LEVEL UP YOUR JAVA STREAMS WITH GATHERERS

Hinse ter Schuur
@coduinix

SDB
Java

WHO AM I?

```
{
  "name": "Hinse ter Schuur",
  "company": "SDB Java",
  "role": "Software Engineer",
  "experience": "20+ years",
  "expertise": ["Java", "Scala"],
  "client_industries": [
    "Transportation", "Financial Services",
    "Logistics", "Government"
  ]
}
```

JAVA STREAMS 😍

- "Functional view" on Collections
  - Chain of transformations
  - Nice fit for lambdas
  - Computed lazily
- Potentially unbounded
- Easy to parallelize

WHAT IS A STREAM (PIPELINE)?

- source of elements
- any nr of intermediate operations
- terminal operation

```
$ git checkout live-demo
```

COMPLICATIONS 💔

INTERMEDIATE OPERATIONS

```
var result = measurements()
      .map(Measurement::temperature)
      // doesn't exist
      .fixedWindow(60)
      .map(sixtyMinuteWindow -> calculateAverage(sixtyMinuteWindow))
      .limit(5)
      .toList();
```

INTERMEDIATE OPERATIONS

```
var result = measurements()
      .map(Measurement::temperature)
      // doesn't exist
      .zipWithIndex((idx, temp) -> idx + ": " + temp)
      .limit(5)
      .toList();
```

