10-605
ML from Large Datasets

Announcements

- HW1b is going out today
- You should now be
  - on autolab
  - have a an account on 'stoat'
    - a locally-administered Hadoop cluster
  - shortly receive a coupon for Amazon Web Services (AWS) for $50
  - problems to 10605-instructors@cs but please check the class web site first

Announcements

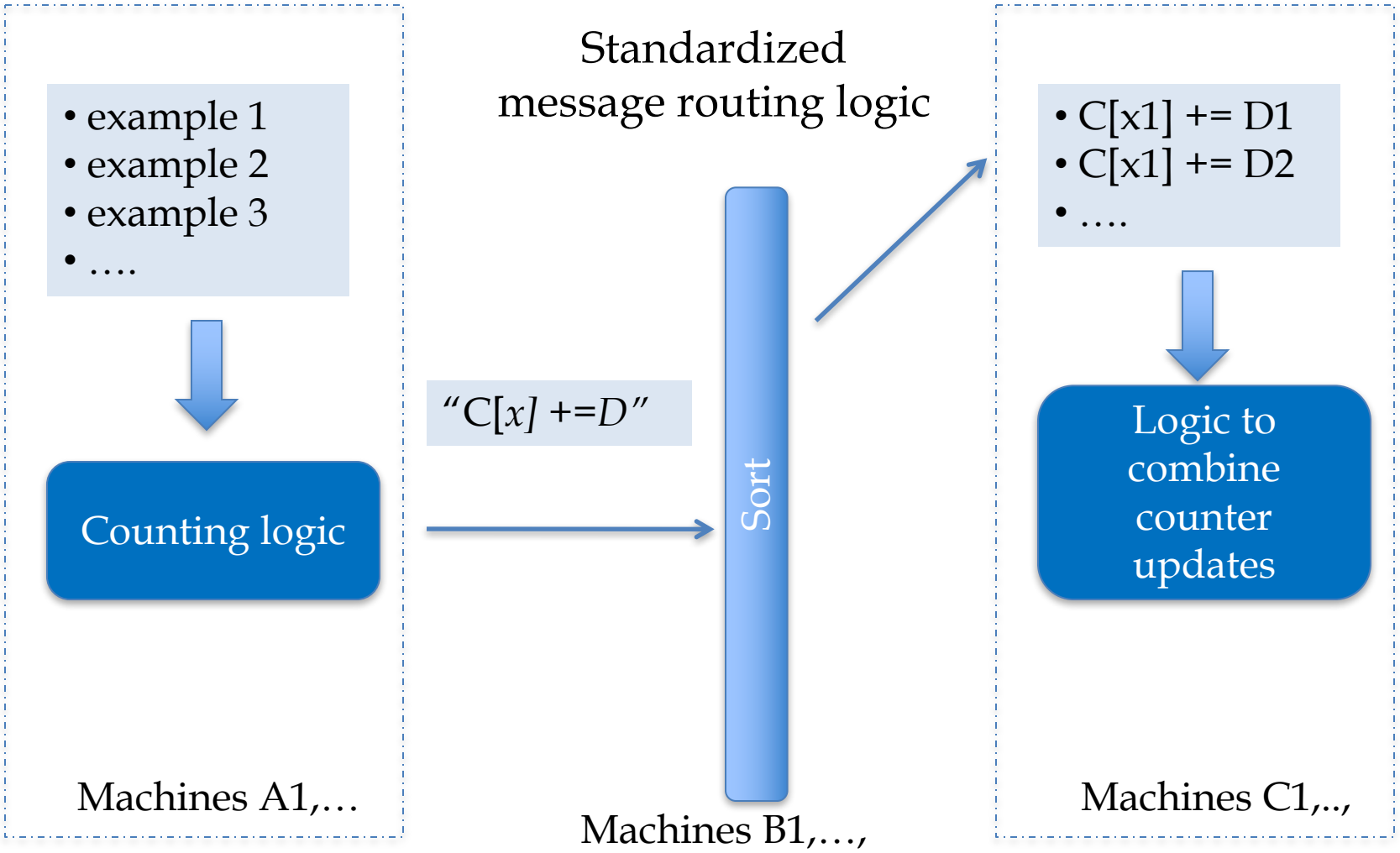
- About AWS:
  - good: Class deadlines are not a problem for it
  - bad: It takes a little while to provision a cluster and start it up (10min?)
  - Basically it's rent-a-cluster *by the hour*

  - Don't leave the meter running
  - Don't leave your credentials unprotected
  - You are responsible for the $$$ if something goes wrong!

  - Sign up for AWS Educate
    - So they know you are a student

PARALLELIZING STREAM AND SORT

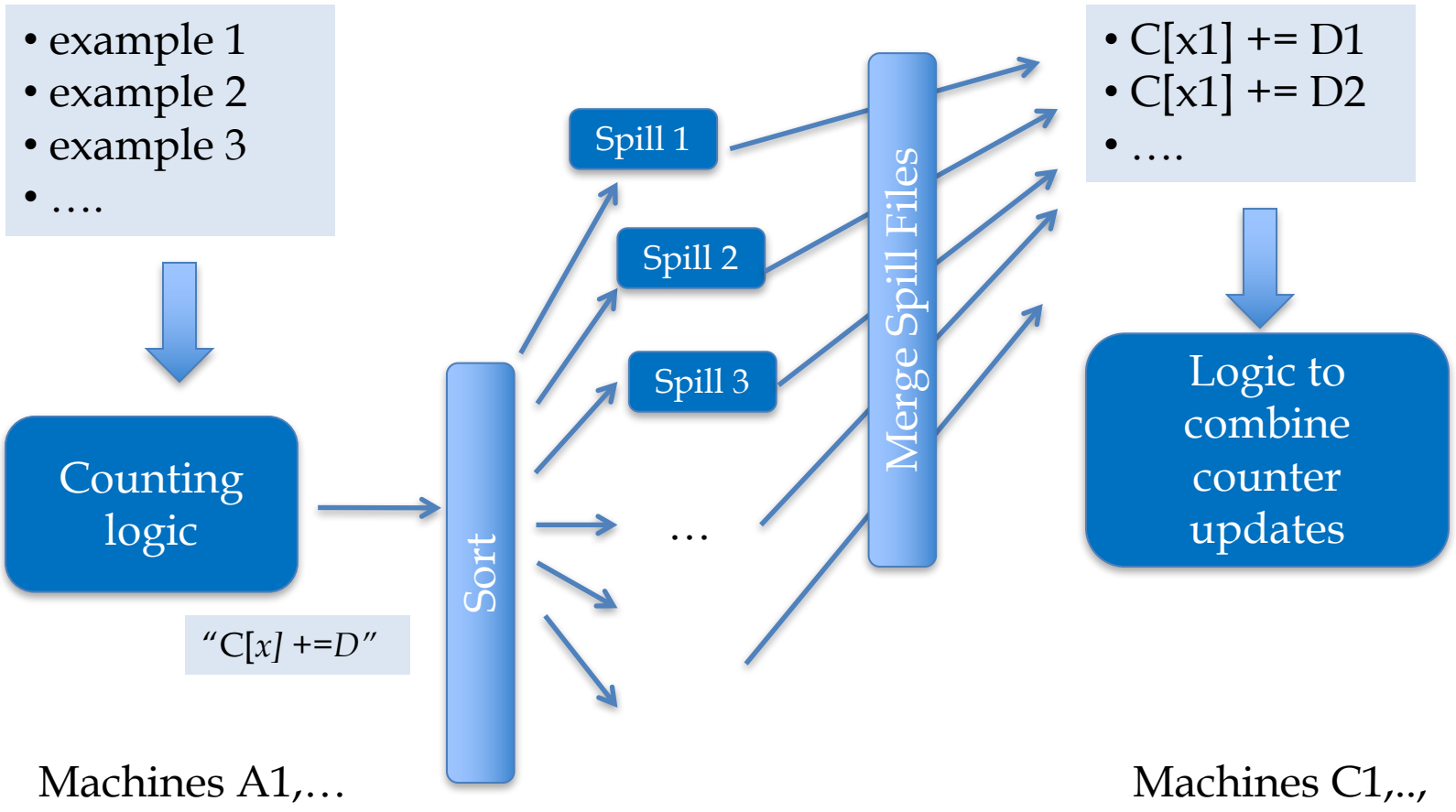
Stream and Sort Counting → Distributed Counting



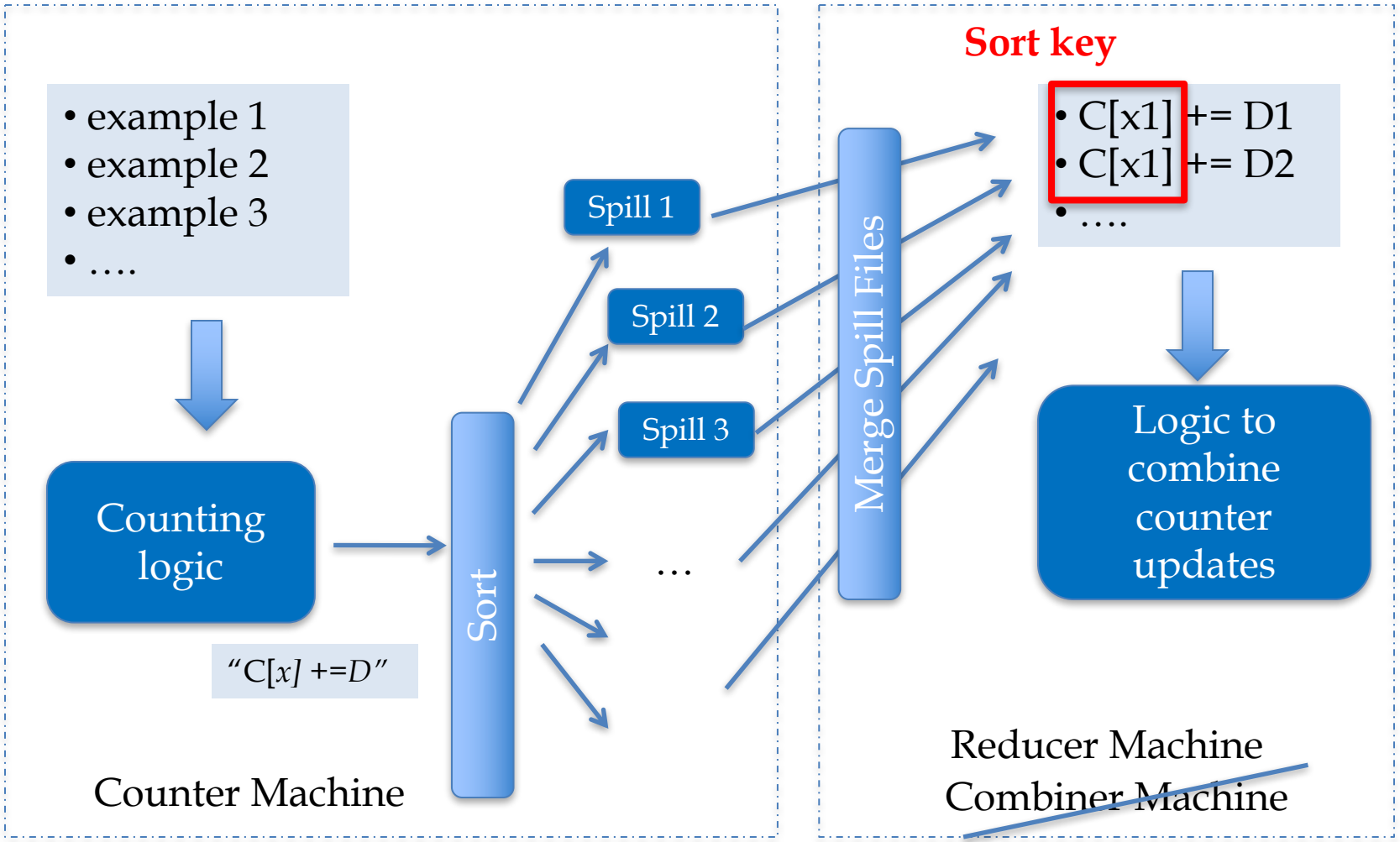
Trivial to parallelize!

Easy to parallelize!

