



# IBM Streams

In-Motion Analytics Platform @IBM

Manoj Singh  
Big Data & Streams Architect  
[manoj.singh101@ibm.com](mailto:manoj.singh101@ibm.com)



## Agenda

- **Streaming Problems**
- Introduction to IBM Streams
- Solutions using IBM Streams

## Streaming Problems

<b>Variety</b>	<b>Data in Many Forms</b> - Structured, unstructured, text, multimedia <b>Data from Many Sources</b> – Sensors, DB, Routers, Logs, Cameras, Medical Devices
<b>Velocity</b>	<b>Data in Motion</b> – Unbounded stream, never stops, continuous stream of incremental analytics and output results, keep up with ingest data rates
<b>Volume</b>	<b>Data at Rest</b> - Scale from terabytes to zettabytes, growing continuously, too big to store and process
<b>Veracity</b>	<b>Data in Doubt</b> - Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

## Streaming Problems - cont'd

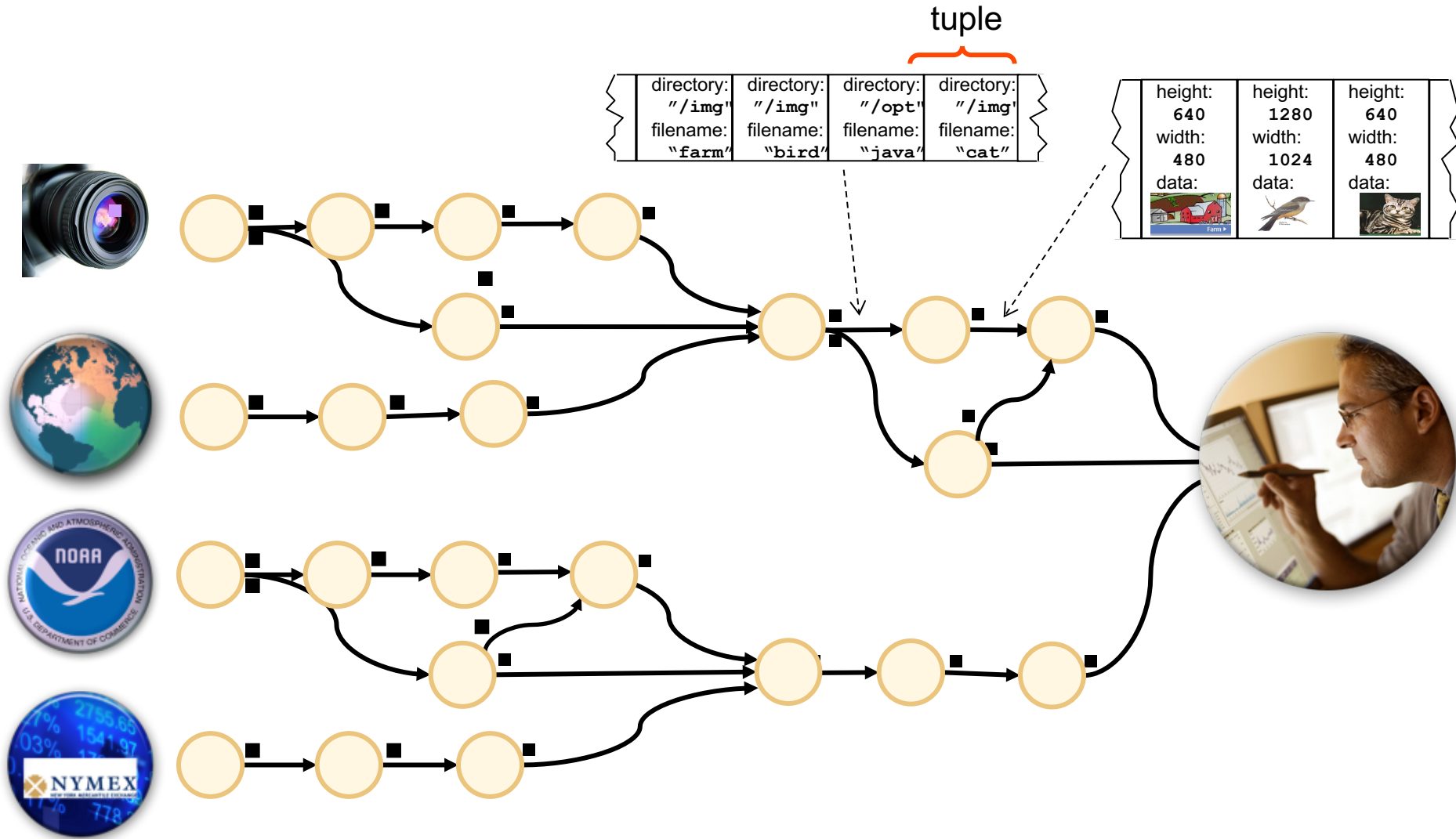
<b>Low Latency</b>	Deliver insights with microsecond latencies, Information can be stale in seconds, maximize end-to-end throughput, and minimize processing and communication latency.
<b>Resiliency</b>	Need to be always up, highly available, fault tolerant, seamless automatic recovery required, information loss while application is down
<b>Consistency</b>	Guaranteed processing, need to cope with data loss, corruption, reordering, state management
<b>Agility</b>	Single instance can support multiple applications, Support multi-team development, Incremental development and deployment
<b>Resource Adaptation</b>	Should be able to optimize resources, reconfigure hardware, repurpose resources to optimize processing



## Agenda

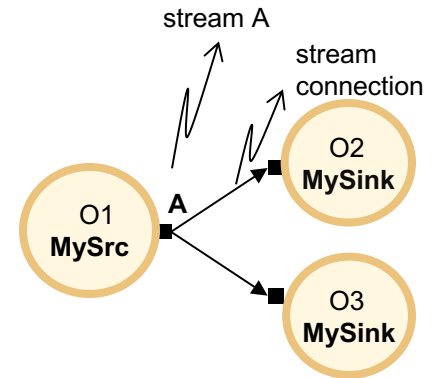
- Streaming Problems
- **Introduction to IBM Streams**
- Solutions using IBM Streams

