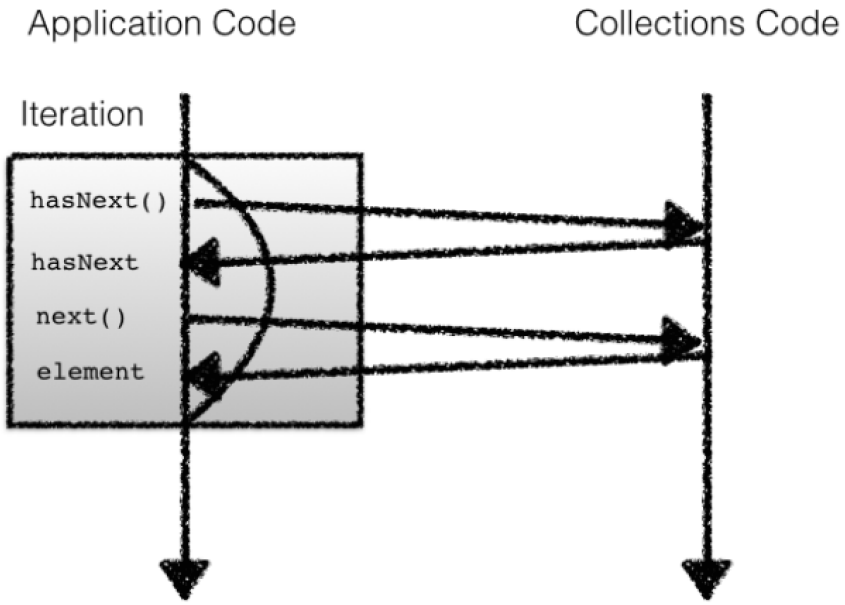
Fachhochschule
Nordwestschweiz

Streams

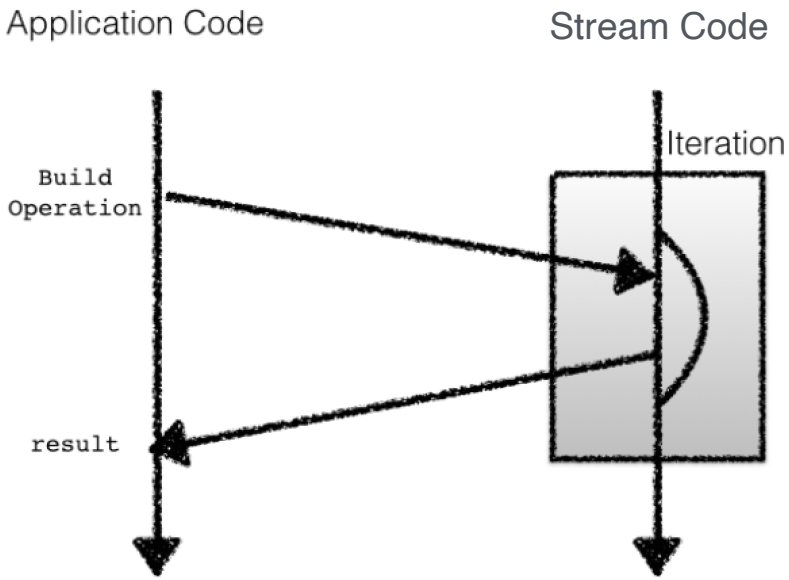
External Iteration

```
int count = 0;
for (Artist artist : artists) {
      if (artist.isFrom("London")) {
            count++;
      }
```



Internal Iteration

```
artists.stream()
        .filter(artist -> artist.isFrom("London"))
        .count();
```



Collections und Streams

Die **Collection Klassen** sind dazu da Daten in einer bestimmten Struktur zu abzulegen. Wichtige Eigenschaften sind das Ablegen und der Zugriff von Elementen in dieser Datenstruktur.
storing and accessing data

Streams sind dazu da die Daten zu verarbeiten, zu manipulieren.
describing computations on data

