Development of a fast Domain-Specific Language: A DSL for Stream Processing

Fall 2012 Semester Project Presentation

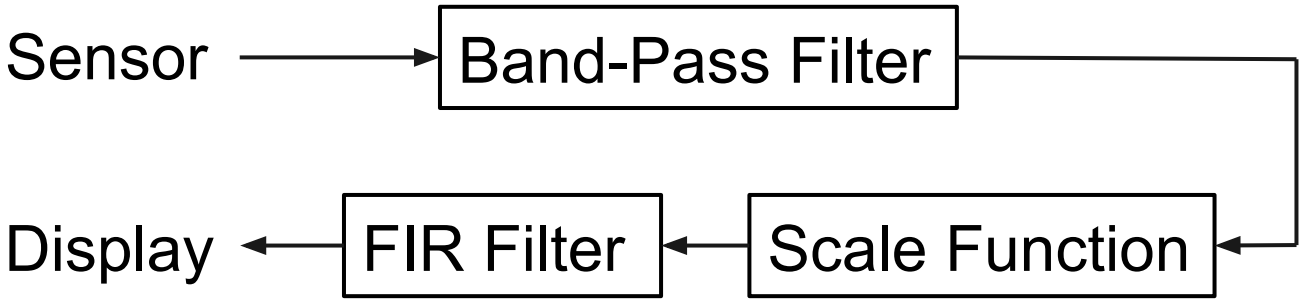
Student: Vera Salvisberg, Master IN, EPFL
Supervisor: Tiark Rompf, LAMP, EPFL
Date: January 17, 2012

Outline

- Motivation
- Scala backend
- API
- DBToaster
- Benchmarks
- LMS
- Future work
- Conclusion

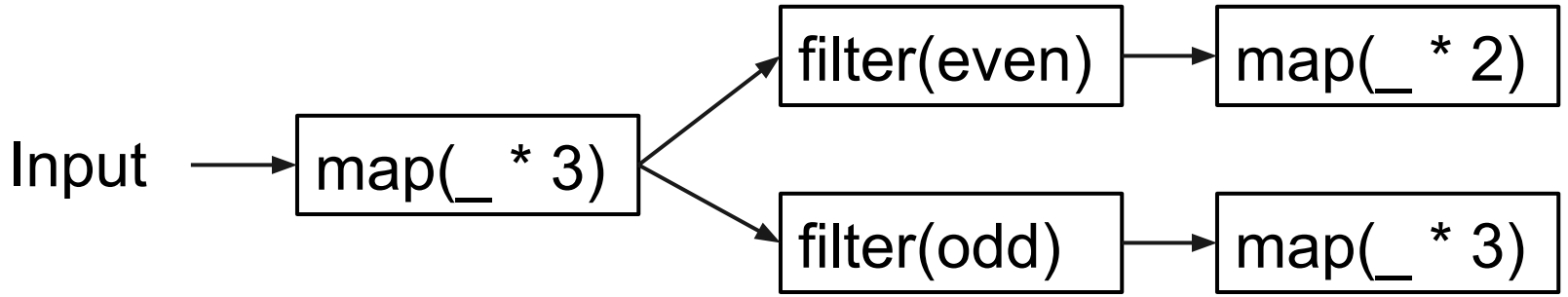
Stream Processing?

Example from Signal Processing:



Stream Processing?

Example from Math:



Scala Backend Code

```
abstract class StreamOp[A] {
  def onData(data: A)
  def flush
}

class MapOp[A, B](f: A => B, next: StreamOp[B])
    extends StreamOp[A] {
  def onData(data: A) = next.onData(f(data))

  def flush = next.flush
}
```

Scala Backend Code

```
def zipWithoutFlush[A, B] (next: StreamOp[Pair[A, B]]):
    (StreamOp[A], StreamOp[B]) = {
  val leftBuffer = new Queue[A]; val rightBuffer = new Queue[B]

  val left = new StreamOp[A] { def onData(data: A) = {
      if (rightBuffer.isEmpty) leftBuffer += data
      else next.onData(data, rightBuffer.dequeue)
    }
  }
  val right = new StreamOp[B] {..

  (left, right)
}
```

