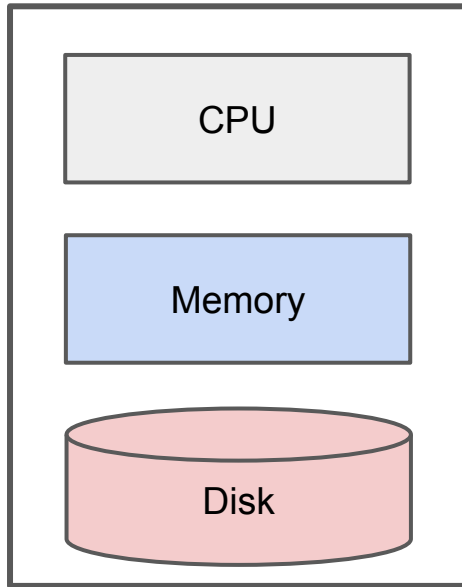


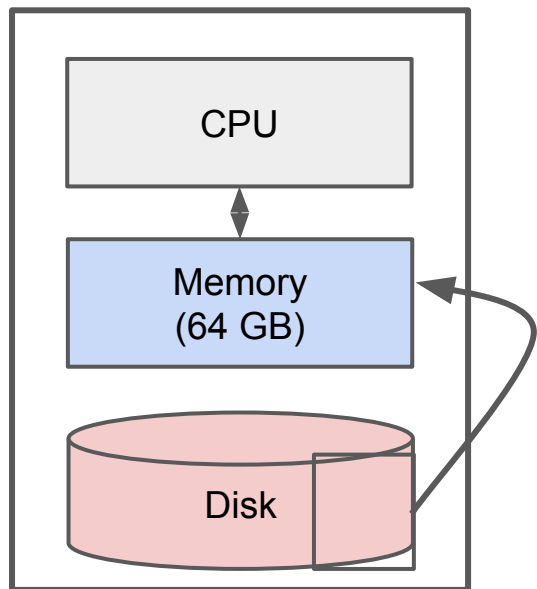
MapReduce

Stony Brook University
CSE545, Fall 2016

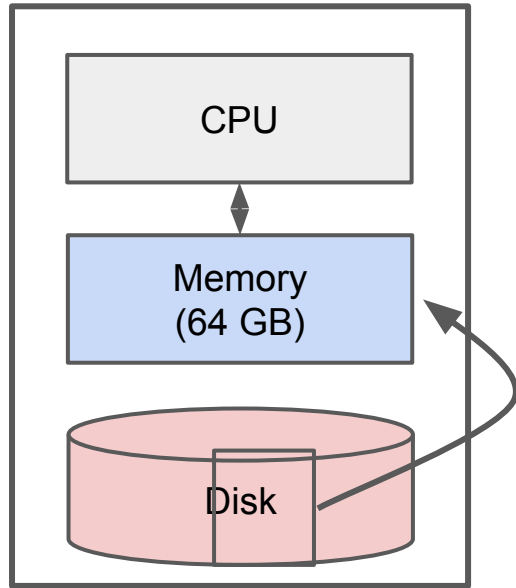
Classical Data Mining



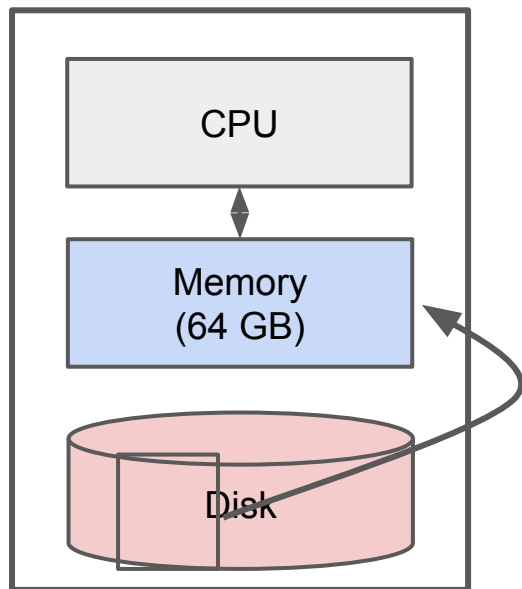
Classical Data Mining



Classical Data Mining



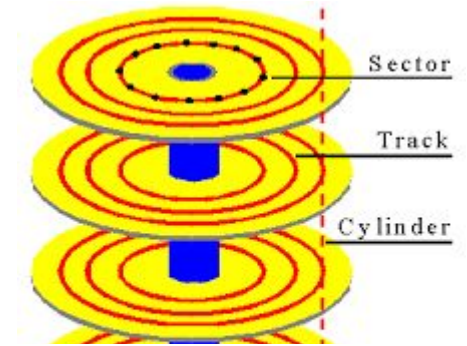
Classical Data Mining



IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

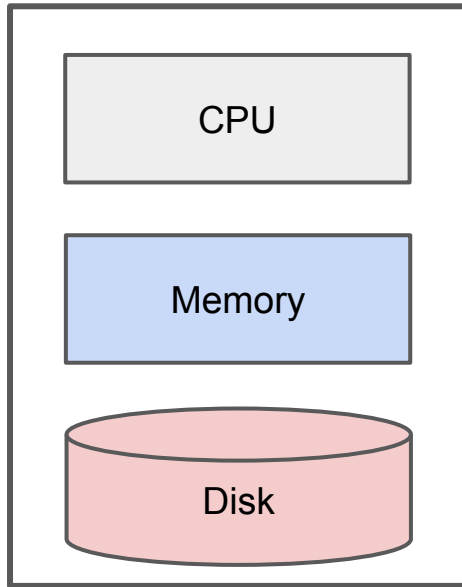
Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.



IO Bound: biggest performance bottleneck is reading / writing to disk.

(starts around 100 GBs; ~10 minutes just to read).

Classical Big Data Analysis



Often focused on efficiently utilizing the disk.

