CS4225/CS5425 Big Data Systems for Data Science

Streams and Revision

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg

School of Computing

Announcements

- Practice test will be posted this week. For those who want to do the test in school, you can come to COM1-02-06 Seminar Room 1.
- HW2 is extended to 7 Nov (Sunday) 11.59pm

Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	19 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	
5	9 Sep	NoSQL Overview 1		Assignment 1 released
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 2 released
8	7 Oct	Apache Spark 2		Assignment 1 due (10 Oct)
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	
12	4 Nov	Deepavali – No Class		Assignment 2 due (7 Nov)
13	11 Nov	**Test**		

Test: Instructions

- Test is on 11 Nov (2 weeks from now), during lecture time
- Administrative details:
  - 6.30pm: please log in to the Zoom call
  - 6.35pm: download test document (as .docx) from LumiNUS > Files > Test. Once everyone is ready, the test will start, and will last for 90 minutes
  - Up to 8:20pm: extended time given as necessary for paper uploading.
- Follow the SOP in the following link (but no need to use Examplify)
  - https://mysoc.nus.edu.sg/academic/e-exam-sop-for-students/
- Submission:
  - Save the document as PDF (Save As > PDF)
  - Filename should be: A1234567Z.pdf (replace with your student ID)
  - Upload this file to LumiNUS > Files > Test Submission
  - Take the screen video (see SOP above) and submit it within two days after the final test to: LumiNUS > Files > Test Video Submission. Name your video, e.g.: A1234567Z.mp4.

Instructions (cont'd)

- Total marks: 50 marks, 50% of final grade
- The test is open book / laptop / internet – you can refer to any resources. However, do not discuss/share/copy with other students. We have zero tolerance on plagiarism and cheating.
- Practice test(s) will be released by this weekend.

If something goes wrong during test…

- What if my network fails? Save your file before the deadline (8:20pm). It is fine as long as the last-modified timestamp of your PDF submission is before 8:20pm.
- I will be standby on Zoom if you have any questions.
  - Please turn on Zoom video so that we can see each other during the exam.

More Questions

- How to fill in answers in the question sheet?
  - It is recommended to type directly in the .docx file. The questions are designed such that they can be answered easily in plain text. But if there are difficulties with this process, it is fine to print out the exam, handwrite and scan.
- Any other questions?

Types of Questions

- Example questions
  - Integrative: Require you to combine knowledge from different chapters of the textbook
  - "Application": Require you to apply your knowledge of fundamental concepts to reasonably practical scenarios.
  - "Why not": Example, Tommy proposed a solution A to solve problem B in the lecture. Tell me what is the problem with solution A and how to overcome this problem
- In general: if you have successfully understood the concepts we have discussed in lecture, you should be able to answer the test questions.
  - General focus is on understanding of concepts / principles and understanding when various algorithms / systems should be used; not on knowing particular details of specific software implementations

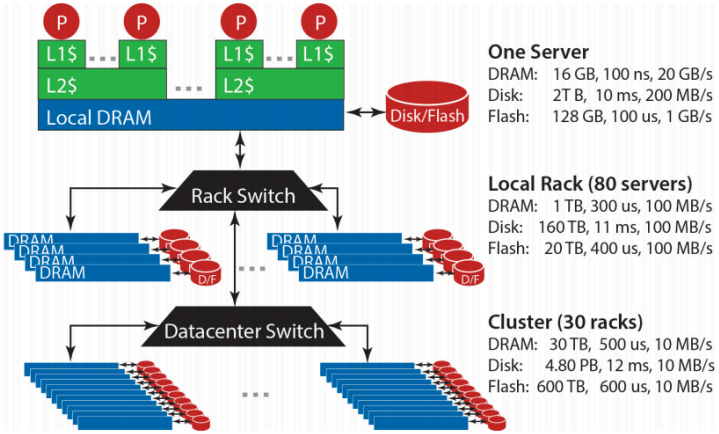
Scope of Test

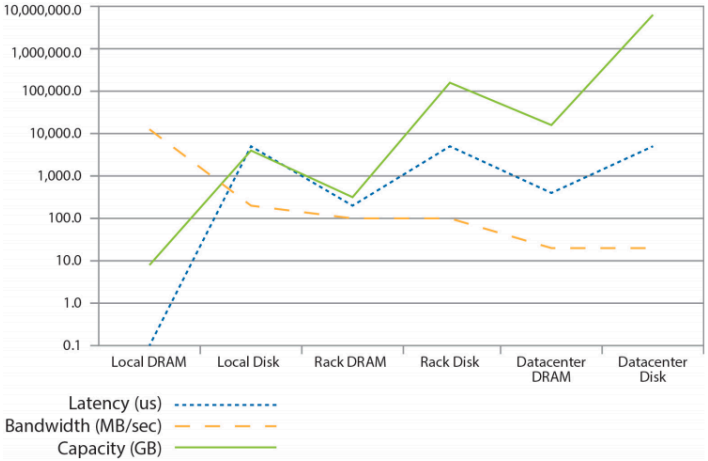
- **Scope**: content discussed in the lecture
- **Out of scope**:
  - Lab-only content (assuming it was not discussed in lecture)
  - Content only found in the notes underneath the slides (the notes can be ignored)
  - Content only discussed in response to student questions (i.e. beyond the scope of the slides)
  - Content marked with "Details" or that I mentioned is out of scope
  - If there are any tasks which ask for code, they will allow pseudocode
  - Historical details

- In the following, I will
  - Revise the key points we talked about during the semester
  - Go through a few example questions (during tutorial session)

Introduction to Data Science

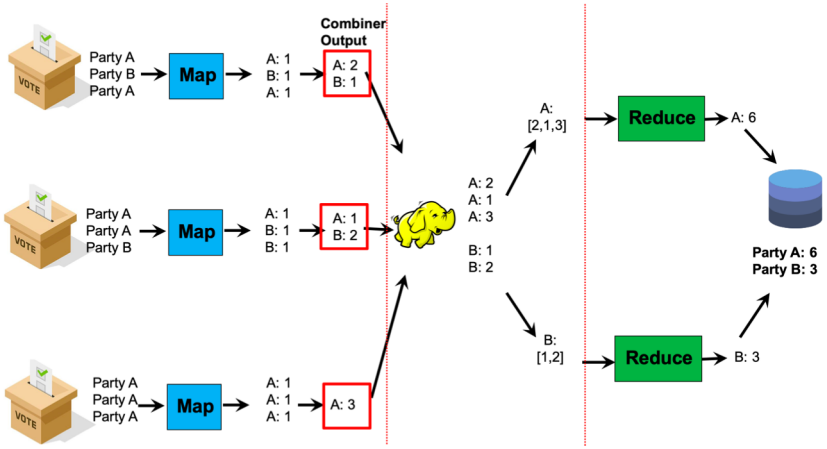
- What is (big) data science?
  - 4V big data challenges
- *Infrastructure* for big data
  - Data center architecture
  - Storage hierarchy in a data center
  - Capacity / latency / bandwidth of moving data in a data center
    - General / conceptual understanding is sufficient, no need exact values
  - "Datacenter as computer" + "Big Ideas" (scale out, move processing to data, sequential processing, scalability)

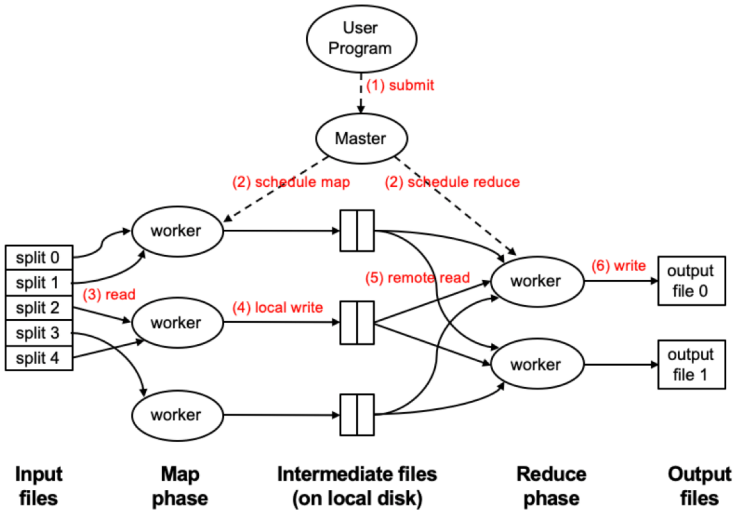




MapReduce

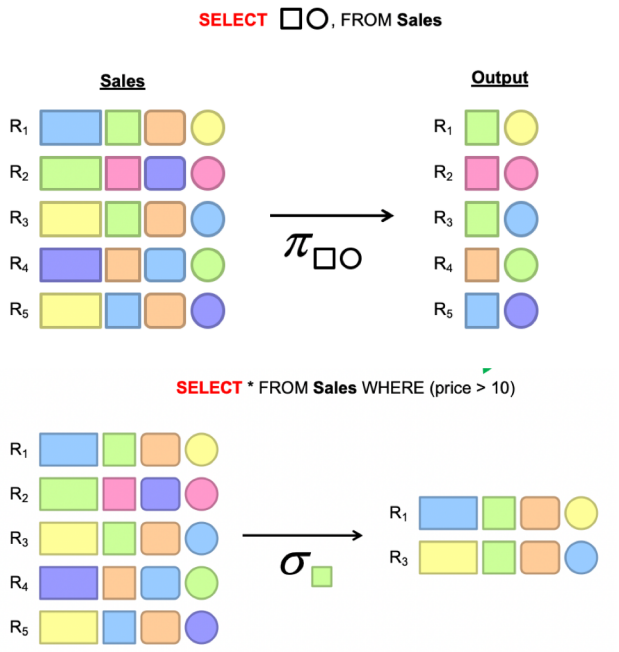
- System design principles
  - Why MapReduce?
- Basic algorithmic design
  - Performance analysis: scalability, network and disk I/O, memory working set. You should be able to apply these to analyze the
- Basic infrastructure (local write, remote read)
- Correct design of partitioners and combiners

