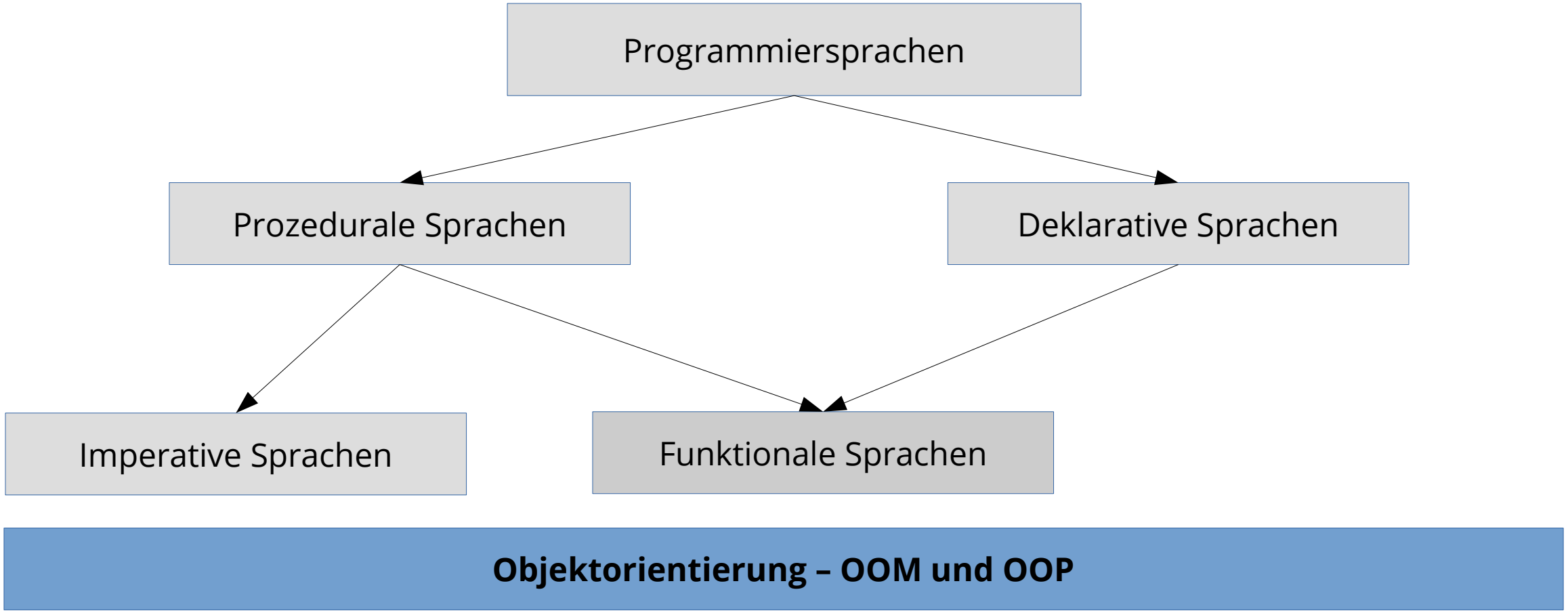
TECHNISCHE UNIVERSITÄT DRESDEN

Dr. Thiemo Leonhardt
Professur für Didaktik der Informatik

Programmierparadigmen

Dynamische Datenstrukturen

Klassifikation von Programmiersprachen



Listen und Listen-Funktionen

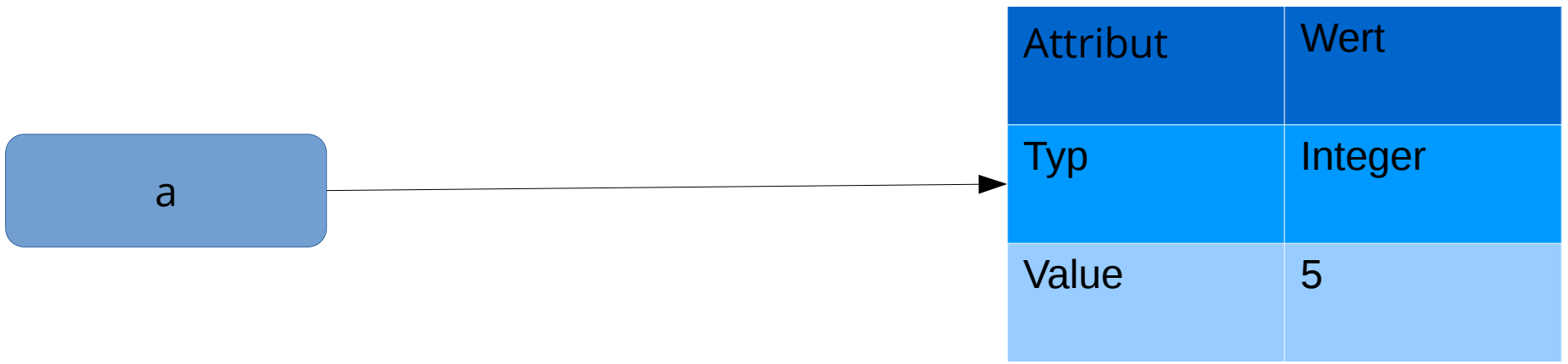
```
liste = [1,2,3,4]

len(liste)
>> 4

liste[0]
>>1
```

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Wie wird eine Liste implementiert?

Alles in Python ist ein Objekt!

Benutzen der Objektorientierung (OOP), um eine Liste zu erzeugen.