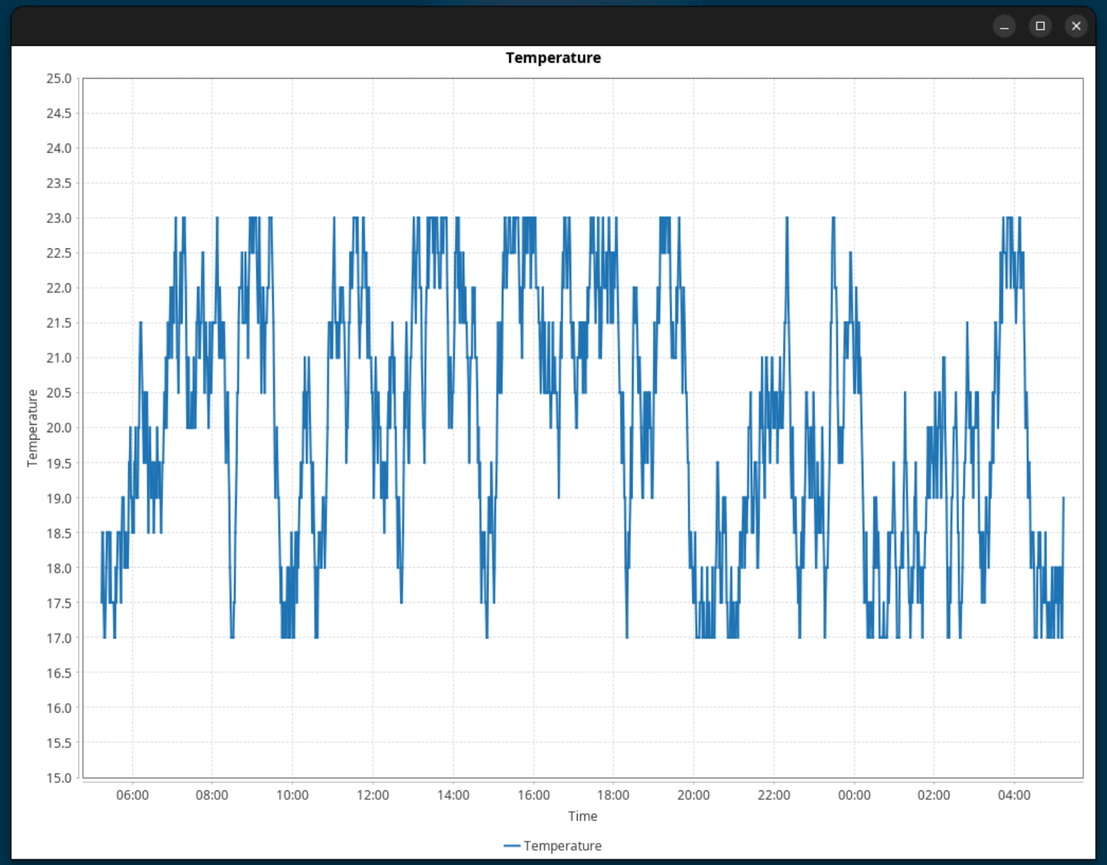
GATHERERS 💓

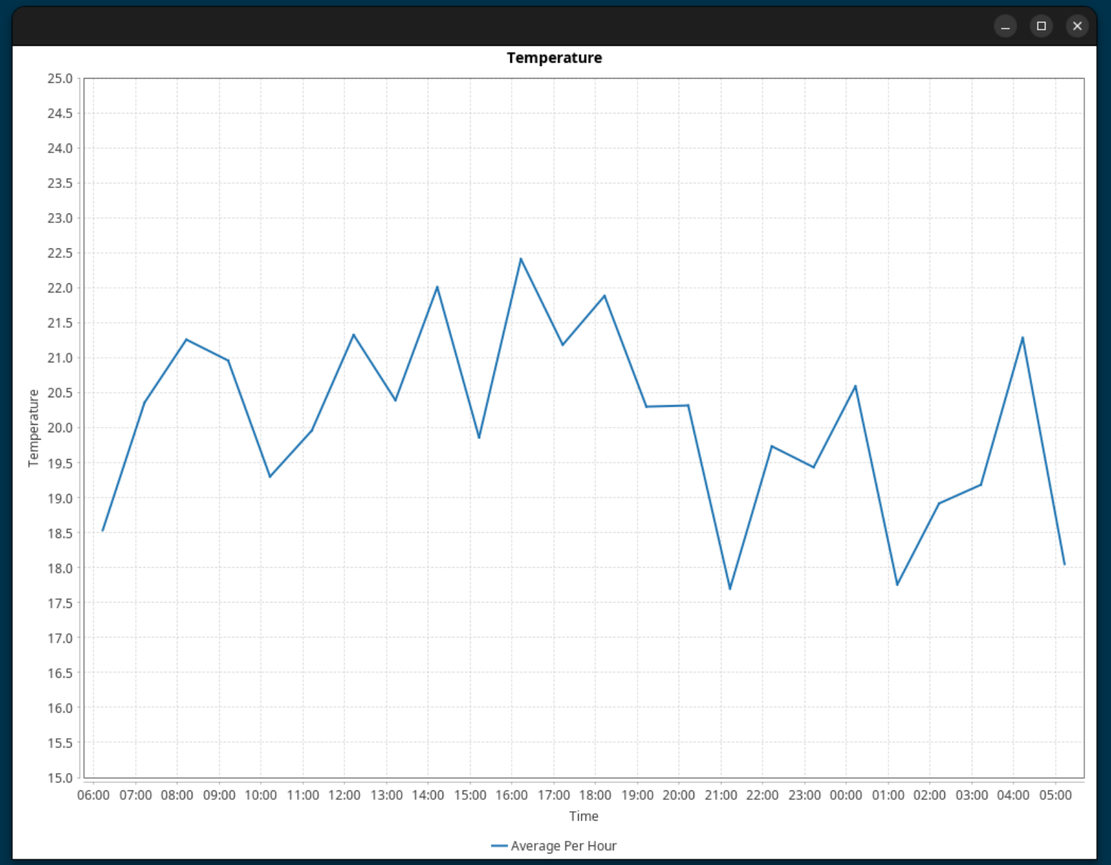
- Generic mechanism for intermediate operations
- Available since Java 24 (JEP 485)
- `.gather(Gatherer)` method on Streams
- Inspired by Collectors

ENTER GATHERERS (2)

- Support parallel streams
- Lazy evaluation
- Can use state
- `1:1`, `1:n`, `n:1` and `n:m`

EXAMPLE


Temperature
25.0
24.5
24.0
23.5
23.0
22.5
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
Temperature
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
Time
Temperature