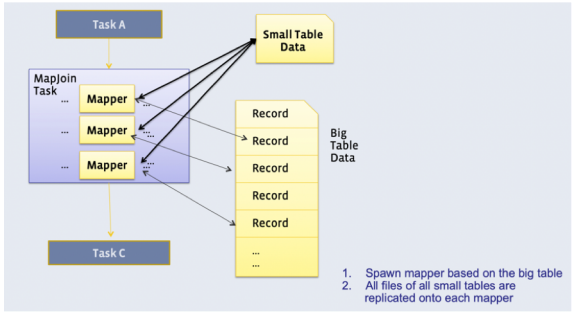


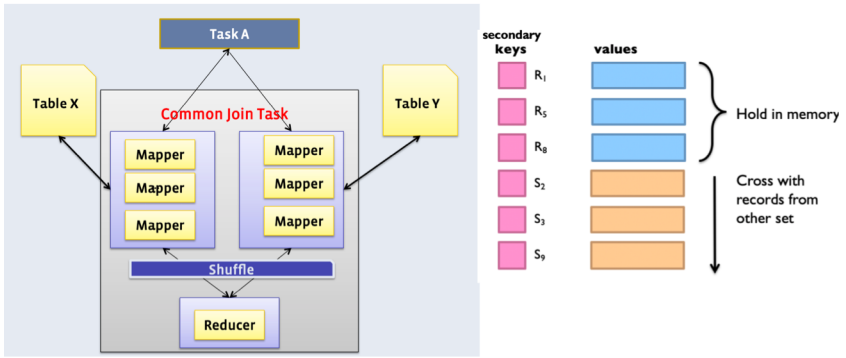


MapReduce and Relational Databases

- Projection
- Selection
- Joins
  - Map Join
  - Reduce Side Join

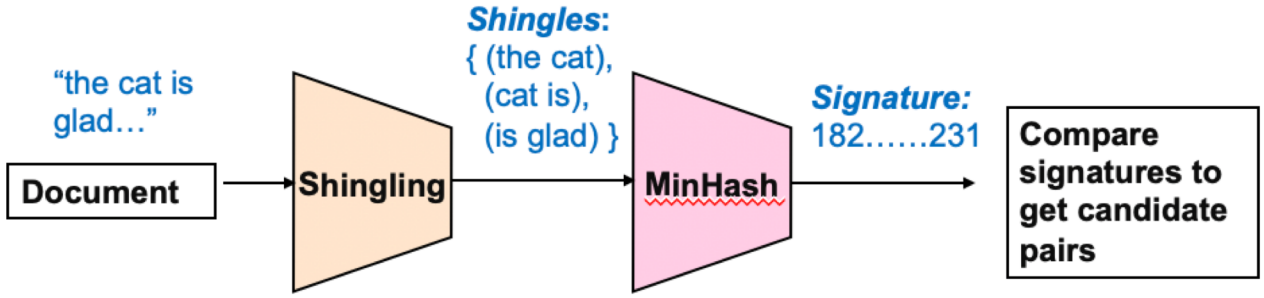


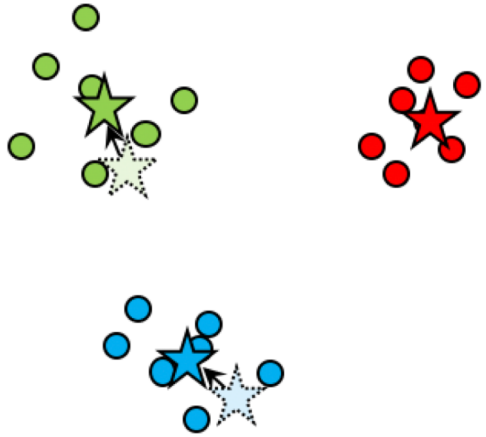




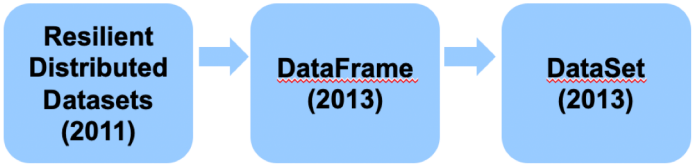
MapReduce and Data Mining

- Similarity Search
  - Similarity measures
  - Shingling
  - MinHash
- K-means clustering
  - MapReduce implementation

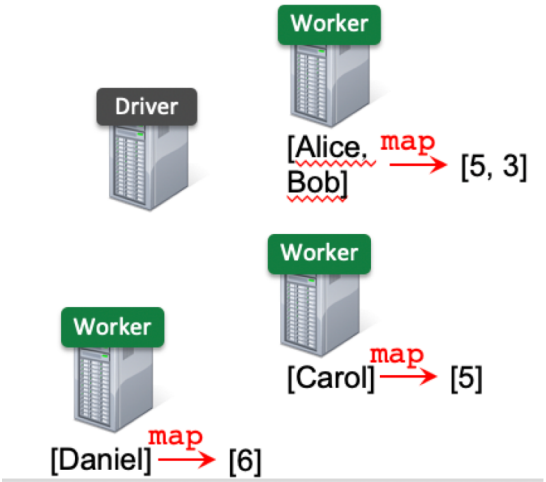


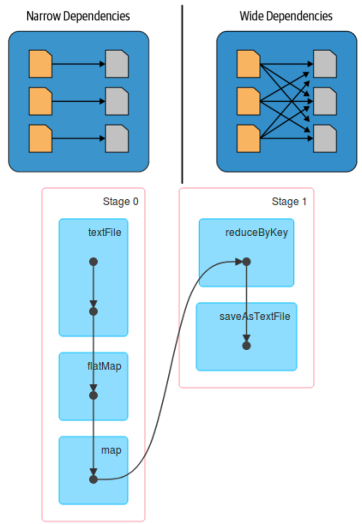


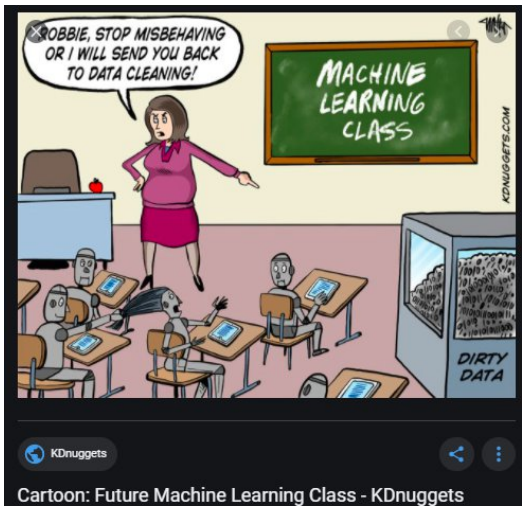
NoSQL & Spark



- NoSQL: types of systems, key concepts, pros and cons
- Spark basic concepts: Spark vs MapReduce; RDDs, DataFrame, Dataset, transformations & actions, laziness, caching, DAG
- Spark internals: how data is partitioned / processed
- Performance analysis of Spark: scalability, network and disk I/O, memory working set
- Basic ML pipeline and MLLib
  - Out of syllabus - logistic regression and decision trees (but: I suggest reviewing them if you go for data science / data analytics job interviews)

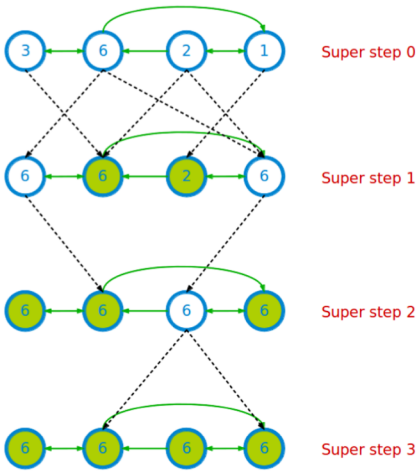


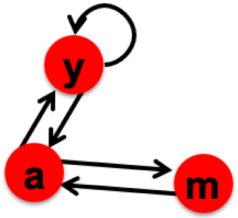




Graphs and Graph Processing

- PageRank
  - Conceptual understanding / interpretation of PageRank, why teleports are needed, computation of PageRank formulas
  - Topic-Sensitive PageRank, TrustRank
- Graph systems (Pregel / Giraph)
  - Basic graph programming model ('think like a vertex', how data is partitioned via edge cut, `compute()` function, vote to halt)





$$A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

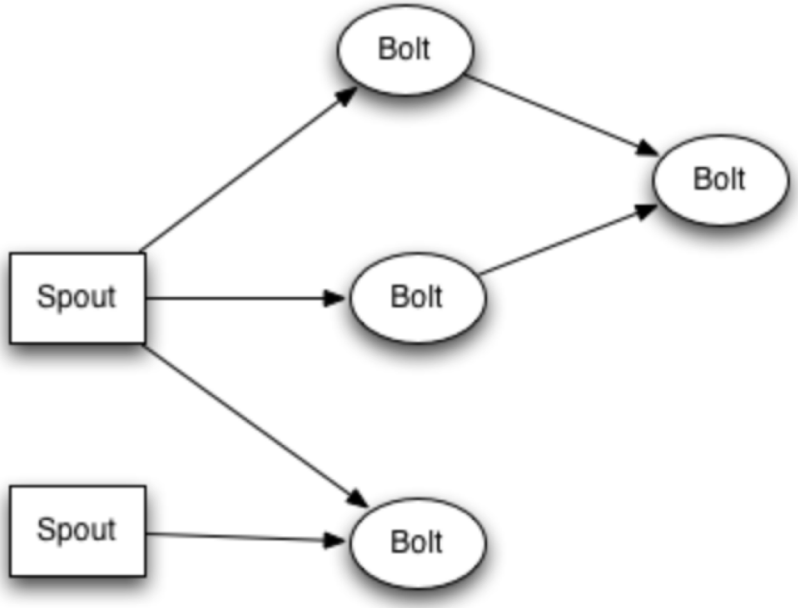
N by N matrix
where all entries are 1/N

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

	y	a	m
y	⅓	⅓	⅓
a	⅓	⅓	⅓
m	⅓	⅓	⅓

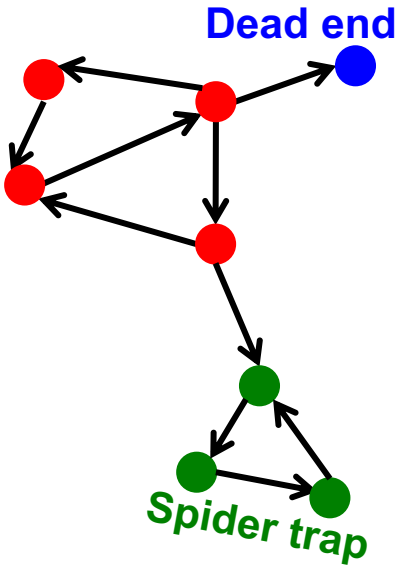
Stream Processing

- Understand the goal of data streams and when to use them
- Reservoir sampling
- Understand basic concepts of Storm programming model (spouts & bolts, stream groupings)