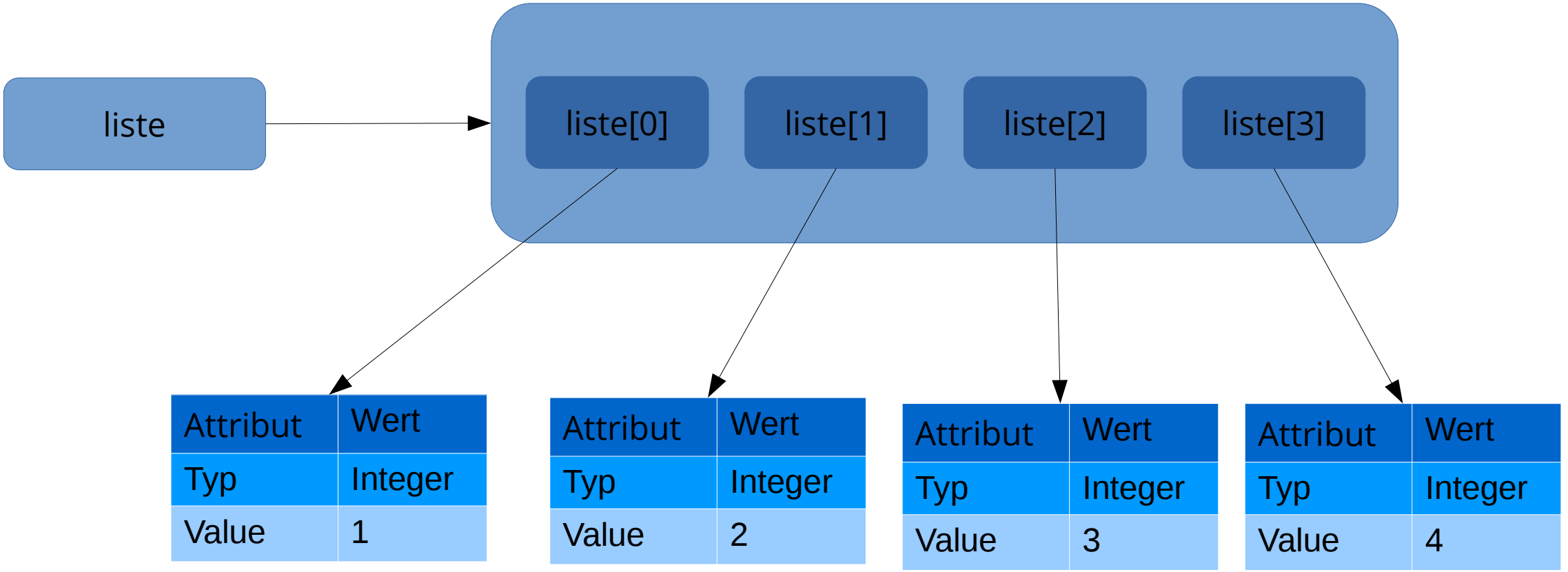
a = 5

Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]

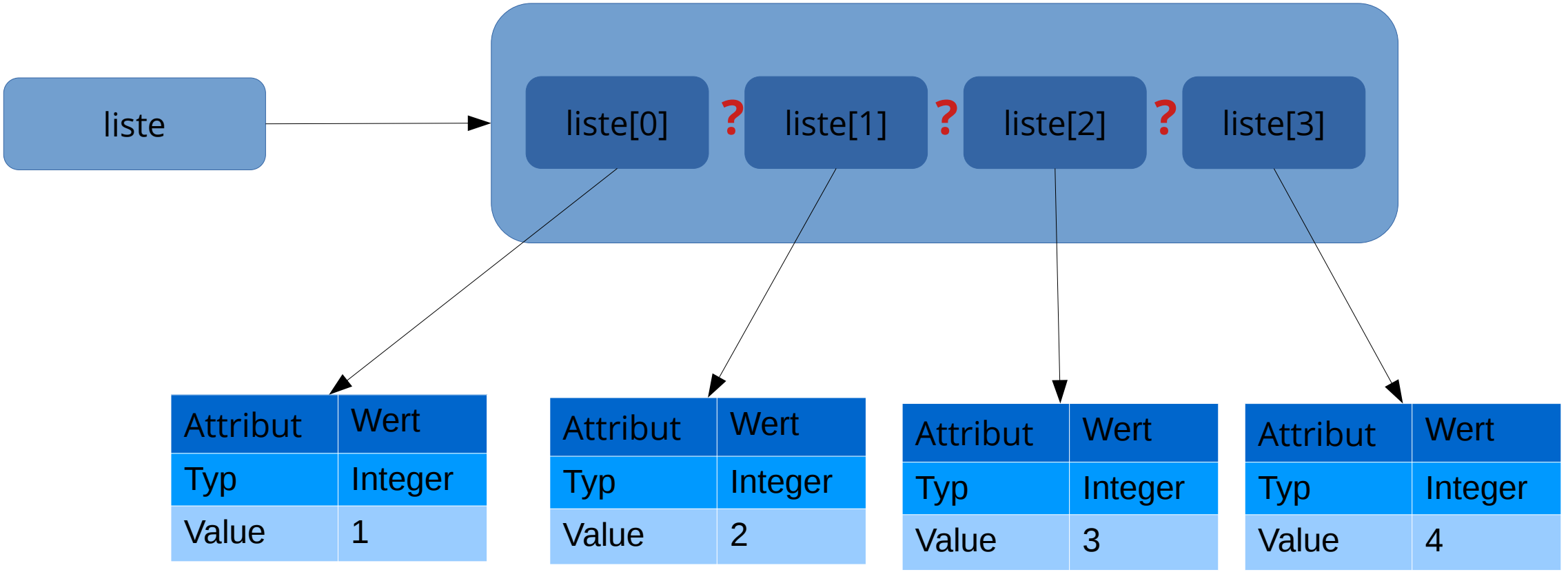
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



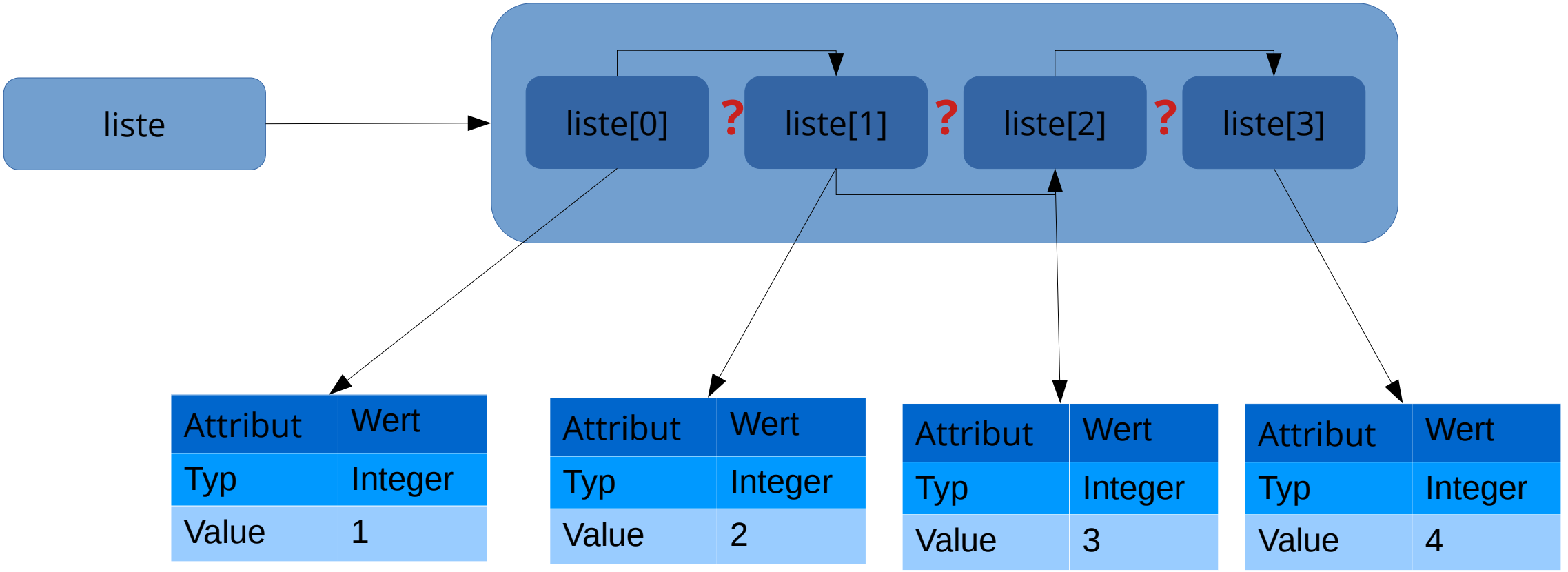
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



