University of Bologna

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Engineering Bologna Campus

Class of

Computer Networks M

Global Stream Processing

Luca Foschini

Academic year 2015/2016

Outline

A set of tools are available to express and design a **complex streaming architecture** to be immediately deployed

- Apache Storm
- Yahoo S4

…

Stream Processing Challenge

- Large amounts of data →
  Need for **real-time views of data**
  - Social network trends, e.g., Twitter real-time search
  - Website statistics, e.g., Google Analytics
  - Intrusion detection systems, e.g., in most datacenters
- Process large amounts of data
  - With latencies of few seconds
  - With high throughput

Not MapReduce

- **Batch Processing →** Need to wait for entire computation on large dataset to complete
- Not intended for long-running stream-processing

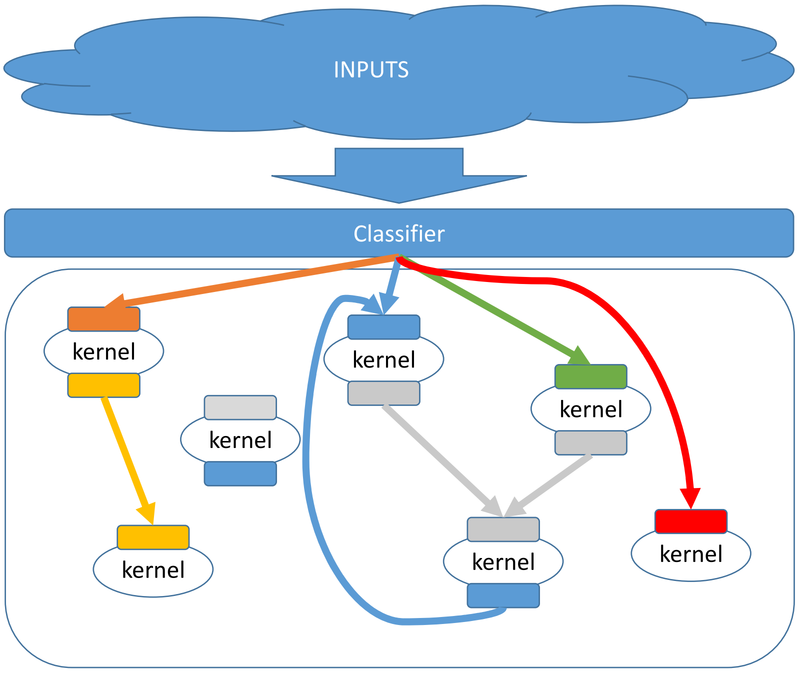
Stream processing model

Stream processing manages:

- Allocation
- Synchronization
- Communication

Application that benefit most the streaming model with requirements:

- High computation resource intensive
- Data parallelization
- Data time locality



Stream processing support functions

Main functions needed to support the **stream processing** model:

- **Resource allocation**
- **Data classification Information routing**
- **Management of execution/processing status**

Enter Storm

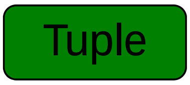
- Apache Project
- http://storm.apache.org/
- Highly active JVM project
- Multiple languages supported via API
  - Python, Ruby, etc.

- Used by over 30 companies including
  - Twitter: For personalization, search
  - Flipboard: For generating custom feeds
  - Weather Channel, WebMD, etc.

Storm Core Components

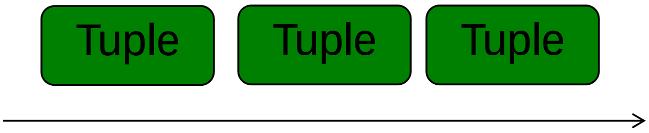
- Tuples
- Streams
- Spouts
- Bolts
- Topologies

Tuple



- An ordered list of elements
- E.g., <tweeter, tweet>
  - E.g., <“Miley Cyrus”, “Hey! Here’s my new song!”>
  - E.g., <“Justin Bieber”, “Hey! Here’s MY new song!”>
- E.g., <URL, clicker-IP, date, time>
  - E.g., <coursera.org, 101.102.103.104, 4/4/2014, 10:35:40>
  - E.g., <coursera.org, 101.102.103.105, 4/4/2014, 10:35:42>

