Kotlin

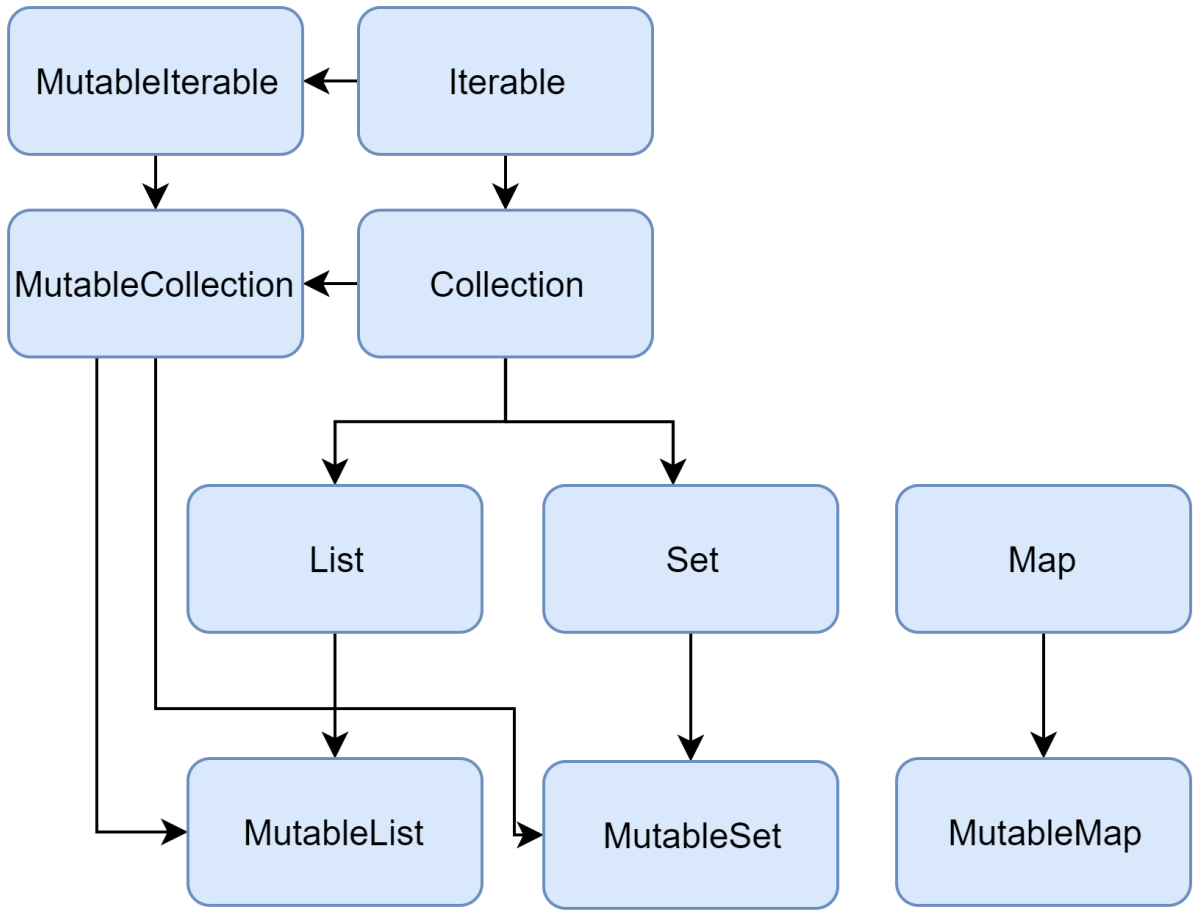
Collection

Kafka

Future/Callback

K8

Mockk – every, verify, justruns


MutableIterable
Iterable
MutableCollection
Collection
List
Set
Map
MutableList
MutableSet
MutableMap

- Iterable: The parent class. Any classes that inherit from this interface represents a sequence of elements we can iterate over.

- Mutable Iterable: Iterables that support removing items during iteration.

- Collection: This class represents a generic collection of elements. We get access to functions that return the size of the collection, whether the collection is empty, contains an item or a set of items. All the methods for this kind of collections are only meant to request data, because collections are immutable.

- Mutable Collection: A Collection that supports adding and removing elements. It provides extra functions such as add, remove or clear, among others.

- List: Probably the most popular collection type. It represents a generic ordered collection of elements. As it's ordered, we can request an item by its position, using the get function.