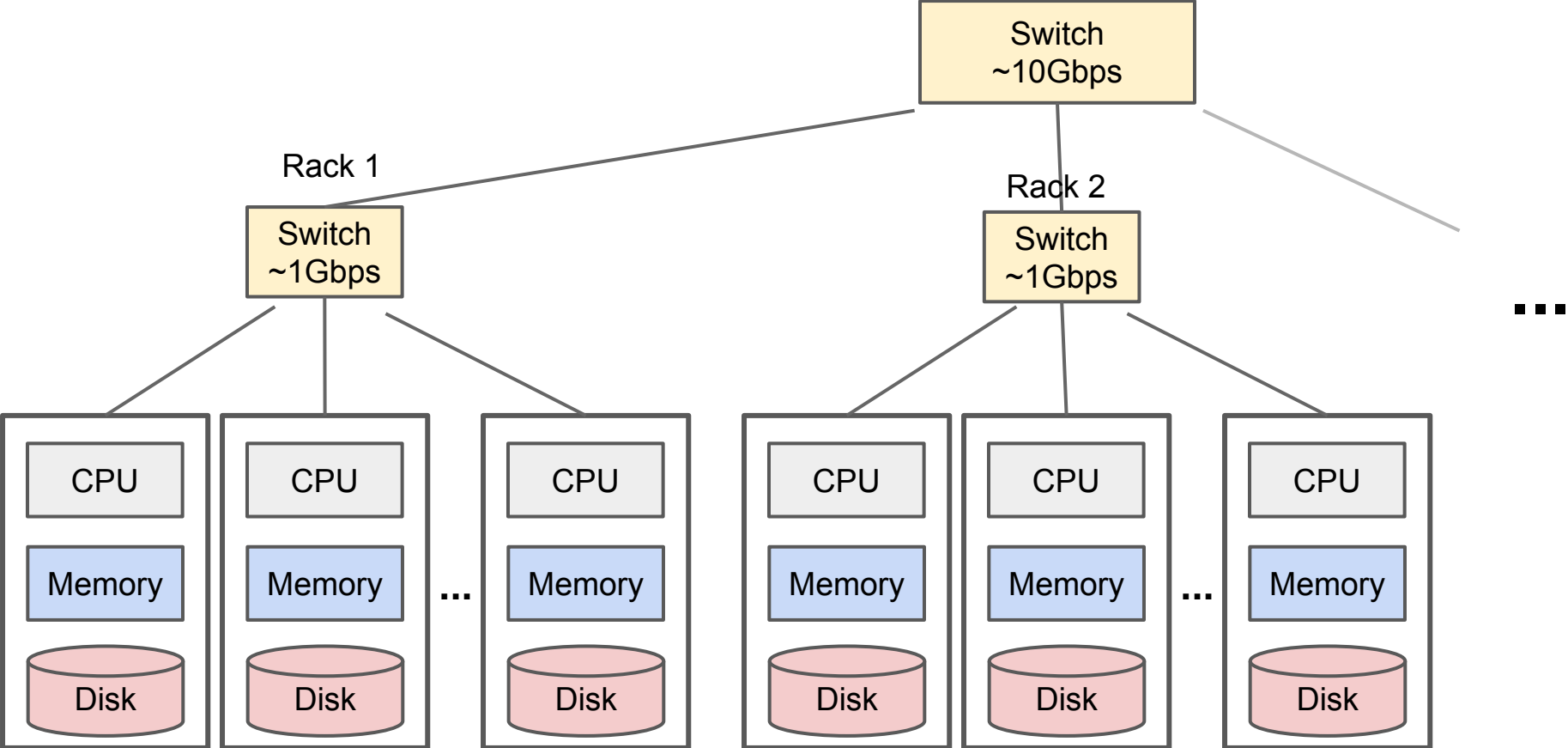
e.g. Apache Lucene / Solr

Still bounded when needing to process all of a large file.

IO Bound

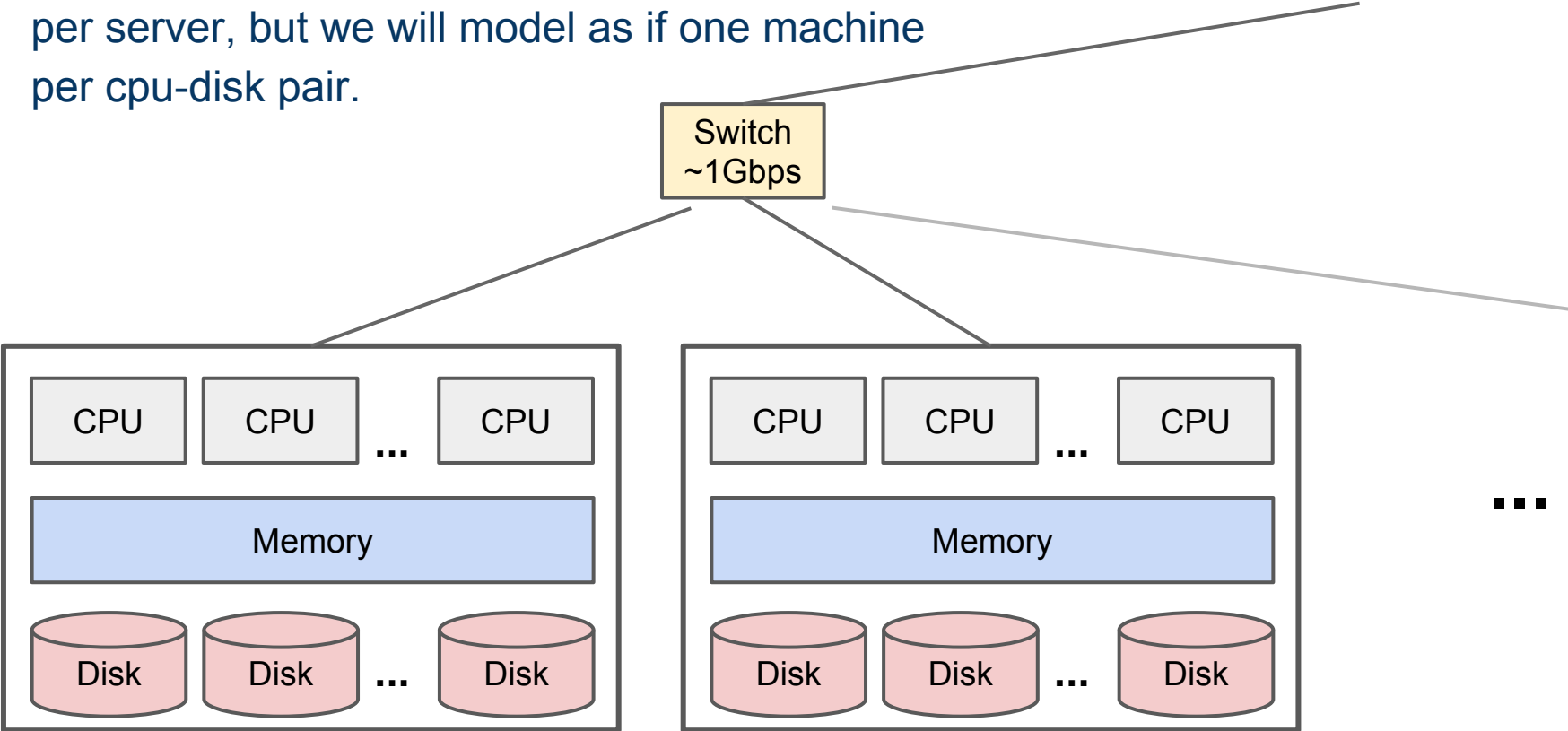
How to solve?

Distributed Architecture (Cluster)



Distributed Architecture (Cluster)

In reality, modern setups often have multiple cpus and disks per server, but we will model as if one machine per cpu-disk pair.



Challenges for IO Cluster Computing

1. Nodes fail
1 in 1000 nodes fail a day
2. Network is a bottleneck
Typically 1-10 Gb/s throughput
3. Traditional distributed programming is often ad-hoc and complicated

Challenges for IO Cluster Computing

1. Nodes fail

1 in 1000 nodes fail a day

Duplicate Data

2. Network is a bottleneck

Typically 1-10 Gb/s throughput

Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated

Stipulate a programming system that can easily be distributed

Challenges for IO Cluster Computing

1. Nodes fail
1 in 1000 nodes fail a day
Duplicate Data
2. Network is a bottleneck
Typically 1-10 Gb/s throughput
Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
Stipulate a programming system that can easily be distributed

MapReduce to the rescue!

Common Characteristics of Big Data

Large files (i.e. >100 GB to TBs)

No need to update in place (append preferred)

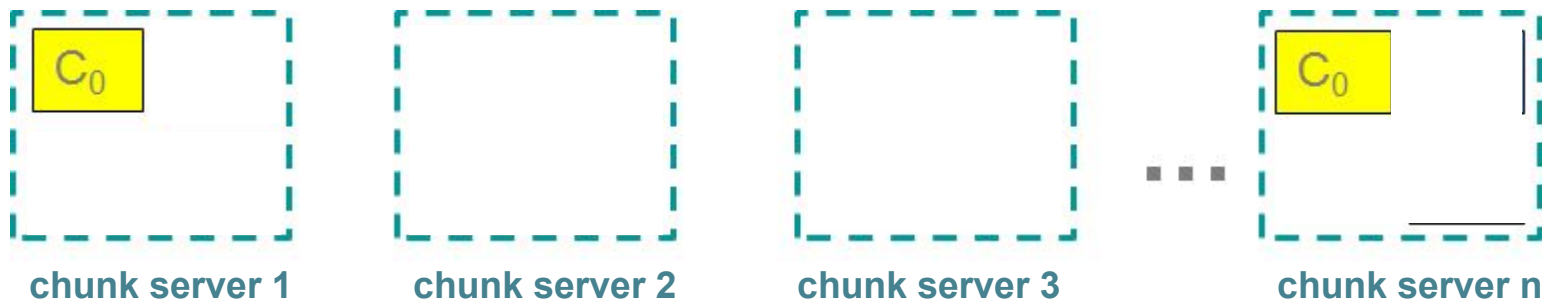
Reads are most common

(e.g. web logs, social media, product purchases)

