CSC 555 and DSC 333
Mining Big Data
Lecture 8

Alexander Rasin
College of CDM, DePaul University

October 29th, 2021

Tonight

- Clustering/Mahout
- Storm (streaming engines)
- Spark (in-memory engines)

RAM

Jaccard Similarity

- Ratio between
  - Intersection of two sets and
  - Union of two sets

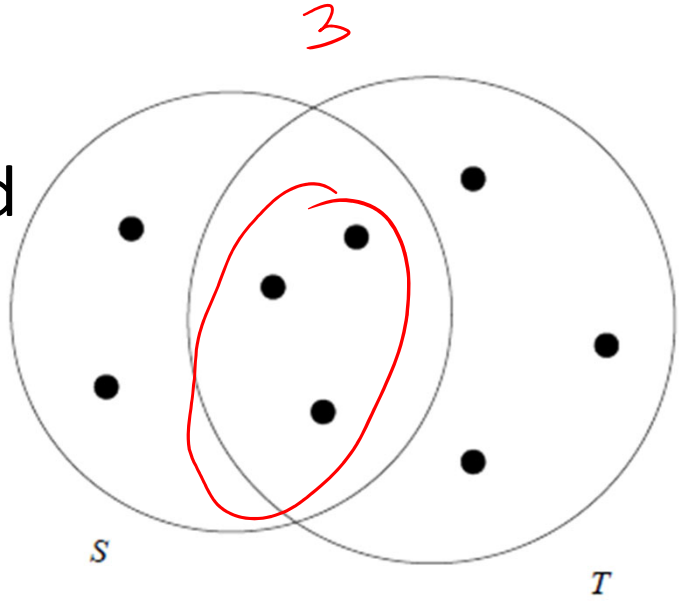


Figure 3.1: Two sets with Jaccard similarity 3/8

- Intuitive tests
  - Similarity S <> S is 1
  - Similarity S <> T with zero overlap is 0
  - Otherwise similarity is between 0 and 1

Distance Measures

- Dist(x, y) >= 0
- Dist(x, y) = 0 iff x=y
- Dist(x,y) = Dist(y, x)
- Dist(x,y) <= Dist(x,z) + Dist(z, y)

- Euclidean Distance

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

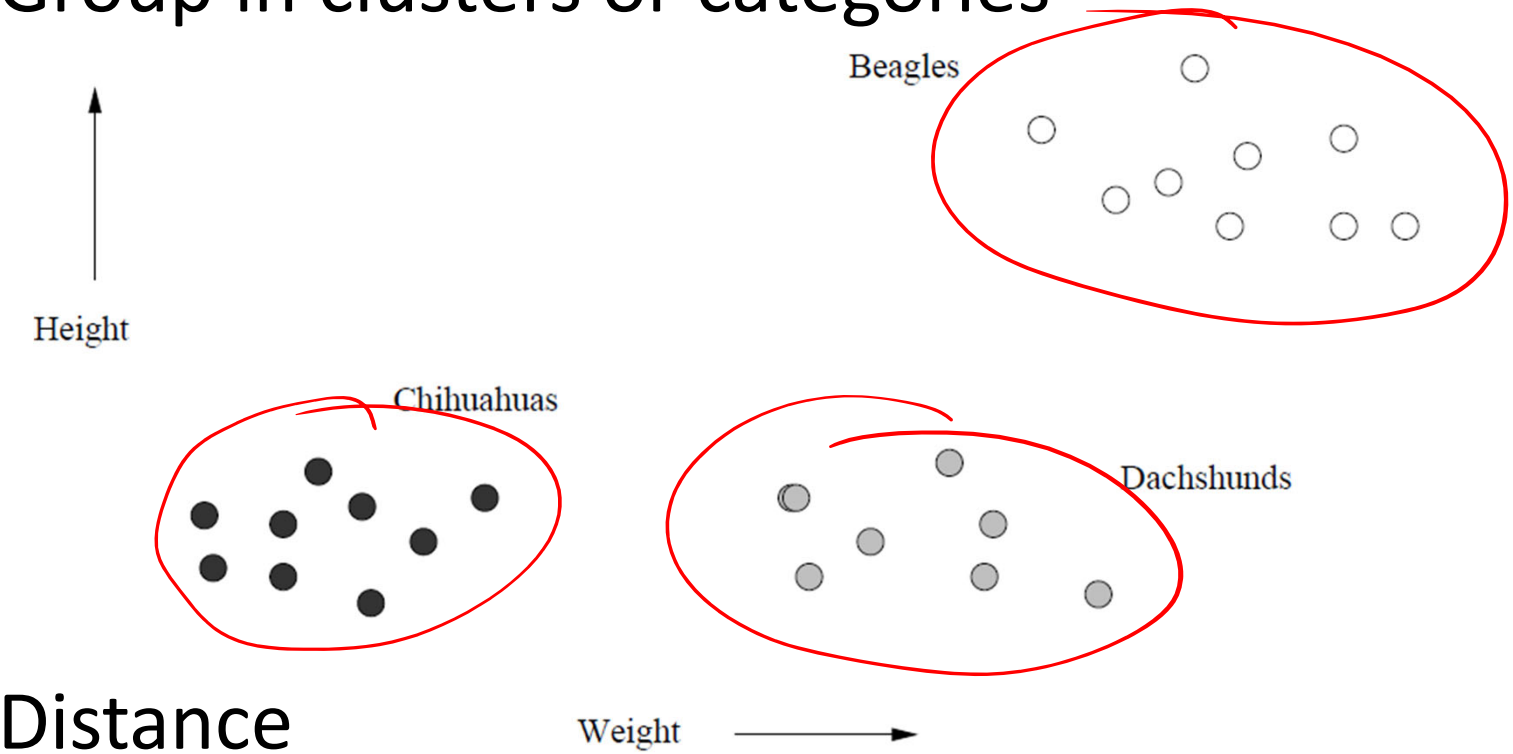
Jaccard Distance

- Dist(x, y) = 1 – SIM(x, y)

- Cosine distance
  - Angle between vectors
- Edit distance
  - # of insertion/deletions for transforming x to y
- Hamming distance
  - # of elements that differ between x and y