Stream and Sort Counting → Distributed Counting

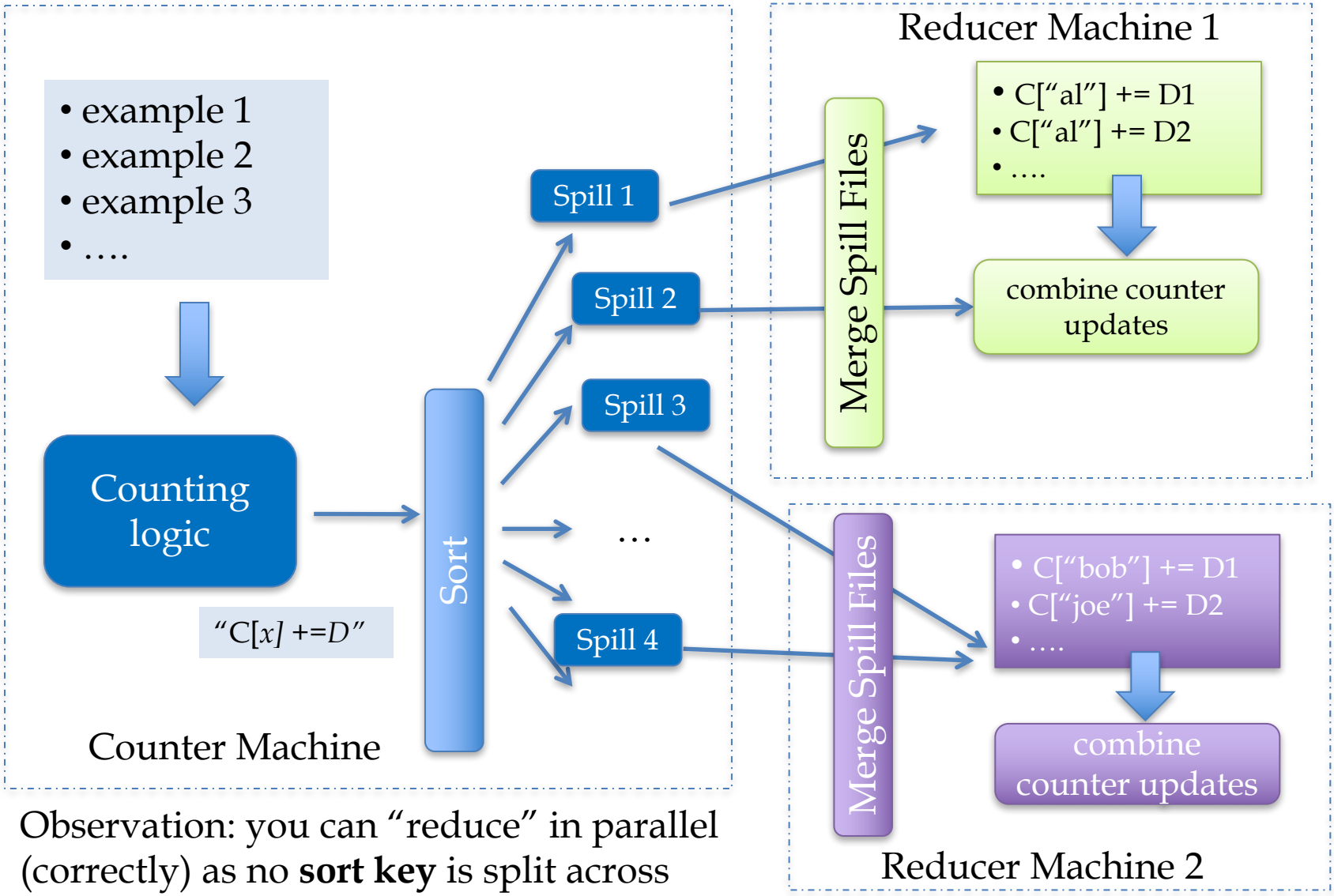


Stream and Sort Counting → Distributed Counting



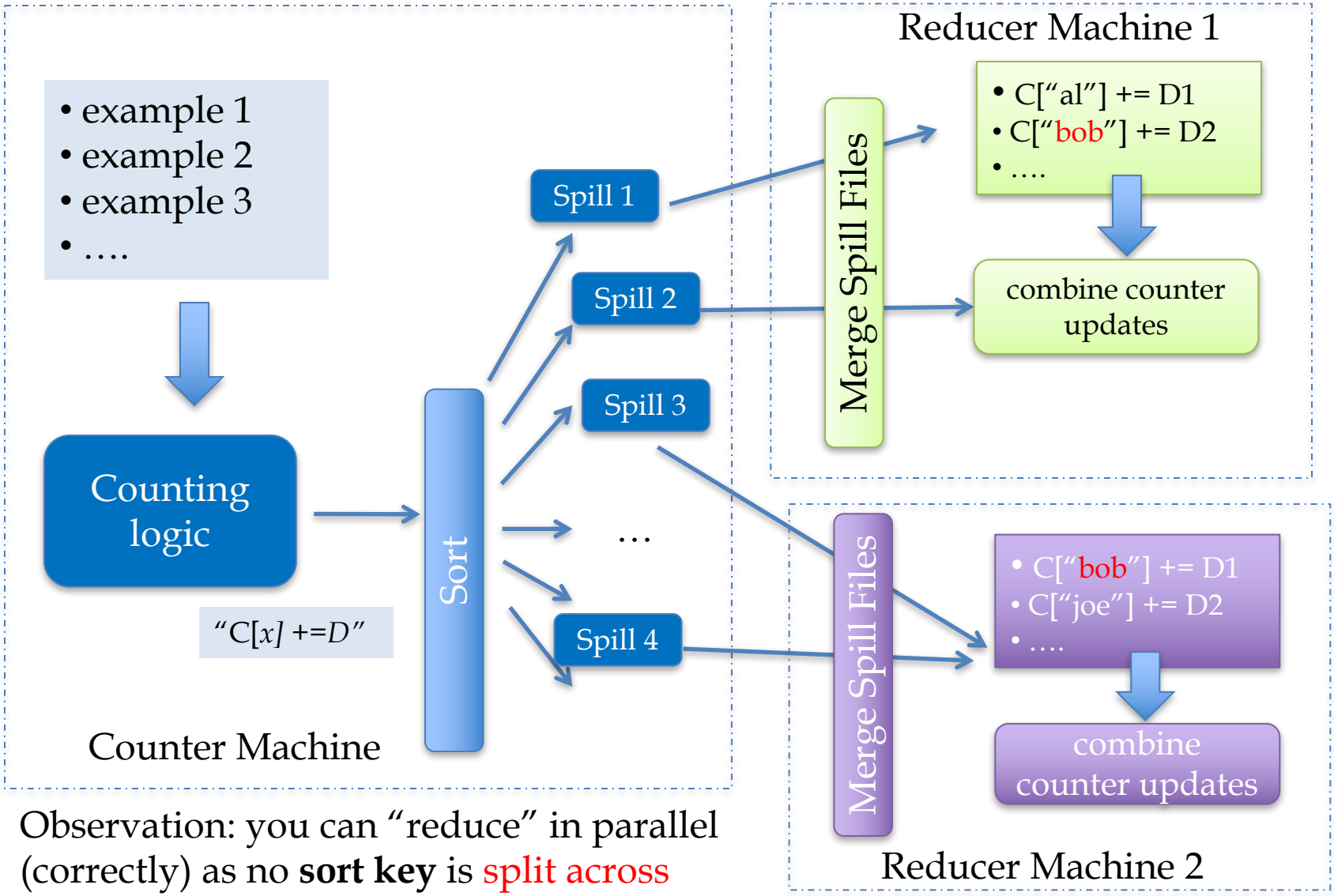
Observation: you can "reduce" in parallel (correctly) as no **sort key** is split across multiple machines

Stream and Sort Counting → Distributed Counting



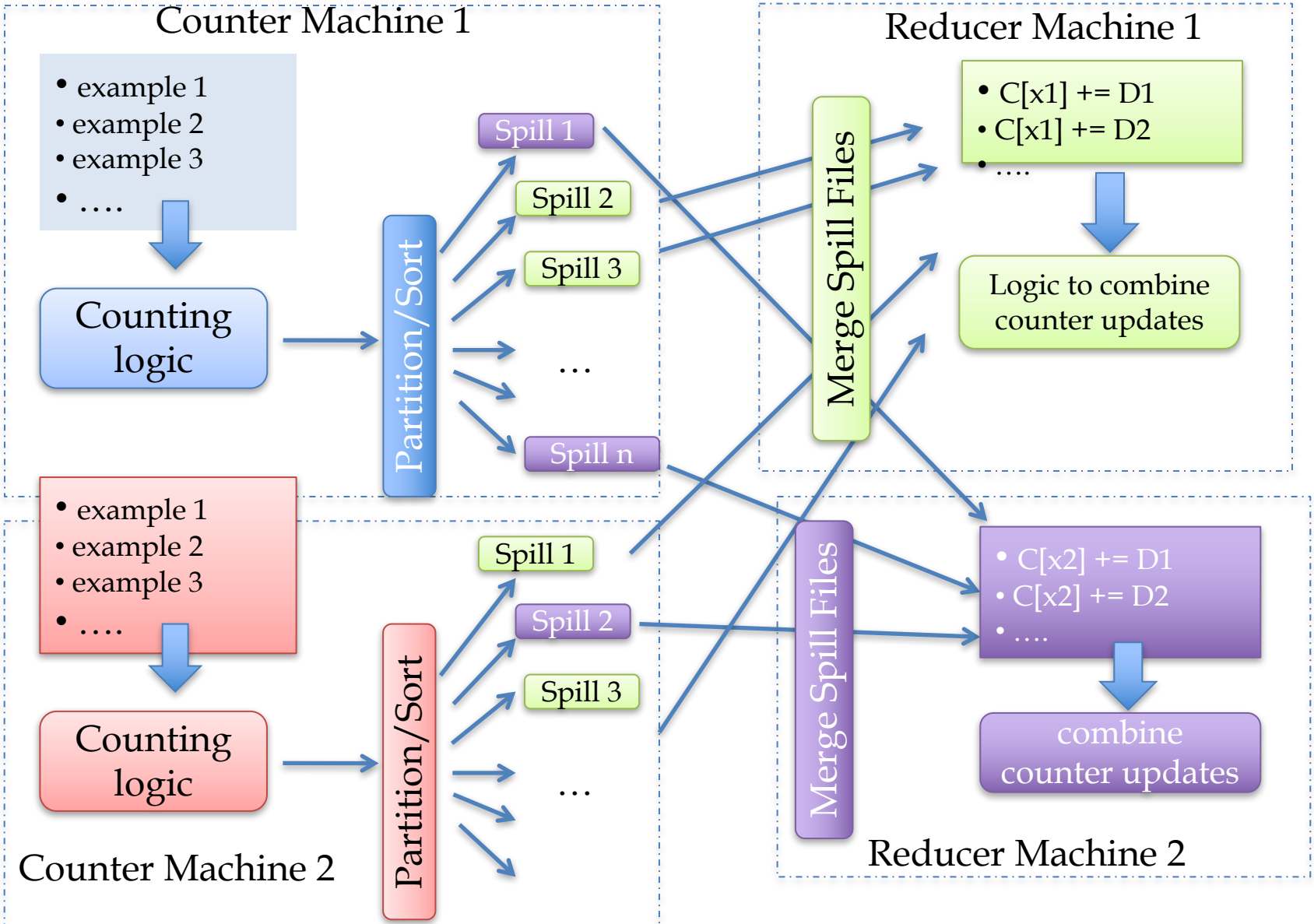
Observation: you can "reduce" in parallel (correctly) as no **sort key** is split across multiple machines

Stream and Sort Counting → Distributed Counting

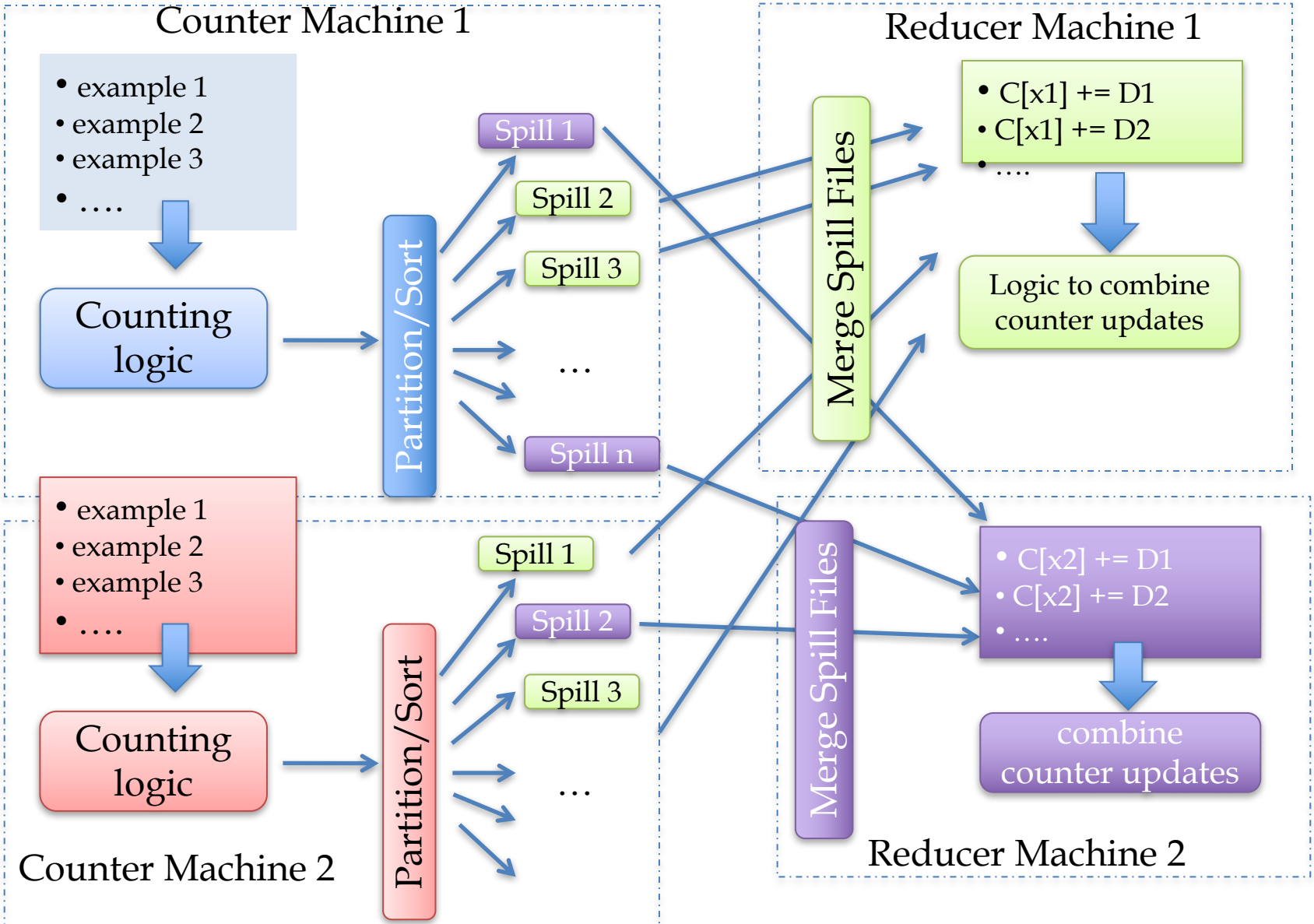


Observation: you can "reduce" in parallel (correctly) as no **sort key** is split across multiple machines