Distributed File System

(e.g. Apache HadoopDFS, GoogleFS)

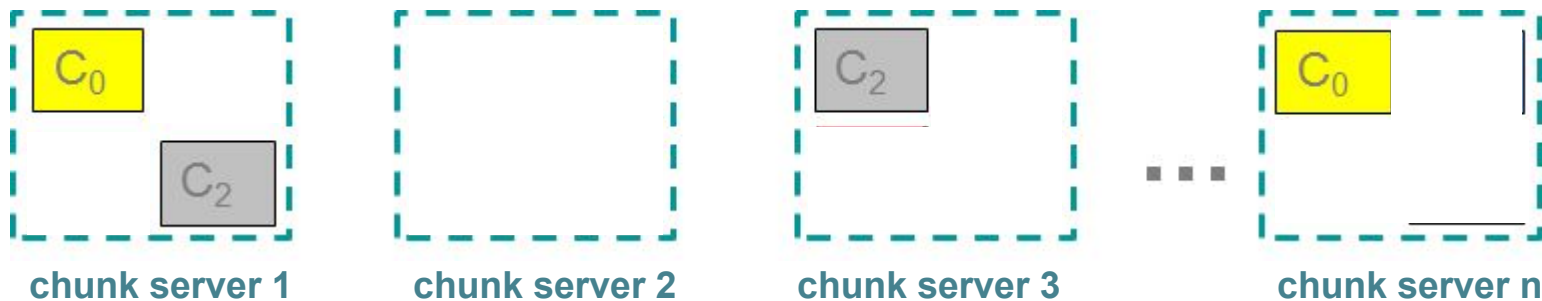
C, D: Two different files



Distributed File System

(e.g. Apache HadoopDFS, GoogleFS)

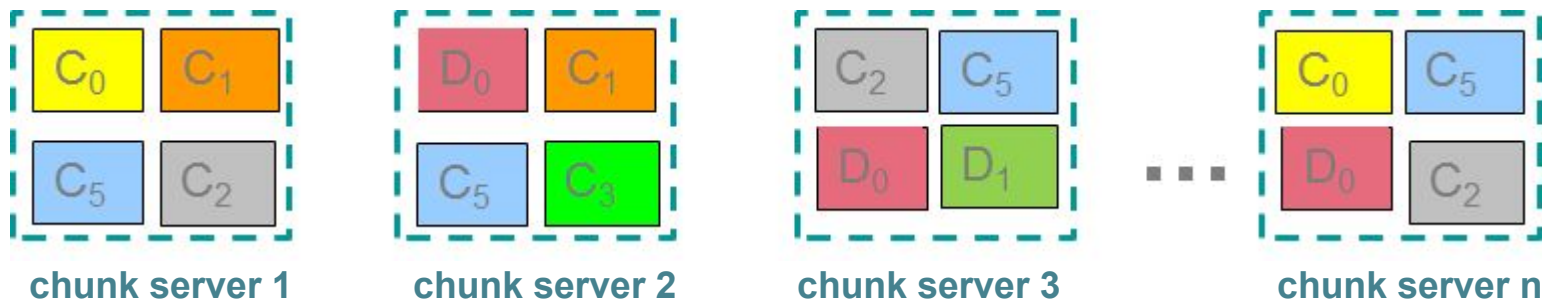
C, D: Two different files



Distributed File System

(e.g. Apache HadoopDFS, GoogleFS)

C, D: Two different files



Components of a Distributed File System

Chunk servers (on Data Nodes)

- File is split into contiguous chunks

- Typically each chunk is 16-64MB

- Each chunk replicated (usually 2x or 3x)

- Try to keep replicas in different racks

Name node (aka master node)

- Stores metadata about where files are stored

- Might be replicated or distributed across data nodes.

Client library for file access

- Talks to master to find chunk servers

- Connects directly to chunk servers to access data

Challenges for IO Cluster Computing

1. Nodes fail

1 in 1000 nodes fail a day

Duplicate Data (Distributed FS)



2. Network is a bottleneck

Typically 1-10 Gb/s throughput

Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated

Stipulate a programming system that can easily be distributed

What is MapReduce?

1. A style of programming

input chunks => **map tasks** | group_by keys | **reduce tasks** => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.

E.g. counting words:

```
tokenize(document) | sort | uniq -C
```

What is MapReduce?

1. A style of programming

input chunks => **map tasks** | group_by keys | **reduce tasks** => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.

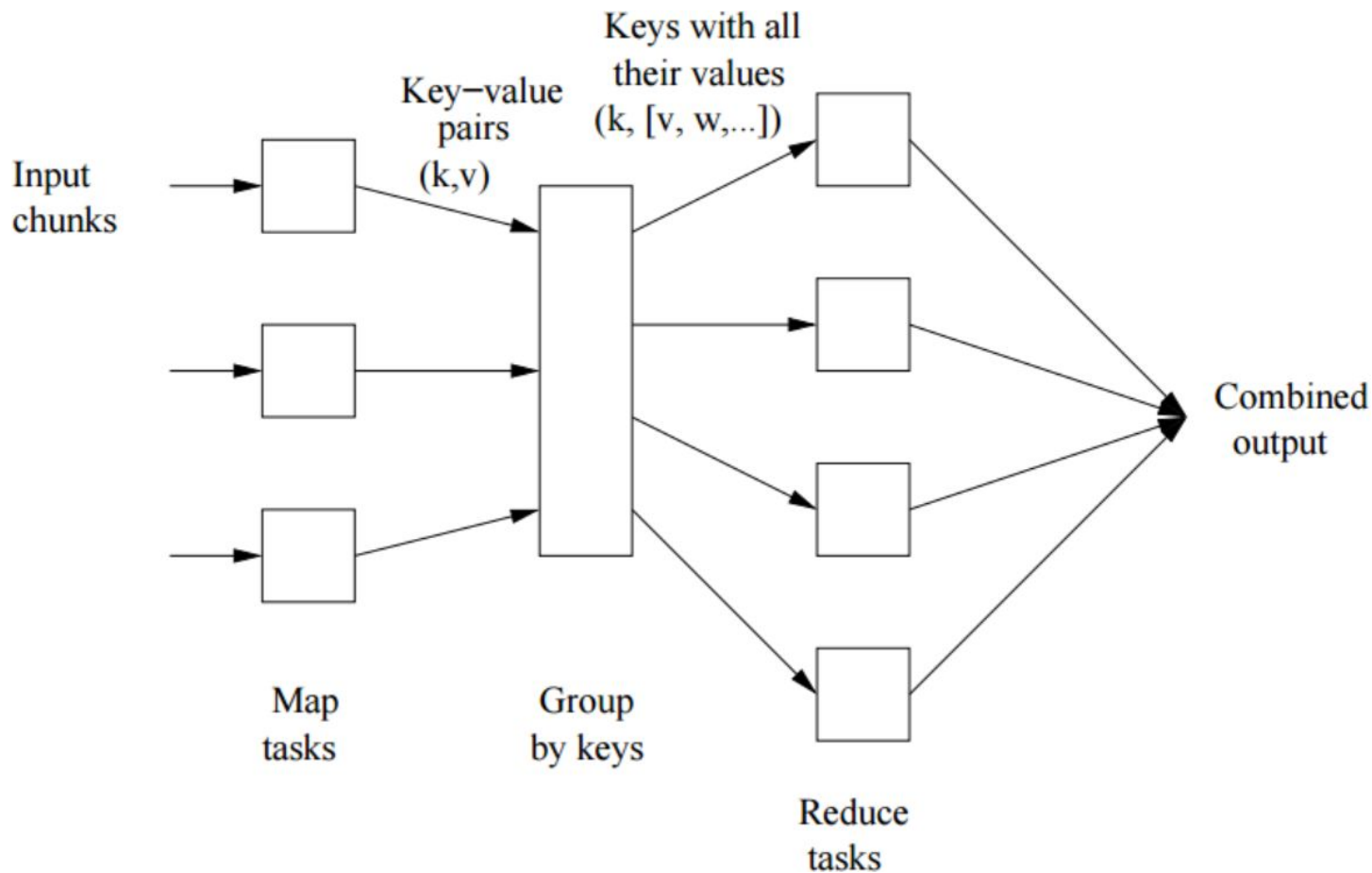
E.g. counting words:

```
tokenize(document) | sort | uniq -C
```