Imperative Datenverarbeitung (Wie)

lat: anordnen, befehlen

```
List<Country> smallCountries = new ArrayList<>();
for(Country c : countries){
    if(c.getArea() < 100_000) {
        smallCountries.add(c);
    }
}

Collections.sort(smallCountries, new Comparator<Country>() {
    @Override
    public int compare(Country o1, Country o2) {
        return o1.getName().compareTo(o2.getName());
    }
});

List<String> names = new ArrayList<>();
for(Country c : smallCountries) {
    names.add(c.getName());
}
```

Filtern

Sortieren

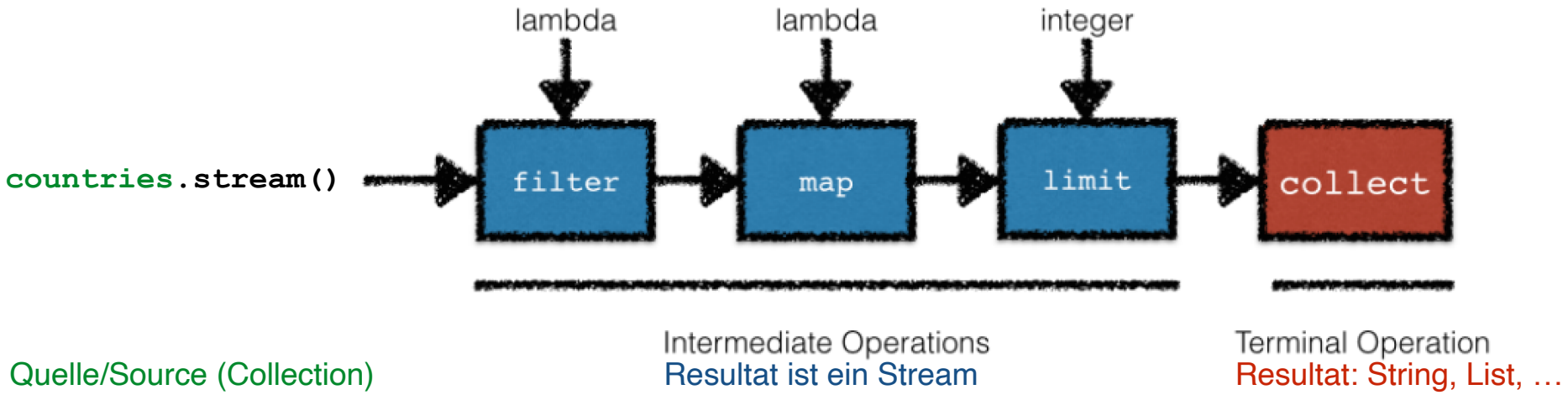
Namen extrahieren

Deklarative Datenverarbeitung (Was)

lat: erklärend

```
List<String> names = countries.stream()
        .filter(c -> c.getArea() < 100_000)
        .sorted(Comparator.comparingDouble(Country::getArea))
        .map(Country::getName)
        .collect(Collectors.toList());
```