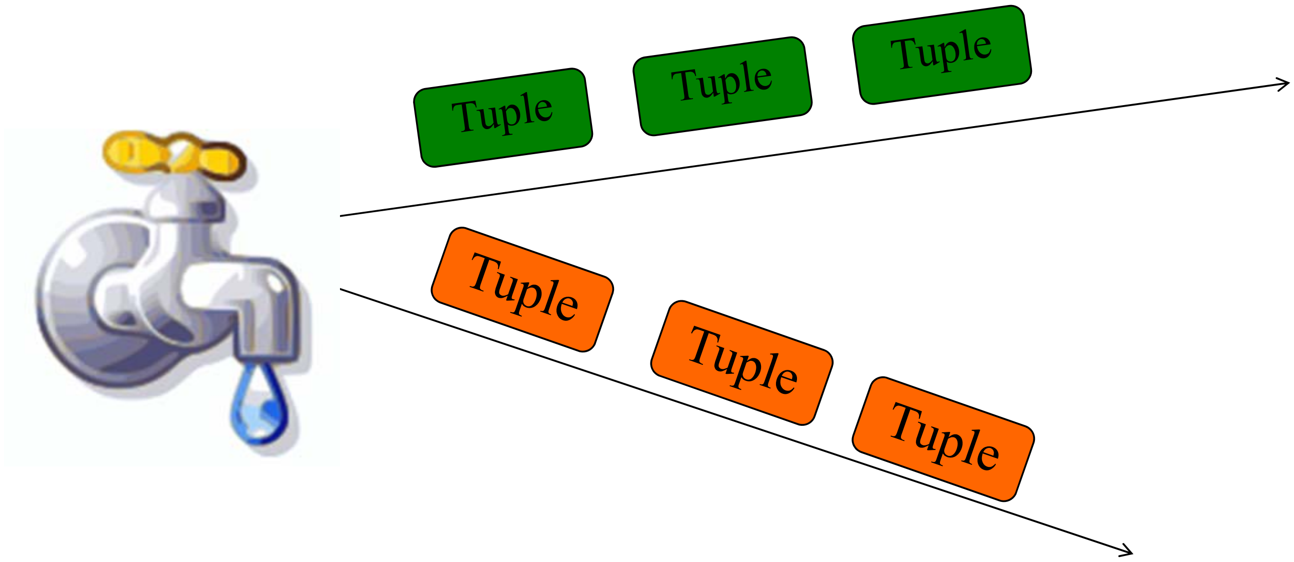
Stream



- Sequence of tuples
  - Potentially unbounded in number of tuples
- Social network example:
  - <“Miley Cyrus”, “Hey! Here’s my new song!”>,
    <“Justin Bieber”, “Hey! Here’s MY new song!”>,
    <“Rolling Stones”, “Hey! Here’s my old song that’s still a super-hit!”>, …
- Website example:
  - <coursera.org, 101.102.103.104, 4/4/2014, 10:35:40>, <coursera.org, 101.102.103.105, 4/4/2014, 10:35:42>, …

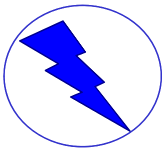
Spout

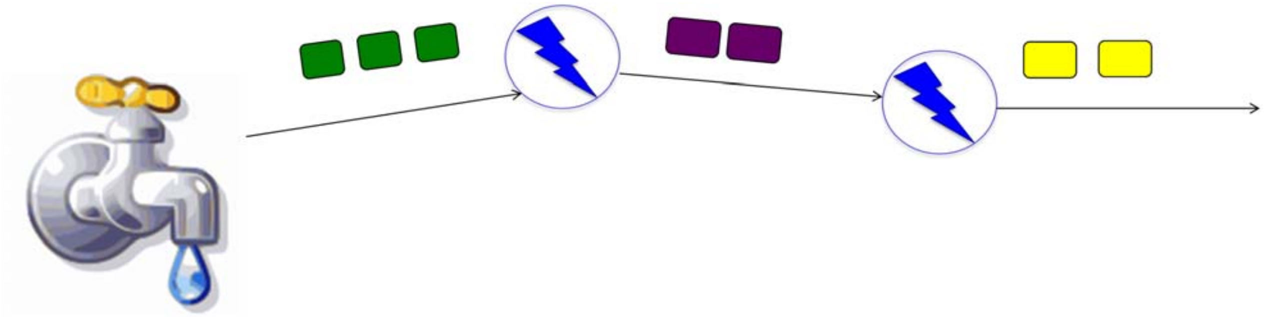
- A Storm entity (process) that is a source of streams
- Often reads from a crawler or DB



