



More about Sequences

Collections vs sequences

```
listOf(1, 2, 3, 4)  
  .map { it * it }  
  .find { it > 3 }
```

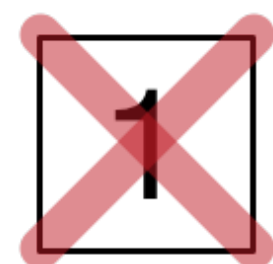
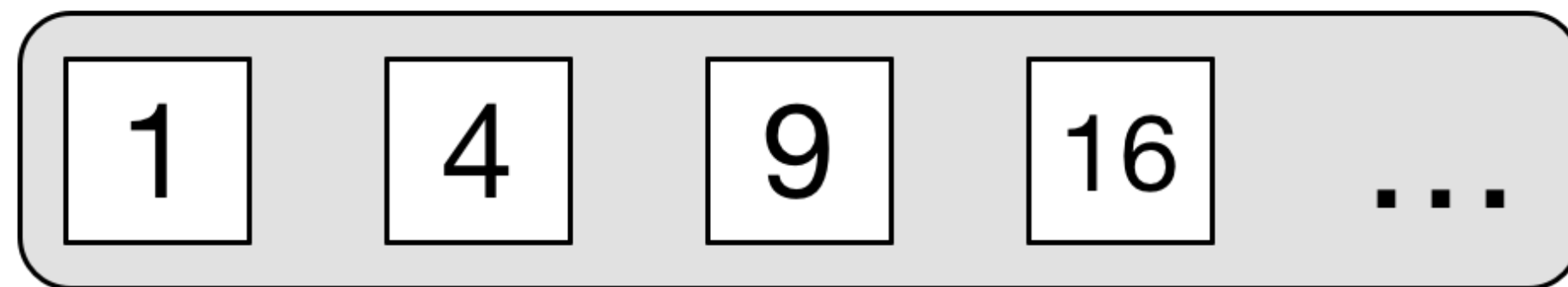
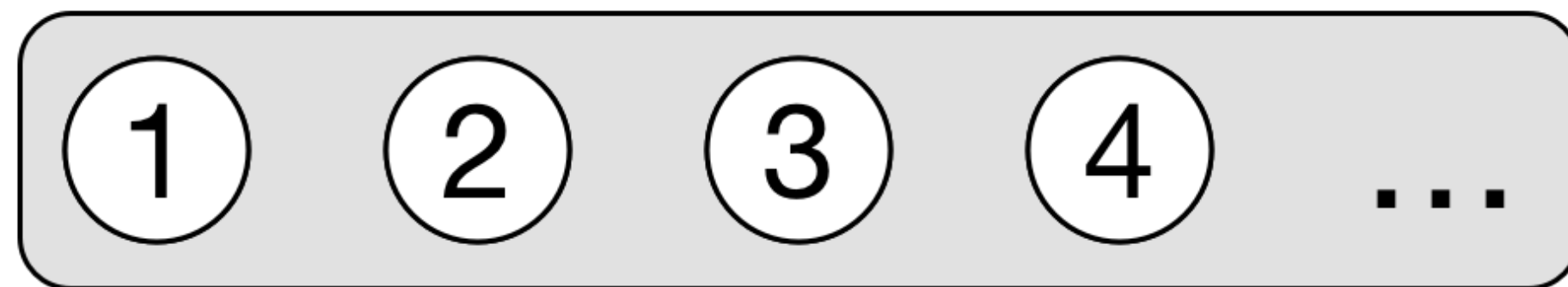
4

```
listOf(1, 2, 3, 4)  
  .asSequence()  
  .map { it * it }  
  .find { it > 3 }
```