Clustering

- Set of elements in space
- Group in clusters or categories

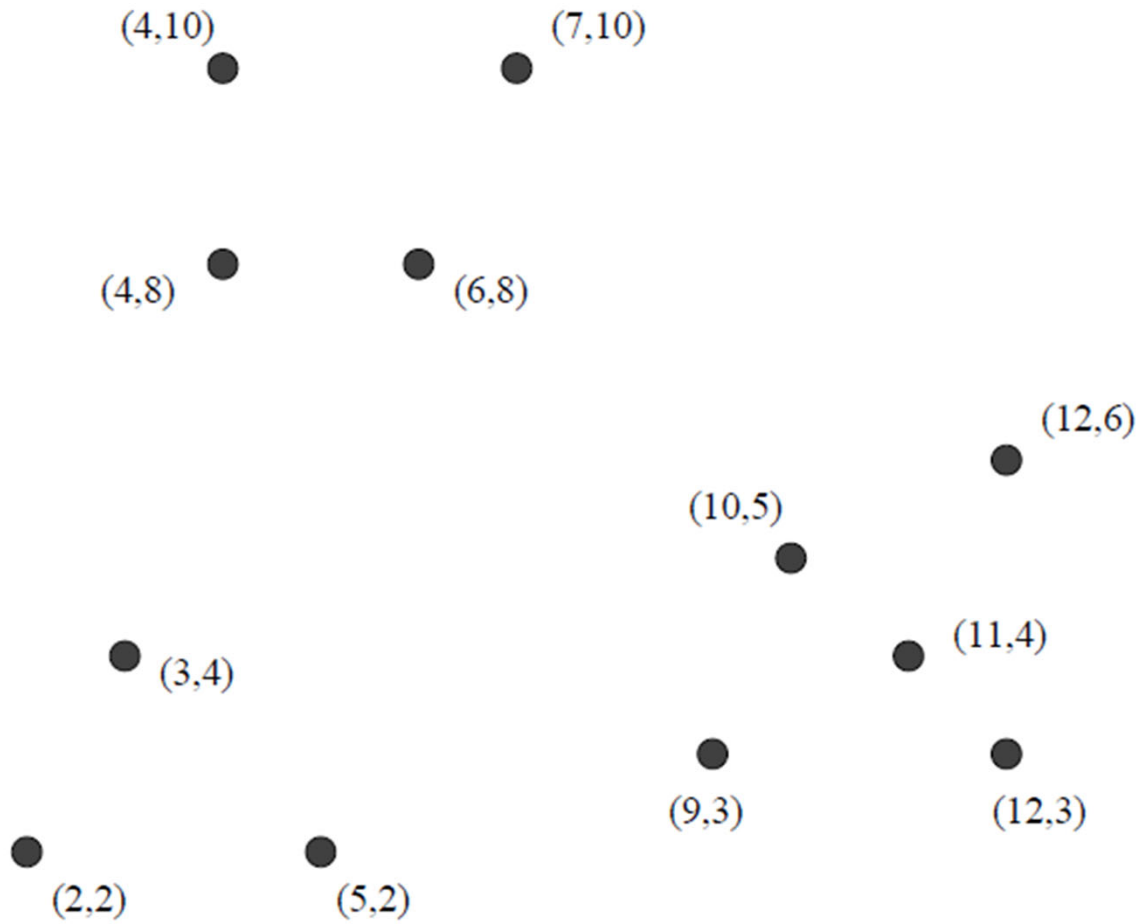


- Distance

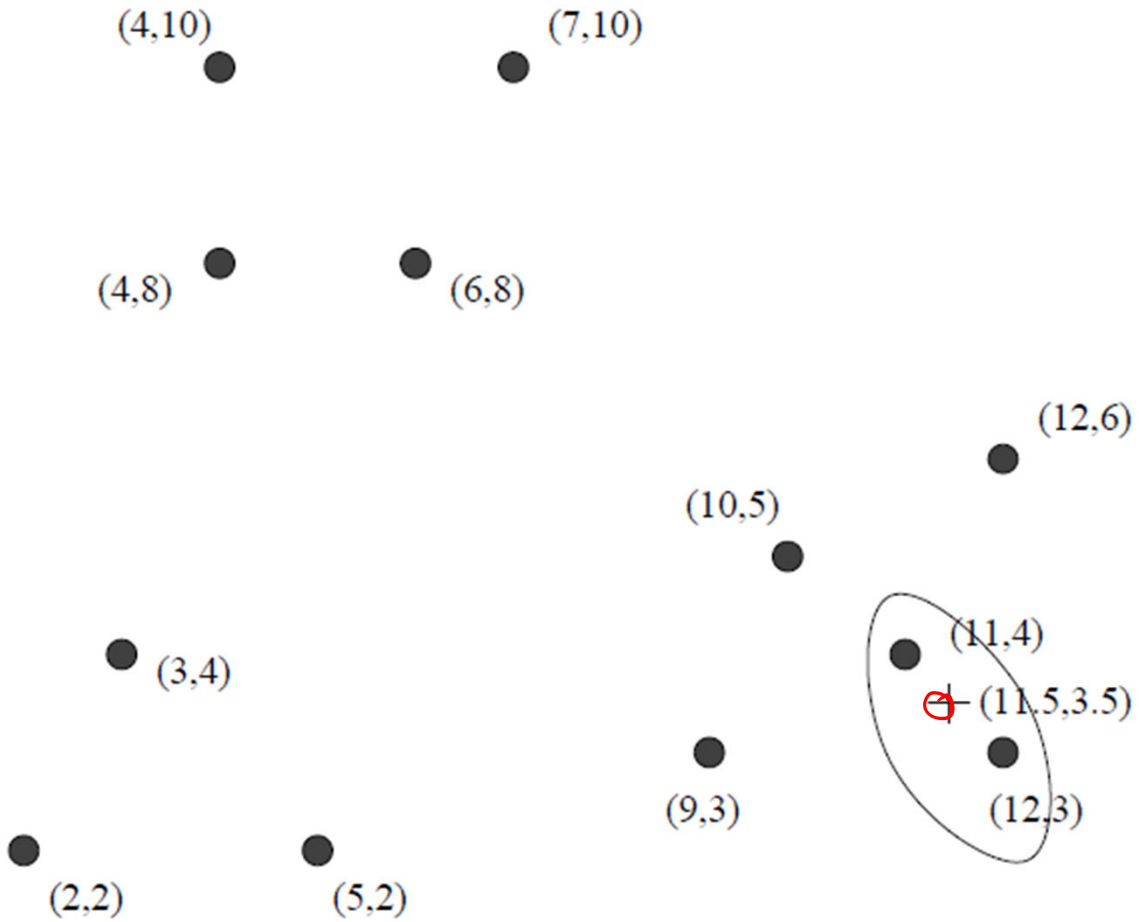
Hierarchical Clustering

- Start from 1 point => 1 cluster
- Apply pairwise merging
  - Simpler distance measure requirements
- Stop condition
  - k # of clusters, cluster too large, etc.
- No back-tracking

