



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

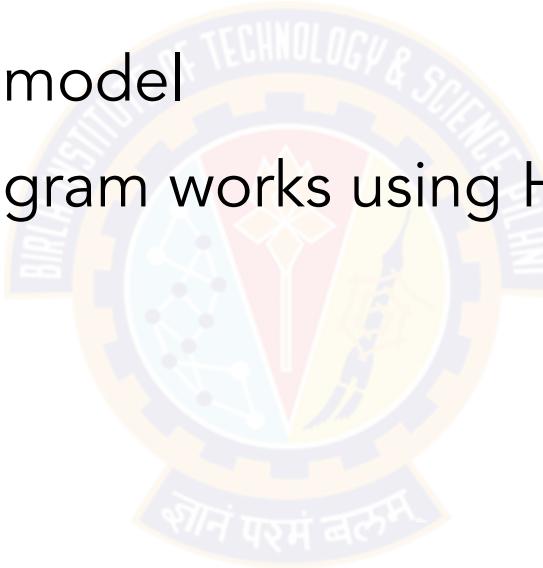
Big Data Systems

Session 5 - Distributed Programming

Dr. Anindya Neogi,
Associate Professor
anindya.neogi@pilani.bits-pilani.ac.in

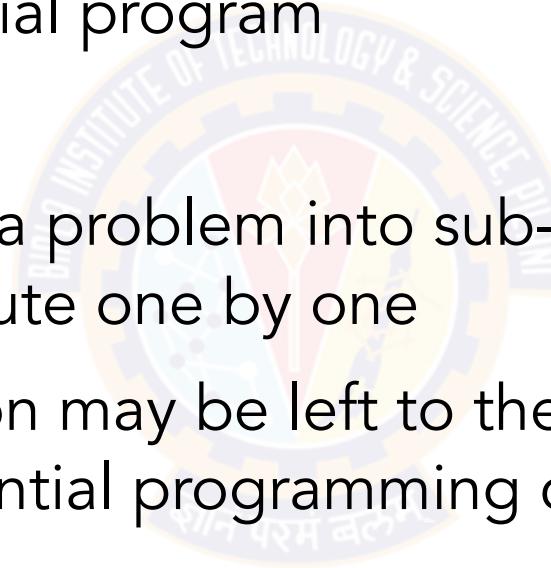
Topics for today

- Top down design
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce



Top down design - sequential context

- In the context of a sequential program
 - Divide and conquer
 - It is easier to divide a problem into sub-problems and execute one by one
 - A sub-problem definition may be left to the programmer in a sequential programming context



main()

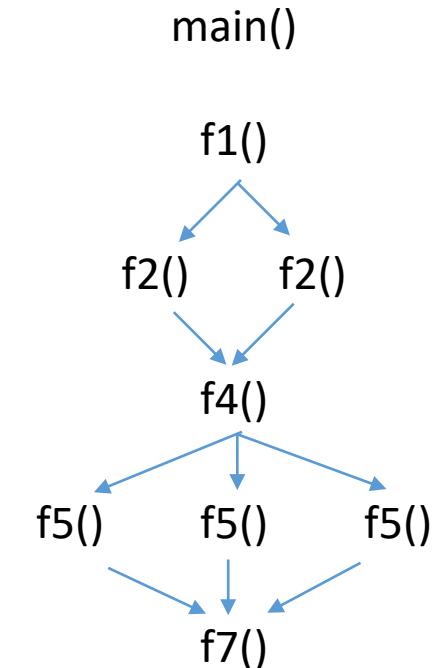
f1() → f2() → f5()

f3()

f4()

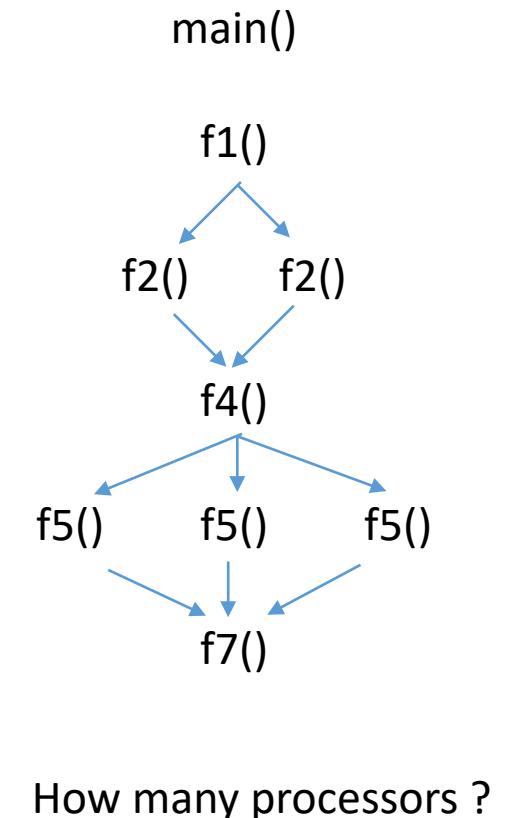
Top down design - parallel context

- We cannot decompose the problem into sub-problems in anyway the programmer chooses to
- Need to think about
 - Each sub-problem needs to be **assigned to a processor**
 - Goal is to get the program work faster
 - Divide the problem only when we can **combine** at the end into the final answer
 - Need to decide where to do the combination
 - Is there any parallelism in combination or is it sequential or trivial



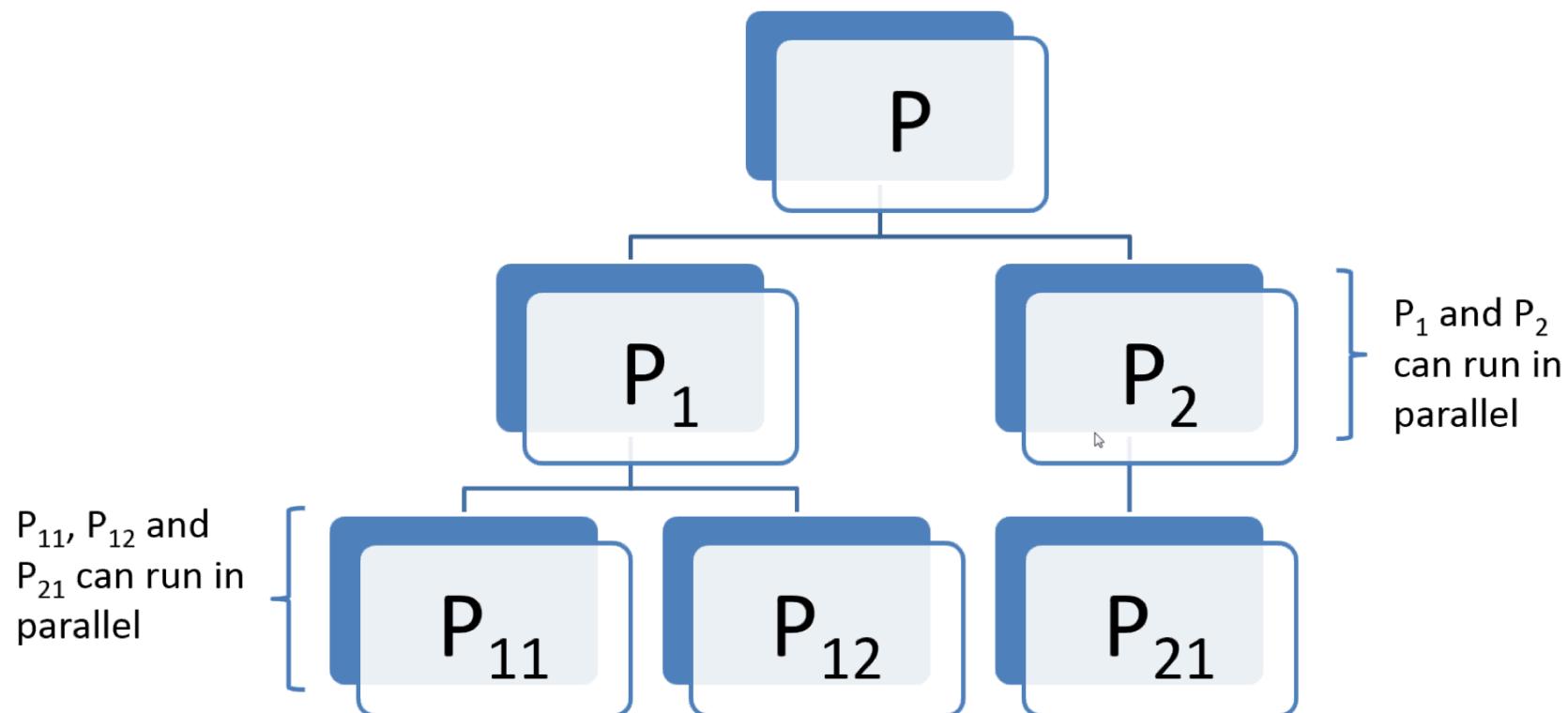
Deciding on number of sub-problems

- In conventional top down design for sequential systems
 - Keep **number of sub-problems manageable**
 - Because need to keep track of them as computation progresses
- In parallel system it is dictated by number of processors
 - **Processor utilisation is the key**
 - If there are N processors, we can potentially have N sub-problems



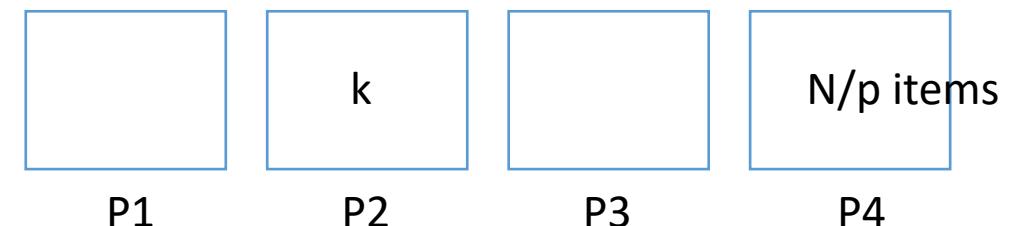
Top-down design

- At each level problems need to run in parallel



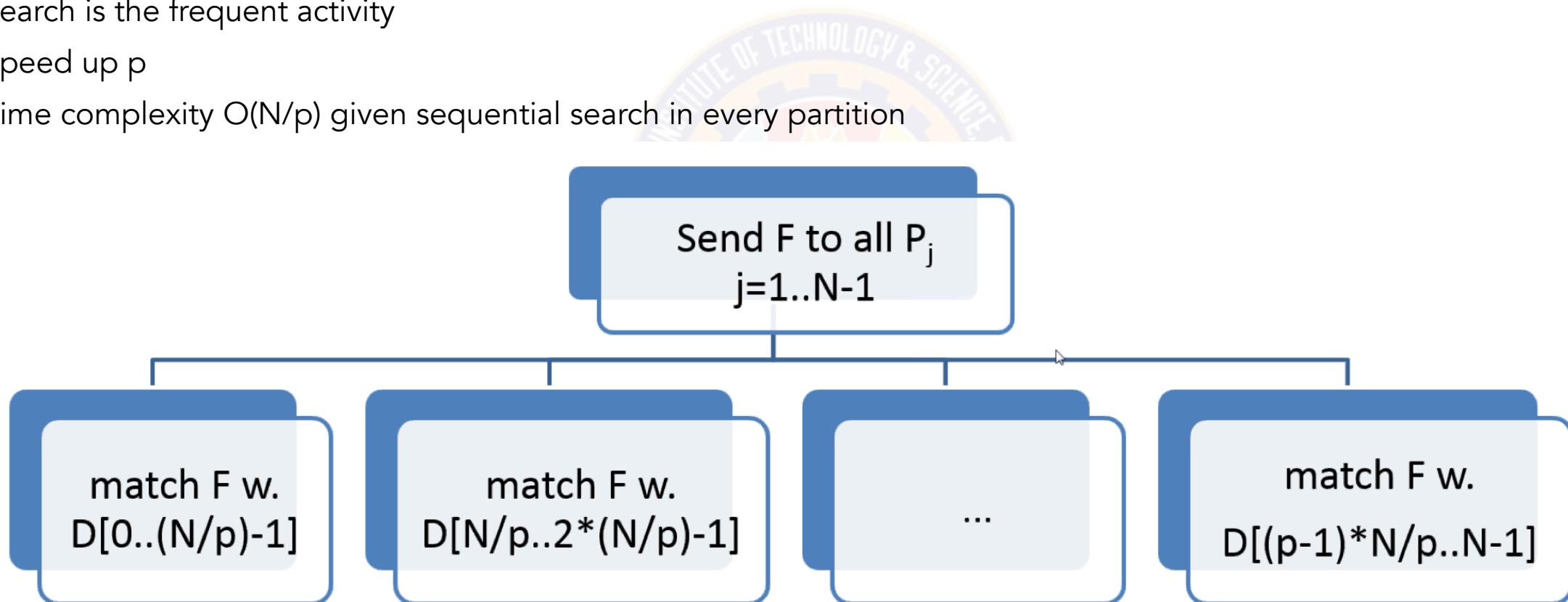
Example 1 - Keyword search in list

- Problem:
 - Search for a key k in a sorted list L_s of size N
- Data:
 - L_s is stored in a distributed system with p processors each storing N/p items
- Solution:
 - Run binary search in each of the p processors in parallel
 - Whichever processor finds k return (i, j) where i th processor has found key in j th position
 - Combination: One or more positions are collected at processor 0
- Speedup: p
- Time complexity: $O(\log(N/p))$