4

Collections vs Sequences

Eager

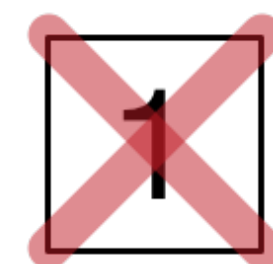
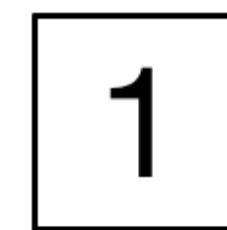
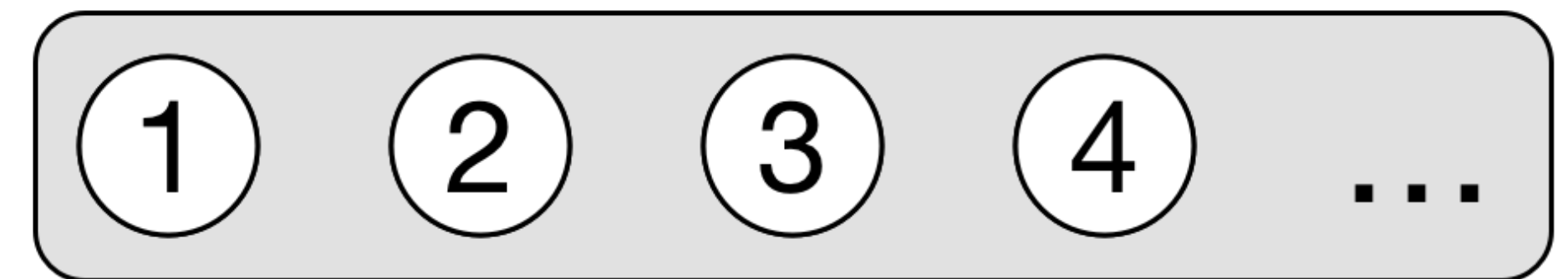


result

map

find

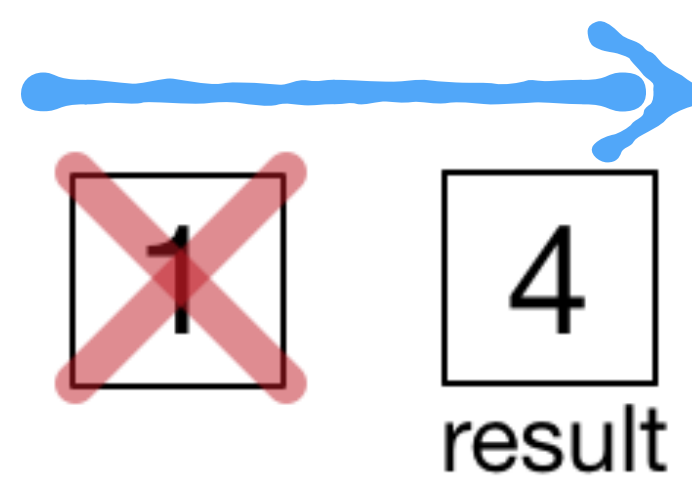
Lazy



result

Collections vs Sequences

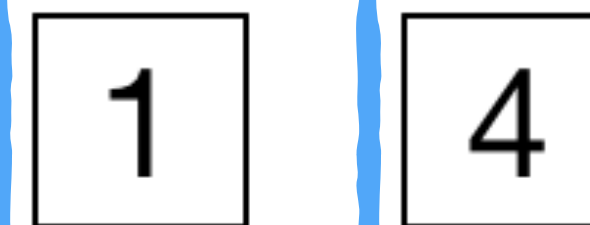
Horizontal evaluation



Vertical evaluation



map



find





Write the output after the evaluation of the expressions below.

```
fun m(i: Int): Int {  
    print("m$i ")  
    return i  
}
```

```
fun f(i: Int): Boolean {  
    print("f$i ")  
    return i % 2 == 0  
}
```

```
val list = listOf(1, 2, 3, 4)
```

```
list.map(::m).filter(::f) m1 m2 m3 m4 f1 f2 f3 f4
```

```
list.asSequence().map(::m).filter(::f).toList() ?
```

```
list.asSequence().map(::m).filter(::f) ?
```





Write the output after the evaluation of the expressions below.

```
fun m(i: Int): Int {  
    print("m$i ")  
    return i  
}
```

```
fun f(i: Int): Boolean {  
    print("f$i ")  
    return i % 2 == 0  
}
```

```
val list = listOf(1, 2, 3, 4)
```

```
list.map(::m).filter(::f) m1 m2 m3 m4 f1 f2 f3 f4
```

```
list.asSequence().map(::m).filter(::f).toList()
```

m1 f1 m2 f2 m3 f3 m4 f4

```
list.asSequence().map(::m).filter(::f) ☐ (nothing is printed)
```