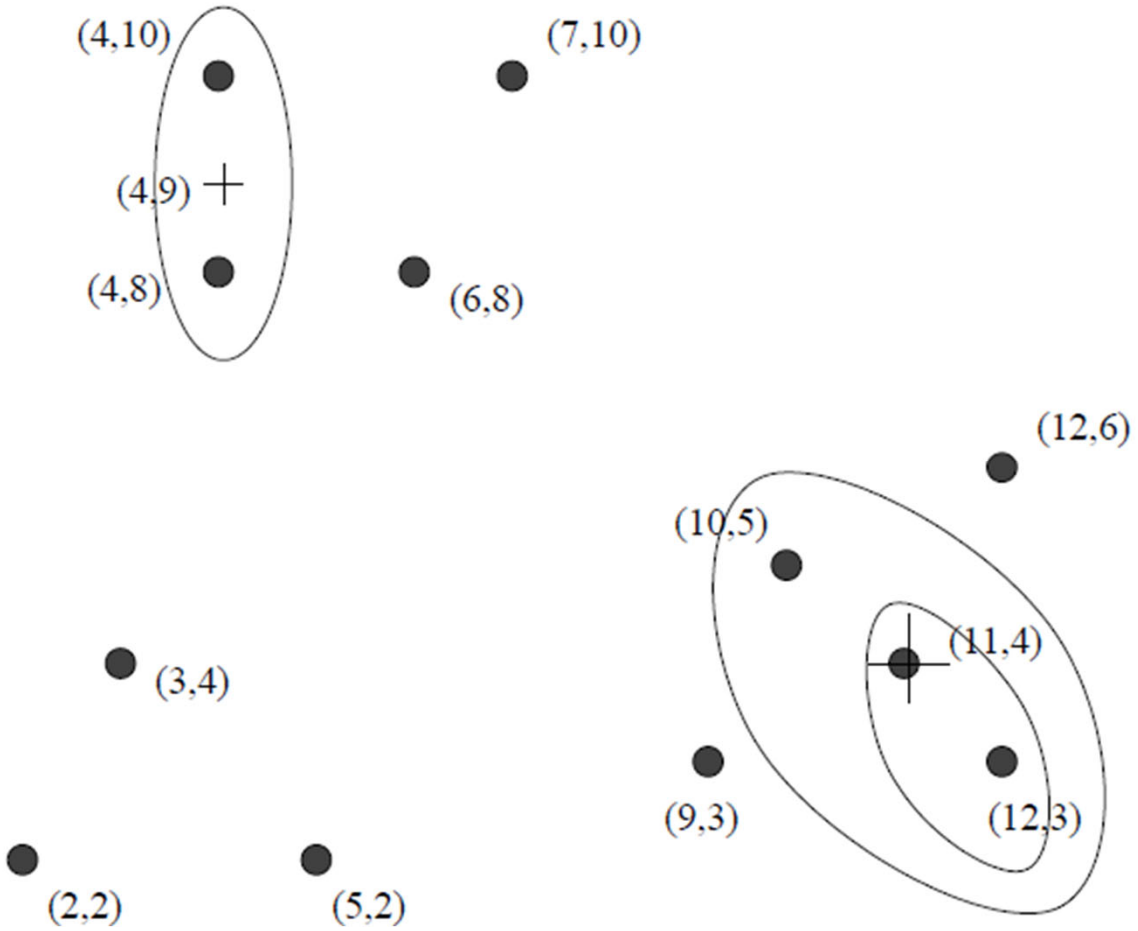
Hierarchical Clustering



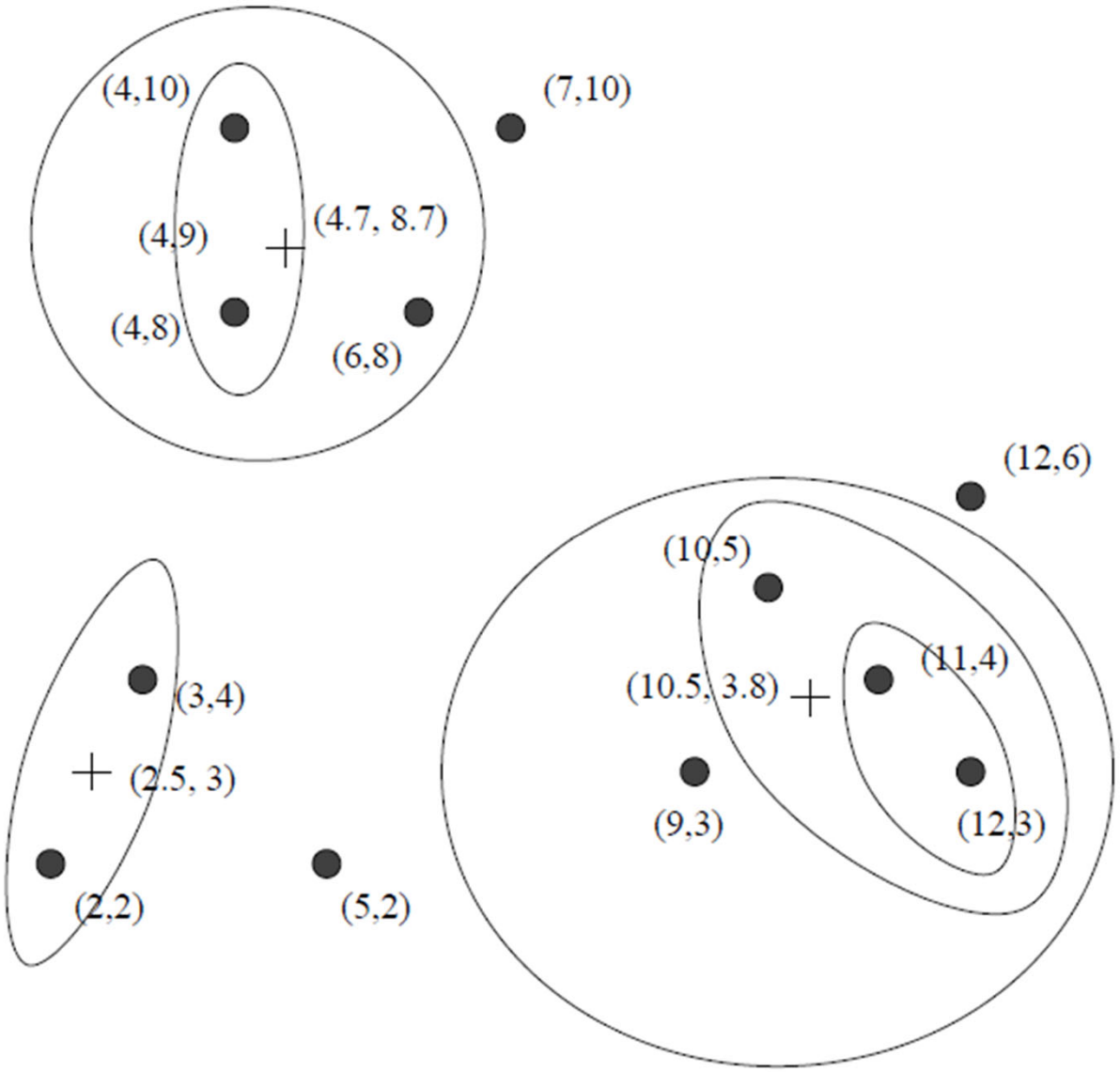
Hierarchical Clustering



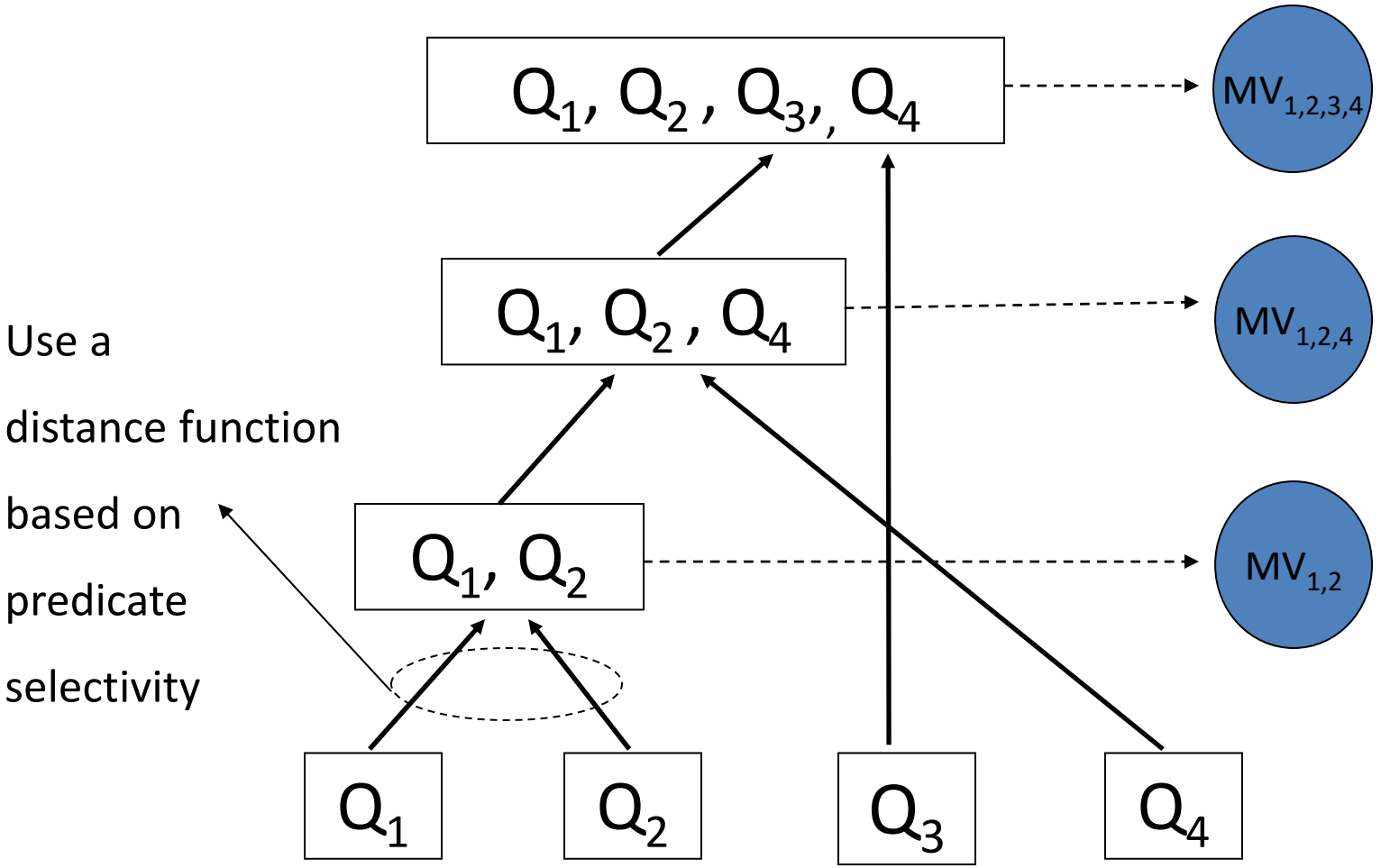
Hierarchical Clustering



Hierarchical Clustering



Merging Materialized Views: Hierarchical Query Clustering



Selectivity Vector for a SQL Query

SELECT SUM(*revenue*)

FROM lineorder, dwdate, part, supplier

WHERE … *category* = 'MFGR#12'

AND *nation* = 'CANADA' **AND** *year* = 1997;

(*category* : **0.27**, *nation* : **0.20**, *year* : **0.14**)

14%

(0.27, 1.0, 0.20, 0.14)

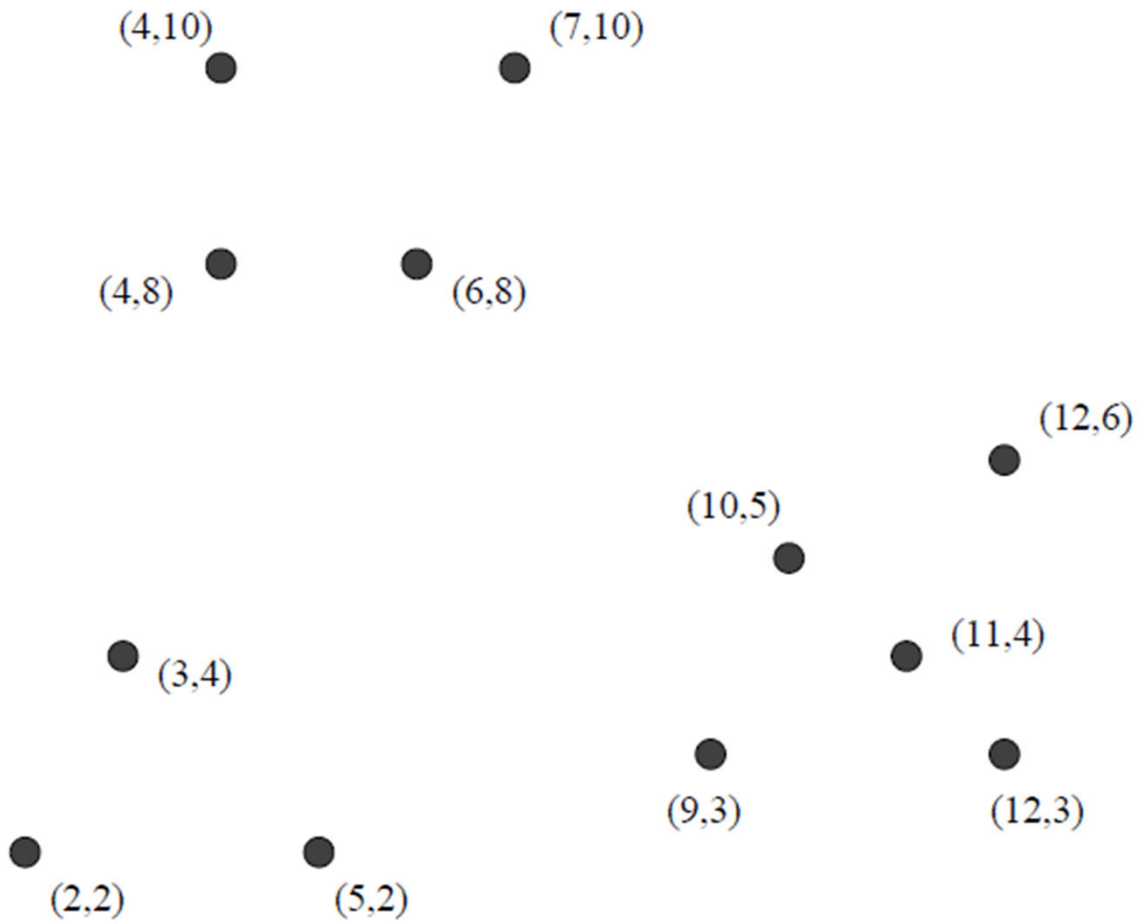
year = 1998

K-Means Clustering

- Choose an initial set of centers
  - Well distributed (likely in different clusters)
  - Converges*
- Assign every point to the closest centroid
- Find the new centroids
- Repeat
  - N steps
  - Or until converges

K-Means Clustering

- What is the best value for k?