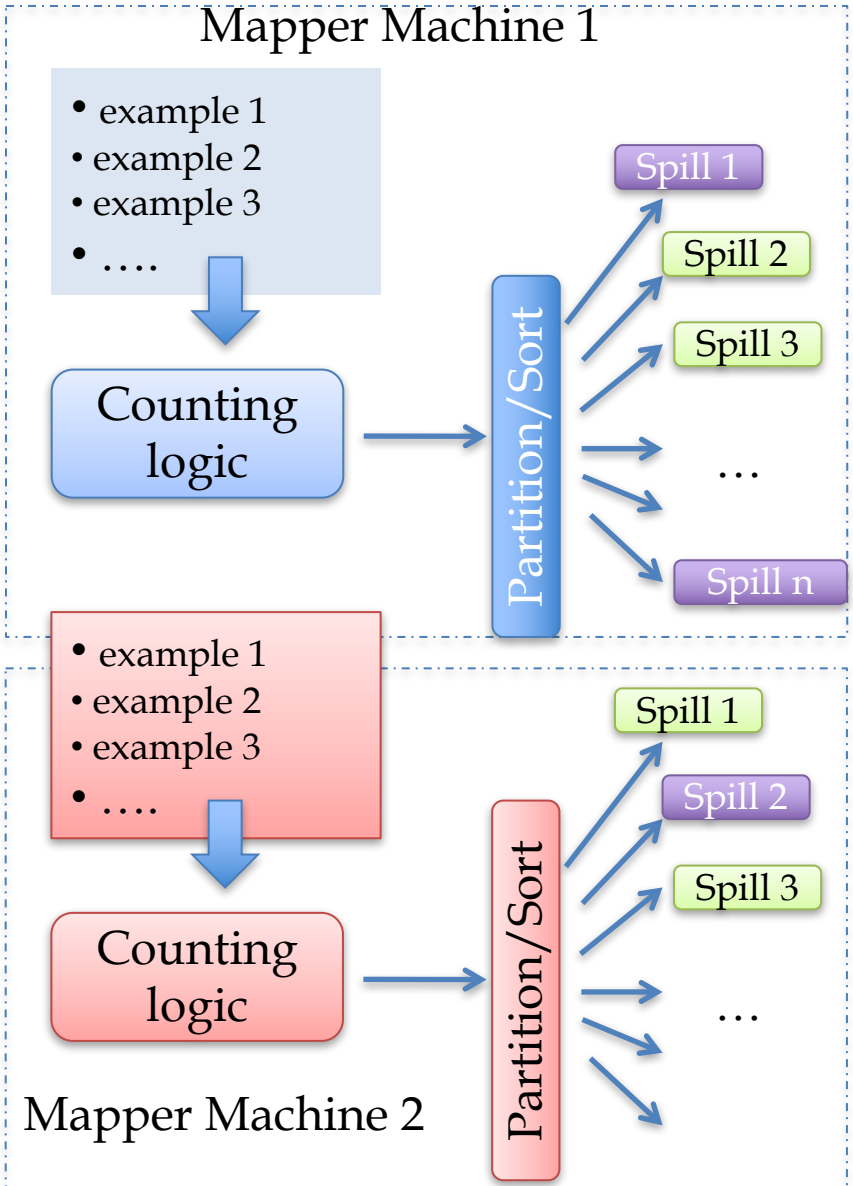
Same holds for counter machines: you can count in parallel as no **sort key** is split across multiple reducer machines



Stream and Sort Counting → Distributed Counting



Stream and Sort Counting → Distributed Counting



Mapper/counter machines run the "Map phase" of map-reduce

- Input different subsets of the total inputs
- Output (sort key,value) pairs

- The (key,value) pairs are **partitioned based on the key**
- Pairs from each partition will be sent to a different reducer machine.

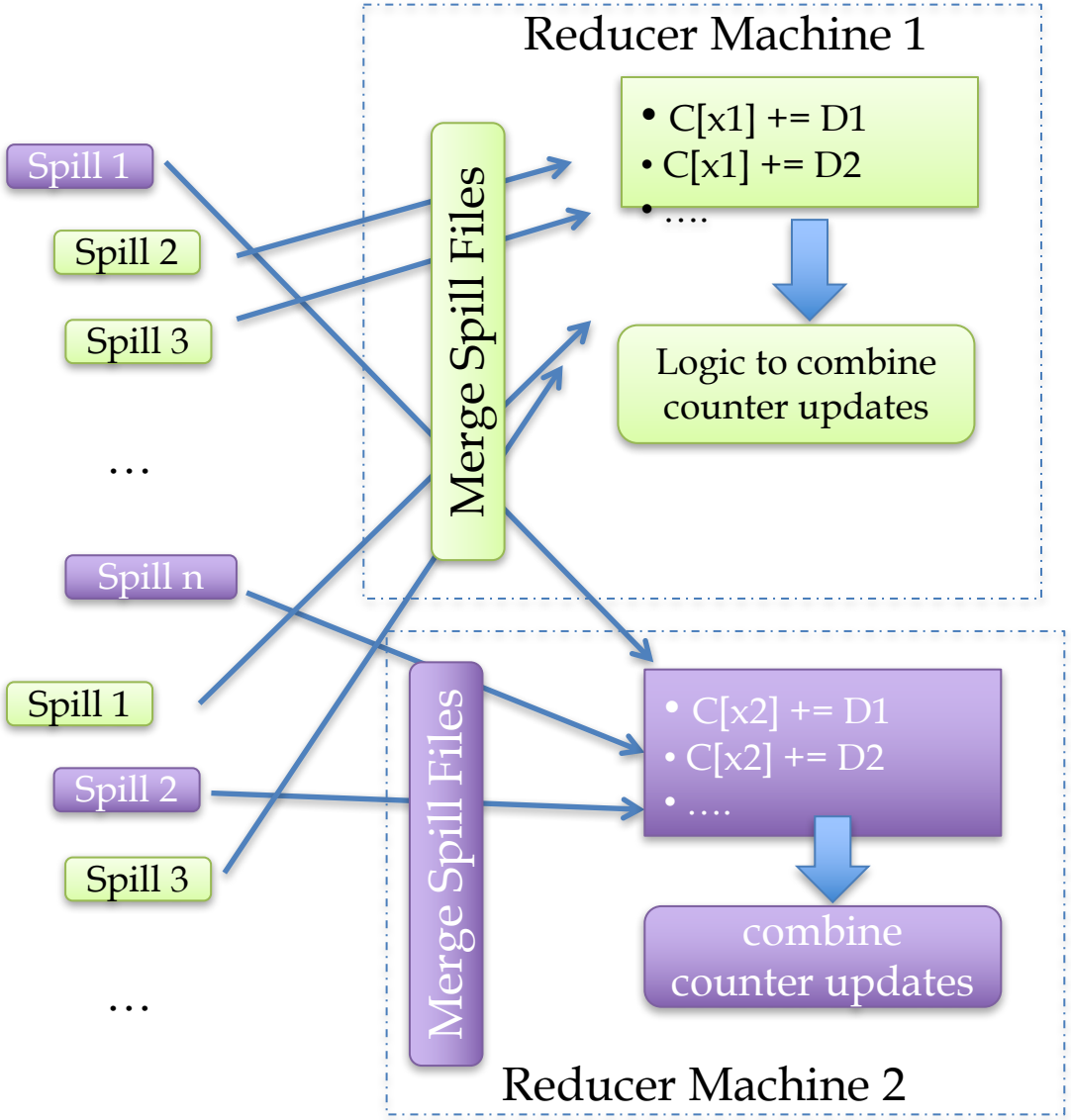
Stream and Sort Counting → Distributed Counting

The shuffle/sort phrase:

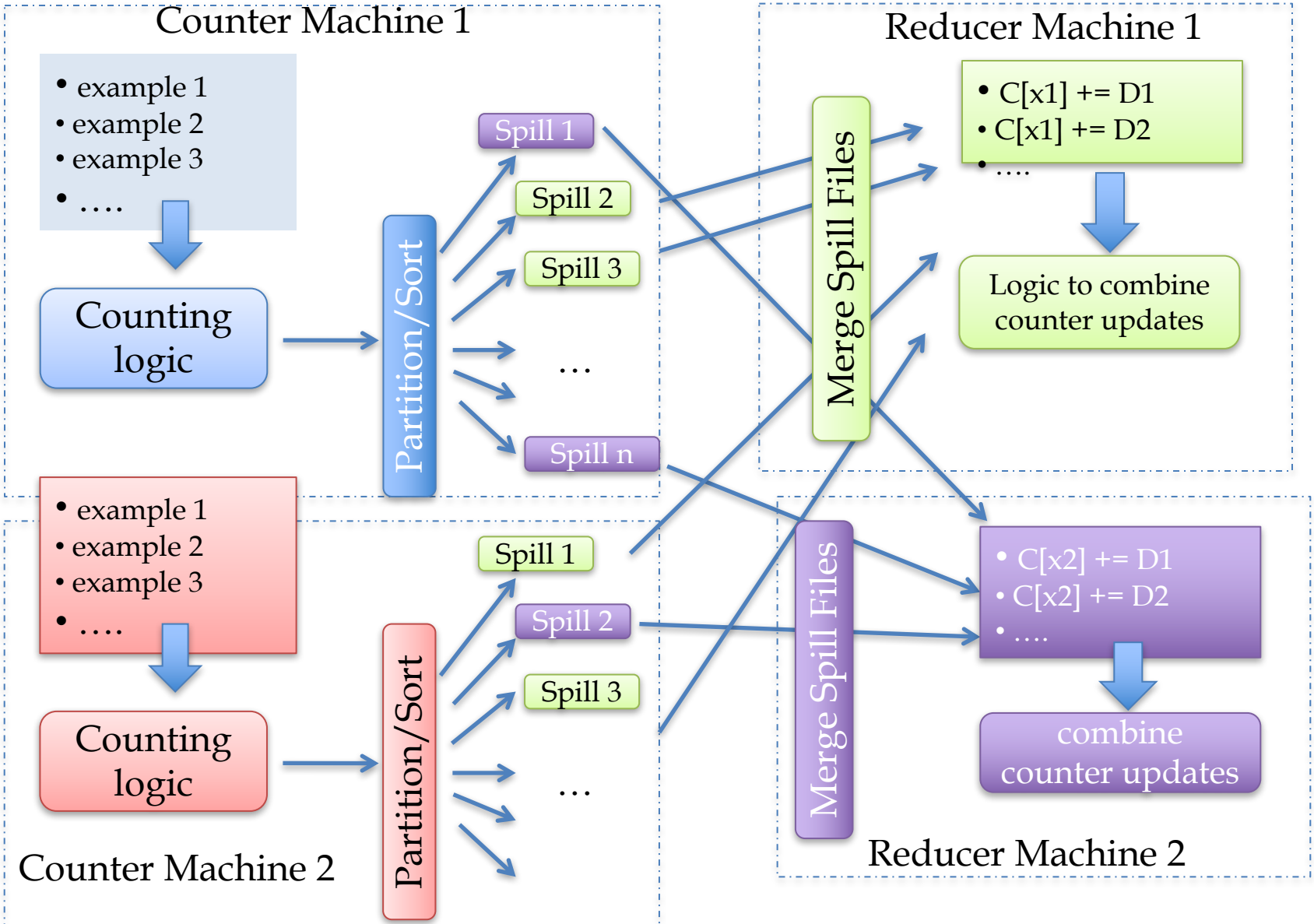
- (key,value) pairs from each **partition** are sent to the right reducer
- The reducer will **sort** the pairs together to get **all the pairs with the same key together.**

The reduce phase:

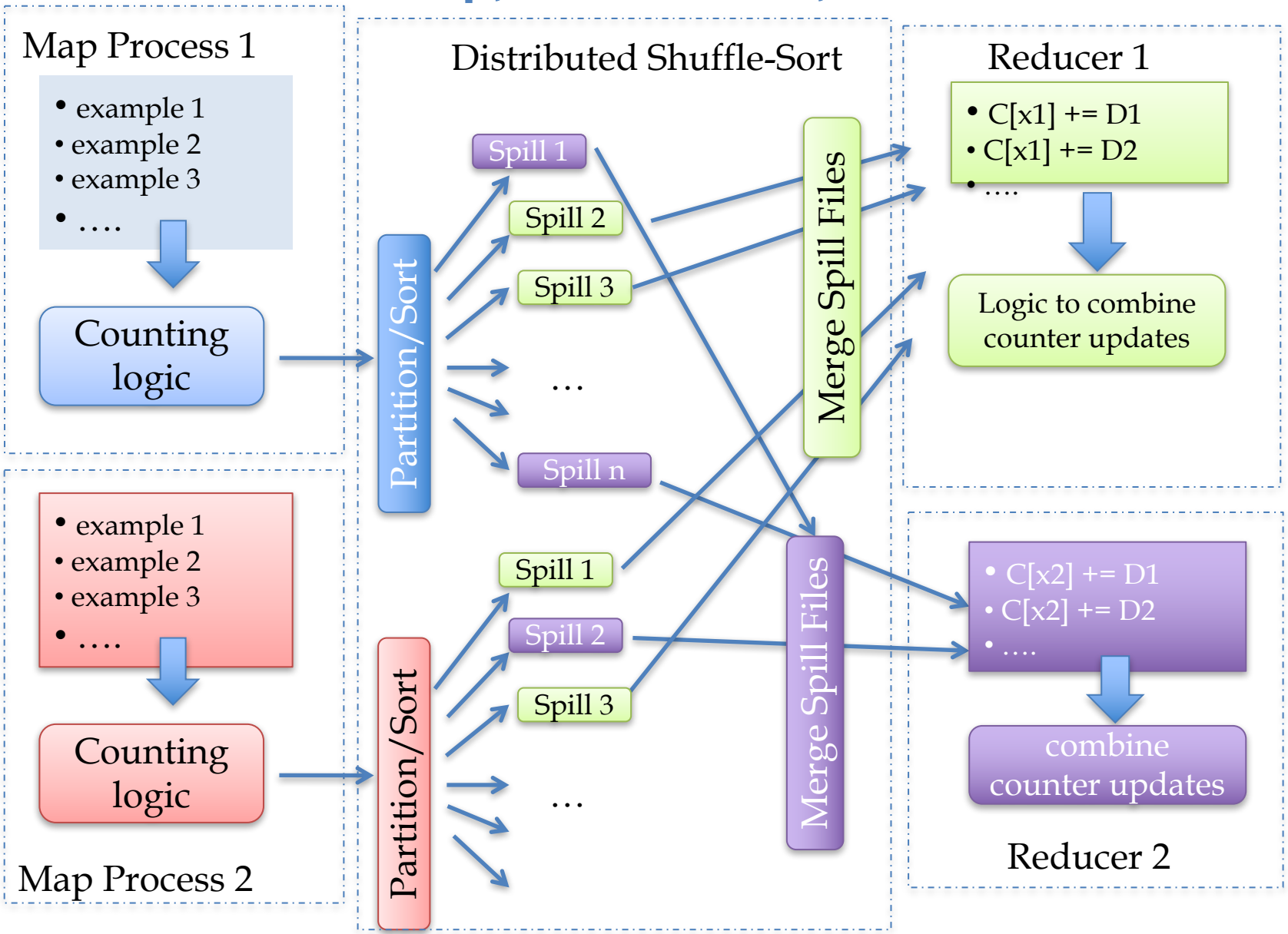
- Each reducer will scan through the sorted key-value pairs.