Bolt

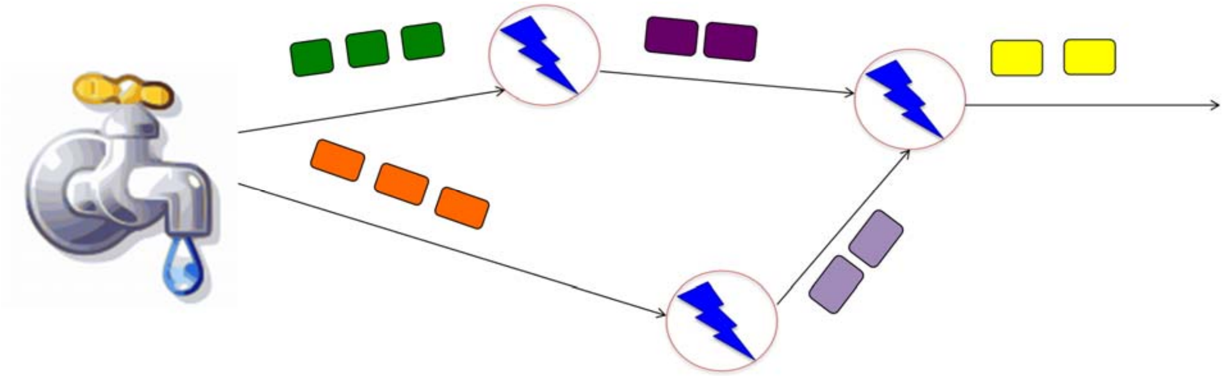
- A Storm entity (process) that
  - Processes input streams
  - Outputs more streams for other bolts





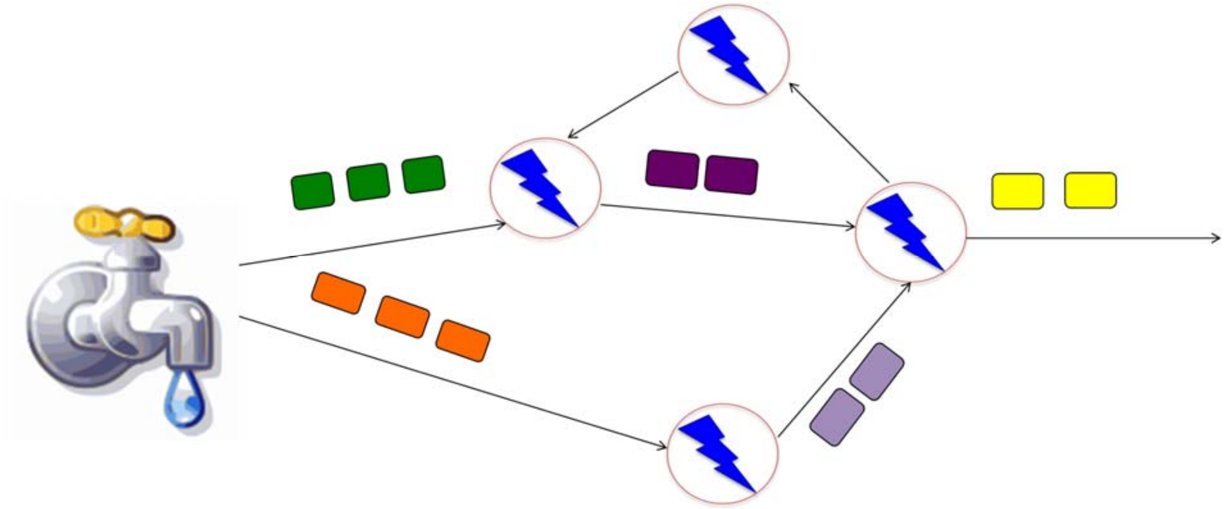
Topology

- A **directed graph** of **spouts** and **bolts** (and output bolts)
- Corresponds to a Storm “application”



Topology

- Can have **cycles** if the application requires it



Bolts come in many Flavors

- Operations that can be performed
  - **Filter**: forward only tuples which satisfy a condition
  - **Joins**: When receiving two streams A and B, output all pairs (A,B) which satisfy a condition
  - **Apply/transform**: Modify each tuple according to a function
  - And many others
- But bolts need to process a lot of data
  - Need to make them fast

