getAttribut()“-Methoden ein (kontrollierter) Zugriff auf einzelne Attribute möglich ist, ist bei einem ADT der direkte Zugriff grundsätzlich nicht gestattet → die Umwelt weiß nicht einmal, welche Attribute die Klasse besitzt

# QUEUE - SCHLANGE/WARTESCHLANGE

## ► Grundidee:

- man muss sich hinten anstellen
- derjenige, der zuerst gekommen ist und deshalb ganz vorne in der Schlange steht, wird als erster bedient

## ► FiFo - Prinzip → „First in First out“

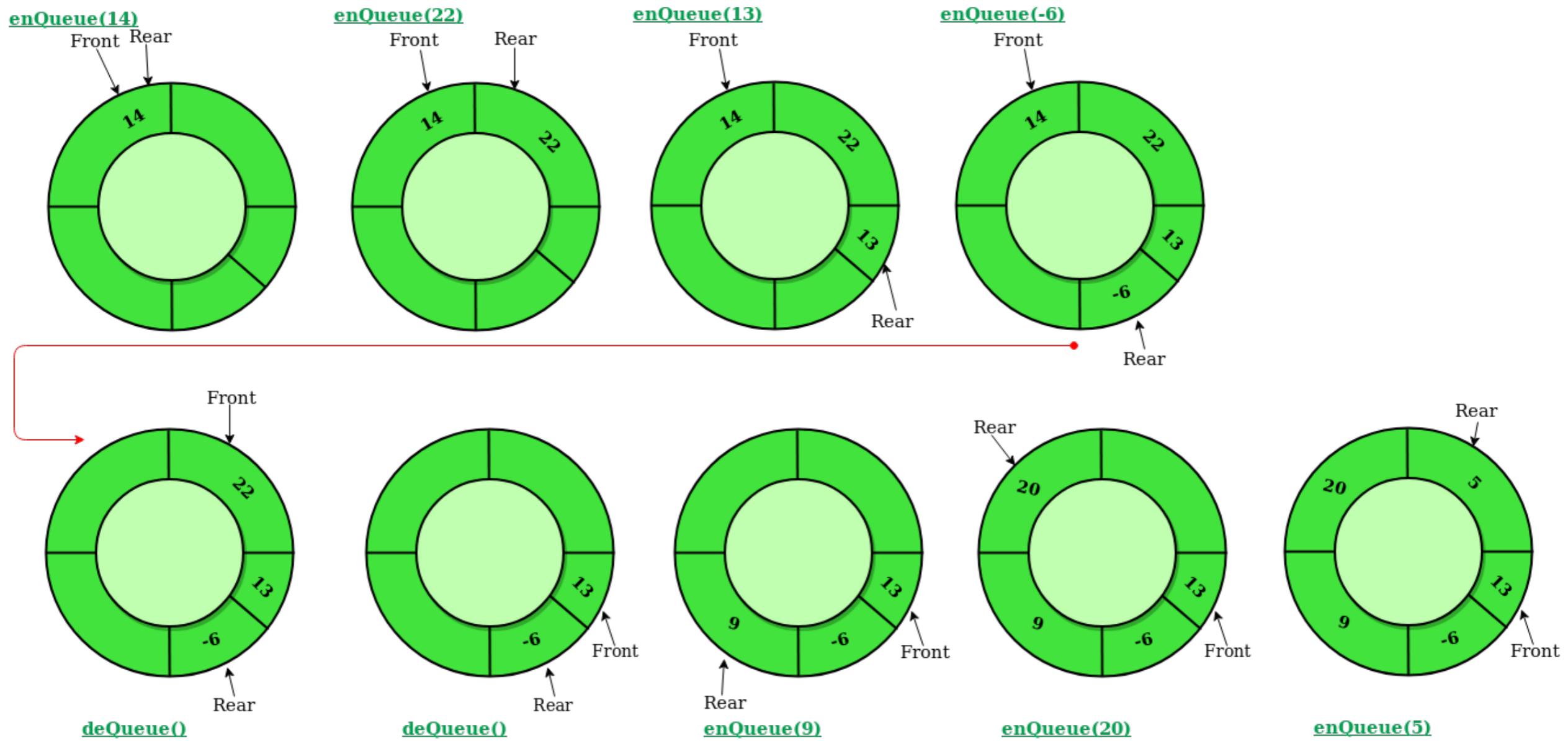
## ► Beispiele für die Anwendung:

- Aufträge an langsame Ausgabegeräte wie Drucker
- Verarbeitung von Mausklicks in graphischen Benutzeroberflächen
- Pufferung von Daten bei Datenübergabe zwischen asynchronen Prozessen