Recap: Dead Ends, Spider Traps

- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"



- **(2) Spider traps:** (all out-links are within the group)
  - Random walk gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Recap: The Google Matrix

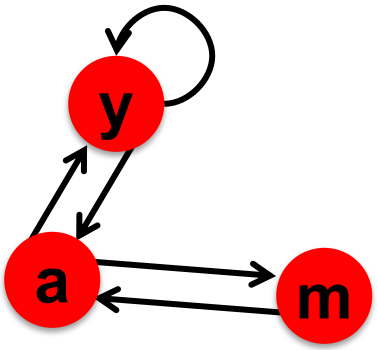
- **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

- We can also write this as a matrix equation, by defining the **Google Matrix *A***:

$$A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

N by N matrix
where all entries are 1/N



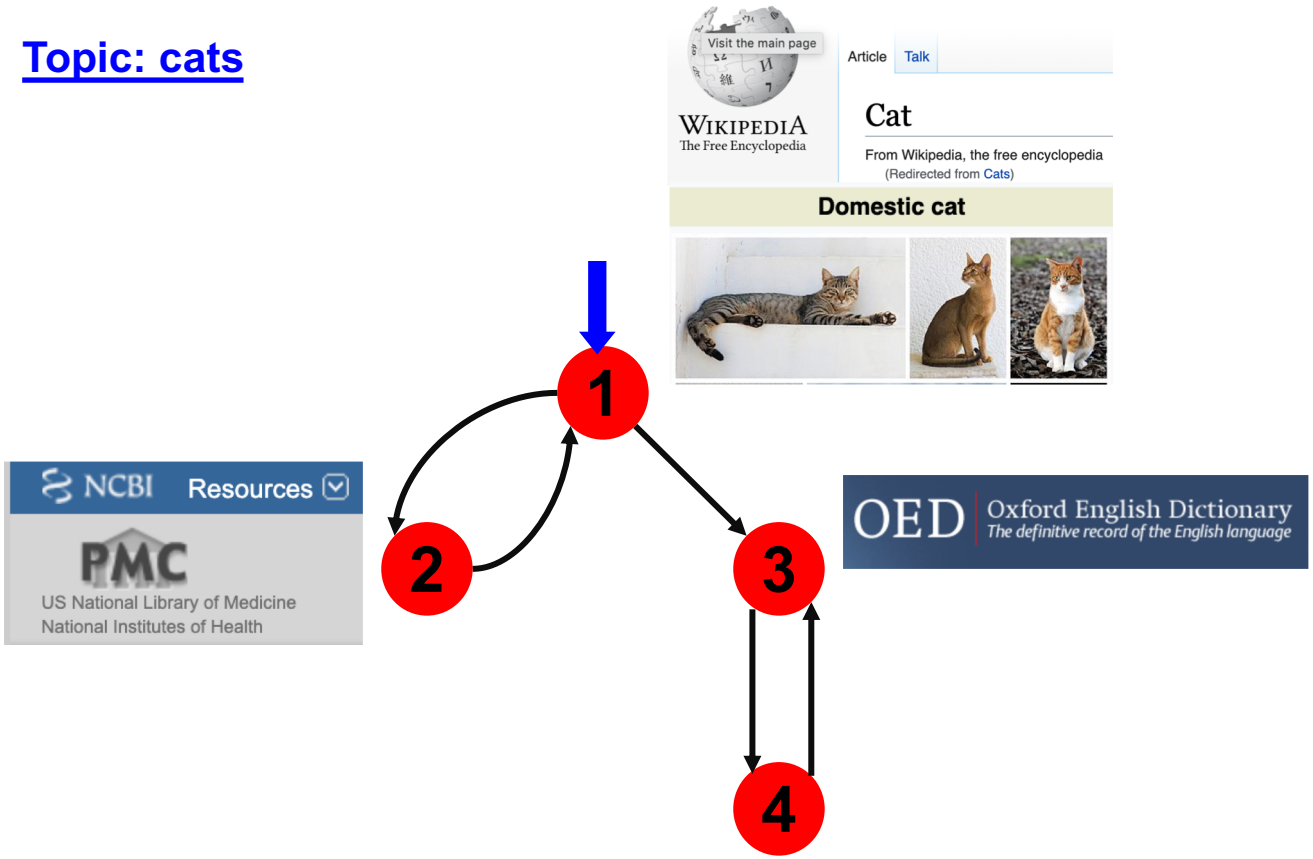
	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

	y	a	m
y	⅓	⅓	⅓
a	⅓	⅓	⅓
m	⅓	⅓	⅓

Recap: Topic-Specific PageRank

Example: S = Wikipedia page on cats

Topic: cats



- **Idea:** By setting the teleport set S as a small set of relevant pages for a topic, higher importance is assigned to pages in S and the web pages closely linked to S

Recap: Topic-Specific PageRank

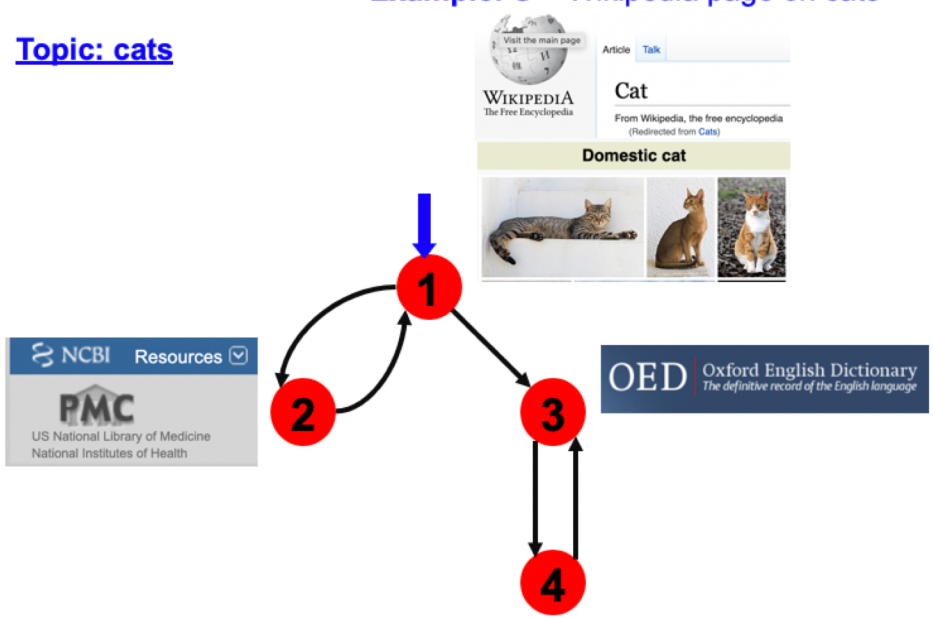
- **To make this work all we need is to update the teleportation part of the PageRank formulation:**

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\ \beta M_{ij} + 0 & \text{otherwise} \end{cases}$$

  - ***A*** is stochastic!

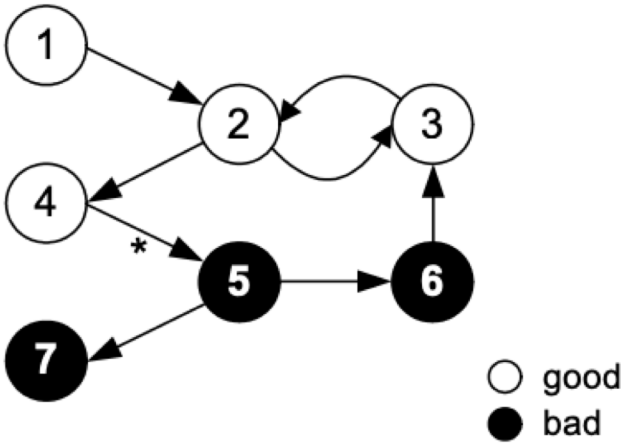
- **Compute using power iteration, like regular PageRank**

Example: S = Wikipedia page on cats



Recap: TrustRank

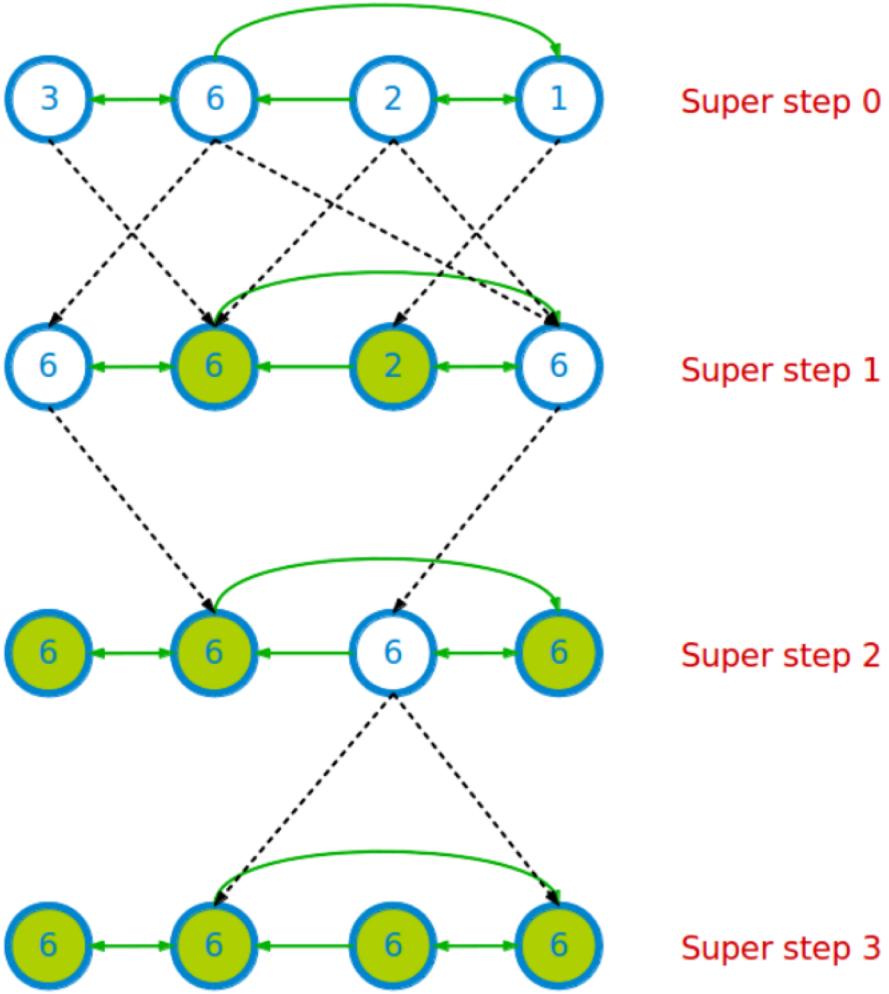
- **Goal**: identify trusted vs untrusted pages (e.g. spam pages). Start with a small set of trusted pages and propagate trust