- MutableList: A List that supports adding and removing elements.

- Set: An unordered collection of elements that doesn't support duplicate elements.

- MutableSet: a Set that supports adding and removing elements.

- Map: a collection of key-value pairs. The keys in a map are unique, which means we cannot have two pairs with the same key un a map.

- MutableMap: A Map that supports adding and removing elements.

Collections Operations

- // Main list we are working with
- val list = listOf(1, 2, 3, 4, 5, 6)

- fun index() {
- aggregateOperations()

- filteringOperations()

- mappingOperations()

- elementsOperations()

- generationOperations()

- orderingOperations()
- }

fun aggregate Operations()

- / any: Returns true if at least one element matches the given predicate. /
- list.any { it % 2 == 0 } // If the lambda has only one parameter, we can use "it" to use it
- list.any { it > 10 } // Should be false

- / all: Returns true if all the elements match the given predicate. /
- list.all { it < 10 } // Should be true
- list.all { it % 2 == 0 } // Should be false

- / count: Returns the number of elements matching the given predicate. /
- list.count { it % 2 == 0 } // Counting all even numbers. Should be 3

- / fold: Accumulates the value starting with an initial value and applying an operation
- from the first to the last element in a collection. /
- list.fold(4) { total, next -> total + next} // Should be 25

- / foldRight: Same as fold, but it goes from the last element to first. /
- list.foldRight(4) { total, next -> total + next} // Should be 25

- / forEach: Performs the given operation to each element. /
- list.forEach { println(it) }

- / forEachIndexed: Same as forEach, though we also get the index of the element. /
- list.forEachIndexed { index, value -> println("position $index contains a $value") }

- / max: Returns the largest element or null if there are no elements. /
- list.max() // Should be 6

- / maxBy: Returns the first element yielding the largest value of the given function or null
- if there are no elements. /
- list.maxBy { -it } // Should be 1

- / min: Returns the smallest element or null if there are no elements. /
- list.min() // Should be 1

- / minBy: Return the first element yielding the smallest value of the given function or null
- if there are no elements./
- list.minBy { -it } // Should be 6

- / none: Returns true if no elements match the given predicate. /
- list.none { it % 7 == 0 } // Should be true: No elements are divisible by 7

- / reduce: Same as fold, but it doesn't use an initial value. It accumulates the value
- applying an operation from the first to the last element in a collection. /
- list.reduce { total, next -> total + next }
- / reduceRight: Same as reduce, but it goes from the last element to first. /
- list.reduceRight { total, next -> total + next }

- / sumBy: Returns the sum of all values produced by the transform function from the
- elements in the collection. /
- list.sumBy { it % 2 } // Should be 3
- }

fun filteringOperations() {

- / drop: Returns a list containing all elements except first n elements. /
- list.drop(4) // Should return listOf(5, 6)

- / dropWhile: Returns a list containing all elements except first elements that satisfy
- the given predicate. /
- list.dropWhile { it < 3 } // Should return listOf(3, 4, 5, 6)

- / dropLast: Same as drop, but drops the last n elements /
- list.dropLast(4) // Should return listOf(1, 2)

- / dropLastWhile: Returns a list containing all elements except last elements that satisfy
- the given predicate /
- list.dropLastWhile { it > 4 } // Should return listOf(1, 2, 3, 4)

- / filter: Returns a list containing all elements matching the given predicate /
- list.filter { it % 2 == 0 } // Should return listOf(2, 4, 6)

- / filterNot: Returns a list containing all elements not matching the given predicate. /
- list.filterNot { it % 2 == 0 } // Should return listOf(1, 3, 5)

- / filterNotNull: Returns a list containing all elements that are not null. /
- val listWithNull = listOf(1, 2, null, 3, null, 4)
- listWithNull.filterNotNull() // Should return listOf(1, 2, 3, 4)

- / slice: Returns a list containing elements at specified indices. /
- val listIndices = listOf(1, 3, 4)
- list.slice(listIndices) // Should return listOf(2, 4, 5)

- / take: Return a list containing first n elements /
- list.take(2) // Should return the first two items (1, 2)

- / Returns a list containing las n elements. /
- list.takeLast(2) // Should return last 2 items (5, 6)

- / takeWhile: Returns a list containing first elements satifying the given predicate /
- list.takeWhile { it < 3 } // Should return listOf(1, 2)
- }

fun mappingOperations() {

- / flatMap: Iterates over the elements creating a new collection for each one, and finally flattens all the collections into a unique list containing all the elements.
- list.flatMap { listOf(it, it + 1) } // Should return listOf(1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7)