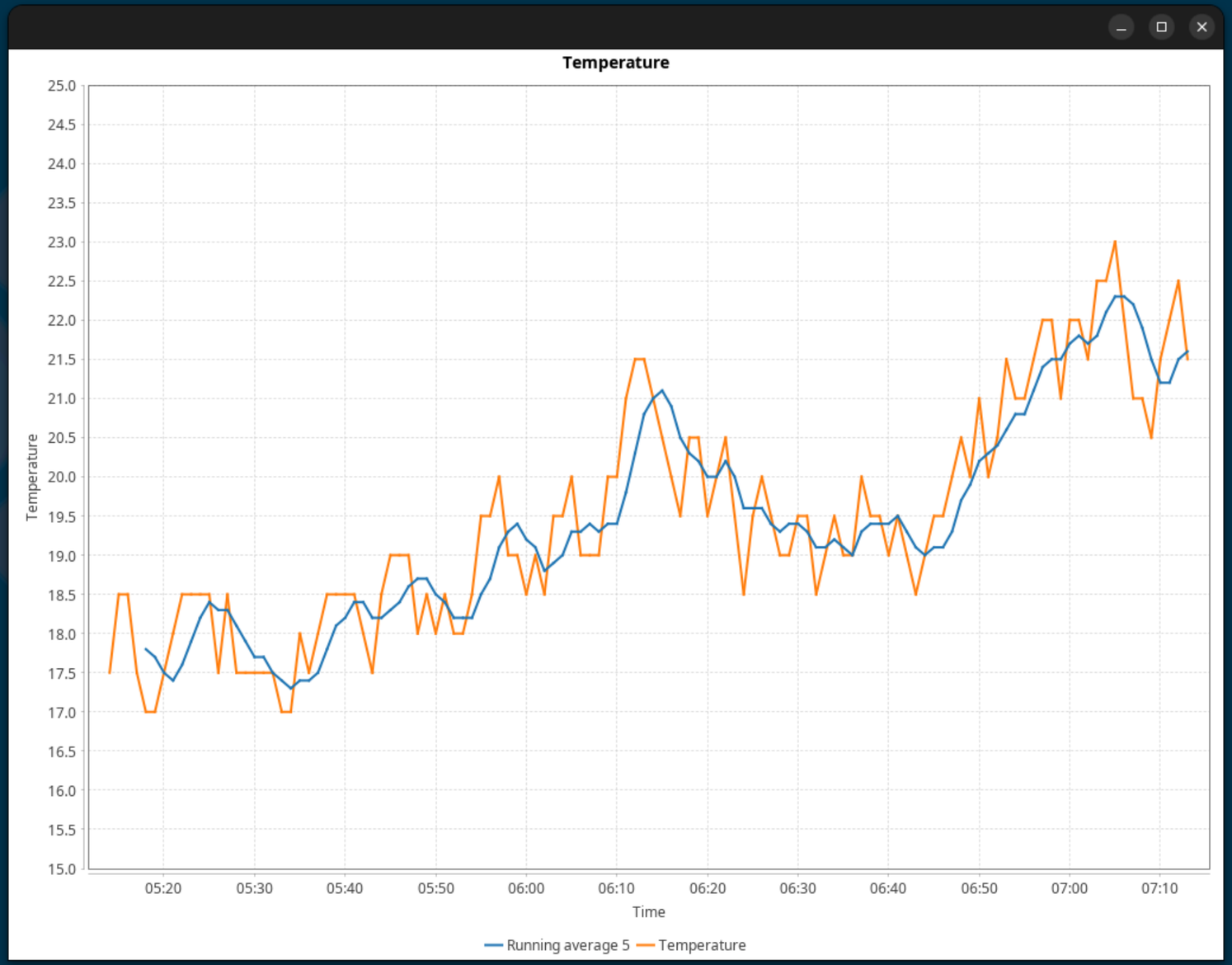

Temperature
25.0
24.5
24.0
23.5
23.0
22.5
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
Temperature
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
Time
Average Per Hour

