Programming at Scale: Dataflow

cs378

Today

Questions?

Administrivia

- Project Proposal Due Soon!

Agenda:

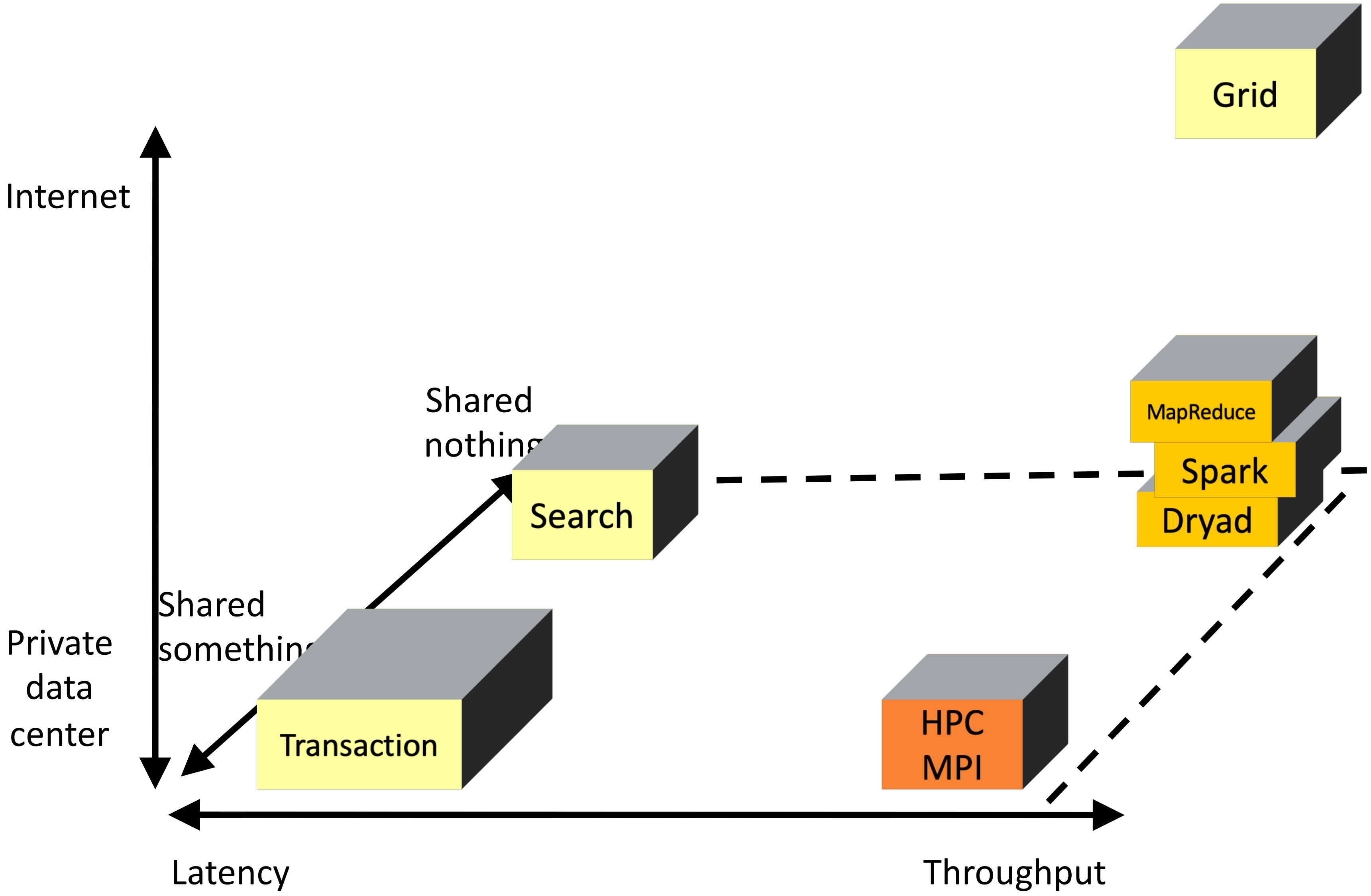
- MPI Wrapup
- Dataflow

MapReduce faux quiz (5 min, any 2):

- Have you ever written a MapReduce program? If so, what was it?
- What phenomena can slow down a map task?
- Do reducers wait for all their mappers before starting? Why/why not?
- What machine resources does the shuffle phase consume most?
- Is it safe to re-execute a failed map/reduce job sans cleanup? Why [not]?
- How does MR handle master failure? What are the alternatives?

Review: Scale: Goal

Review: Design Space



You are an engineer at: Hare-brained-scheme.com

Your boss,  comes to your office and says:

> *"We're going to be super rich! We just need a program to search for strings in text files..."*

Input: <search_term>, <files>

Output: list of files containing <search_term>

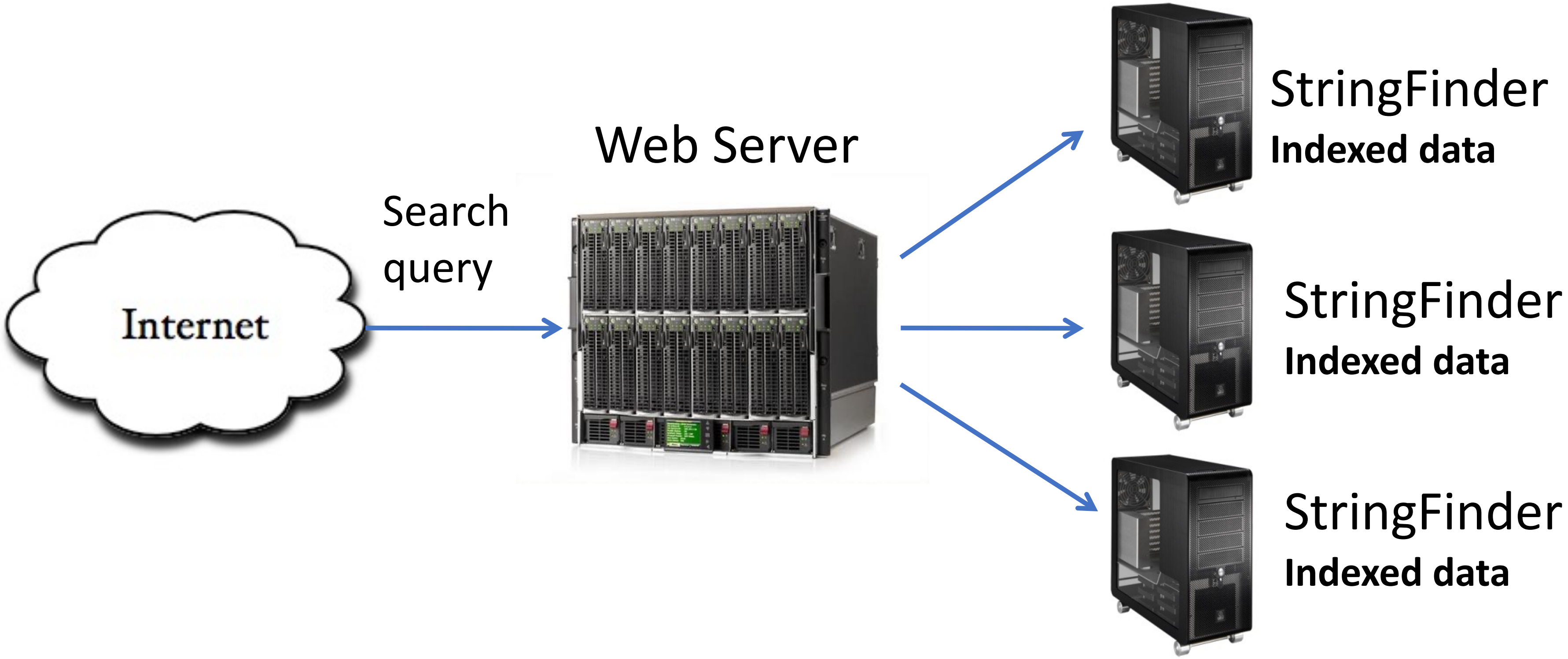
One Solution

```
public class StringFinder {
   int main(…) {
         foreach(File f in getInputFiles()) {
              if(f.contains(searchTerm))
                   results.add(f.getFileName());
         }
         System.out.println(“Files:” + results.toString());
    }
}
```

Another Solution

```
public class StringFinder {
    int main(…) {
            foreach(File F in getInputFiles()) {
                partitions = partitionFile(F, num_hosts)
                foreach(host h, partition f in partitions) {
                        h.send(f)
                        h.runAsync({
                                if(f.contains(searchTerm))
                                results.add(f.getFileName())});
            }}
            System.out.println(“Files:” + results.toString());
}}
```

Infrastructure is hard to get right



1. How do we distribute the searchable files on our machines?
2. What if our webserver goes down?
3. What if a StringFinder machine dies? How would you know it was dead?
4. **What if marketing comes and says, “well, we also want to show pictures of the earth from space too! Ooh..and the moon too!”**

StringFinder was the easy part!