Stream and Sort Counting → Distributed Counting

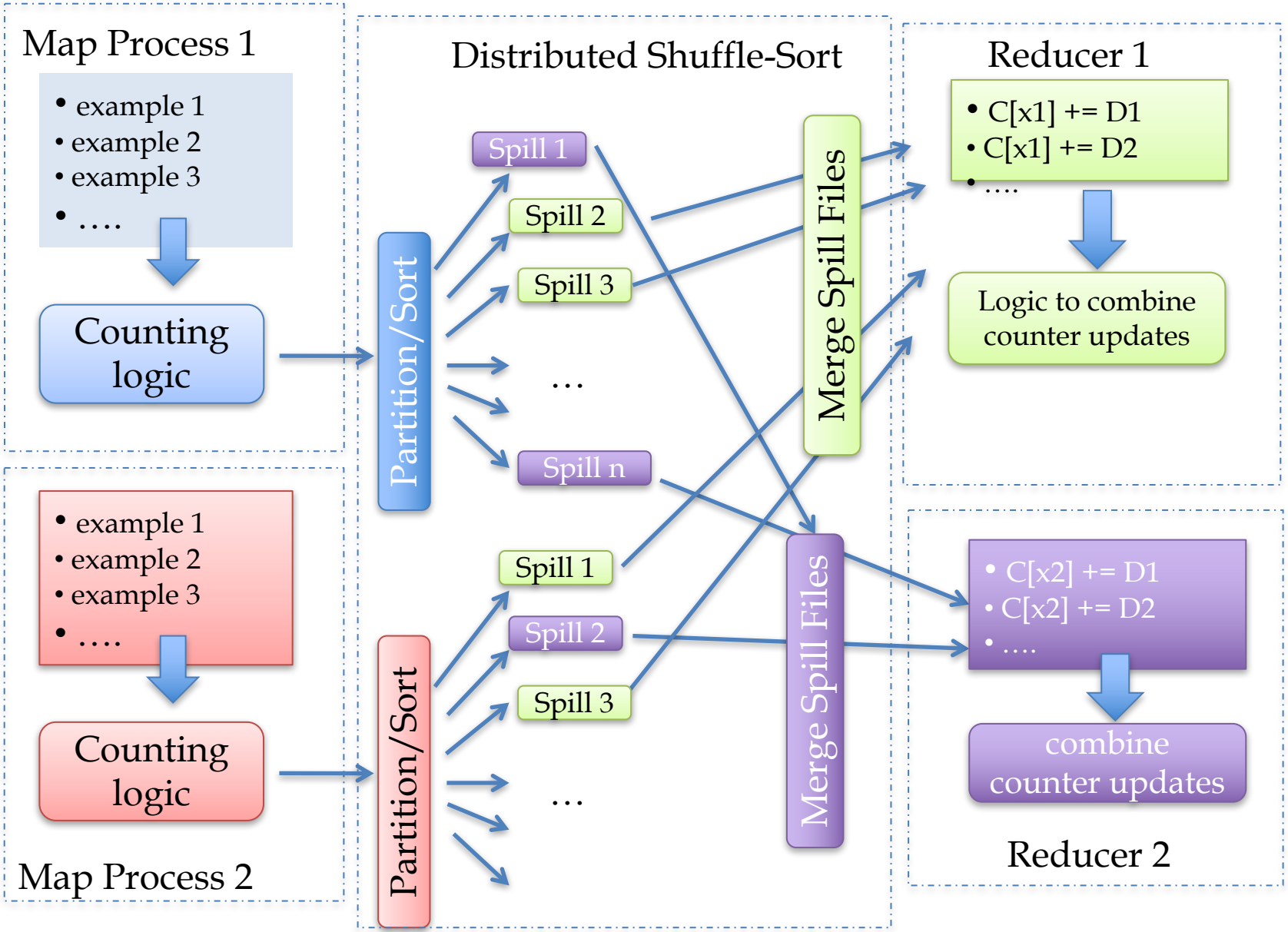


Distributed Stream-and-Sort: Map, Shuffle-Sort, Reduce



HADOOP AS PARALLEL STREAM-AND-SORT

Hadoop: Distributed Stream-and-Sort

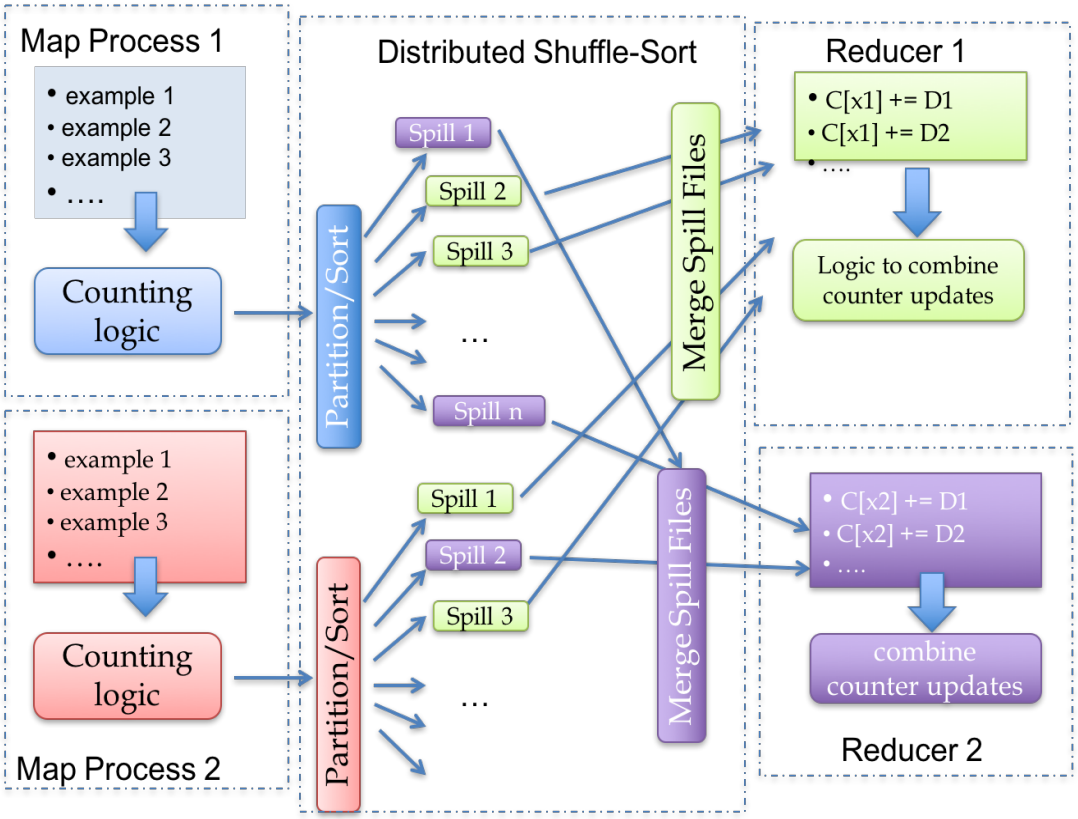


Hadoop: Distributed Stream-and-Sort

Local implementation:

cat input.txt | MAP | sort | REDUCE > output.txt

1. Split the input across N mapper machines: input1.txt, ….
2. In parallel, run N mappers mout.txt, mout2.txt, …
3. In parallel, partition mouti.txt for the M mapper machines: part1.1.txt, part1.2..txt, … partN.M.txt
4. In parallel, send each partition to the right reducer

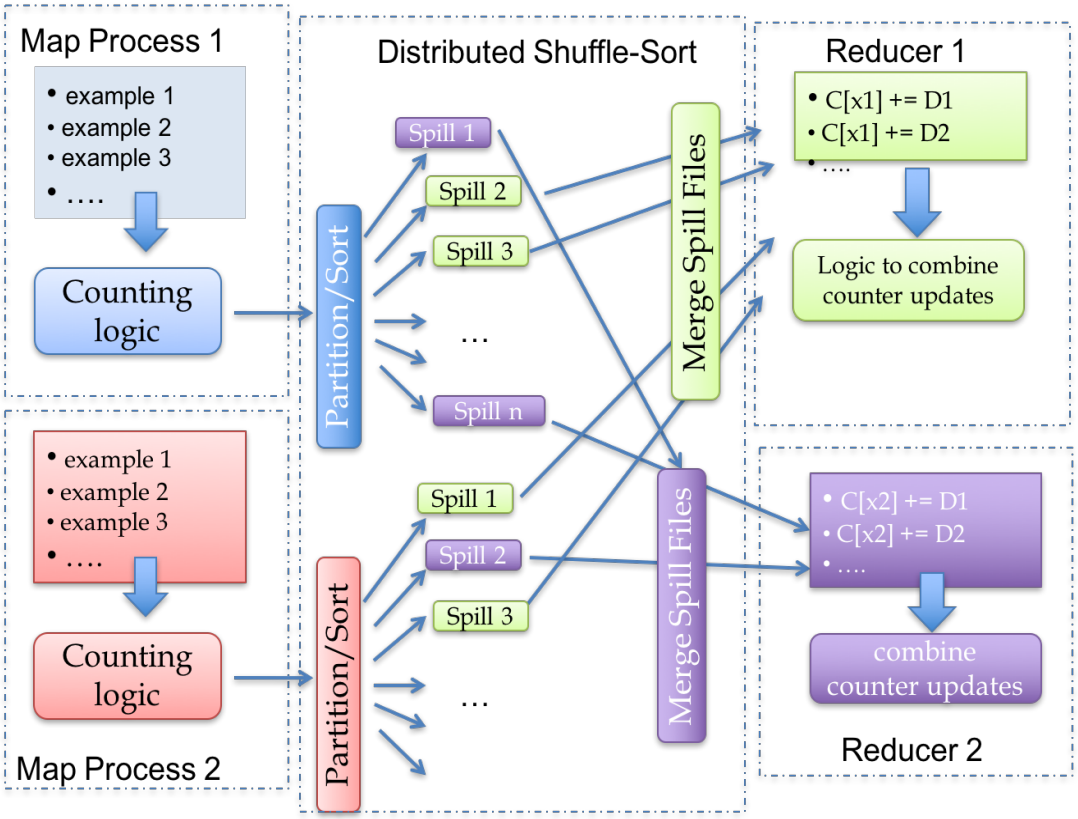


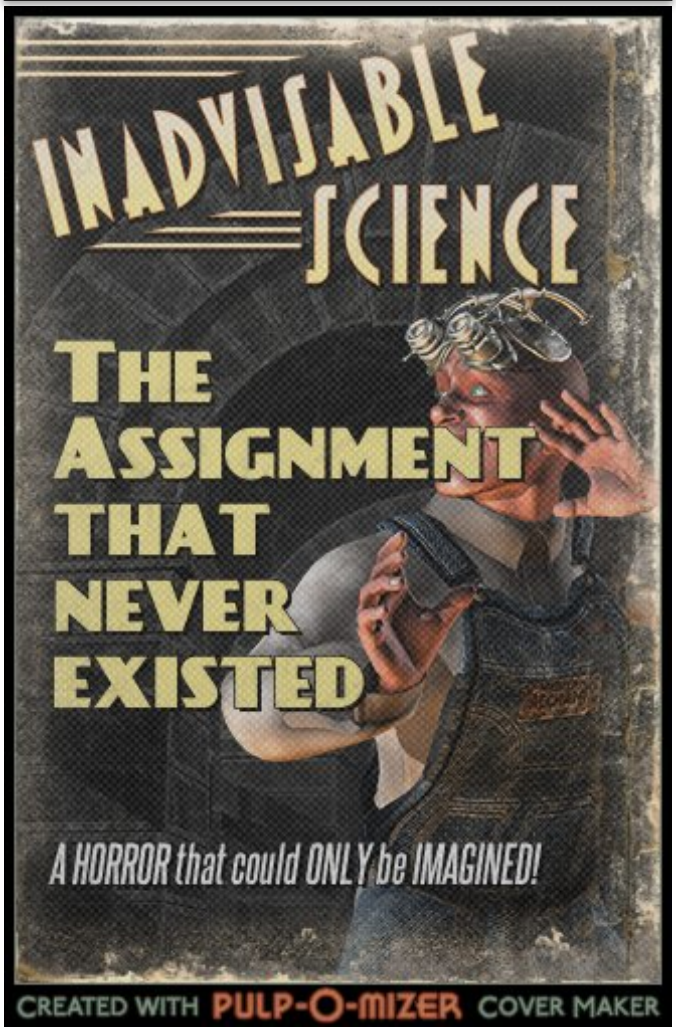
Hadoop: Distributed Stream-and-Sort

Local implementation:

cat input.txt | MAP | sort | REDUCE > output.txt

How could you parallelize this?





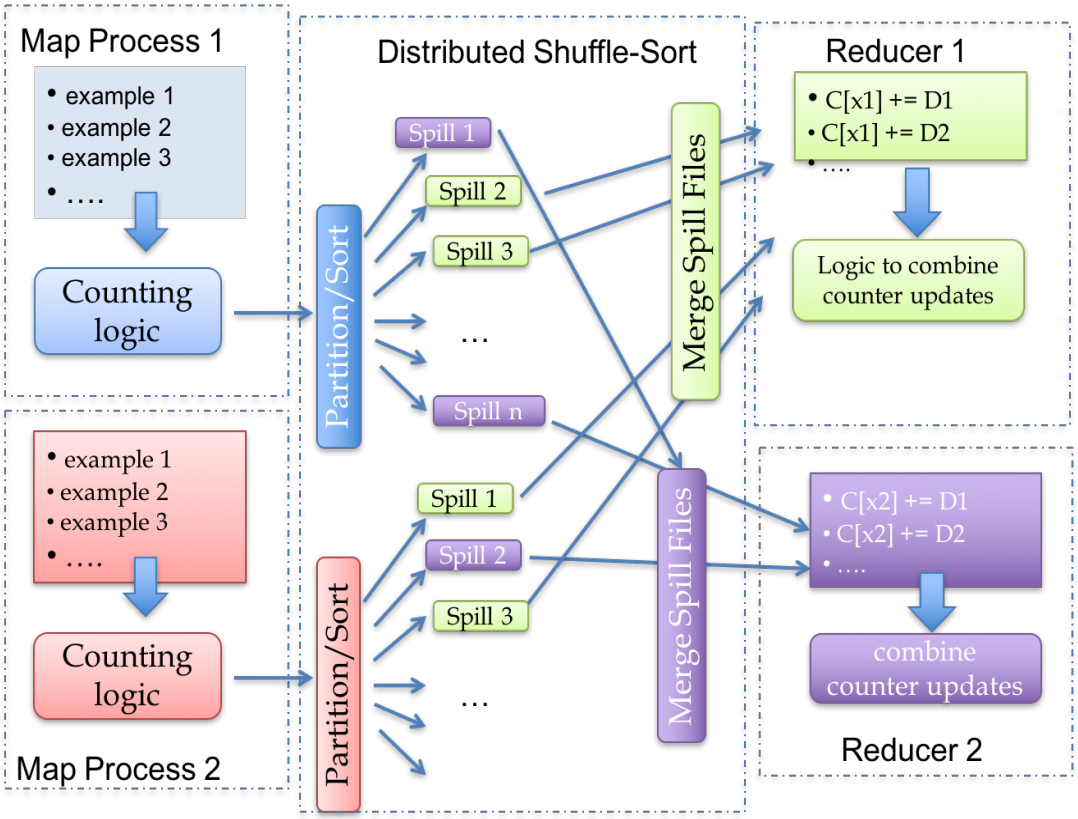
Hadoop: Distributed Stream-and-Sort

Local implementation:

cat input.txt | MAP | sort | REDUCE > output.txt

In parallel

1. Run N mappers mout.txt, mout2.txt, …
2. Partition mouti.txt for the M mapper machines: part1.1.txt, part1.2..txt, … partN.M.txt
3. Send each partition to the right reducer