Parallelizing Bolts

- Have multiple processes ("tasks") constitute a bolt
- Incoming streams split among the tasks
- Typically each incoming tuple goes to one task in the bolt
  - Decided by "Grouping strategy"
- Three types of grouping are popular

Grouping

- **Shuffle Grouping**
  - Streams are distributed evenly among the bolt’s tasks
  - Round-robin fashion
- **Fields Grouping**
  - Group a stream by a subset of its fields
  - E.g., All tweets where twitter username starts with [A-M,a-m,0-4] goes to task 1, and all tweets starting with [N-Z,n-z,5-9] go to task 2
- **All Grouping**
  - All tasks of bolt receive all input tuples
  - Useful for joins

Failures

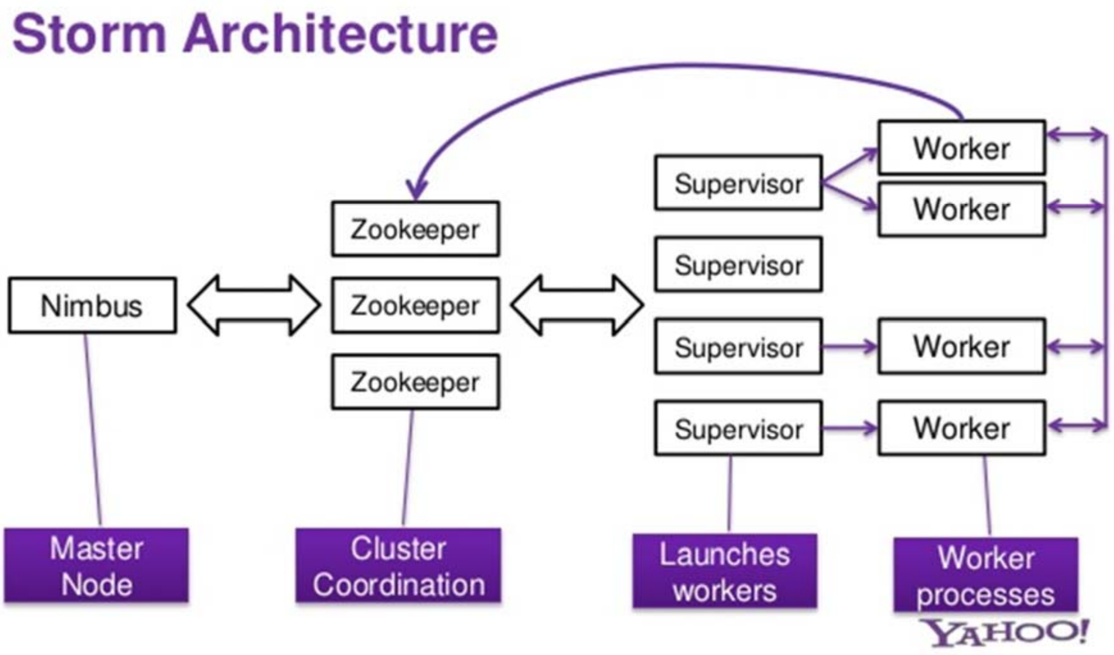
- A tuple is considered failed when its topology (graph) of resulting tuples fails to be fully processed within a specified timeout
- **Anchoring**: Anchor an output to one or more input tuples
  - Failure of one tuple causes one or more tuples to replayed

API For Fault-Tolerance (OutputCollector)

- **Emit**(tuple, output)
  - Emits an output tuple, perhaps anchored on an input tuple (first argument)
- **Ack**(tuple)
  - Acknowledge that you (bolt) finished processing a tuple
- **Fail**(tuple)
  - Immediately fail the spout tuple at the root of tuple topology if there is an exception from the database, etc.
- Must remember to ack/fail each tuple
  - Each tuple consumes memory. Failure to do so results in memory leaks.