You really need general infrastructure.

Many different tasks

Want to use hundreds or thousands of PC's

Continue to function if something breaks

Must be easy to program…

Dataflow Engines

Programming model + infrastructure

Write programs that run on lots of machines

Automatic parallelization and distribution

Fault-tolerance

I/O and jobs Scheduling

Status and monitoring

Key Ideas:

*All modern "big data" platforms are **dataflow engines!***

Differences:

1. what graph structures are allowed?
2. How does this impact programming model?

MapReduce

- Input & Output: sets of <key, value> pairs
- Programmer writes 2 functions:

```
map (in_key, in_value) -> list(out_key,
  intermediate_value)
```

  - Processes <k,v> pairs
  - Produces intermediate pairs

```
reduce (out_key, list(interm_val)) ->
    list(out_value)
```

  - Combines intermediate values for a key
  - Produces a merged set of outputs

Indexing (1)

```
public void map() {
  String line = value.toString();
  StringTokenizer itr = new StringTokenizer(line);
    if(itr.countTokens() >= N) {
      while(itr.hasMoreTokens()) {
        word = itr.nextToken()+"|"+key.getFileName();
              output.collect(word, 1);
    }
  }
}
```

Input: a line of text, e.g. "**mistakes were made**" from **myfile.txt**

Output:

mistakes|myfile.txt
were|myfile.txt
made|myfile.txt

Indexing (2)

```
public void reduce() {
    int sum = 0;
    while(values.hasNext()) {
        sum += values.next().get();
    }
    output.collect(key, sum);
}
```

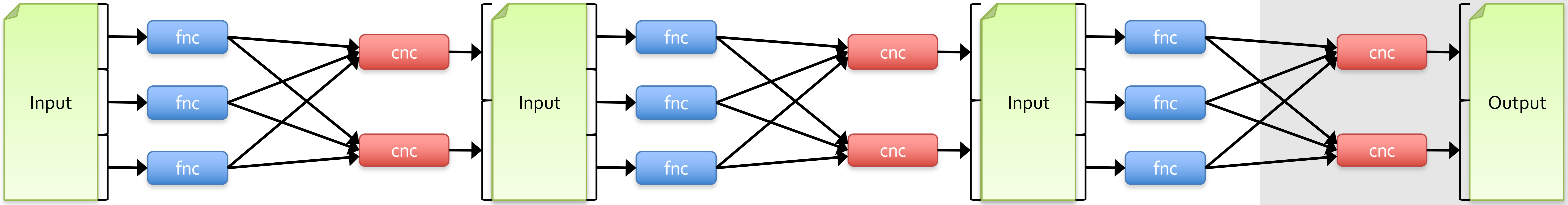
Input: a <term,filename> pair, list of occurrences (e.g. {1, 1,..1})

Output:

mistakes|myfile.txt	**10**
were|myfile.txt	**45**
made|myfile.txt	**2**

Review: K-Means

```
public void kmeans() {
  while(...) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```



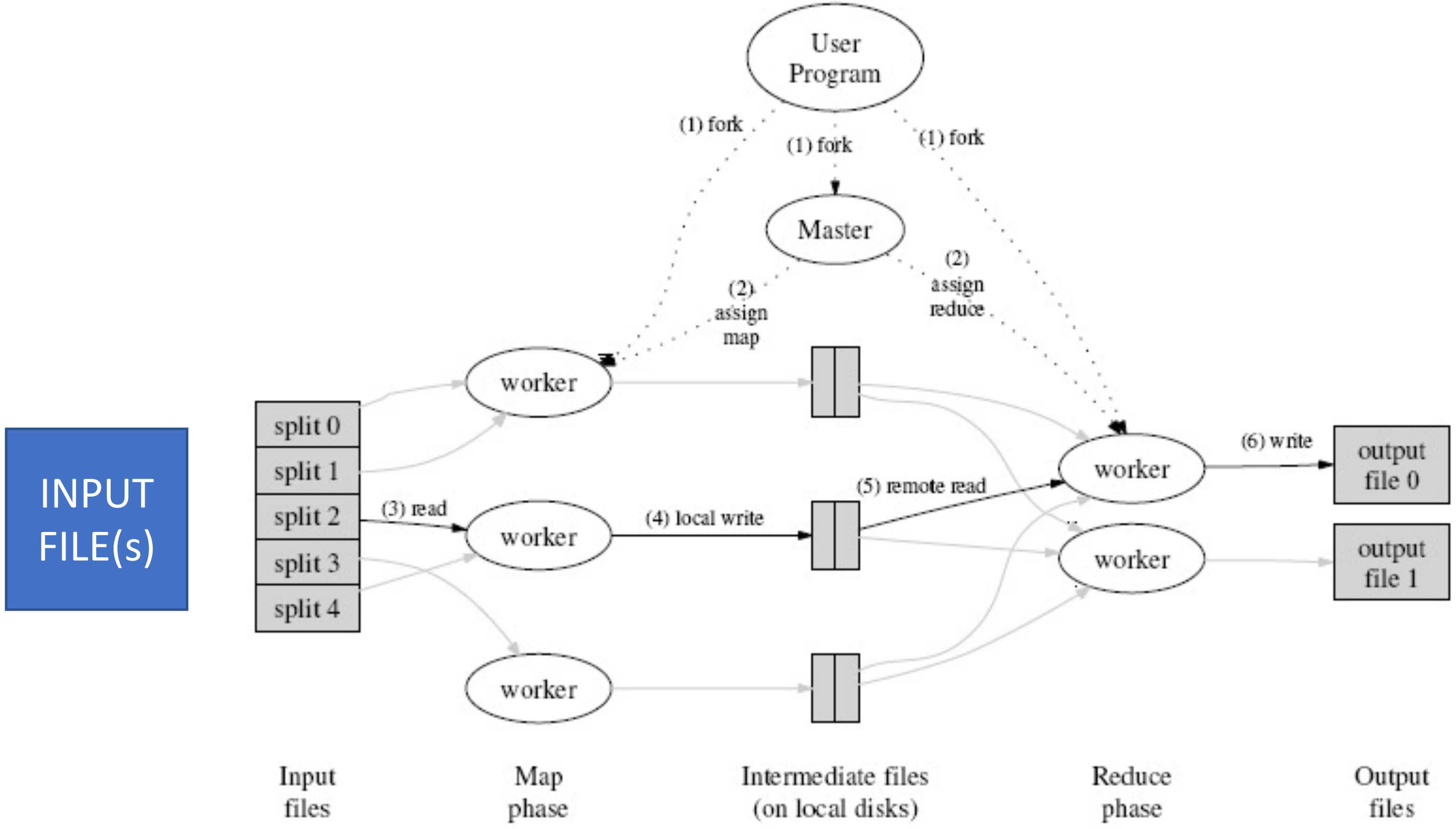
Example: K-Means Mapper