Example Usage → API

```
new ListInput(List.range(0, 6),
    new MapOp({x: Int => 3 * x},
        new DuplicateOp(
            new FilterOp({x: Int => x % 2 == 0},
                new MapOp({x: Int => 2 * x + " (even)"},
                    new PrintlnOp)),
            new FilterOp({x: Int => x % 2 == 1},
                new MapOp({x: Int => 3 * x + " (odd)"},
                    new PrintlnOp)))))
```

```
0  (even)
9  (odd)
12  (even)
27  (odd)
24  (even)
45  (odd)
```

API code:

```
new ListInput(List.range(0, 6),
    Stream[Int] map {3 * _} duplicate (
        Stream[Int] filter {_ % 2 == 0} map {2 * _ + " (even)"} print,
        Stream[Int] filter {_ % 2 == 1} map {3 * _ + " (odd)"} print))
```

Stream Composition

```
// Creating a StreamOp:
val stream01 = new MapOp({x: Int => 2 * x}, new PrintlnOp)
// Adding on the left:
val stream02 = new FilterOp({x: Int => x % 2 == 0}, stream01)
// Cannot add on right, stream finishes with PrintlnOp
```

Remember:

```
class MapOp[A, B](f: A => B, next: StreamOp[B])
    extends StreamOp[A] {
  def onData(data: A) = next.onData(f(data))

  def flush = next.flush
}
```

API Stream Composition

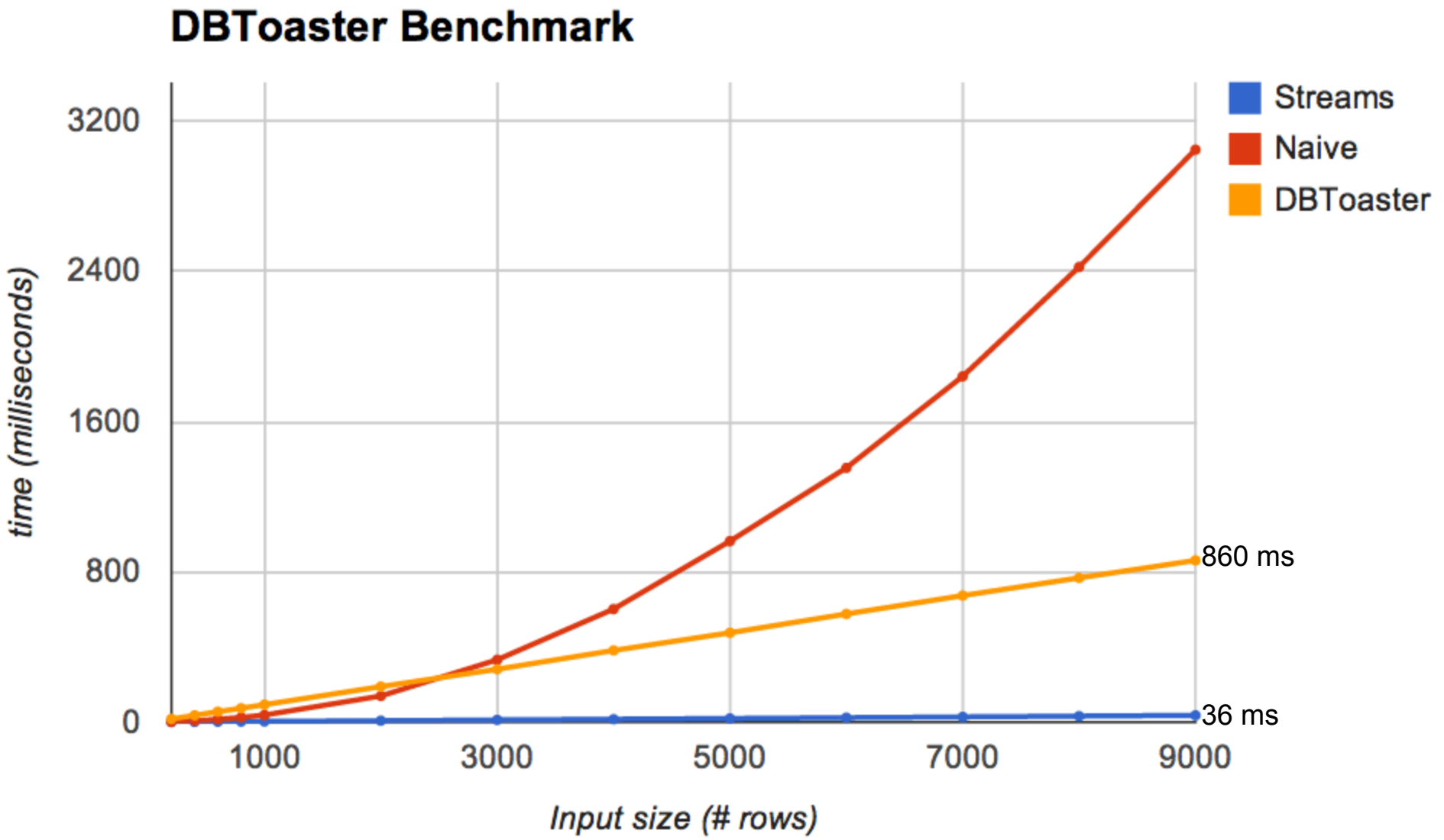
```
// Creating a Stream of Ints:
val stream1: Stream[Int, Int] = Stream[Int] map {3 * _}
// Adding operations on the right:
val stream2: Stream[Int, Int] = stream1 map {_ - 1}
// Creating another Stream of Ints:
val stream3: Stream[Int, Int] = Stream[Int] filter {_ % 2 == 0}
// Adding it on the left:
val stream4: Stream[Int, Int] = stream3 into stream2
// Instantiating the StreamOps:
val streamOp: StreamOp[Int] = stream4.print
// Use the StreamOps:
new ListInput(List.range(0, 6), streamOp)
```