Storm Cluster

Several components in a Cluster



Storm Cluster

- Master node
  - Runs a daemon called *Nimbus*
  - Responsible for
    - Distributing code around cluster
    - Assigning tasks to machines
    - Monitoring for failures of machines
- Worker node
  - Runs on a machine (server)
  - Runs a daemon called *Supervisor*
  - Listens for work assigned to its machines
  - Runs "Executors"(which contain groups of tasks)
- Zookeeper
  - Coordinates Nimbus and Supervisors communication
  - All state of Supervisor and Nimbus is kept here

Twitter Heron System

- Fixes the inefficiencies of Storm's acking mechanism (among other things)
- Uses **backpressure**: a congested downstream tuple will ask upstream tuples to slow or stop sending tuples

1. TCP Backpressure: uses TCP windowing mechanism to propagate backpressure

2. Spout Backpressure: node stops reading from its upstream spouts

3. Stage by Stage Backpressure: think of the topology as stage-based, and propagate back via stages

- Use:
  - Spout+TCP, or
  - Stage by Stage + TCP
- Beats Storm throughput handily (see Heron paper)

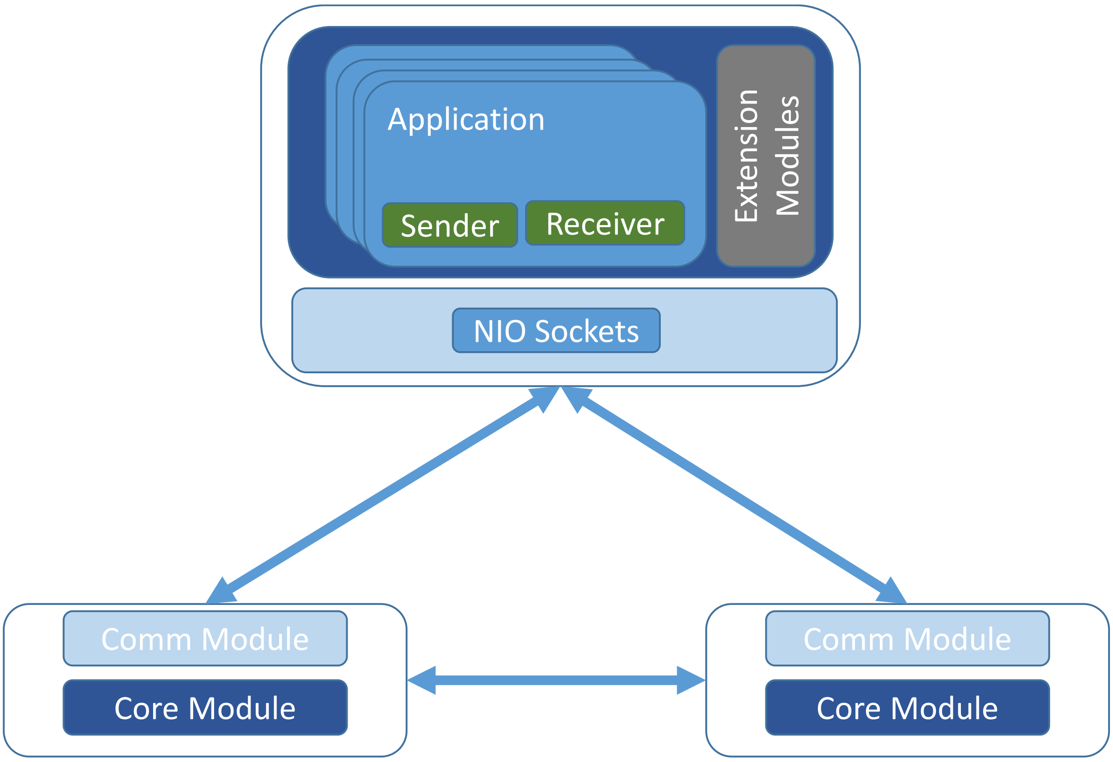
S4 Platform

Simple Scalable Streaming System (S4)

