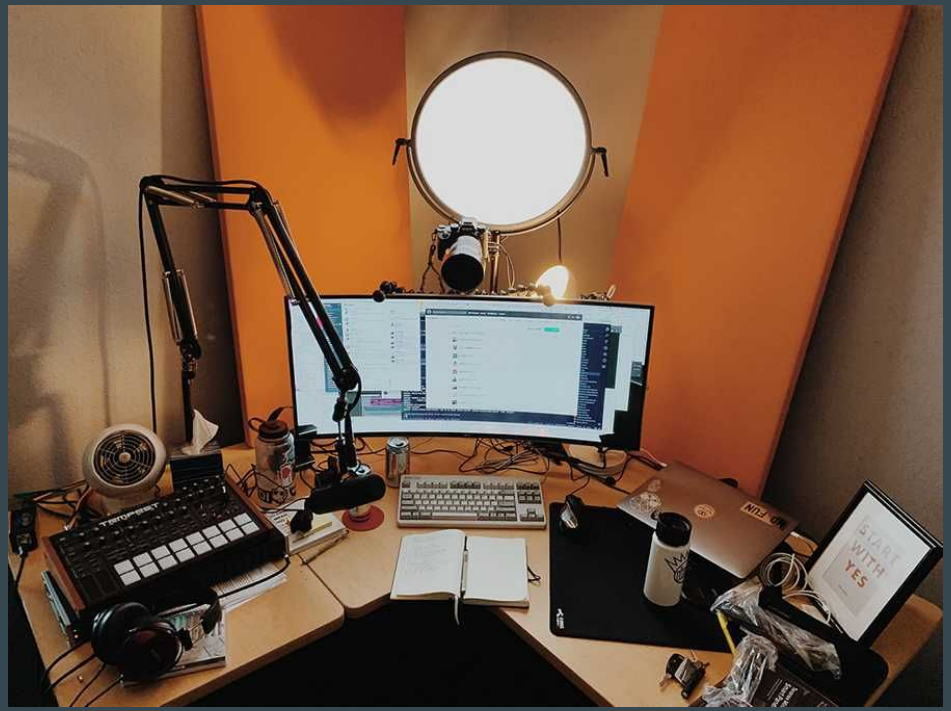
Akka Stream Processing

Khinshan Khan & Adeeb Rahman

A Familiar Introduction

Think of a Streamer

- streamer vs viewer
- they send audio/ video
- viewers get the data real-time
- viewer device interprets data
- no idea when the streamer ends



Stream Processing

Stream Processing [Formal]

- processing a number (possibly infinite) of elements
- by pushing/pulling them through a “pipeline”
- such pipeline is composed of operations that modify the elements *functionally*
- operations often expressed as DSL similar to Scala collections (map, flatMap, filter)

Stream Processing Motivation

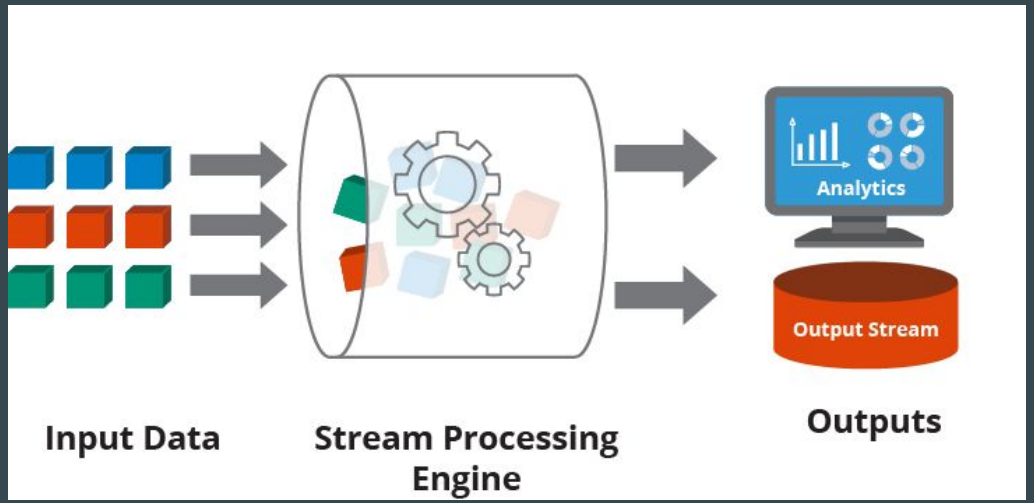
- lots of applications are about processing data
- data sources can be intermittent or unbounded
- data has to flow throughout distributed nodes
- stream processing = manipulation of data whose sources are intermittent and potentially unbounded