## ► Operationen, die Schlangen ausführen können:

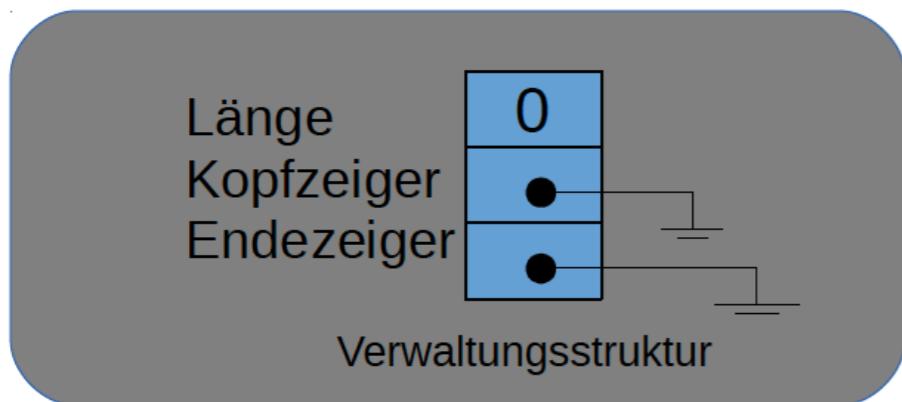
- eine neue leere Schlange erzeugen = Schlange()
- ein Objekt hinten an die Schlange anhängen = Einreihen()
- ein Objekt vorne von der Schlange entfernen = Bedienen()
- das vorderste Element lesen, ohne es aus der Schlange zu entfernen = Kopf()
- Testen, ob die Schlange leer ist = istLeer()
- ergänzend: Laenge()
- Implementieren und Testprogramm schreiben

# QUEUE - RINGPUFFER

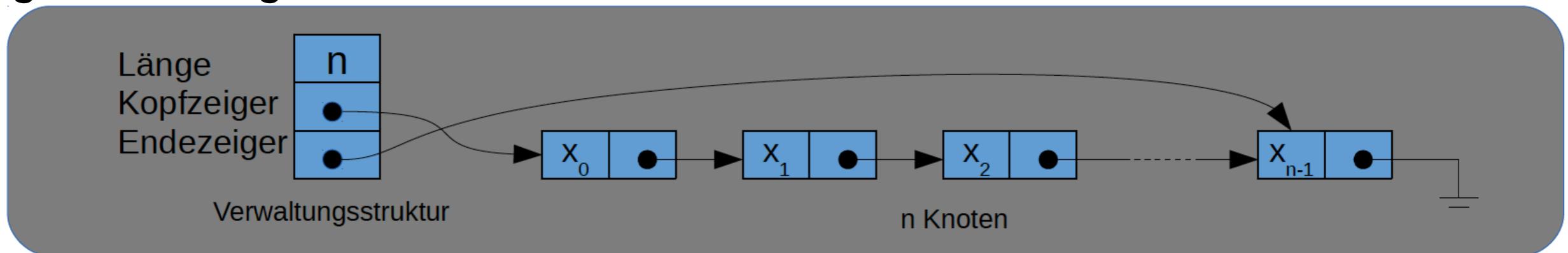


# EINFACH VERKETTETE LISTE

## leere Schlange:



## gefüllte Schlange:



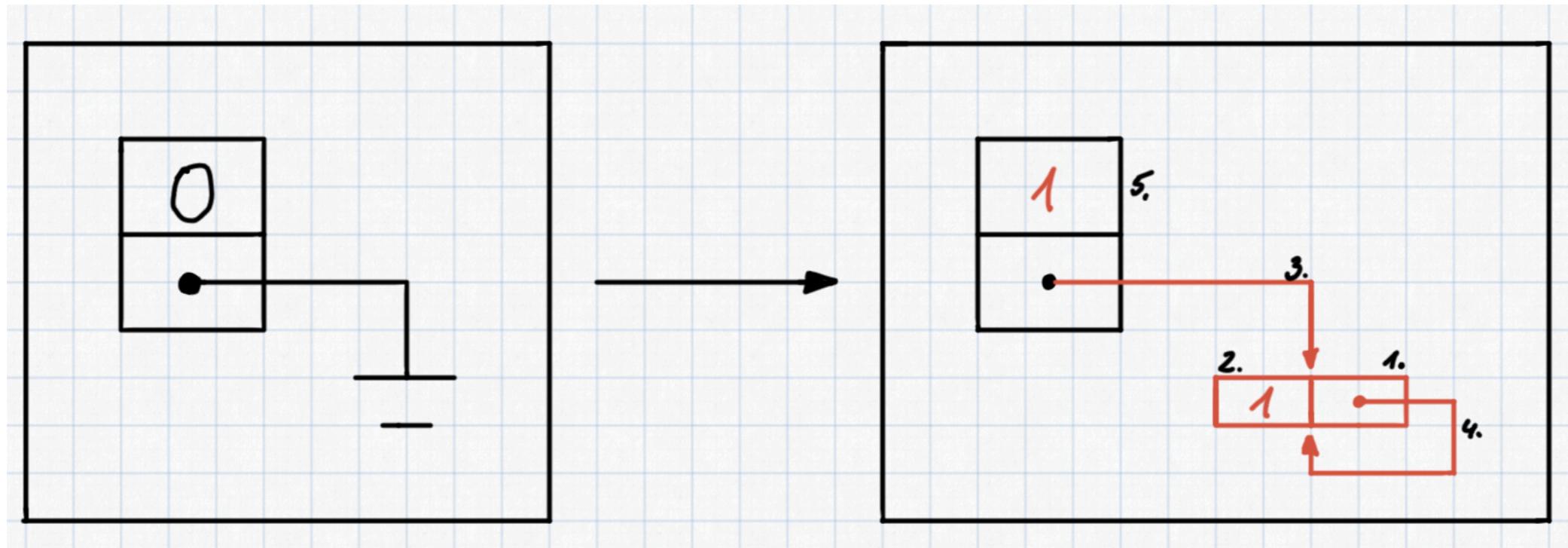
## Repräsentation in Python:

```
class Schlaenge():
    def __init__(self):
        self.laenge = 0
        self.kopfzeiger = None
        self.endzeiger = None
```

```
class Knoten():
    def __init__(self):
        self.inhalt = None
        self.naechster = None
```

