Building Custom Collectors for Advanced Data Processing

José Paumard

@JosePaumard | blog.paumard.org

Agenda

Building custom collectors

A complex custom collector in action

Building Custom Collectors

Crafting our own collectors from scratch

What if I Need Another Collector?

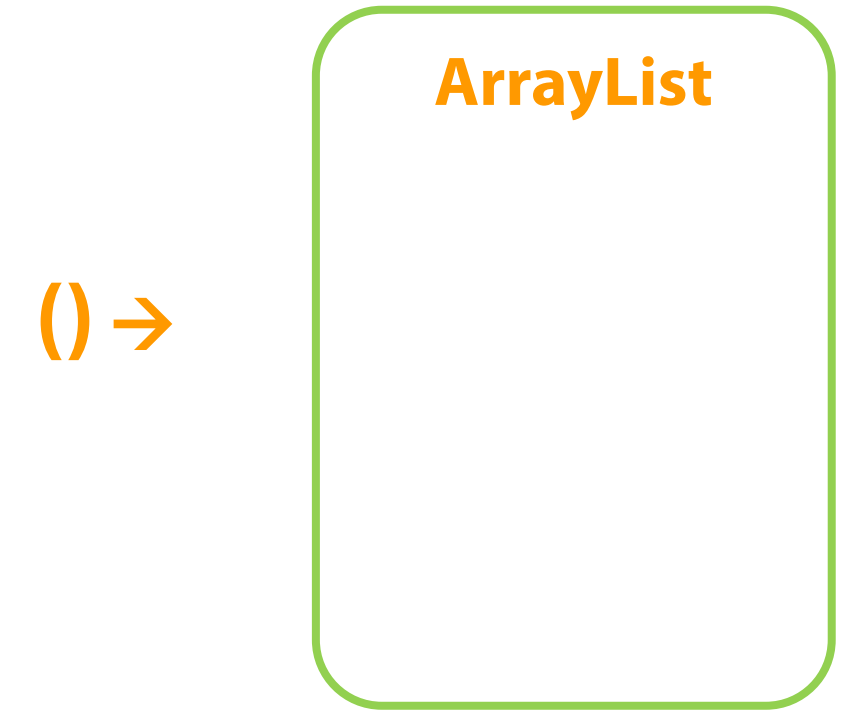
- So far we saw ready to use Collectors…
- But of course, there are cases that do not fit in this catalog
- We need a way to build our own collectors!

Collectors Under the Hood

- A collector is in fact made of three elements:
- The first element is used to build the resulting container
  - For instance an ArrayList, or a HashMap
- The second element adds an object from the stream to the container
  - For instance, it adds an object to an ArrayList
- The third element is used for parallelism
  - It is used to merge together two partially filled containers

Building an Empty Container

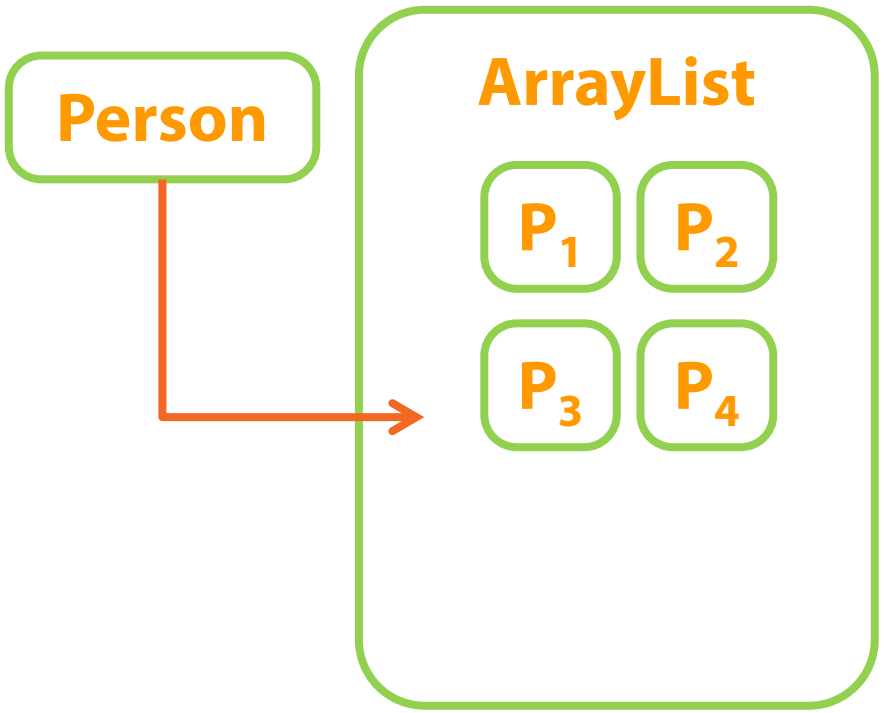
- Example of an ArrayList:




```
Suppier<List<Person>> supplier =
   () -> new ArrayList<Person>() ;
```