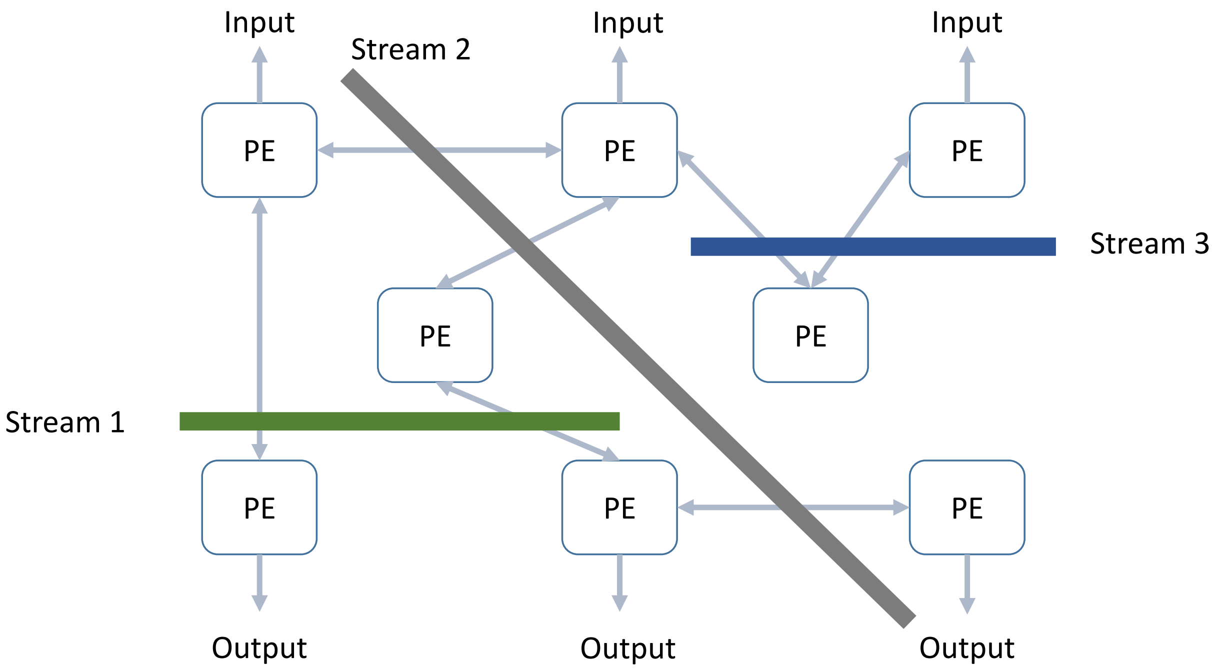
Design goals:

- Scalability
- Decentralization
- Fault-tolerance (partially supported)
- Elasticity
- Extensibility
- Object oriented