```
/*
 * Map: find minimum distance center for point, emit to reducer
 */
@Override
public void map(LongWritable key, Text value,
                OutputCollector<DoubleWritable, DoubleWritable> output,
                Reporter reporter) throws IOException {
    String line = value.toString();
    double point = Double.parseDouble(line);
    double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters.get(0);
    // Find the minimum center from a point
    for (double c : mCenters) {
        min1 = c - point;
        if (Math.abs(min1) < Math.abs(min2)) {
            nearest_center = c;
            min2 = min1;
        }
    }
    // Emit the nearest center and the point
    output.collect(new DoubleWritable(nearest_center),
            new DoubleWritable(point));
}
```

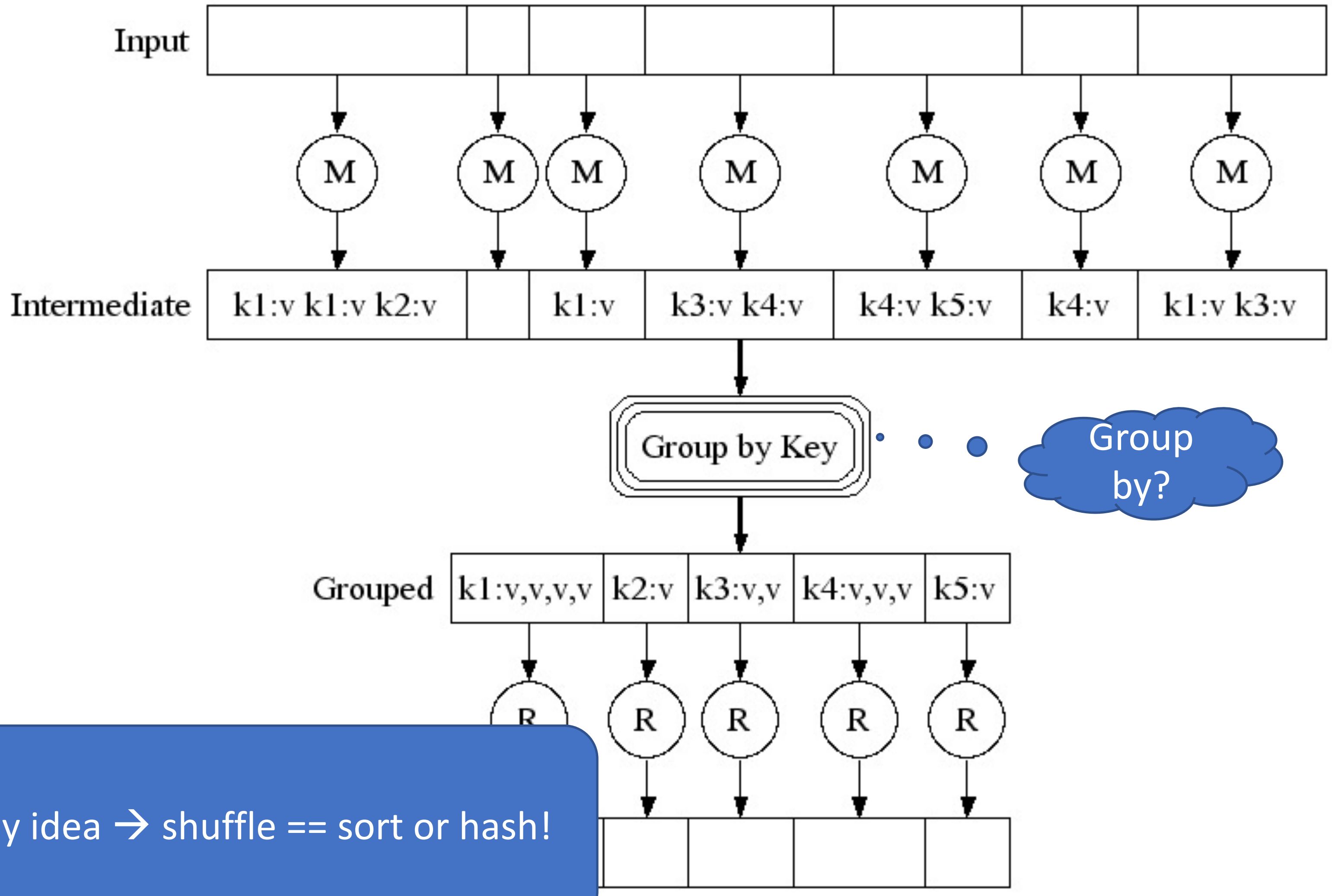
Example: K-Means Reducer

```java
/*
 * Reduce: collect all points per center and calculate
 * the next center for those points
 */
@Override
public void reduce(
        DoubleWritable key, Iterator<DoubleWritable> values,
        OutputCollector<DoubleWritable, Text> output, Reporter reporter)
        throws IOException {
    double newCenter;
    double sum = 0;
    int no_elements = 0;
    String points = "";
    while (values.hasNext()) {
        double d = values.next().get();
        points = points + " " + Double.toString(d);
        sum = sum + d;
        ++no_elements;
    }

    // We have a new center now
    newCenter = sum / no_elements;