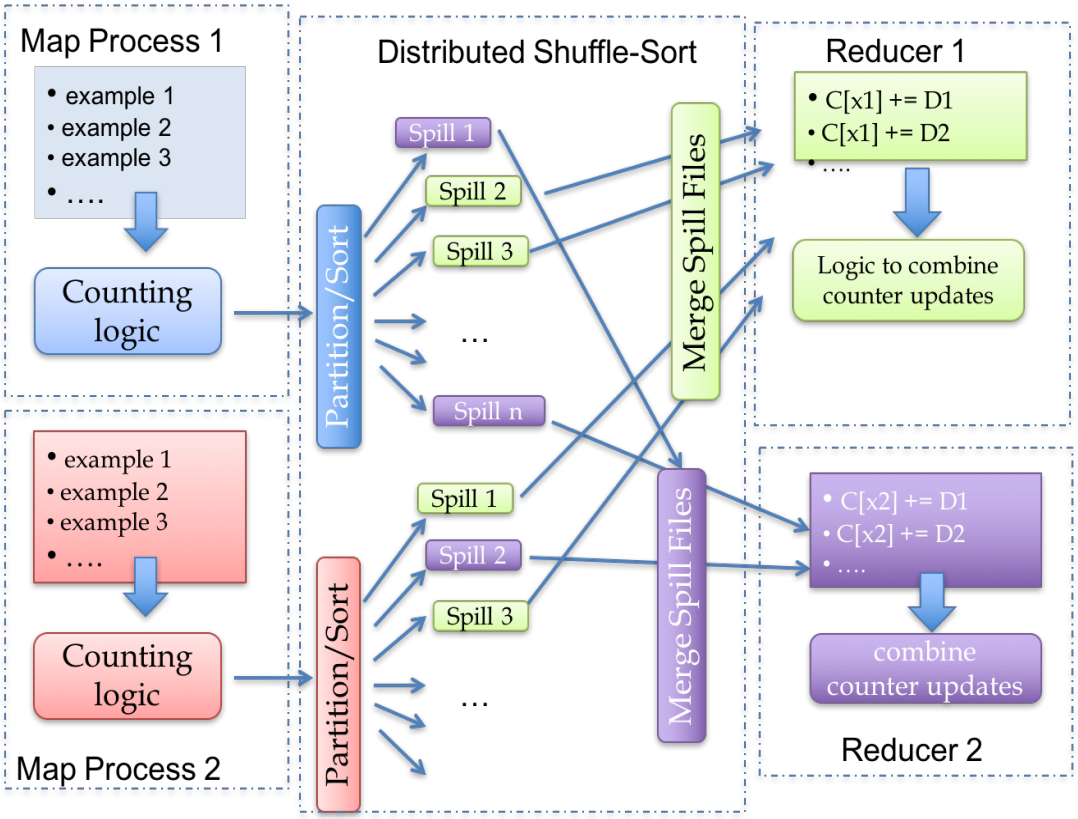


Hadoop: Distributed Stream-and-Sort

Local implementation:

cat input.txt | MAP | sort | REDUCE > output.txt

1. Map
2. Partition
3. Send



In parallel:

1. Accept N partition files, part1.j.txt, … partN.j.txt
2. Sort/merge them together to rinj.txt
3. Reduce to get the final result (reduce output for each key in partition j)

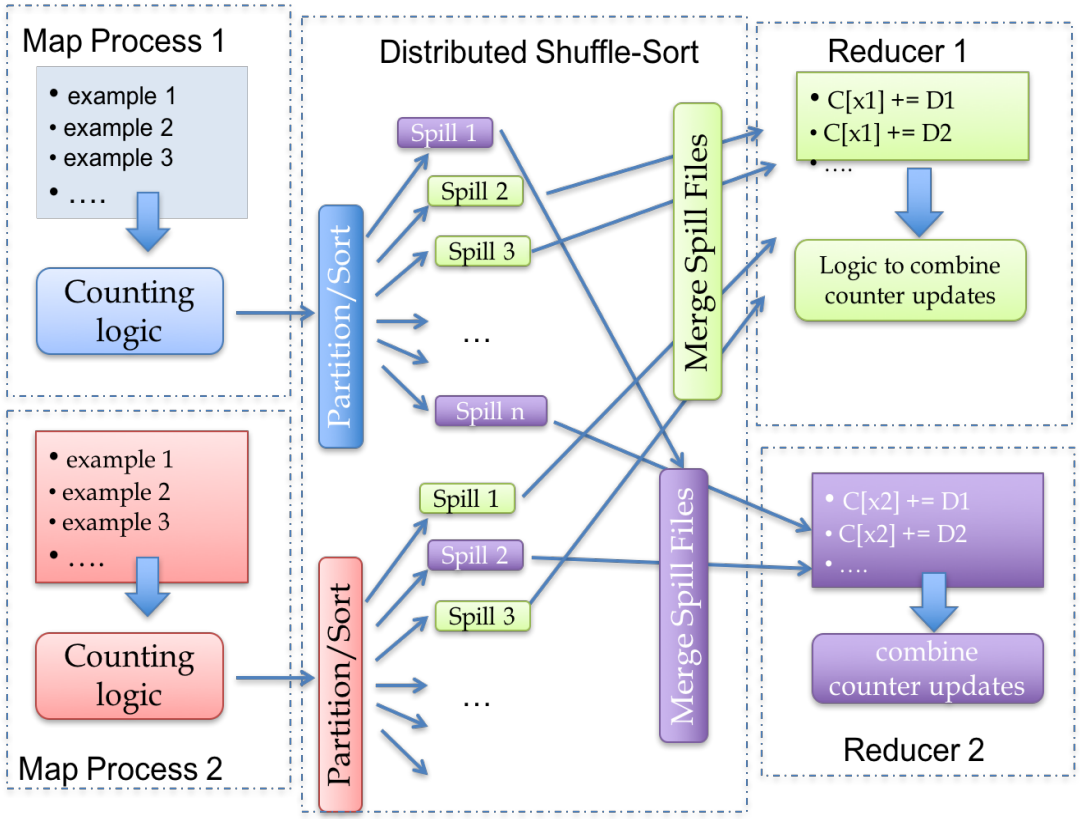
If necessary concatenate the reducer outputs to a single file.

Hadoop: Distributed Stream-and-Sort

Local implementation:

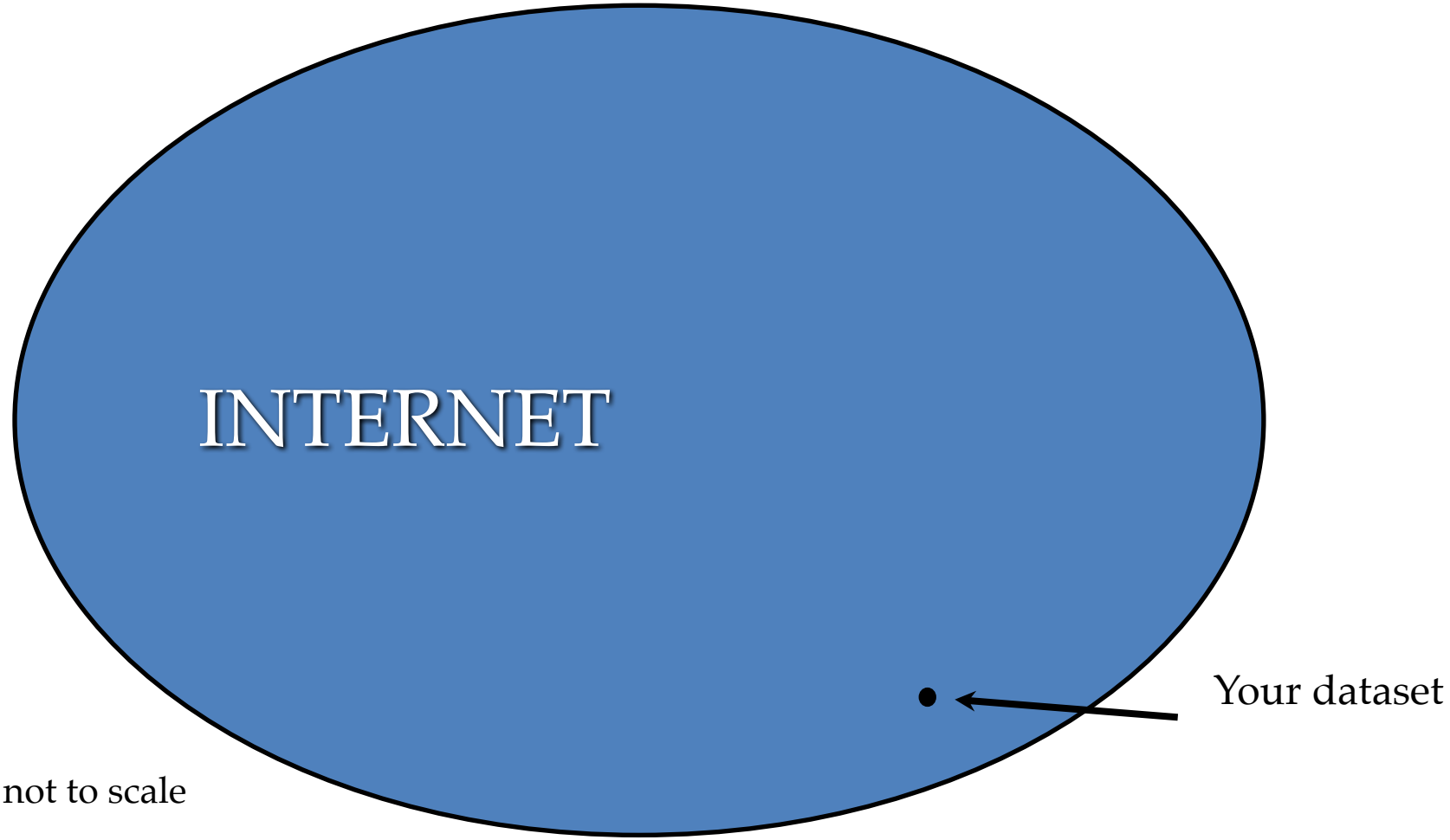
cat input.txt | MAP | sort | REDUCE > output.txt

In parallel
1. Map 2. Partition 3. Send
In parallel: 1. Accept 2. Merge 3. Reduce



You could do this easily as a class project ... but …

Motivating Example*



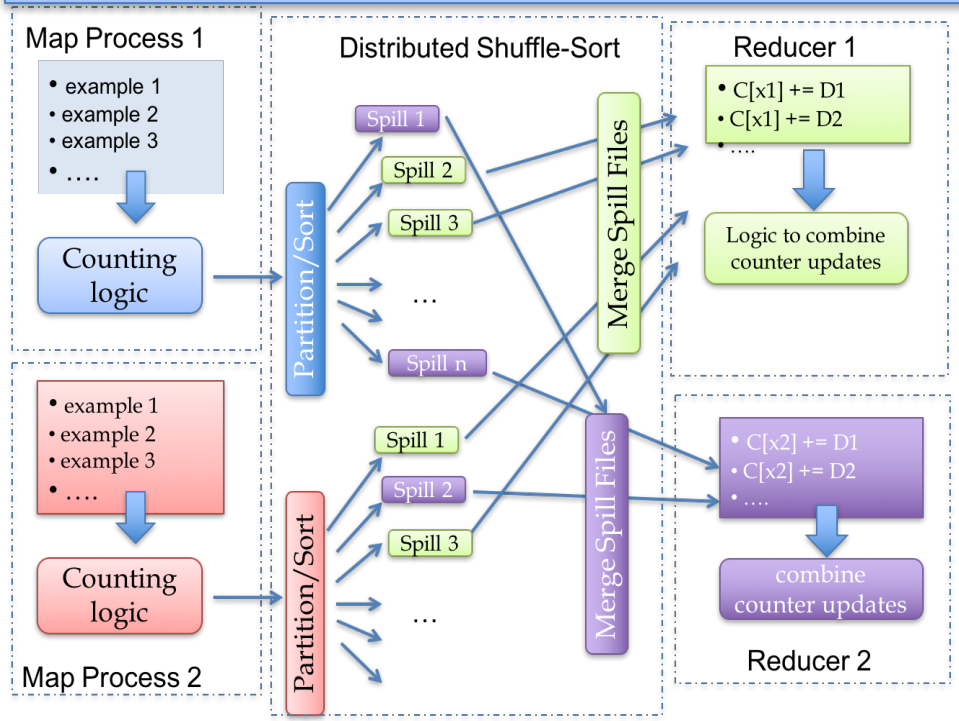
* not to scale

Hadoop: Distributed Stream-and-Sort

Robustness: with 10,000 machines, machines and disk drives fail *all the time*. How do you detect and recover?

Efficiency: with many jobs and programmers, how do you balance the loads? How do you limit network traffic and file i/o?

Usability: can programmers keep track of their data and jobs?