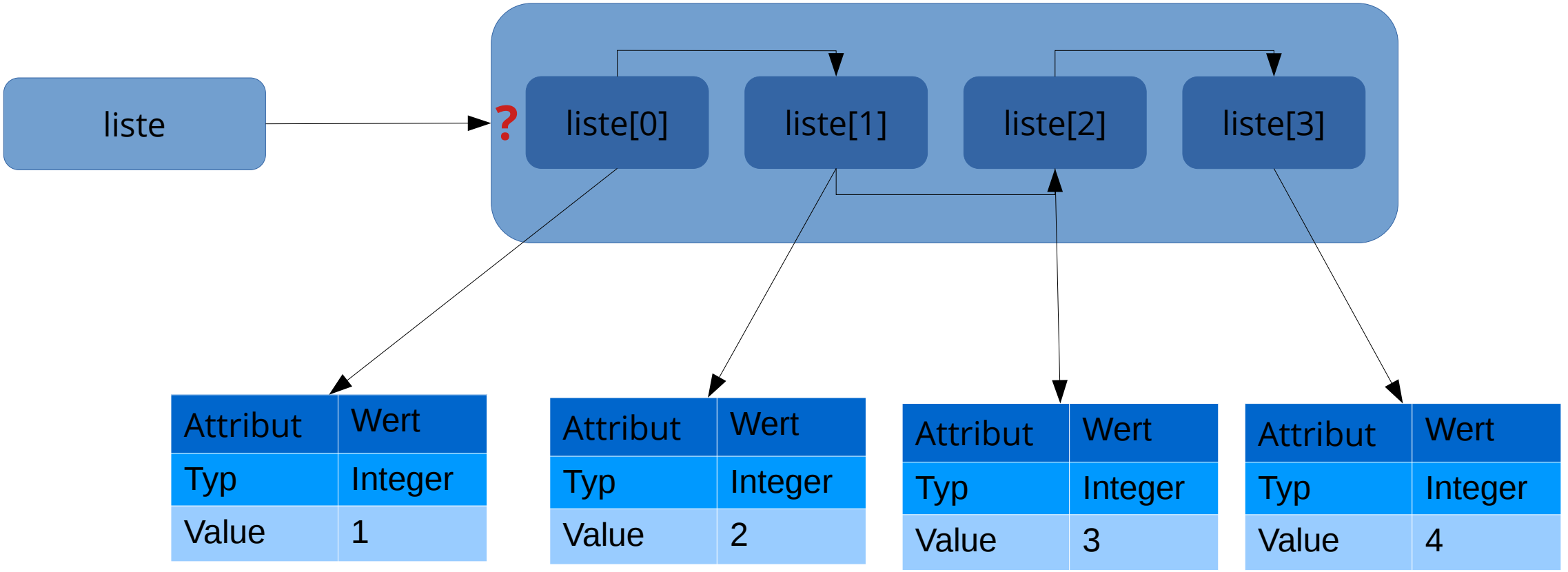
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



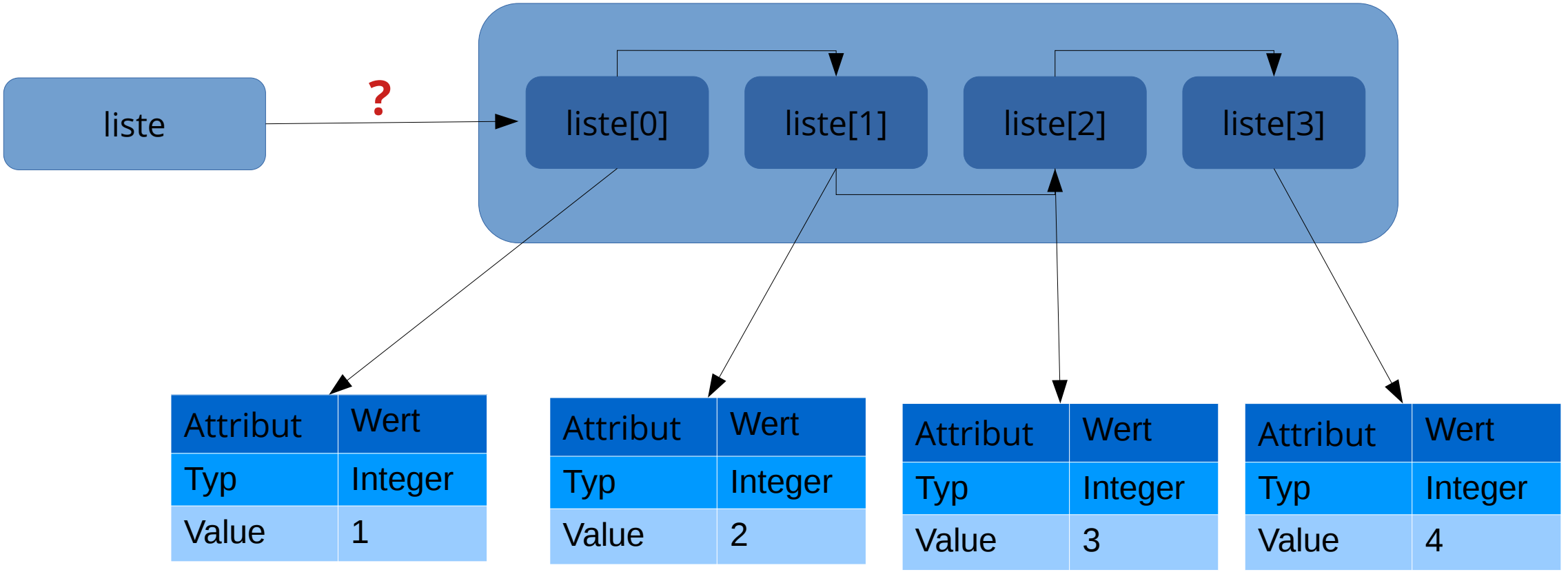
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



