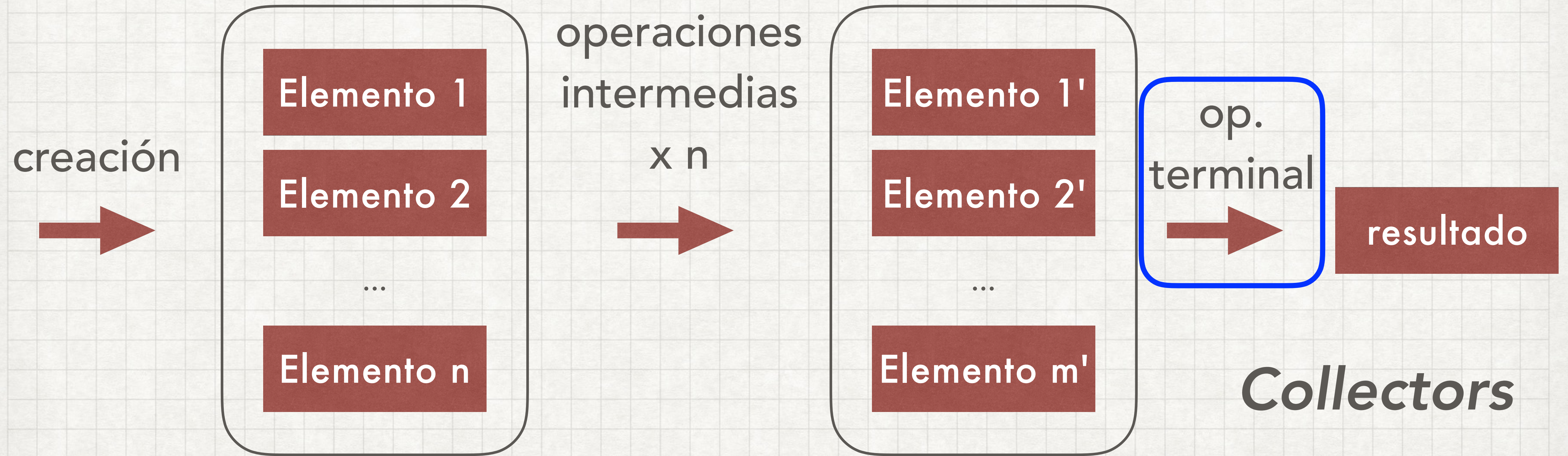


COLLECTORS



# *Stream pipeline*





## Agrupaciones "a la antigua"

```
List<Alumno> alumnos = ...

Map<String, List<Alumno>> porEscuela = new HashMap<>();

for (Alumno alumno: alumnos) {
    if (! porEscuela.containsKey(alumno.getEscuela())) {
        porEscuela.put(alumno.getEscuela(), new ArrayList<>());
    }
    porEscuela.get(alumno.getEscuela()).add(alumno);
}
```



## Collectors

```
porEscuela = alumnos.stream()  
                .collect(Collectors.groupingBy(Alumno::getEscuela));
```

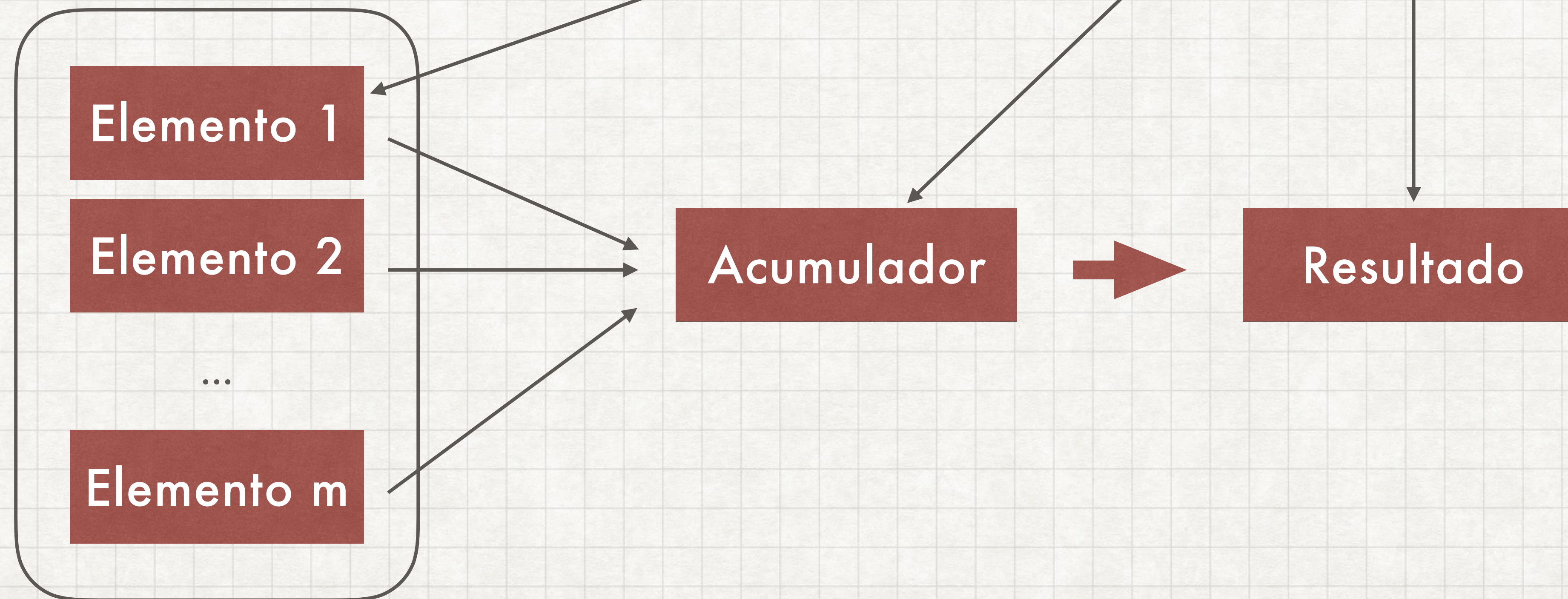
○

```
import static java.util.stream.Collectors.*;
```

```
porEscuela = alumnos.stream().collect(groupingBy(Alumno::getEscuela));
```