    // Emit new center and point
    output.collect(new DoubleWritable(newCenter), new Text(points));
}
```

How Does Parallelization Work?



Execution



Group by?

Key idea → shuffle == sort or hash!

Task Granularity And Pipelining

|map tasks| >> |machines| -- why?

Minimize fault recovery time

Pipeline map with other tasks

Easier to load balance dynamically

The end of your career at: Hare-brained-scheme.com

Your boss,  comes to your office and says:

"I can't believe you used ***MapReduce!!!***

You're fired..."

Why might he say this?

MapReduce: not without Controversy

MapReduce: A major step backwards | The Database Column

http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/ September 6, 2011

on Jan 17 in Database architecture, Database history, Database innovation posted by DeWitt

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called "cloud computing." This paradigm entails harnessing large numbers of (low-end) processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of "jelly beans" rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

Why is MapReduce backwards?

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

Not novel: 25 year-old ideas from DBMS lit

- It’s just a group-by aggregate engine

Missing most DBMS features

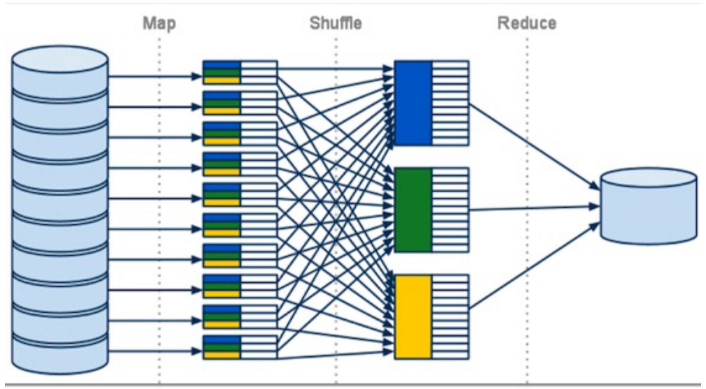
- Schema, foreign keys, …

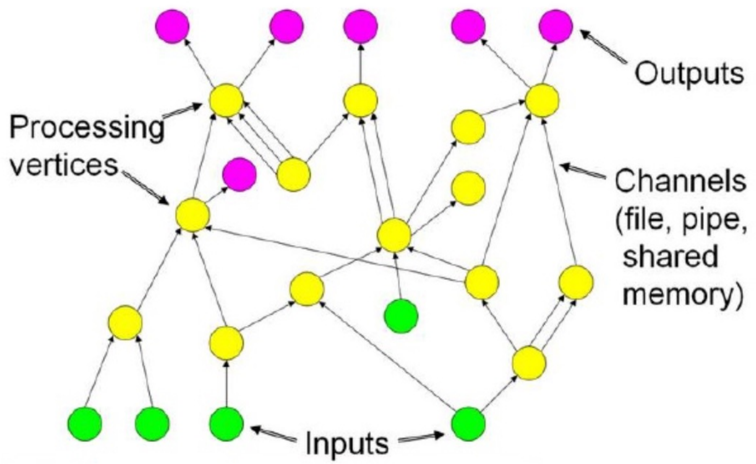
Incompatible with most DBMS tools

So why is it such a big success?

MapReduce and Dataflow

- MR is a ***dataflow*** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark





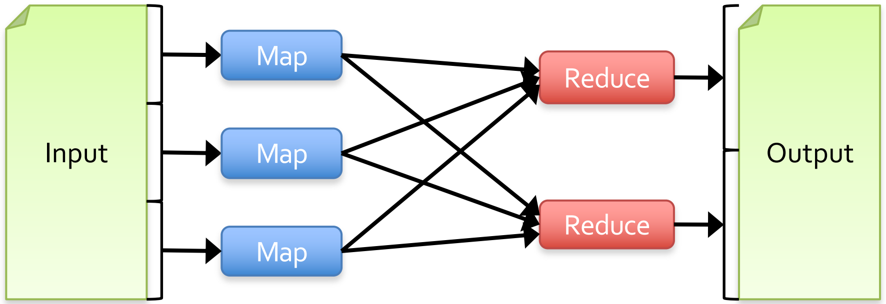
MapReduce vs Dryad (and others…)

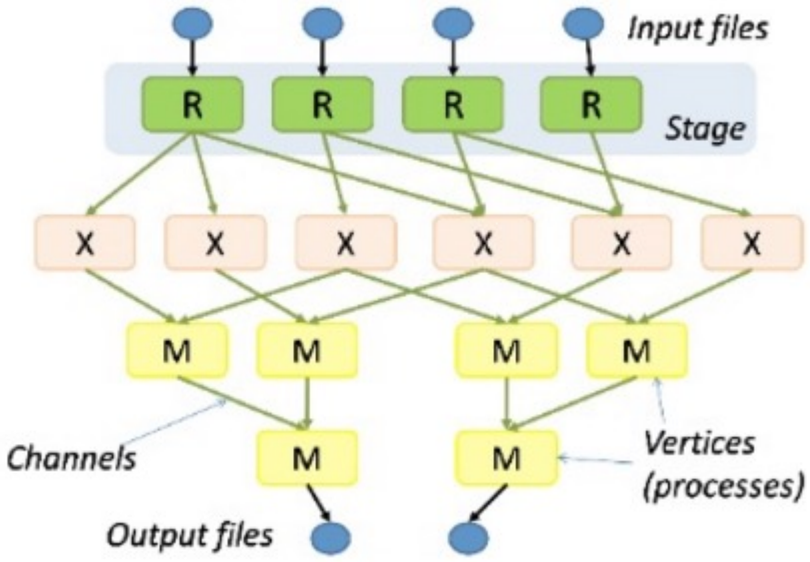