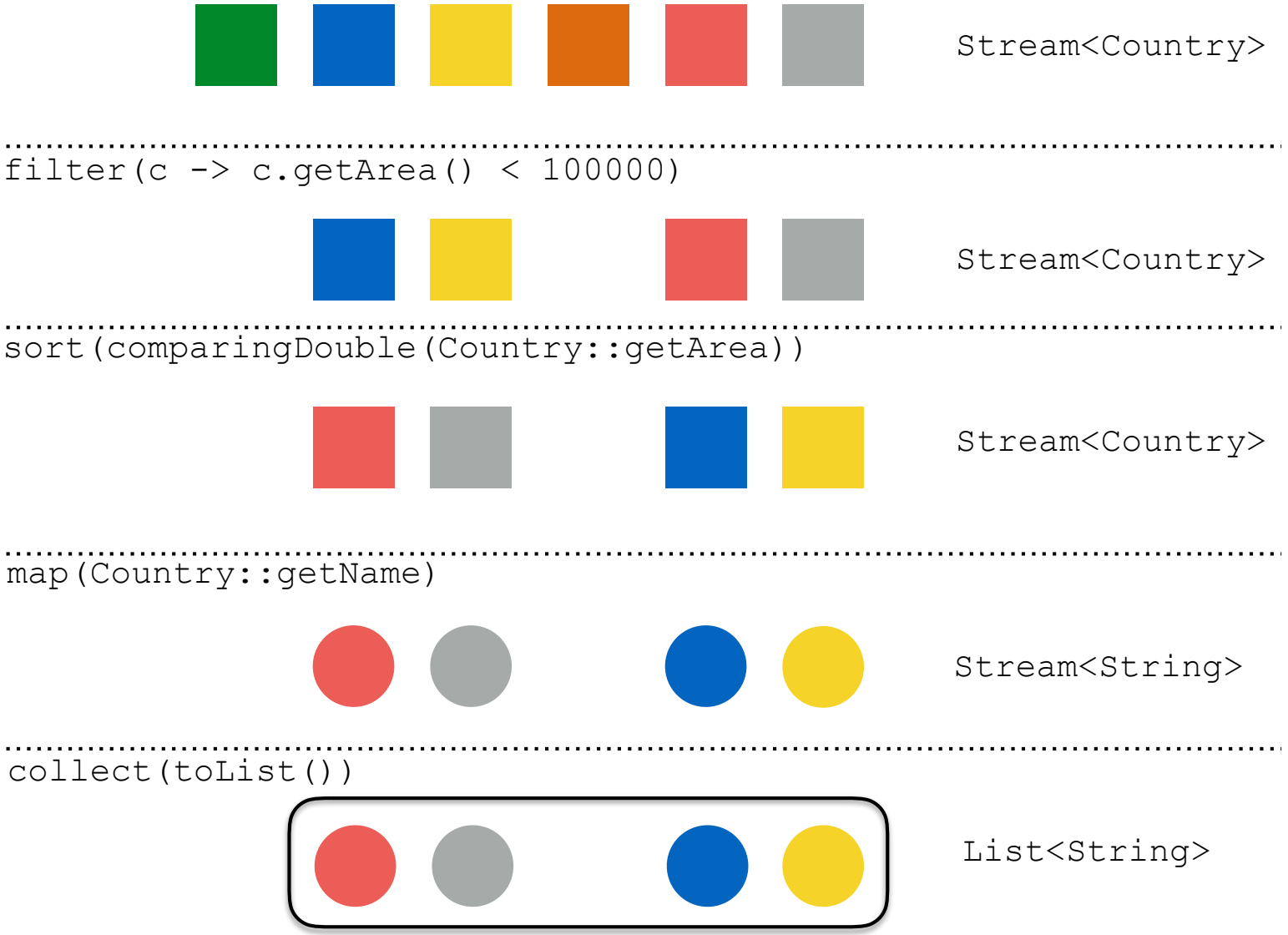
Intermediate und Terminal Operations



Wie kommt man an einen Stream?

- `Stream<SomeType> stream = myCollection.stream();`
- `Stream<String> stream = Stream.of("Java", "8", "Stream");`
- ```
  int[] numbers = {2, 3, 5, 7, 11, 13};
  int sum = Arrays.stream(numbers).sum();
  ```
- ```
  Stream.iterate(0, n -> n + 2)
    .limit(10)
    .forEach(System.out::println);
  ```
- ```
  Stream.generate(Math::random)
    .limit(5)
    .forEach(System.out::println);
  ```
  ```

filter, sort, map, collect



Arbeitsblatt (1)

Die Fibonacci Folge hat die Form: 1, 1, 2, 3, 5, 8, 13, 21, …

$$f_n = f_{n-1} + f_{n-2} \quad \text{für } n > 2$$
$$f_1 = f_2 = 1$$

Generieren Sie mittels der `Stream.iterate` Methode die ersten 30 Fibonacci Zahlen. In der JavaDoc ist ersichtlich, dass der zweite Parameter eine einwertige Funktion (nur ein Parameter) ist.