Prints: `-1, 5, 11`

Code

DBToaster

```
SELECT returnflag, linestatus,
  SUM(quantity) AS sum_qty,
  SUM(extendedprice) AS sum_base_price,
  SUM(extendedprice * (1-discount)) AS sum_disc_price,
  SUM(extendedprice * (1-discount)*(1+tax)) AS sum_charge,
  AVG(quantity) AS avg_qty,
  AVG(extendedprice) AS avg_price,
  AVG(discount) AS avg_disc,
  COUNT(*) AS count_order
FROM lineitem
WHERE shipdate <= DATE('1997-09-01')
GROUP BY returnflag, linestatus;
```



Stream versions

LMS example

- Double instead of generic type
- scale and general map
- function combination

```
def test(s: Rep[DoubleStream]): Rep[DoubleStream] =
  map(map(s, {(x: Rep[Double]) => Math.pow(unit(2.0), x)}),
      {(x: Rep[Double]) => x + unit(3.0)})
```

```
// Without optimization:
class Test extends (SDD=>SDD) {
  def apply(x0:SDD): SDD = {
    val x3 = {x1: (Double) =>
      val x2 = Math.pow(2.0,x1)
      x2: Double
    }
    val x4 = x0.map(x3)  // MapOp
      val x7 = {x5: (Double) =>
      val x6 = x5 + 3.0
      x6: Double
    }
    val x8 = x4.map(x7)  // MapOp
    x8
  }
}
```

```
// With optimization:
class TestOpt extends (SDD=>SDD) {
  def apply(x0:SDD): SDD = {
    val x8 = {x5: (Double) =>
      val x6 = java.lang.Math.pow(2.0,x5)
      val x7 = x6 + 3.0
      x7: Double
    }
    val x11 = x0.map(x18) // MapOp
    x11
  }
}
```

where SDD = Stream[Double, Double]

Future work

LMS:

- function combination
- buffer removal
- split-merge removal
- parallelization

Delite?

Conclusion

Thank you for your attention!