Stream Processing Goals

- compositional building-blocks
- handle flow-control through such stream pipeline
- process many, possibly infinite, elements at optimal rate

Real World Use Cases

- tweet stream analysis
- analysis of logs from a distributed system
- chat systems, merging many incoming messages to a chat feed



Potential Challenges

- resource efficiency
- flow controlled processing
- failure handling

- reactive streams were designed to provide a way for exchanging streams of data
- streams are defined by two main objects a “publisher” and a “subscriber”
- a publisher is the information provider
- a subscriber can subscribe to a publisher and uses the information as necessary.
- there also exists “processors” which essentially serve as a middle man to further manipulate data streams.

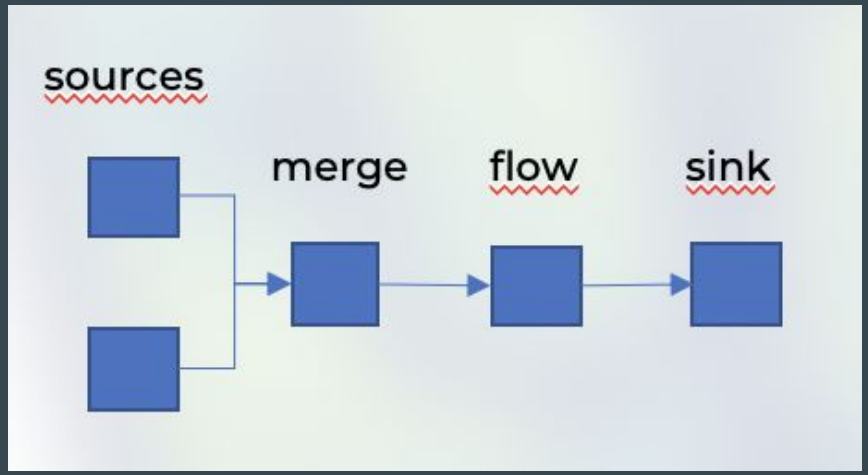
Flow-control

- flow-control is a mechanism that allows us to control the rate at which a subscriber receives information from a publisher
- when a subscriber has too much information from a publisher it can:
  - tell the publisher to wait to send data
  - ignore new data
  - purge the oldest data in the buffer
  - purge the newest data in the buffer
  - purge the entire buffer
  - fail

