CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

STREAM PROCESSING

Lecture A

STREAM PROCESSING IN STORM

What We'll Cover

- Why Stream Processing
- Storm

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

MapReduce?

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

Enter Storm

- Apache Project
- https://storm.incubator.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 Components

- Tuples
- Streams
- Spouts
- Bolts
- Topologies

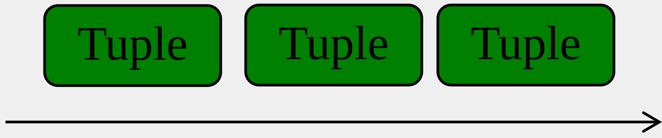
Tuple

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, 101.201.301.401, 4/4/2014, 10:35:40>
  - E.g., <coursera.org, 901.801.701.601, 4/4/2014, 10:35:42>

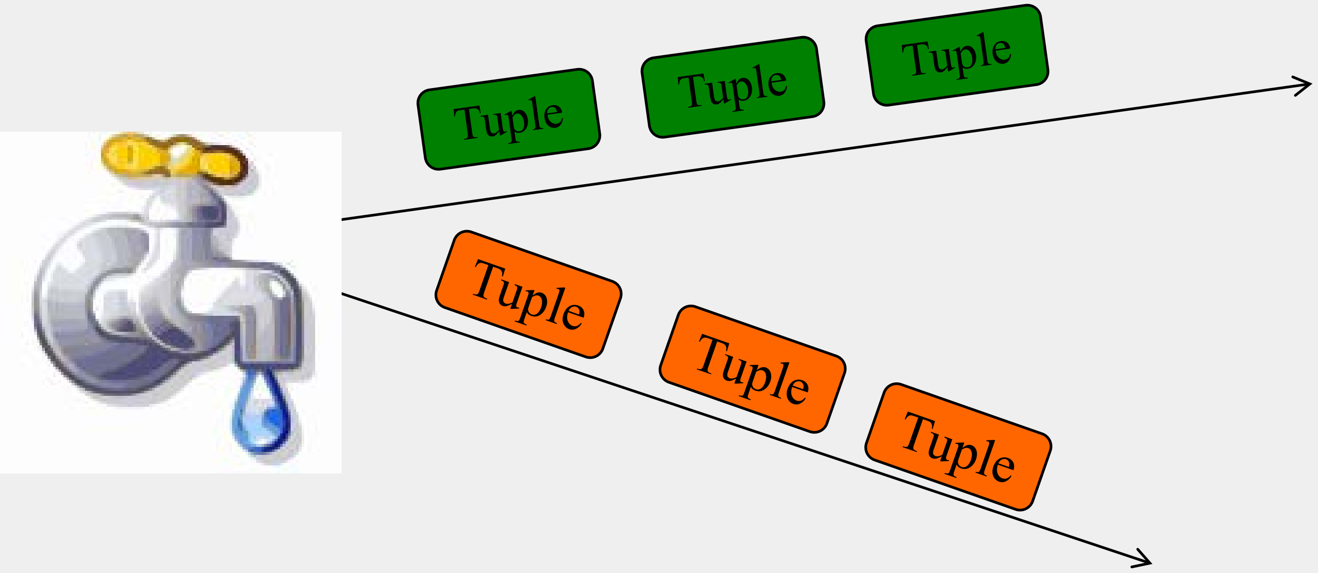
Stream

- 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, 101.201.301.401, 4/4/2014, 10:35:40>, <coursera.org, 901.801.701.601, 4/4/2014, 10:35:42>, …