Contact: vera.salvisberg@epfl.ch
Code: https://github.com/vsalvis/DslStreams

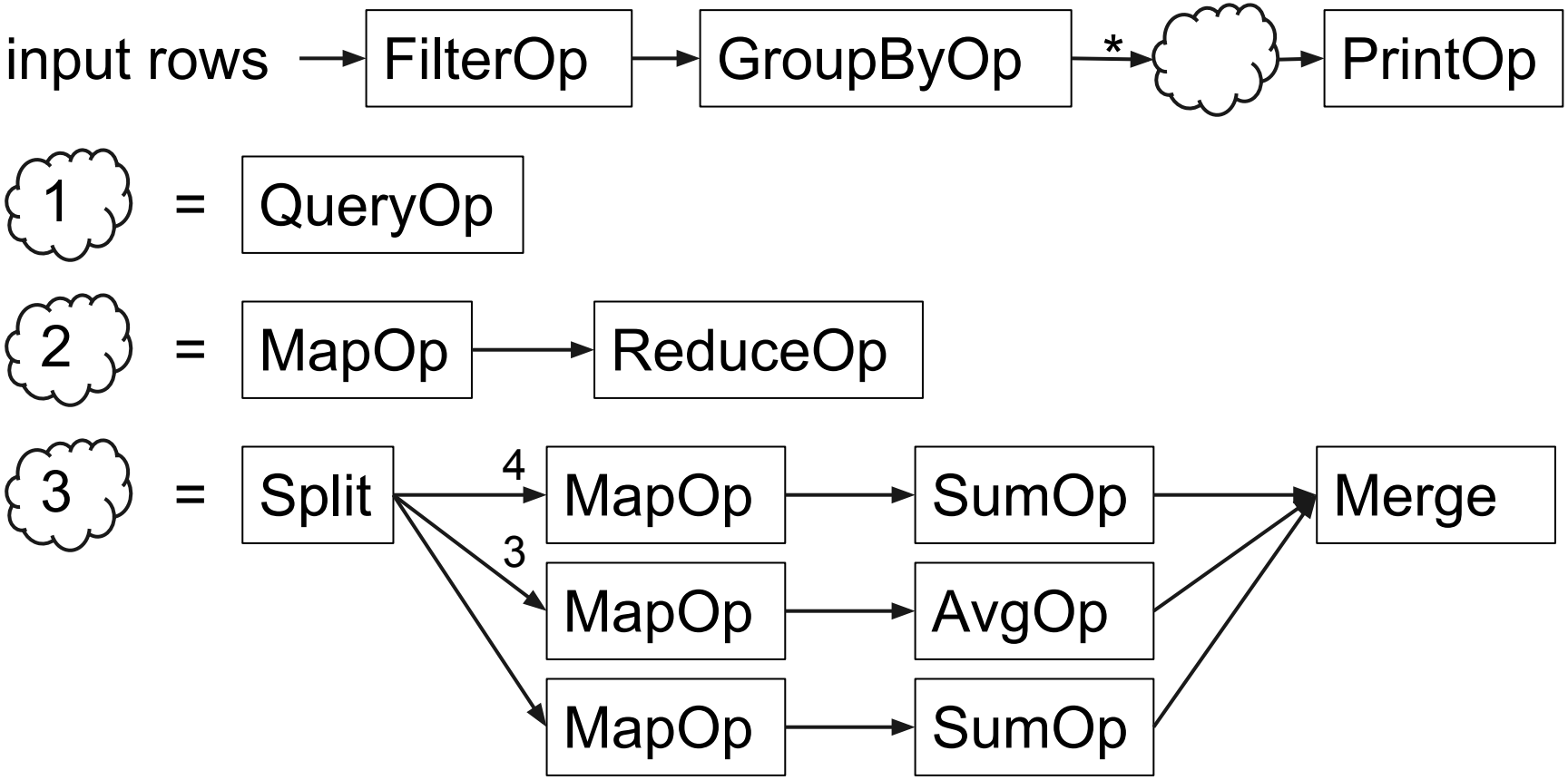
Questions?