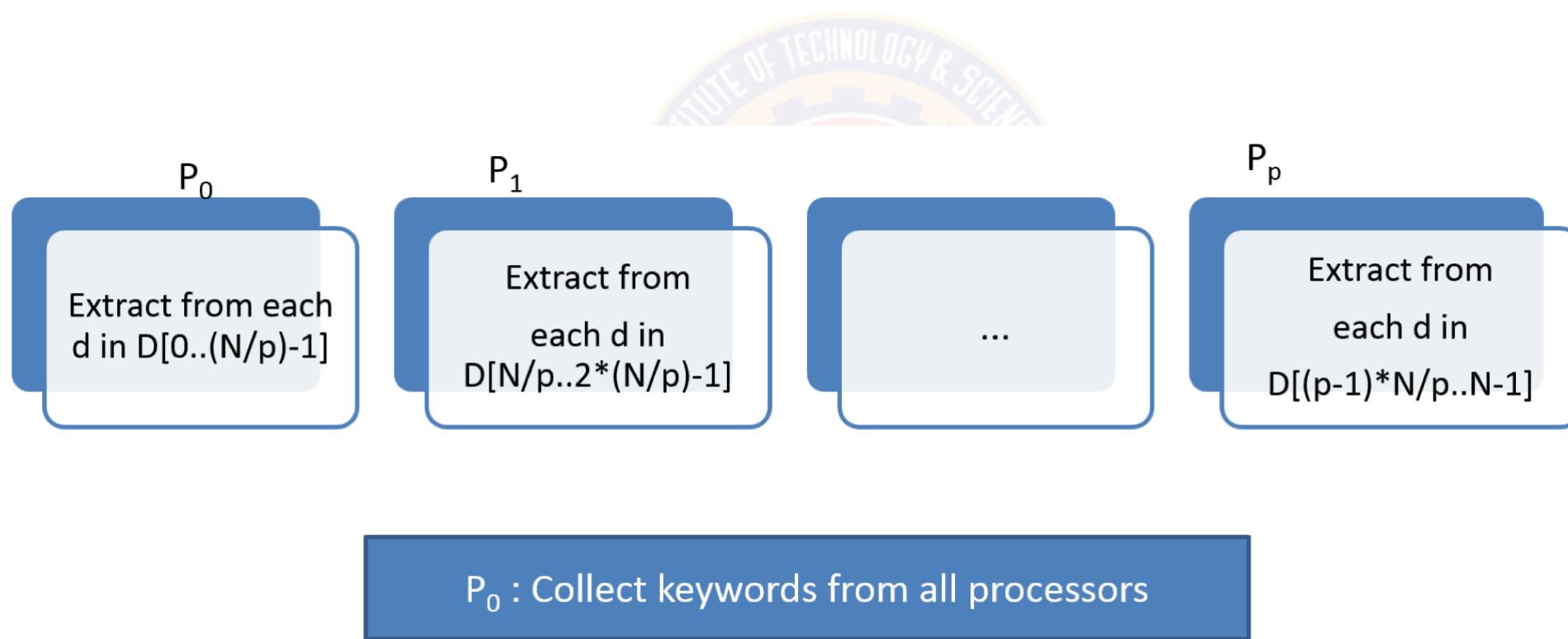
Example 2 - Fingerprint matching

- Find matches for a fingerprint F in a database of D prints
- Set of D prints is partitioned and evenly stored in a distributed database
- Partitioning is an infrequent activity - only when many new entries in database
- Search is the frequent activity
- Speed up p
- Time complexity $O(N/p)$ given sequential search in every partition



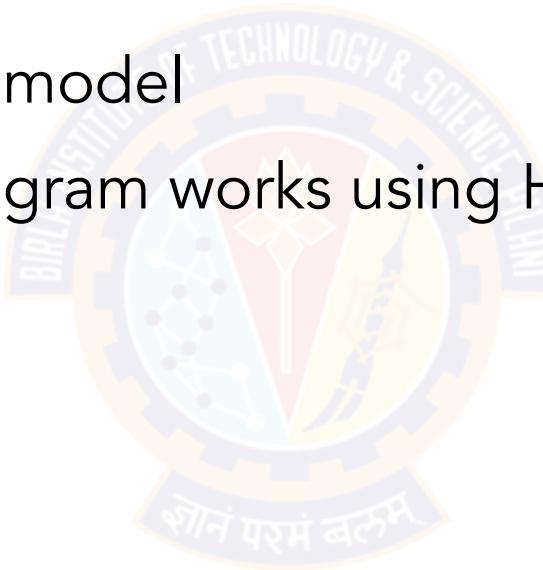
Example 3: Document search

- Find keywords from each document d in a distributed document collection D



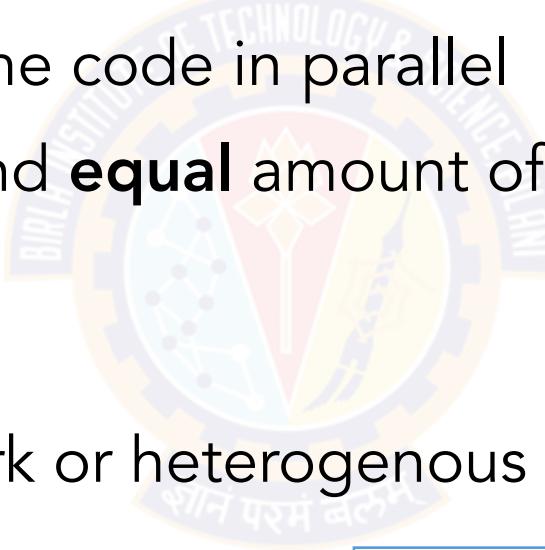
Topics for today

- Top down design
- **Types of parallelism**
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce



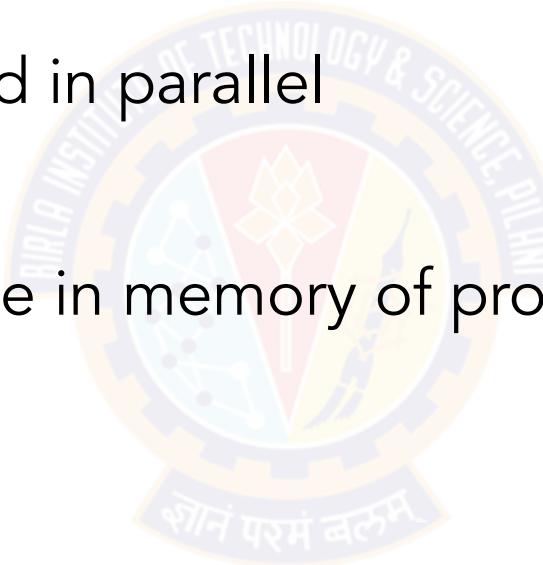
Data parallel execution model

- Data is partitioned to multiple nodes / processors
 - Try to make partitions equal or balanced
- All processors execute the same code in parallel
 - For **homogenous** nodes and **equal** amount of work, the utilization will be close to 100%
 - Execution time is minimal
 - Unbalanced data size / work or heterogenous nodes will lead to higher execution time



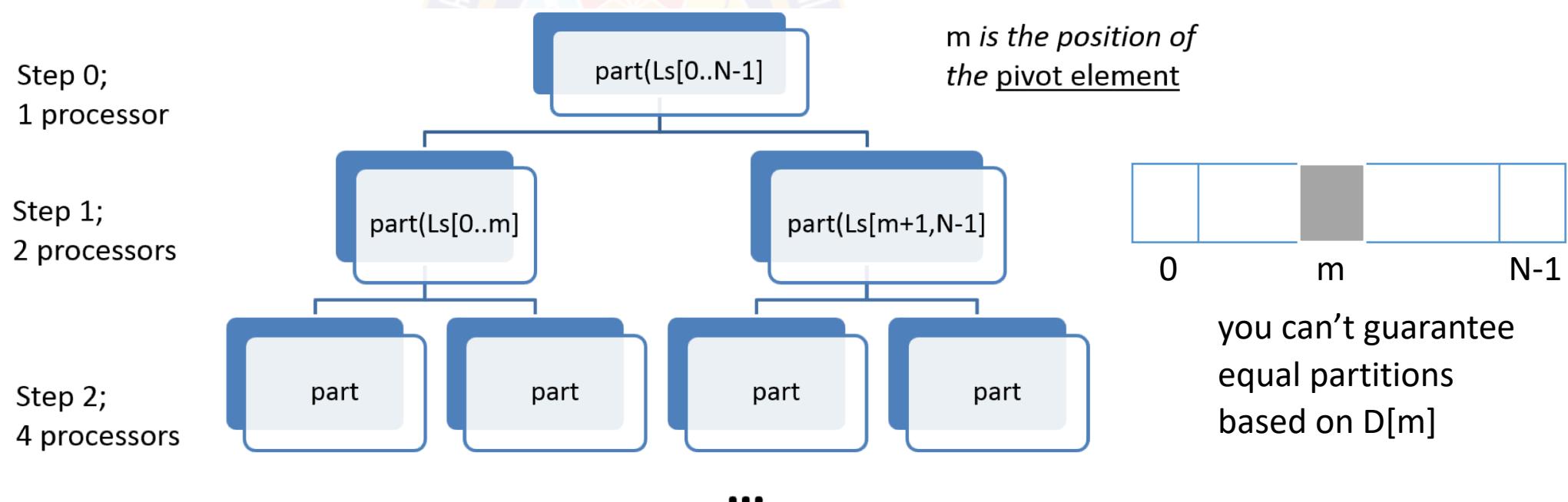
Where data parallelism is not possible

- There are problems where you cannot divide the work
 1. equally
 2. independently to proceed in parallel
- QuickSort(L_s, N)
 - All N items in L_s have to be in memory of processor 0



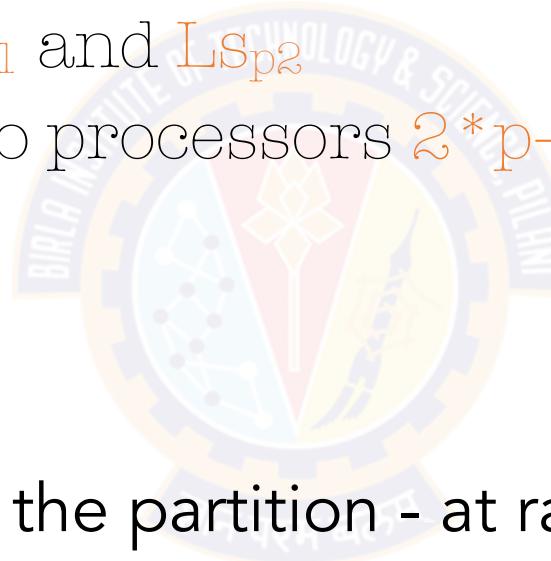
Example : QuickSort

- Pick a **pivot element** at position m to partition L_s and partition into sub-problems
- Do that in each level
- There is dependency on parent level to partition - not a single level problem ($\log N$ or $\log p$ levels which ever is lower)
- Choice of m cannot guarantee equal partition
 - At a level one set of processors can get large partitions and another set small partitions
 - Could be techniques to maintain balanced partitions and improve processor utilization uniformly



Tree parallel execution model for Quicksort parallel logic

- In step j
 - foreach processor p from 1 to 2^{j-1} do
 - partition L_{Sp} into L_{Sp1} and L_{Sp2}
 - assign L_{Sp1} and L_{Sp2} to processors 2^*p-1 and 2^*p
 - $j = j + 1$
 - repeat until $2^j == N$
- Depends on how good is the partition - at random ?
 - May be over long term for large lists and many processors
 - Time taken may be as bad as sequential with bad partitioning