# Stream Computing Illustrated



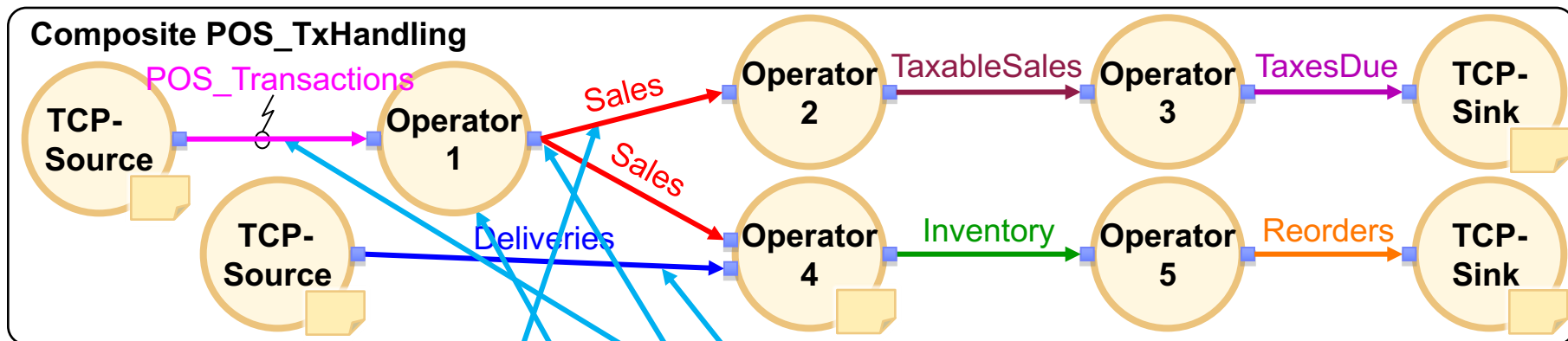
## Terminology

- Application
  - Data flow graph of *operator instances* connected to each other via *stream connections*
- Operator
  - Reusable stream analytic
    - Input ports: receives data / Output ports: produces data
    - Source: No input ports / Sink: No output ports
- Operator Instance
  - A specific instantiation of an operator
- Stream
  - Continuous series of tuples, generated by an operator instance's output port
- Stream connection
  - A stream connected to a specific operator instance input port
- PE
  - A runtime *process* that executes a set of operator instances
- Job
  - An application instance running on a set of hosts



```
(stream<Type> A) as O1 = MySrc() {}  
( ) as O2 = MySink(A) {}  
( ) as O3 = MySink(A) {}
```

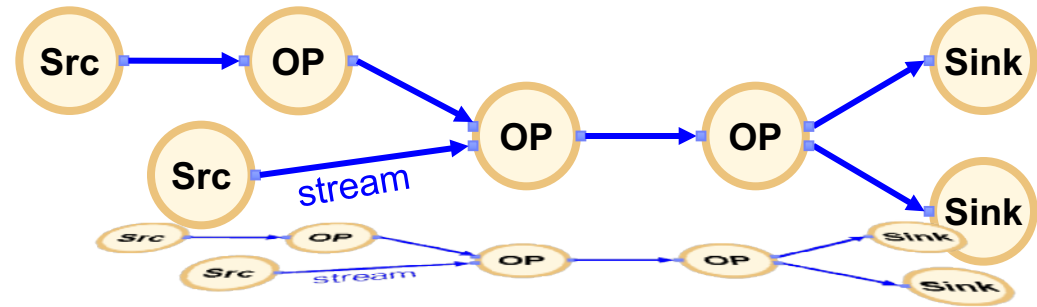
## Composing a Flow Graph with Stream Definitions



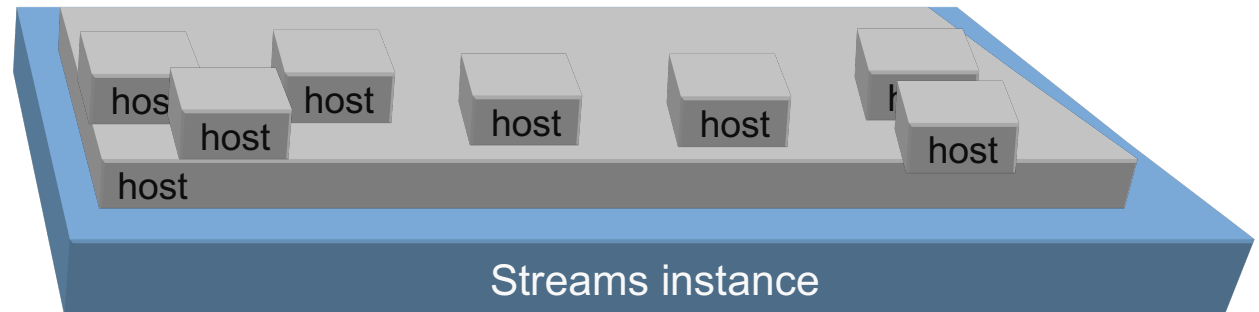
```

composite POS_TxHandling
{
  graph
    stream<...> POS_Transactions = TCPSource() {...}
    stream<...> Sales = Operator1(POS_Transactions) {...}
    stream<...> TaxableSales = Operator2(Sales) {...}
    stream<...> TaxesDue = Operator3(TaxableSales) {...}
    () as Sink1 = TCPSink(TaxesDue) {...}
    stream<...> Deliveries = TCPSource() {...}
    stream<...> Inventory = Operator4(Sales, Deliveries) {...}
    stream<...> Reorders = Operator5(Inventory) {...}
    () as Sink2 = TCPSink(Reorders) {...}
}
  
```

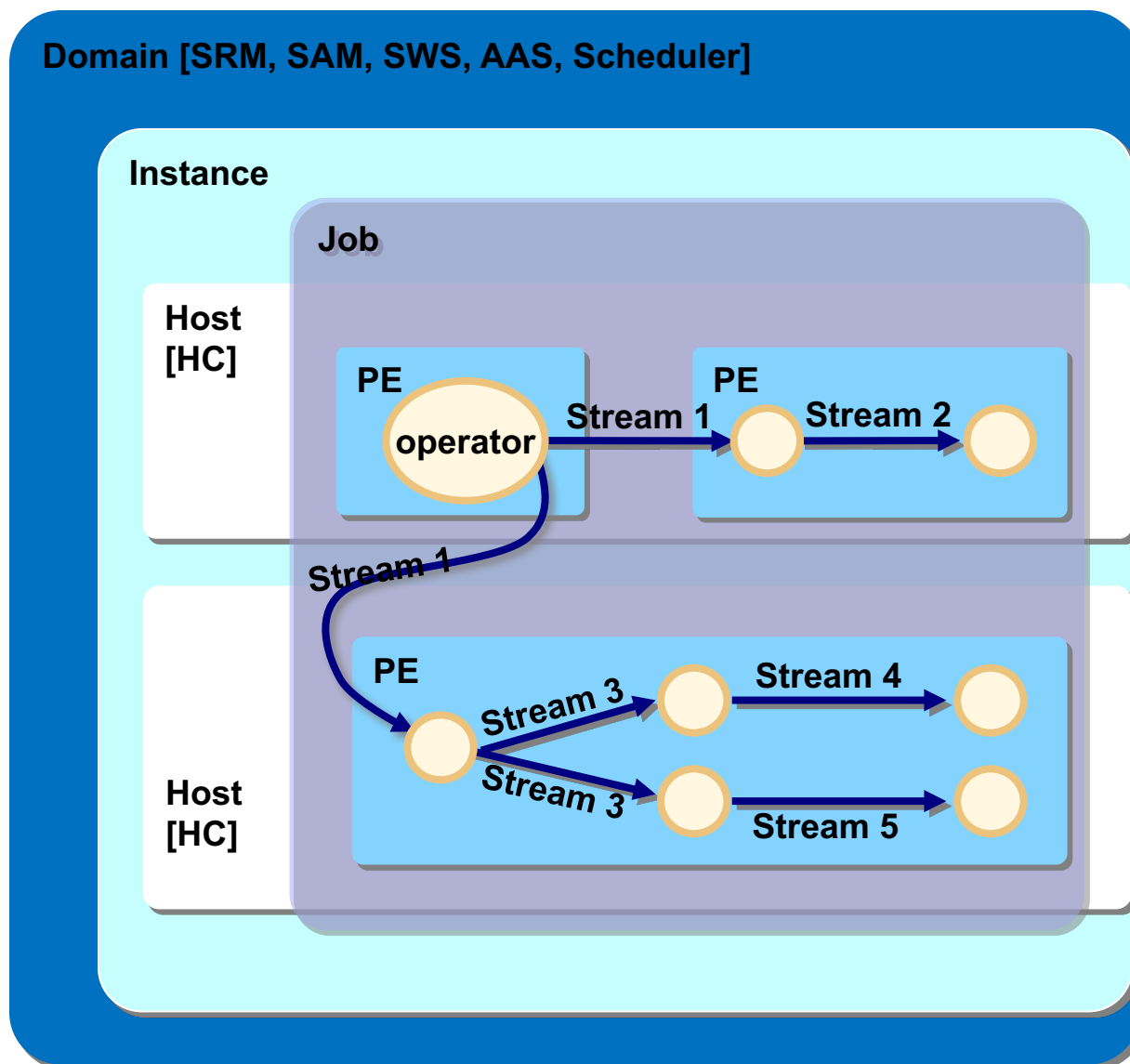
- Streams application graph:



- Each complete application is a potentially deployable job
- Jobs are deployed to a Streams runtime environment
  - known as a Streams Instance
- An instance can include a single host (hardware)
- Or multiple hosts



## Runtime Components



File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access InfoSphere Streams

Projec Strea \*GeofenceMarketing.spl

Find

To begin: Drag and drop an item from the palette to the canvas.

Design

- Composite
- Input Port
- Operator
- Output Port
- Stream

Toolkits

- spl [1.2.0]
- AdminDisplayApp [1.0.0]
- GeoFenceMarketing [1.0.0]
- Practice [1.0.0]
- com.ibm.streams.c
- com.ibm.streams.d
- com.ibm.streams.d
- com.ibm.streams.d

Layers:

- ☒ Consistent Region
- ☒ Note Annotation
- ☒ Validation

Color Schemes:

- ☒ Category
- ☐ Consistent Region
- ☐ Host Colocation
- ☐ Operator Type

Console Problems Properties Instance Graph Metrics

GeoAdvertising@TargetedAdvertising [Running, 4:04:21 AM] - ...althy, 0 with alerts; 8 Hosts (0 unhealthy, 0 with alerts)

Operator: com.ibm.streamsx.hbase::HBASEGet GetOfferBasedOnInterest [Healthy, No Alerts]

Operator Metrics:

No metrics defined

Tuple Flows:

- Job: 87.PE:532.GetOfferBasedOnInterest.Input[0]: Total tuples: [131869668, 131869668, 0] Rate: 3199/sec (over 3 seconds ending at 4:04:09 AM)
- Job: 87.PE:532.GetOfferBasedOnInterest.Output[0]: Total tuples: [131869667, 131869667, 0] Rate: 3199/sec (over 3 seconds ending at 4:04:09 AM)

Alerts and Health Issues:

none

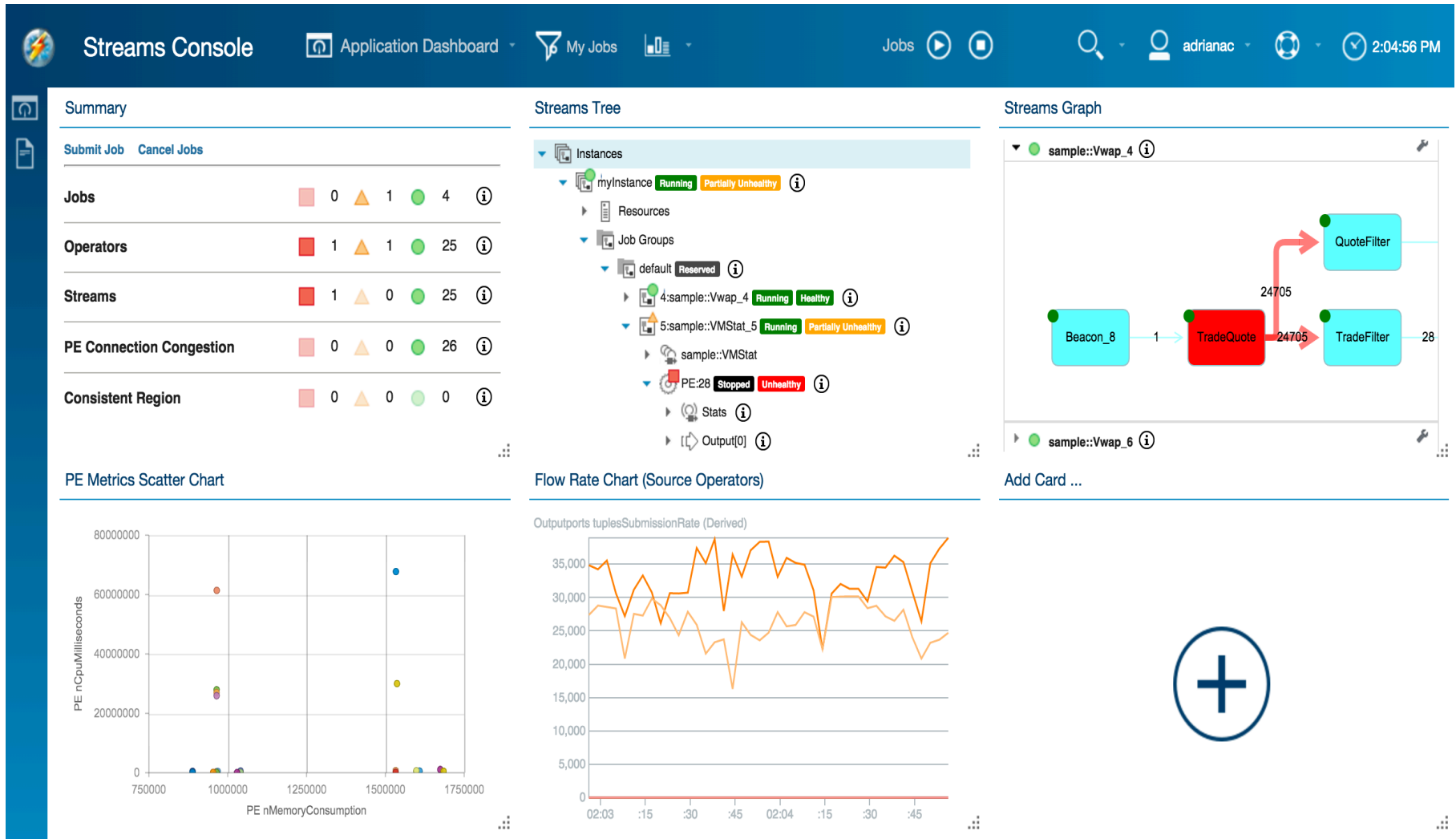
Deployment Info:

- Job: 87.com.ibm.streamsx.geospatial.sample::GeofenceMarketingMain\_87 PE:532 [Healthy, No Alerts]

Flow Summary

Time	Source	Current Tuple	Previous Tuple
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	131902394	131893
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	131901669	131892
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	1	1
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	43967223	439641
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	43967223	439641
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	43967223	439641
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	131901310	131891
8/27/15 4:04:19 AM EDT	Job: 87.PE:52	131901310	131891
8/27/15 4:04:20 AM EDT	Job: 87.PE:53	131906105	131896
8/27/15 4:04:19 AM EDT	Job: 87.PE:53	43967058	439639
8/27/15 4:04:19 AM EDT	Job: 87.PE:53	43967058	439639
8/27/15 4:04:18 AM EDT	Job: 87.PE:53	131898112	131888
8/27/15 4:04:18 AM EDT	Job: 87.PE:53	131898111	131888
8/27/15 4:04:20 AM EDT	Job: 87.PE:53	1	1


# Streams Console – Metrics





# Streaming Analytics in Action





Verizon uses **IBM Streams** and cognitive analytics to deliver dynamic contextual content management



## Speech to Text

Listens side by side to agent-customer conversation



## Intent Detection

Comprehends the discussion and classifies the intent



## Scoring & Next Best Action

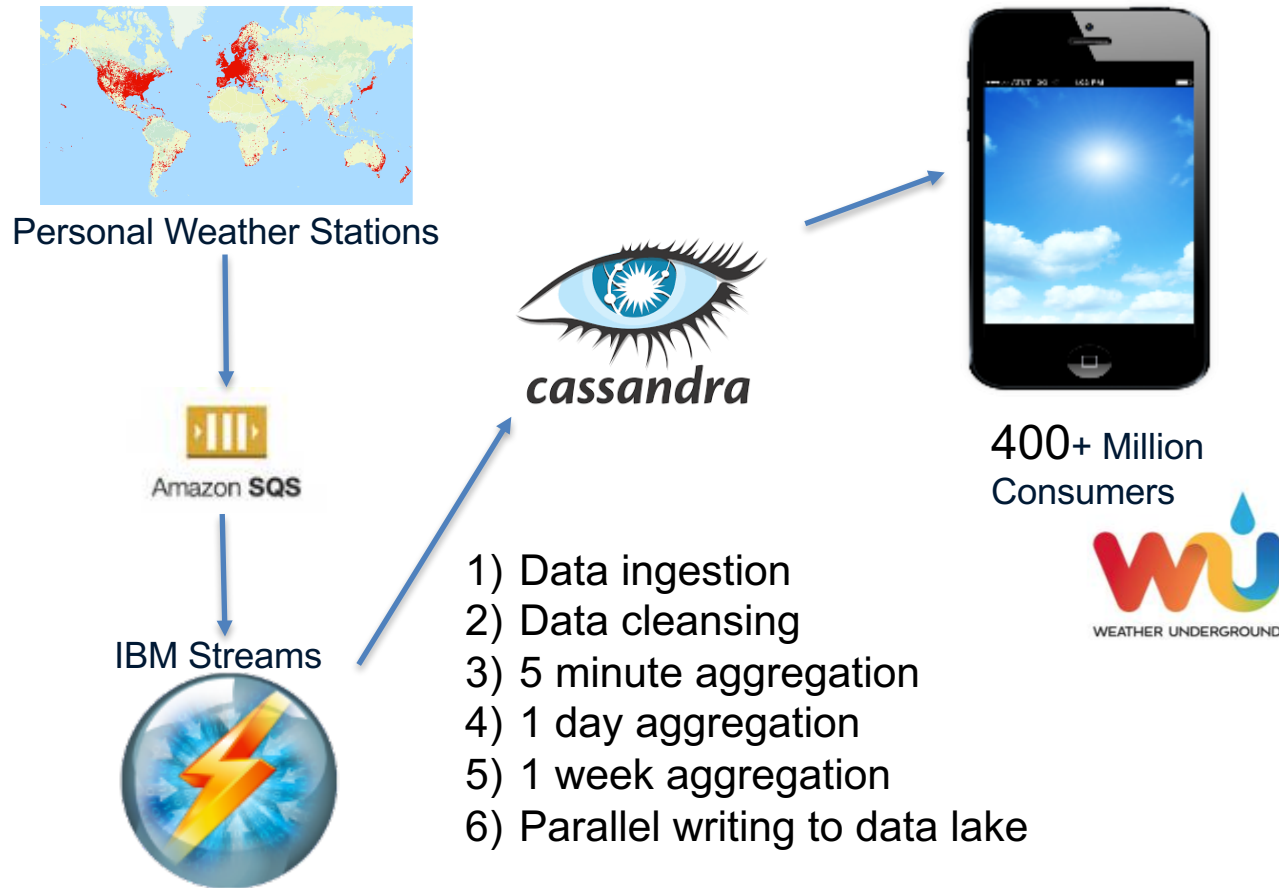
Identifies proactive and reactive relevant content