```
static <T> Stream<T>          iterate(T seed, UnaryOperator<T> f)
```

Tipp: Verwenden Sie darum die Klasse Tuple und fassen Sie zwei Werte zusammen. Der Typ T ist demzufolge ein Tuple und `seed` ist `new Tuple(0, 1)`.

Nur einmal durchlaufbar

```
 7 public class TraverseOnlyOnce {
 8
 9     public static void main(String[] args) {
10
11         List<String> languages = Arrays.asList("Java", "Scale", "Lisp", "Scheme");
12
13         Stream<String> stream = languages.stream();
14
15         long n = stream.count();  // n = 4
16         System.out.println(n);
17         stream.forEach(System.out::println);
18     }
19 }
```


java.lang.IllegalStateException: stream has already been operated upon or closed

Optimiertes Durchlaufen

```
 9          Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
10              .filter(n -> {
11                  System.out.println("filtering: " + n);
12                  return n % 2 == 0; })
13              .map(n -> {
14                  System.out.println("mapping: " + n);
15                  return n * n; })
16              .limit(3)
17              .forEach(System.out::println);
18
19      }
```

```
filtering: 1
filtering: 2
mapping: 2
4
filtering: 3
filtering: 4
mapping: 4
16
filtering: 5
filtering: 6
mapping: 6
36
```

Keine Auswirkungen auf die Quelle

```
10    public static void main(String[] args) {
11
12        List<String> collection = Arrays.asList("A", "B", "C");
13
14        List<String> filteredCollection = collection.stream()
15            .filter(s -> s.startsWith("A"))
16            .collect(toList());
17
18        print(filteredCollection);
19        print(collection);
20    }
21
22    private static void print(List<?> list) {
23        System.out.println("----------------------");
24        for(Object o : list) {
25            System.out.println(o);
26        }
27    }
```