- groupBy: Returns a map of the elements in original collection grouped by the result of given function. /
- list.groupBy { if (it % 2 == 0) "even" else "odd" } / Should return a map with "even"/"odd" as key and the number as the value/
- map: Returns a list containing the results of applying the given transform function to each element of the original collection./
- list.map { it 2 } // Should return listOf(2, 4, 6, 8, 10, 12)

- mapIndexed: Returns a list containing the results of applying the given transform function to each element and its index of the original collection.
- list.mapIndexed { index, it -> index it } // Should return listOf(0, 2, 6, 12, 20, 30)

- mapNotNull: Returns a list containing the results of applying the given transformation to each non-null element of the original collection.
- val listWithNull = listOf(1, 2, null, 3, null, 4)
- listWithNull.mapNotNull { / it 2 / } // Should return listOf(2, 4, 6, 8)

fun elementsOperations() {

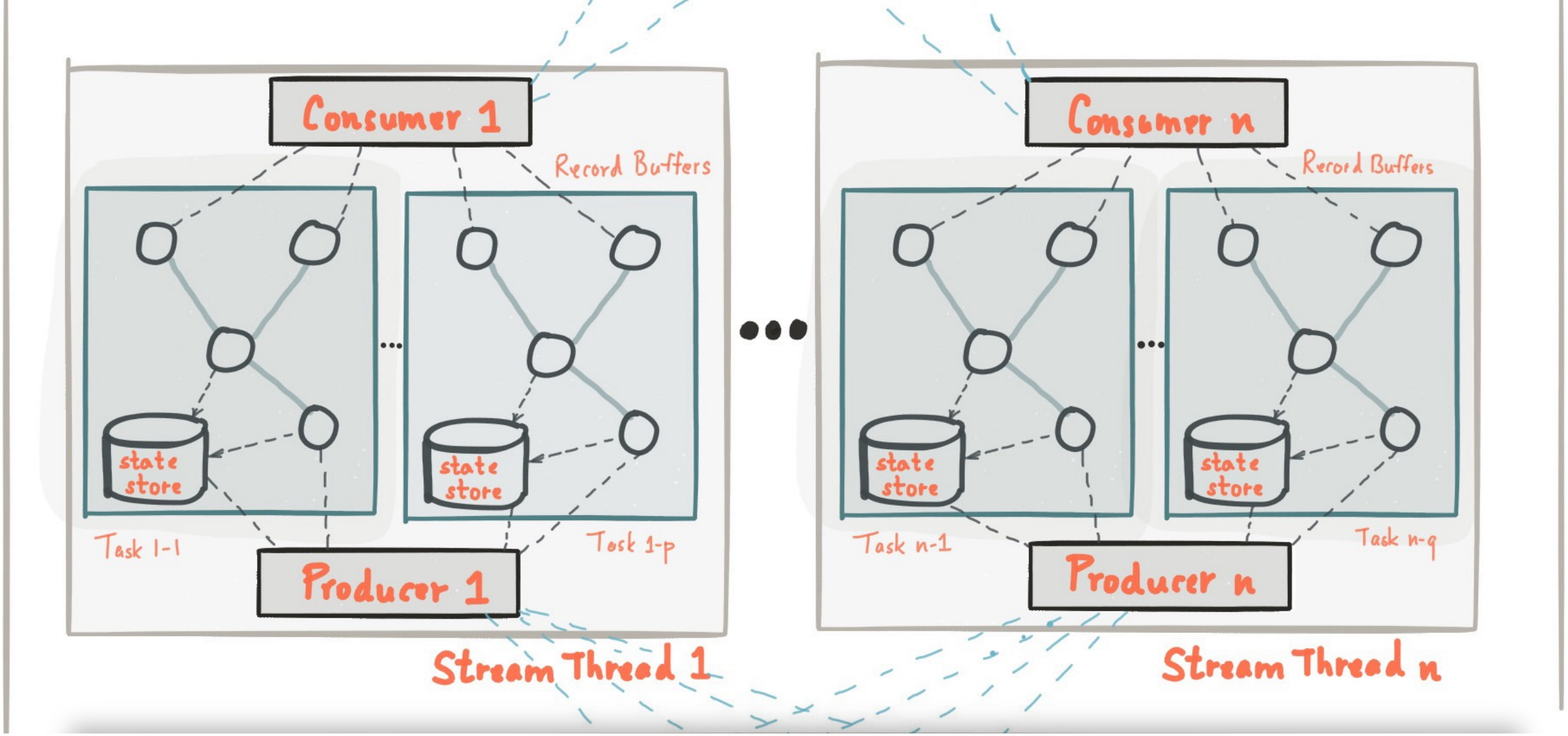
```
    /* contains: Returns true if the element is found in the collection*/
    list.contains(2) // Should return true

    /* elementAt: Returns an element at the given index or throws an IndexOutOfBoundsException if
                the index is out of bounds of this collection. */
    list.elementAt(1) // Should return 2

    /* elementAtOrElse: Returns an element at the given index or the result of calling the
     *                default function if the index is out of bounds of this collection */
    list.elementAtOrElse(10, { 2 * it }) // Should return 20

    /* elementAtOrNull: Returns an element at the given index or null if the index is out of
    *                bounds of this collection.*/
    list.elementAtOrNull(10) // Should return null

    /* first: Returns the first element matching the given predicate. It will show a