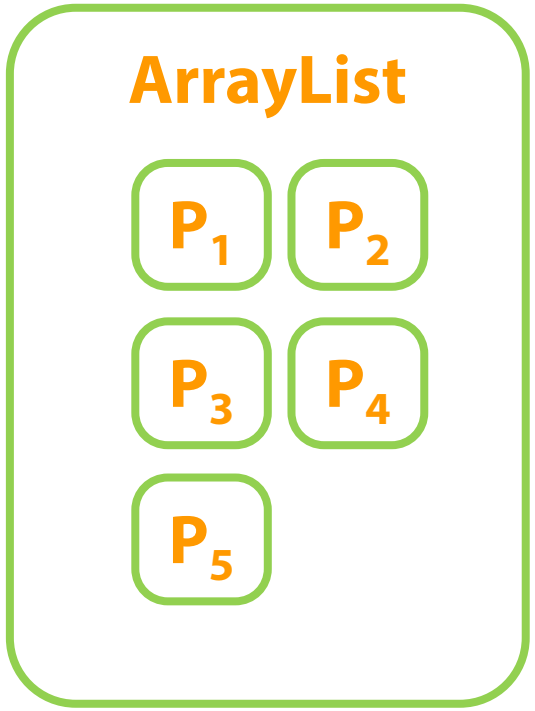
Adding an Object to the Container

- Example of an ArrayList:



Adding an Object to the Container

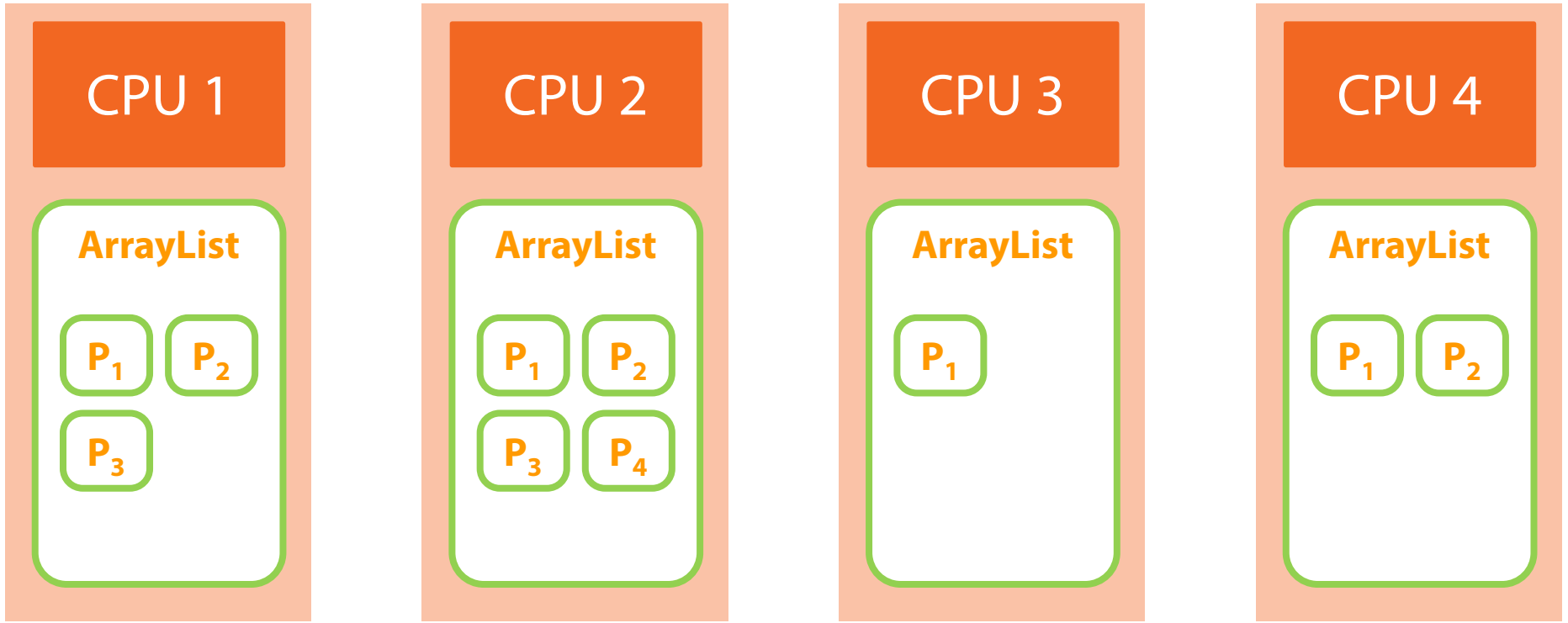
- This step is the accumulating step, and requires an accumulator:




```
BiConsumer<Person, List<Person>> accumulator =
    (p, list) -> list.add(p) ;
```