- **Approach:**
  - Sample a set of **seed pages** from the web
  - Have an **oracle** (**human**) to identify the good pages and the spam pages in the seed set
  - Perform a topic-sensitive PageRank with **teleport set = trusted pages**

Recap: Pregel Computational Model

```
Compute(v, messages):
  changed = False
  for m in messages:
    if v.getValue() < m:
      v.setValue(m)
      changed = True
  if changed:
    for each outneighbor w:
      sendMessage(w, v.getValue())
  else:
    voteToHalt()
```

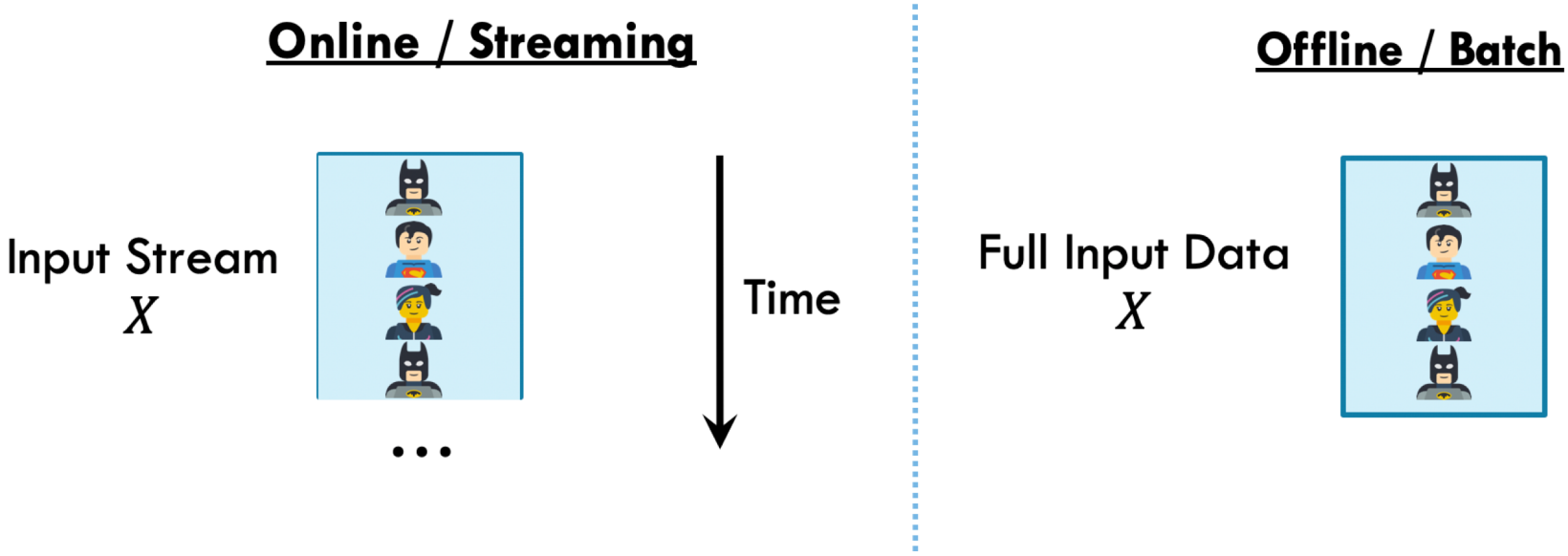


Today's Plan

- **Streams: Introduction and Data Model**
- Stream Algorithms
  - Reservoir Sampling
  - Running Mean and Variance
- Stream Data Processing Systems: Storm

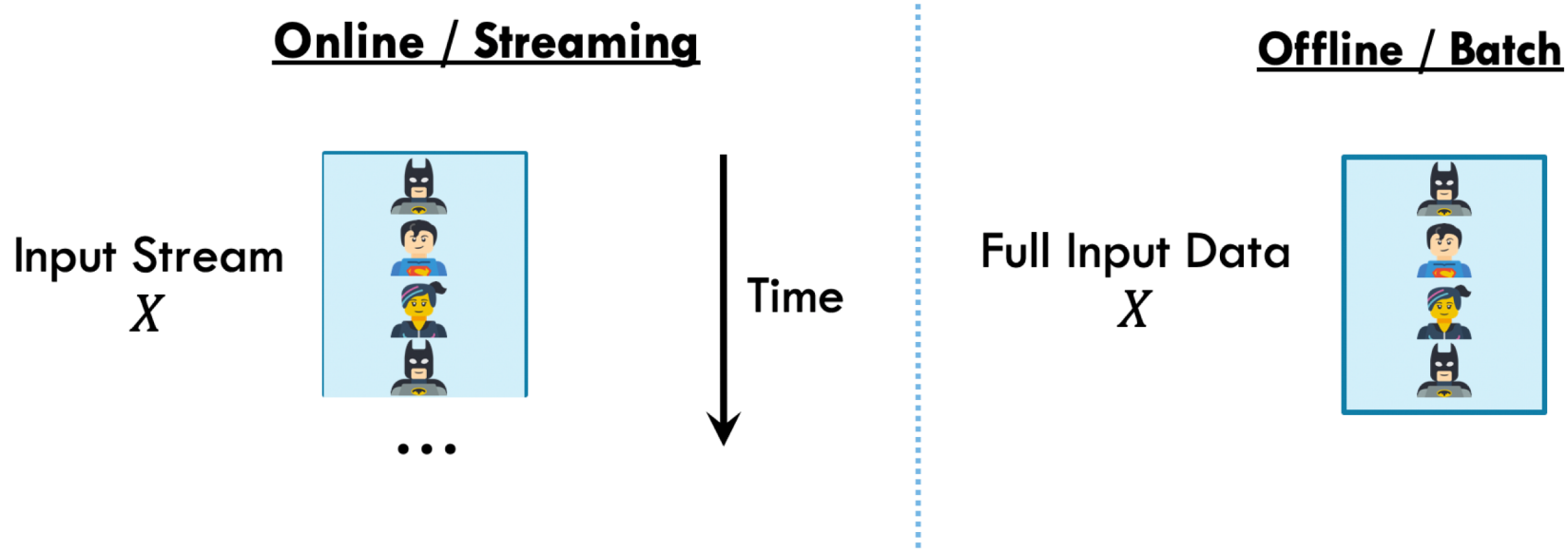
Streams: Motivation

- In many settings, the data is **arriving over time**; it is not received all at once
  - Streaming (or "online") approaches are designed to process their input **as it is received**
  - This is in contrast to offline or batch approaches that operate on the full dataset, all at once



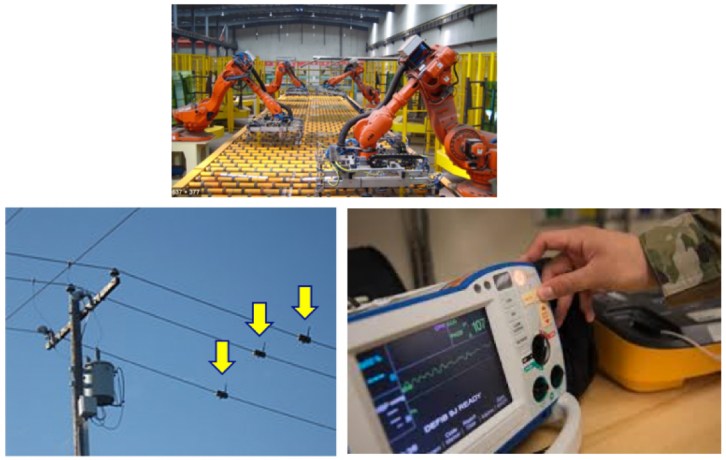
Streaming Data

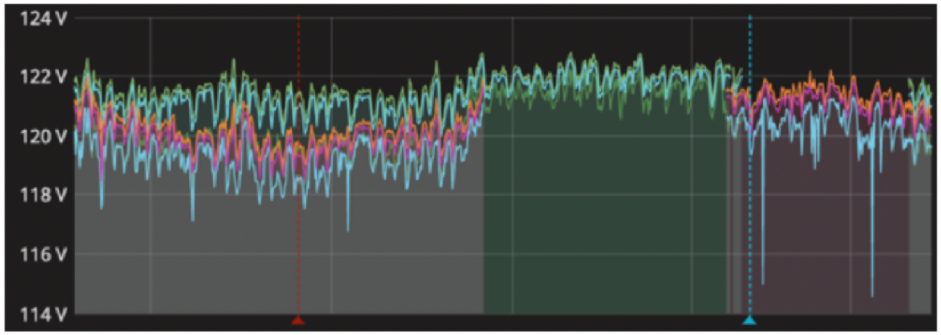
- Input elements enter at a rapid rate from **input ports** (e.g. receiving data from a sensor, from a TCP connection, or from a file stream)
  - Elements of the stream are sometimes referred to as 'tuples'
  - The stream is potentially infinite; **the system cannot store the entire stream accessibly** (due to limited memory)



Example Applications

- In many real settings, data is **high volume** and **constantly arriving over time**





Sensor data (industrial, environmental, medical, etc)

Example Applications

- In many real settings, data is **high volume** and **constantly arriving over time**

Online user activity data

Example Applications

- In many real settings, data is **high volume** and **constantly arriving over time**