In parallel
1. Map
2. Partition
3. Send

In parallel:
1. Accept
2. Merge
3. Reduce

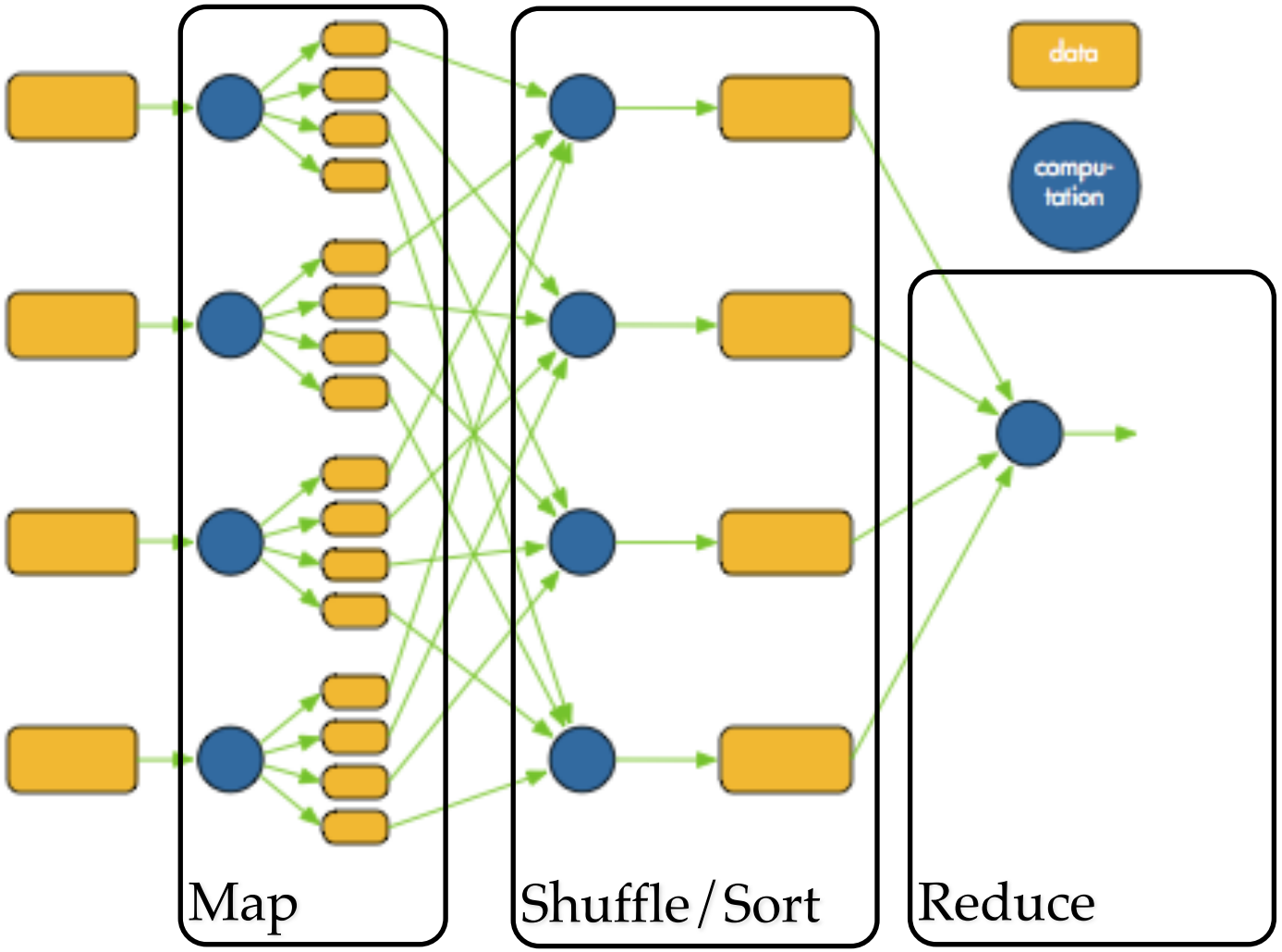
Hadoop: Intro

- pilfered from: Alona Fyshe, Jimmy Lin, Google, Cloudera
  - http://www.umiacs.umd.edu/~jimmylin/cloud-computing/SIGIR-2009/Lin-MapReduce-SIGIR2009.pdf
  - http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html
  - http://code.google.com/edu/submissions/mapreduce/listing.html

Surprise, you mapreduced!

- Mapreduce has three main phases
  - Map (send each input record to a key)
  - Sort (put all of one key in the same place)
    - handled behind the scenes
  - Reduce (operate on each key and its set of values)
  - Terms come from functional programming:
    - map(lambda x:x.upper(),["william","w","cohen"])➔['WILLIAM', 'W', 'COHEN']
    - reduce(lambda x,y:x+"-"+y,["william","w","cohen"])➔"william-w-cohen"

Mapreduce overview



Mapreduce: slow motion

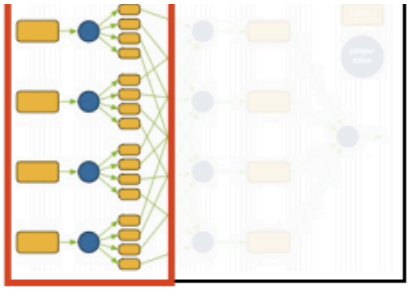
- The canonical mapreduce example is word count
- Example corpus:

    Joe likes toast

    Jane likes toast with jam

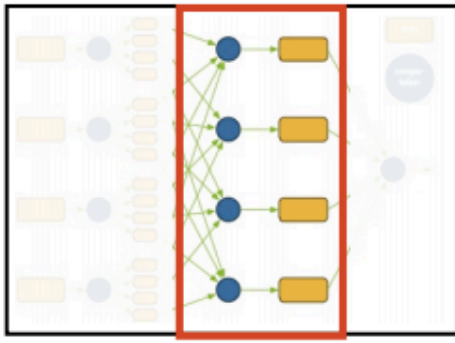
    Joe burnt the toast

MR: slow motion: Map



Input	Output	
Joe likes toast (Map 1)	Joe	1
	likes	1
	toast	1
Jane likes toast with jam (Map 2)	Jane	1
	likes	1
	toast	1
	with	1
	jam	1
Joe burnt the toast (Map 3)	Joe	1
	burnt	1
	the	1
	toast	1

MR: slow motion: Sort



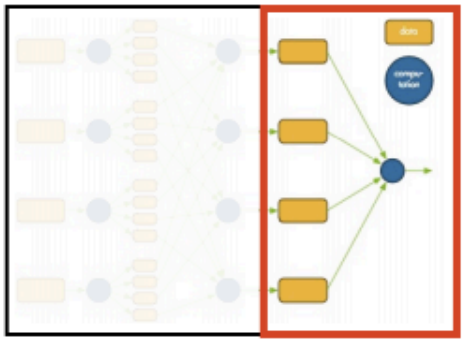
Input

Joe	1
likes	1
toast	1
Jane	1
likes	1
toast	1
with	1
jam	1
Joe	1
burnt	1
the	1
toast	1

Output

Joe	1
Joe	1
Jane	1
likes	1
likes	1
toast	1
toast	1
toast	1
with	1
jam	1
burnt	1
the	1

MR: slow mo: Reduce



Input

Joe	1	Reduce 1
Joe	1	
Jane	1	Reduce 2
likes	1	Reduce 3
likes	1	
toast	1	Reduce 4
toast	1	
toast	1	
with	1	Reduce 5
jam	1	Reduce 6
burnt	1	Reduce 7
the	1	Reduce 8

Output

Joe	2
Jane	1
likes	2
toast	3
with	1
jam	1
burnt	1
the	1

Issue: reliability

- Questions:
  - How will you know when each machine is done?
    - Communication overhead
  - How will you know if a machine is dead?
    - Remember: we can to use a huge pile of cheap machines, so failures will be common!
  - Is it dead or just really slow?

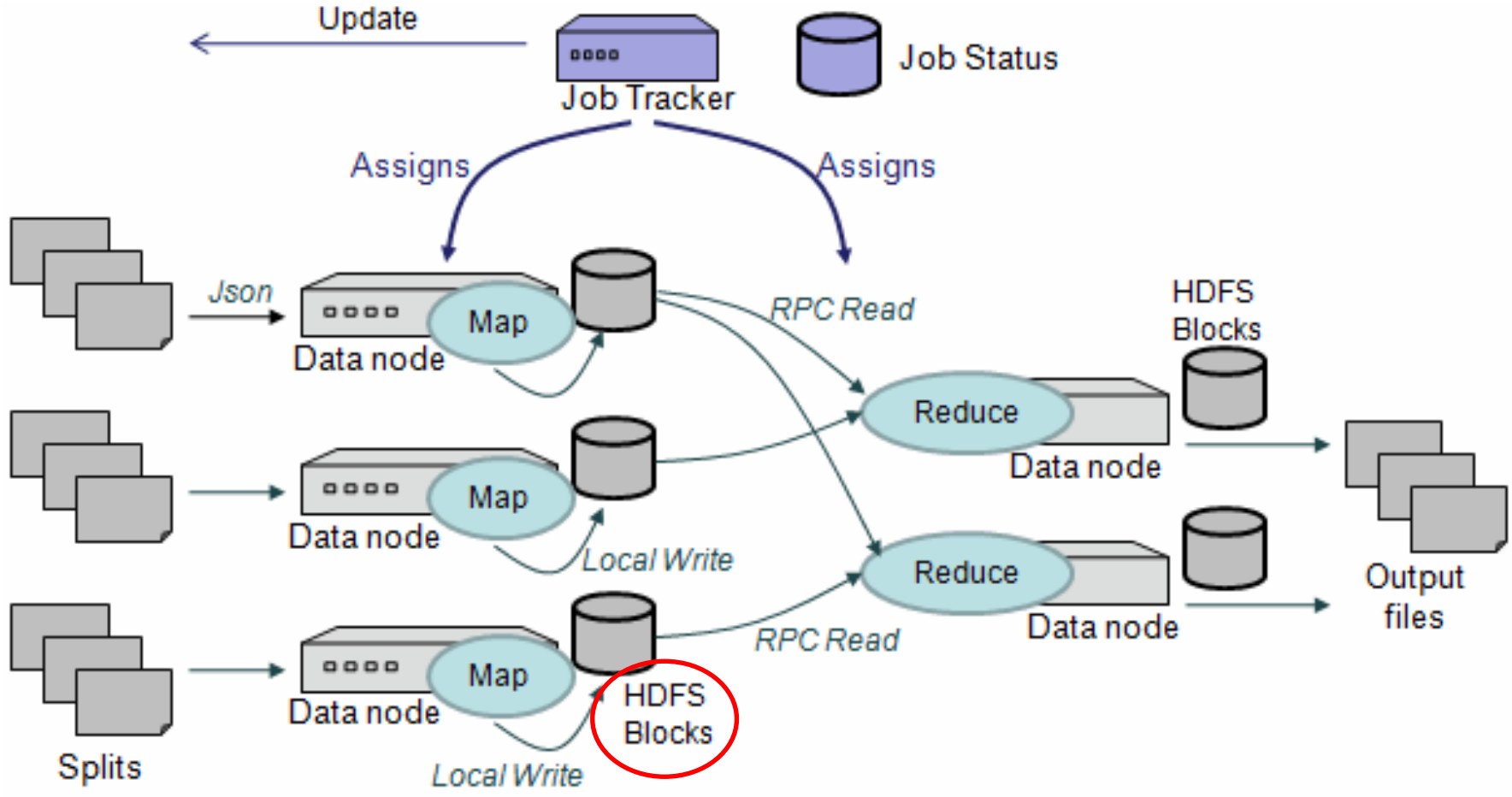
Issue: reliability

- What's the difference between slow and dead?
  - Who cares? Start a backup process.
    - If the process is slow because of machine issues, the backup may finish first
    - If it's slow because you poorly partitioned your data... waiting is your punishment

Issue: reliability

- If a disk fails you can lose some intermediate output
  - Ignoring the missing data could give you wrong answers

- Who cares? if I'm going to run backup processes I might as well have backup copies of the intermediate data also

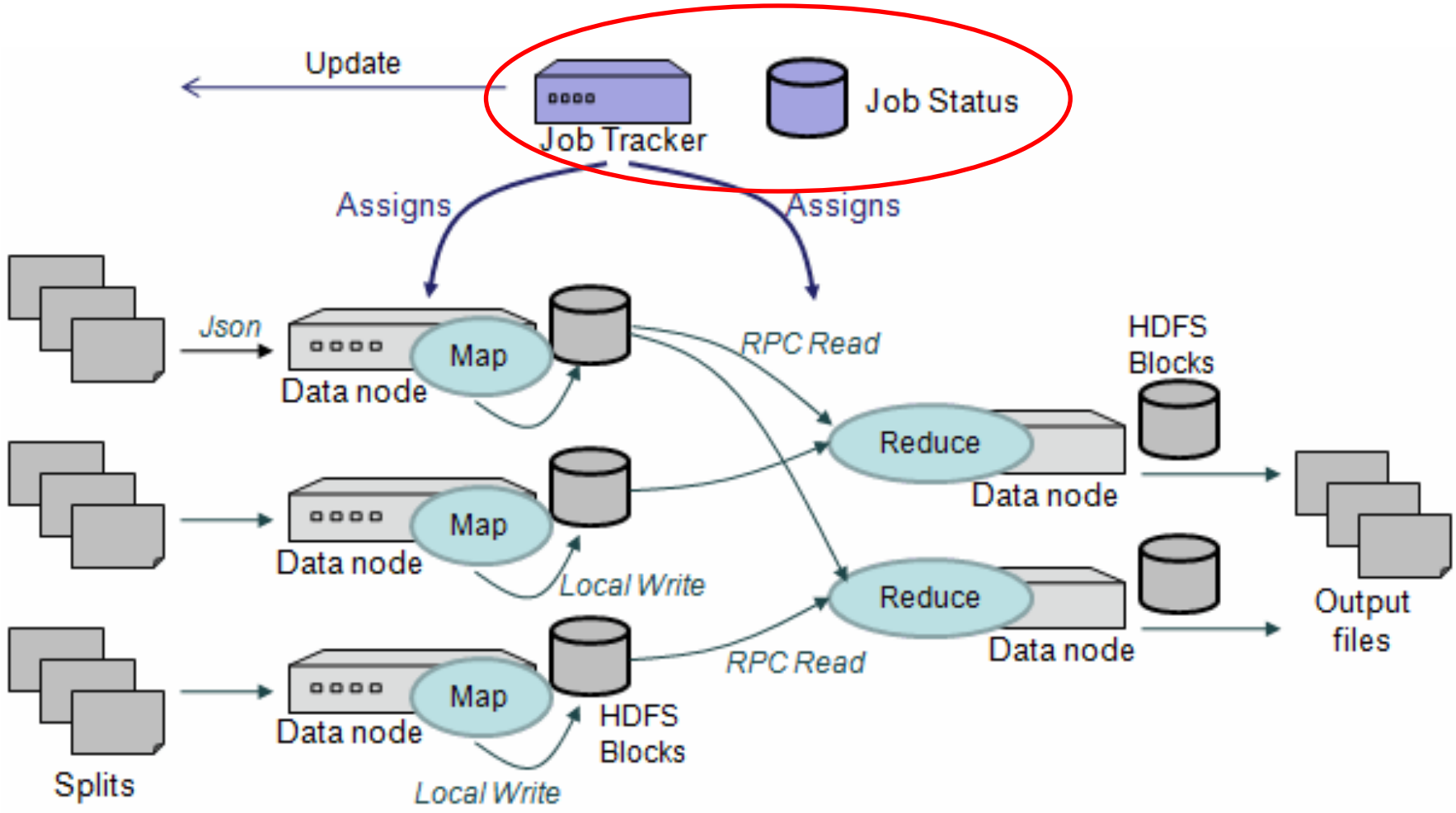

Update
Job Tracker
Job Status
Assigns
Assigns
Json
Map
Data node
RPC Read
HDFS
Blocks
Reduce
Data node
Map
Data node
Local Write
Reduce
Data node
Output
files
Map
Data node
RPC Read
HDFS
Blocks
Splits
Local Write