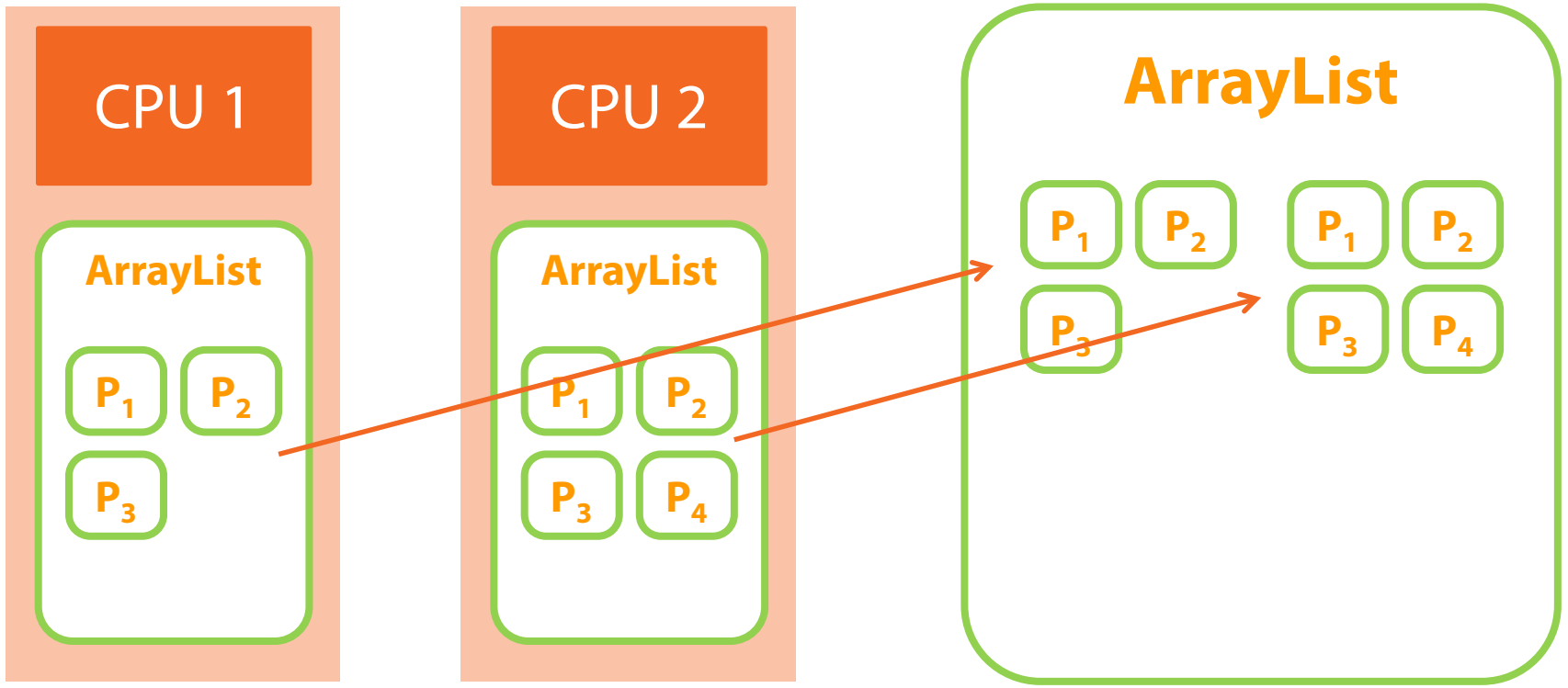
Merging Partially Filled Containers

- Example of an ArrayList:



Merging Partially Filled Containers

- Example of an ArrayList:



Merging Partially Filled Containers

- This step is call the combining step, and requires a combiner

```
BinaryOperator<List<Person>> combiner =
   (list1, list2) -> {
      list1.addAll(list2) ;
      return list1 ;
   }
```

Putting It All Together

- A collector can be built out of these three elements

```
Collector collector = Collector.of(
    () -> new ArrayList(),
    (person, list) -> list.add(person),
    (list1, list2) -> { list1.addAll(list2) ; return list1 ; }

) ;
```

Putting It All Together

- A collector can be built out of these three elements

```
Collector collector = Collector.of(
    () -> new ArrayList(),
    (person, list) -> list.add(person),
    (list1, list2) -> { list1.addAll(list2) ; return list1 ; }

) ;
```

Putting It All Together

- A collector can be built out of these three elements

```
Collector collector = Collector.of(
    () -> new ArrayList(),
    (person, list) -> list.add(person),
    (list1, list2) -> { list1.addAll(list2) ; return list1 ; }

) ;
```

Putting It All Together

- A collector can be built out of these three elements

```
Collector collector = Collector.of(
    () -> new ArrayList(),
    (person, list) -> list.add(person),
    (list1, list2) -> { list1.addAll(list2) ; return list1 ; }

) ;
```

Putting It All Together

- A collector can be built out of these three elements

```
Collector collector = Collector.of(
    () -> new ArrayList(),
    (person, list) -> list.add(person),
    (list1, list2) -> { list1.addAll(list2) ; return list1 ; },
    Collectors.Characteristics.IDENTITY_FINISH
) ;
```

- Three flags in Collectors.*Characteristics*
  - IDENTITY_FINISH
  - CONCURRENT
  - UNORDERED