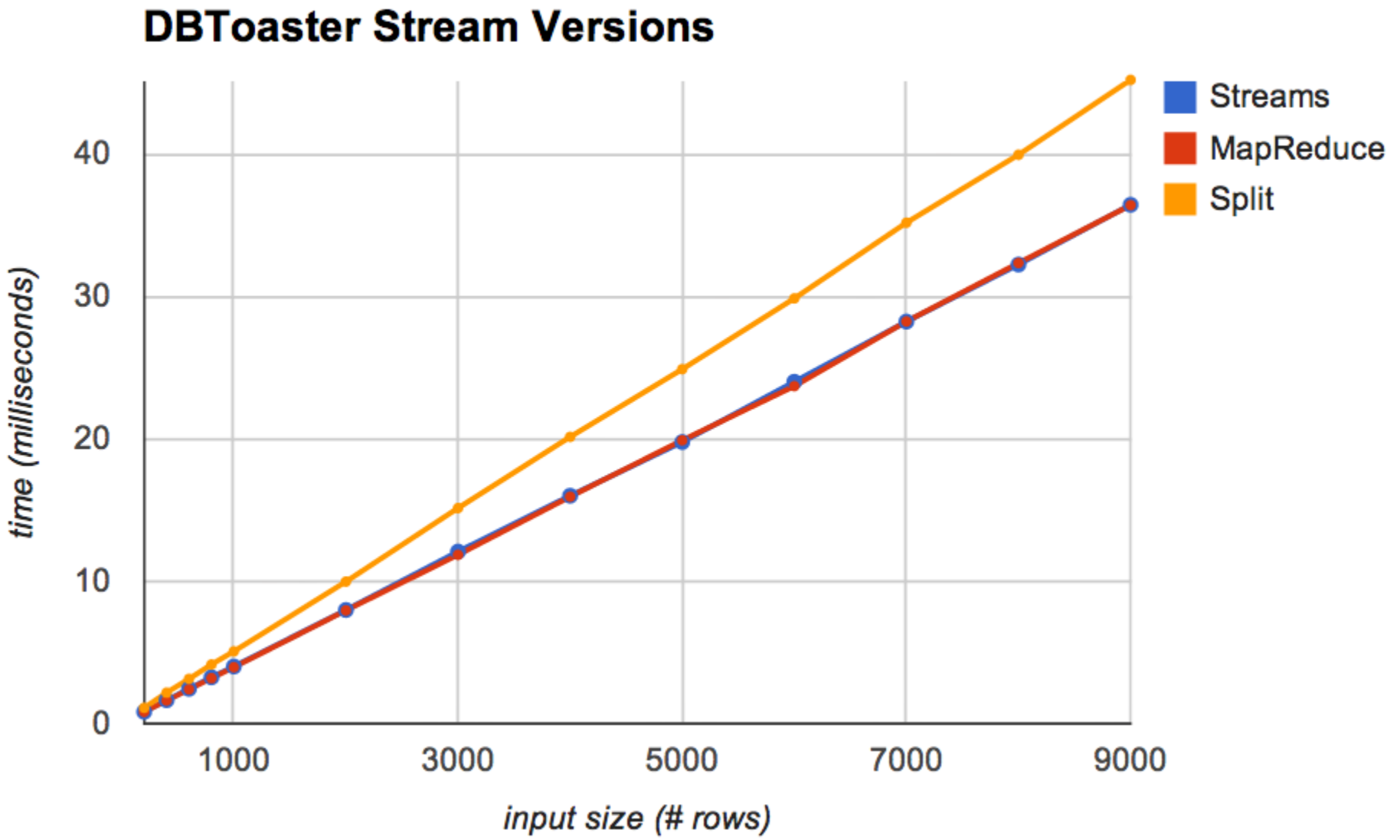
API code

```
abstract class Stream[A,B] { self =>
  def into(out: StreamOp[B]): StreamOp[A]
  def map[C](f: B => C) = new Stream[A,C] {
    def into(out: StreamOp[C]) = self.into(new MapOp(f, out))
  }
  def print = into(new PrintlnOp[B]())
  def into[C](next: Stream[B, C]): Stream[A, C] = new Stream[A, C] {
    def into(out: StreamOp[C]) = self.into(next.into(out))
  }
}
object Stream {
  def apply[A] = new Stream[A,A] {
    def into(out: StreamOp[A]): StreamOp[A] = out
  }
}
```

back

DBToaster Stream versions




DBToaster Stream Versions
Streams
MapReduce
Split
time (milliseconds)
0
10
20
30
40
1000
3000
5000
7000
9000
input size (# rows)

back