## Contextual Assist

Delivers cognitive agent assist

## TWC and Streams: Architecture to Handle PWS Growth

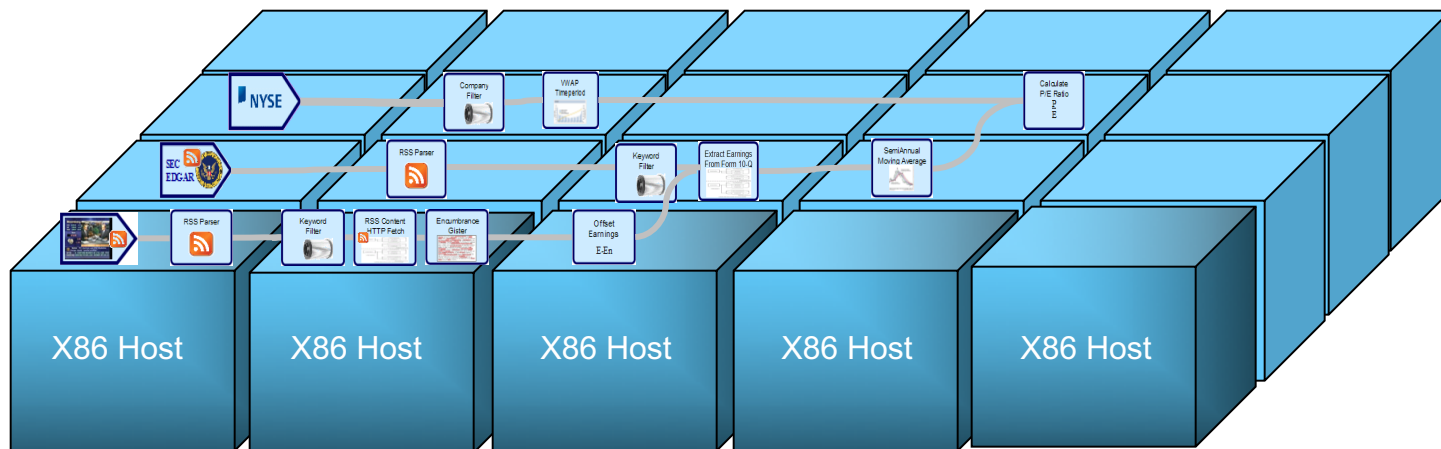


## Agenda

- Streaming Problems
- Introduction to IBM Streams
- **Solutions using IBM Streams**

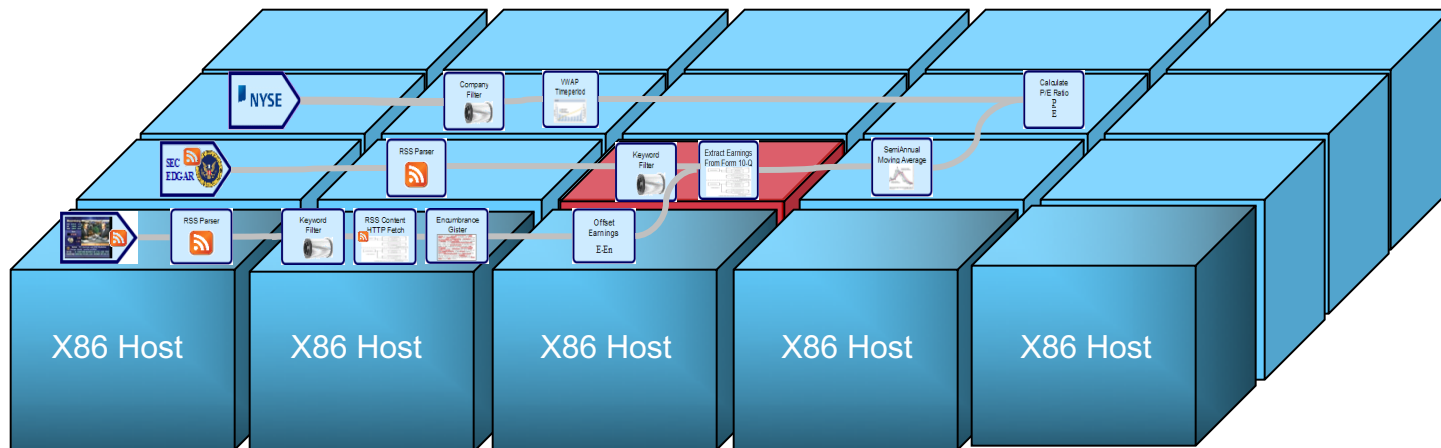
## High Availability: Restart, Fault Tolerance

- Runs on commodity hardware
  - From single node to blade to high performance multi-rack clusters
- Adapts to changes :



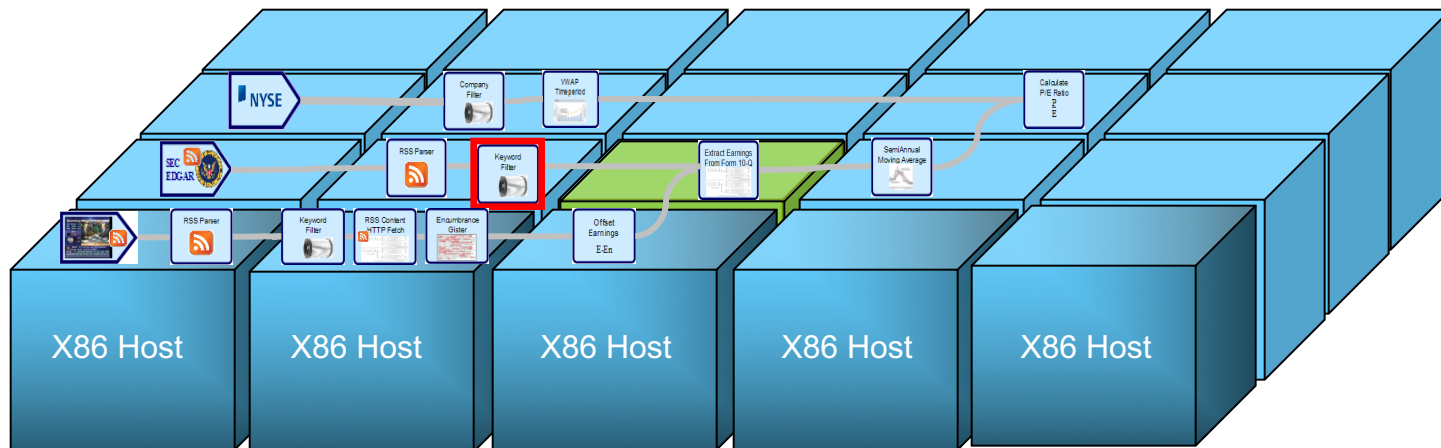
## High Availability: Restart, Fault Tolerance - cont'd

- Runs on commodity hardware
  - From single node to blade to high performance multi-rack clusters
- Adapts to changes :
  - In workloads



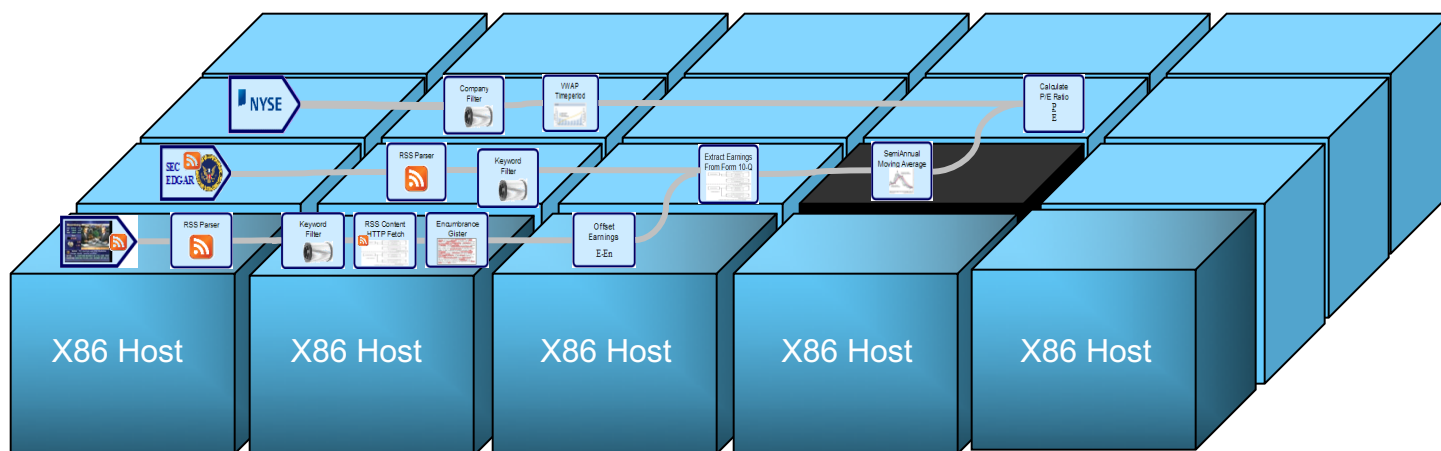
## High Availability: Restart, Fault Tolerance - cont'd