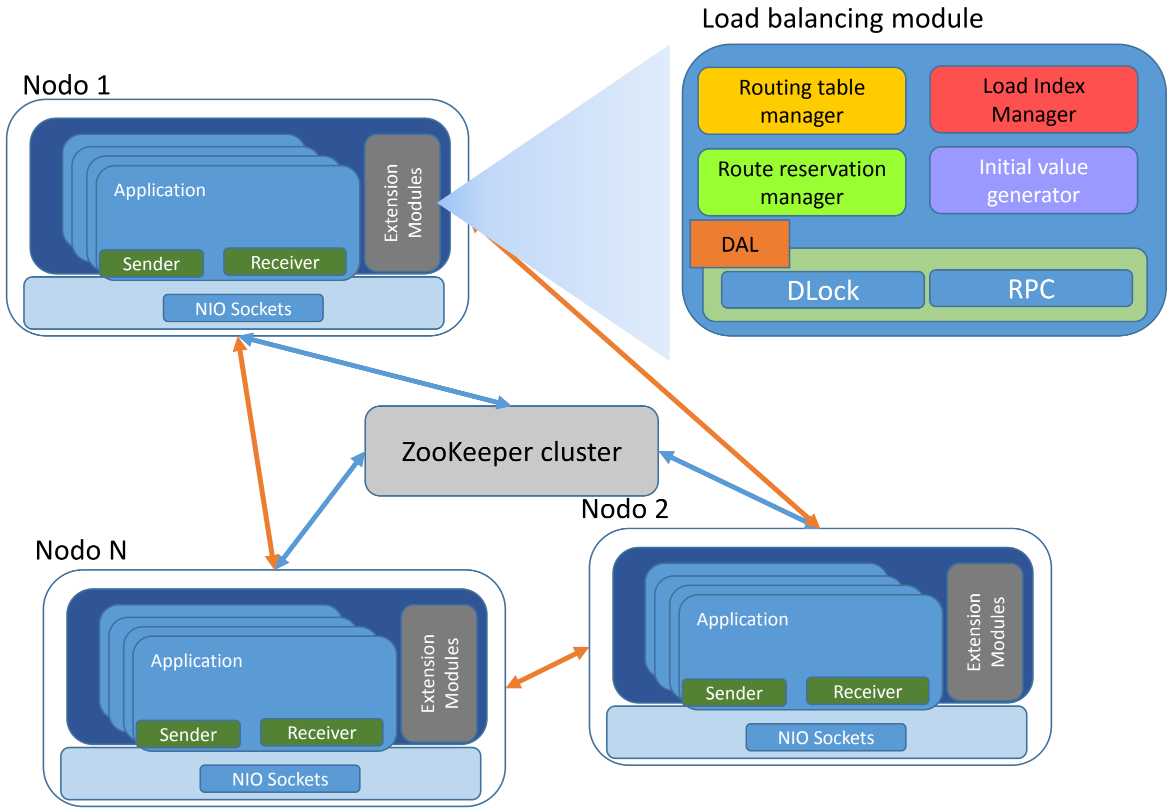
S4 Platform - architecture



S4 Platform - application



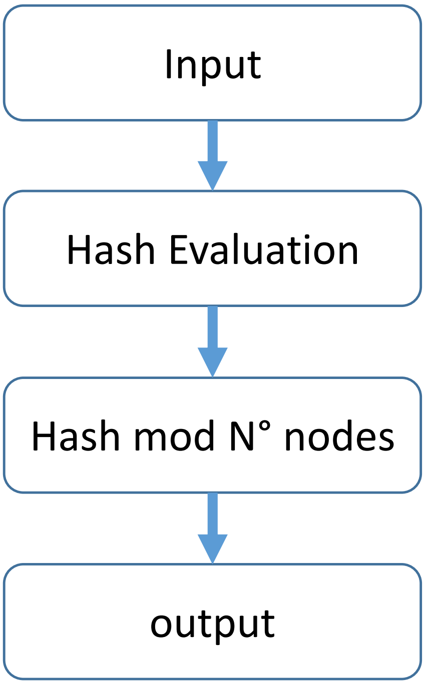
S4 Platform – overall view



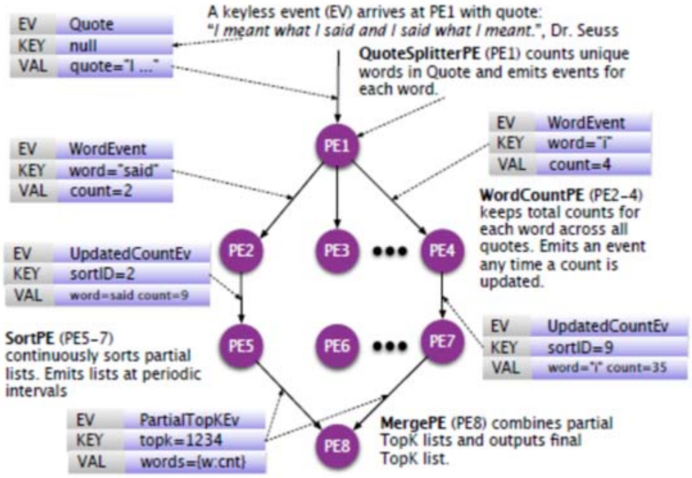
Load balancing support & open issues

Not really supported…

- There is no real load balancing support
- Load sharing on cluster nodes based on very simple hash functions
- No guarantees of effectively balanced load sharding



An example: Word Count (sounds familiar?)



A keyless event (EV) arrives at PE1 with quote: *"I meant what I said and I said what I meant."*, Dr. Seuss

QuoteSplitterPE (PE1) counts unique words in Quote and emits events for each word.

WordCountPE (PE2–4) keeps total counts for each word across all quotes. Emits an event any time a count is updated.

SortPE (PE5–7) continuously sorts partial lists. Emits lists at periodic intervals

MergePE (PE8) combines partial TopK lists and outputs final TopK list.

PE ID	PE Name	Key Tuple
PE1	QuoteSplitterPE	null
PE2	WordCountPE	word="said"
PE4	WordCountPE	word="i"
PE5	SortPE	sortID=2
PE7	SortPE	sortID=9
PE8	MergePE	topK=1234

Figure 1. Word Count Example

For more details refer to the S4 presentation paper: L. Neumeyer *et al.*, *"S4: Distributed Stream Computing Platform"*, KDCloud 2010.