Live Coding

The Actors and Movies example

Live Coding Summary

- We saw how to build a complex data processing stream
  - We built maps, extracted their values
- We could solve a non-trivial and computationaly intensive case
  - 170k actors processed in less than a second
- We built a complex custom collector to fine tune our computation
  - The Collector API has been made for that!

Other Custom Collectors

- Back on the IntStream.summaryStatistics() pattern

```
public final IntSummaryStatistics summaryStatistics() {

   return collect(IntSummaryStatistics::new,
                  IntSummaryStatistics::accept,
                  IntSummaryStatistics::combine
          );
}
```

Other Custom Collectors

- Back on the IntStream.summaryStatistics() pattern

```
public final IntSummaryStatistics summaryStatistics() {

   return collect(IntSummaryStatistics::new,      // supplier
                  IntSummaryStatistics::accept,   // accumulator
                  IntSummaryStatistics::combine   // combiner
          );
}
```

Other Custom Collectors

- Back on the IntStream.summaryStatistics() pattern

```
public void accept(int value) {
    ++count;
    sum += value;
    min = Math.min(min, value);
    max = Math.max(max, value);
}
```

```
public void combine(IntSummaryStatistics other) {
    count += other.count;
    sum += other.sum;
    min = Math.min(min, other.min);
    max = Math.max(max, other.max);
}
```

Summary

- The collector pattern is the right pattern to compute complex reduction
  - Especially in mutable containers
- We can build our own collectors if the standard ones are not enough!
  - With the Collector.of pattern
  - No need to implement the Collector interface
- We saw how to do that on a complex case: the Movies and Actors example
  - We combined complex collectors
  - We added complex post processings