    *        NoSuchElementException if no elements are found.*/
    list.first { it % 2 == 0 } // Should return 2

    /* firstOrNull:  */
    list.firstOrNull { it % 7 == 0 } // Should return null

    /* indexOf: Returns the first index of element, or -1 if the collection does not contain element.*/
    list.indexOf(4) // Should return 3

    /* indexOfFirst: Returns index of the first element matching the given predicate, or -1
    *               if the collection does not contain such element.*/
    list.indexOfFirst { it % 2 == 0 } // Should return 1
    /* indexOfLast: Returns the index of the last element matching the given predicate, or -1
    *              if the collection does not contain such element.*/
    list.indexOfLast { it % 2 == 0 } // Should return 5

    /* last:  Returns the last element matching the given predicate. It will throw a
    *        NoSuchElementException if no elements are found.*/
    list.last { it % 2 == 0 } // Should return 6
    /* lastIndexOf: Returns last index of element, or -1 if the collection
                does not contain element.*/
    val listRepeated = listOf(2, 2, 3, 4, 5, 5, 6)  listRepeated.lastIndexOf(5) // Should return 5

    /* lastOrNull: Returns the last element matching the given predicate, or null if no such element was found.*/
    list.lastOrNull { it % 7 == 0 } // Should return null
    /* single: Returns the single element matching the given predicate, or throws exception
    *          if there is no or more than one matching element.*/
    list.single { it % 5 == 0 } // Should return 5

    /* singleOrNull: Returns the single element matching the given predicate, or null if element
    *               was not found or more than one element was found.*/
    list.singleOrNull { it % 7 == 0 } // Should return null
}
```