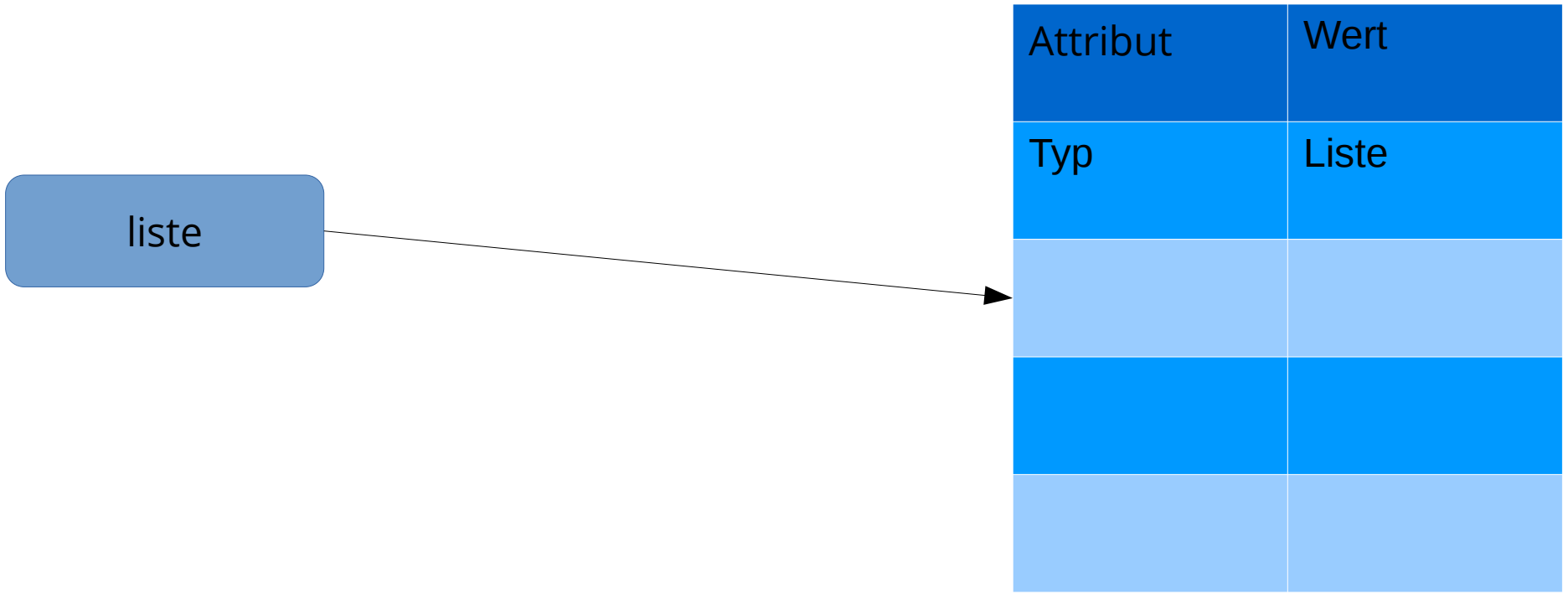
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



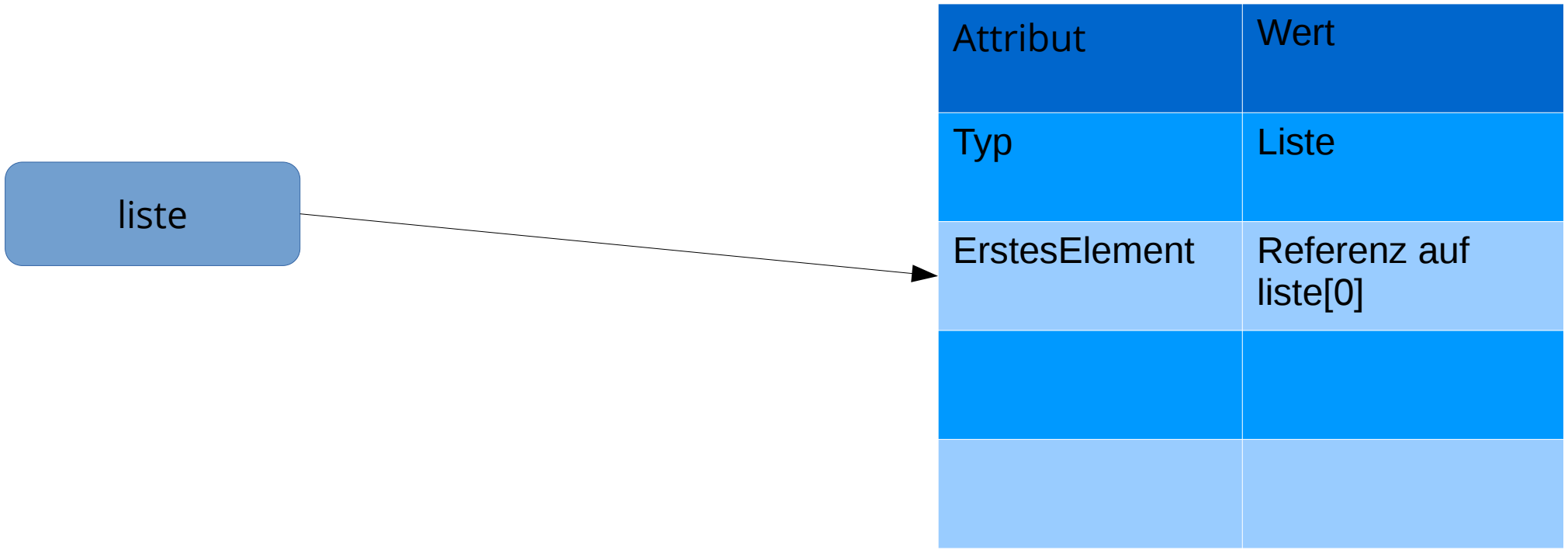
Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]



Wie ist eine Liste im Speicher realisiert?

liste = [1,2,3,4]