- Runs on commodity hardware
  - From single node to blade to high performance multi-rack clusters
- Adapts to changes :
  - In workloads



## High Availability: Restart, Fault Tolerance - cont'd

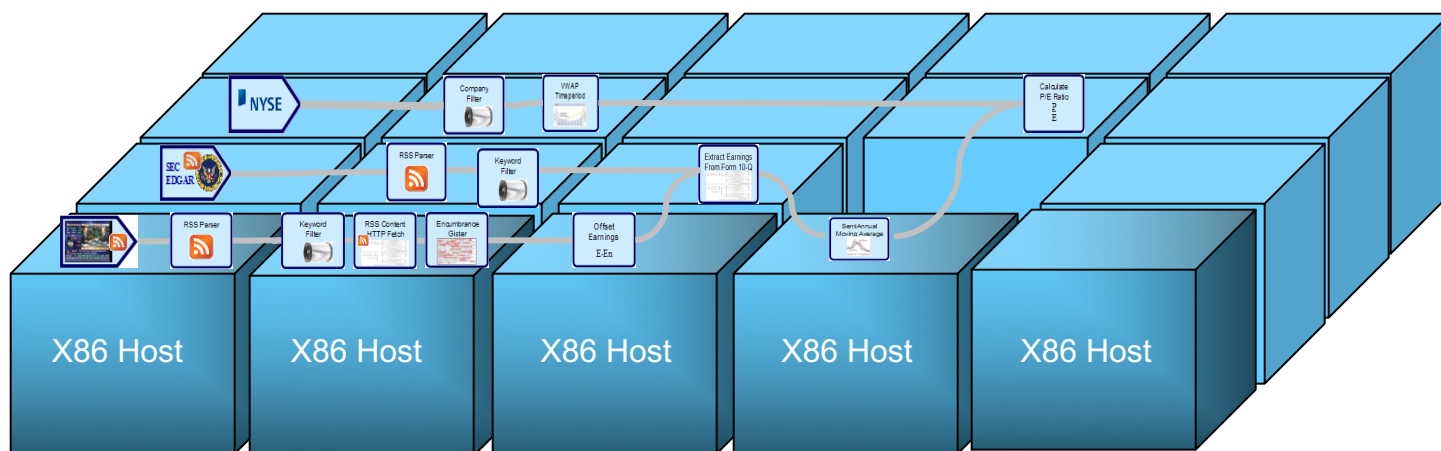
- Runs on commodity hardware
  - From single node to blade to high performance multi-rack clusters
- Adapts to changes :
  - In workloads
  - In resources





## High Availability: Restart, Fault Tolerance - cont'd

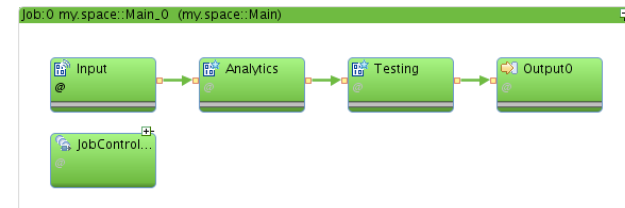
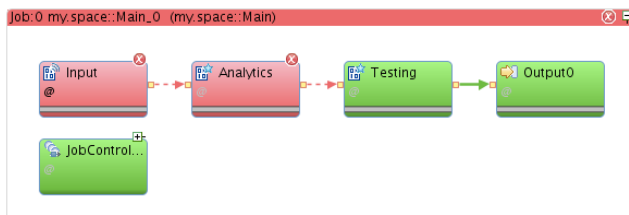
- Runs on commodity hardware
  - From single node to blade to high performance multi-rack clusters
- Adapts to changes :
  - In workloads
  - In resources