DEFAULT GATHERERS

windowFixed(windowSize)

```
$ git checkout live-demo
```

EXAMPLE

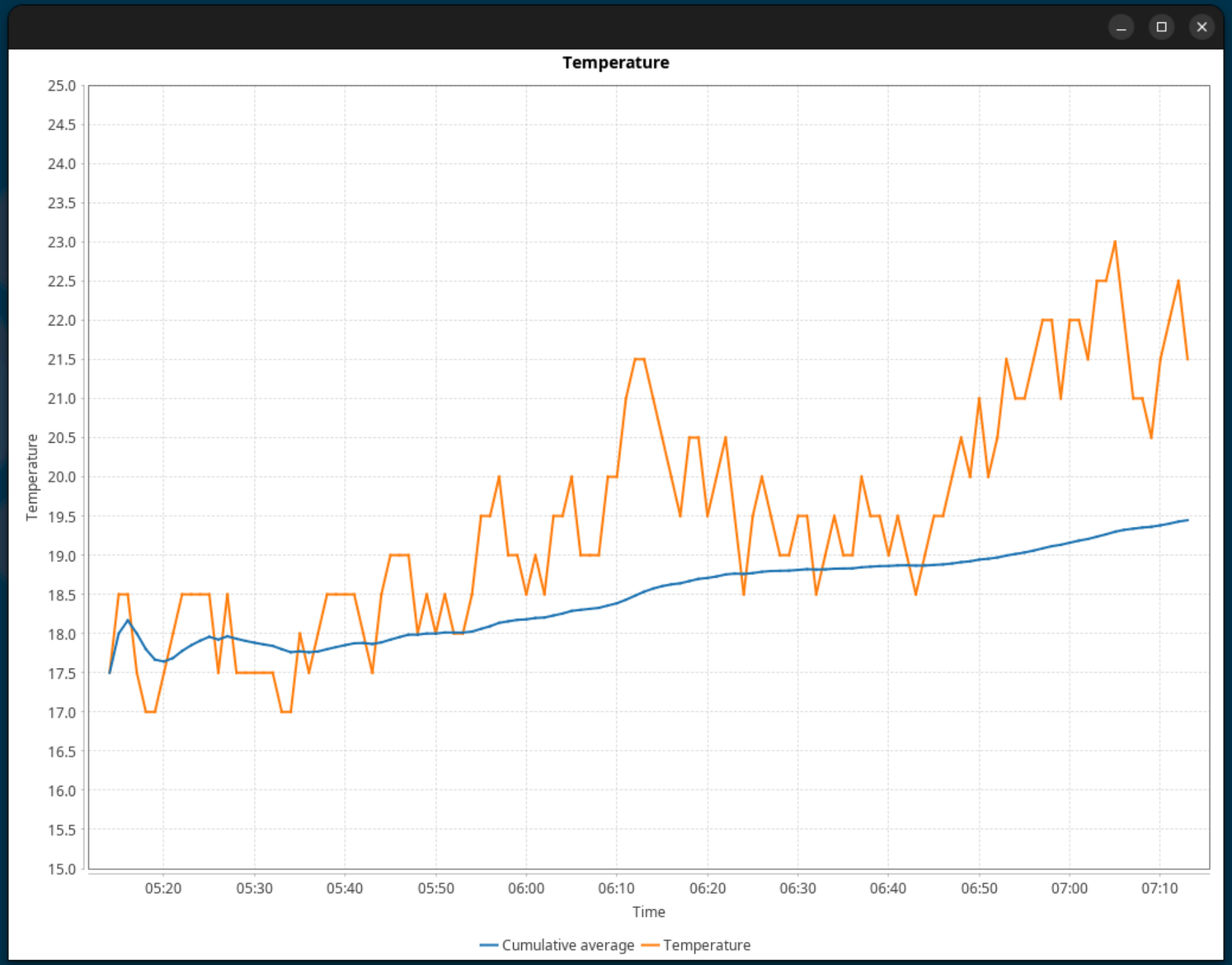

Temperature
25.0
24.5
24.0
23.5
23.0
22.5
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
Temperature
05:20
05:30
05:40
05:50
06:00
06:10
06:20
06:30
06:40
06:50
07:00
07:10
Time
Running average 5
Temperature

DEFAULT GATHERERS

windowSliding(windowSize)