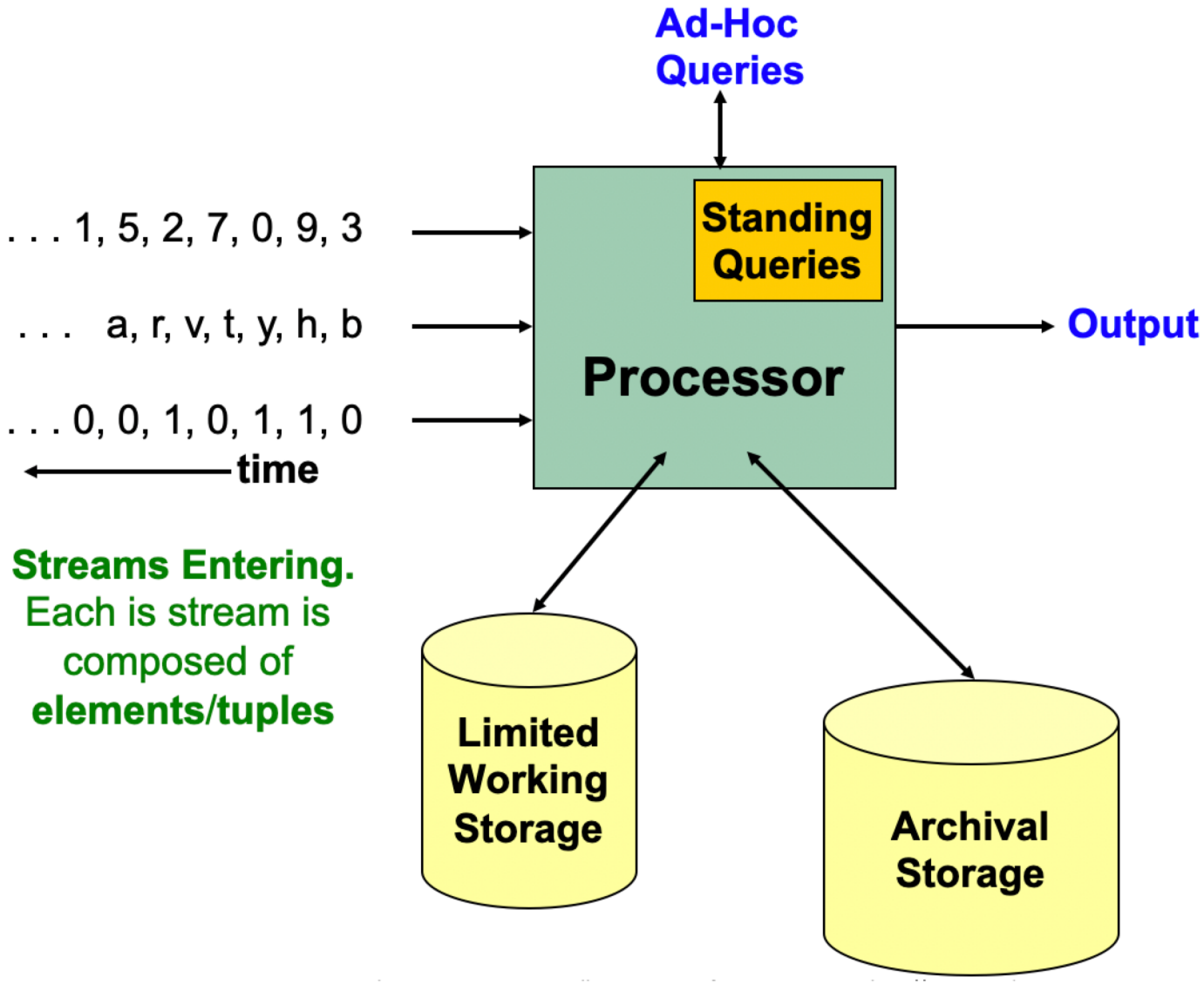
Financial Transactions
(e.g. credit card fraud detection)

Stock Trades Data
(e.g. trades, orders, prices)

https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs

02 Sep 2020	OMO STORE	S$3.30
02 Sep 2020	FACEBK [redacted] V2 650-5434800 CA USD2.00	S$2.81
02 Sep 2020	FACEBK [redacted] 650-5434800 CA USD3.00	S$4.22
02 Sep 2020	FACEBK [redacted] 650-5434800 CA USD2.00	S$2.81
02 Sep 2020	FACEBK [redacted] 650-5434800 CA USD3.00	S$4.22
02 Sep 2020	FACEBK [redacted] 650-5434800 CA USD5.00	S$7.03
02 Sep 2020	FACEBK [redacted] 650-5434800 CA USD50.00	S$70.45
02 Sep 2020	FACEBK [redacted] 2 650-5434800 CA USD10.00	S$14.10
03 Sep 2020	FACEBK [redacted] 50-5434800 CA USD50.00	S$70.45
03 Sep 2020	FACEBK [redacted] 650-5434800 CA USD75.00	S$105.84
03 Sep 2020	FACEBK [redacted] 650-5434800 CA USD1.33	
03 Sep 2020	OMO STORE	S$3.00
03 Sep 2020	GRAB*GRAB* [redacted] SINGAPORE SG	S$50.00

General Stream Processing Model



Today's Plan

- Streams: Introduction and Data Model
- **Stream Algorithms**
  - **Reservoir Sampling**
  - Running Mean and Variance
- Stream Data Processing Systems: Storm

Reservoir Sampling

- **Given**: a stream of items $\{a_1, a_2, ...\}$
- **Goal**: maintain a '**uniform random sample'** of fixed size B over time
  - **Uniform random sample**: each item has the same probability of being sampled
- **What's the benefit?** Allows us to estimate almost any statistics of the data distribution in a streaming manner: e.g. mean, variance, median etc.
  - In many large data scenarios, approximations using a reasonable sized sample are accurate enough
  - **Example:** we may want to standardize data as it arrives, based on such running statistics over time

Reservoir Sampling: Example

Stream:

$t = 1$ 

$t = 2$ 

$t = 3$ 

$t = 4$ 

$t = 5$ 

Reservoir ($B = 2$):

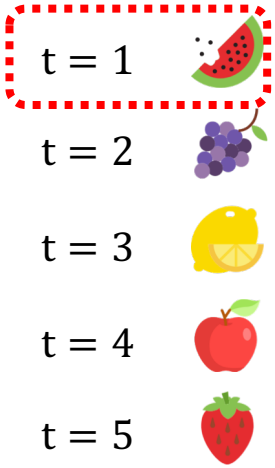
Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

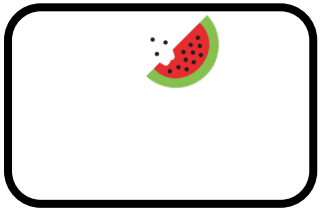
1. The first B items are inserted into the reservoir
2. When receiving a_t ($t > B$):
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

Stream:



Reservoir (B = 2):



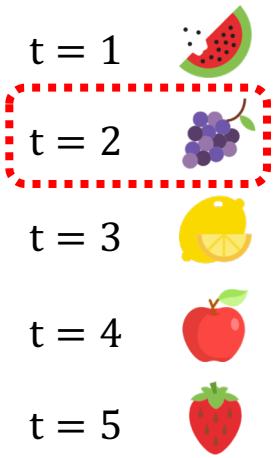
Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

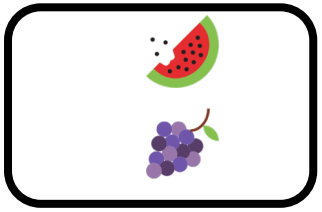
1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

Stream:



Reservoir (B = 2):



Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

Stream:

Desired Probability

$t = 1$  $2/3$

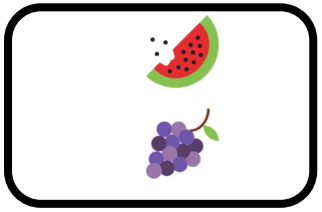
$t = 2$  $2/3$

$t = 3$  $2/3$

$t = 4$ 

$t = 5$ 

Reservoir ($B = 2$):

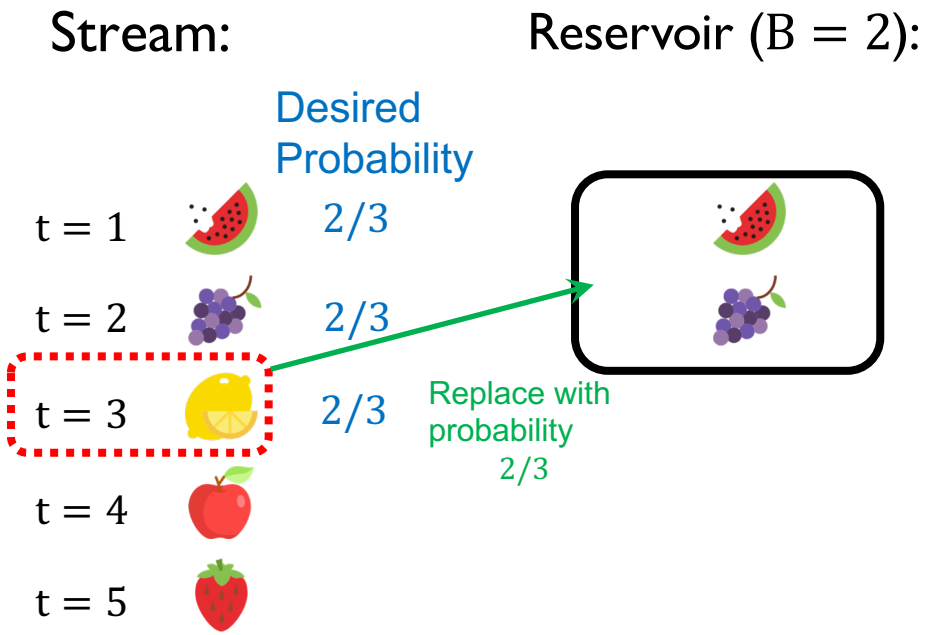


Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t ($t > B$):
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