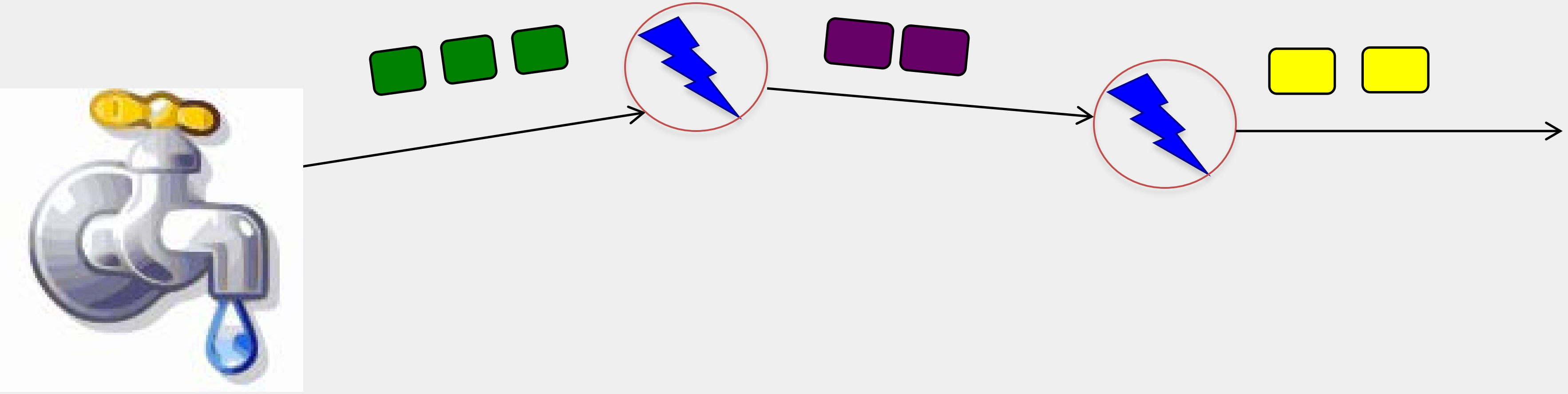
Spout

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



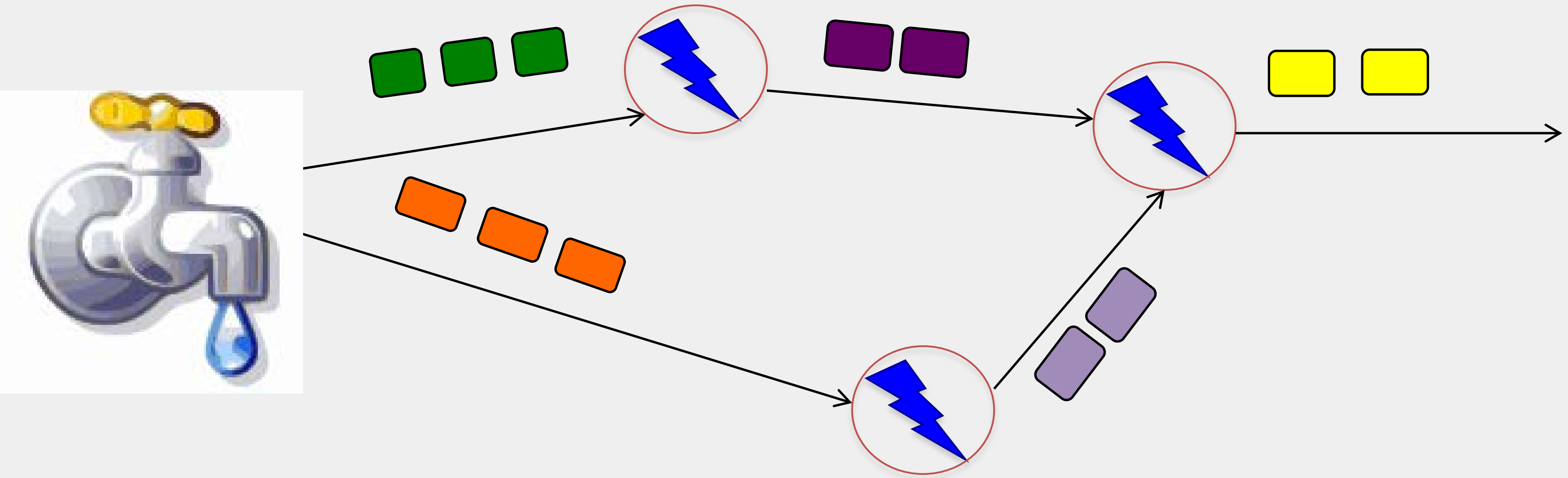
Bolt

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



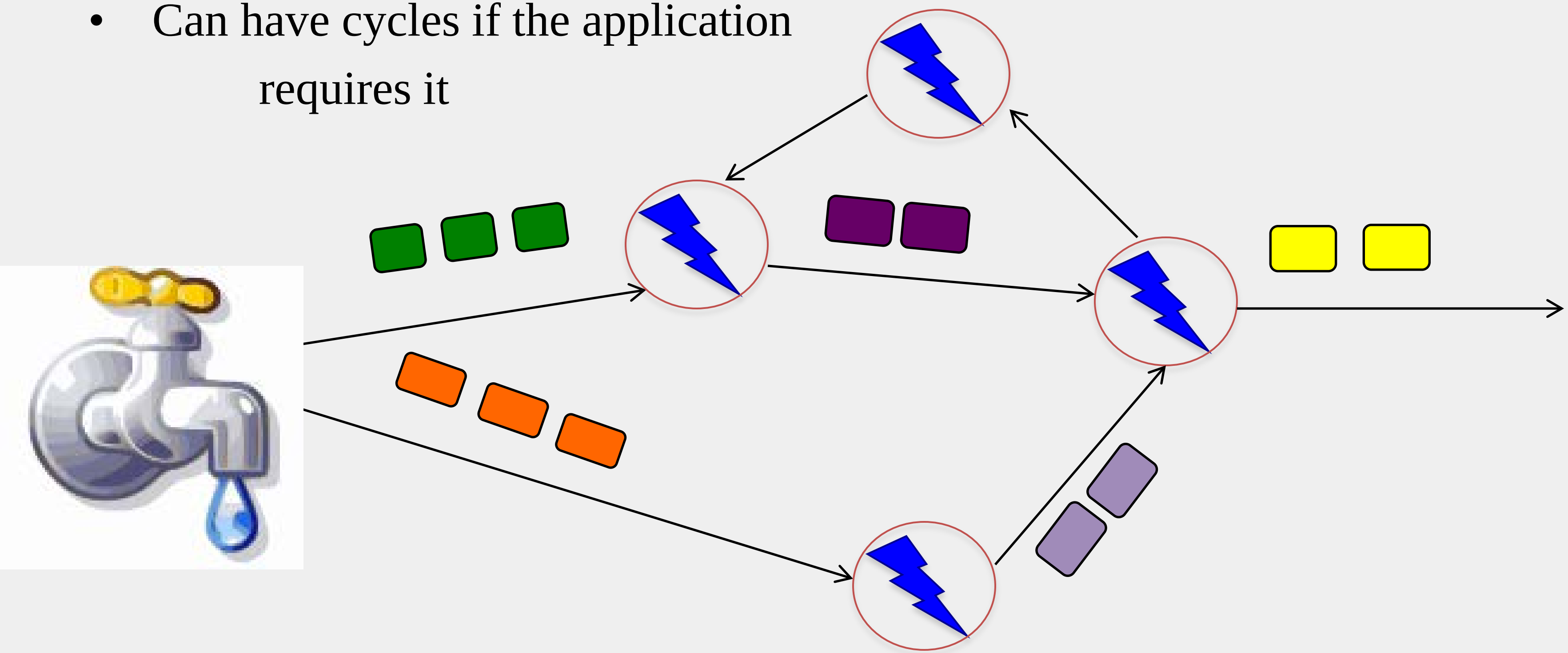
Topology

- 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 across 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

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
- Zookeeper
  - Coordinates Nimbus and Supervisors communication
  - All state of Supervisor and Nimbus is kept here

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 finish 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.

Summary

- Processing data in real-time a big requirement today
- Storm
  - And other sister systems, e.g., Spark Streaming
- Parallelism
- Application topologies
- Fault-tolerance