```
----------------------
A
----------------------
A
B
C
```

Eager (Collection), Lazy (Stream)

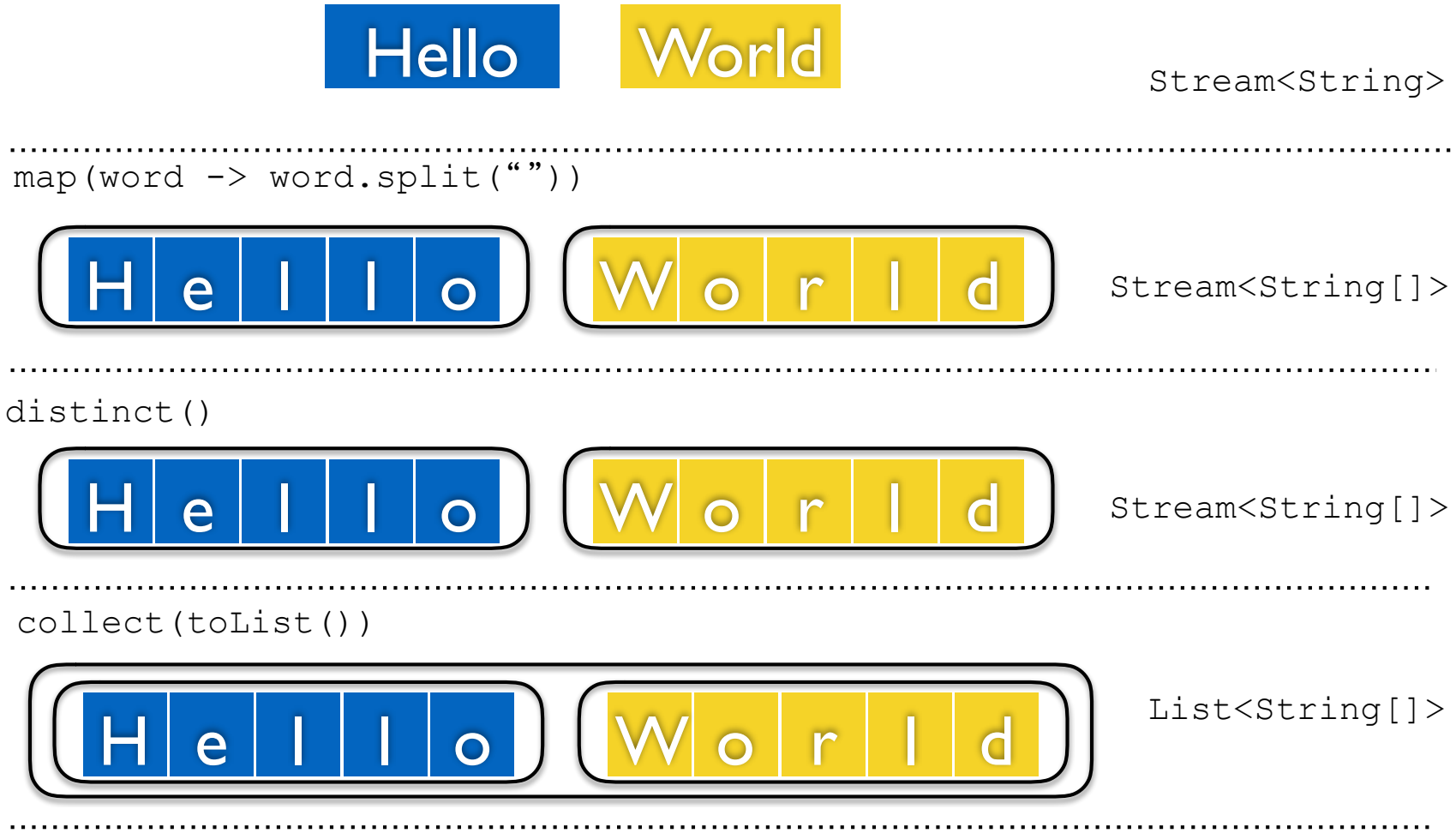
Collection werden mit einer initialen festen Grösse angelegt. Alle Elemente sind in Memory. Sie werden Eager alloziert, also ***on supply***.