# EINFACH VERKETTETE RINGLISTE

## leere Schlange:

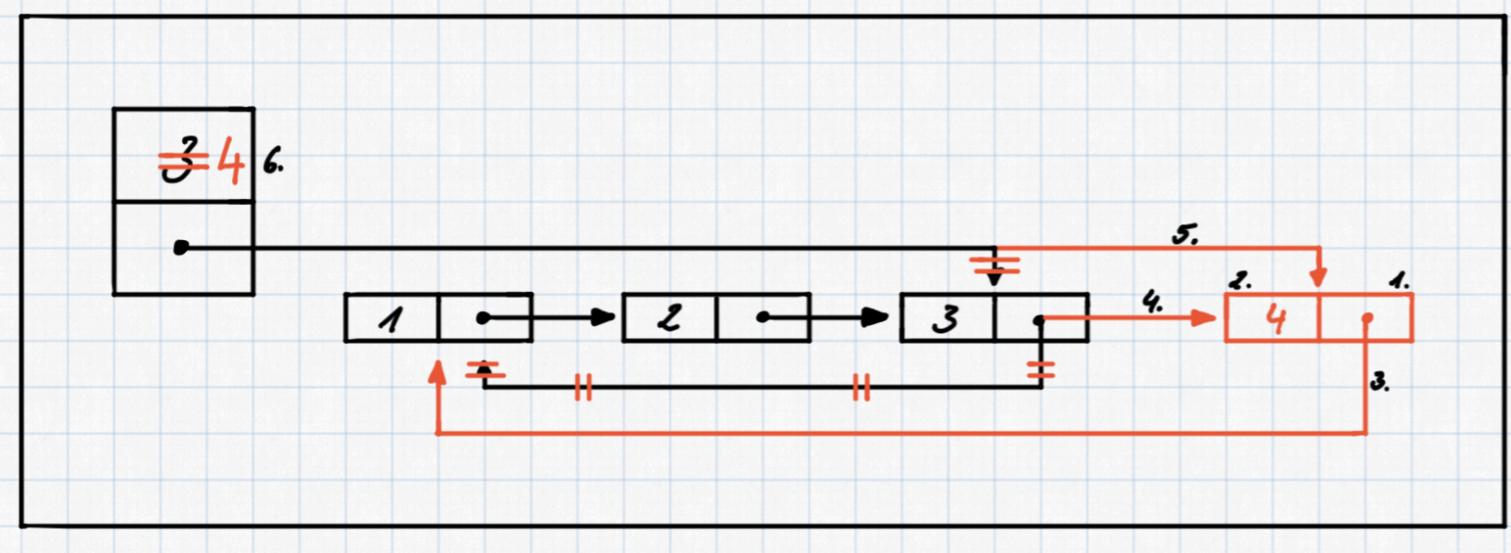
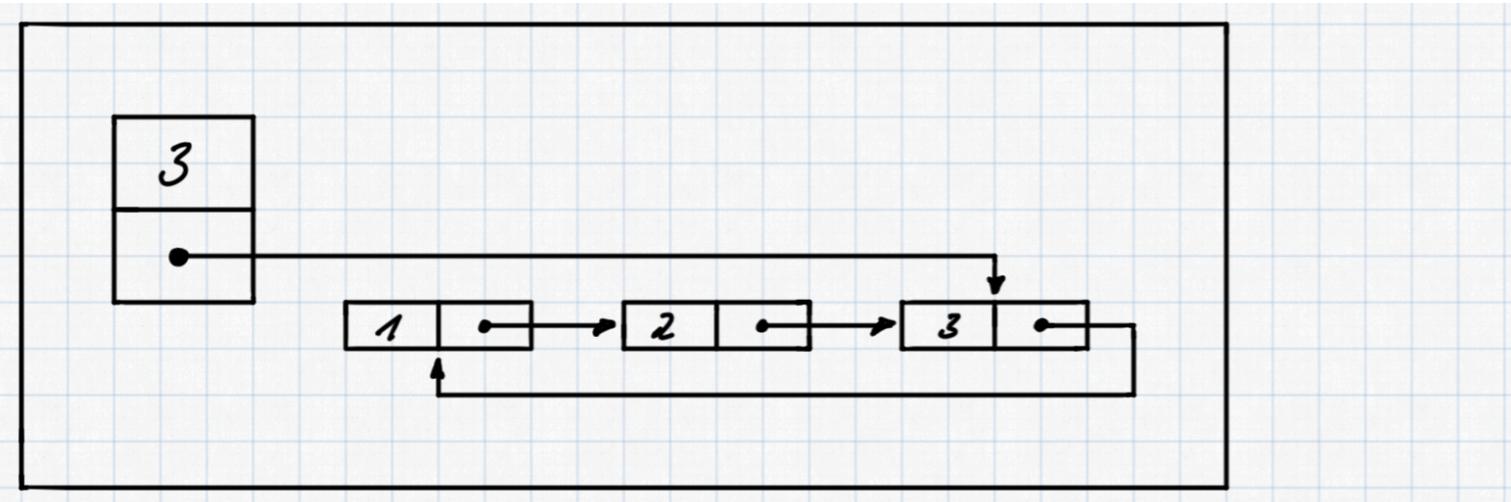