public interface Collector<T, A, R>

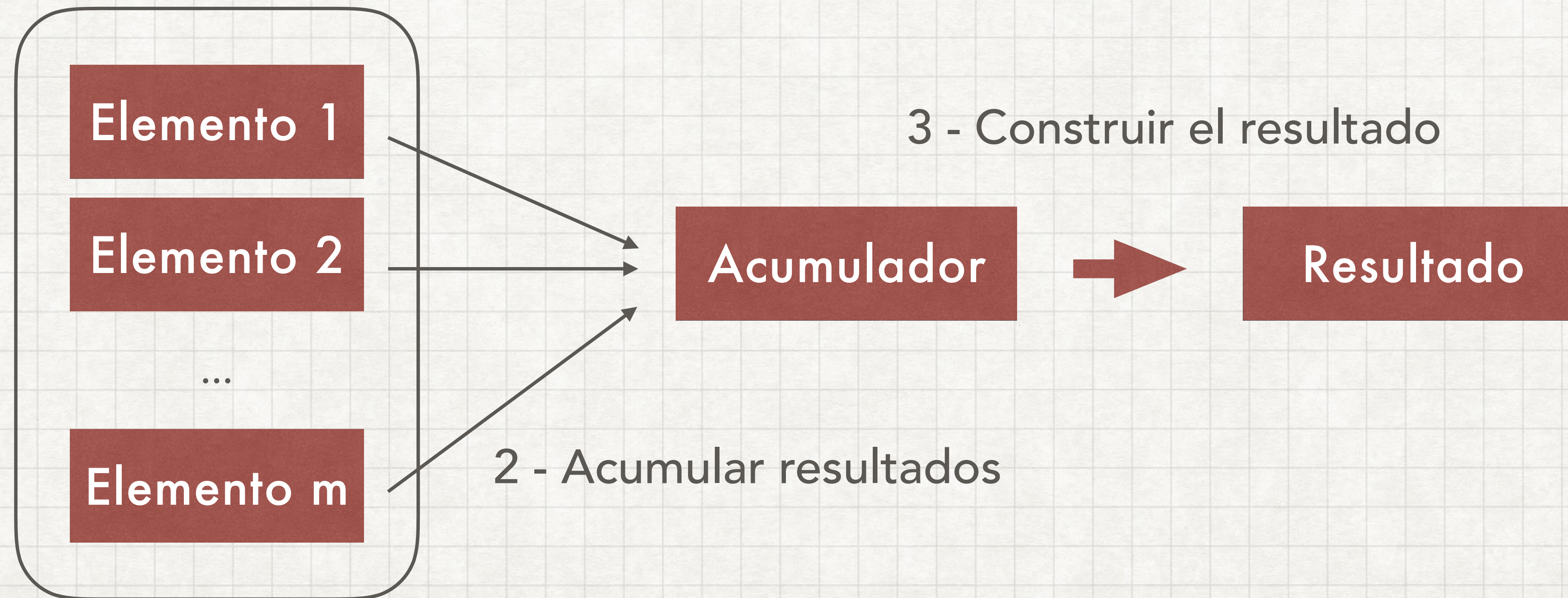




```
public interface Collector<T, A, R>
```