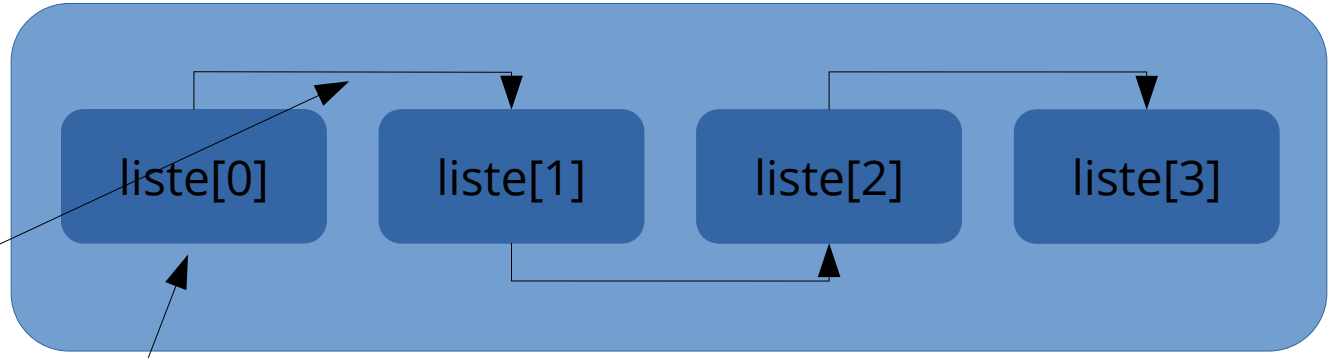


Liste in Python – Element einfügen durch append

```
class Liste:

    def __init__(self):
        self.erstesElement = None
```

- Klassendefinition
  - erstesElement soll die Referenz auf das erste Element der Liste werden
- Methoden
  - Init-Methode
    - Setzt erstes Element auf None

Implementation einer Liste in Python - Listenelemente



Jedes Element der Liste zeigt auf/ kennt sein Nachfolgeelement.

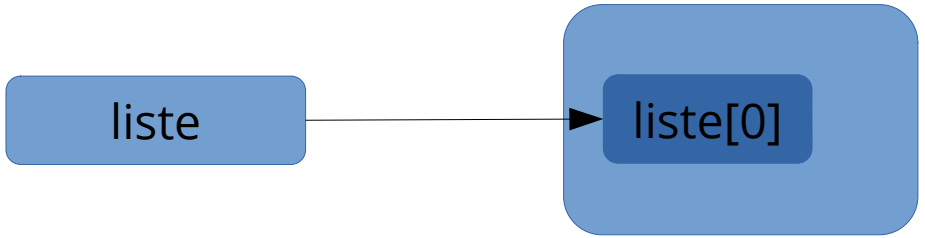
Jedes Element der Liste hat außerdem noch eine Referenz auf einen Wert gespeichert.

Implementation eines Listenelements in Python

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

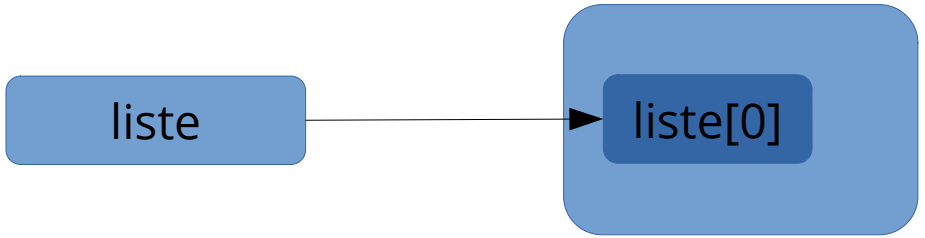
- Klassendefinition
  - nextElement soll eine Referenz auf das nächste Listenelement sein
  - value ist eine Referenz auf den Wert des Listenelements
- Init-Methode
  - Setzt die Referenz value der Klasse Listenelement auf den den übergebenen Parameter value.
  - Setzt die Referenz nextElement auf None, da noch kein Nachfolger bestimmt wurde.

Hinzufügen des ersten Listenelements zur Liste



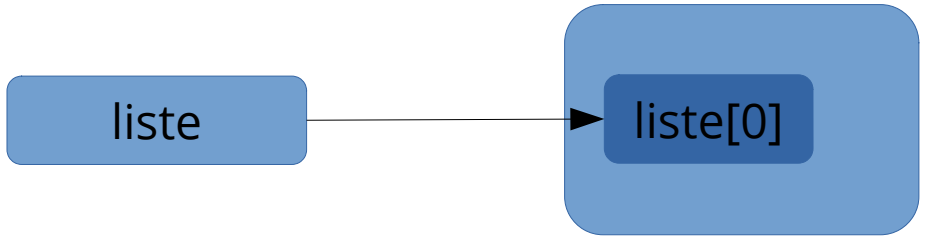

```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self):
        pass
```

Hinzufügen des ersten Listenelements zur Liste




```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

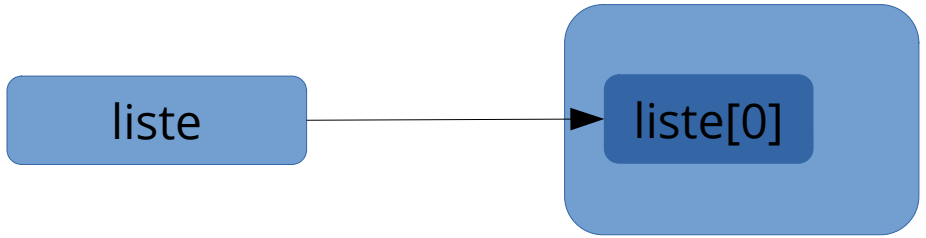
Hinzufügen des ersten Listenelements zur Liste




```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

Problem?