HDFS: The Hadoop File System

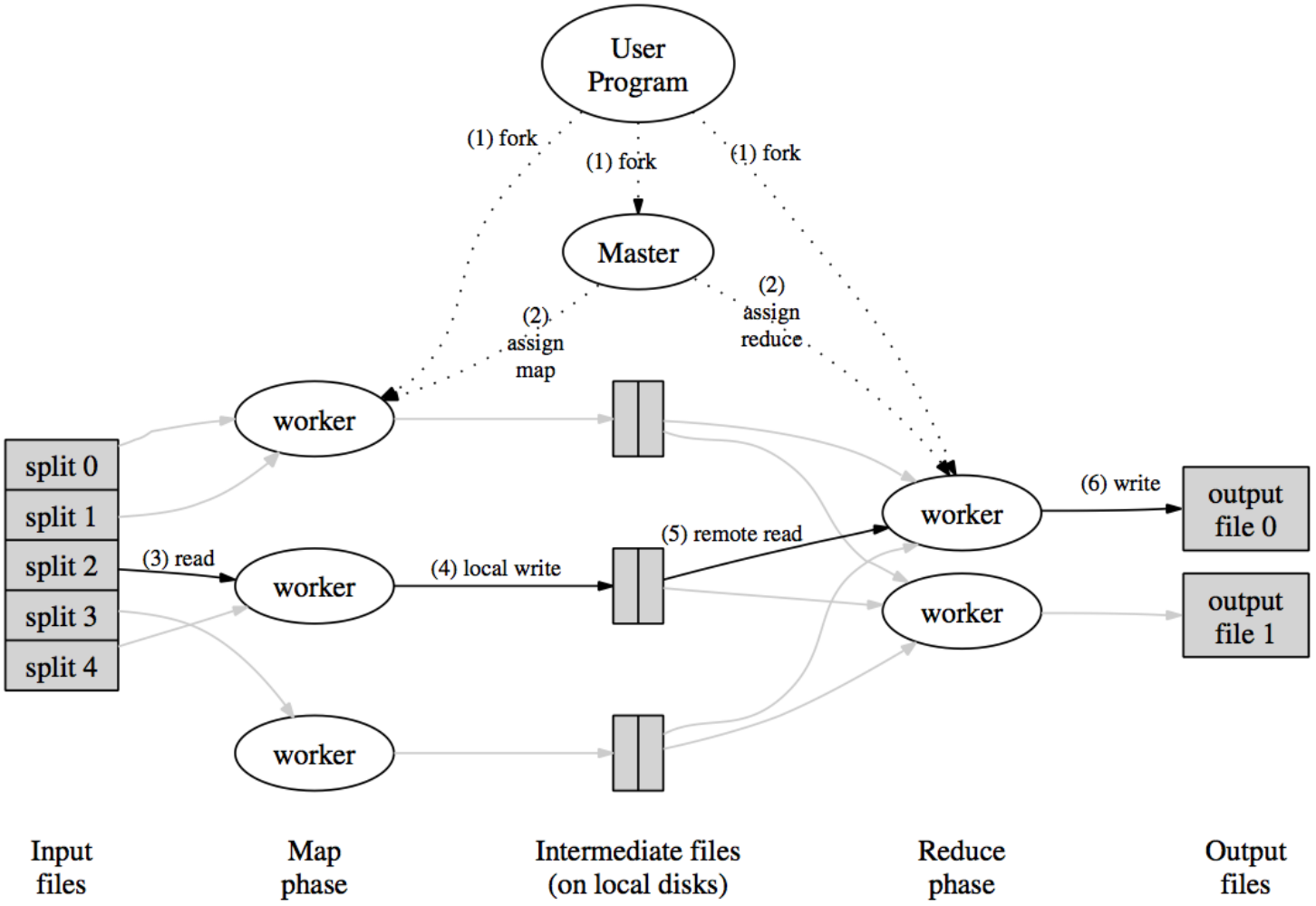
- Distributes data across the cluster
  - distributed file *looks like* a directory with shards as files inside it
  - makes an effort to run processes *locally* with the data
- Replicates data
  - default 3 copies of each file
- Optimized for streaming
  - really really big “blocks”

```
$ hadoop fs -ls rcv1/small/sharded
Found 10 items
-rw-r--r--   3 …  606405 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00000
-rw-r--r--   3 … 1347611 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00001
-rw-r--r--   3 …  939307 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00002
-rw-r--r--   3 … 1284062 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00003
-rw-r--r--   3 … 1009890 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00004
-rw-r--r--   3 … 1206196 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00005
-rw-r--r--   3 … 1384658 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00006
-rw-r--r--   3 … 1299698 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00007
-rw-r--r--   3 …  928752 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00008
-rw-r--r--   3 …  806030 2013-01-22 16:28  /user/wcohen/rcv1/small/sharded/part-00009

$ hadoop fs -tail rcv1/small/sharded/part-00005
weak as the arrival of arbitraged cargoes from the West has put the local market under pressure…
M14,M143,MCAT    The Brent crude market on the Singapore International …
```


Update
Job Tracker
Job Status
Assigns
Assigns
Json
Map
Data node
RPC Read
HDFS
Blocks
Reduce
Data node
Map
Data node
Local Write
Reduce
Data node
Output
files
Map
Data node
HDFS
Blocks
RPC Read
Splits
Local Write

MR Overview


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Hadoop job_201301231150_0778 on hadoopjt

User: wcohen
Job Name: streamjob6055532903853567038.jar
Job File: hdfs://hdfsname.opencloud/l/a2/scratch/hadoop-data/global/mapred/system/job_201301231150_0778/job.xml
Job Setup: Successful
Status: Failed
Started at: Wed Jan 30 11:46:47 EST 2013
Failed at: Wed Jan 30 11:47:28 EST 2013
Failed in: 41sec
Job Cleanup: Successful
Black-listed TaskTrackers: 2
Job Scheduling information: 5 running map tasks using 5 map slots, 0 running reduce tasks using 0 reduce slots.

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	10	0	0	0	10	35 / 5
reduce	00%	10	0	0	0	10	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Rack-local map tasks	0	0	38
	Launched map tasks	0	0	40
	Data-local map tasks	0	0	2
	Failed map tasks	0	0	1

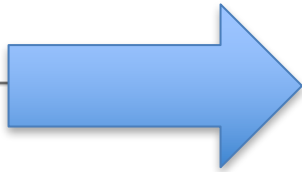
Map Completion Graph - close

Hadoop map task list for job_201301231150_0778 on hadoop

All Tasks

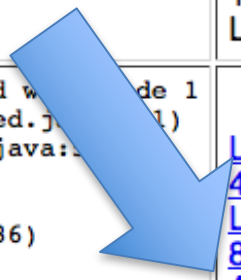
Task	Complete	Status	Start Time	Finish Time	Errors
task_201301231150_0778_m_000000	0.00%		30-Jan-2013 11:47:01	30-Jan-2013 11:47:25 (24sec)	java.lang.RuntimeException: PipeMa at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.mapre at org.apache.hadoop.mapre java.lang.RuntimeException: PipeMa at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.mapre at org.apache.hadoop.mapre java.lang.RuntimeException: PipeMa at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.strea at org.apache.hadoop.mapre at org.apache.hadoop.mapre at org.apache.hadoop.mapre java.lang.RuntimeException: PipeMa at org.apache.hadoop.strea

Job job_201301231150_0778



All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Error
attempt_201301231150_0778_m_000000_0	/default-rack/cloud3u12.opencloud	FAILED	0.00%	30-Jan-2013 11:47:01	30-Jan-2013 11:47:06 (4sec)	java.
attempt_201301231150_0778_m_000000_1	/default-rack/cloud2u28.opencloud	FAILED	0.00%	30-Jan-2013 11:47:07	30-Jan-2013 11:47:11 (4sec)	java.



me	Errors	Task Logs	Counters	Actions
2013	java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed w de 1 at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.j 1) at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java: at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132) at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:57) at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:36) at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:358) at org.apache.hadoop.mapred.MapTask.run(MapTask.java:307) at org.apache.hadoop.mapred.Child.main(Child.java:170)	Last 4KB Last 8KB All	1	
2013	java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1 at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311) at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:540) at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132) at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:57) at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:36) at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:358) at org.apache.hadoop.mapred.MapTask.run(MapTask.java:307) at org.apache.hadoop.mapred.Child.main(Child.java:170)	Last 4KB Last 8KB All	1	
2013	java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1 at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311) at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:540) at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132) at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:57) at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:36) at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:358) at org.apache.hadoop.mapred.MapTask.run(MapTask.java:307) at org.apache.hadoop.mapred.Child.main(Child.java:170)	Last 4KB Last 8KB All	1	

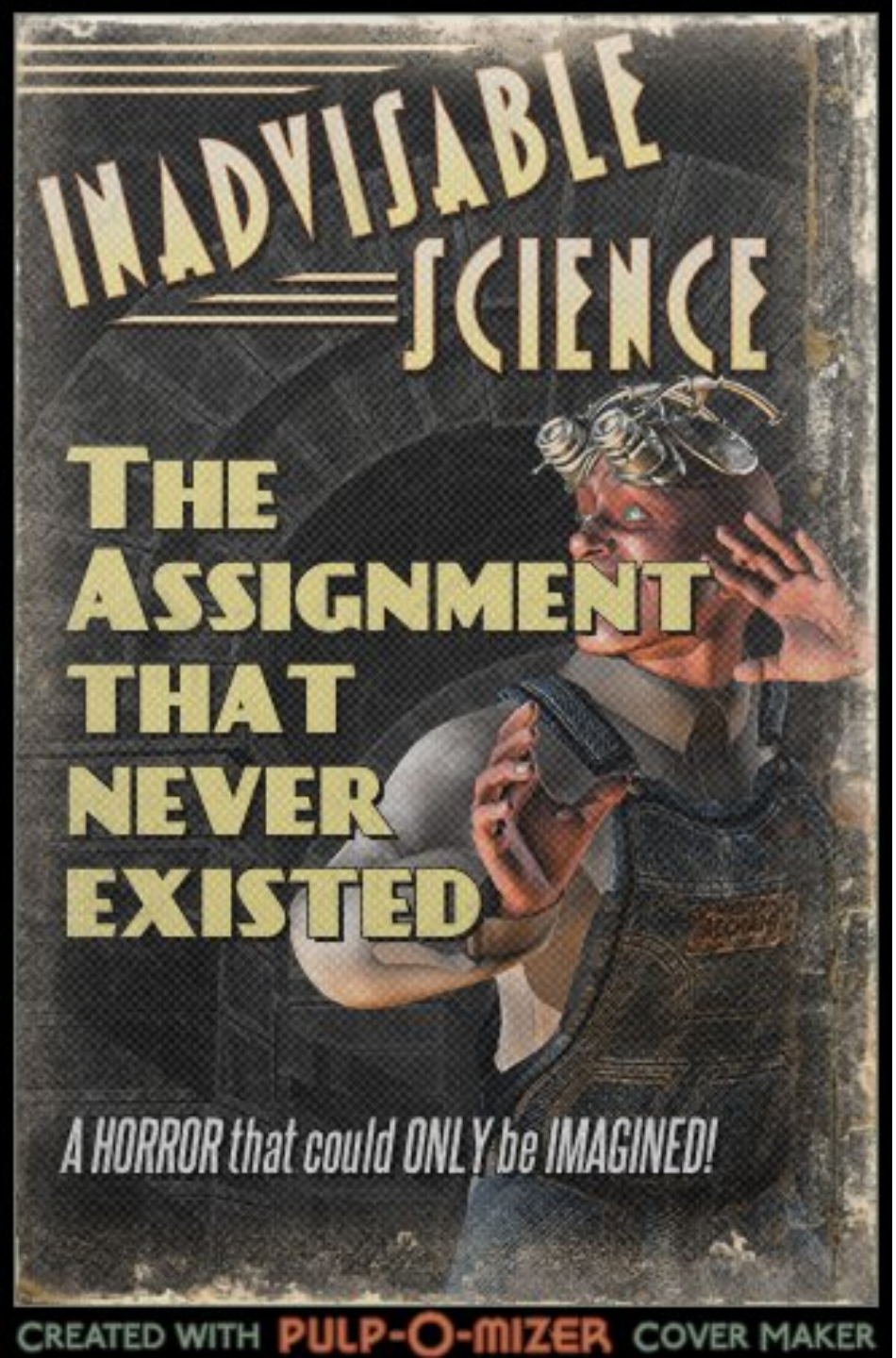
Task Logs: 'attempt_201301231150_0778_m_000000_0'

stdout logs

---

stderr logs

```
Exception in thread "main" java.lang.NoClassDefFoundError: com/wcohen/StreamNB
Caused by: java.lang.ClassNotFoundException: com.wcohen.StreamNB
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: com.wcohen.StreamNB.  Program will exit.
```



1. This would be pretty systems-y (remote copy files, waiting for remote processes, …)
2. It would take work to make run for 500 jobs

- Reliability: *Replication, restarts, monitoring jobs,…*
- Efficiency: *load-balancing, reducing file/network i/o, optimizing file/network i/o,…*
- **Useability**: *stream defined datatypes, simple reduce functions, ….*

Map reduce with Hadoop streaming

Breaking this down…

- Our imaginary assignment uses key-value pairs. What's the data structure for that? How do you interface with Hadoop?
- One very simple way: Hadoop's *streaming* interface.
  - Mapper outputs key-value pairs as:
    - One pair per line, key and value tab-separated
  - Reducer reads in data in the same format
    - Lines are sorted so lines with the same key are adjacent.