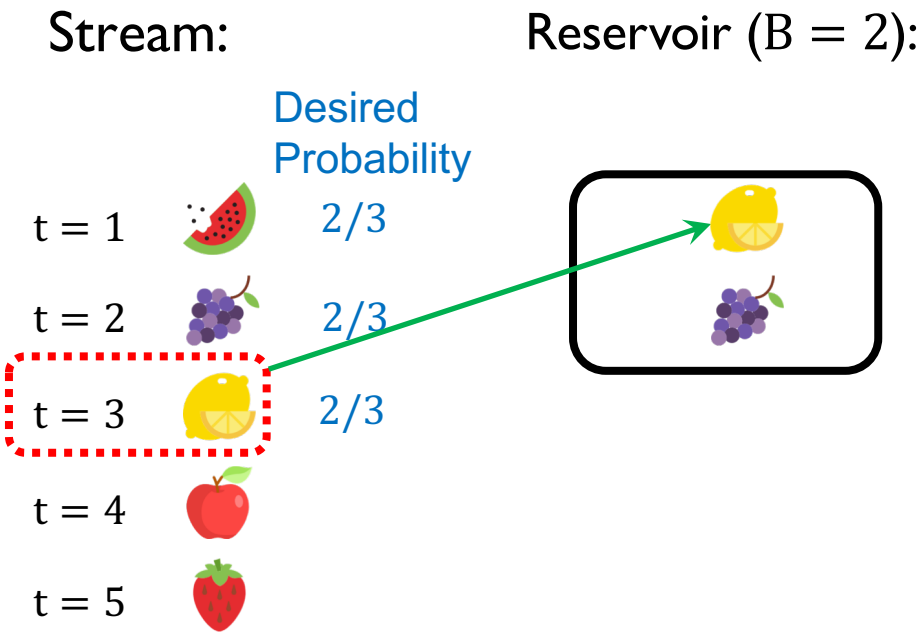


Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

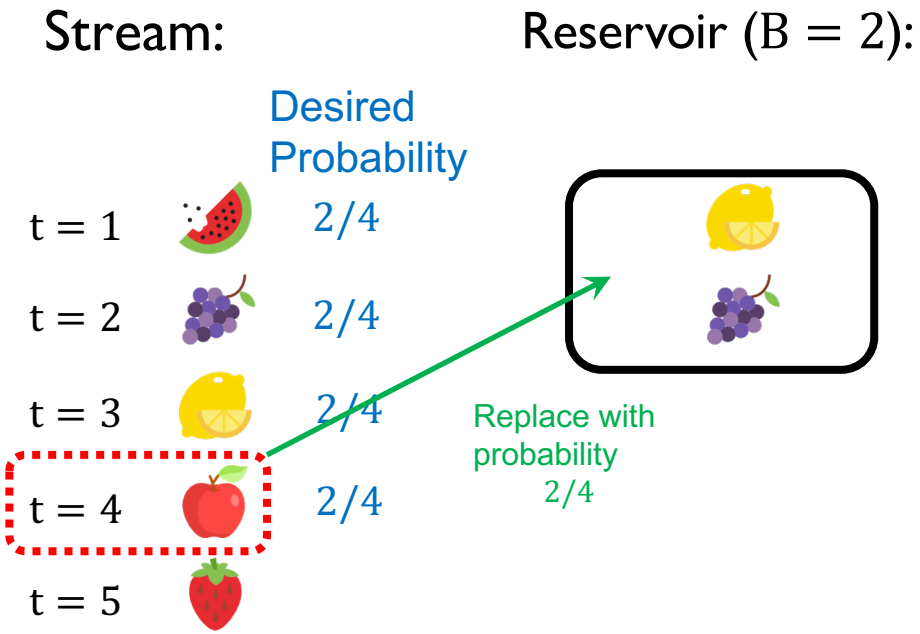


Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example

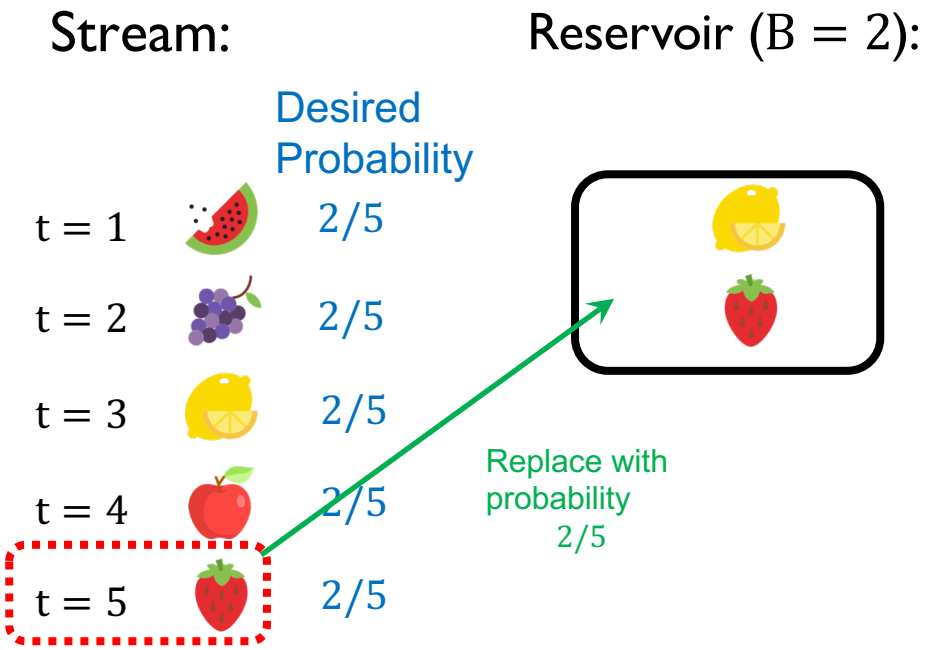


Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Reservoir Sampling: Example



Reservoir Sampling

Input: stream $\{a_1, a_2, \ldots\}$, reservoir size B

1. The first B items are inserted into the reservoir
2. When receiving a_t $(t > B)$:
   - with probability B/t, replace a random item in the reservoir with this item

Today's Plan

- Streams: Introduction and Data Model
- **Stream Algorithms**
  - Reservoir Sampling
  - **Running Mean and Variance**
- Stream Data Processing Systems: Storm

Computing Mean / Variance in a Stream

Optional

- **Goal:** Given a stream of values $\{x_1, x_2, ...\}$, can we keep track of the mean and variance of all the values that have appeared in the stream so far?
- This is also called "running mean / running variance", and is useful for standardizing data in a streaming manner, e.g. standardizing sensor data (by subtracting the mean and dividing by the standard deviation) in a streaming manner.
- This is useful but as it is mostly centred around a mathematical 'trick', it is marked as out of scope.

Running Mean

Optional

- (Sample) Mean is given by:
  $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i.$
- To compute this, we just need to keep track of the number of items so far n and the current sum $\sum_{i=1}^{n} x_i$, both of which can be easily updated over time.

Running Mean

Input: stream of real numbers $\{x_1, x_2, \ldots\}$

1. When receiving x_i:
   n += 1
   sum += x_i
2. When queried:
   return sum / n

Running Variance

Optional

- (Sample) Variance is given by $\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$.
- A useful alternate form is: $\frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}\right)$.
- This has 2 terms:
  - The first is the sum of squares, which can be computed in a streaming manner by incrementally adding up the squares
  - The second can be computed like in the previous slide

Running Variance

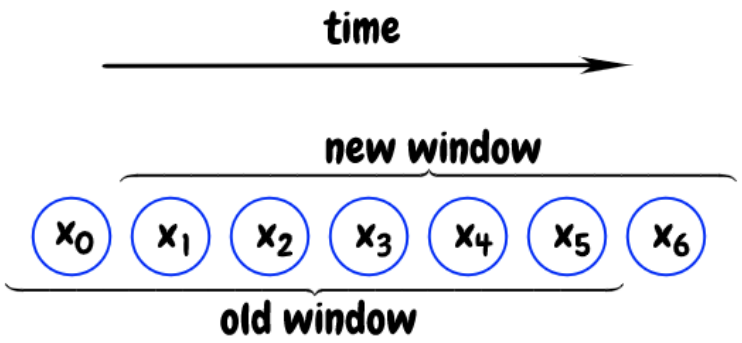
Input: stream of real numbers $\{x_1, x_2, \ldots\}$

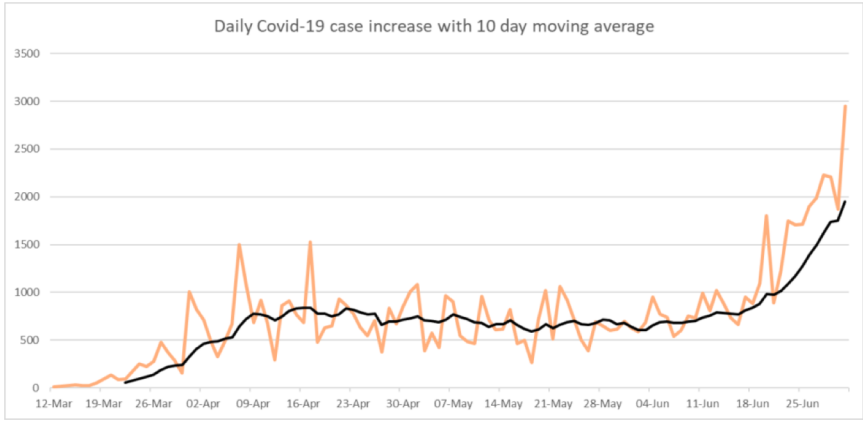
1. When receiving x_i:
   - n += 1
   - sum += x_i
   - sum_sq += x_i^2
2. When queried:
   - return 1/(n-1) * (sum_sq – (sum^2/n))

Extension: Running Mean / Variance in Sliding Window

Optional

- Consider a "**sliding window**" consisting of the 6 most recent items up to the current time. Each time we receive a new item (e.g. x_6), the window moves 1 step to the right.





- The approaches in the last 2 slides can be extended to compute mean / variance of the sliding window ("moving average"). When the window moves, we are adding an item (e.g. x_6) and removing an item (e.g. x_0).
- Thus we would update sum by adding x_6 and subtracting x_0, and sum_sq by adding x_6^2 and subtracting x_0^2.