K-Means Clustering

- http://shabal.in/visuals/kmeans/4.html

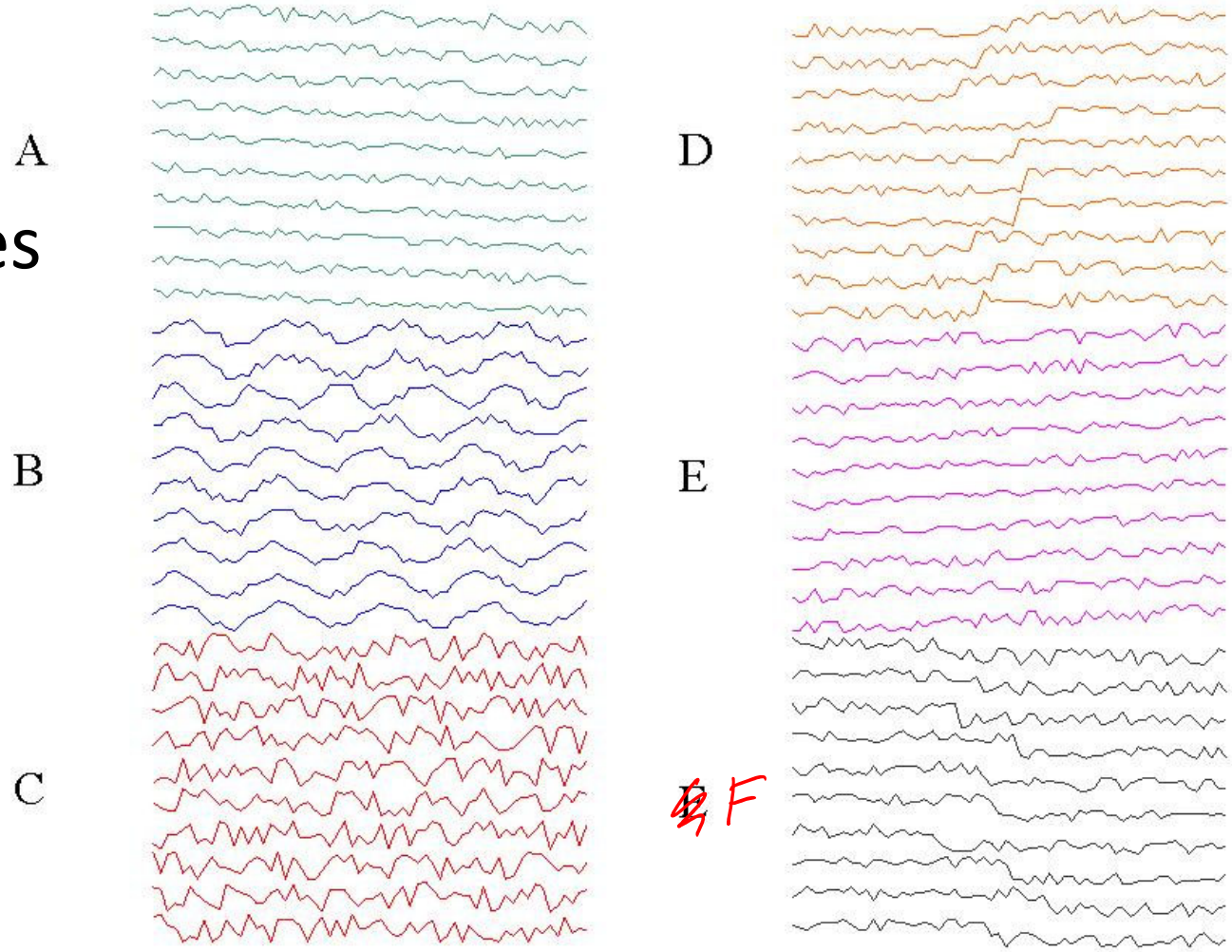
Mahout

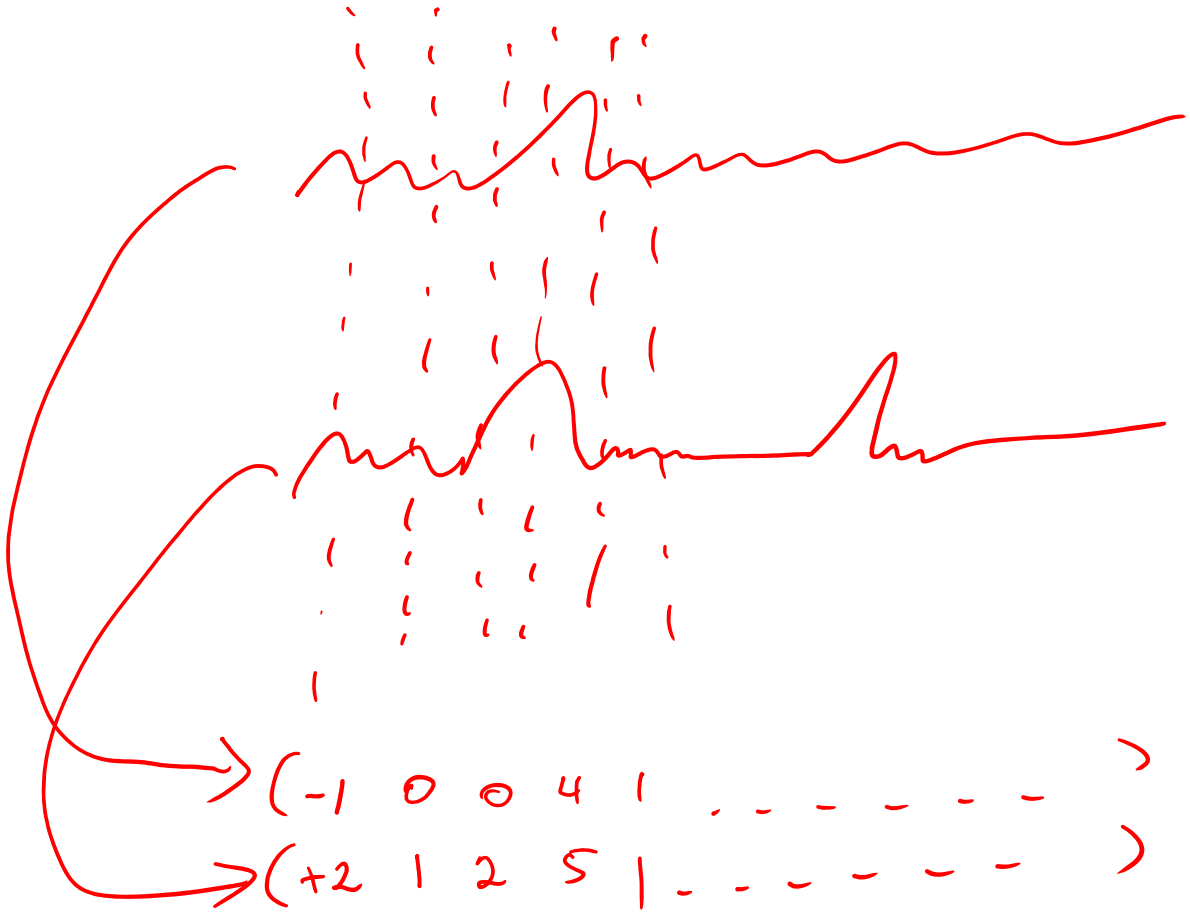
- Machine learning library
- Community contribution
- Provides a number of algorithms
  - Clustering
    - K-Means, Fuzzy K-Means, …
  - Recommenders
    - User and item based recommenders
  - Classifiers
- Runs on top of Hadoop/Spark/etc.

Mahout Example

600 items

6 categories

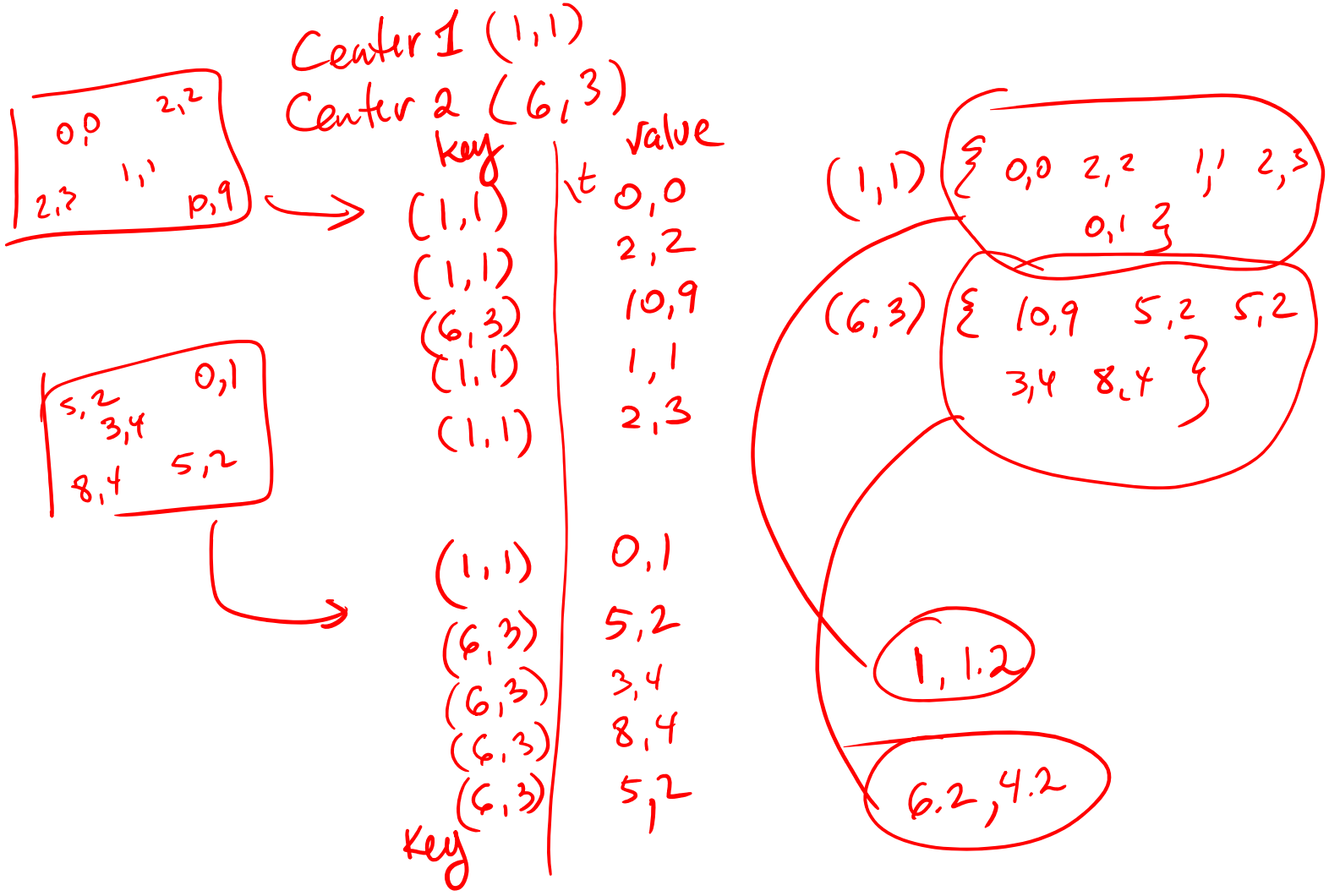



(-1 0 0 4 | . . . . .)
(+2 1 2 5 | . . . . .)