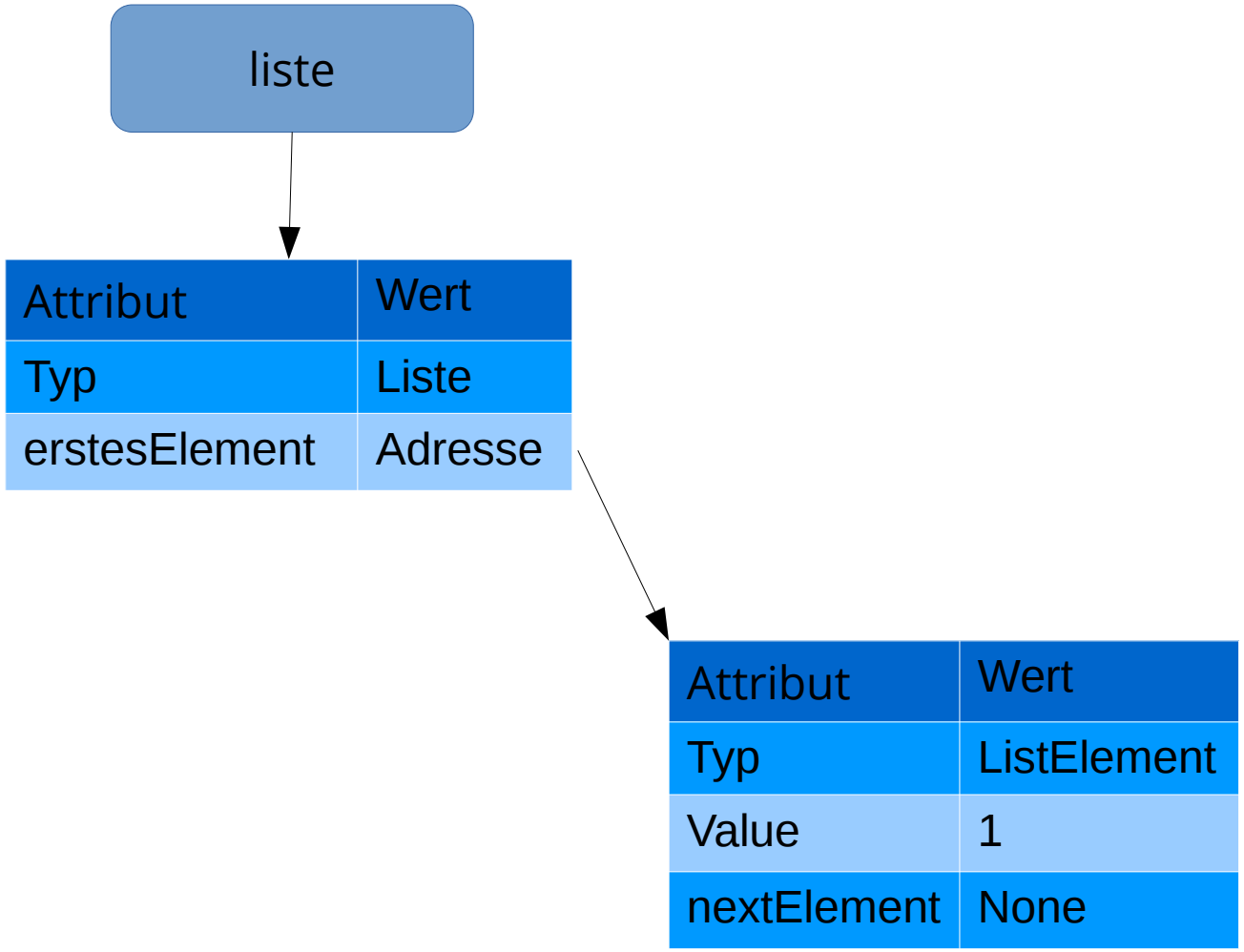
Hinzufügen des ersten Listenelements zur Liste