Other Streaming Tasks and Algorithms

Optional

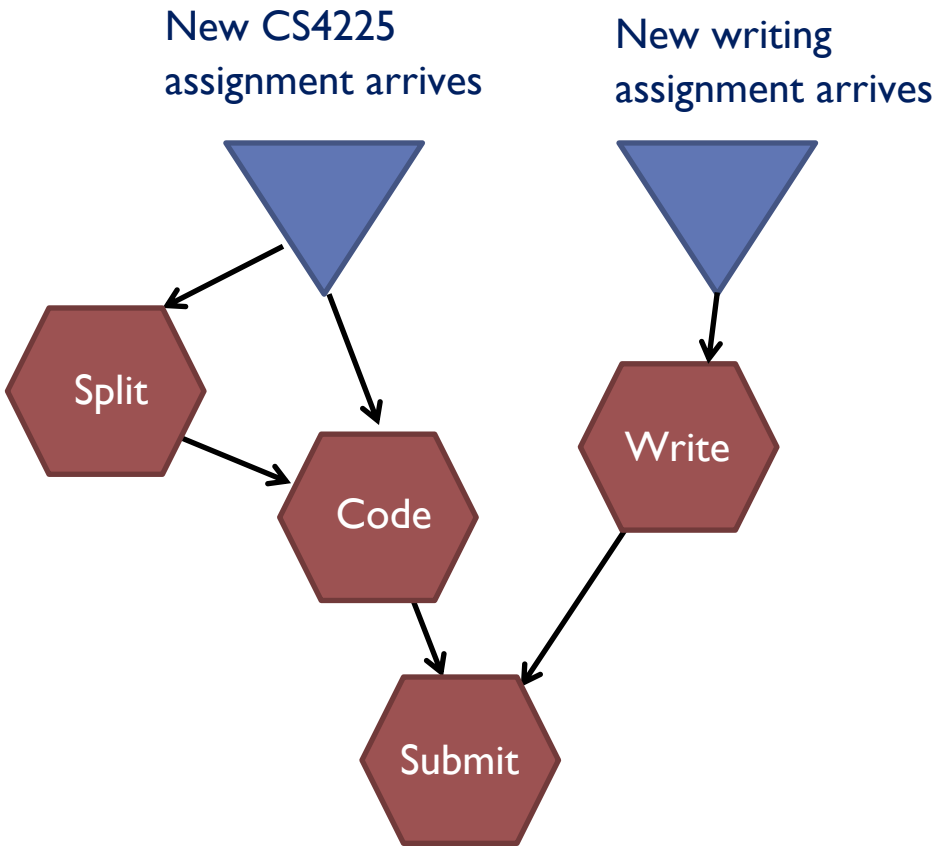
- Checking whether items have appeared in stream before
  - Bloom Filters
- Counting number of distinct elements
  - Flajolet-Martin / HyperLogLog
- Estimating how many times an element has appeared
  - Count Min Sketch
- …
- In general, answering "simple" questions like these is much more complex in the streaming setting (as we have limited memory but potentially infinite data)

Today's Plan

- Streams: Introduction and Data Model
- Stream Algorithms
  - Reservoir Sampling
  - Running Mean and Variance
- **Stream Data Processing Systems: Storm**

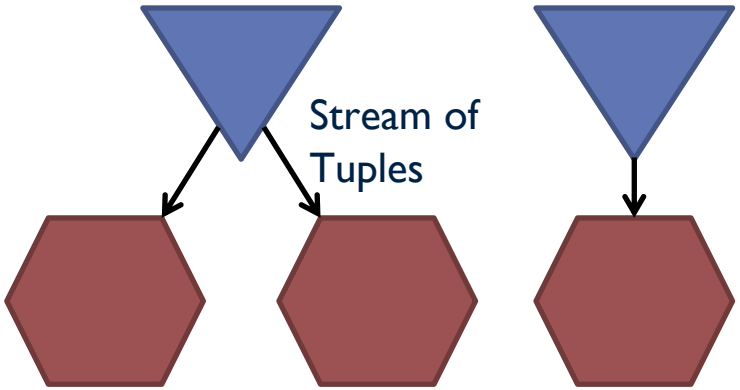
Stream Processing: Illustrative Example

- We can represent a student completing coursework as a series of operations in a graph structure:
  - Events (assignments) arrive at the top and "flow" through the graph, generating unknown numbers of further events at subsequent nodes



Storm: Basics

- **Tuple**: an ordered list of named values, like a database row
- To do computation in Storm, the user creates a **topology**, which is a computation graph
  - Nodes hold processing logic (i.e., transformation over its input)
  - Edges indicate communication between nodes
  - Each topology corresponds to a Storm **job**. Once started, a job continues running forever until killed.

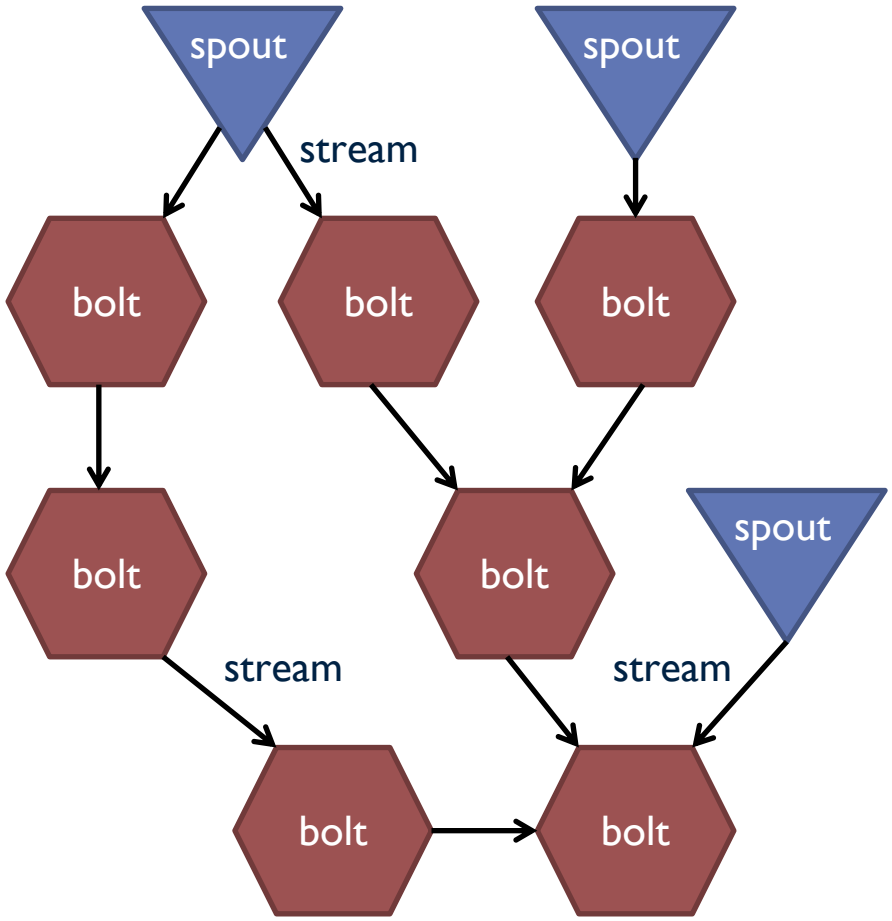


Streams, Spouts, and Bolts

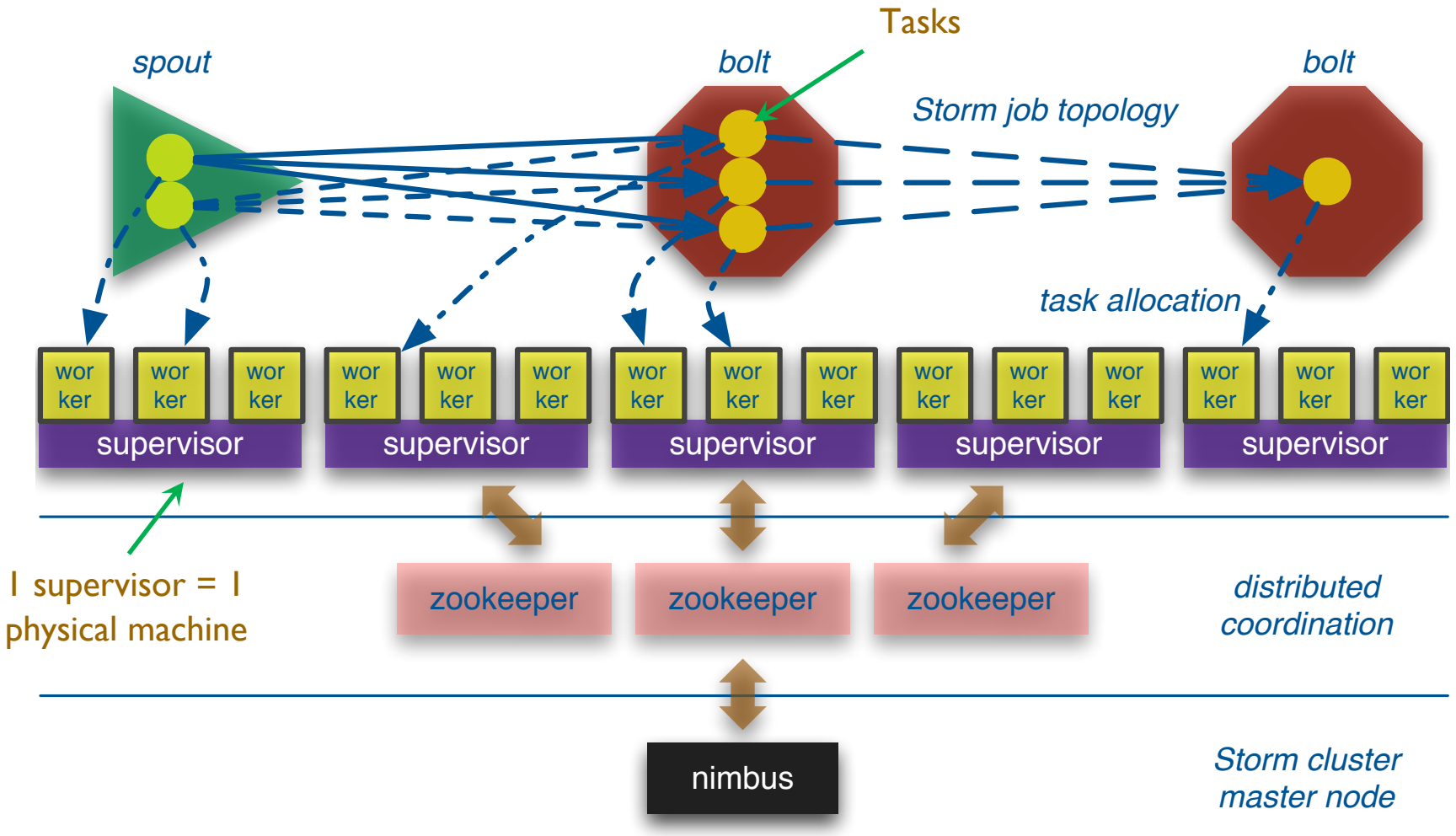
- Streams
  - The basic collection abstraction: an unbounded sequence of tuples
  - Streams are transformed by the processing elements of a topology