DAG instead of BSP

Interface variety

- Memory FIFO
- Disk
- Network

Flexible Modular Composition

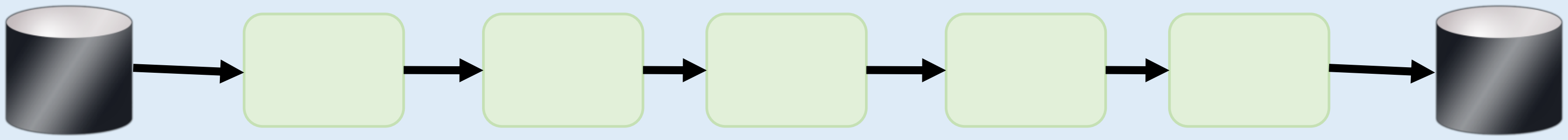




Dryad (2007): 2-D Piping

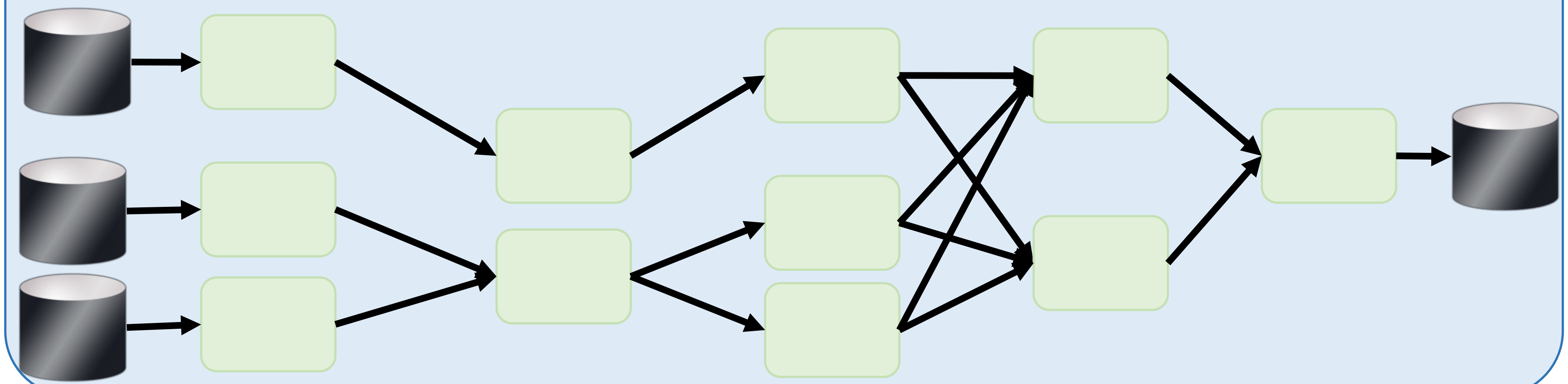
- Unix Pipes: 1-D

grep | sed | sort | awk | perl

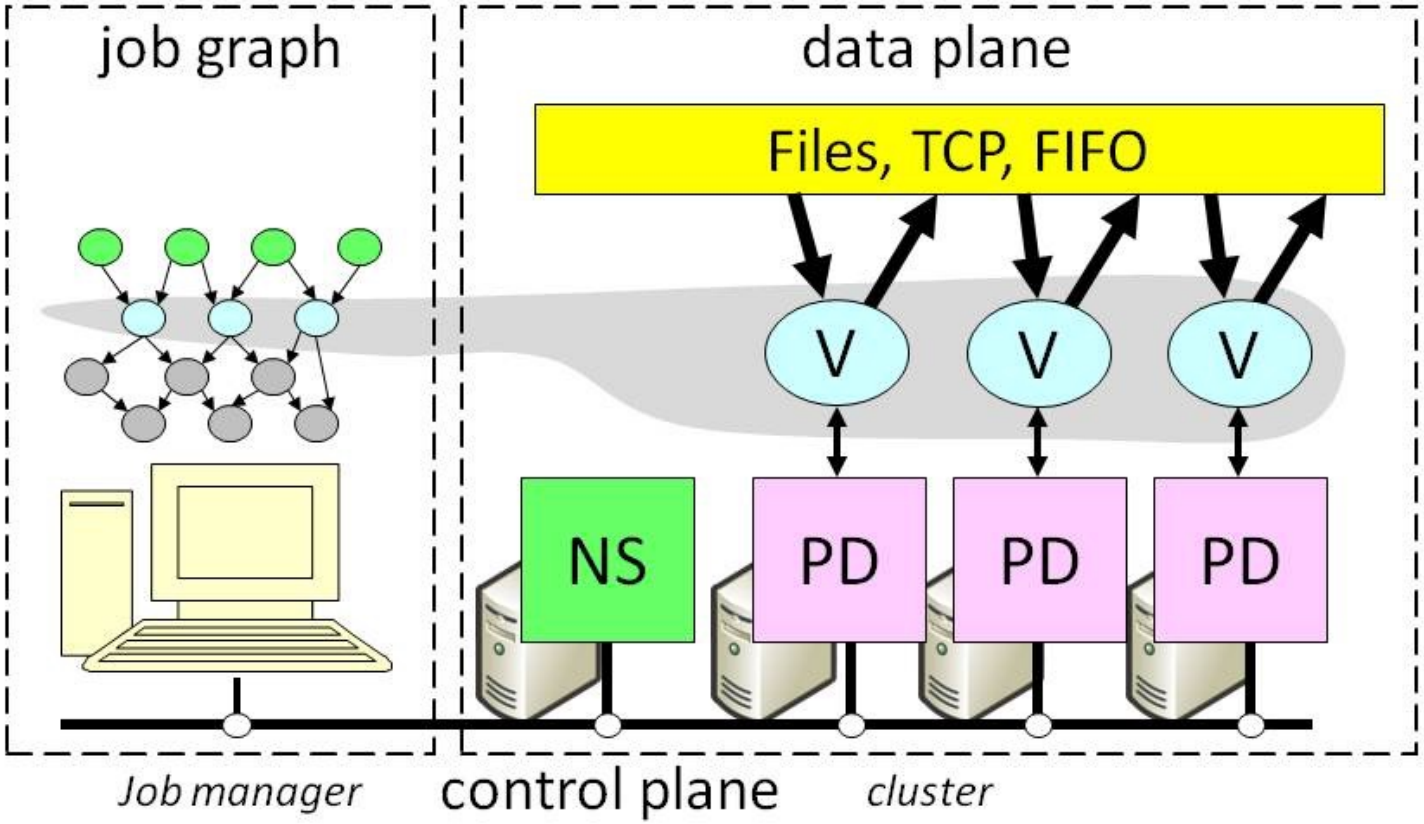


- Dryad: 2-D

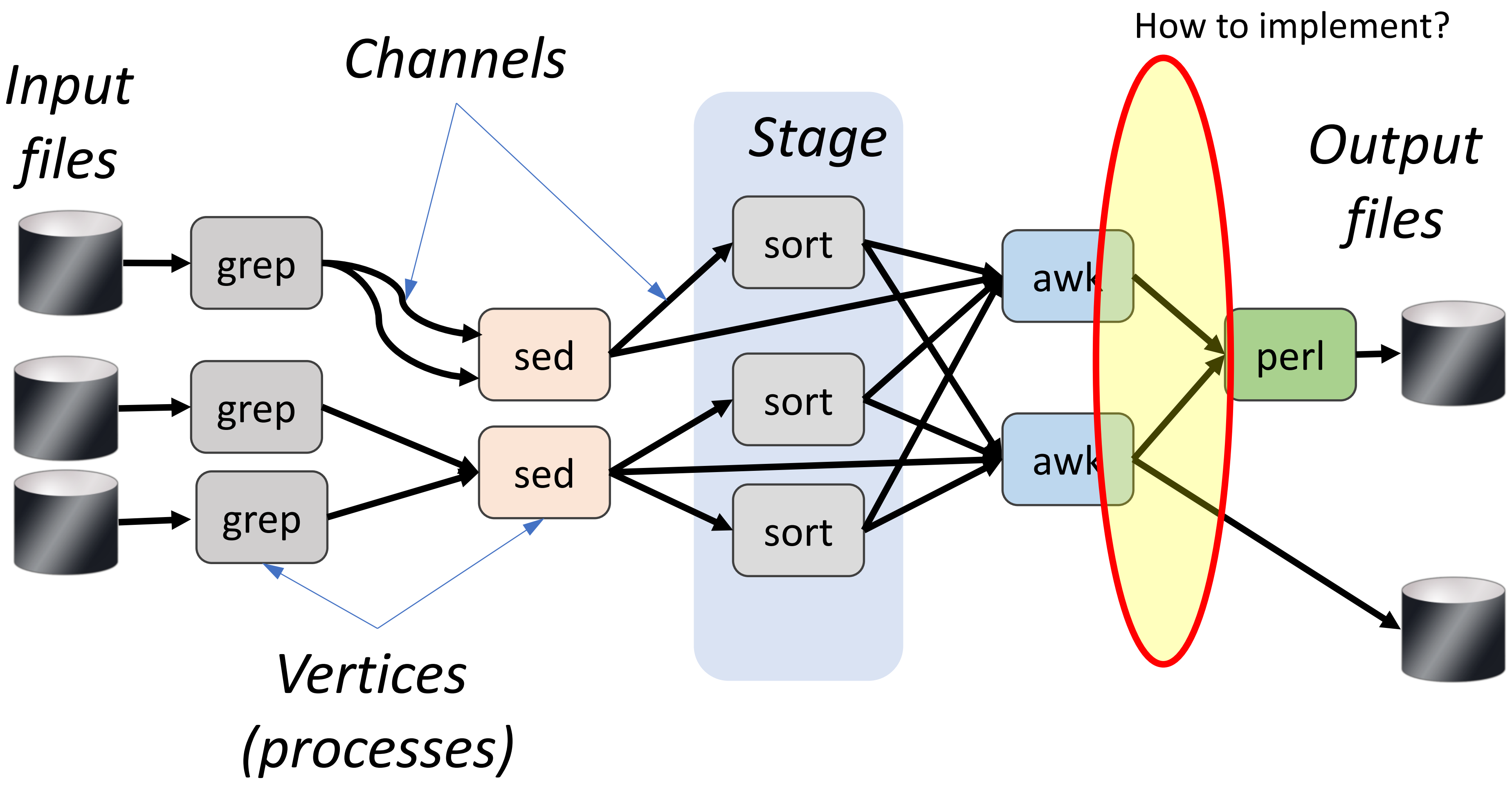
grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}



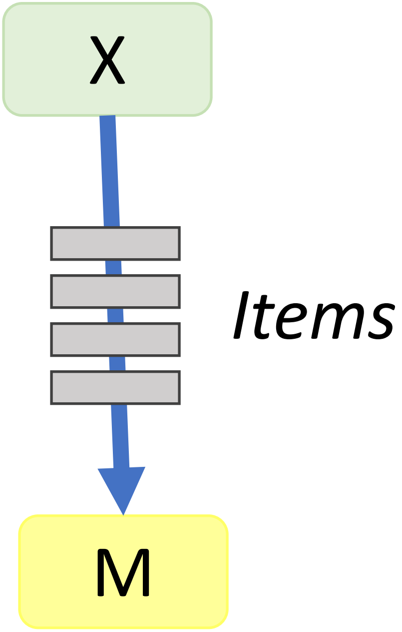
Dataflow Engines



Dataflow Job Structure



Channels



Key idea:
Encapsulate data movement behind channel abstraction → gets programmer out of the picture

Finite streams of items

- distributed filesystem files
  (persistent)
- SMB/NTFS files
  (temporary)
- TCP pipes
  (inter-machine)
- memory FIFOs
  (intra-machine)

Spark (2012) Background

Commodity clusters: important platform

- **In industry:** search, machine translation, ad targeting, …
- **In research:** bioinformatics, NLP, climate simulation, …

Cluster-scale models (e.g. MR) de facto standard

- Fault tolerance through replicated durable storage
- Dataflow is the common theme

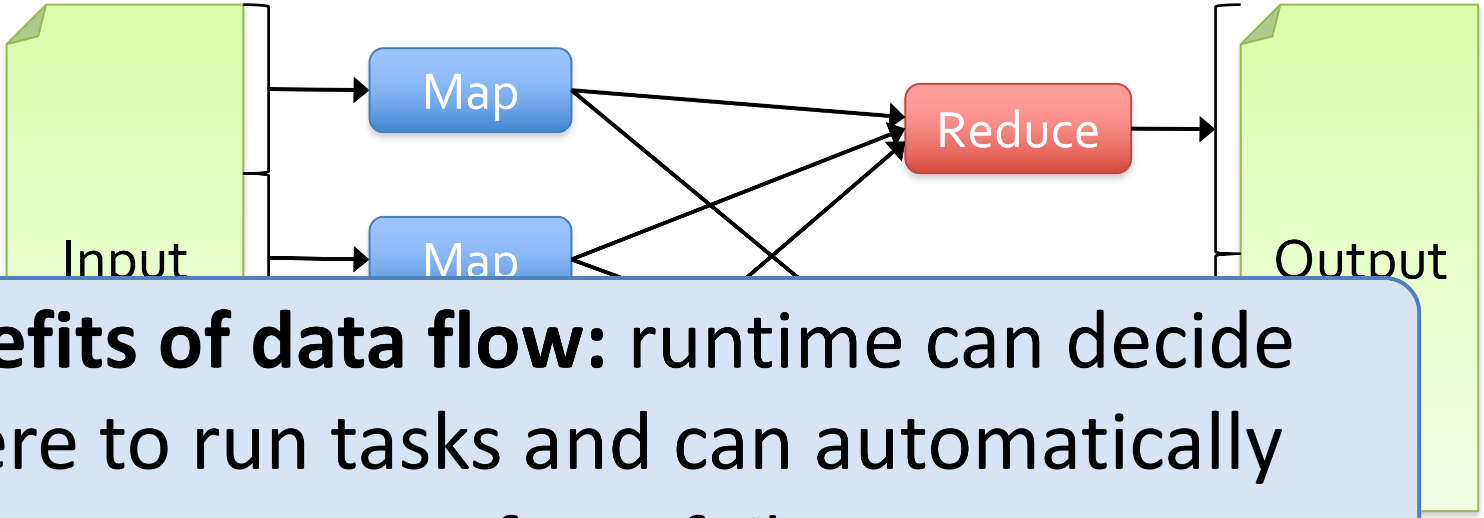
Multi-core

Iteration

Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:



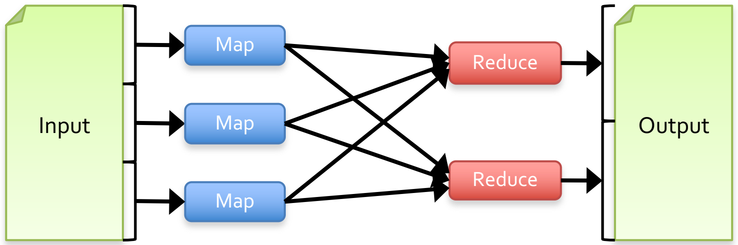
Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in \text{neighbors}} \text{rank}_i \,/\, |\text{neighbors}_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```



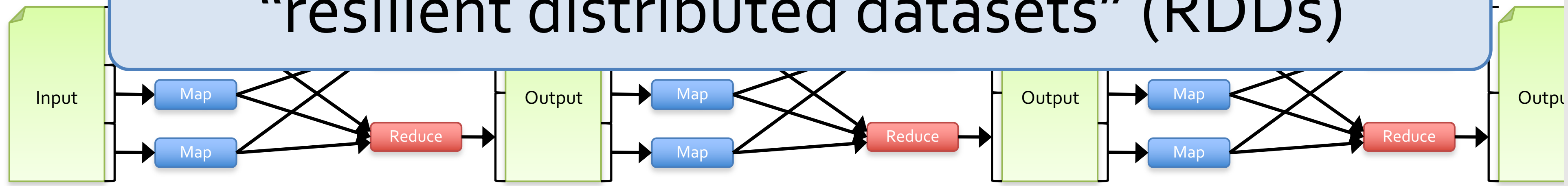
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in \text{neighbors}} \text{rank}_i \,/\, |\text{neighbors}_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
```