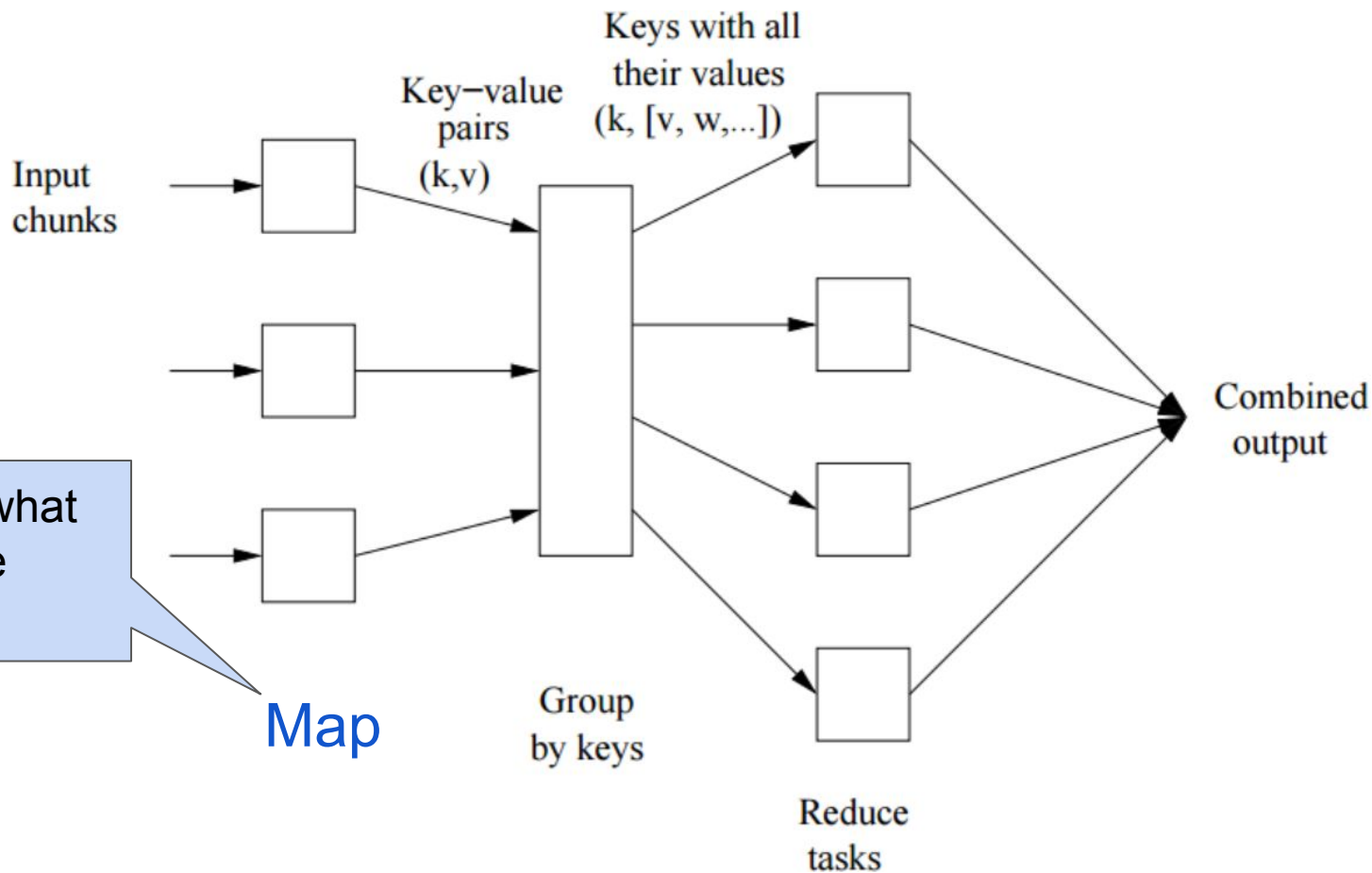
2. A system that distributes MapReduce style programs across a distributed file-system.

(e.g. Google’s internal “MapReduce” or apache.hadoop.mapreduce with hdfs)

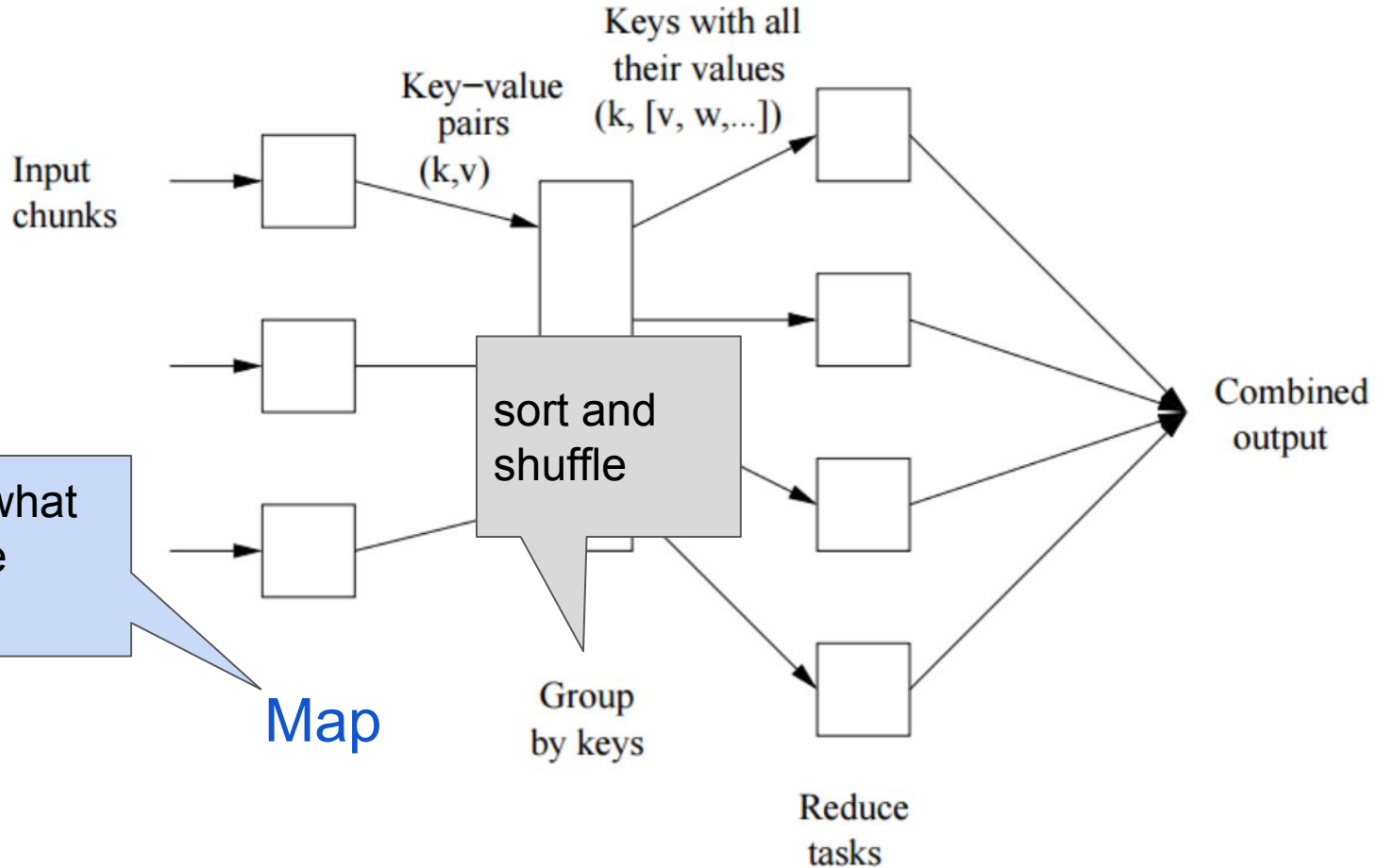
What is MapReduce?