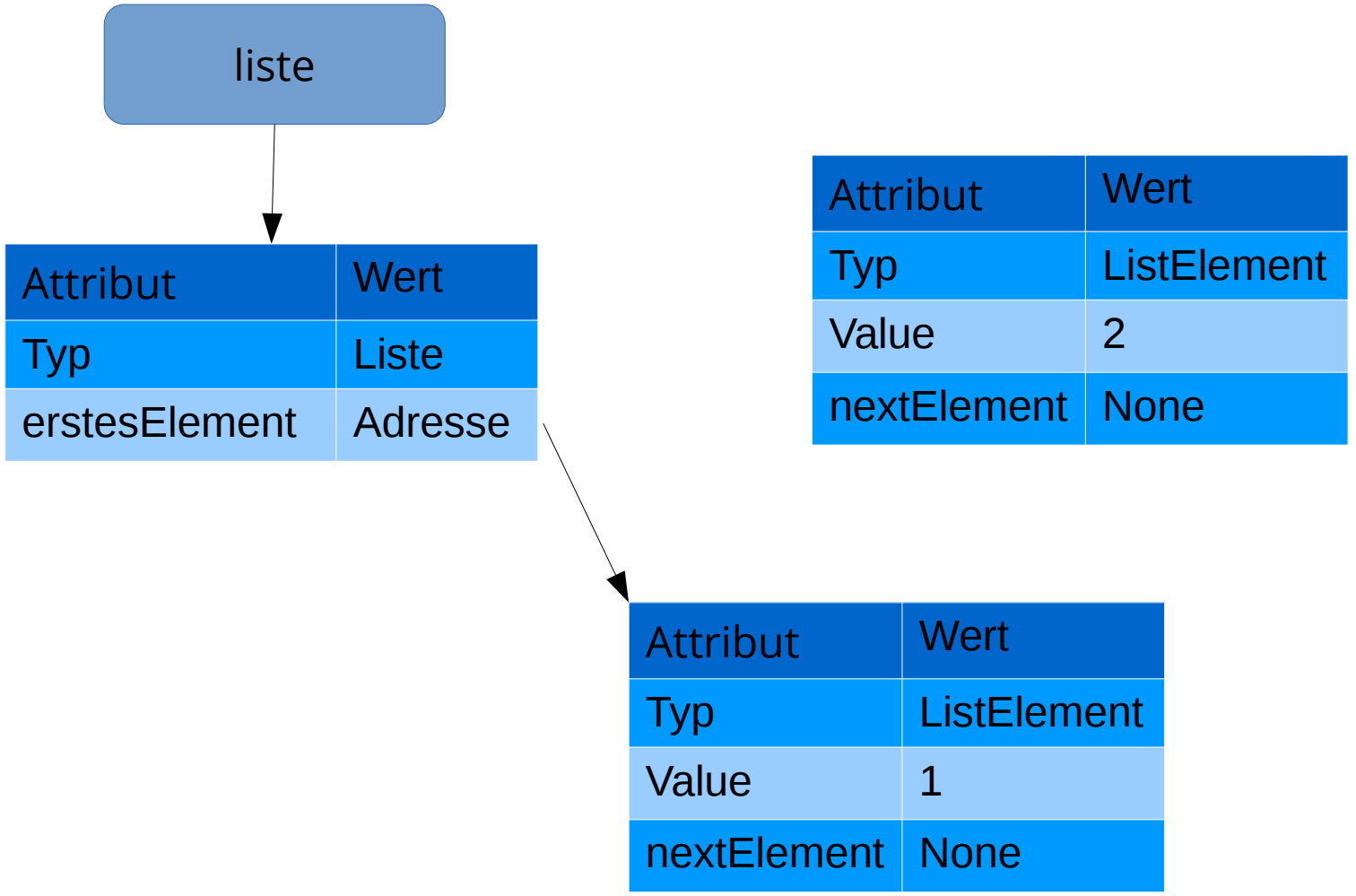

```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

Beim Hinzufügen von neuen Elementen wird immer das erste Element überschrieben.

Wie sieht die Implementation der Liste im Moment aus?

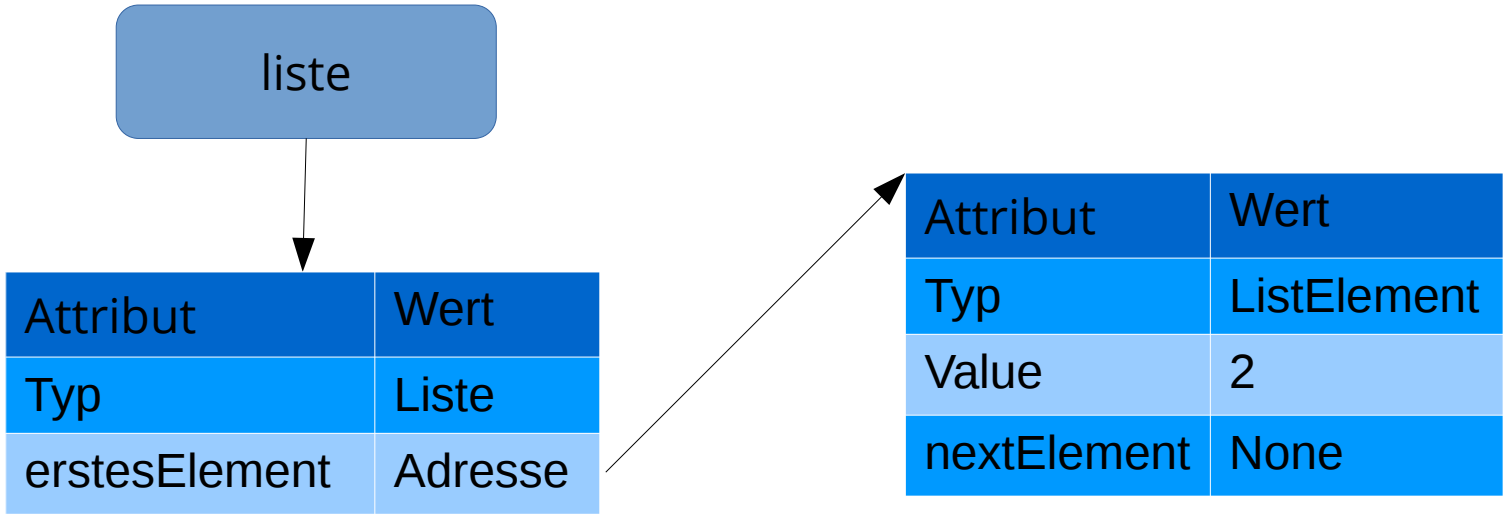


Wie sieht die Implementation der Liste im Moment aus?



liste.append(element)

Wie sieht die Implementation der Liste im Moment aus?

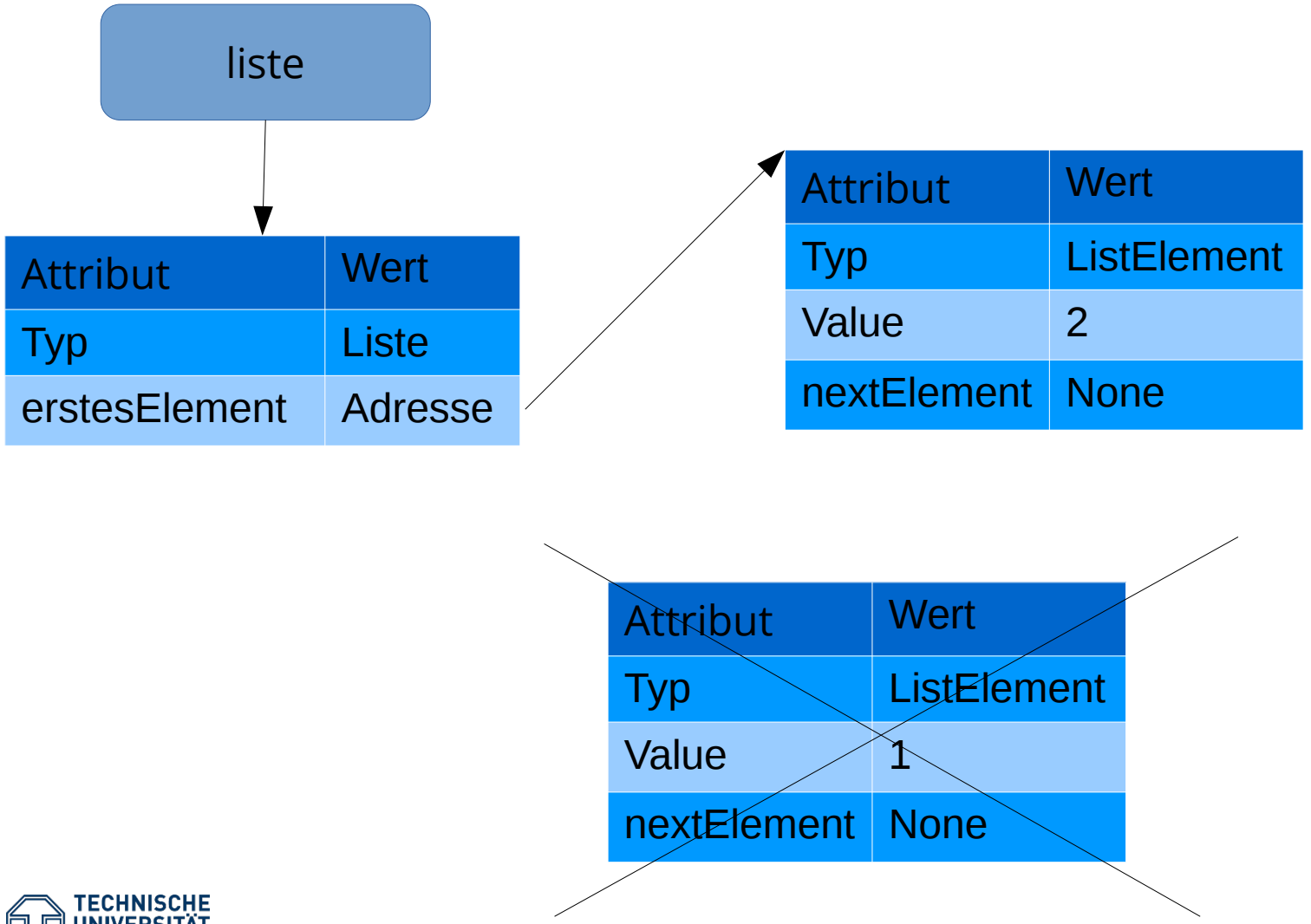


liste.append(element)