Solution: augment data flow model with "resilient distributed datasets" (RDDs)



Programming Model

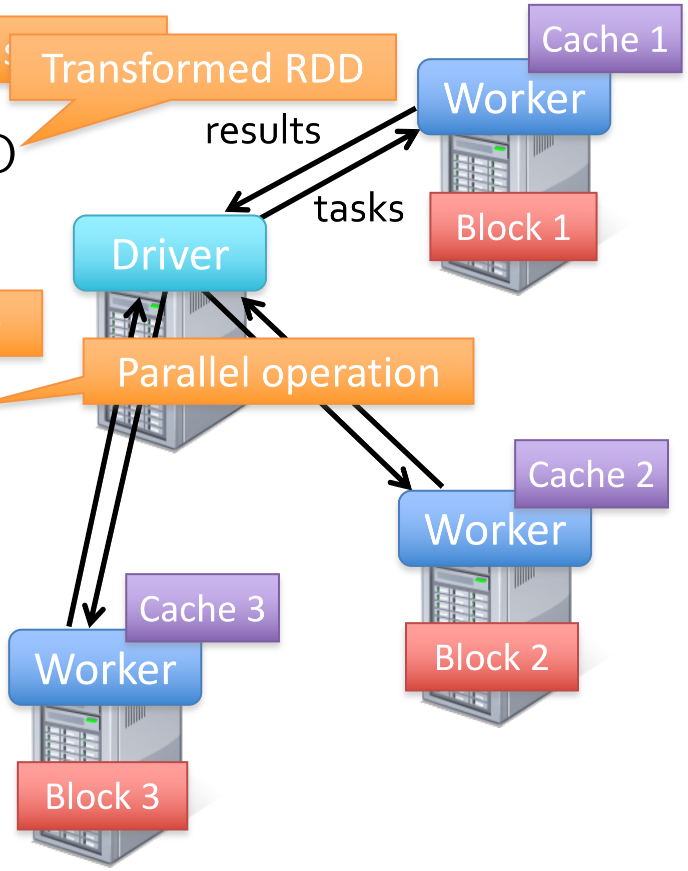
- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be *cached* across parallel operations
- Parallel operations on RDDs
  - Reduce, collect, count, save, …
- Restricted shared variables
  - Accumulators, broadcast variables

Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Cached RDD

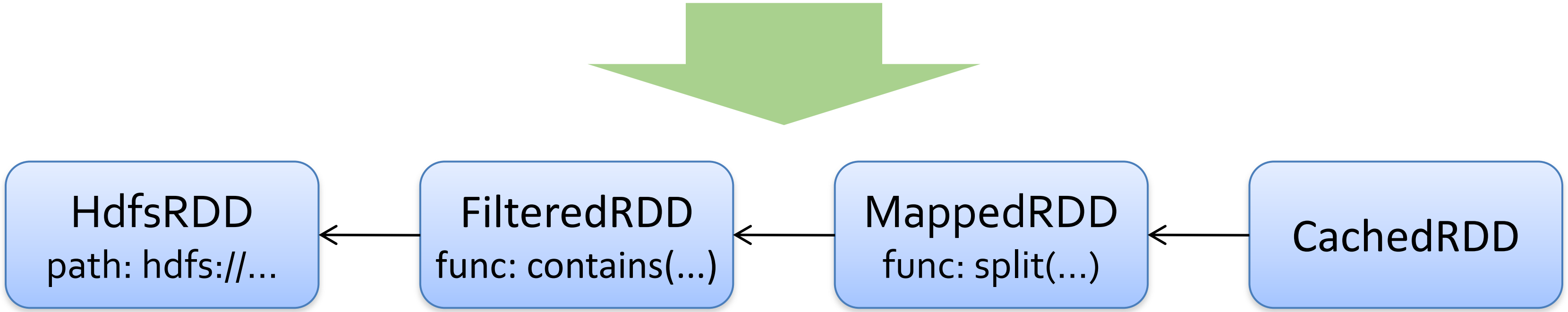


Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

RDD Fault Tolerance

- RDDs maintain *lineage* information that can be used to reconstruct lost partitions
- Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .persist()
```



Systems

