

# We are all to gather

Rémi Forax

Université Gustave Eiffel – January 2024

# We are all together

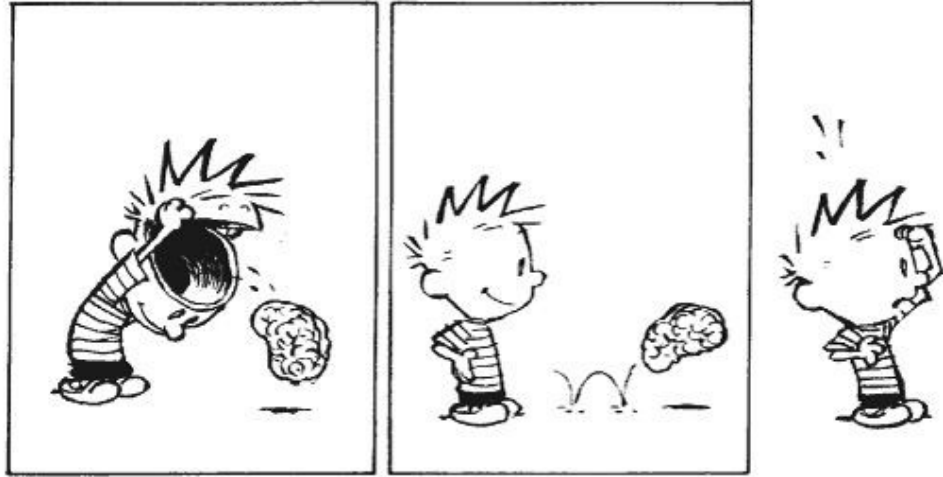
Rémi Forax

Université Gustave Eiffel – January 2024

# We are all to **gather**

Rémi Forax

Université Gustave Eiffel – January 2024



CALVIN & HOBBS © BIL WATTERSON

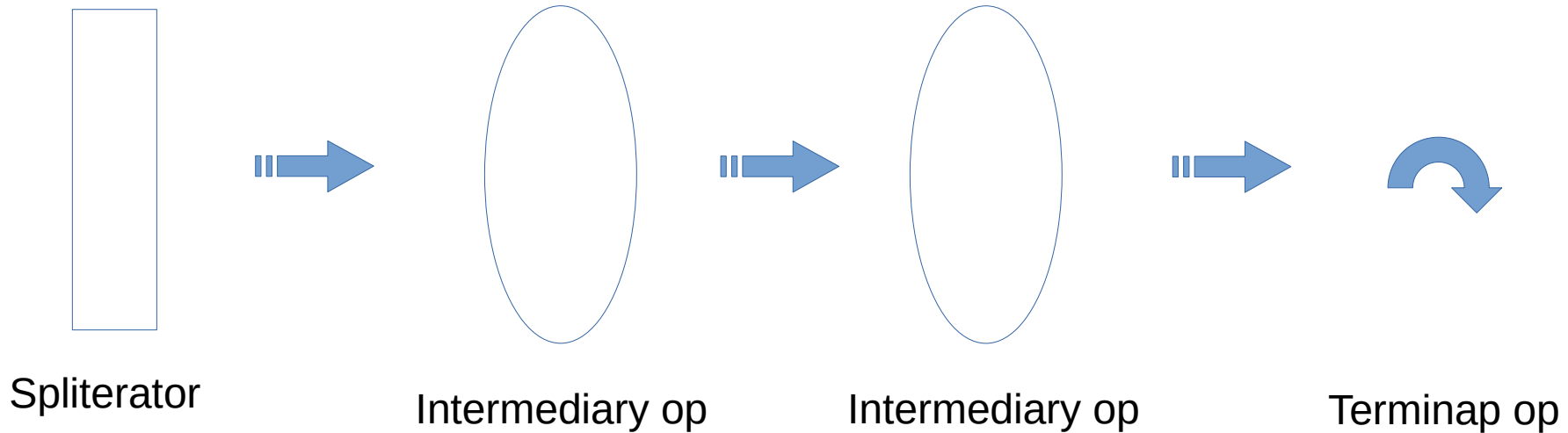
Don't believe what I'm saying !