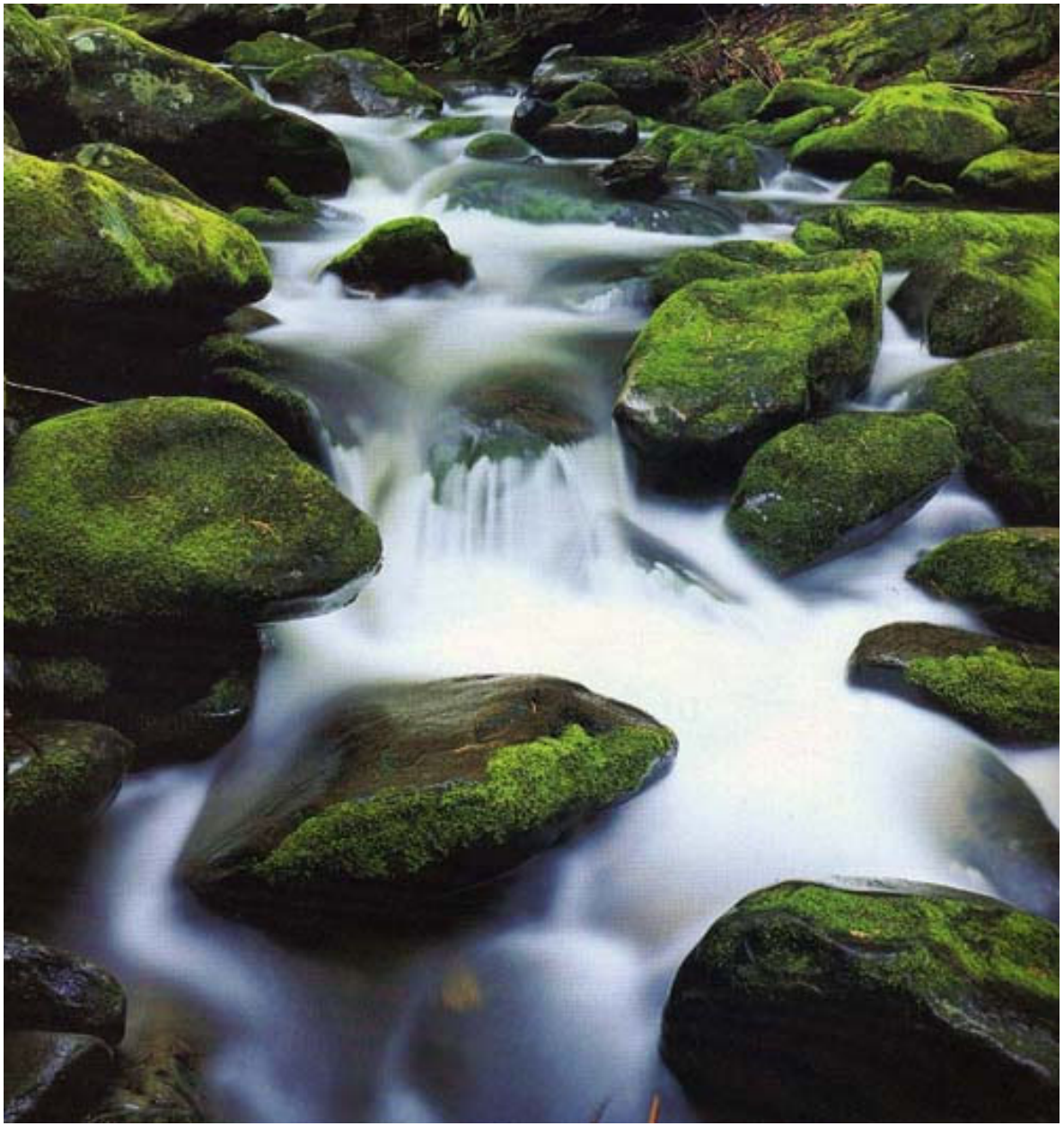
Example KMeans Run

$MAHOUT_HOME/bin/mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job

- Defaults – testdata, Euclidian vector dist, 10 iterations
- KMeans Mapper
  - Assign the point to nearest cluster
- KMeansCombiner
  - Partially aggregate the points towards the new centroid
- KMeansReducer
  - Receive all keys and compute the new cluster centroid
- KMeansDriver – iterate n times (or converge)


Center 1 (1,1)
Center 2 (6,3)
0,0
2,2
1,1
2,3
9,9
key
value
(1,1) 0,0
(1,1) 2,2
(6,3) 10,9
(1,1) 1,1
(1,1) 2,3
5,2
3,4
8,4
0,1
5,2
(1,1) 0,1
(6,3) 5,2
(6,3) 3,4
(6,3) 8,4
(6,3) 5,2
key
(1,1) { 0,0 2,2 1,1 2,3 0,1 }
(6,3) { 10,9 5,2 5,2 3,4 8,4 }
1,1.2
6.2,4.2

Streaming Database Engines

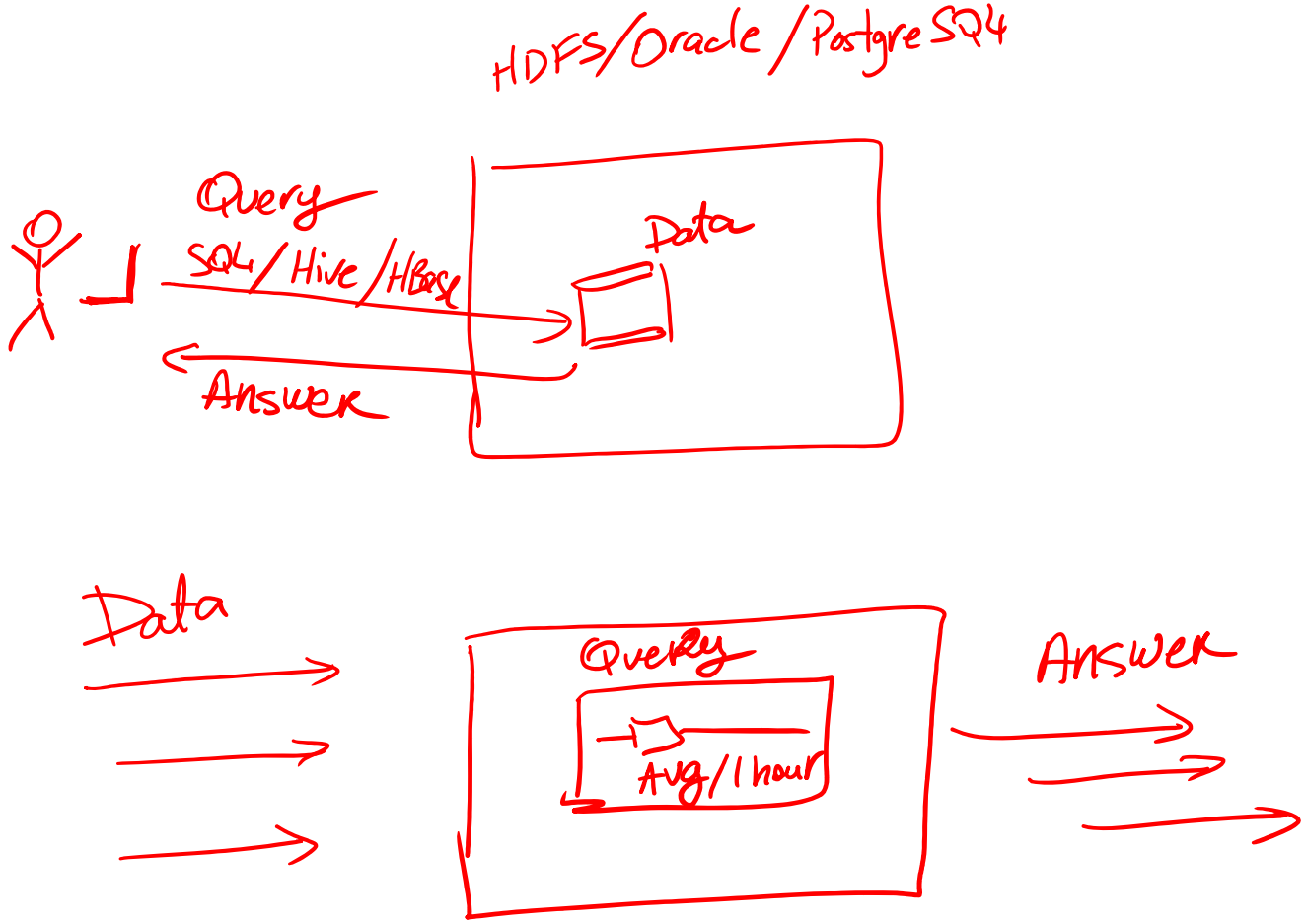


Data-Stream Management

- Traditional DBMS
  - Data stored in finite, persistent data sets
- Data streams
  - Distributed, continuous, unbounded, time-varying
- Data-Stream Management
  - Network monitoring
  - Data security
  - Sensor network
  - Financial applications

Streaming Data

- Types of data (infinite)
  - Sensor data
  - Stock market quotes
  - Network monitoring data
- Types of queries
  - One pass (cannot rewind or store)
  - Report last day average
  - Report changes in sensor data
  - Collect information about single user


HDFS/Oracle/PostgreSQL
Query
SQL/Hive/HBase
Data
Answer
Data
Query
Avg/1 hour
Answer