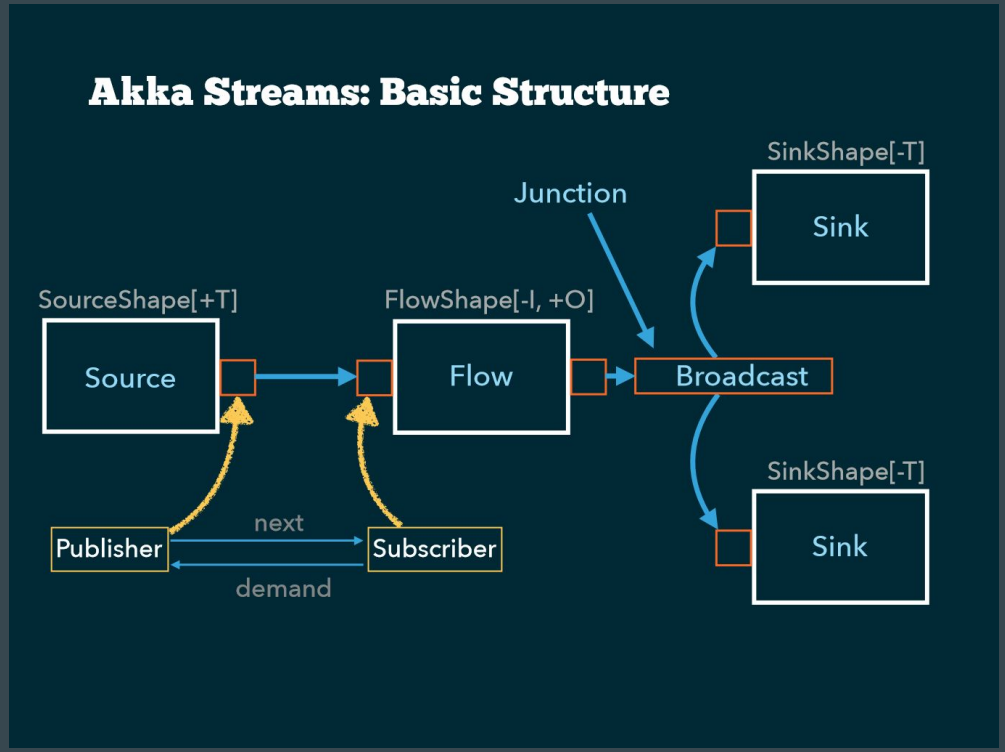
Akka Streams

Akka Streams

- Akka Streams provides a high-level API for streams processing.
- Akka Streams function through the actor system
- Publisher -> Source (one output)
- Processor -> Flow (one input and one output)
- Subscriber -> Sink (one output)



Akka Streams Visualized



Getting Started with Akka Streams

Akka Docs: https://doc.akka.io/docs/akka/current/stream/index.html

To use Akka Streams, add the module to your project (build.sbt):

```
val AkkaVersion = "2.6.10"

libraryDependencies += "com.typesafe.akka" %% "akka-stream" % AkkaVersion
```

Example Code

```
import akka.stream._

import akka.stream.scaladsl._

import akka.actor.ActorSystem

implicit val system = ActorSystem()

Source(1 to 20).runWith(Sink.foreach(println))
```

https://asciinema.org/a/8GBuTl8mmZ4fCwmoSxPWRnkJO

Basic Processor Usage with map and filter

```
// double every digit

Source(1 to 10).map(_ * 2).runWith(Sink.foreach(println))

// print evens

Source(1 to 10).filter(_ % 2 == 0).runWith(Sink.foreach(println))

// print odds

Source(1 to 10).filter(_ % 2 == 1).runWith(Sink.foreach(println))
```

https://asciinema.org/a/Ps4FhqNWEx9Uw9OXewSOVDDDy

Modular Usage / Composition of Parts

```
val numbers = Source(1 to 10)

val doubling = Flow.fromFunction((x: Int) => x * 2)

val printer = Sink.foreach(println)

numbers.via(doubling).runWith(printer)


numbers.via(doubling).via(doubling).runWith(printer)
```

https://asciinema.org/a/h1O4nbXMiizjDpbTOsxcZPBAg

Asynchronous Operators

```
Source(1 to 3)
  .map { i => println(s"A: $i"); i }.async
  .map { i => println(s"B: $i"); i }.async
  .map { i => println(s"C: $i"); i }.async
  .runWith(Sink.ignore)
```

https://asciinema.org/a/KScJ7xu1i7qrsnsuykjykODEm

Compounded Results with Futures

```
val numbers = Source(1 to 10)

val sum = Sink.fold(0)((acc: Int, x: Int) => acc + x)

val result = numbers.runWith(sum)

val prod = Sink.fold(1)((acc: Int, x: Int) => acc * x)

val result2 = numbers.runWith(prod)
```

https://asciinema.org/a/1tVzlok5CCpguV1ik74AN8KAx

Flow-control

```
val numbers = Source(1 to 100)

val printer = Sink.foreach(println)

numbers.buffer(10, OverflowStrategy.dropNew).runWith(printer)

numbers.buffer(10, OverflowStrategy.dropNew).throttle(1, 10.millisecond).runWith(printer)
```

https://doc.akka.io/japi/akka/current/akka/stream/OverflowStrategy.html

https://asciinema.org/a/fUVL61yewN2BLuobDbFGqF55j