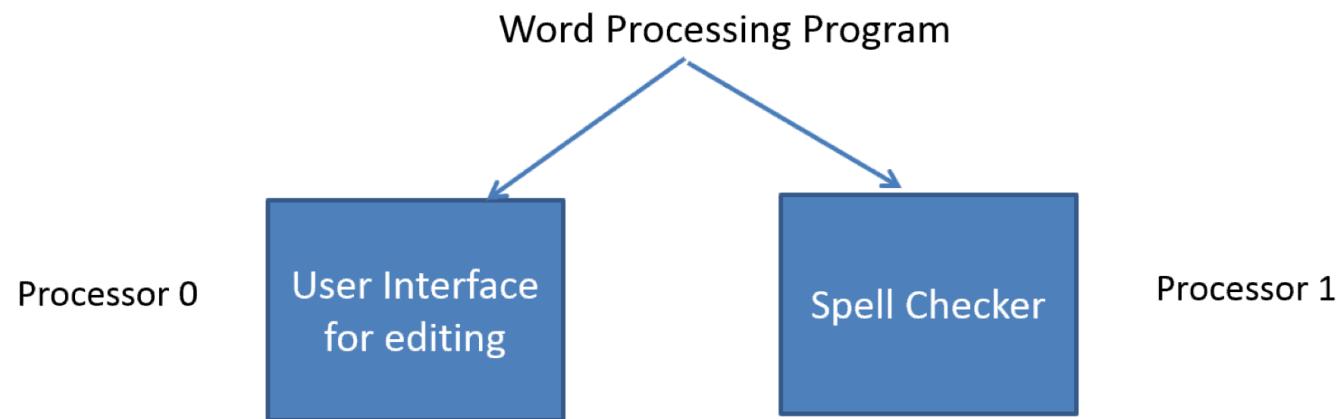


Tree parallelism summary

- Dynamic version of divide and conquer - **partitions are done dynamically**
- Division of problem into sub-problems happens execution time
 - **Sub-problem is identical in structure to the larger problem**
 - **What is the division step ?**
 - In quick sort it was picking m to split into 2 sub-problems
 - Division / partitioning logic is important to find almost equal sub-problems
- If problem is divided into k sub-problems
 - then in $\log_k N$ steps needed if N processors execute in parallel
 - If $p = N$ then work gets done in $\log(N)$ time with each list item assign to one processor finally
- What if we assign p processors with
 - $p < N$: so that all processors are utilised
 - $p > N$: under-utilised processors

Task parallelism - Example 1 : Word processor

- Parallel tasks that work on the same data
 - Unlike data and tree parallel **Data doesn't need to be divided**, the Task gets divided into sub-tasks
 - May work on same data instance, else need to make data copies and keep them in sync
- If on multiple core, different threads can execute tasks in parallel accessing same data instance in memory



Task parallelism - Example 2 : Independent statistics

- Given a list L_s of numeric values find its mean, median and mode
- Solution
 - Independent tasks on same data
 - Each task can find a statistic on L_s
 - Run tasks in parallel

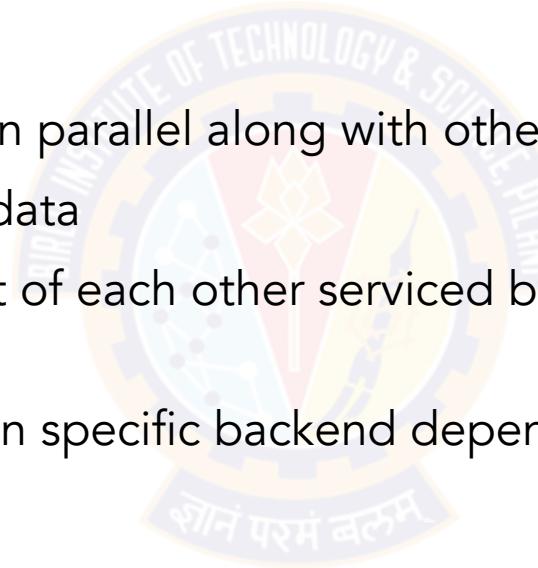


Task parallelism summary

- Identify sub-tasks based on functionality with no common function
 - **In Tree and Data parallel the tasks are identical function**
- Sub-tasks are not identified based on data
- Independent sub-tasks are executed in parallel
- Sub-tasks are often limited and known statically in advance
 - We know in a word processor what are the sub-tasks
 - We know in statistical analysis what functions we will run in advance
 - So **limited parallelism scope** - not scalable with more resources
 - **In data or tree parallelism we can potentially get more parallelism with more data** - more scalable with more resources at same time interval

Request parallelism

- Problem
 - **Scalable execution of independent tasks in parallel**
 - Execute same code but in many parallel instances
- Solution
 - On arrival, each request is serviced in parallel along with other existing tasks servicing prior requests
 - Could be processing same or fixed data
 - Request-reply pairs are independent of each other serviced by a different thread or process in the backend
 - There could be some application specific backend dependency, e.g. GET and POST on same data item
- Systems Fit
 - Servers in client-server models
 - e.g. email server, HTTP web-server, cloud services with API interface, file / storage servers
- Scalability metrics : Requests / time (throughput)



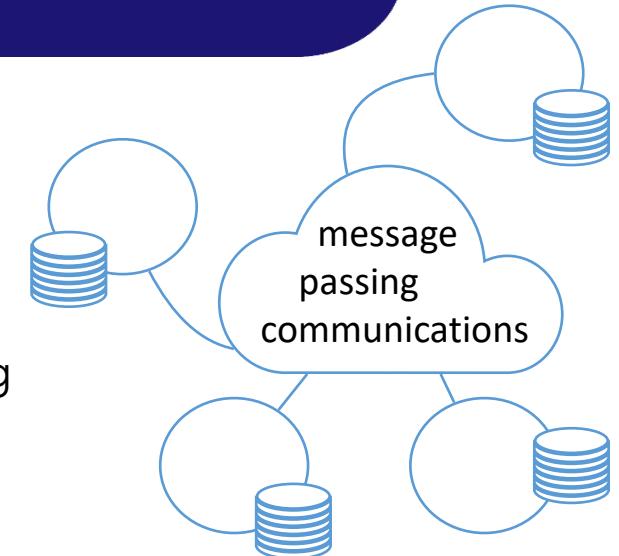
Exercise: What type of parallelism

- Historical sales data analysis to find monthly sales aggregates
- Web crawler - traverses the web as a graph following links (edges) to visit websites (nodes) to download unique documents



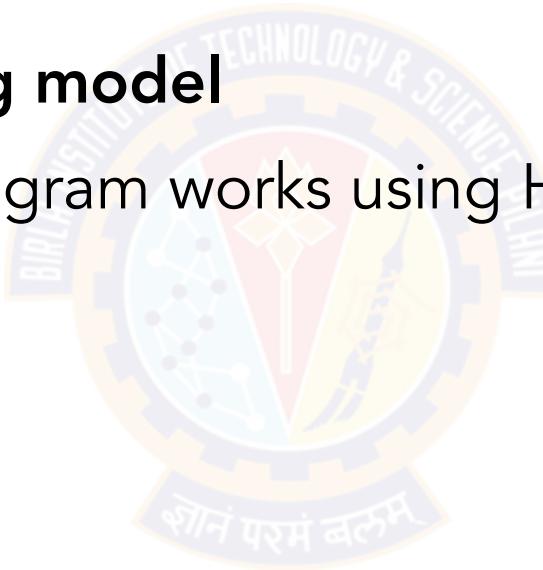
What happens in a loosely coupled distributed system

- Divide
 - No shared memory
 - Memory / Storage is on separate nodes
 - So any exchange of data or coordination between tasks is via message passing
 - **Divide the problem in a way that computation task can run on local data**
- Conquer / Merge
 - In shared memory merge it is simpler with each process writing into a memory location
 - In distributed
 - **Need to collect data from the different nodes**
 - In search example, it is a simpler merge to just collect result - so low cost
 - In quick sort, it is simple append whether writing in place for shared memory or sending a message
 - **Sometimes merges may become sequential**
 - e.g. k-means - in each iteration (a) guess clusters in parallel to improve the clusters but (2) checking if we have found right clusters is sequential



Topics for today

- Top down design
- Types of parallelism
- **MapReduce programming model**
- See how a map reduce program works using Hadoop
- Iterative MapReduce



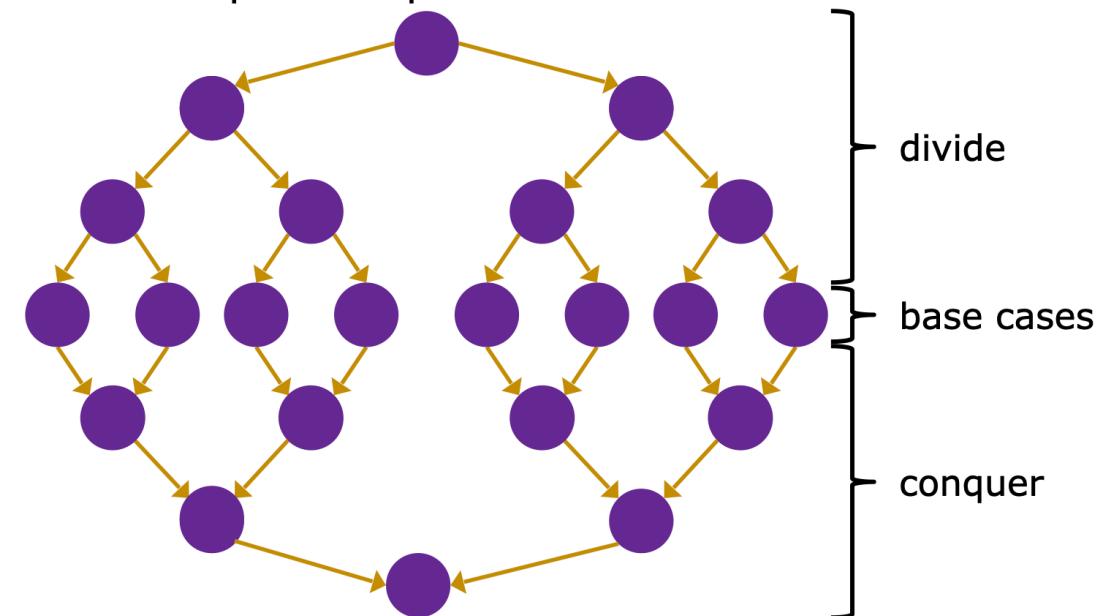
MapReduce in terms of Data and Tree parallelism

- **Map**

- Data parallelism
- Divide a problem into sub-problems based on data

- **Reduce**

- Inverse tree parallelism
- With every merge / reduce the parallelism reduces until we get one result
- Depending on the problem “reduce” step may be simple or sequential



Example: Word Count using MapReduce

map(key, value):

```
// key: document name; value: text of document  
for each word w in value:
```

```
emit(w, 1)
```

reduce(key, values):

```
// key: a word; value: an iterator over counts
```

```
result = 0
```

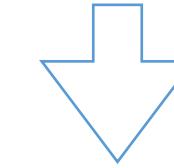
```
for each count v in values:
```

```
    result += v
```