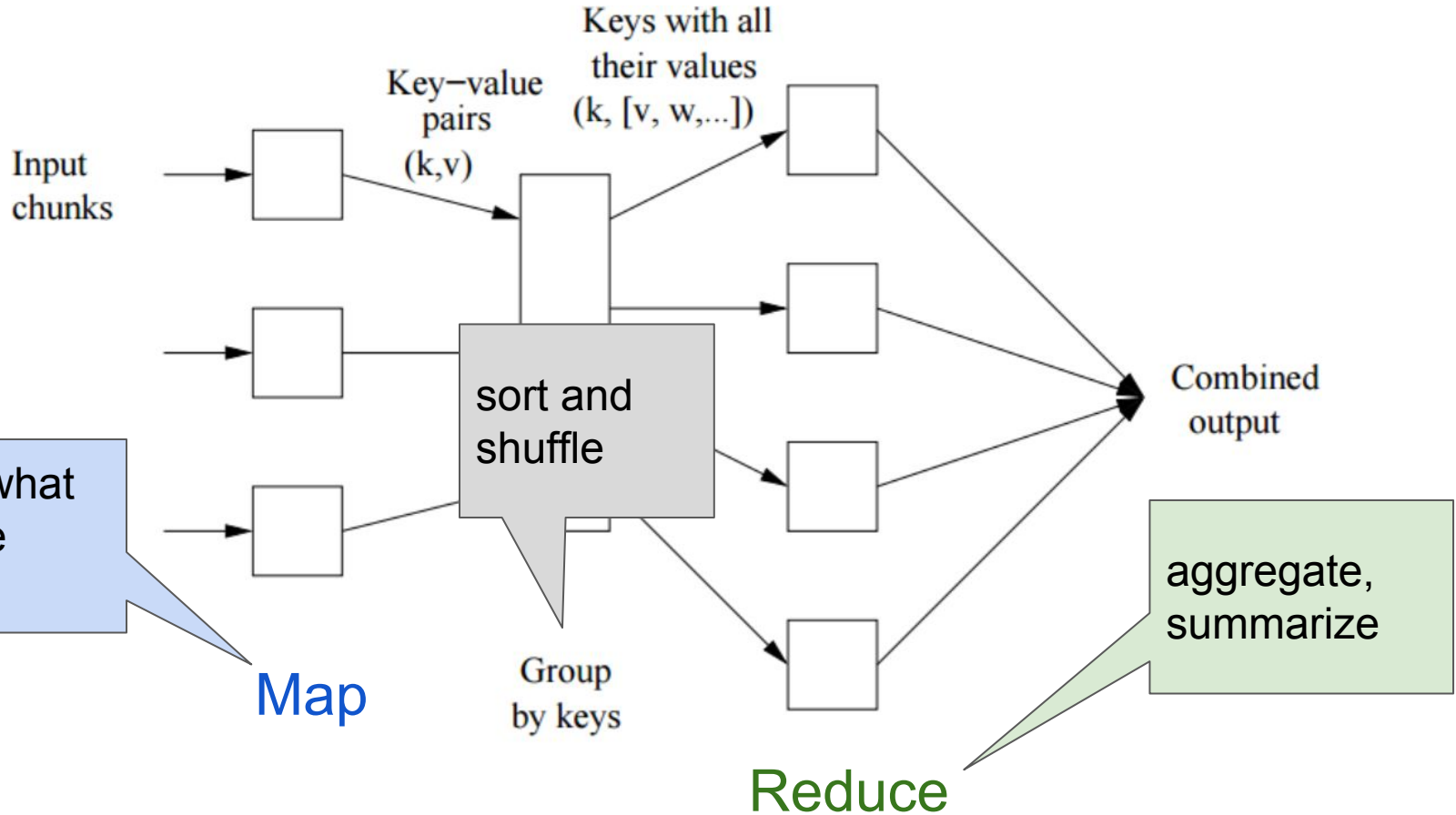
What is MapReduce?



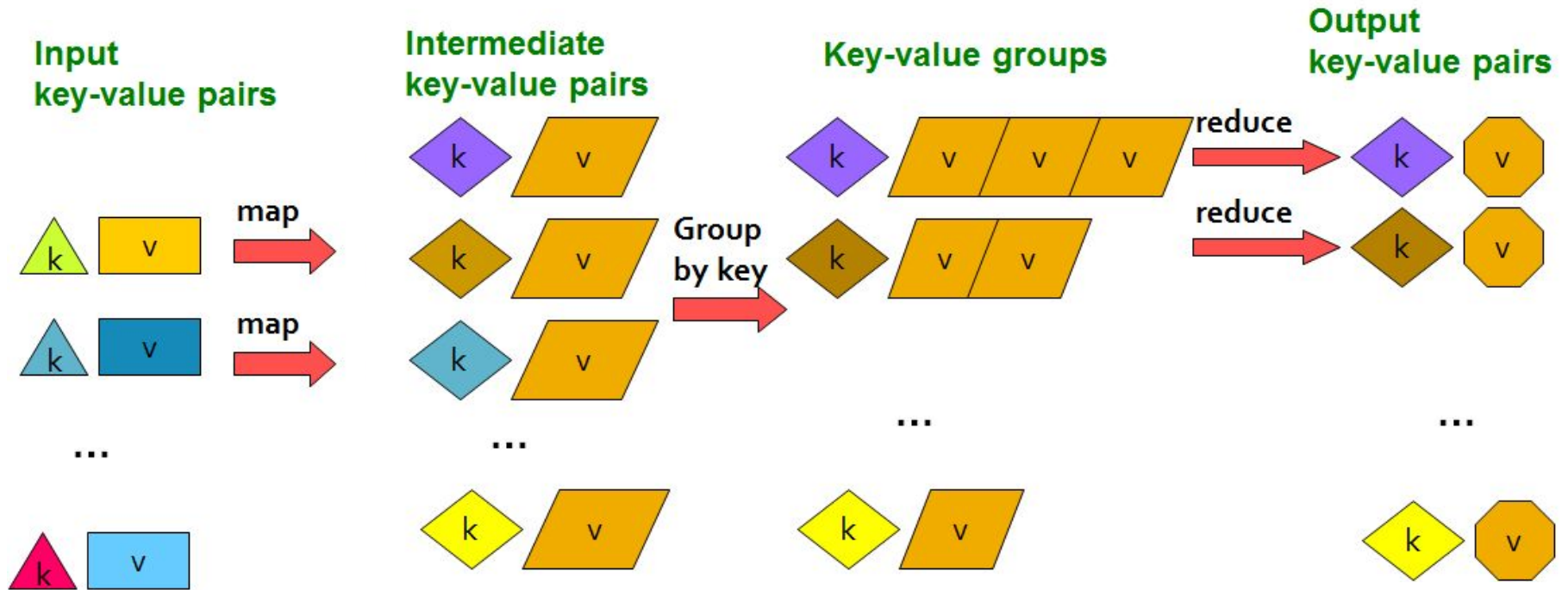
What is MapReduce?