Data-stream-management System

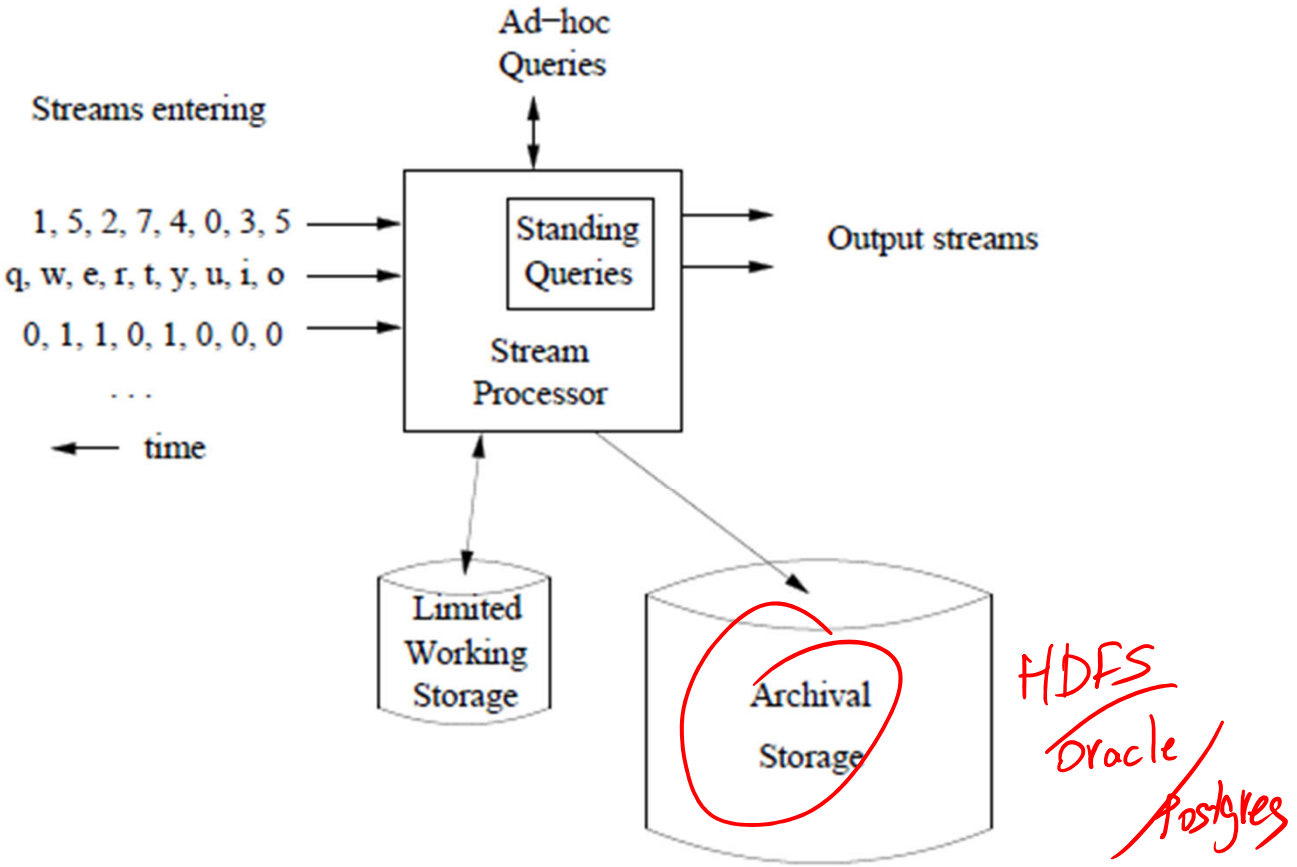
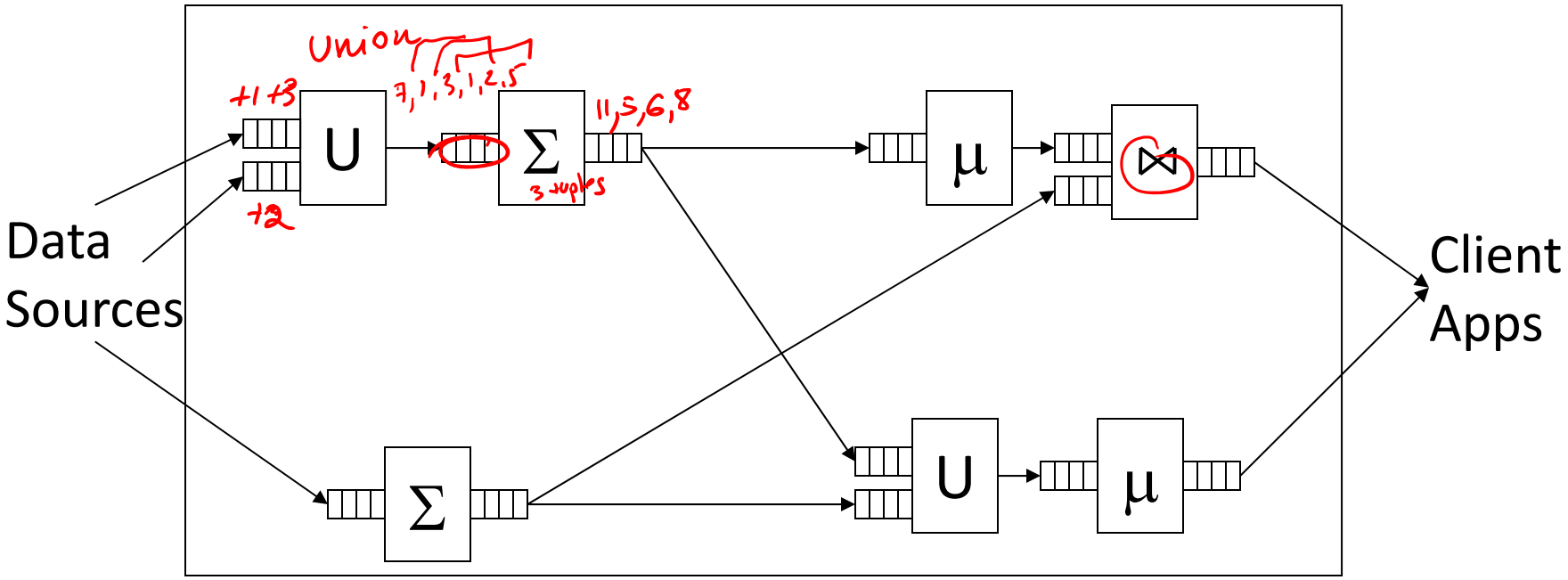


Figure 4.1: A data-stream-management system

Stream Processing



- Data sources push tuples continuously
- Operators process windows of tuples

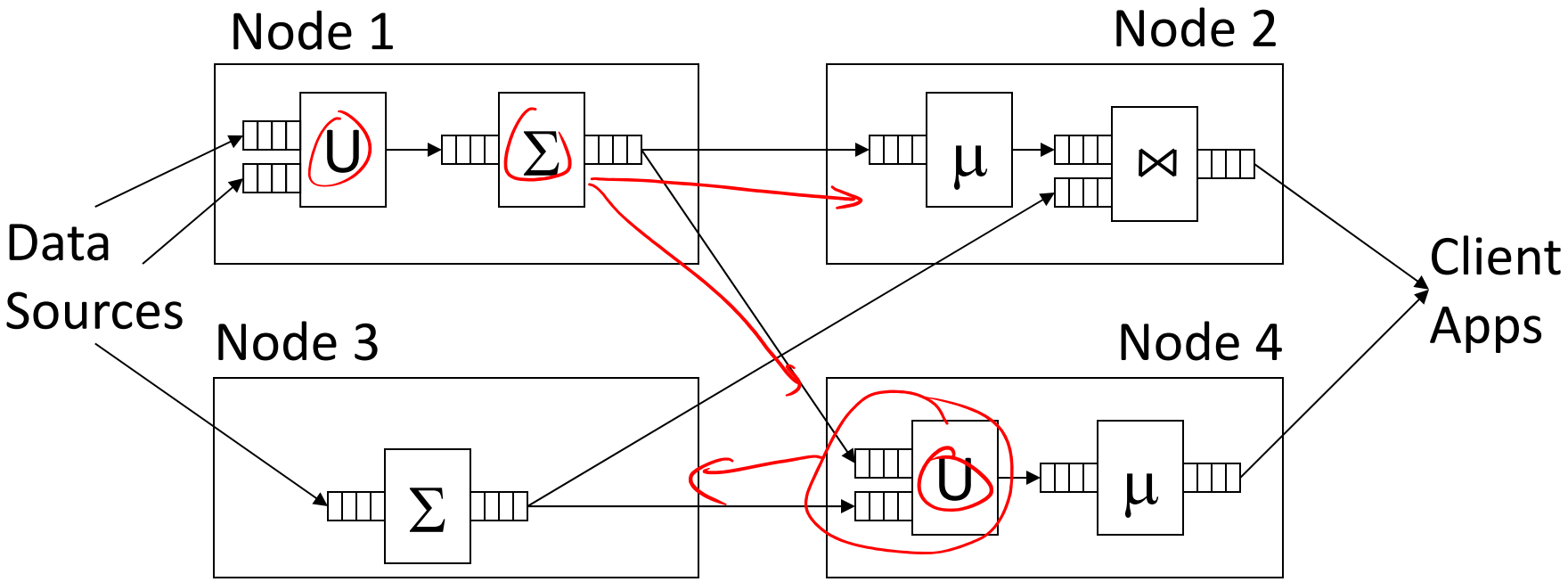
Examples

- Ocean temperature sensors
  - GPS+temperature every 1/10 second (3.5MB)
  - More sensors (x1000, or x1000000)
- Image data
  - London has millions cameras
- Internet monitoring
  - Intruder alert
  - Any real-time activity spike, traffic