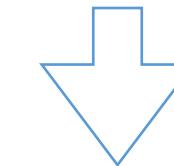
```
emit(result)
```

D1 : the blue ship on blue sea

the, 1 | blue, 1 | ship, 1 | on, 1
blue, 1 | sea, 1

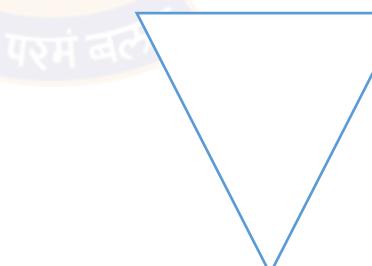
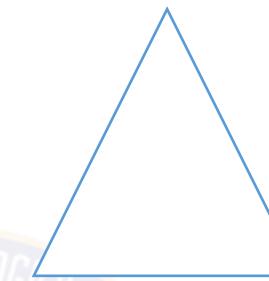


blue, [1,1] | on, 1 | sea, 1 | ship, 1 | the, 1

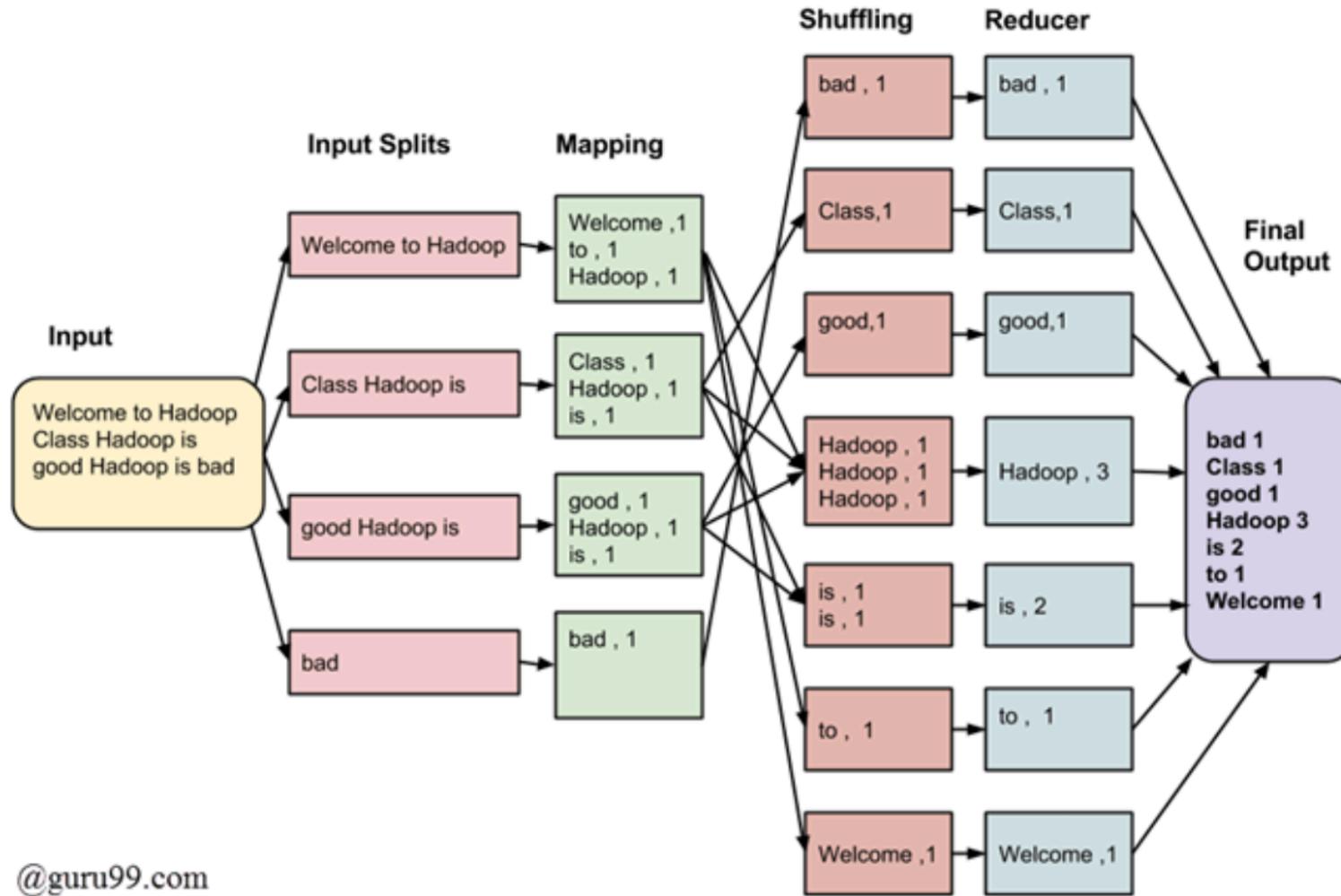


sort is done on keys
to have k,v with same k
value together

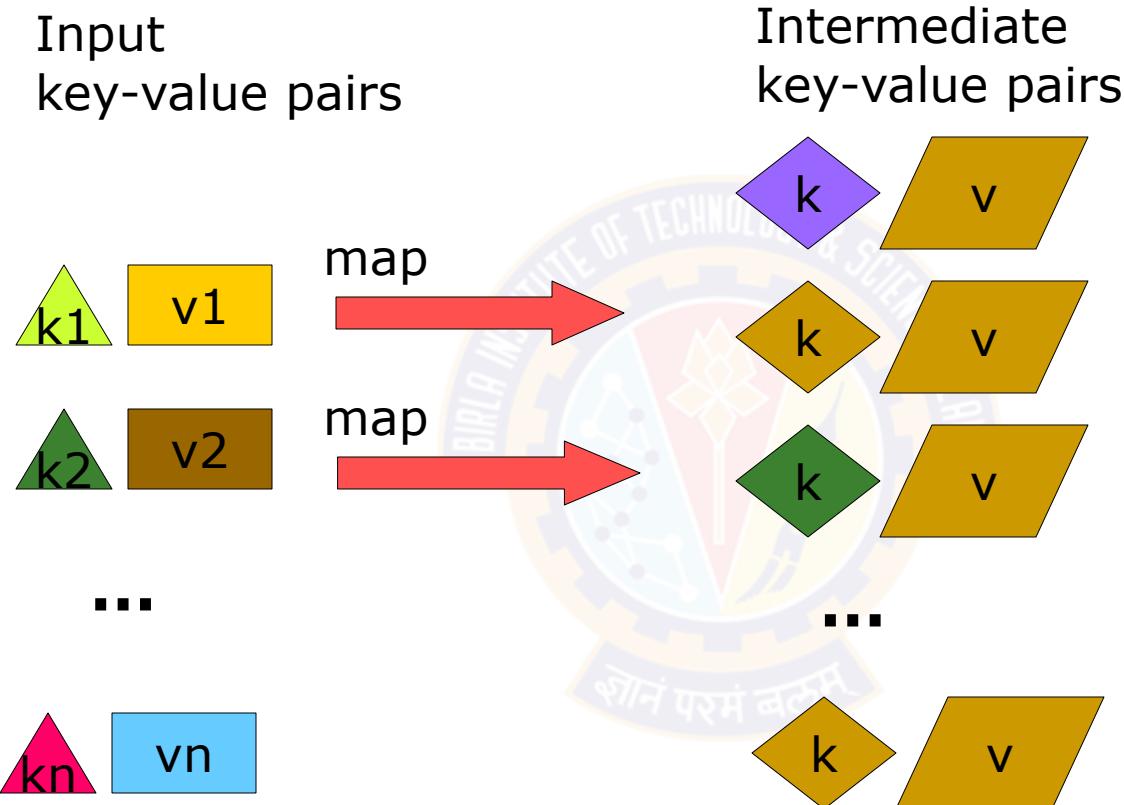
blue, 2 | on, 1 | sea, 1 | ship, 1 | the, 1



Word count (2)



MapReduce: The Map Step

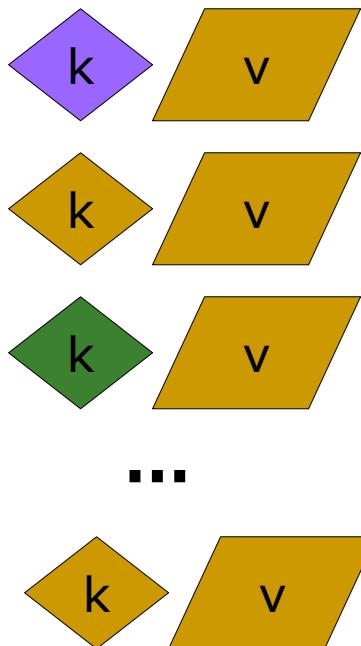


E.g. (doc—id, doc-content)

E.g. (word, wordcount-in-a-doc)

MapReduce: The Reduce Step

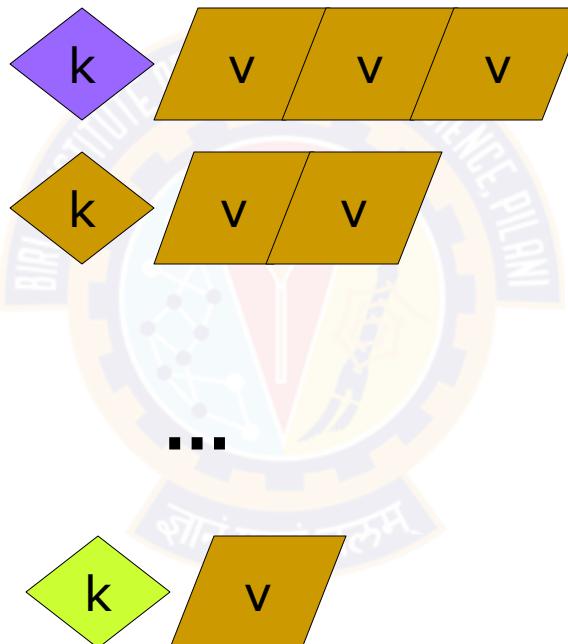
Intermediate key-value pairs



E.g.
(word, wordcount-in-a-doc)

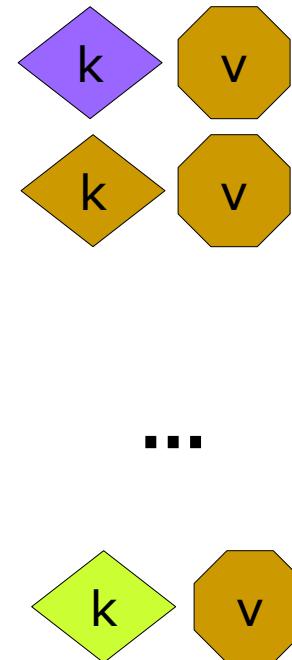
group

Key-value groups



(word, list-of-wordcount)
~ SQL Group by

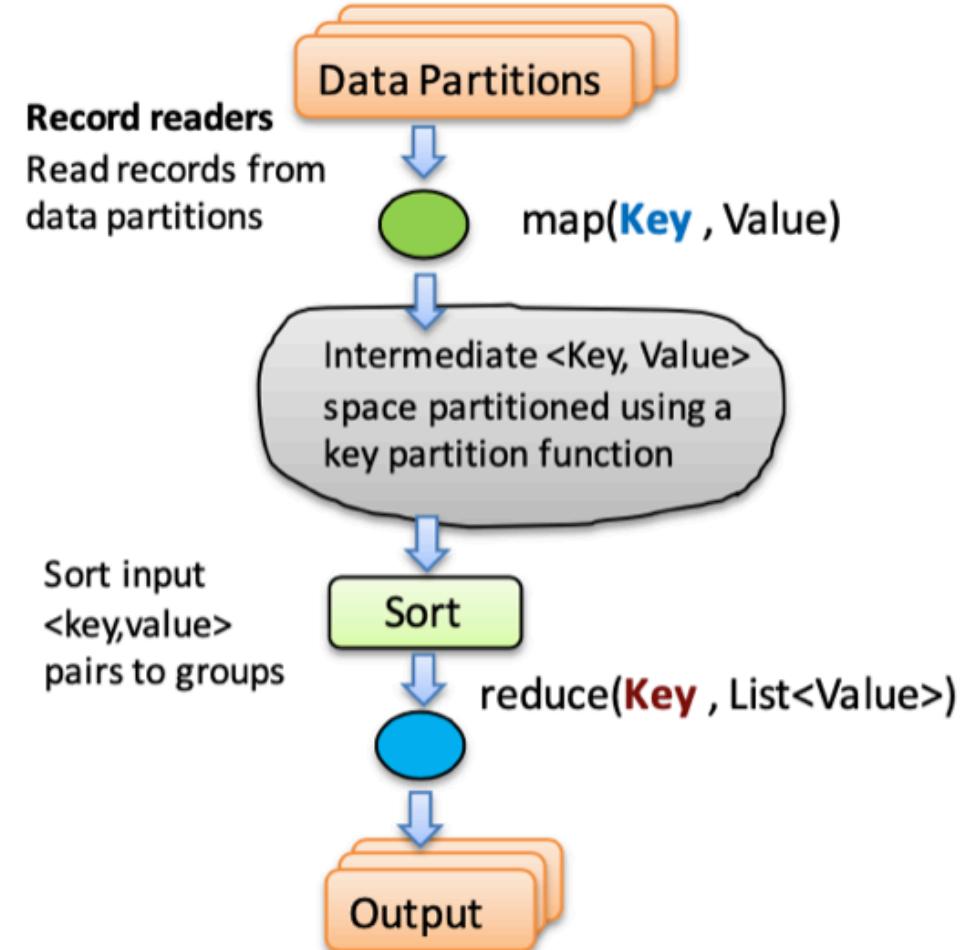
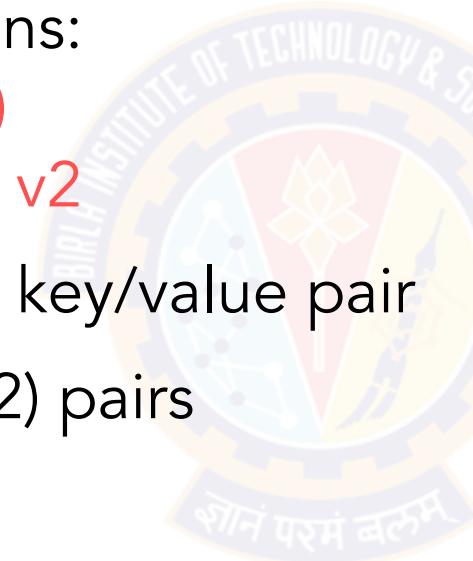
Output key-value pairs