Streams beginnen mit der Arbeit erst, wenn die Terminal Funktion aufgerufen wird. Sie liefern die Elemente ***on demand***, also Lazy.

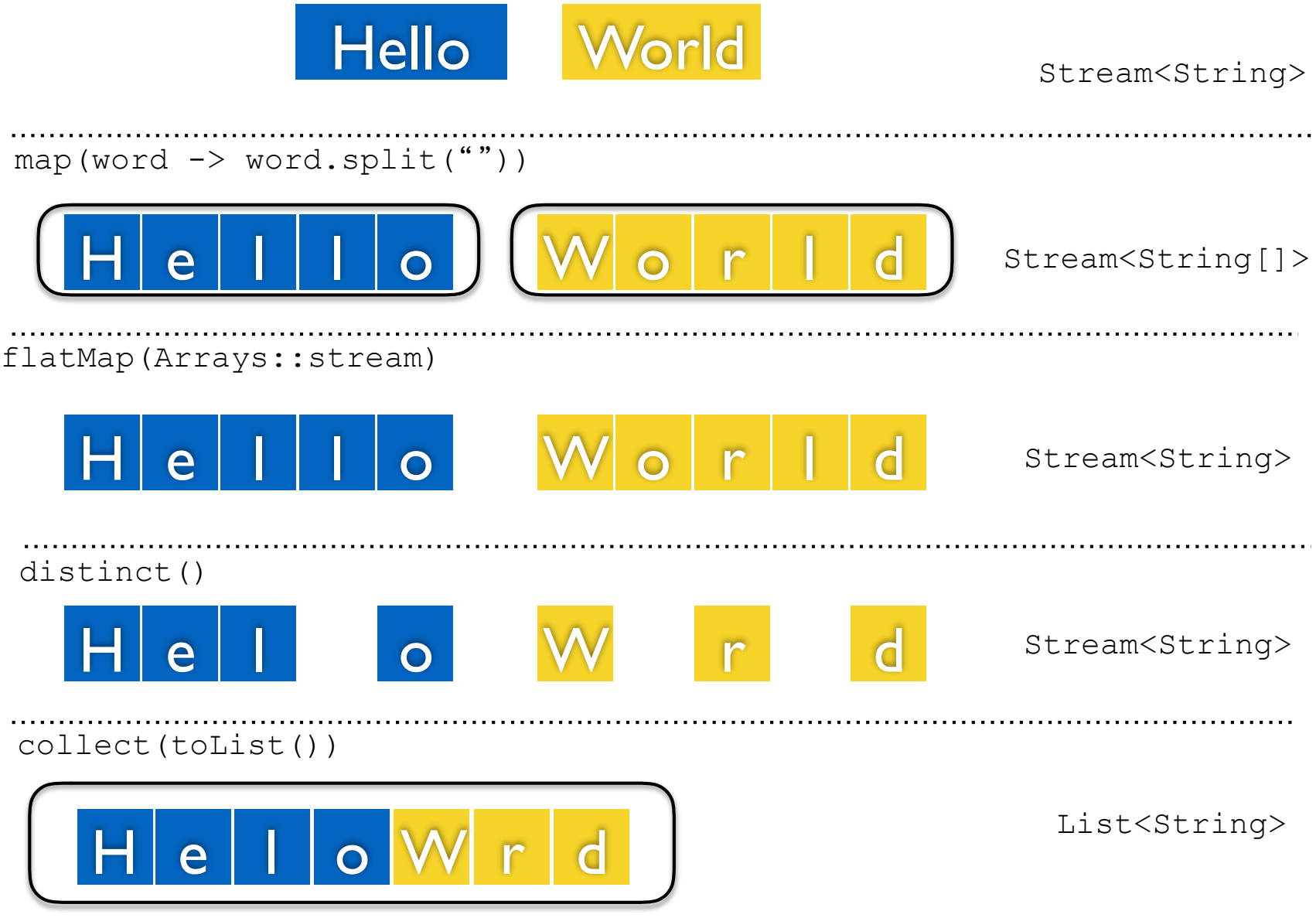
Mit Streams kann man also unendlich viele Elemente erzeugen. Die Terminal Funktion konsumiert diese Elemente nach und nach.