An example:

- SmallStreamNB.java and StreamSumReducer.java:

annotated-log.txt

```
wcohen@shell2:~/naive-bayes-demo$ ls -l
total 1170892
-rw-r--r--. 1 wcohen lcs       1288 Feb  5 11:16 Makefile
-rw-r--r--. 1 wcohen lcs  118656462 Jan 22  2013 RCV1.full_test.txt
-rw-r--r--. 1 wcohen lcs 1065782405 Jan 22  2013 RCV1.full_train.txt
-rw-r--r--. 1 wcohen lcs    1198026 Jan 22  2013 RCV1.small_test.txt
-rw-r--r--. 1 wcohen lcs   10812609 Jan 22  2013 RCV1.small_train.txt
-rw-r--r--. 1 wcohen lcs      18771 Jan 22  2013 RCV1.very_small_test.txt
-rw-r--r--. 1 wcohen lcs      99397 Jan 22  2013 RCV1.very_small_train.txt
-rw-r--r--. 1 wcohen lcs       7721 Jan 30  2013 SmallStreamNB.java
-rw-r--r--. 1 wcohen lcs       4162 Jan 26  2013 StreamNB.java
-rw-r--r--. 1 wcohen lcs        772 Jan 29  2013 StreamSumReducer.java
drwxr-xr-x. 3 wcohen lcs       4096 Jan 25  2013 classes
-rw-r--r--. 1 wcohen lcs      10802 Jan 30  2013 nb.jar
-rw-r--r--. 1 wcohen lcs        576 Jan 28  2013 notes.txt
wcohen@shell2:~/naive-bayes-demo$ cat Makefile
--:---  annotated-log.txt   Top L1     (Text)----------------------------------------
```

To run locally:

```
test-small: small-events.txt nb.jar
        time java -cp nb.jar com.wcohen.SmallStreamNB \
        RCV1.small_test.txt MCAT,CCAT,GCAT,ECAT 2000 < small-events.txt \
        | cut -f3 | sort | uniq -c

small-events.txt:  nb.jar
        time java -cp nb.jar com.wcohen.SmallStreamNB \
        < RCV1.small_train.txt | sort -k1,1 \
        | java -cp nb.jar com.wcohen.StreamSumReducer> small-events.txt
```

To train with streaming Hadoop you do this:

```
STRJAR = /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar

small-events-hs:
        hadoop fs -rmr rcv1/small/events
        hadoop jar $(STRJAR) \
           -input rcv1/small/sharded -output rcv1/small/events \
           -mapper 'java -Xmx512m -cp nb.jar com.wcohen.StreamNB' \
           -reducer 'java -Xmx512m -cp nb.jar com.wcohen.StreamSumReducer' \
           -file nb.jar -numReduceTasks 10
```

But first you need to get your code and data to the "Hadoop file system"

```
hadoop fs -rmr rcv1/small/events
Moved to trash: hdfs://t1disc-pnn:8020/user/wcohen/rcv1/small/events
time hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-1.2.0.1.3.0.0-107.jar  \
          -input rcv1/small/sharded -output rcv1/small/events \
          -mapper 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamNB' \
          -reducer 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamSumReducer' \
          -file nb.jar -numReduceTasks 10
packageJobJar: [nb.jar, /tmp/hadoop-wcohen/hadoop-unjar8935719391413249732/] [] /tmp/streamjob
l
14/02/05 11:24:56 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
14/02/05 11:24:56 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hac
08101c2729dc0c9ff3]
14/02/05 11:24:56 WARN snappy.LoadSnappy: Snappy native library is available
14/02/05 11:24:56 INFO util.NativeCodeLoader: Loaded the native-hadoop library
14/02/05 11:24:56 INFO snappy.LoadSnappy: Snappy native library loaded
14/02/05 11:24:56 INFO mapred.FileInputFormat: Total input paths to process : 10
14/02/05 11:24:57 INFO streaming.StreamJob: getLocalDirs(): [/l/a/hadoop/mapred, /l/b/hadoop/m
doop/mapred]
14/02/05 11:24:57 INFO streaming.StreamJob: Running job: job_201312100900_1189
14/02/05 11:24:57 INFO streaming.StreamJob: To kill this job, run:
14/02/05 11:24:57 INFO streaming.StreamJob: /usr/lib/hadoop/libexec/../bin/hadoop job  -Dmapre
mu.local:50300 -kill job_201312100900_1189
14/02/05 11:24:57 INFO streaming.StreamJob: Tracking URL: http://t1disc-jt.disc.pdl.cmu.local:
312100900_1189
14/02/05 11:24:58 INFO streaming.StreamJob:  map 0%  reduce 0%
14/02/05 11:25:09 INFO streaming.StreamJob:  map 20%  reduce 0%
14/02/05 11:25:14 INFO streaming.StreamJob:  map 57%  reduce 0%
14/02/05 11:25:15 INFO streaming.StreamJob:  map 77%  reduce 0%
14/02/05 11:25:16 INFO streaming.StreamJob:  map 79%  reduce 0%
14/02/05 11:25:17 INFO streaming.StreamJob:  map 79%  reduce 4%
14/02/05 11:25:18 INFO streaming.StreamJob:  map 80%  reduce 6%
14/02/05 11:25:19 INFO streaming.StreamJob:  map 70%  reduce 7%
14/02/05 11:25:24 INFO streaming.StreamJob:  map 80%  reduce 7%
14/02/05 11:25:25 INFO streaming.StreamJob:  map 80%  reduce 10%
14/02/05 11:25:26 INFO streaming.StreamJob:  map 90%  reduce 17%
14/02/05 11:25:27 INFO streaming.StreamJob:  map 100%  reduce 20%
14/02/05 11:25:28 INFO streaming.StreamJob:  map 100%  reduce 22%
14/02/05 11:25:31 INFO streaming.StreamJob:  map 100%  reduce 34%
14/02/05 11:25:32 INFO streaming.StreamJob:  map 100%  reduce 38%
14/02/05 11:25:33 INFO streaming.StreamJob:  map 100%  reduce 40%
14/02/05 11:25:34 INFO streaming.StreamJob:  map 100%  reduce 60%
14/02/05 11:25:35 INFO streaming.StreamJob:  map 100%  reduce 87%
14/02/05 11:25:36 INFO streaming.StreamJob:  map 100%  reduce 100%
14/02/05 11:25:37 INFO streaming.StreamJob: Job complete: job_201312100900_1189
14/02/05 11:25:37 INFO streaming.StreamJob: Output: rcv1/small/events
2.59user 0.13system 0:42.14elapsed 6%CPU (0avgtext+0avgdata 314512maxresident)k
0inputs+1080outputs (0major+25504minor)pagefaults 0swaps
```

To train with streaming Hadoop:

- First, you need to prepare the corpus by splitting it into *shards*
- … and distributing the shards to different machines:

```
wcohen@shell2:~/naive-bayes-demo$ hadoop fs --help
-help: Unknown command
Usage: java FsShell
           [-ls <path>]
           [-lsr <path>]
           [-du <path>]
           [-dus <path>]
           [-count[-q] <path>]
           [-mv <src> <dst>]
           [-cp <src> <dst>]
           [-rm [-skipTrash] <path>]
           [-rmr [-skipTrash] <path>]
           [-expunge]
           [-put <localsrc> ... <dst>]
           [-copyFromLocal <localsrc> ... <dst>]
           [-moveFromLocal <localsrc> ... <dst>]
           [-get [-ignoreCrc] [-crc] <src> <localdst>]
           [-getmerge <src> <localdst> [addnl]]
```

```
wcohen@shell2:~/naive-bayes-demo$ hadoop fs -ls rcv1/small/sharded
Found 10 items
-rw-r--r--   3 wcohen supergroup     606405 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00000
-rw-r--r--   3 wcohen supergroup    1347611 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00001
-rw-r--r--   3 wcohen supergroup     939307 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00002
-rw-r--r--   3 wcohen supergroup    1284062 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00003
-rw-r--r--   3 wcohen supergroup    1009890 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00004
-rw-r--r--   3 wcohen supergroup    1206196 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00005
-rw-r--r--   3 wcohen supergroup    1384658 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00006
-rw-r--r--   3 wcohen supergroup    1299698 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00007
-rw-r--r--   3 wcohen supergroup     928752 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00008
-rw-r--r--   3 wcohen supergroup     806030 2013-06-08 01:45 /user/wcohen/rcv1/small/sharded/part-00009
```

```
wcohen@shell2:~/naive-bayes-demo$ hadoop fs -tail rcv1/small/sharded/part-00005
weak as the arrival of arbitraged cargoes from the West has put the local market under pressure. In Singapor
e, May swaps fell to $21.30/$21.50 per barrelin late trade on Thursday from $21.70/$21.90 on Wednesday. Whil
e in Tokyo, first-half June open-spec naphtha was assessed at $207.00/$208.00 per tonne, compared with late
Wednesday's $213.00/$214.00. Added to these factors, traders said that a few petrochemicals were due to go i
nto turnaround in the next few weeks which would further dampen demand. -- Melanie Goodfellow, London Newsro
om, +44 171 542 7714.
M14,M143,MCAT    The Brent crude market on the Singapore International Monetary Exchange (SIMEX) will be clo
sed on Friday and Monday, SIMEX officials said on Tuesday. The closure will mark the corresponding closure o
n Friday and Monday of the Brent market on the International Petroleum Exchange (IPE) in London. SIMEX Brent
 operates a mutual offset system with the IPE in London, so SIMEX tends to close in line with the U.K.. --Si
ngapore Newsroom (+65 870 3081)
```

To train with streaming Hadoop:

- One way to shard text:
  - hadoop fs -put LocalFileName HDFSName
  - then run a streaming job with 'cat' as mapper and reducer
  - and specify the number of shards you want with option

    -numReduceTasks

To train with streaming Hadoop:

- Next, prepare your code for upload and distribution to the machines cluster

```
nb.jar: StreamSumReducer.java StreamNB.java SmallStreamNB.java
        javac -d classes StreamSumReducer.java StreamNB.java SmallStreamNB.java
        jar -cvf nb.jar -C classes .
```

To train with streaming Hadoop:

- Next, prepare your code for upload and distribution to the cluster

```
nb.jar: StreamSumReducer.java StreamNB.java SmallStreamNB.java
        javac -d classes StreamSumReducer.java StreamNB.java SmallStreamNB.java
        jar -cvf nb.jar -C classes .
```

Now you can run streaming Hadoop:

```
STRJAR = /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar

small-events-hs:
        hadoop fs -rmr rcv1/small/events
        hadoop jar $(STRJAR) \
           -input rcv1/small/sharded -output rcv1/small/events \
           -mapper 'java -Xmx512m -cp nb.jar com.wcohen.StreamNB' \
           -reducer 'java -Xmx512m -cp nb.jar com.wcohen.StreamSumReducer' \
           -file nb.jar -numReduceTasks 10
```

Map reduce without Hadoop streaming

"Real" Hadoop

- Streaming is simple but
  - There's no typechecking of inputs/outputs
  - You need to parse strings a lot
  - You can't use compact binary encodings
  - …
  - basically you have limited *control* over the messages you're sending
    - i/o costs = O(message size) often dominates

MR code: Word count Main

```
    public static void main(String[] args) throws Exception {

     Configuration conf = new Configuration();

     Job job = new Job(conf, "wordcount");

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

      job.setOutputKeyClass(Text.class);

     job.setOutputValueClass(IntWritable.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
}
```

MR Code: Word Count Map

```
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context) throws <stuff> {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
          word.set(tokenizer.nextToken());
          context.write(word, one);
      }
  }
}
```

MR code: Word count Reduce

```
    public static class Reduce extends Reducer<Text, IntWritable, Text,
    IntWritable> {

  public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
          sum += val.get();
      }
      context.write(key, new IntWritable(sum));
  }
}
```

"Real" Hadoop vs Streaming Hadoop

- Tradeoff: simplicity vs control
- In general:
  - If you want to really optimize you need to get down to the actual Hadoop layers
  - Often it's better to work with abstractions "higher up"

Debugging Map-Reduce

Some common pitfalls

- You have no control over the order in which reduces are performed
- You have no* control over the order in which you encounter reduce values
  - *by default anyway
- The only ordering you should assume is that Reducers always start after Mappers

Some common pitfalls

- You should assume your Maps and Reduces will be taking place on different machines with different memory spaces
- Don't make a static variable and assume that other processes can read it
  - They can't.
  - It appear that they can when run locally, but they can't
  - No really, don't do this.

Some common pitfalls

- Do not communicate between mappers or between reducers
  - overhead is high
  - you don't know which mappers/reducers are actually running at any given point
  - there's no easy way to find out what machine they're running on
    - because you shouldn't be looking for them anyway

When mapreduce doesn't fit

- The beauty of mapreduce is its separability and independence
- If you find yourself trying to communicate between processes
  - you're doing it wrong
    - »or
  - what you're doing is not a mapreduce

When mapreduce doesn't fit

- Not everything is a mapreduce
- Sometimes you need more communication
  - We'll talk about other programming paradigms later

What's so tricky about MapReduce?

- Really, nothing. It's easy.
- What's often tricky is figuring out how to write an *algorithm* as a series of *map-reduce* substeps.
  - How and when do you parallelize?
  - When should you even try to do this? when should you use a different model?

Performance

- IMPORTANT
  - You may not have room for all reduce values in memory
    - In fact you should PLAN not to have memory for all values
    - Remember, small machines are much cheaper
      - you have a limited budget

Combiners in Hadoop

Some of this is wasteful

- Remember - moving data around and writing to/reading from disk are very expensive operations
- No reducer can start until:
  - all mappers are done
  - data in its partition has been sorted

How much does buffering help?

BUFFER_SIZE	Time	Message Size
none		1.7M *words*
100	47s	1.2M
1,000	42s	1.0M
10,000	30s	0.7M
100,000	16s	0.24M
1,000,000	13s	0.16M
limit		0.05M

Combiners

- Sits between the map and the shuffle
  - Do some of the reducing while you're waiting for other stuff to happen
  - Avoid moving all of that data over the network
- Only applicable when
  - order of reduce values doesn't matter
  - effect is cumulative

```
  public static class Combiner extends Reducer<Text, IntWritable, Text,
  IntWritable> {

  public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
          sum += val.get();
      }
      context.write(key, new IntWritable(sum));
  }
}
```

Deja vu: Combiner = Reducer

- Often the combiner is the reducer.
  - like for word count
  - but not always

  - remember you have no control over when/whether the combiner is applied