fun generationOperations() {
& fun orderingOperations() {

```
    /* partition: Splits the original collection into pair of lists, where the first list
                contains elements for which the predicate returned true, while the second list
                contains elements for which the predicate returned false.*/

    list.partition { it % 2 == 0 } /* Should return
                                Pair( listOf(2, 4, 6) , listOf(1, 3, 5) ) */
    /* plus: Returns a list containing all elements of the original collection and then
            all elements of the given collection. Because of the name of the function, we can use
            the "+" operator with it.*/
    list + listOf(7, 8) // Should return listOf(1, 2, 3, 4, 5, 6, 7, 8)

    /* zip: Returns a list of pairs built from the elements of both collections with the same
    *       indexes. The list has the length of the shortest collection.*/
    list.zip(listOf(7, 8)) // Should return listOf( Pair(1, 7), Pair(2, 8) )

    /* unzip: Generates a Pair of Lists from a List of Pairs.*/
    val listOfPairs = listOf( Pair(5, 7), Pair(6, 8) )
    listOfPairs.unzip() // Should return Pair( listOf(5, 6), listOf(7, 8) )
}
```

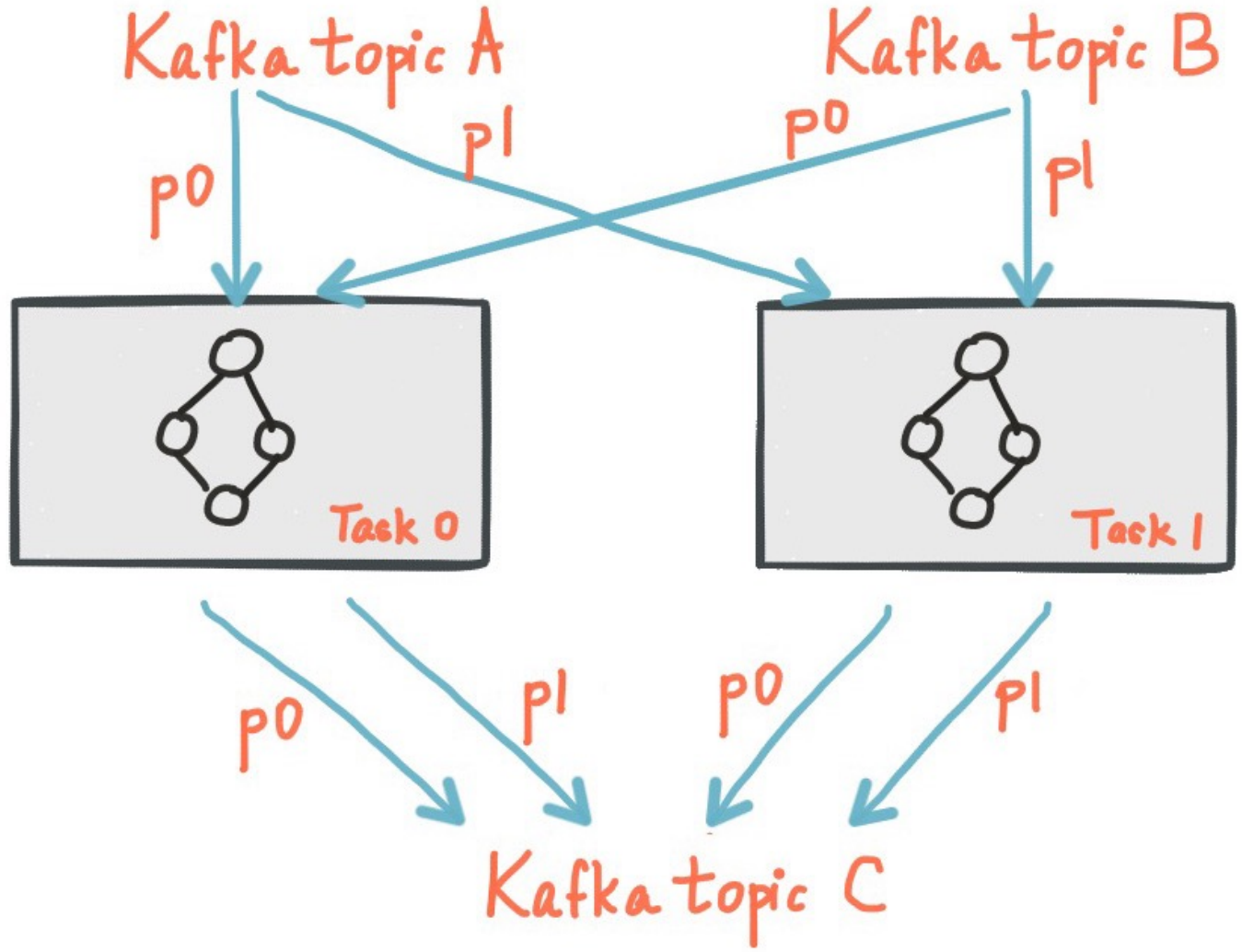
```
    /* reversed: Returns a list with elements in reversed order.*/
    list.reversed() // Should return listOf(6, 5, 4, 3, 2, 1)
    /* sorted: Returns a sorted list of all elements.*/
    val unsortedList = listOf(3, 2, 7, 5)
    unsortedList.sorted() // Should return listOf(2, 3, 5, 7)

    /* sortedBy: Returns a list of all elements, sorted by the specified comparator.*/
    unsortedList.sortedBy { it % 3 } // Should return listOf(3, 7, 2, 5)

    /* sortedDescending: Returns a sorted list of all elements in descending order.*/
    unsortedList.sortedDescending() // Should return listOf(7, 5, 3, 2)

    /* sortedByDescending: Returns a sorted list of all elements, in descending order by the
    *                   results of the specified order function.*/
    unsortedList.sortedByDescending { it % 3 } // Should return listOf(2, 5, 7, 3)
}
```

Kafka



Producer / Consumer

Topic


Kafka topic A
Kafka topic B
P0
P1
P0
P1
Task 0
Task 1
P0
P1
P0
P1
Kafka topic C