- Spouts
  - Stream generators
  - May propagate a single stream to multiple consumers

- Bolts
  - Subscribe to streams
  - Streams transformers
  - Process incoming streams and produce new ones

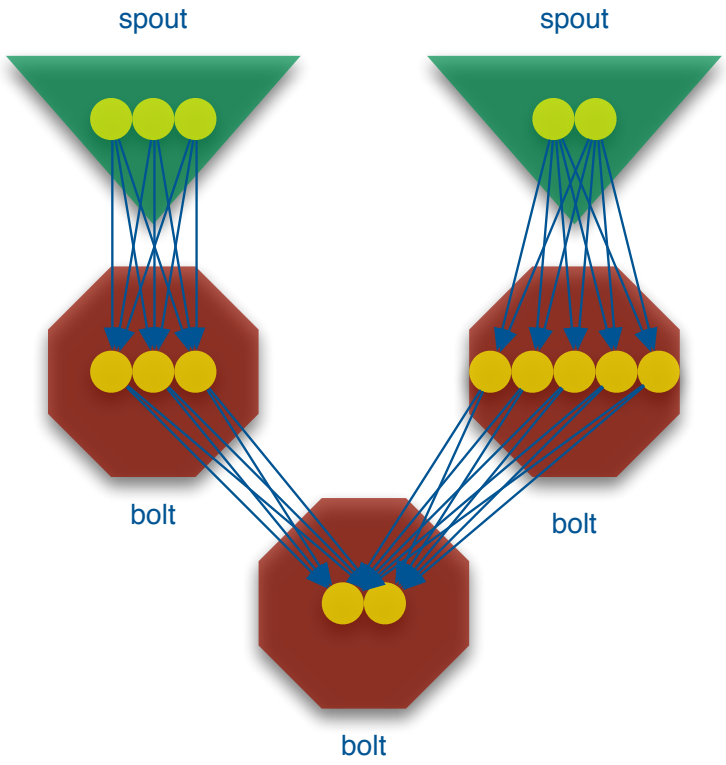


Storm Architecture



Stream Groupings

- Bolts are executed by multiple executors in parallel, called "tasks"
- When a bolt emits a tuple, which task should it go to in the next bolt?
- Stream groupings:
  - **Shuffle** grouping: tuples are randomly distributed across tasks
  - **Field** grouping: based on the value of a (user-specified) field
  - **All**: replicate tuple across all tasks of the target bolt



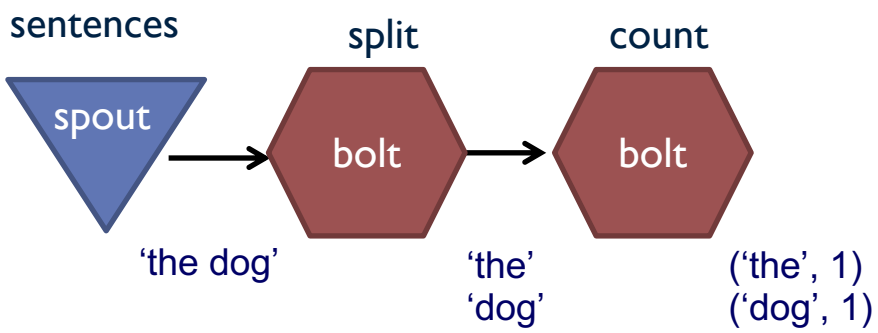
Storm Example: Constructing Topology

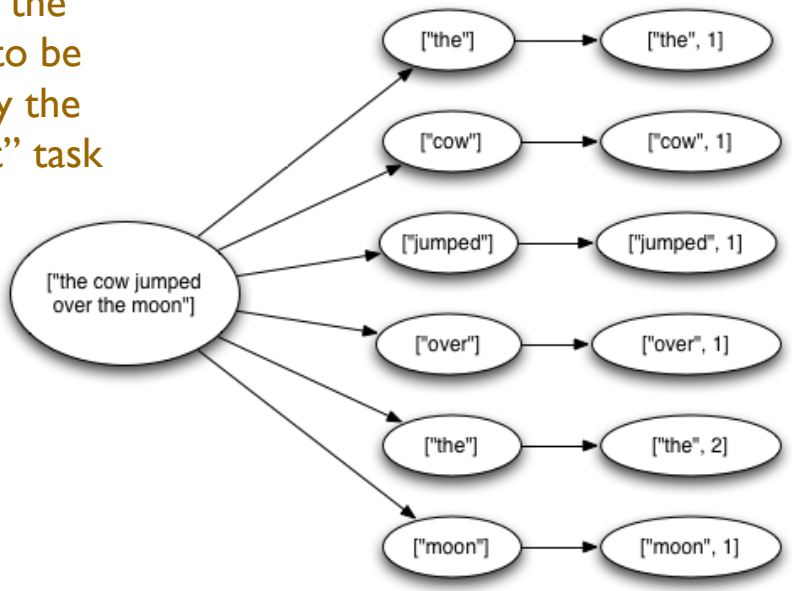
```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("sentences", new KestrelSpout("kestrel.backtype.com",
                                               22133,
                                               "sentence_queue",
                                               new StringScheme()));
builder.setBolt("split", new SplitSentence(), 10)
        .shuffleGrouping("sentences");
builder.setBolt("count", new WordCount(), 20)
        .fieldsGrouping("split", new Fields("word"));
```

Parallelism hint

Subscribe to "sentences" spout; *shuffle grouping* allows tuples to be randomly distributed to tasks

Fields grouping: ensures that the same word to be processed by the same "count" task



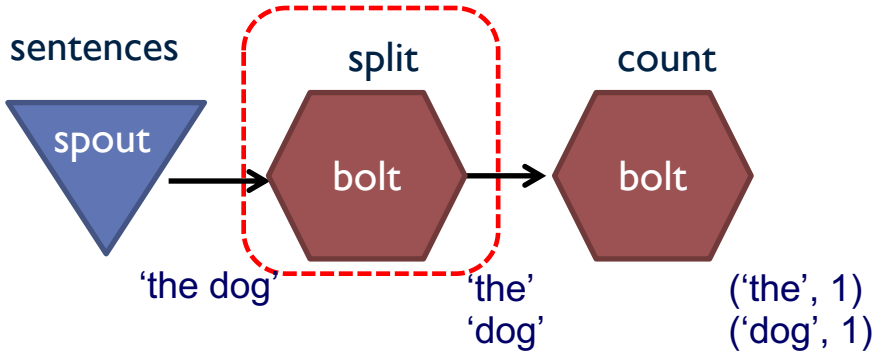


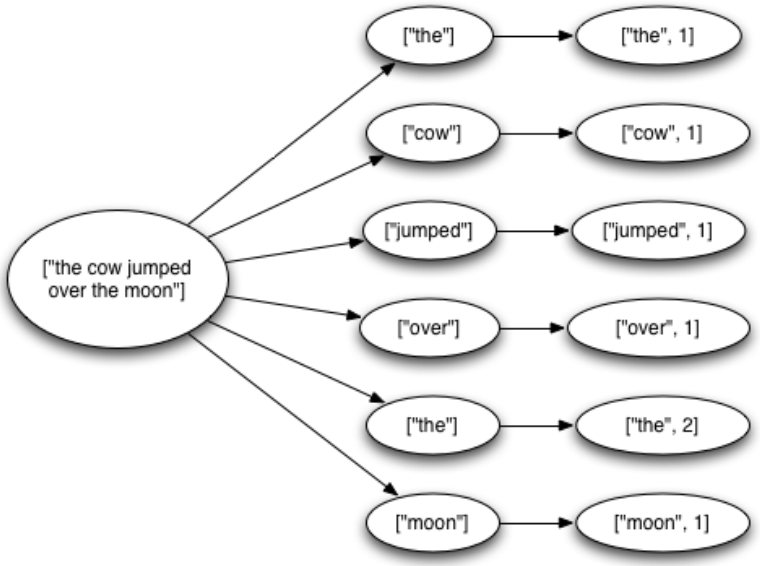
Storm Example: Bolt ("Split")

```
public class SplitSentence extends BaseBasicBolt {
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String sentence = tuple.getString(0);
        for(String word: sentence.split(" ")) {
            collector.emit(new Values(word));
        }
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

For each tuple, split it by space, and emit the resulting tuples





Storm Example: Bolt ("Count")

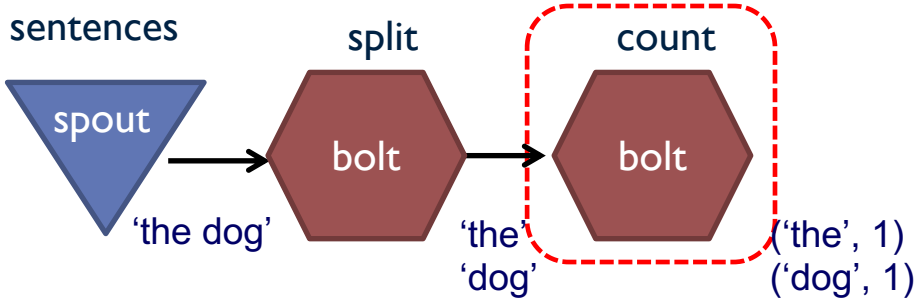
```
public static class WordCount extends BaseBasicBolt {
   Map<String, Integer> counts = new HashMap<String, Integer>();

   @Override
   public void execute(Tuple tuple, BasicOutputCollector collector) {
     String word = tuple.getString(0);
     Integer count = counts.get(word);
     if (count == null)
       count = 0;
     count++;
     counts.put(word, count);
     collector.emit(new Values(word, count));
   }

   @Override
   public void declareOutputFields(OutputFieldsDeclarer declarer) {
     declarer.declare(new Fields("word", "count"));
   }
 }
```

Count the arriving tuples using a Map

Emit the associated count tuple



Acknowledgements

- Julian M. Kunkel - Stream Processing (with Storm, Spark, Flink)
- https://storm.apache.org/index.html
- mmds.org: Mining of Massive Datasets - Jure Leskovec, Anand Rajaraman, Jeff Ullman