# Outline

- Stream operations
- The Gatherer API
- Performance and limitations

# Stream == pipeline

The terminal operation drives the pipeline



# Intermediary Ops

## 3 axis

- Can be parallelizable ? sequential/parallel
- Have an internal state ? stateless/stateful
- Can stop the computation ? greedy/short-circuiting

# Intermediary Ops (examples)

## Operations

- map() ??



# Intermediary Ops (examples)

## Operations

- map() parallelizable, stateless, greedy
- filter() ??

# Intermediary Ops (examples)

## Operations

- `map()` parallelizable, stateless, greedy
- `filter()` parallelizable, stateless, greedy
- `takeWhile()` ??

# Intermediary Ops (examples)

## Operations

- `map()` parallelizable, stateless, greedy
- `filter()` parallelizable, stateless, greedy
- `takeWhile()` sequential, stateless, short-circuit
- `limit()` ??

# Intermediary Ops (examples)

## Operations

- `map()` parallelizable, stateless, greedy
- `filter()` parallelizable, stateless, greedy
- `takeWhile()` sequential, stateless, short-circuit
- `limit()` sequential, stateful, short-circuit
- `reduce()` ??

# Intermediary Ops (examples)

## Operations

- `map()` parallelizable, stateless, greedy
- `filter()` parallelizable, stateless, greedy
- `takeWhile()` sequential, stateless, short-circuit
- `limit()` sequential, stateful, short-circuit
- `reduce()` parallelizable, stateful, greedy

Live Code !

# Gatherer API

# Gatherer<E, A, T>

initializer: Supplier<A>

- Create a state

integrator (A state, E element, Downstream<T> downstream) → boolean

- Accumulate in state and/or push downstream (back-propagate return type)

combiner: BinaryOperator<A>

- Combine two states, return a new state

finisher: BiConsumer<A, Downstream<T>>

- push downstream



# Creating a Gatherer : 3 axis

## Sequential only vs Parallelizable

- Gatherer.ofSequential() vs Gatherer.of() + combiner?

## Stateless vs Stateful

- integrator vs initializer + integrator + finisher?

## Short-circuit vs Greedy

- integrator vs Integrator.ofGreedy()

What's missing ?

# Performance (map + sum)

```
public int stream_map_sum() {  
    return values.stream().map(String::length).reduce(0, Integer::sum);  
}    // 481.222 ± 1.560 us/op  
  
public int stream_mapToInt_sum() {  
    return values.stream().mapToInt(String::length).sum();  
}    // 102.089 ± 0.672 us/op  
  
public int gatherer_map_sum() {  
    return values.stream().gather(map(String::length)).reduce(0, Integer::sum);  
}    // 552.384 ± 3.405 us/op
```

No primitive specialization ...

\* `values` is a List of 100\_000 strings

# Performance (map + toList)

```
public List<Integer> stream_map_toList() {  
    return values.stream().map(String::length).toList();  
}    // 332.322 ± 0.512 us/op  
  
public List<Integer> gatherer_map_toList() {  
    return values.stream().gather(map(String::length)).toList();  
}    // 558.873 ± 6.200 us/op
```

Why using a Gatherer is slower ? ...

# Performance (map + count)

```
public long stream_map_count() {  
    return values.stream().map(String::length).count();  
}    // 0.009 ± 0.001 us/op  
  
public long stream_mapToInt_count() {  
    return values.stream().mapToInt(String::length).count();  
}    // 0.009 ± 0.001 us/op  
  
public long gatherer_map_count() {  
    return values.stream().gather(map(String::length)).count();  
}    // 101.993 ± 0.105 us/op
```

=> spliterator characteristics are not propagated !

# Performance issues

## No primitive specialization

- mapToInt/flatMapToInt, etc
  - Same issue with collectors
  - Valhalla generics to the rescue ??

## Splititerator characteristics are not propagated

- Same issue with collectors
  - For ex: Stream.toList() can pre-size, not Collectors.toList()

# Executive Summary

# Gatherer API

User defined intermediary operations

- 3 axis: short-circuitability / statefulness / only-sequential

Gatherers contains predefined Gatherers

Still In preview

- Not enough predefined Gatherers
- Splitter characteristics should be propagated
- “default operations” design is controversial



# Questions ?

[https://github.com/forax/we\\_are\\_all\\_to\\_gather](https://github.com/forax/we_are_all_to_gather)