```
 5 public class EvenNumbers {
 6
 7     public static void main(String[] args) {
 8
 9         Stream
10             .iterate(0, n -> n + 2)
11             .limit(10)
12             .forEach(System.out::println);
13     }
14 }
```

Buchstaben zählen (1)



Buchstaben zählen (2)



FlatMap

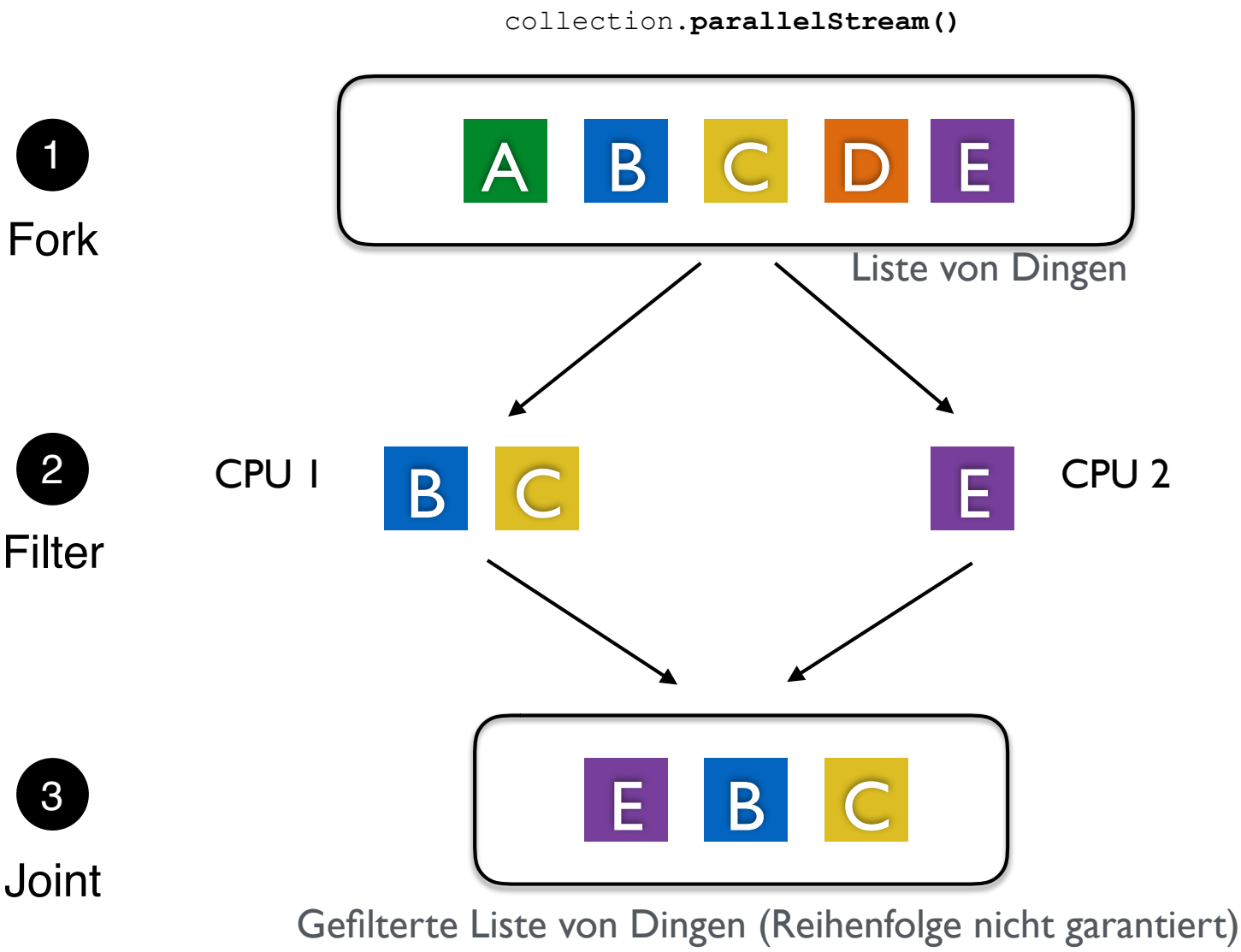
```
10          List<String> words = Arrays.asList("An",  "Analogy",  "with",  "Structured",  "Programming");
11
12          // Wir wollen hier eine Liste mit allen Buchstaben haben, die in words
13          // verwendet wurden. Die Liste soll keine Duplikate aufweisen.
14
15
16          long n = words.stream()
17              .map(word -> word.split(""))  // Gibt String[] pro Wort zurück
18              .distinct()
19              .count();
20
21          // Das sind fünf Listen mit den Buchstaben der jeweiligen Wörter.
22          System.out.println(n);
23
24
25          long m = words.stream()
26              .map(word -> word.split(""))  // Liste von Arrays von Strings (warum nicht Methoden-Referenz?)
27              .flatMap(Arrays::stream)      // INHALT der Arrays in ein Stream von Strings
28              .map(String::toLowerCase)     // 19 -> 18, wegen Aa
29              .distinct()                   // Keine Dubletten
30              .count();                     // Zählen
31
32          // Für die Wörter wurden 18 Buchstaben aus dem Alphabet verwendet.
33          System.out.println(m);
```

Group By