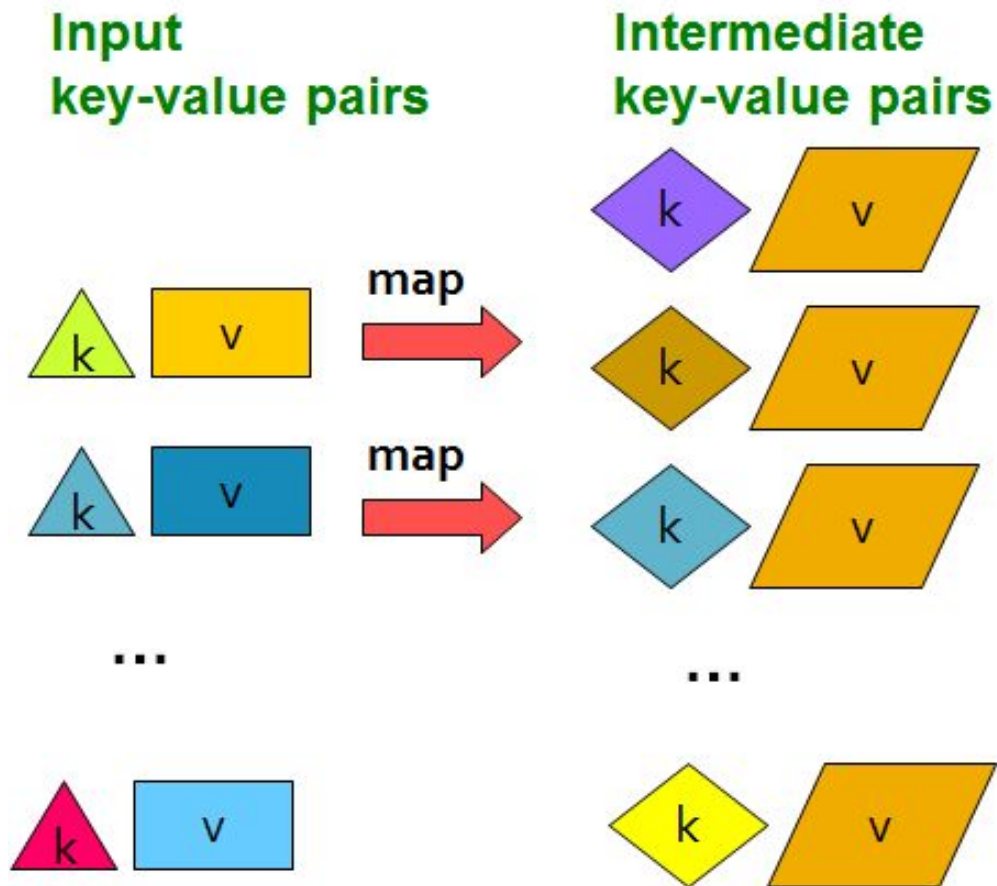
What is MapReduce?



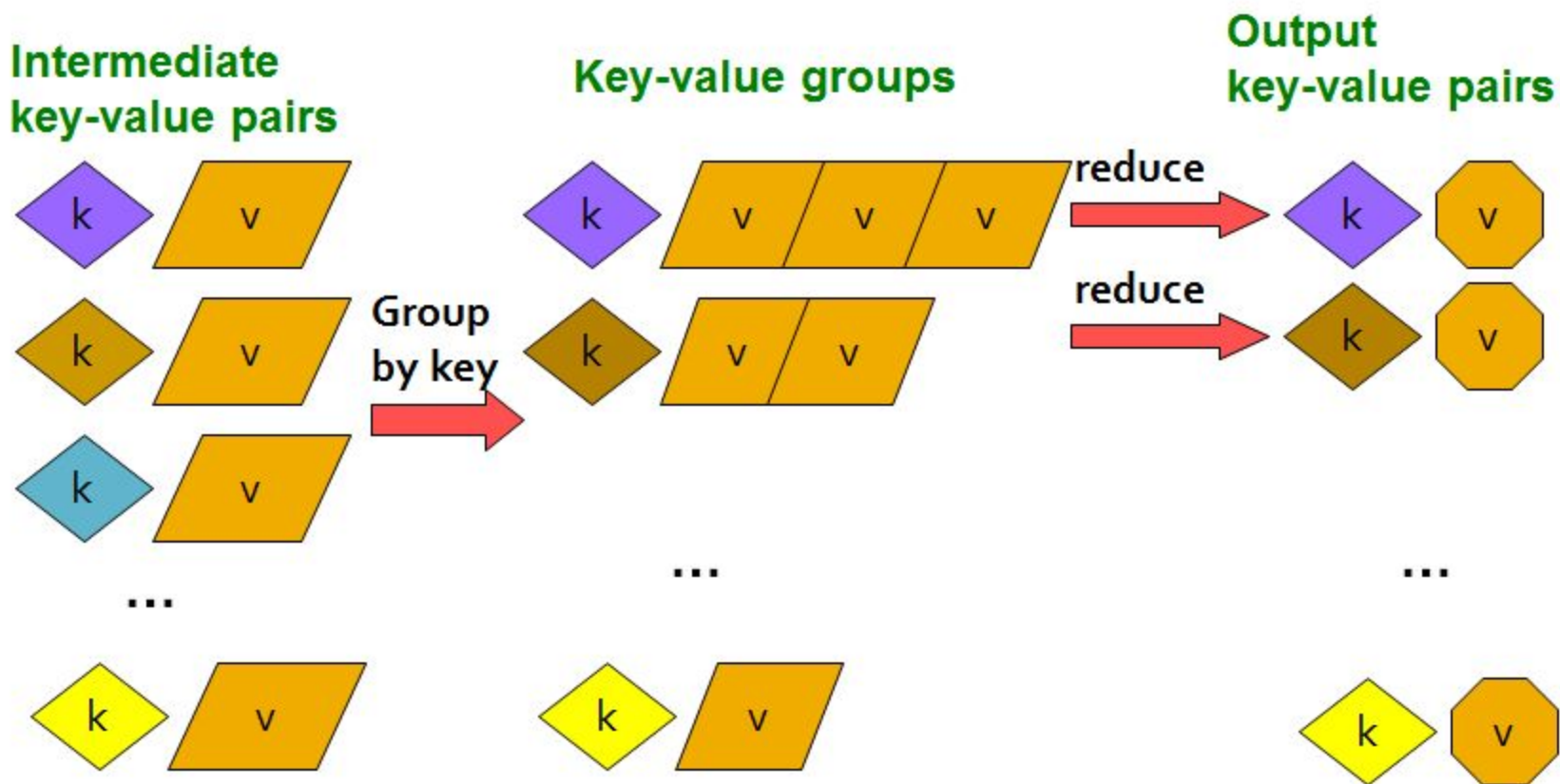
What is MapReduce?



The Map Step



The Reduce Step



What is MapReduce?

Map: $(k, v) \rightarrow (k', v')^*$

(Written by programmer)

Group by key: $(k_1', v_1'), (k_2', v_2'), \dots \rightarrow (k_1', (v_1', v', \dots)),$
(system handles) $(k_2', (v_1', v', \dots)), \dots$

Reduce: $(k', (v_1', v', \dots)) \rightarrow (k', v'')^*$

(Written by programmer)

Example: Word Count

```
tokenize(document) | sort | uniq -C
```

Example: Word Count

```
tokenize(document) | sort | uniq -C
```

Map: extract
what you
care about.

sort and
shuffle

Reduce:
aggregate,
summarize

Example: Word Count

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

Big document

(Leskovec et al., 2014; <http://www.mmms.org/>)

Provided by the programmer

MAP:

Read input and produces a set of key-value pairs

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
.....