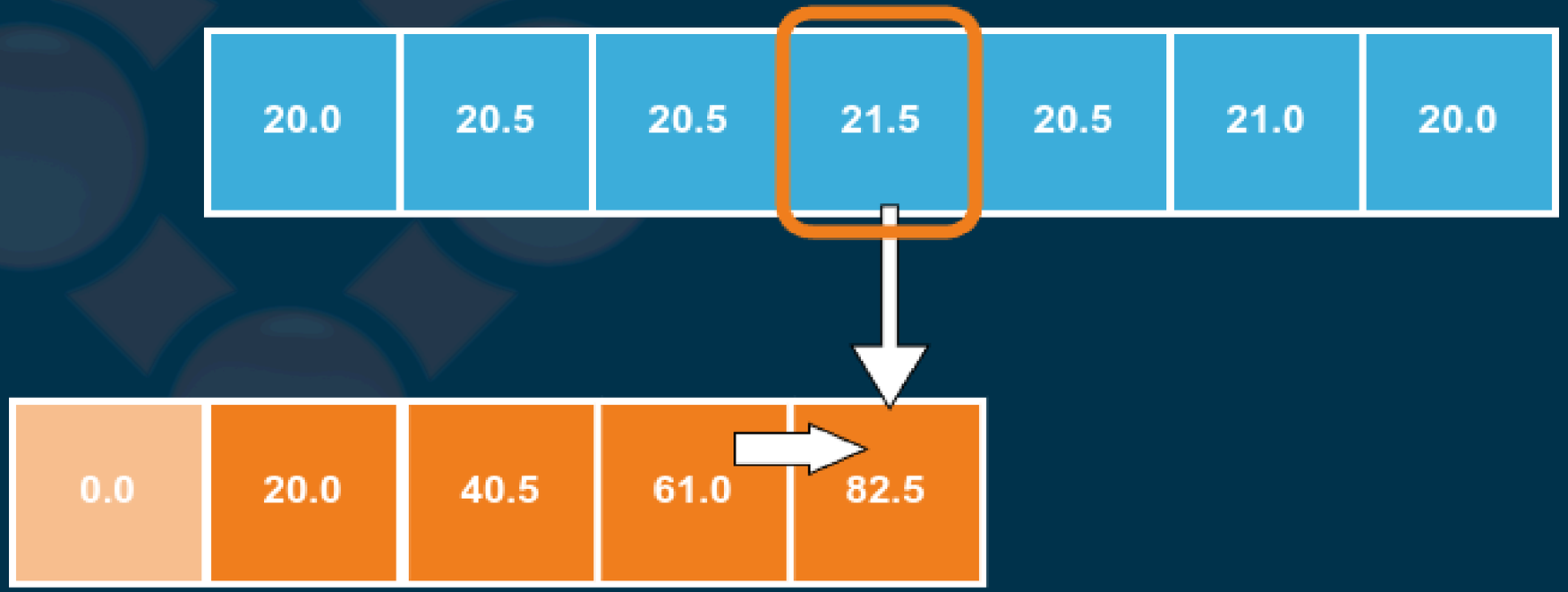
```
$ git checkout live-demo
```

EXAMPLE


Temperature
25.0
24.5
24.0
23.5
23.0
22.5
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
Temperature
05:20
05:30
05:40
05:50
06:00
06:10
06:20
06:30
06:40
06:50
07:00
07:10
Time
Cumulative average
Temperature