```
13          List<Employee> employees = new ArrayList<>();
14          employees.add(new Employee("A", "Q"));
15          employees.add(new Employee("B", "R"));
16          employees.add(new Employee("C", "S"));
17          employees.add(new Employee("D", "Q"));
18          employees.add(new Employee("E", "R"));
19
20          Map<String, List<Employee>> employeeByDepartments =
21                  employees
22                      .stream()
23                      .collect(groupingBy(employee -> employee.department)); // Key Function
24
25          Employee.print(employeeByDepartments);
26      }
```

```
Department Q
A
D
-------
Department R
B
E
-------
Department S
C
-------
```

Nebenläufige Streams



Arbeitsblätter 2-7

Öffnen Sie in Ihrer IDE eine Task View (alternativ können Sie auch nach TODO suchen).
Dort sehen Sie die Operationen, welche Sie implementieren müssen, damit der Test durchläuft.

Description	Resource	Path	Location	Type
TODO: AB02	TransactionLis...	/05_Streams/src/ch...	line 22	Java Task
TODO: AB03	TransactionLis...	/05_Streams/src/ch...	line 34	Java Task
TODO: AB04	TransactionLis...	/05_Streams/src/ch...	line 44	Java Task
TODO: AB05	TransactionLis...	/05_Streams/src/ch...	line 55	Java Task
TODO: AB06	TransactionLis...	/05_Streams/src/ch...	line 66	Java Task
TODO: AB07	TransactionLis...	/05_Streams/src/ch...	line 76	Java Task

Reflexionsfragen zur PL (1)

- Warum ist es effizienter Streams zu verwenden, statt eine Collection mit einem Iterator zu traversieren?
- Was ist gemeint mit dem Satz: "Streams follow the what not how principle"?
- Was sind die Unterschiede von Streams zu Collections?
- Was ist mit dem Begriff 'lazy' gemeint? Machen Sie ein Beispiel. Wie können Sie zeigen, dass Streams tatsächlich 'lazy' arbeiten? Tipp: Benutzen Sie peek…
- Wie sieht ein typischer Workflow aus, wenn mit Streams gearbeitet werden soll?

Reflexionsfragen zur PL (2)

- Wie können Streams erzeugt werden? Nennen Sie jeweils ein Beispiel.
- Wie funktioniert die Stream.iterate Funktion? Schreiben Sie ein Beispiel mit der JShell. Geben Sie z.B. die Zahlen 1...100 aus.
- Was sind Simple Reductions und wozu werden sie gebraucht?
- Wie funktioniert die Optional-Methode orElse? Sie mussten das Kapitel 8.7 zwar nicht lesen, Optional.orElse, Optional.isPresent und Optional.get sollten sie kennen.
- Wie können Sie ein Stream in einer java.util.Map sammeln? Machen Sie ein Beispiel mit Country (Key: name, Value: Country)
- Was ist der Unterschied zwischen groupingBy und partitioningBy?

Abschluss

- **Take-Home-Message**
  - Streams erzeugen keine Side Effects auf die Quellen.
  - Streams sind nur einmal benutzbar.
  - Wenn immer möglich Streams verwenden: Code knapper und besser lesbar (deklarative Programmierung und interne Optimierung).
- **Pflichtlektüre**
  - keine Pflichtlektüre.
- **Thema der nächsten Vorlesung**
  - Java FX, Einführung in Basis-Konzepte
- **Übung**
  - Ergänzen Sie das TaskyCLI so, dass es neu ein ***filter*** Befehl gibt. Implementieren Sie diesen Filter mit Streams.