(word, final-count)
~ SQL aggregation

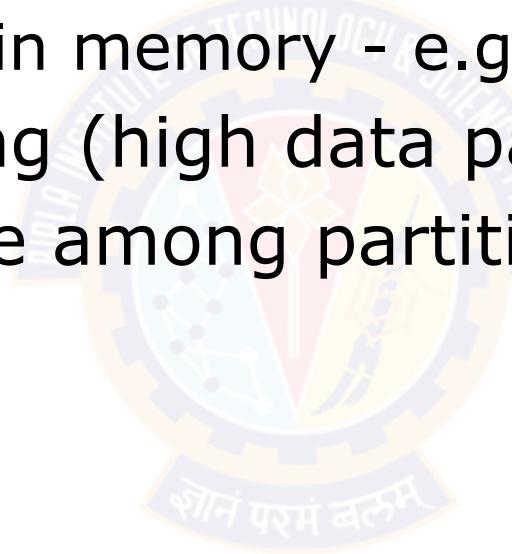
Formal definition of a MapReduce program

- Input: a set of key/value pairs
- User supplies two functions:
 - $\text{map}(k,v) \rightarrow \text{list}(k_1, v_1)$
 - $\text{reduce}(k_1, \text{list}(v_1)) \rightarrow v_2$
- (k_1, v_1) is an intermediate key/value pair
- Output is the set of (k_1, v_2) pairs



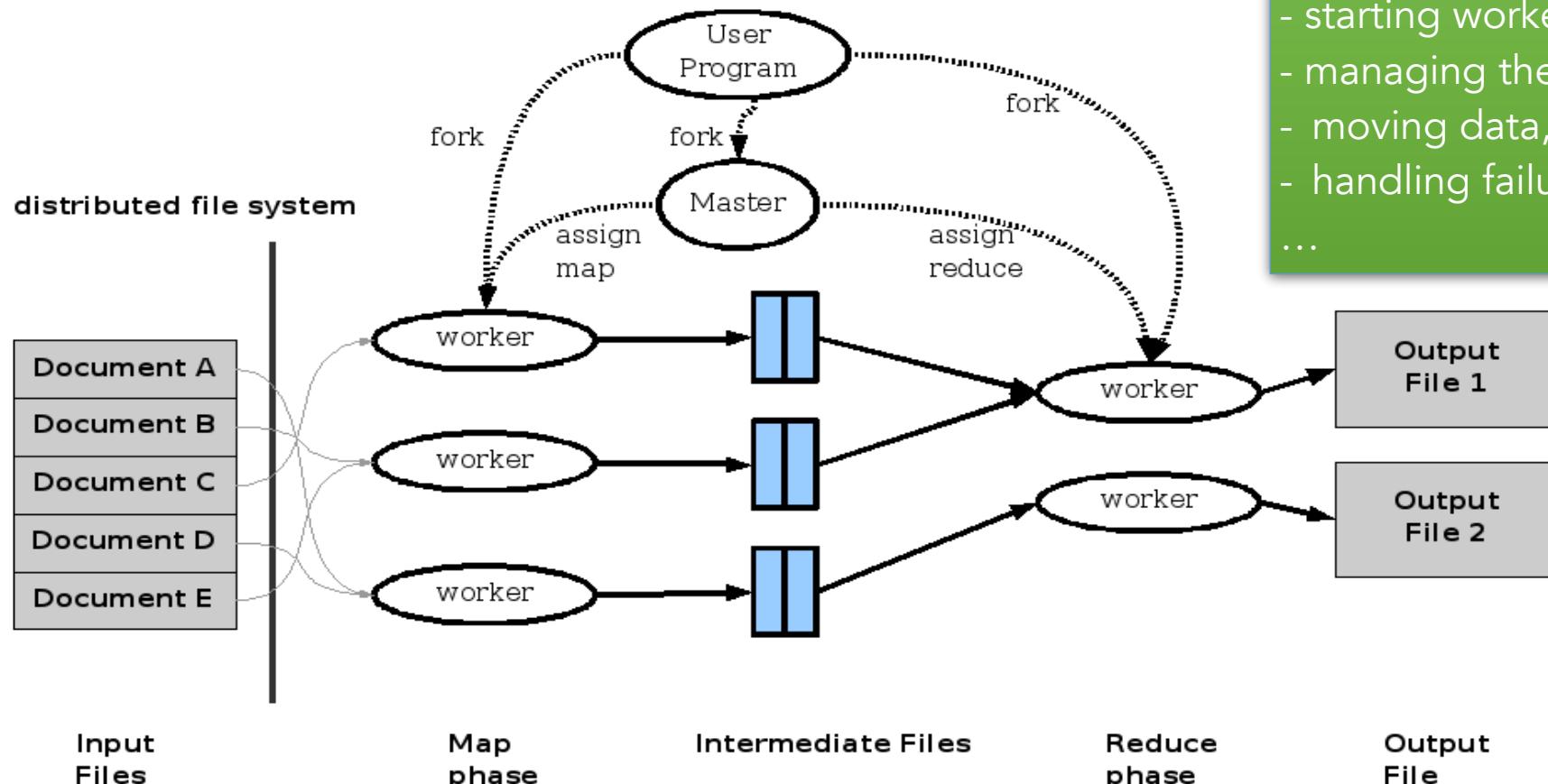
When will you use this ?

- Huge set of documents that don't fit into memory
 - So need file based processing in stages, e.g. Hadoop
 - But can also do this in memory - e.g. Spark
- Lot of data partitioning (high data parallelism)
- Possibly simple merge among partitions (low cost inverse tree parallelism)



MapReduce: Execution overview

- Data centric design
- Move computation closer to data
- Intermediate results on disk
- Dynamic task scheduling



A MapReduce library and runtime does all the work for

- allocating resources,
- starting workers,
- managing them,
- moving data,
- handling failures
- ...

MapReduce origins

Created in Google on GFS

Open source version created as Apache Hadoop

Perform maps/reduces on data using many machines

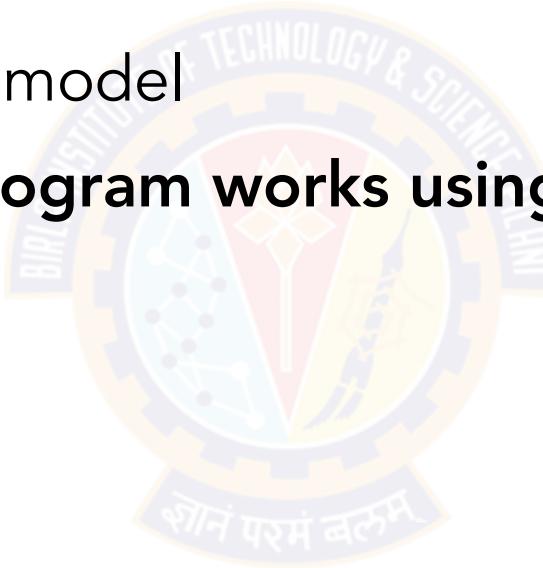
- The system takes care of distributing the data and managing fault tolerance
- You just write code to map one element and reduce elements to a combined result

Separates how to do recursive divide-and-conquer from what computation to perform

- Old idea in higher-order functional programming transferred to large-scale distributed computing
- Complementary approach to database declarative queries
 - In SQL you don't actually write the low level query execution code
- Programmer needs to focus just on map and reduce logic and rest of the work is done by the map-reduce framework.
 - So **restricted programming interface** to the system to let the system do the distribution of work, job tracking, fault tolerance etc.

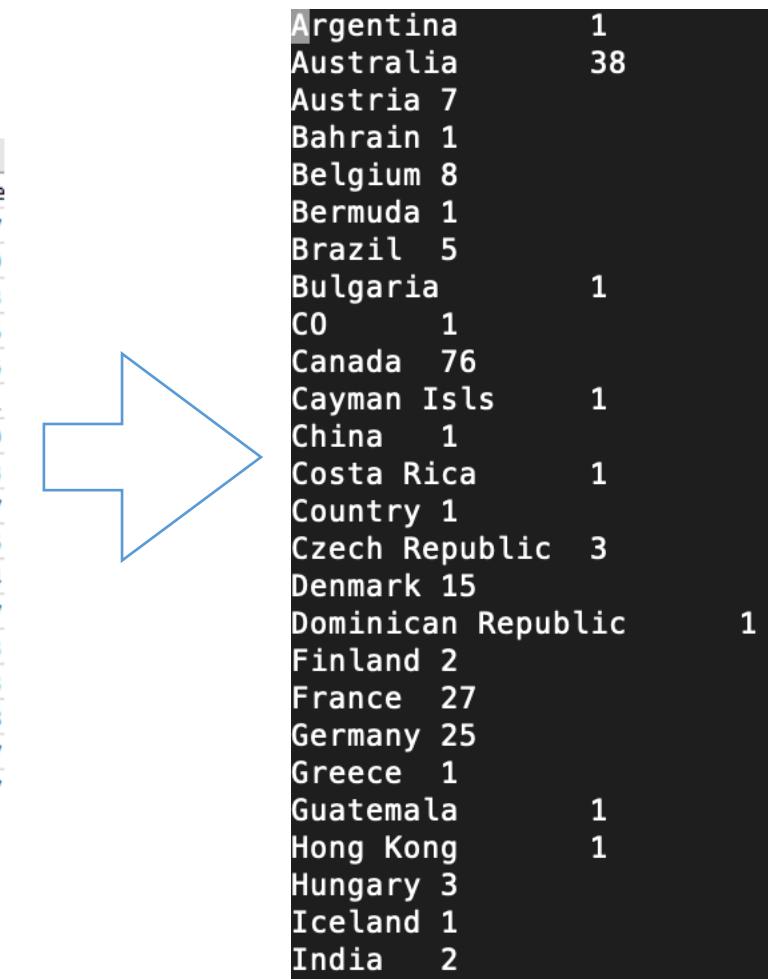
Topics for today

- Top down design
- Types of parallelism
- MapReduce programming model
- **See how a map reduce program works using Hadoop**
- Iterative MapReduce



More complex example - sales data processing

	A	B	C	D	E	F	G	H	I	J	K	L	
1	Transaction_date	Product	Price	Payment_Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude		
2	01-02-2009 06:17	Product1	1200	Mastercard	Carolina	Basildon	England	United Kingdom	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667	
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United States	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819	
4	01-02-2009 13:08	Product1	1200	Mastercard	Federica	Astoria	OR	United States	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83	
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75	
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W.	Cahaba	HeAL	United States	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025	
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United States	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381	
8	01-04-2009 20:11	Product1	1200	Mastercard	Fleur	Peoria	IL	United States	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889	
9	01-02-2009 20:09	Product1	1200	Mastercard	adam	Martin	TN	United States	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503	
10	01-04-2009 13:17	Product1	1200	Mastercard	Renee	Eli	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-France	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15	
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United States	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064	
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhoven	Noord-Brabant	Netherlands	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667	
14	01-02-2009 09:16	Product1	1200	Mastercard	Sean	Shavano	PTX	United States	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933	
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United States	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353	
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United States	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578	
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917	
18	01-05-2009 11:27	Product1	1200	Visa	Isaac	Ottawa	Ontario	Canada	01-05-2009 00:25	01-05-2009 10:24	45.41557	76.7	



count tx by country

<https://www.guru99.com/create-your-first-hadoop-program.html>

Running and checking status

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# ls -l
total 148
-rw-r--r--. 1 root root 44 Apr 18 01:11 Manifest.txt
-rw-r--r--. 1 root root 2966 Apr 18 01:12 ProductSalePerCountry.jar
drwxr-xr-x. 2 root root 96 Apr 19 00:57 SalesCountry
-rw-r--r--. 1 root root 1529 Apr 18 00:57 SalesCountryDriver.java
-rw-r--r--. 1 root root 746 Apr 18 00:56 SalesCountryReducer.java
-rw-r--r--. 1 root root 123637 Jun 6 16:56 SalesJan2009.csv
-rw-r--r--. 1 root root 661 Apr 18 00:56 SalesMapper.java
drwxr-xr-x. 3 root root 157 Jun 6 16:53 units
-rw-r--r--. 1 root root 219 Apr 18 21:10 units.csv
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# hadoop jar ProductSalePerCountry.jar /SalesJan2009.csv /output
21/06/06 17:27:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/06/06 17:27:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute remedy this.
21/06/06 17:27:40 INFO mapred.FileInputFormat: Total input files to process : 1
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: number of splits:2
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1622978356181_0002
21/06/06 17:27:41 INFO conf.Configuration: resource-types.xml not found
21/06/06 17:27:41 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/06/06 17:27:41 INFO impl.YarnClientImpl: Submitted application application_1622978356181_0002
21/06/06 17:27:41 INFO mapreduce.Job: The url to track the job: http://centos-s-4vcpu-8gb-blr1-01:8088/proxy/application_1622978356181_0002/
21/06/06 17:27:41 INFO mapreduce.Job: Running job: job_1622978356181_0002
21/06/06 17:27:49 INFO mapreduce.Job: Job job_1622978356181_0002 running in uber mode : false
21/06/06 17:27:49 INFO mapreduce.Job: map 0% reduce 0%
21/06/06 17:28:03 INFO mapreduce.Job: map 100% reduce 0%
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# jps
7744 SecondaryNameNode
8357 ResourceManager
8581 NodeManager
13541 Jps
7238 DataNode
10428 MRAppMaster
6910 NameNode
```



MapReduce Application application_1622978356181_0002

Logged in as: dr.who

Active Jobs										
Show 20 entries Search:										
Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed		
job_1622978356181_0002	SalePerCountry	RUNNING	2	2		1		0		

Showing 1 to 1 of 1 entries First Previous 1 Next Last

MapReduce stats

File System Counters