## Schrittfolge Einfügen:

- #1. Ein neuer Knoten wird erstellt.
- #2. Dem neuen Knoten wird ein Inhalt gegeben.
- #3. Der aktuelle Endzeiger wird auf den Knoten gebogen
- #4. Der Verweis "nächster" des Endzeigers wird auf den Knoten gebogen (zeigt also in diesem Fall auf sich selbst).
- #5. Die Länge der Schlange wird um 1 erhöht.

# EINFACH VERKETTETE RINGLISTE

## gefüllte Schlange:

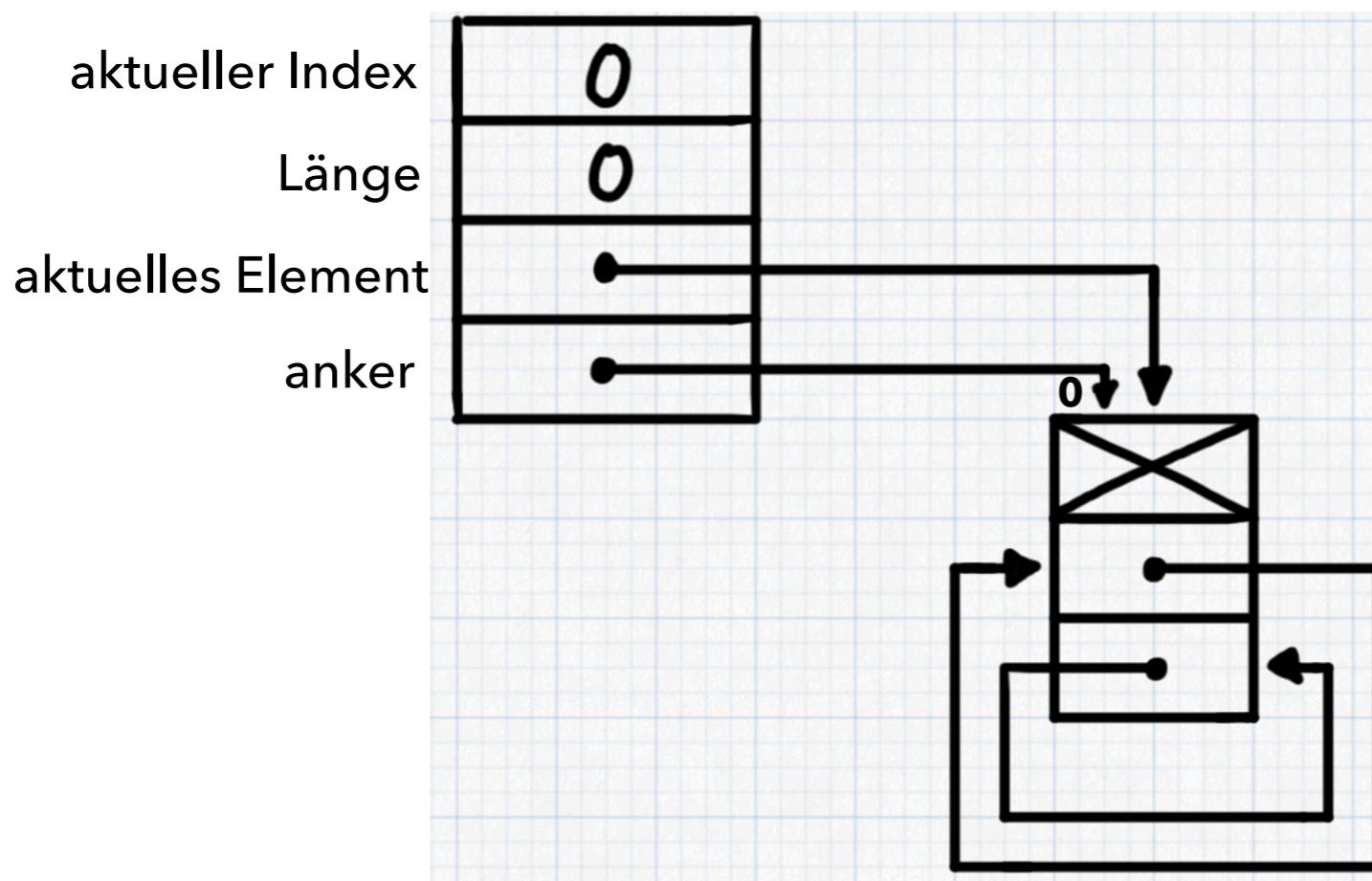


## Schrittfolge Einfügen:

- #1. Ein neuer Knoten wird erstellt.
- #2. Dem neuen Knoten wird ein Inhalt gegeben.
- #3. Der Verweis "nächster" des neuen Knotens wird auf das nächste Element des Endzeigers gebogen.
- #4. Der Verweis "nächster" des Endzeigers wird auf den Knoten gebogen.
- #5. Der Endzeiger wird auf den neuen Knoten gerichtet, da dieser das neue letzte Element in der Schlange ist.
- #6. Die Länge der Schlange wird um 1 erhöht.