```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

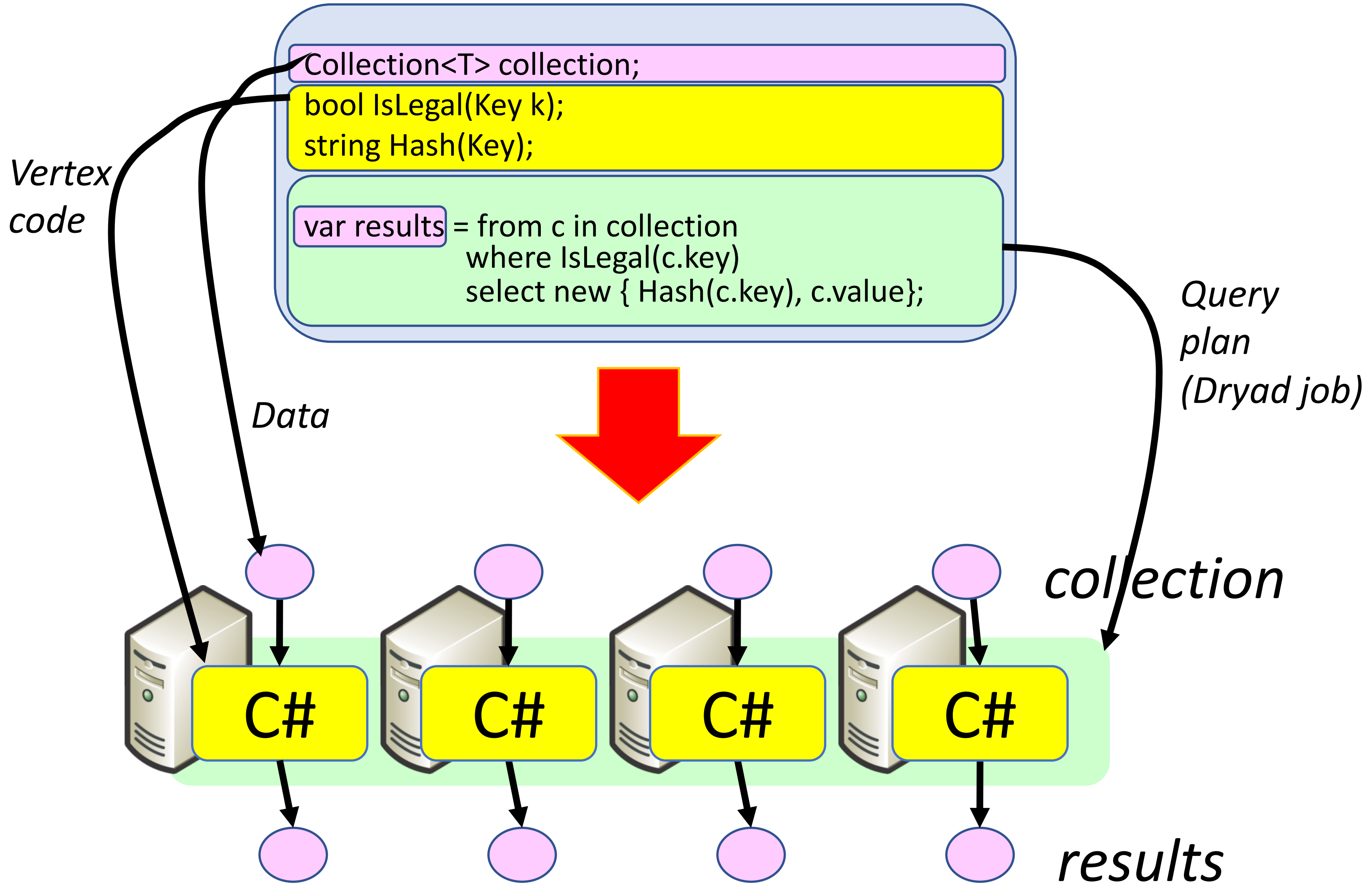
App

	SQL	Sawzall	≈SQL	LINQ, SQL
Language		Sawzall	Pig, Hive	DryadLINQ
Execution	Paralle Databas			
Storage		BigTable	S3	SQL Server

```
-- import the file as lines
CREATE EXTERNAL TABLE lines(line string)
LOAD DATA INPATH 'books' OVERWRITE INTO TABLE lines;

-- create a virtual view that splits the lines
SELECT word, count(*) FROM lines
    LATERAL VIEW explode(split(text, ' ' )) lTable as word
    GROUP BY word;
```

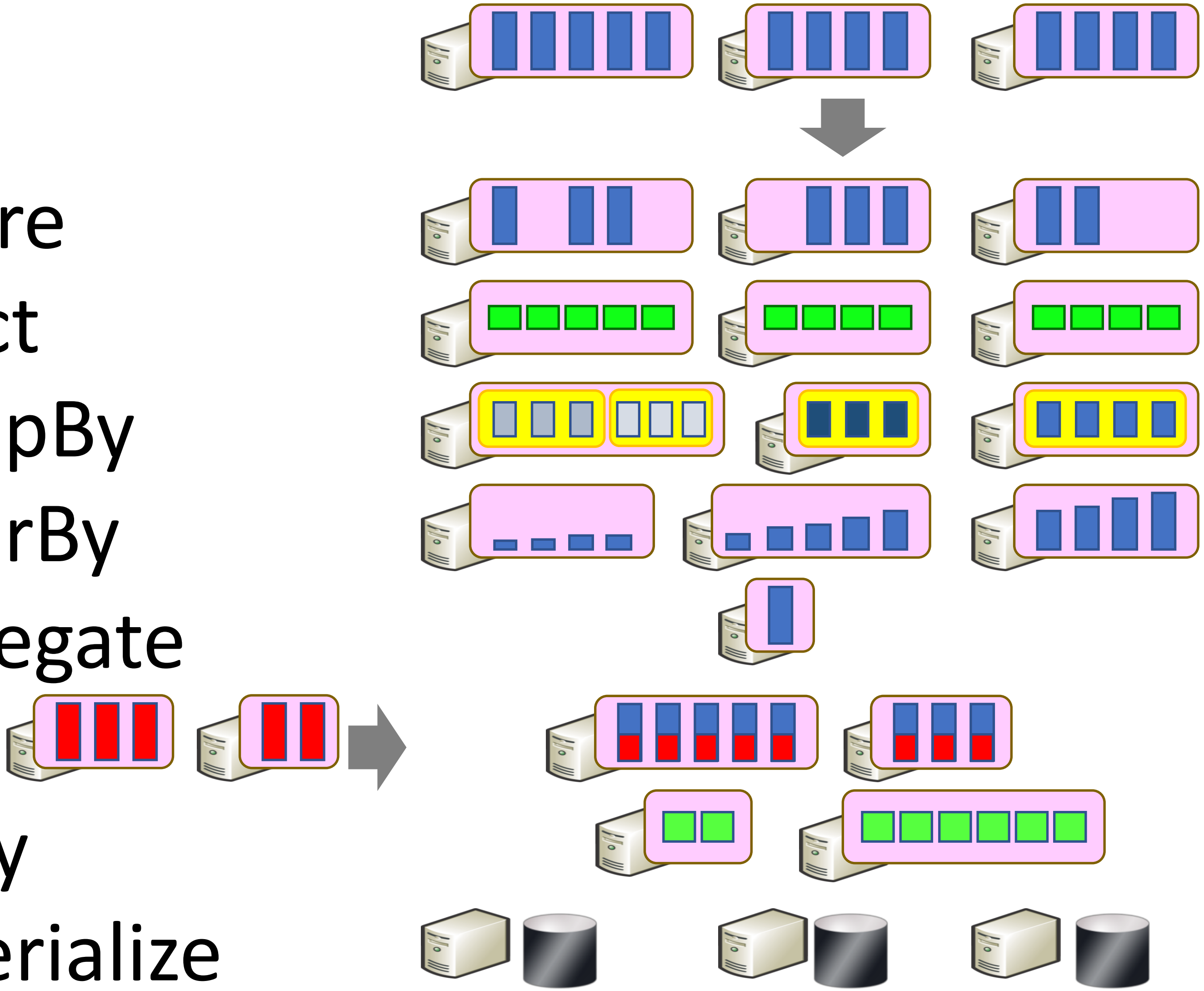
```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

DryadLINQ = LINQ + Dryad



Programming Model

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize



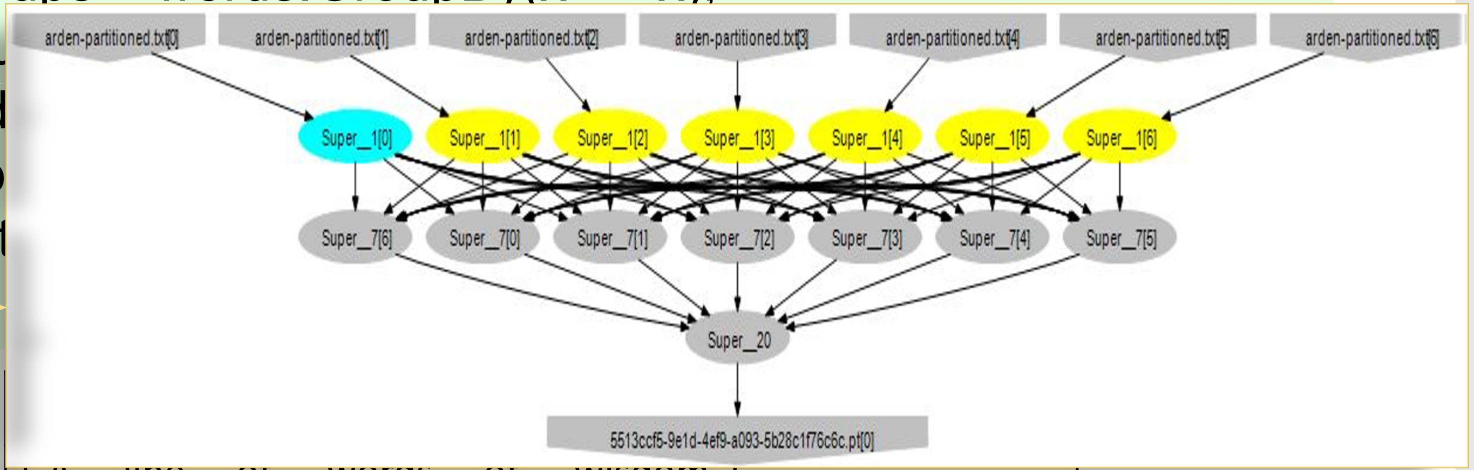
Example: Histogram