```
FILE: Number of bytes read=17747
FILE: Number of bytes written=660936
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=127535
HDFS: Number of bytes written=661
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
```

Job Counters

```
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=14658
Total time spent by all reduces in occupied slots (ms)=7011
Total time spent by all map tasks (ms)=14658
Total time spent by all reduce tasks (ms)=7011
Total vcore-milliseconds taken by all map tasks=14658
Total vcore-milliseconds taken by all reduce tasks=7011
Total megabyte-milliseconds taken by all map tasks=15009792
Total megabyte-milliseconds taken by all reduce tasks=7179264
```

Map-Reduce Framework

```
Map input records=999
Map output records=999
Map output bytes=15743
Map output materialized bytes=17753
Input split bytes=180
Combine input records=0
Combine output records=0
Reduce input groups=58
Reduce shuffle bytes=17753
Reduce input records=999
Reduce output records=58
Spilled Records=1998
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=585
CPU time spent (ms)=2510
Physical memory (bytes) snapshot=772923392
Virtual memory (bytes) snapshot=6393163776
Total committed heap usage (bytes)=527433728
```

Shuffle Errors

```
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
```

File Input Format Counters

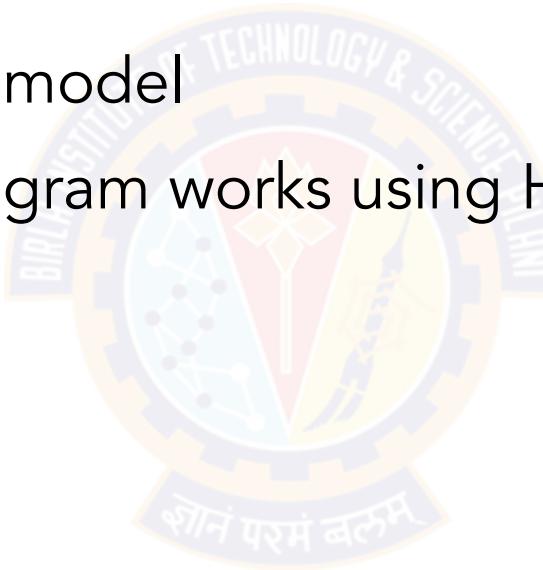
```
Bytes Read=127355
```

File Output Format Counters