Operations

- Filter the stream
  - E.g., select (temperature > 100)
- Aggregate the stream
  - # of cars entered the garage
    - Window (per-hour? Over last hour?)
- Join two streams
  - Match “buy” and “sell” offers

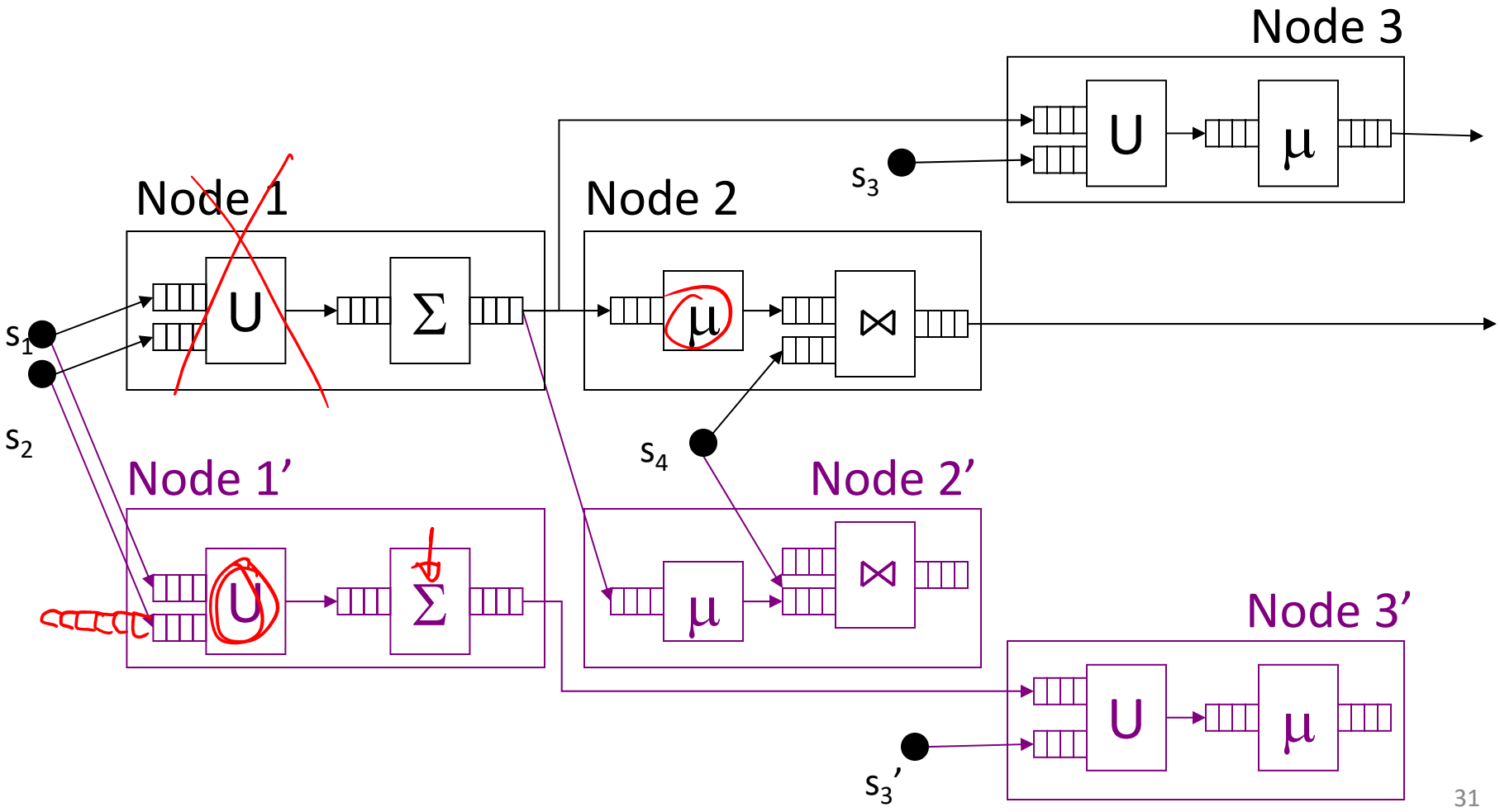
Distributed Stream Processing



- Data sources push tuples continuously
- Operators process windows of tuples

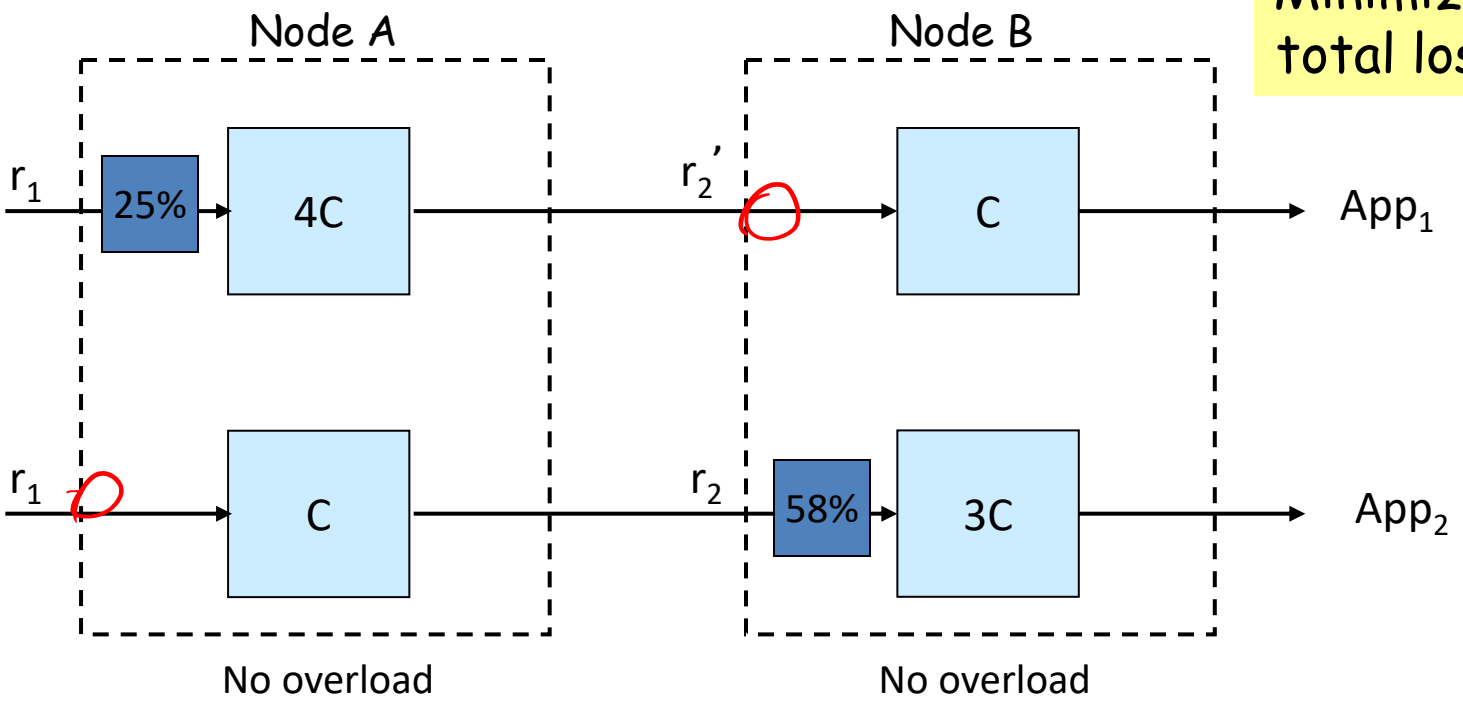
Fault-Tolerance through Replication

- Goal: Tolerate node and network failures



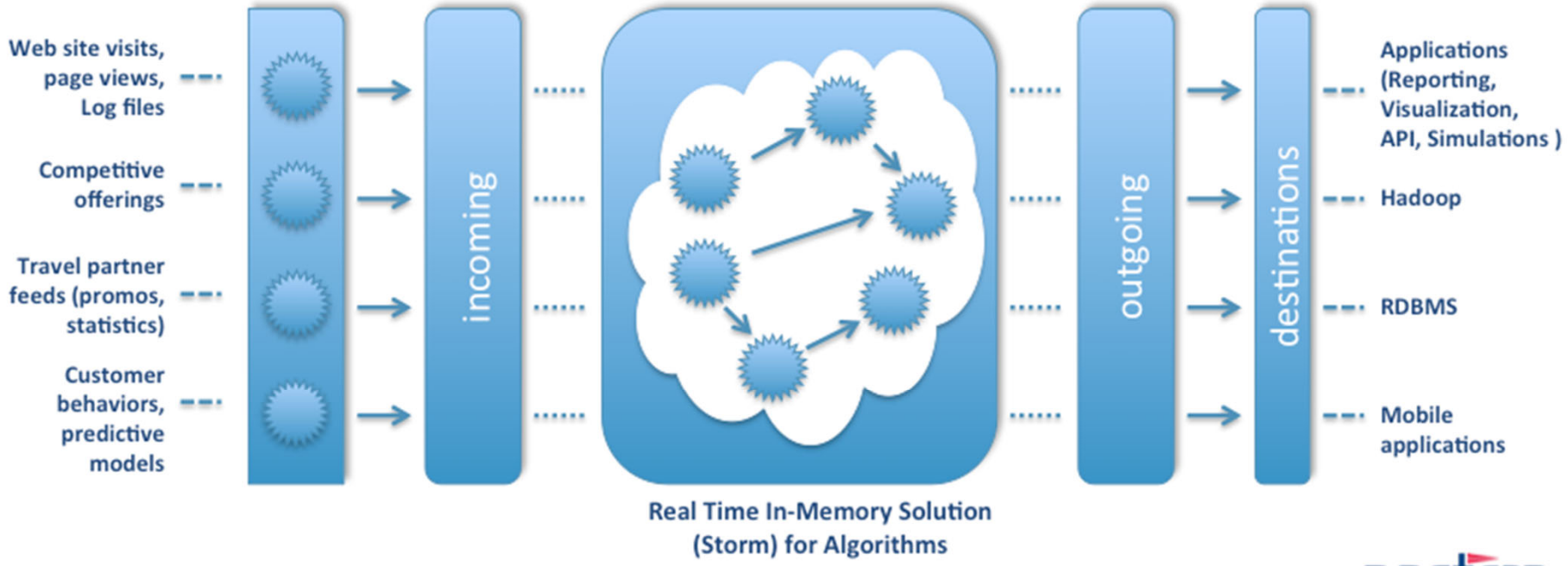
Load Shedding

Goal: Minimize total loss



Plan	*Loss*
Local	App_1: 25% App_2: 58%

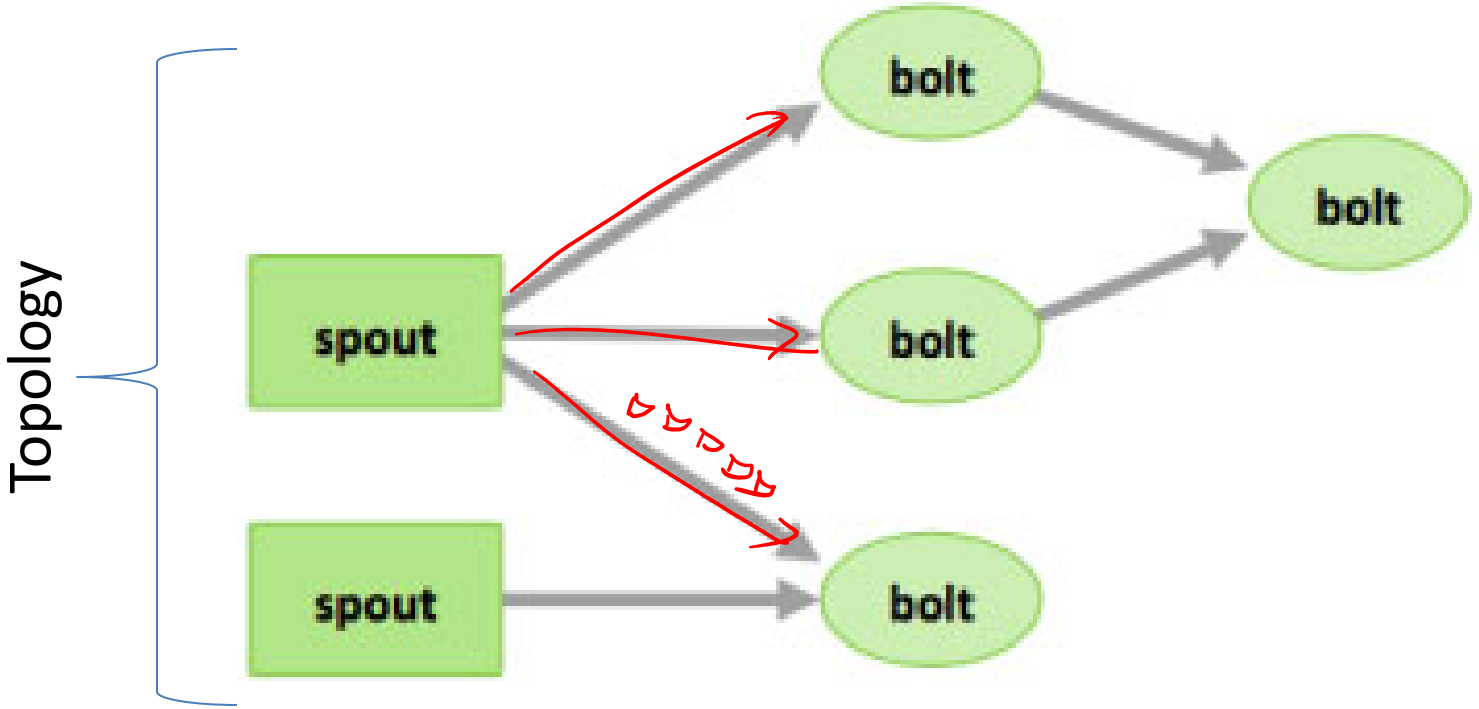
Storm



pactera

Storm Components

- Spout = source of tuples
- Bolt = processing operator

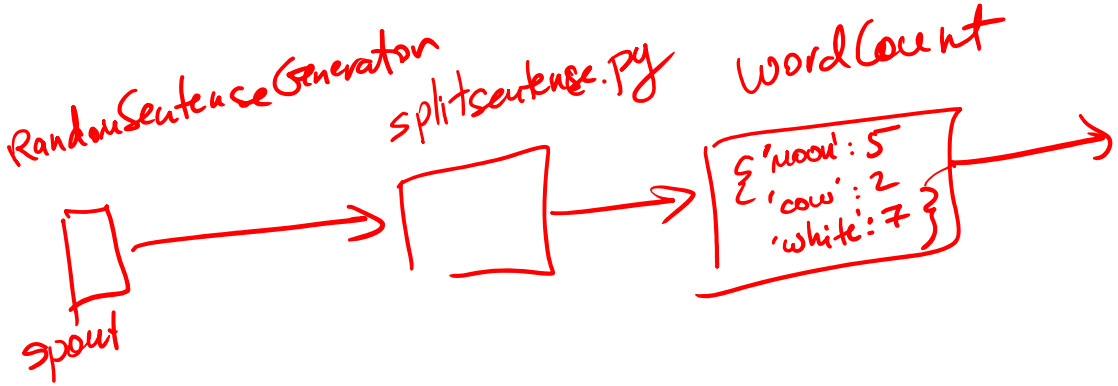


Storm Components

- Nimbus node
  - JobTracker
- Supervisor Nodes
- ZooKeeper Nodes
  - Coordination
- Stream
  - Unbound sequence of tuples

Running Storm

- Submit "topologies"
  - WordCountTopology.java
- Spouts / Bolts
- Can use any language



In-Memory Databases

Main Memory database system (MMDB)

- Data resides permanently on main physical memory
- Backup copy on disk

Disk Resident database system (DRDB)

- Data resides on disk
- Data may be cached into memory for access

  Main difference is that in MMDB, the primary copy lives *permanently* in memory

MMDB?

- *Is it reasonable to assume that the entire database fits in memory?*
  - Yes, for some applications
  - 100 machines x 64GB RAM = 6.4TB RAM
- *What is the difference between a MMDB and a regular database with a very large cache?*
  - Even if all data fits in memory, the structures and algorithms are designed for disk access.