Big document

(key, value)

Provided by the programmer

MAP:

Read input and produces a set of key-value pairs

Group by key:

Collect all pairs with same key

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

Big document

(key, value)

(key, value)

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

Big document

(key, value)

(key, value)

(key, value)

Chunks

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
...

(key, value)

Group by key:
Collect all pairs with same key

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(key, value)

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

(key, value)

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

Big document



Example: Word Count (version 1)

```
def map(k, v):  
    for w in tokenize(v):  
        yield (w,1)
```

```
def reduce(k, vs):  
    return len(vs)
```

Example: Word Count (version 2)

```
def map(k, v):  
    counts = dict()  
    for w in tokenize(v):  
        try:  
            counts[w] += 1  
        except KeyError:  
            counts[w] = 1  
    for item in counts.iteritems():  
        yield item
```

} counts each word within the chunk
(try/except is faster than
“if w in counts”)

```
def reduce(k, vs):  
    return sum(vs)
```

} sum of counts from different chunks

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

Example: Relational Algebra

Select

$R(A_1, A_2, A_3, \dots)$, Relation R , Attributes A_*

return only those attribute tuples where condition C is true

Example: Relational Algebra

Select

$R(A_1, A_2, A_3, \dots)$, Relation R , Attributes A_*

return only those attribute tuples where condition C is true

```
def map(k, v): #v is list of attribute tuples
    for t in v:
        if t satisfies C:
            yield (t, t)
```

```
def reduce(k, vs):
    For each v in vs:
        yield (k, v)
```

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

```
def map(k, v): #v is (R1=(A, B), R2=(B, C)); B are matched attributes
    for (a, b) in R1:
        yield (b, (R1, a))
    for (b, c) in R2:
        yield (b, (R2, c))
```

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

```
def map(k, v): #v is (R1=(A, B), R2=(B, C)); B are matched attributes
    for (a, b) in R1:
        yield (b, (R1, a))
    for (b, c) in R2:
        yield (b, (R2, c))

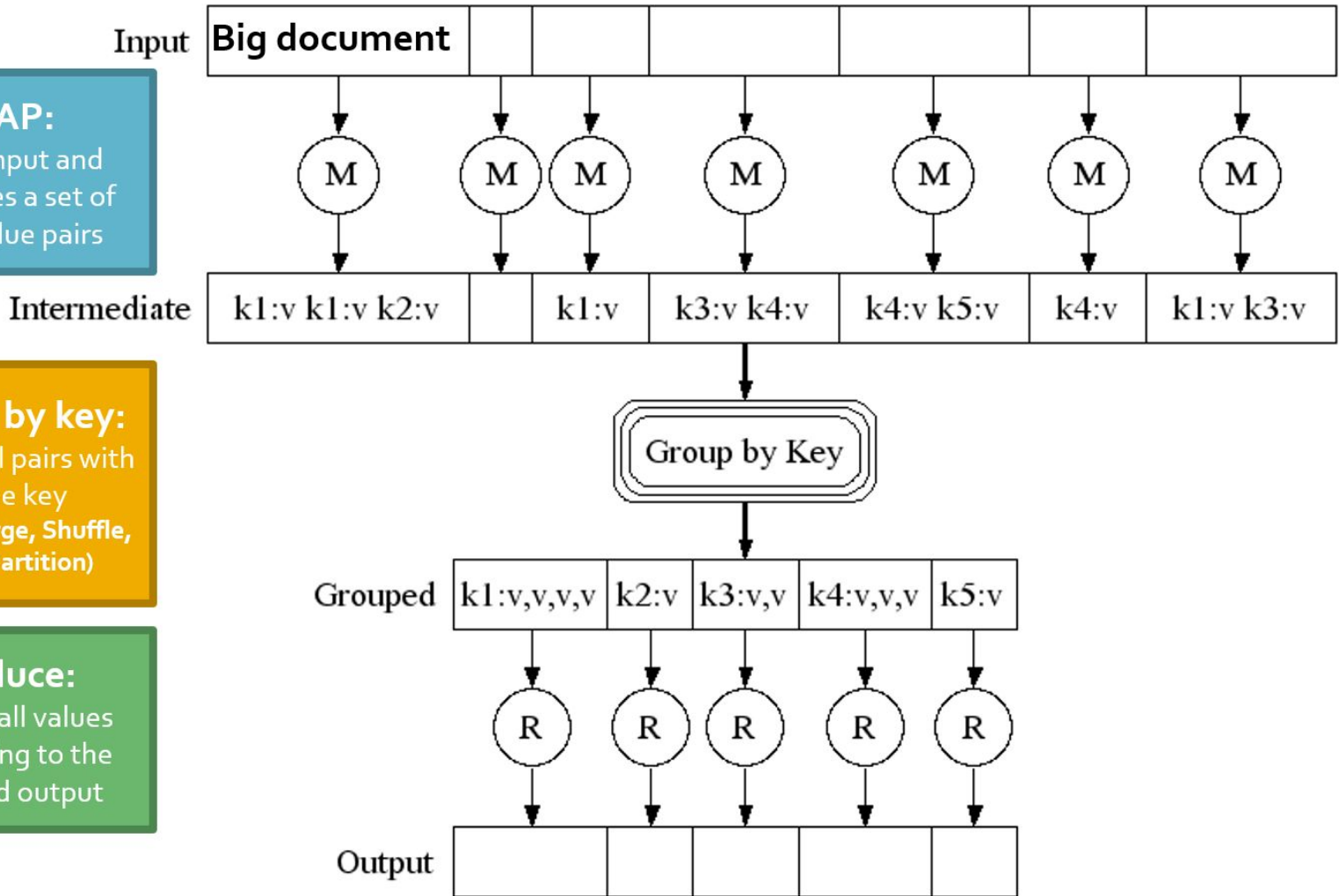
def reduce(k, vs):
    r1, r2 = [], []
    for (S, x) in vs: #separate rs
        if S == r1: r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin, (a, k, c)) #k is
```

Data Flow

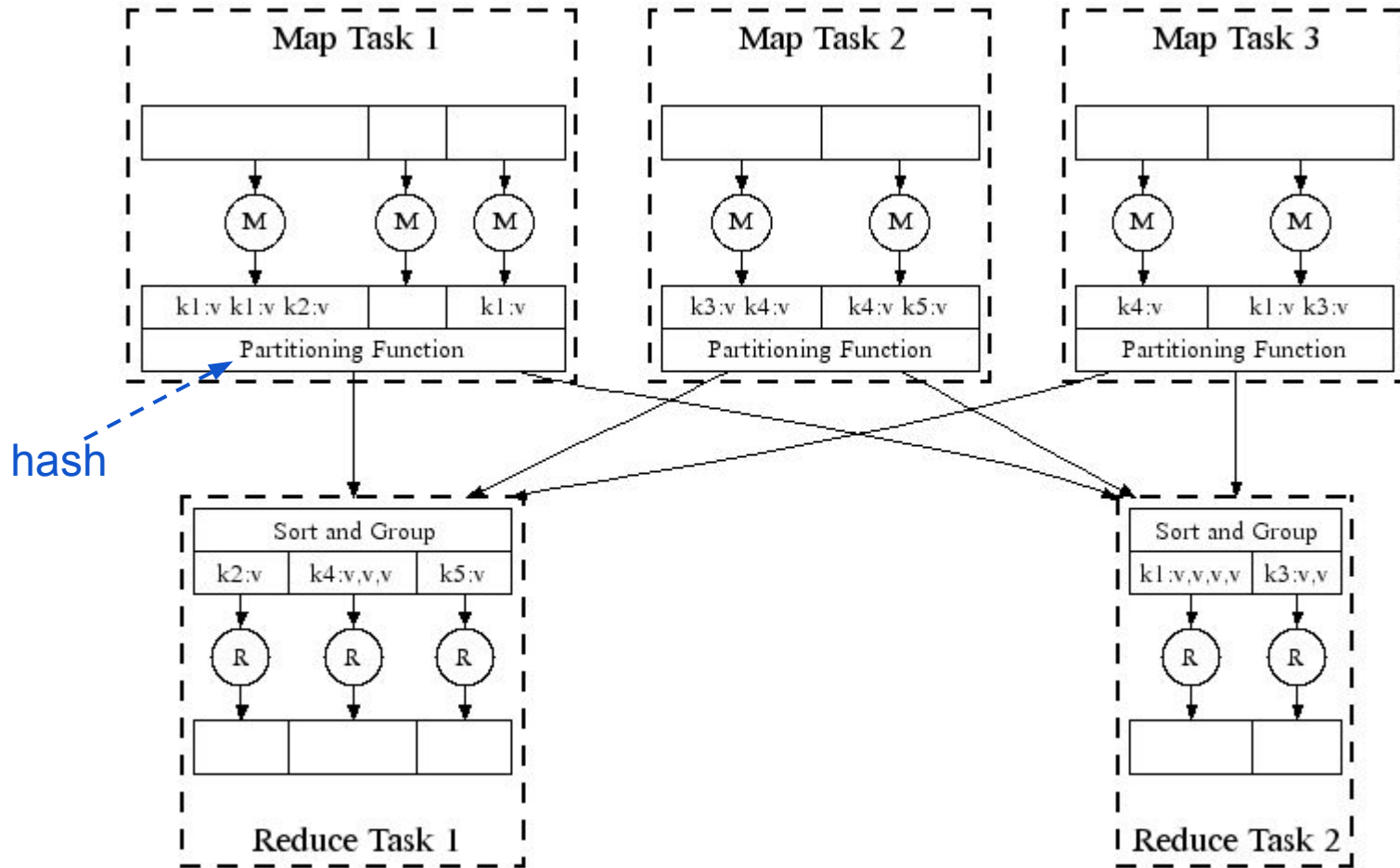
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