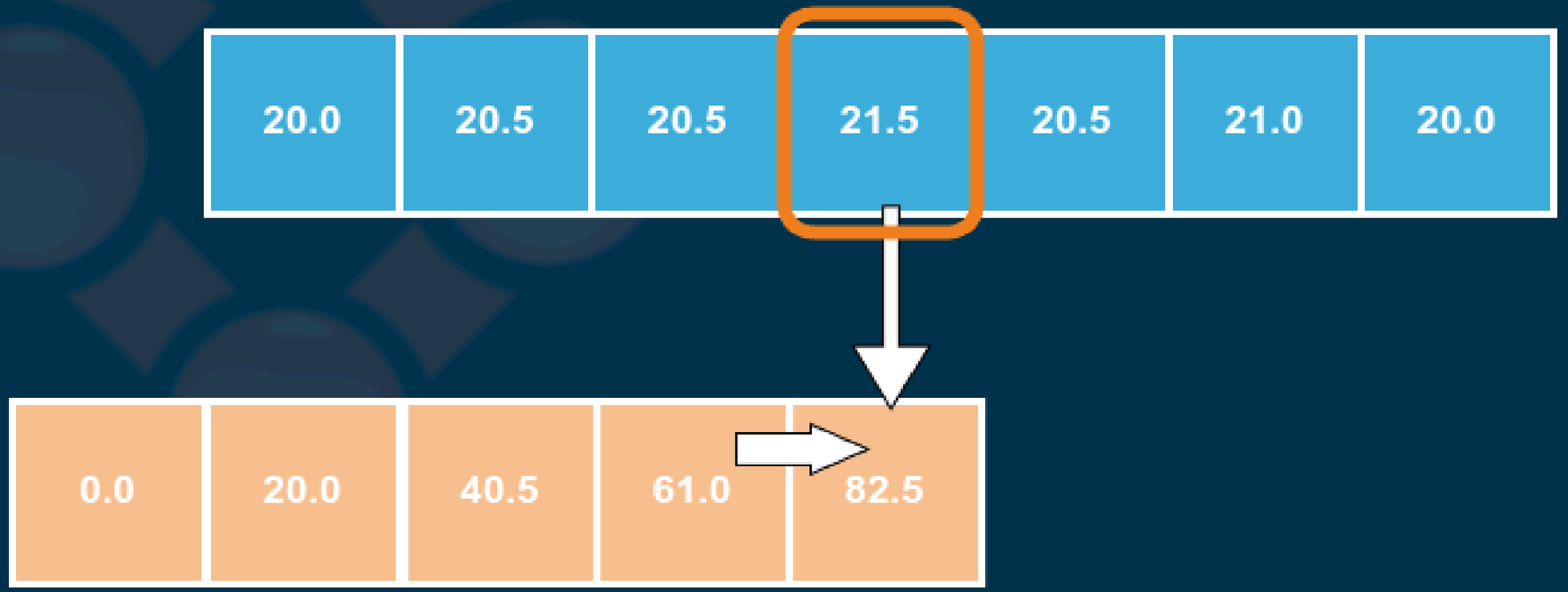
DEFAULT GATHERERS

scan(initial, scanner)



DEFAULT GATHERERS

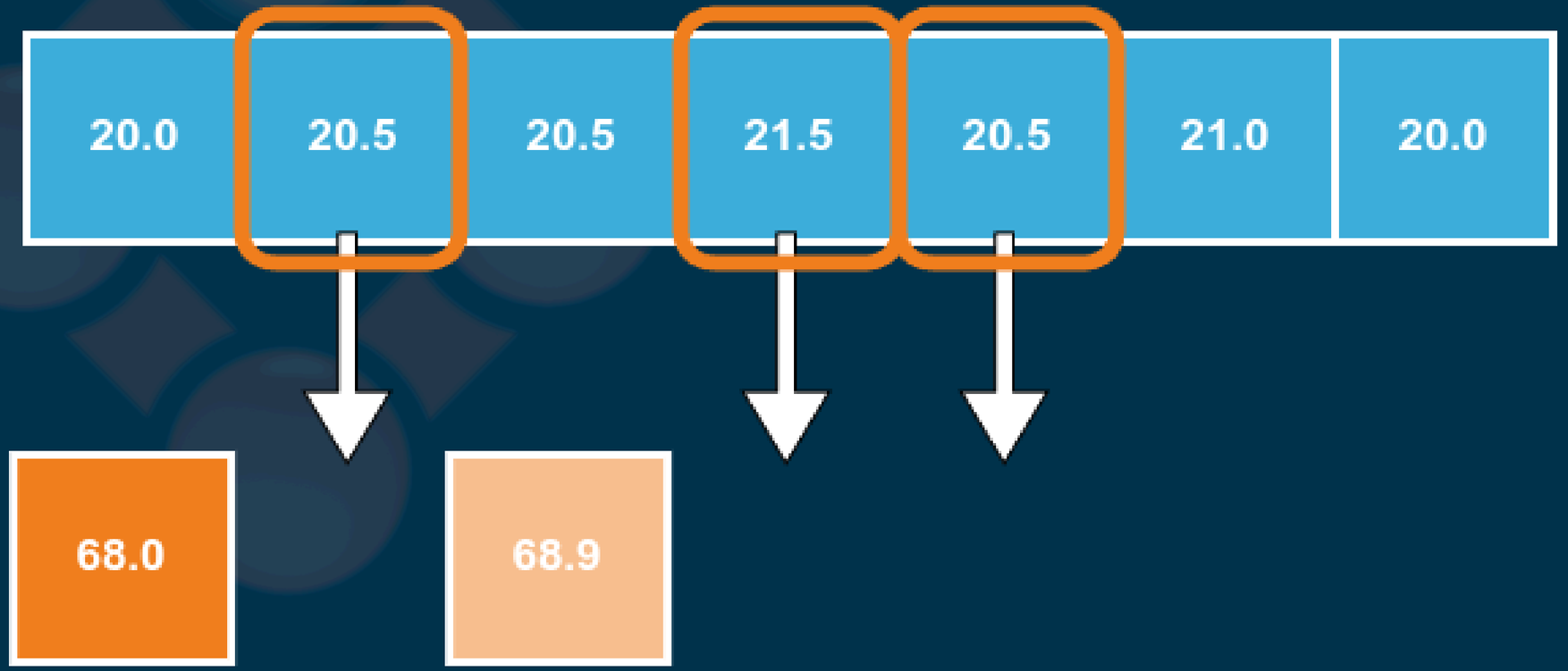
fold(initial, folder)



```
$ git checkout live-demo
```