Nothing happens until the terminal operation is called

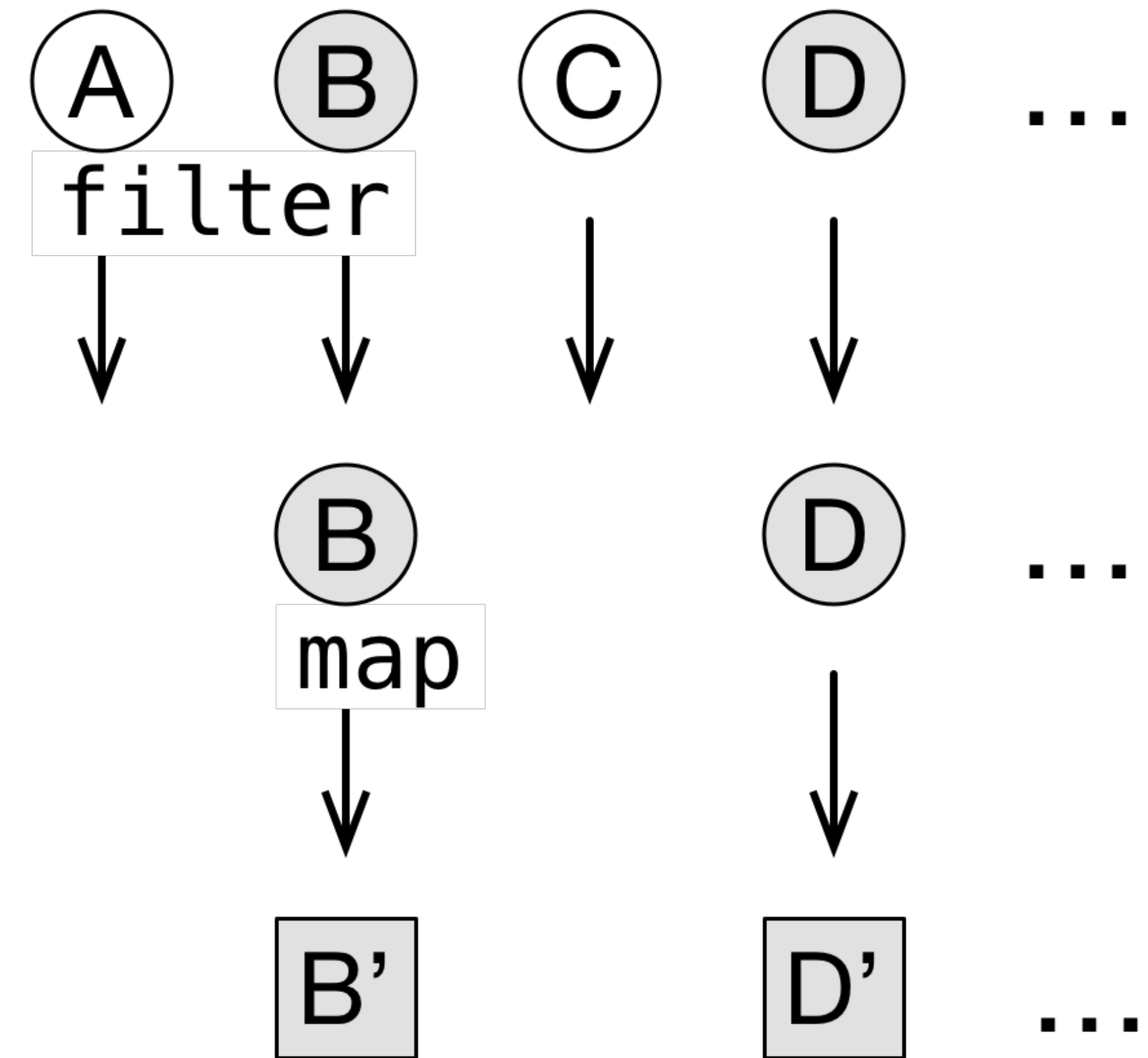
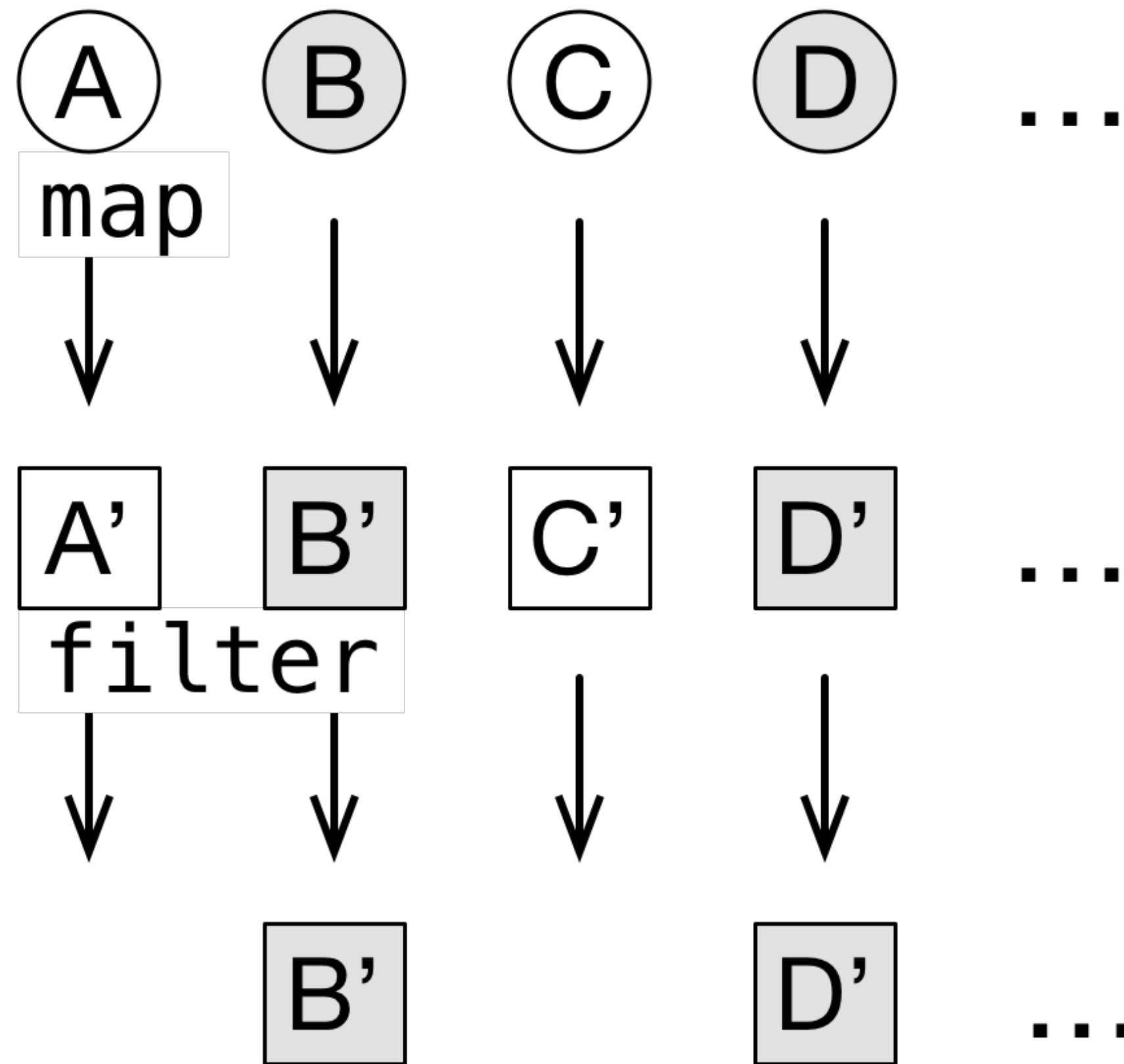
The diagram illustrates the execution of a sequence of operations. The text `sequence.map { ... }.filter { ... }.toList()` is shown. Above the `map` and `filter` methods, a bracket spans both, with the label "intermediate operations" above it. Two diagonal lines point from the label to the bracket. Below the `toList()` method, a bracket spans the method name, with the label "terminal operation" below it. A diagonal line points from the label to the bracket.

```
sequence.map { ... }.filter { ... }.toList()
```

intermediate operations

terminal operation

Order of operations is important





Write the output after the evaluation of the last expression below.

```
fun m(i: Int): Int {  
    print("m$i ")  
    return i  
}
```

```
fun f(i: Int): Boolean {  
    print("f$i ")  
    return i % 2 == 0  
}
```

```
val list = listOf(1, 2, 3, 4)
```

```
list.asSequence().map::m).filter::f).toList()  
                                     m1 f1 m2 f2 m3 f3 m4 f4
```

```
list.asSequence().filter::f).map::m).toList()    ?
```





Write the output after the evaluation of the last expression below.

```
fun m(i: Int): Int {  
    print("m$i ")  
    return i * i  
}
```

```
fun f(i: Int): Boolean {  
    print("f$i ")  
    return i % 2 == 0  
}
```

```
val list = listOf(1, 2, 3, 4)
```

```
list.asSequence().map(::m).filter(::f).toList()  
                                     m1 f1 m2 f2 m3 f3 m4 f4
```

```
list.asSequence().filter(::f).map(::m).toList()  
                                     f1 f2 m2 f3 f4 m4
```

Collections vs Sequences

intermediate collections are created on chained calls

vs

lambdas are not inlined