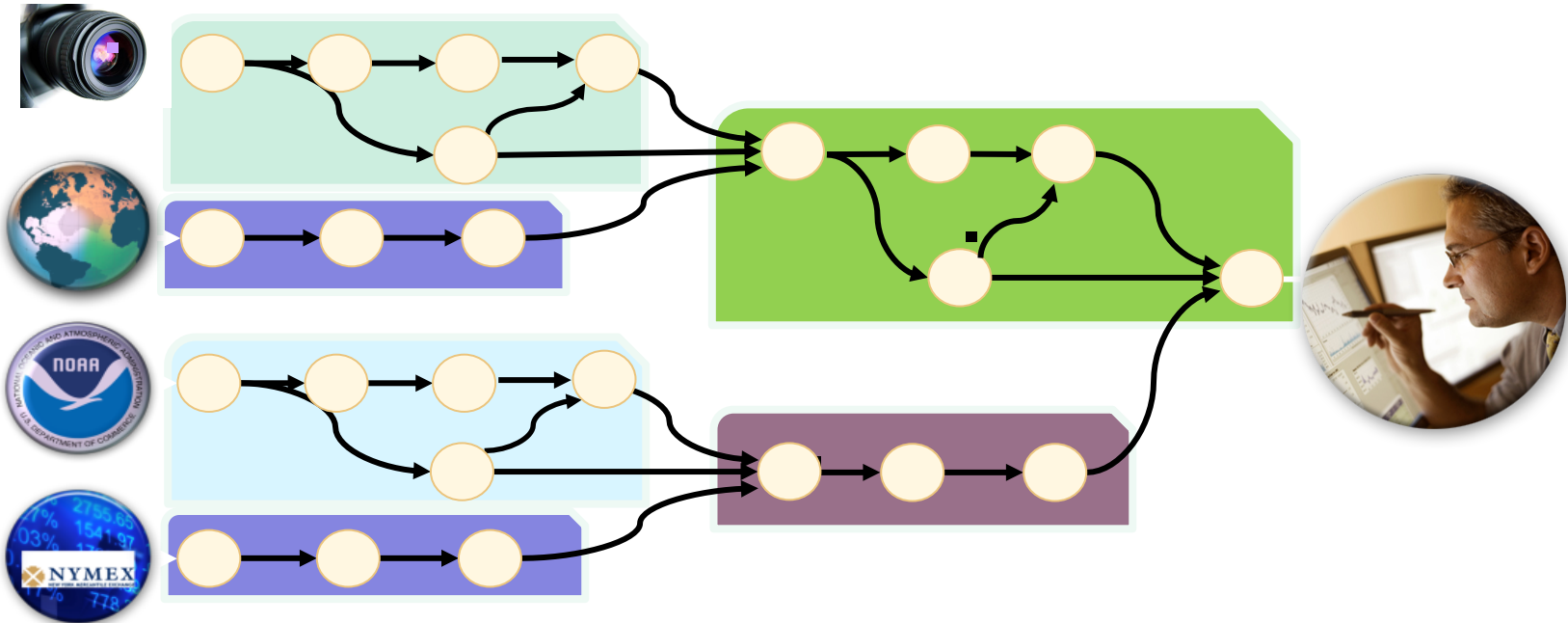
## Fault Tolerance, Resource Adaptation

- Use relocatable and restartable properties for automatic fail over
- Servers added to cluster using tags
- Reserve resources for specific purpose by tags
- User specified placement constraints
  - absolute host location
    - IP, Name, Pool
  - Relative host or partition constraint
    - hostColocation, hostExlocation, hostIsolation
    - partitionColocation, partitionExlocation, partitionIsolation

```
stream<int64 count> outStream = Custom(inStream) {
    .
    .
    config
        checkpoint: periodic(2.0);
        restartable : true;
        relocatable : true;
        placement: partitionColocation("Grp1"),
                  host(Pool1[0]);
}
```



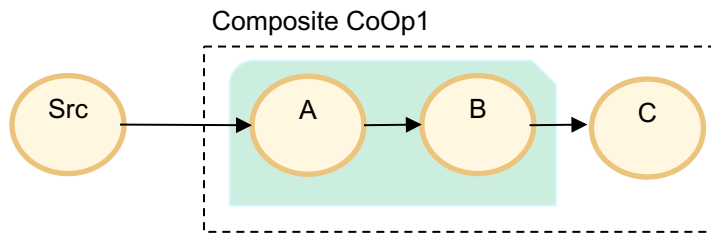
## Parallelism : Automated Fusion and Threading



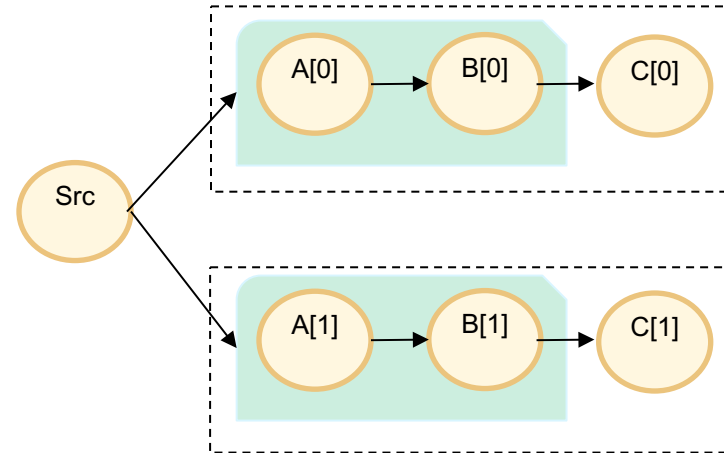
- Profile driven fusion at compile time
- Automatic fusion at submission time based on the resources available
  - reduce the number of PE processes, reduce load on a system, improve default performance
- Manual placement constraints precedence
- Manual/Dynamic/Automatic threading at runtime
  - Determine pool of worker threads and dynamically adjust as the throughput and load for the application changes

## User-defined parallelism

Logical



Physical



```

composite CoOp1(input In) {
  graph
    stream<Type> A = Functor(In) {
      config placement: hostColocation("AB" + (rstring)getChannel()),

    stream<Type> B = Functor(A) {
      config placement: hostColocation("AB" + (rstring)getChannel()); }

    stream<Type> C = Functor(B) {
      config placement: hostColocation("C" + (rstring)getChannel()); }
}
composite Main() {
  graph
    stream<Type> Src = Source() {}

    @parallel(width=2)
    () as Sink = CoOp1(Src) {}
}
  
```

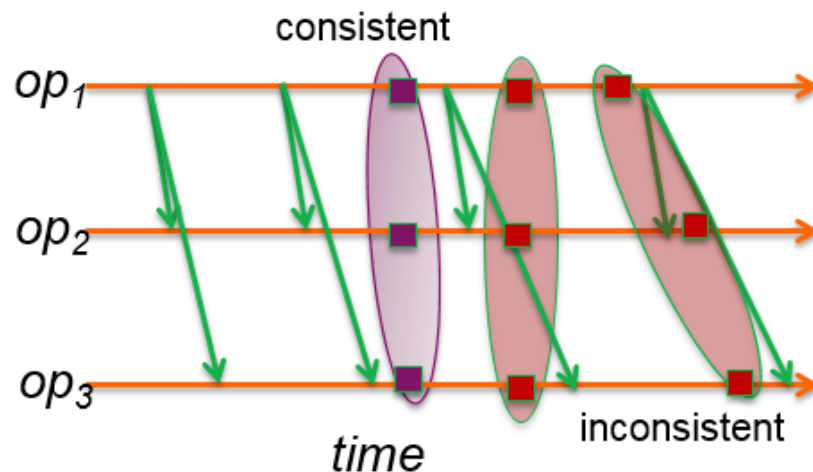
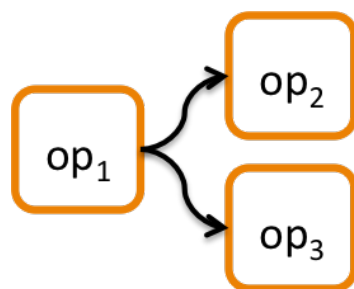
## Maintaining State in Streams - Windows, State, Checkpoints

```
stream<int64 minValue> outputStream = Aggregate(inputStream) {  
    window inputStream: sliding, count(50), count(1);  
    output outputStream : minValue = Min(j);  
}
```

```
stream<int64 count> outputStream = Custom(inputStream) {  
    logic state: mutable int64 cnt = 0;  
    .  
    .  
    config  
        checkpoint: periodic(2.0);
```