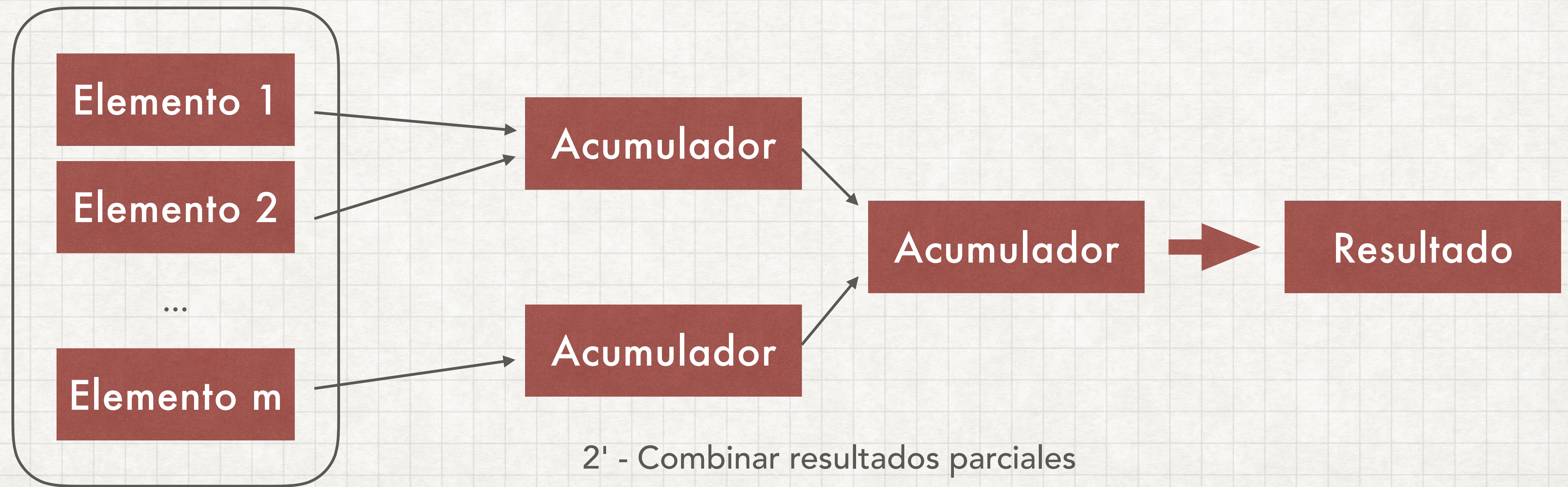
**collect**

1 - Crear el Acumulador



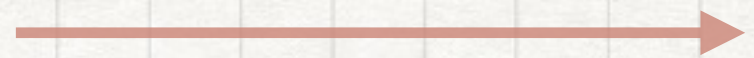


```
public interface Collector<T, A, R>
```






```
public interface Collector<T, A, R>
```

1 - Crear el Acumulador  Supplier<A> supplier

2 - Acumular resultados  BiConsumer<A, T> accumulator

2' - Combinar resultados parciales  BinaryOperator<A> combiner

3 - Construir el resultado  Function<A, R> finisher



```
public interface Collector<T, A, R>
```

1 - Crear el Acumulador  $\longrightarrow$  Supplier<A> supplier

2 - Acumular resultados  $\longrightarrow$  BiConsumer<A, T> accumulator

2' - Combinar resultados parciales  $\longrightarrow$  `BinaryOperator<A>` combiner

3 - Construir el resultado  $\longrightarrow$  `Function<A, R> finisher`

[illegible]



## enum Characteristics

CONCURRENT	El collector puede ser invocado concurrentemente
UNORDERED	El collector no necesita ser invocado en orden
IDENTITY_FINISH	La función de finalización es la identidad



# Collectors

Amplio catálogo de "acumuladores" típicos

- Agrupaciones
- Operaciones aritméticas (medias, sumas)
- Conversión a Collections (List, Set, Map)
- ...



# Agrupaciones

```
Map<String, List<Persona>> porMunicipio =  
    personas.stream()  
        .collect(Collectors.groupingBy(Persona::getMunicipio));
```

"municipio 1"	[ Persona1, Persona2 ]
"municipio 2"	[ Persona n, Persona m ... ]
"municipio 3"	[ Persona y, Persona z ... ]



## Operaciones aritméticas

```
Double edadMedia = personas.stream()  
    .collect(Collectors.averagingInt(Persona::getEdad));
```

## Conversión a Collections

```
Set<Persona> set = personas.stream().collect(Collectors.toSet());
```



## Cobinación de collectors

```
Map<String, Double> porMunicipio = personas.stream().collect(
    Collectors.groupingBy(
        Persona::getMunicipio,
        Collectors.averagingInt(Persona::getEdad)));
```

"municipio 1"	edad media municipio1
"municipio 2"	edad media municipio 2