```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

Attribut	Wert
Typ	ListElement
Value	1
nextElement	None

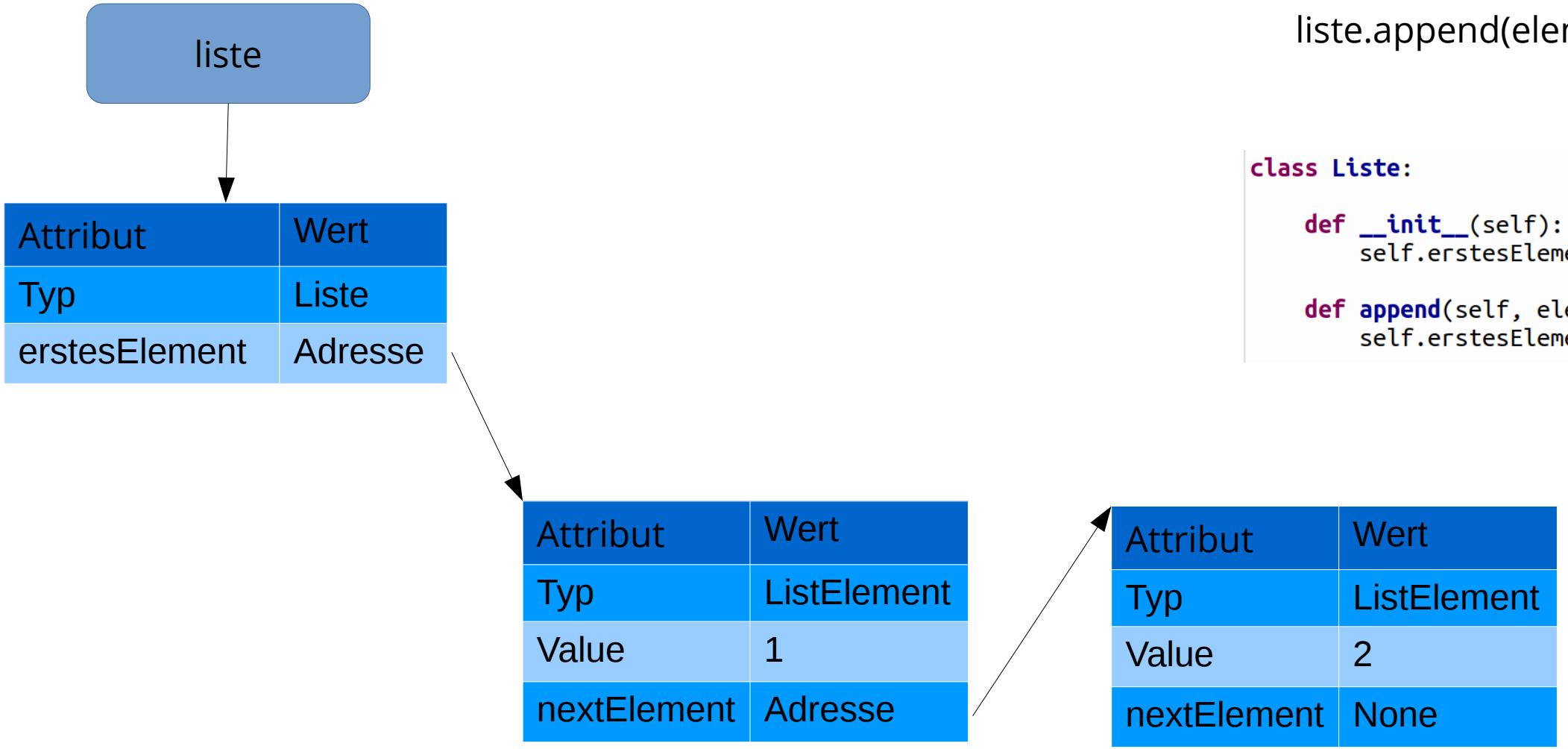
Wie sieht die Implementation der Liste im Moment aus?



liste.append(element)

```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

Wie sieht die Implementation der Liste im Moment aus?



liste.append(element)

```
class Liste:

    def __init__(self):
        self.erstesElement = None

    def append(self, element):
        self.erstesElement = element
```

Anpassen der append-Methode

```
def append(self, element):
```

Erstes Element nur überschreiben, wenn noch keins existiert.

Anpassen der append-Methode

```
def append(self, element):
    if self.erstesElement == None:
```

Erstes Element nur überschreiben, wenn noch keins existiert.

Anpassen der append-Methode

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
```

Erstes Element nur überschreiben, wenn noch keins existiert.

Anpassen der append-Methode

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
```

Im anderen Fall, soll das Element an das Ende der Liste angehängt werden.

Anpassen der append-Methode

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
        lastElement =
```

Ermitteln des letzten Elements

Anpassen der append-Methode

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
        lastElement = self._findLastElement(self.erstesElement)
```

Nutzung einer Hilfsmethode

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
```

Wie könnten wir dies lösen?
Iterativ oder Rekursiv?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
```

Rekursiver Ansatz:
Was ist der Basisfall? Was ist die Abbruchbedingung?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
```

Rekursiver Ansatz:
Was ist der Basisfall? Was ist die Abbruchbedingung?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
        return element
```

Rekursiver Ansatz:
Was ist der Basisfall? Was ist die Abbruchbedingung?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
        return element
```

Rekursiver Ansatz:
Was passiert, wenn *element* nicht existiert? Kann das passieren?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
        return element
```

Rekursiver Ansatz:
Wie beschreiben wir den rekursiven Fall?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
        return element
    else:
```

Rekursiver Ansatz:
Wie beschreiben wir den rekursiven Fall?

Einschub Hilfsfunktion – Finde das letzte Element der Liste

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def _findLastElement(self, element):
    if element.nextElement == None:
        return element
    else:
        return self._findLastElement(element.nextElement)
```

Rekursiver Ansatz:
Wie beschreiben wir den rekursiven Fall?

Anpassen der append-Methode

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
        lastElement = self._findLastElement(self.erstesElement)
```

Hilfsfunktion ermittelt das letzte Element der Liste

Anpassen der append-Methode

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
        lastElement = self._findLastElement(self.erstesElement)
        lastElement.nextElement =
```

Das letzte Element der Liste hat keinen Nachfolger. Dies muss nun geändert werden.

Anpassen der append-Methode

```
class ListElement:

    def __init__(self, value):
        self.value = value
        self.nextElement = None
```

```
def append(self, element):
    if self.erstesElement == None:
        self.erstesElement = element
    else:
        lastElement = self._findLastElement(self.erstesElement)
        lastElement.nextElement = element
```

Damit ist die append-Methode vollständig.