DEFAULT GATHERERS

mapConcurrent(maxConcurrency, mapper)



ROLL YOUR OWN GATHERERS

DOWNSTREAM

```
interface Downstream<T> {
  boolean push(T element);
  boolean isRejecting();
}
```

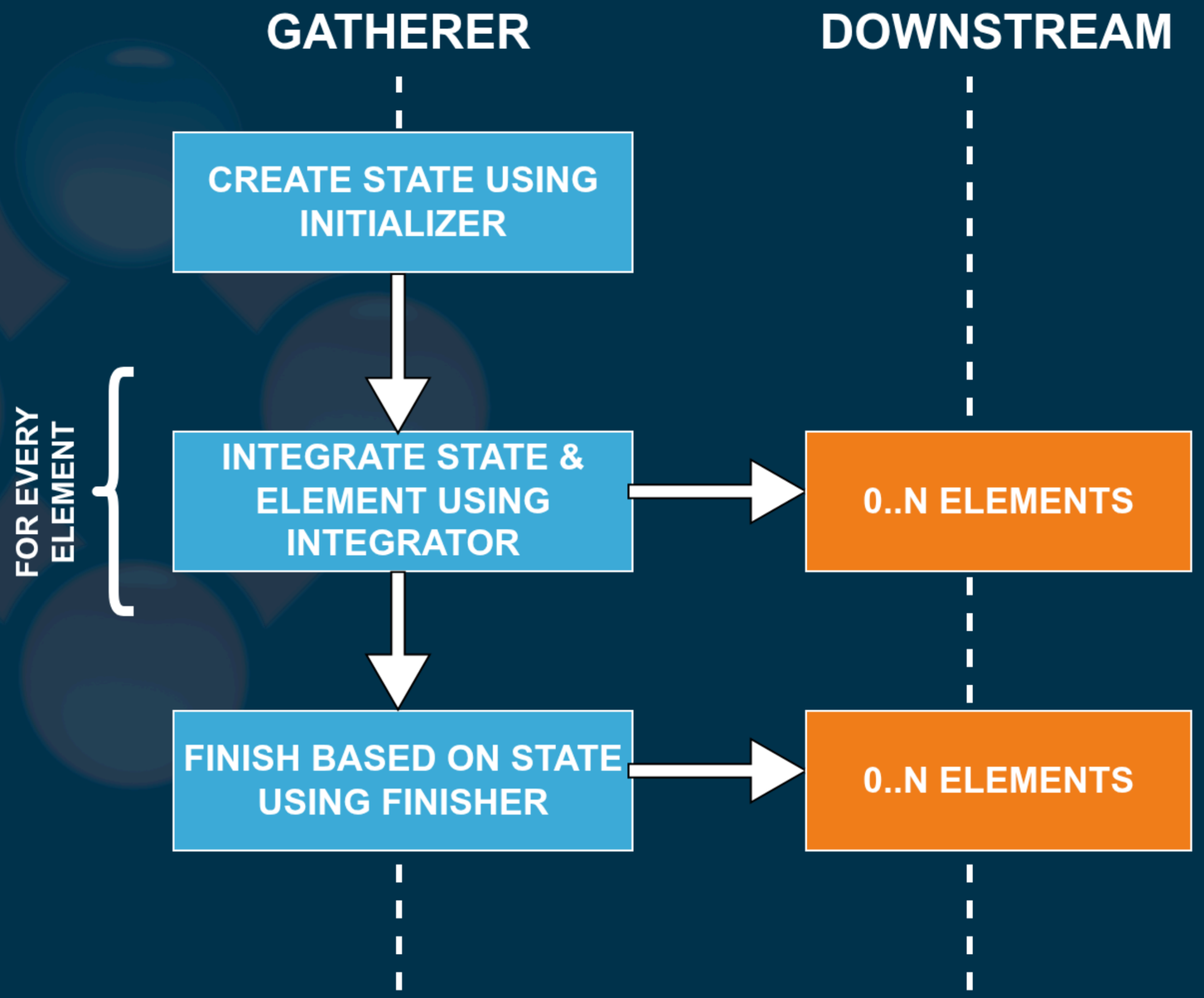
GATHERER

```
interface Gatherer<Input, State, Output> {
  Supplier<State> initializer(){};

  Integrator<State, Input, Output> integrator();
  // for parallel support
  BinaryOperator<State> combiner();

  BiConsumer<State, Downstream<Output>> finisher() {};
}
```