```
public static IQueryable<Pair> Histogram(
      IQueryable<LineRecord> input, int k)
{
    var words = input.SelectMany(x => x.line.Split(' '));
    ...roups = words.GroupBy(x => x);
```

SelectMany
Sort
GroupBy+Select
HashDistribute

MergeSort
GroupBy
Select
Sort
Take

MergeSort
Take



[“A”, “line”, “of”, “words”, “of”, “wisdom”]
[[“A”], [“line”], [“of”, “of”], [“words”], [“wisdom”]]
[{“A”, 1}, {“line”, 1}, {“of”, 2}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}]

RDDs

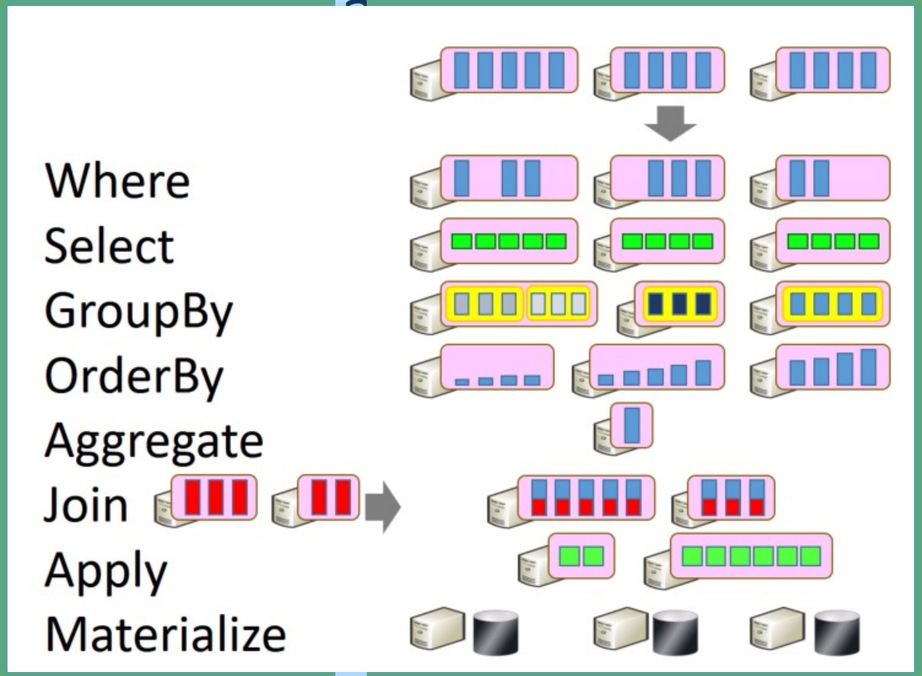
- Immutable, partitioned, logical collection of records
  - Need not be materialized

Transformations (define a new RDD)

map
filter
sample
union
groupByKey
reduceByKey
join
persist/cache
...

Parallel operations (return a result to driver)

reduce



RDDs vs Distributed Shared Memory

Concern	RDDs	Distr. Shared Mem.
Reads	Fine-grained	Fine-grained
Writes	Bulk transformations	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using speculative execution	Difficult
Work placement	Automatic based on data locality	Up to app (but runtime aims for transparency)

Summary

Dataflow key enabler for cluster-scale parallelism

Key issues become runtime's responsibility

- Data movement
- Scheduling
- Fault-tolerance

Example: Counting Words...

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key,
          Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
      result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce handles all the other details!

Redundant Execution

Slow worker can throttle performance: why?

What makes a worker slow?

- Other Jobs on machine (how could we fix)
- Bad disks, soft errors
- Exotica (processor caches disabled!)

Solution: spawn backups near end of phase

MapReduce is sub-optimal

Modern DBMSs: hash + B-tree indexes to accelerate data access.

- Indexes are user-defined
- Could MR do this?

No query optimizer! (oh my, terrible…but good for researchers! ☺)

Skew: wide variance in distribution of keys

- E.g. “the” more common than “zyzzyva”

Materializing splits

- N=1000 mappers → M=500 keys = 500,000 local files
- 500 reducer instances “pull” these files
- DBMSs push splits to sockets (no local temp files)

MapReduce: !novel && feature-poor

- Partitioning data sets (map) == Hash join
- Parallel aggregation == reduce
- User-supplied functions differentiates from SQL:
  - POSTGRES user functions, user aggregates
  - PL/SQL: Stored procedures
  - Object databases

Absent features:

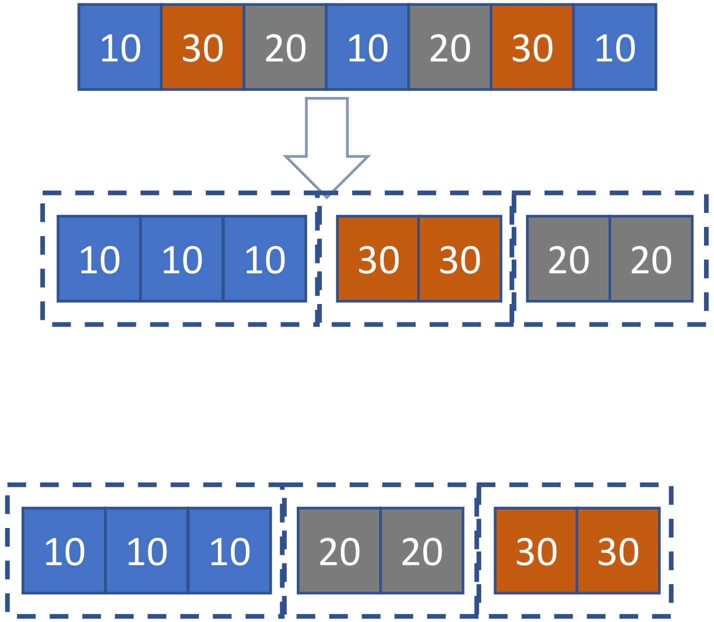
- Indexing
- Update operator
- Transactions
- Integrity constraints, referential integrity
- Views

Review: What is GroupBy?

Group a collection by key

Lambda function maps elements → key

```
var res = ints.GroupBy(x => x);
```




```
foreach(T elem in PF(ints))
{
  key   = KeyLambda(elem);

  group = GetGroup(key)

  group.Add(elem);
}
```

Why is MapReduce backwards?

Map == group-by

Reduce == aggregate

```
SELECT job, COUNT(*) as "numemps"
    FROM employees
    WHERE salary > 1000
    GROUP BY job;
```

- Where is the aggregate in this example?
- Is the DBMS analogy clear?

Why is MapReduce backwards?

Schemas are good (what's a schema?)

Separation of schema from app is good (why?)

High-level access languages are good (why?)