Differences in properties of main memory and disk

- The access time for main memory is orders of magnitude less than for disk storage
- Main memory is normally volatile, while disk storage is not
- The layout of data on disk is much more critical than the layout of data in main memory

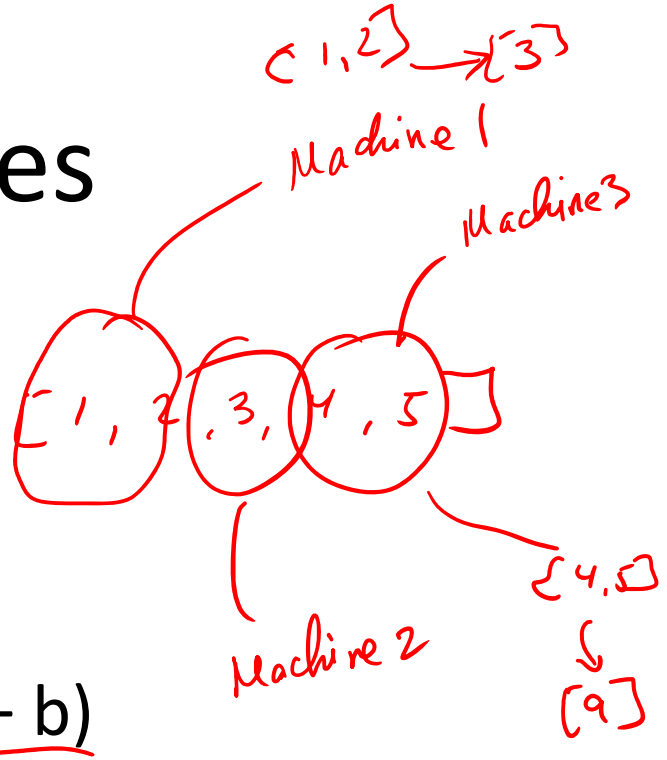
Spark

- General engine for large-scale data processing
  - Batch/real-time/stream
  - Machine learning
  - Graph workloads
- In memory processing
  - Resilient Distribute Dataset (RDD)
  - HDFS or in-memory*

Spark

- bin/pyspark (shell)
  - textFile = sc.textFile("README.md")
  - textFile.count() # number of items/lines
  - textFile.first() # first item
  - linesWithSpark = textFile.filter(lambda line: "Spark" in line)

Spark Principles



- Distributed datasets
  - data = [1, 2, 3, 4, 5]
  - distData = sc.parallelize(data)
  - distData.reduce(lambda a, b: a + b)
- distFile = sc.textFile("data.txt")
  - hdfs://… , s3n://
  - lengths = distFile.map(lambda s: len(s))
  - totalLength = lengths.reduce(lambda a, b: a + b)

Spark Setup

- Download Spark binary
  - http://spark.apache.org/downloads.html
- Mode
  - Using stand-alone mode
  - YARN
    - Spark pre-built with support, security

Spark Setup

- sbin/start-all.sh
  - Workers file
- http://public-IP:8080/
  - Cluster view

Spark Example

```
# Read file from HDFS
text_file = sc.textFile("hdfs://ec2-18-144-12-192.us-west-1.compute.amazonaws.com/data/README.md")
lengths = text_file.map(lambda s: len(s))
print text_file.take(100)

lengths.foreach(myprint)
print lengths.take(100)
totalLength = lengths.reduce(lambda a, b: a + b)
```