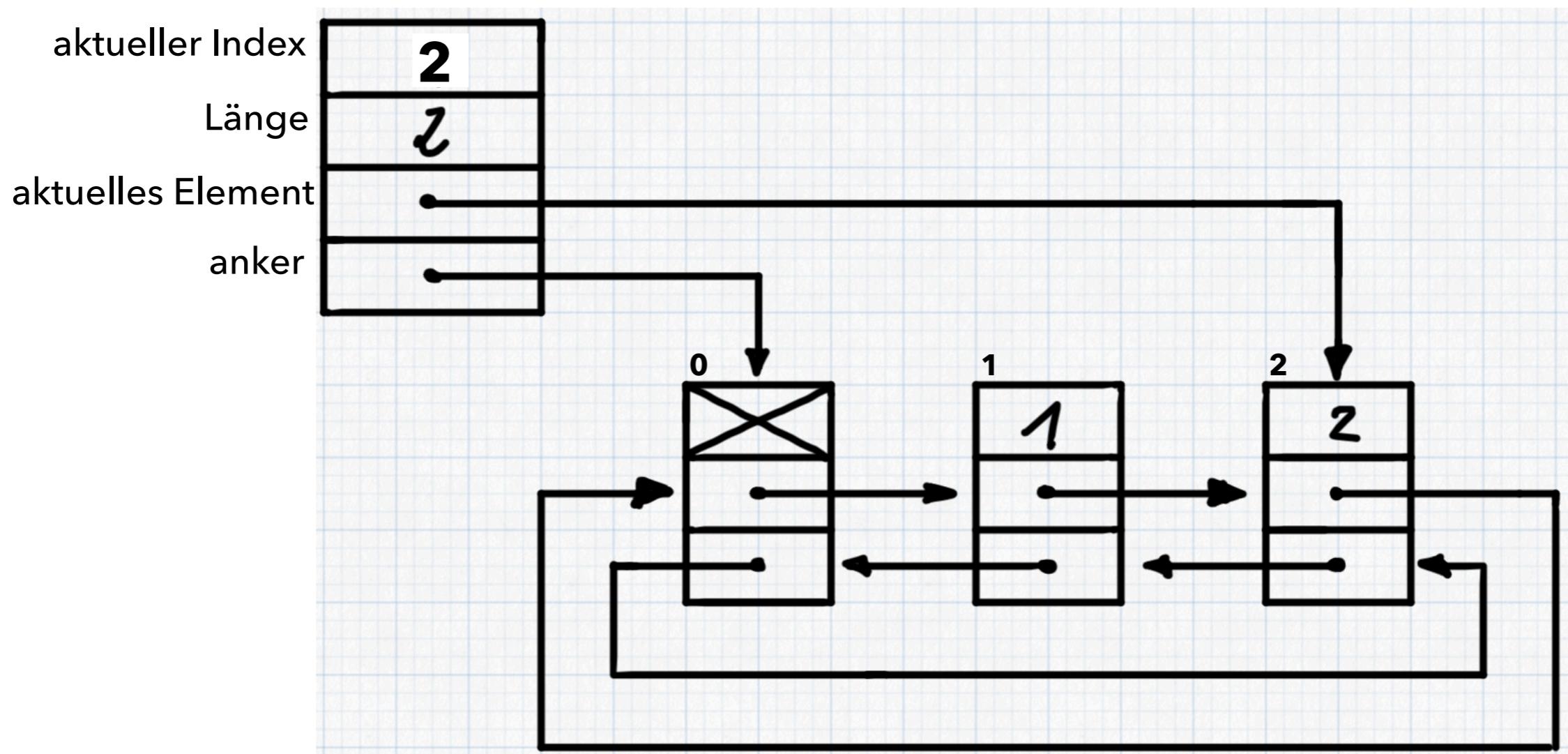
# DOPPELT VERKETTETE RINGLISTE MIT ANKERELEMENT

leere Schlange:



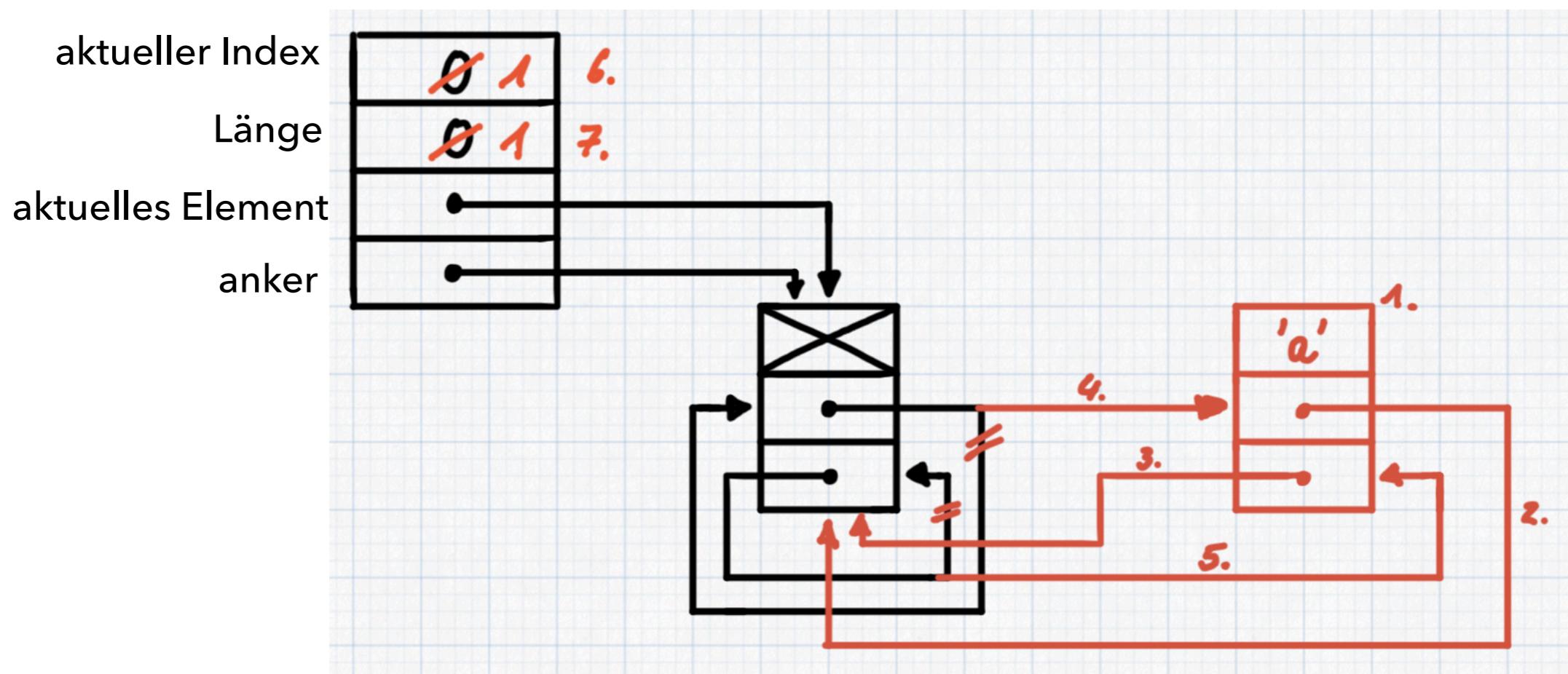
# DOPPELT VERKETTETE RINGLISTE MIT ANKERELEMENT

leere Schlange:



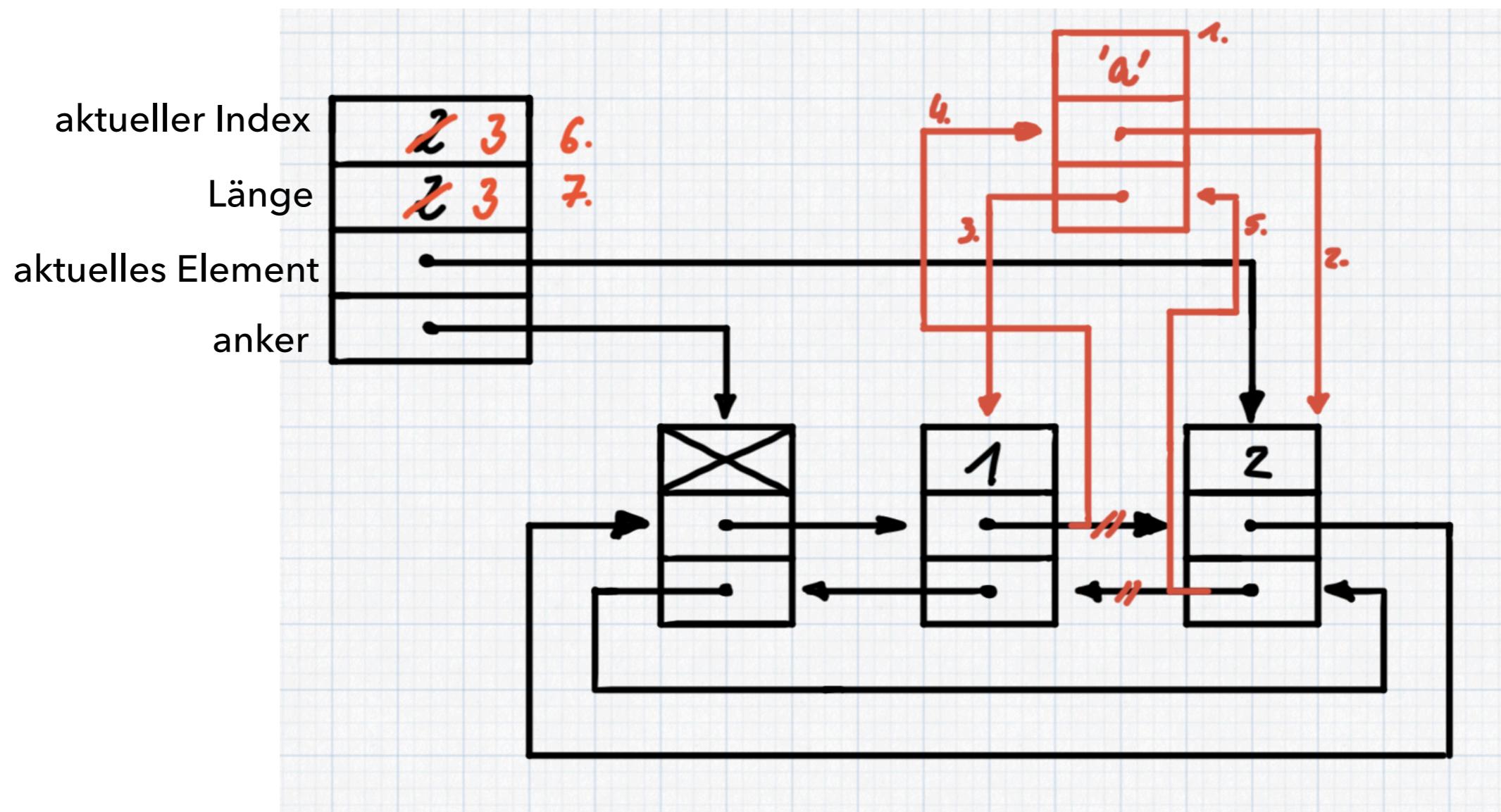
# DOPPELT VERKETTETE RINGLISTE MIT ANKERELEMENT

## Einfügen in eine leere Schlange:



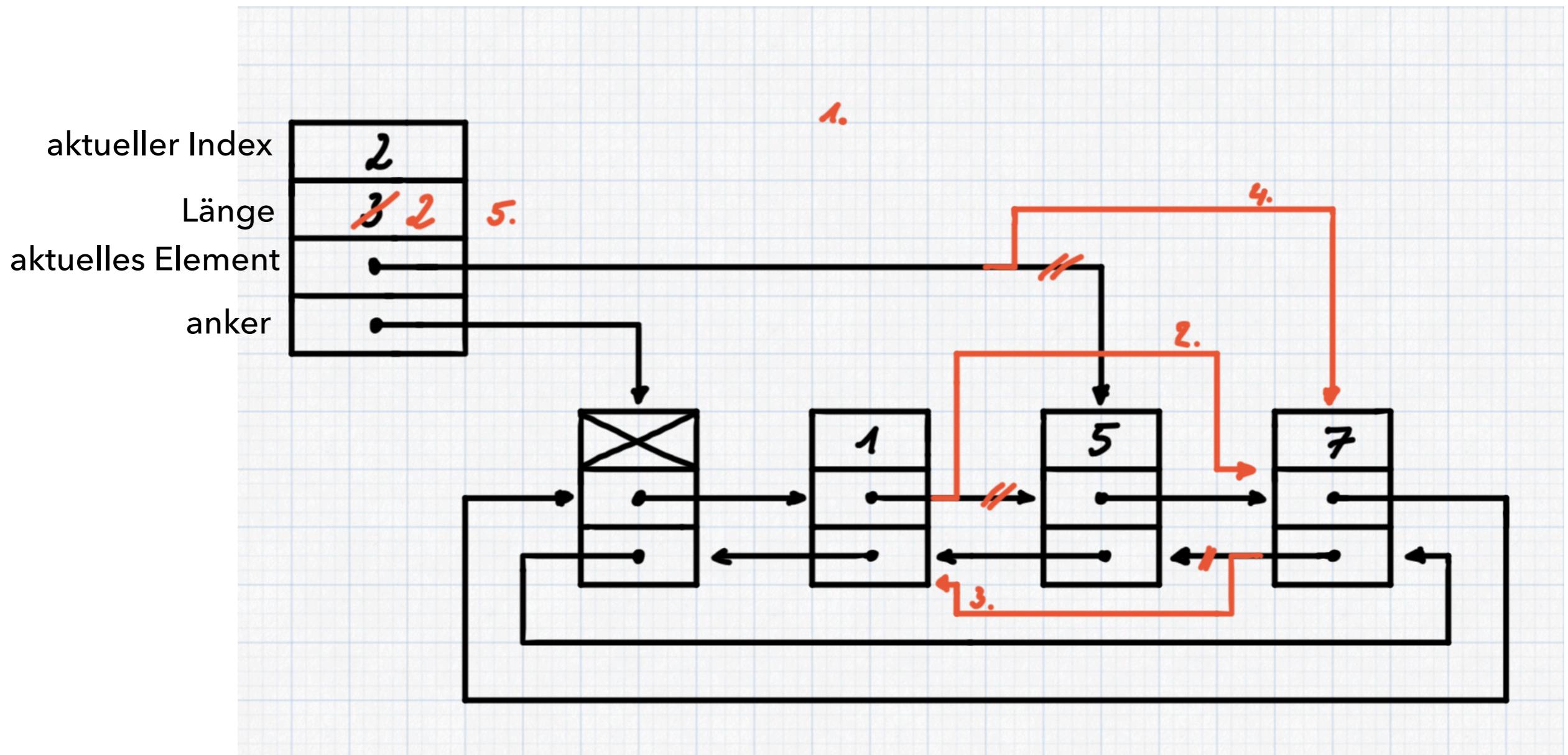
# DOPPELT VERKETTETE RINGLISTE MIT ANKERELEMENT

Einfügen in eine nicht leere Schlange:



# DOPPELT VERKETTETE RINGLISTE MIT ANKERELEMENT

# Löschvorgang:



# STAPEL (STACK)

## ► Grundidee:

Ein Stapel oder auch Kellerspeicher (engl. stack) repräsentiert ein Behälter -Datenobjekt, in das andere Datenobjekte nur von „oben“ eingefügt (eingekellert) und nur von „oben“ wieder entnommen (ausgekellert) werden können, d.h., das jeweils zuletzt eingefügte Datenobjekt wird als erstes wieder entnommen.

## ► LiFo - Prinzip -> „Last in First out“

► Es gilt wieder: Benötigt ein Algorithmus einen Stack, so kann er diesen nutzen, ohne dass er wissen muss, wie die konkrete Datenstruktur, welche diesen Kellerspeicher realisiert, aufgebaut ist (information hiding).

## ► Operationen, die Stacks ausführen können:

- eine neuen leeren Stack erzeugen = Stack()
  - Testen, ob der Stack leer ist = istLeer()
  - ein Objekt oben auf dem Stack ablegen = push()
  - Insofern der Stack nicht leer ist, wird das oberste Element vom Stack entfernt = pop()
  - Insofern der Stack nicht leer ist, wird das oberste Element zurückgegeben, ohne es aus dem Stack zu entfernen = top()
  - der Stack wird mithilfe einer Anzeigefunktion ausgegeben = anzeige()
- Implementieren und Testprogramm schreiben