INTEGRATOR

```
interface Integrator<State, Input, Output> {
  boolean integrate(State state, Input elem, Downstream<Output> downstream);
}
```

FLOW


GATHERER
DOWNSTREAM
CREATE STATE USING INITIALIZER
FOR EVERY ELEMENT
INTEGRATE STATE & ELEMENT USING INTEGRATOR
0..N ELEMENTS
FINISH BASED ON STATE USING FINISHER
0..N ELEMENTS

FACTORY METHODS (SEQUENTIAL)

```
Gatherer.ofSequential(integrator); // state: Void

Gatherer.ofSequential(integrator, finisher);  // state: Void

Gatherer.ofSequential(initializer, integrator);  // state: Void

Gatherer.ofSequential(initializer, integrator, finisher);
```

FACTORY METHODS (PARALLEL SUPPORT)

```
Gatherer.of(integrator); // state: Void

Gatherer.of(integrator, finisher); // state: Void

Gatherer.of(initializer, integrator, combiner, finisher);
```

```
$ git checkout live-demo
```

GATHER MORE IDEAS

- segmentBy / windowBy
- filterIndexed, takeEvery, dropEvery
- collapseConsecutive
- distinctBy
- frequency
- midpointInterpolator
- ...
- Gatherers4J

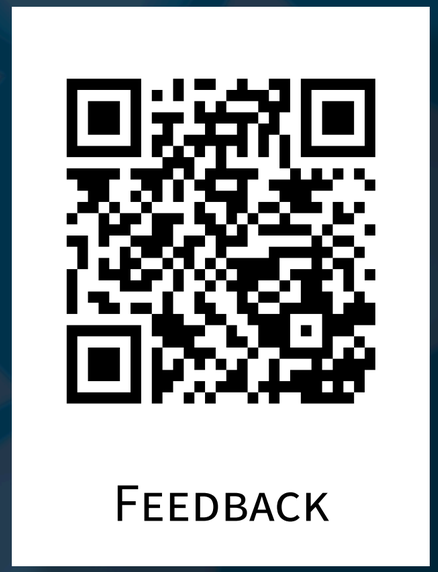
*Left as an exercise for the reader

WHAT WE GATHERED TODAY

- Gatherers are "Collectors" for intermediate operations
- Default gatherers:
  - `windowFixed`
  - `windowSliding`
  - `scan`
  - `fold`
  - `mapConcurrent`

WHAT WE GATHERED TODAY (2)

- When implement your own Gatherer
  - prefer factory methods
  - hide (internal) state
  - start with sequential
  - add parallel support if necessary



THANK YOU!

- Find me online
  - coduinix.bsky.social
  - @coduinix
  - hinseterschuur