```
Bytes Written=661
```

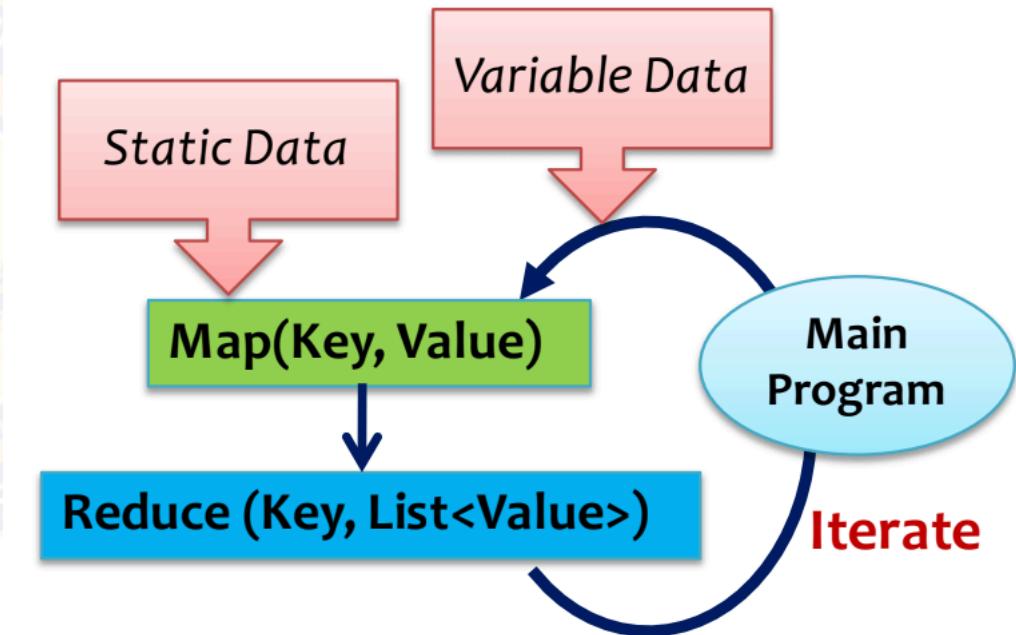
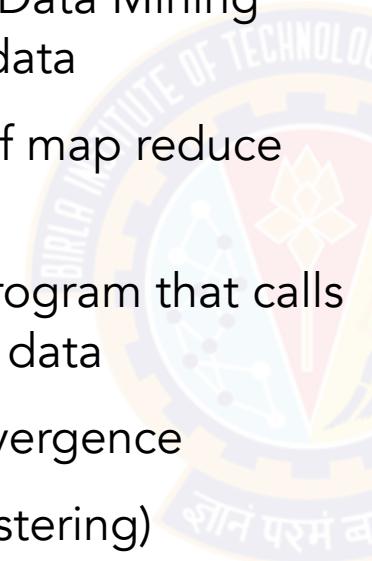
Topics for today

- Top down design
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works using Hadoop
- **Iterative MapReduce**

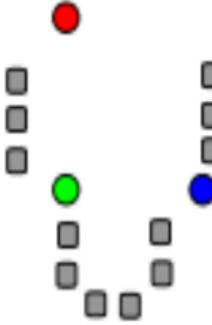


Iterative map reduce

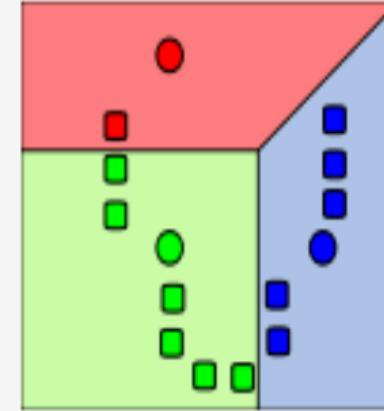
- MapReduce is a one-pass computation
- Many applications, esp in ML and Data Mining areas, need to iteratively process data
- So they need iterative execution of map reduce jobs
- An approach is to create a main program that calls the core map reduce with variable data
- Core program also checks for convergence
 - error bound (e.g. k-means clustering)
 - fixed iterations



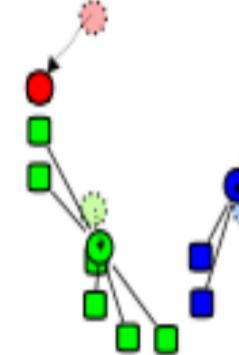
Example 1: K-means clustering



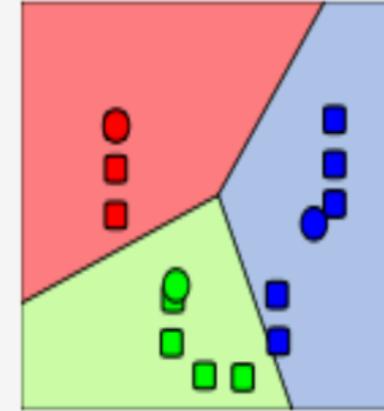
1) k initial "means" (in this case $k=3$) are randomly selected from the data set (shown in color).



2) k clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

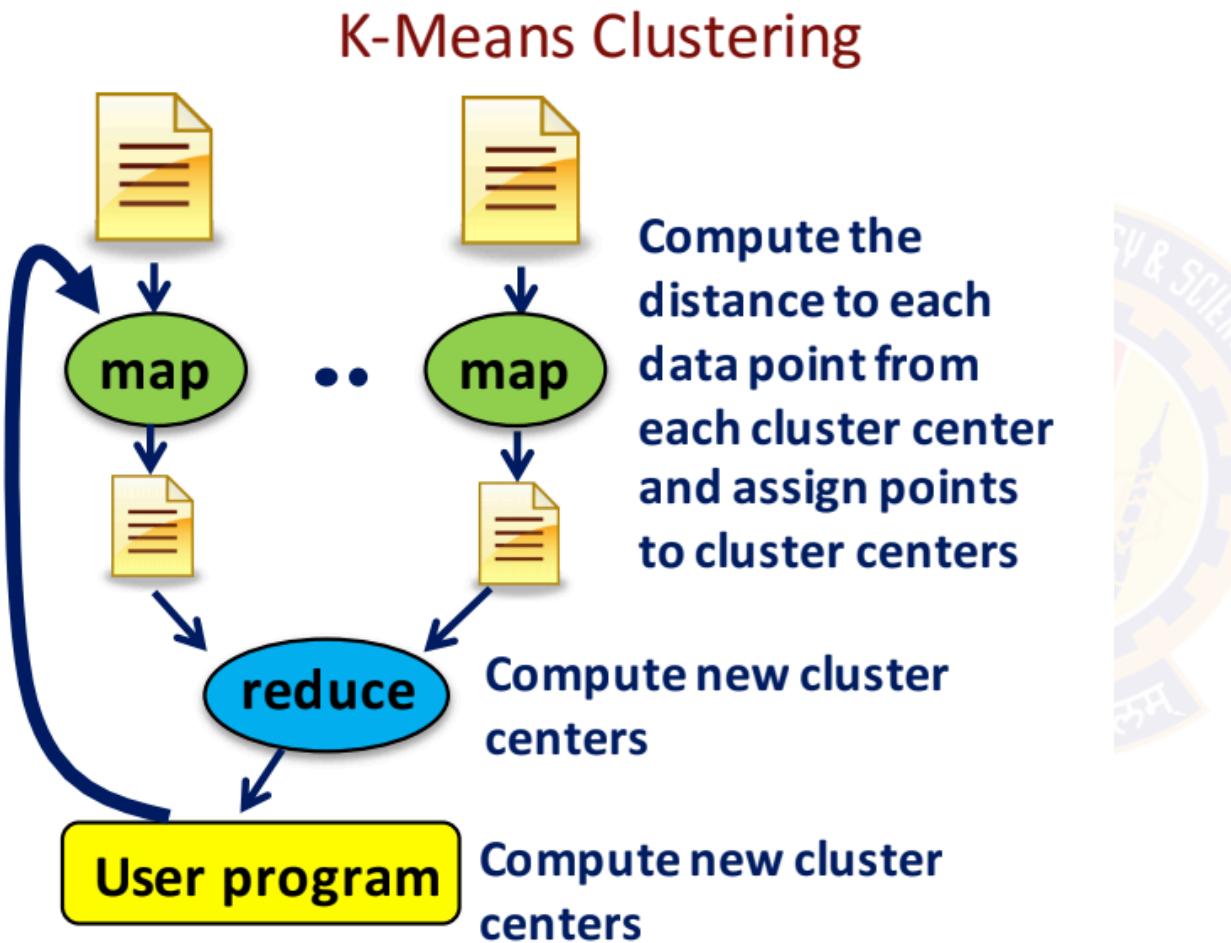


3) The [centroid](#) of each of the k clusters becomes the new means.



4) Steps 2 and 3 are repeated until convergence has been reached.

K-means as iterative map reduce



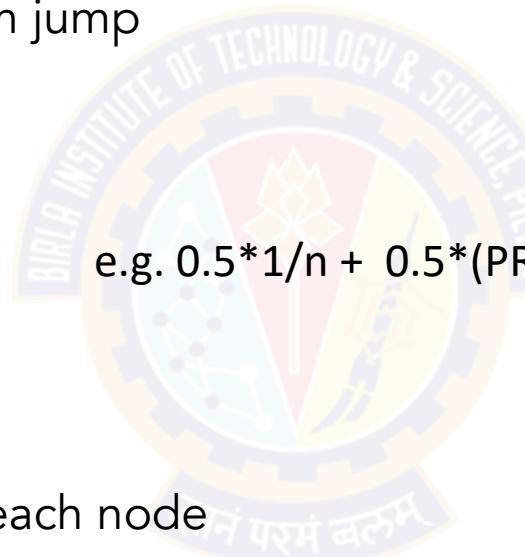
- The MapReduce program driver is responsible for repeating the steps via an iterative construct.
 - Within each iteration map and reduce steps are called.
 - Each map step reuses the result produced in previous reduce step.
 - e.g. k centers computed

<https://github.com/thomasjungblut/mapreduce-kmeans/tree/master/src/de/jungblut/clustering/mapreduce>

Example 2: PageRank

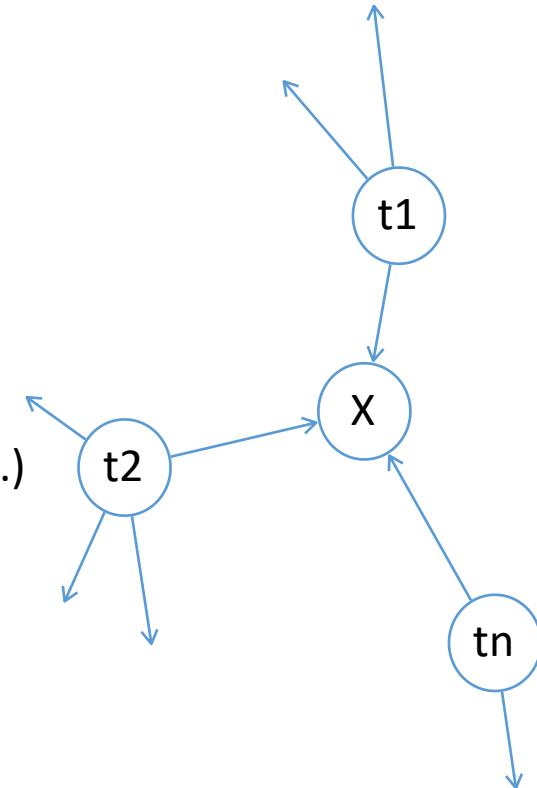
- Given a page X with n in-bound links t_1, t_2, \dots, t_n
 - $C(t_i)$ is out-degree of t_i
 - α is the probability of a random jump
 - N is total number of nodes

$$PR(x) = \alpha \left(\frac{1}{N} \right) + (1 - \alpha) \sum_{i=1}^n \frac{PR(t_i)}{C(t_i)}$$

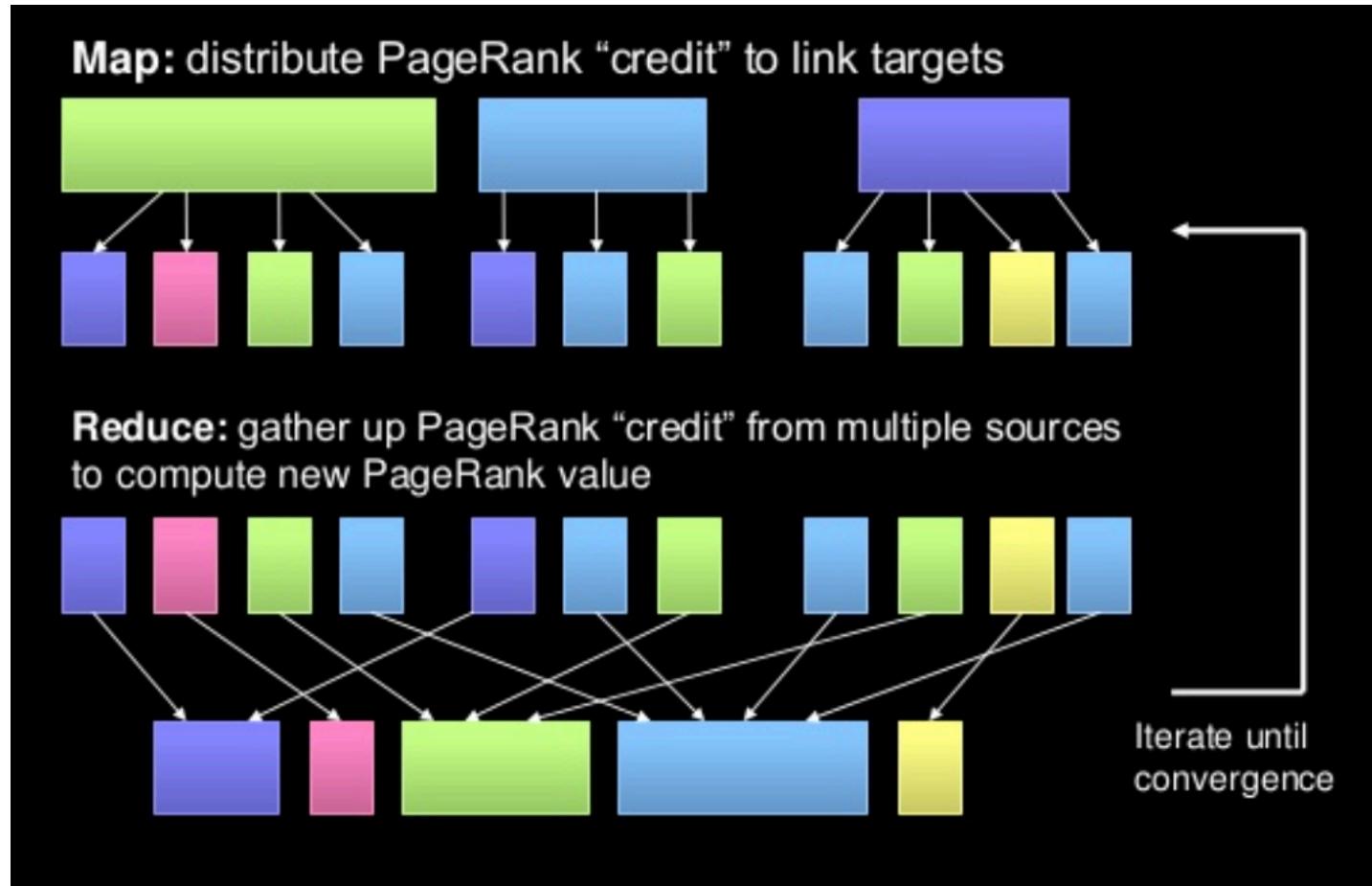


e.g. $0.5 * 1/n + 0.5 * (PR(t1)/3 + PR(t2)/4 + \dots)$

- Iterative process
 - Start with initial PR_i values for each node
 - Each page distributes its PR_i credits to all pages it links to (out-bound)
 - Each page adds-up all the credits it gets from in-bound links to compute PR_{i+1}
 - Iterate till values converge

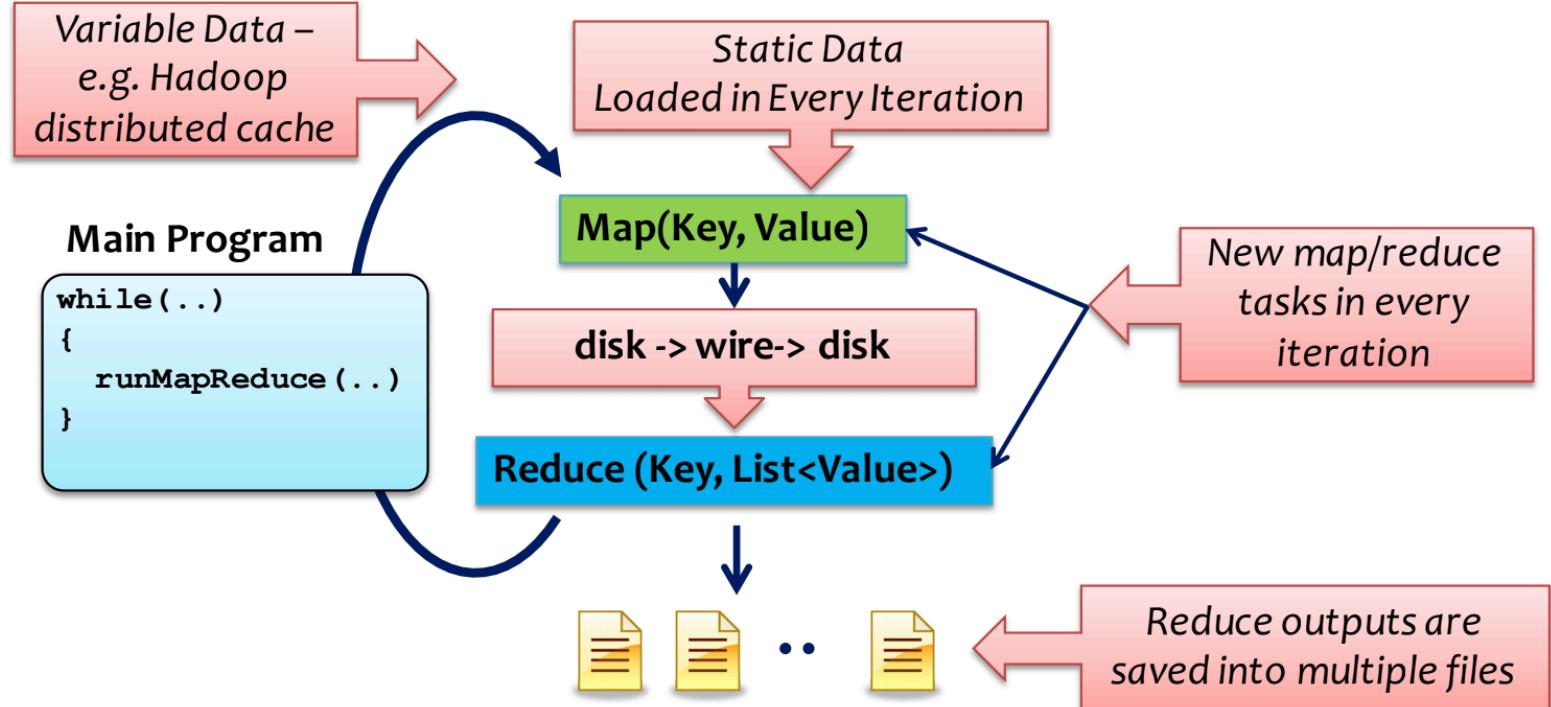


PageRank as Iterative MapReduce



Iterations using existing runtimes

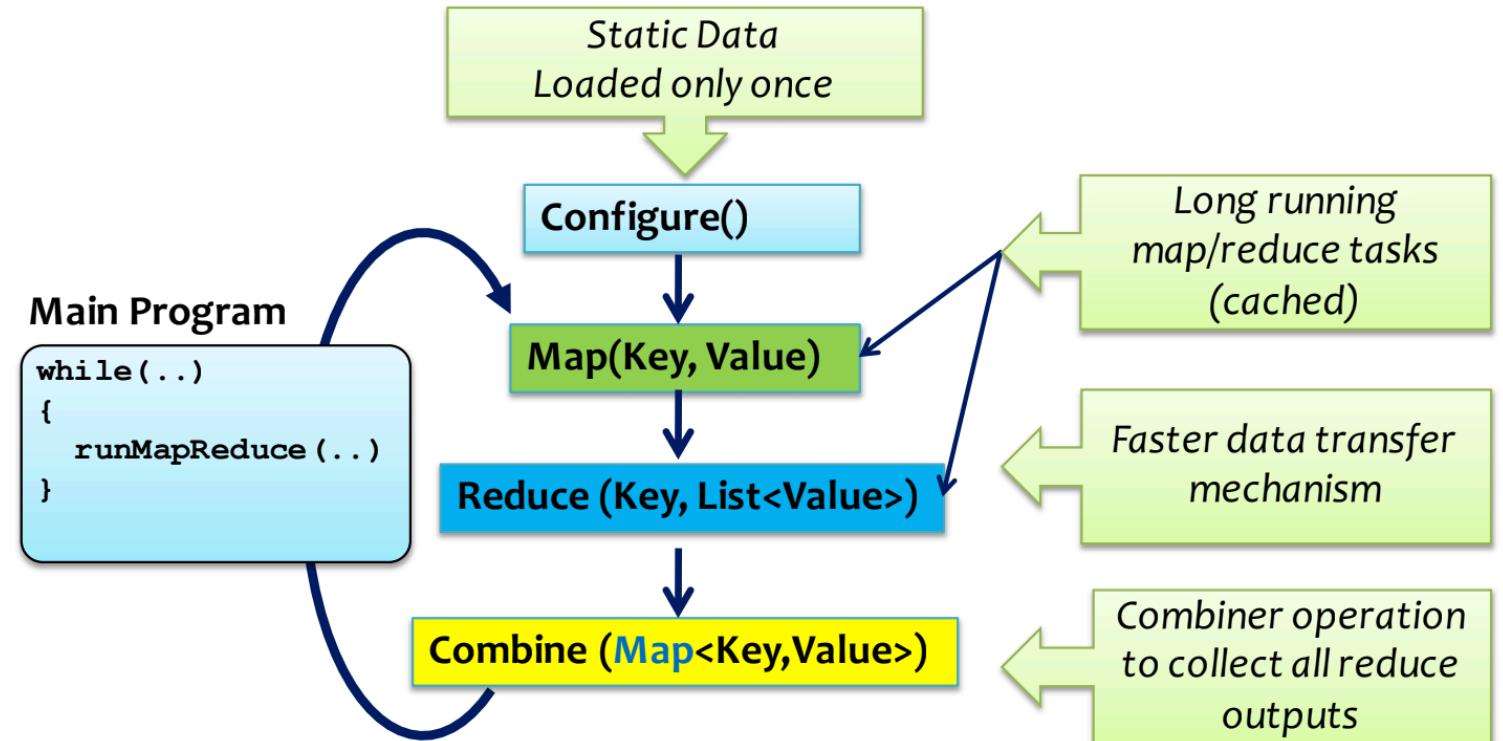
- Loop implemented on top of existing file-based single step map-reduce core
- Large overheads from
 - re-initialization of tasks
 - reloading of static data
 - communication and data transfers



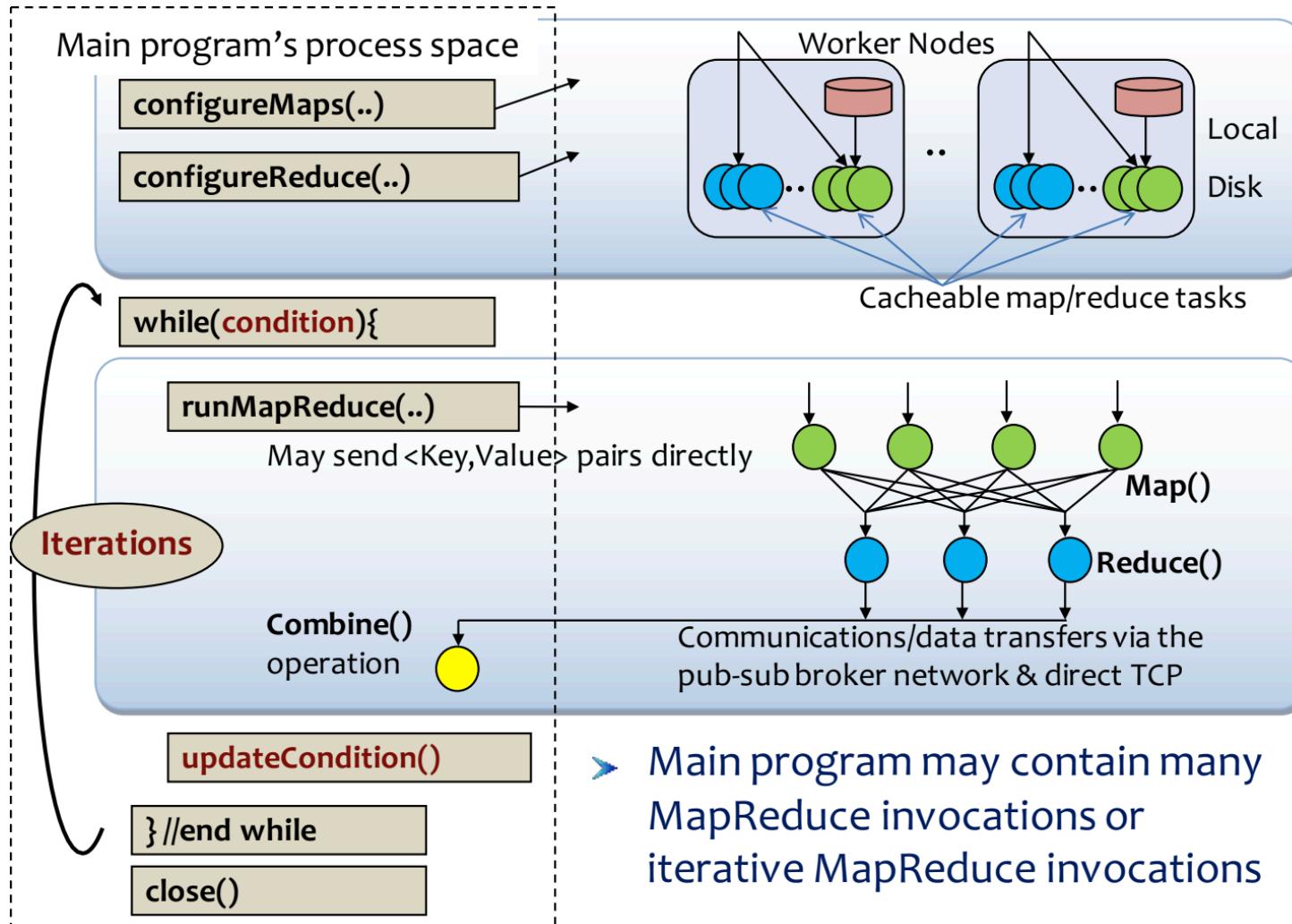
DistributedCache: <https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html>

MapReduce++ : Iterative MapReduce

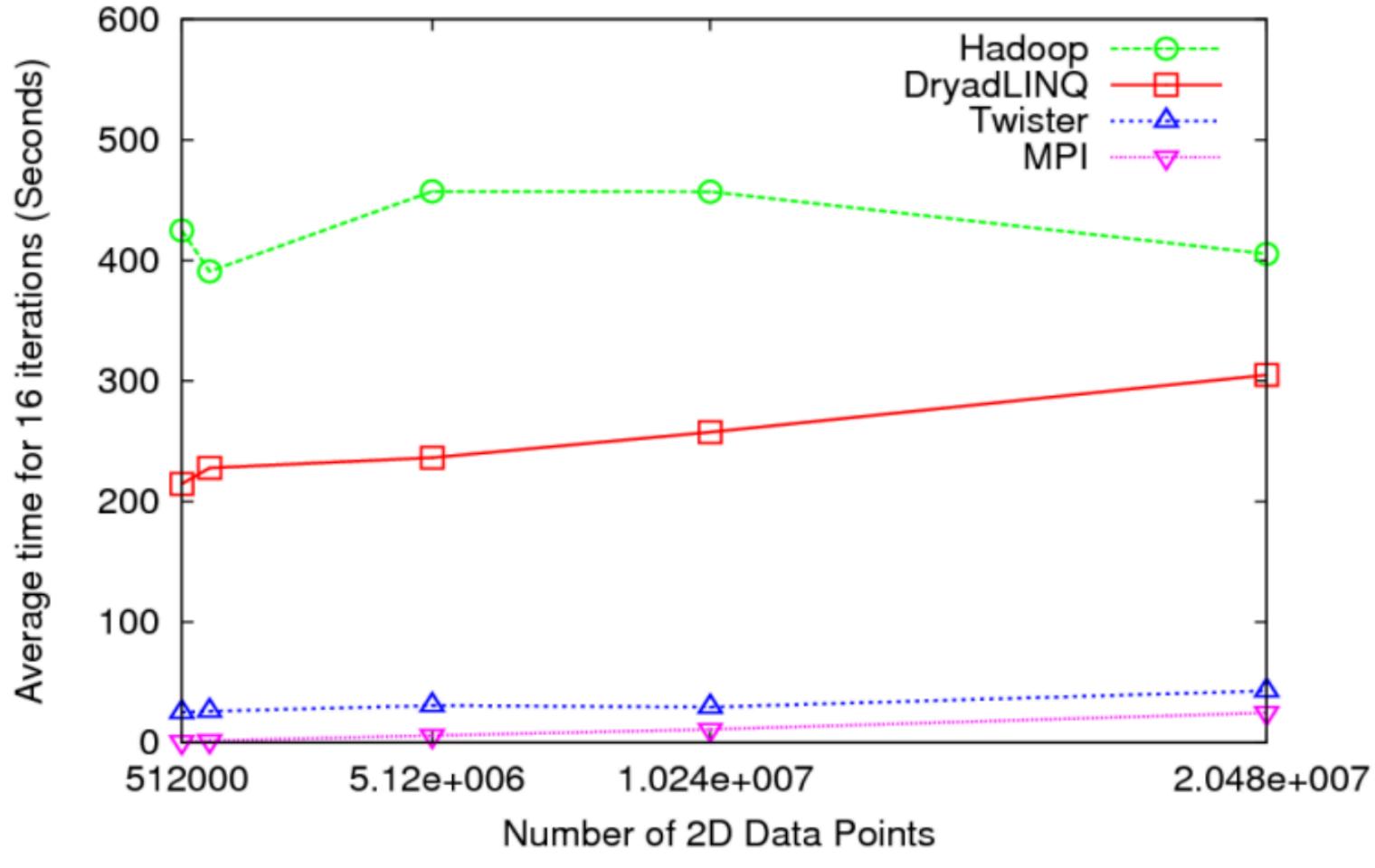
- Some optimizations are done on top of existing model
 - Static data loaded once
 - Cached tasks across invocations
 - Combine operations



Example in Twister: Enables more APIs



The optimisations indeed help



K-means clustering using various programming models

Iterative MapReduce: Other options

- **HaLoop**
 - Modifies Hadoop scheduling to make it loop aware
 - Implements caches to avoid going to disk between iterations
 - Optional reading: Paper in [Proceedings of the VLDB Endowment](#) 3(1):285-296, Sep 2010
- **Spark**
 - Uses in-memory computing to speed up iterations
 - An in-memory structure called RDD : Resilient Distributed Dataset replaces files on disk
 - Ideal for iterative computations that reuse lot of data in each iteration

Naive PageRank using Spark