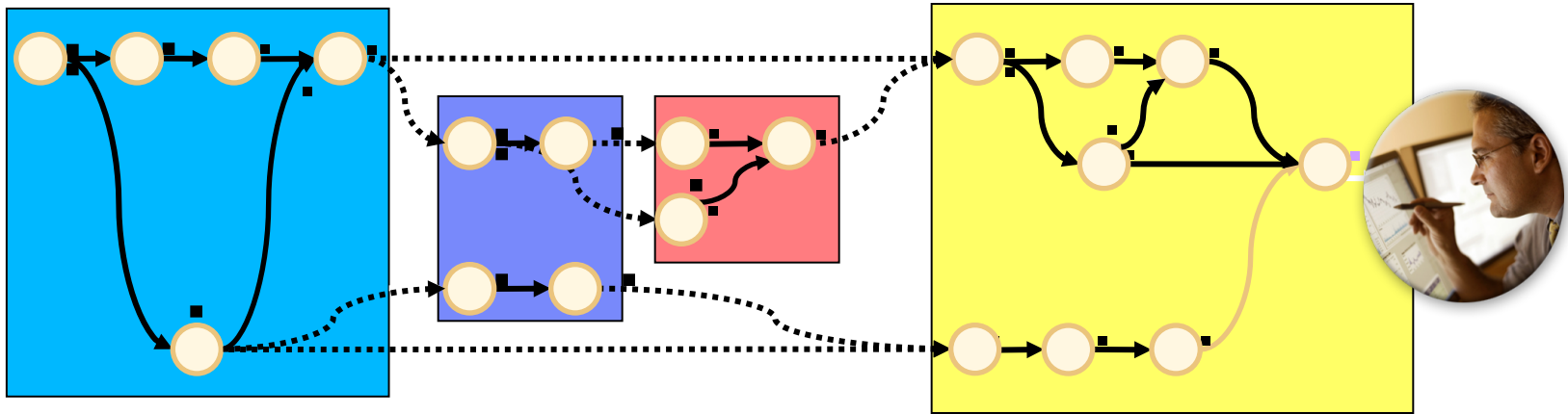
- State can be maintained using windows or mutable variables
- Out of box support for windows – no code required for spatial and temporal state management and event triggering
- Window types
  - Tumbling – Collective eviction
  - Sliding – Incremental eviction
  - Partitioned - multiple window with streams partitioned on some attribute, eg. Stock data (partitioned on ticker symbol)
- State recovery through check pointing

## Guaranteed Processing – Consistent region



- Annotate a region/subgraph as consistent
- Check pointing at regular intervals to save state
- On failure, state restored and tuples replayed
- Can have zero to many consistent regions
- Annotations
  - `@consistent {trigger, period, drainTimeout, resetTimeOut, maxConsecutiveResetAttempts}`
  - `@autonomous`

## Static vs. Dynamic Composition



- Static connections
  - Fully specified at application development-time and do not change at run-time
- Dynamic connections
  - Partially specified at application development-time (Name or Properties)
  - Established at run-time, as new jobs come and go
- Dynamic application composition
  - Incremental deployment of applications
  - Dynamic adaptation of applications

## Flow control – Punctuation, Throttling, Feedback loops

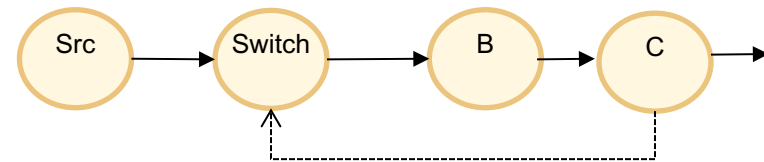
```
Stream<....> outStream = Operator1(inStream) {  
    ....  
    onTuple inStream: {...}  
    onPunct inStream : {...}  
}
```

- **Punctuation:** Control messages to logical partition streams, operators can execute logic based on appearance of punctuation

- **Throttle Operator :** used to pace a stream, control flow rate

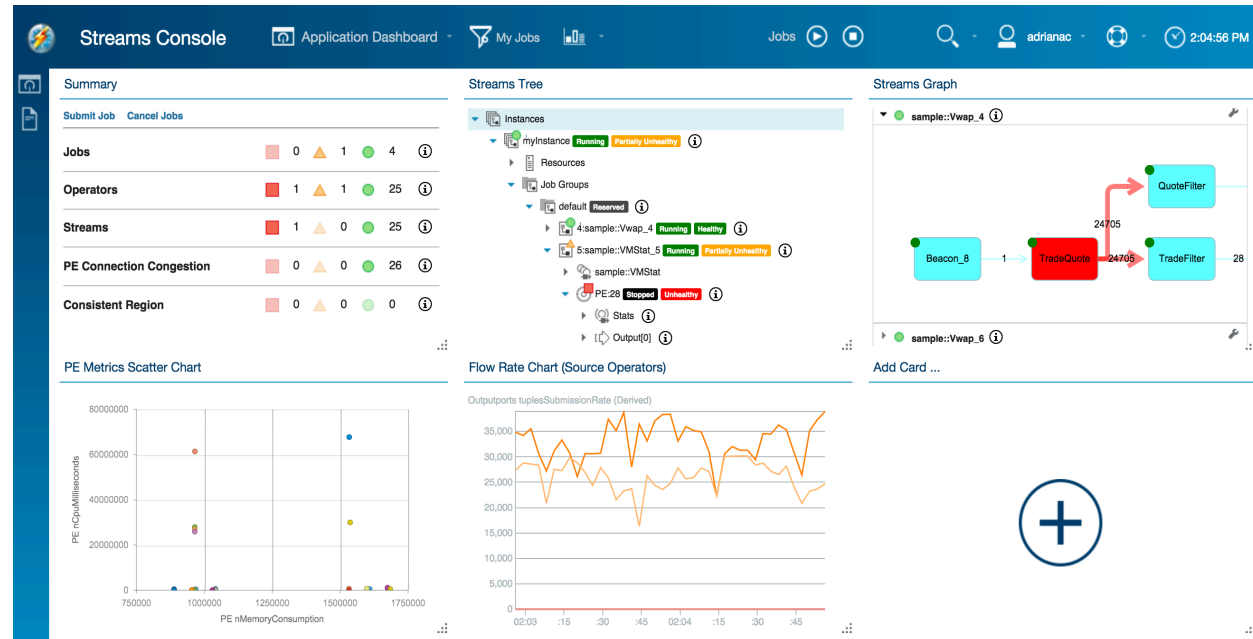
- **Switch Operator:**

- used to temporarily stop tuples from flowing
- used to hold tuples until a downstream operator is ready to process them
- Has control port which is triggered from downstream operator





## Managing & Monitoring Health



- Rest API
- JMX API
- Command line tool
- Customizable GUI
- Color coded Visual graph of metrics and topology
- A number of metrics and counters available
  - CPU, Memory, Tuple in/out rate, Congestion, Health, Flow rate, Byte rate, punctuation count, tuples dropped, etc



## Reference Resources

- Toolkits, Samples, Documentation

<https://ibmstreams.github.io/>

- Developer Community

<https://developer.ibm.com/streamsdev/>

- Product Page

<https://www.ibm.com/us-en/marketplace/stream-computing>

# *Questions?*



Thank  
You