```
// load data from file - can be in HDFS
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

// links is a pair of URLs
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();

// ranks stores the initial page ranks init to 1
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);

// Start loop to update URL ranks iteratively
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1()) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });
    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()
        .mapValues(sum -> 0.15 + sum * 0.85));
}

// end of iterations ranks contains the result to be collected
}
```

[Example code link](#)

1 2
1 3
1 4
4 1
3 1
2 1

links :

(1,2) (1,3) (1,4) (4,1) (3,1) (2,1)

ranks:

(1,1) (1,1) (1,1) (4,1) (3,1) (2,1)

1
2

urlCount: 4

First iteration **contribs:**

(1,3/4) (2,1/4) (3,1/4) (4,1/4)

4 recompute contribs

(1, 0.78)
(2, 0.36)
(3, 0.36)
(4, 0.36)

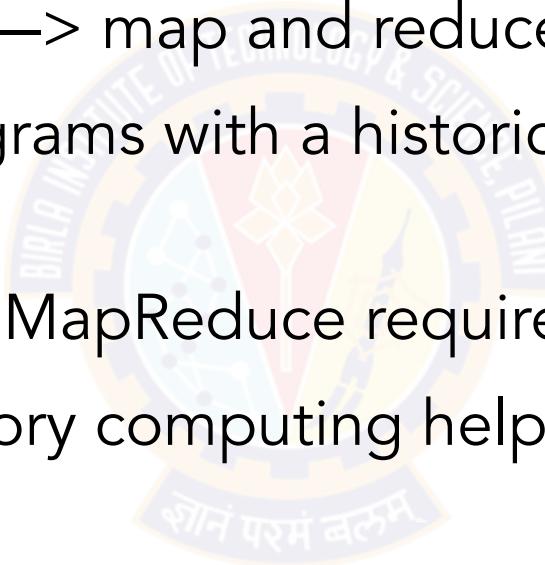
3 recompute ranks

First iteration **ranks:**

(1, 0.15 + 0.85 * 0.75)
(2, 0.15 + 0.85 * 0.25)
(3, 0.15 + 0.85 * 0.25)
(4, 0.15 + 0.85 * 0.25)

Summary

- Different types of parallelism
- Data and tree parallelism —> map and reduce
- Basics of MapReduce programs with a historical sales data processing example
- Optimizations for iterative MapReduce requirements
- How does Spark / in-memory computing help for iterative MapReduce programming





Next Session:

Hadoop Architecture and File System