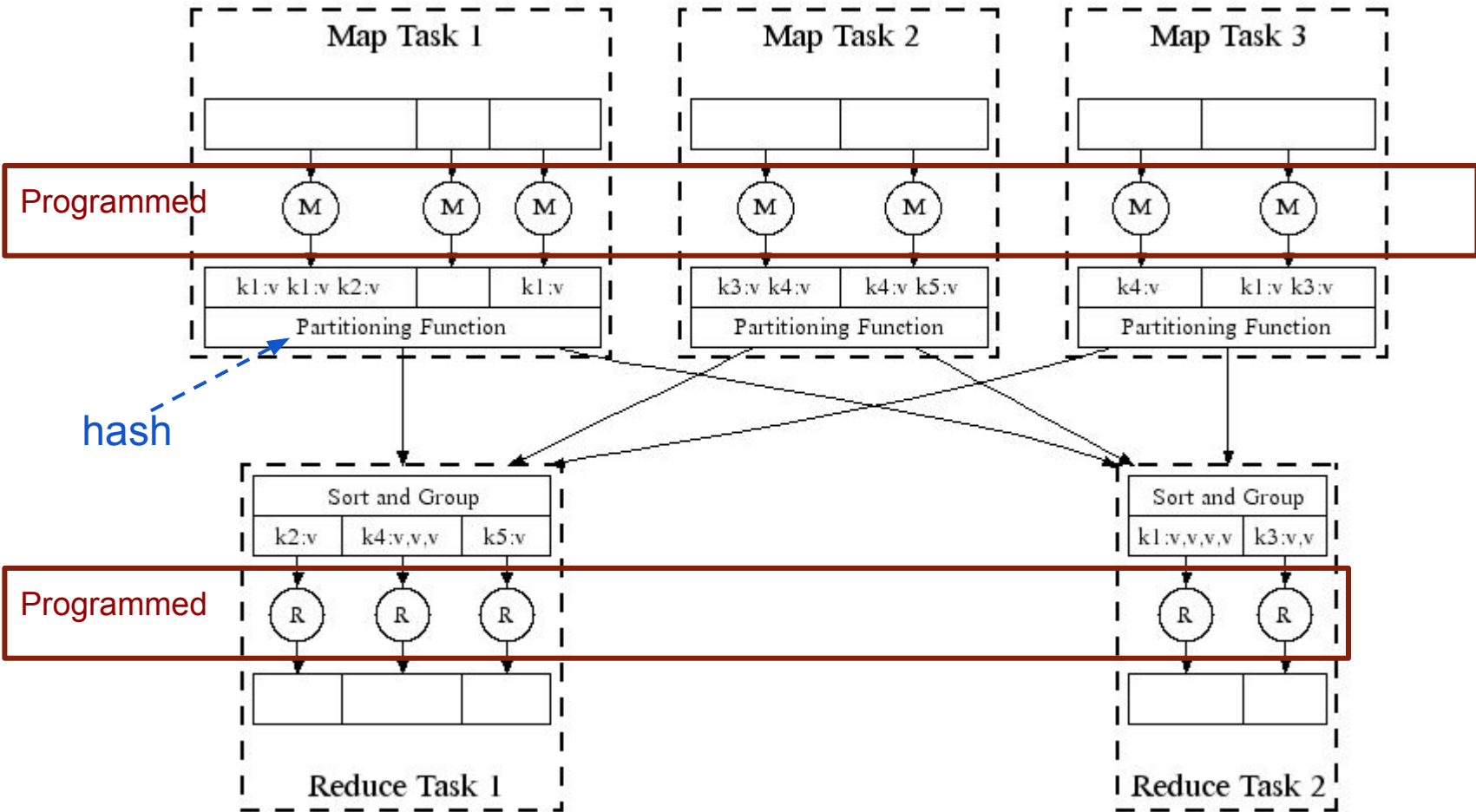
Reduce:
Collect all values belonging to the key and output



Data Flow: In Parallel



Data Flow: In Parallel



Data Flow

DFS → Map → Map's Local FS → Reduce → DFS

```
graph LR; DFS1[DFS] --> Map[Map]; Map --> LocalFS[Map's Local FS]; LocalFS --> Reduce[Reduce]; Reduce --> DFS2[DFS]
```

Data Flow

MapReduce system handles:

- Partitioning
- Scheduling map / reducer execution
- Group by key

- Restarts from node failures
- Inter-machine communication

Data Flow

DFS → MapReduce → DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates

Data Flow

DFS  MapReduce  DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
 - Task status: idle, in-progress, complete
 - Receives location of intermediate results and schedules with reducer
 - Checks nodes for failures and restarts when necessary
 - All map tasks on nodes must be completely restarted
 - Reduce tasks can pickup with reduce task failed

Data Flow

DFS  MapReduce  DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
 - Task status: idle, in-progress, complete
 - Receives location of intermediate results and schedules with reducer
 - Checks nodes for failures and restarts when necessary
 - All map tasks on nodes must be completely restarted
 - Reduce tasks can pickup with reduce task failed

DFS  MapReduce  DFS  MapReduce  DFS

Data Flow

Skew: The degree to which certain tasks end up taking much longer than others.

Handled with:

- More reducers than reduce tasks
- More reduce tasks than nodes

Data Flow

Key Question: *How many Map and Reduce jobs?*

Data Flow

Key Question: How many Map and Reduce jobs?

M: map tasks, *R*: reducer tasks

A: If possible, one chunk per map task.

and $M \gg |\text{nodes}|$

(better handling of node failures, better load balancing)

$R < M$

(reduces number of files stored in DFS)

Communication Cost Model

How to assess performance?

- (1) Computation: Map + Reduce + System Tasks
- (2) Communication: Moving key, value pairs

Communication Cost Model

How to assess performance?

- (1) Computation: Map + Reduce + System Tasks
- (2) Communication: Moving key, value pairs

Ultimate Goal: wall-clock Time.

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

- Mappers and reducers often single pass $O(n)$ within node
- System: sort the keys is usually most expensive
- In any case, can add more nodes

(2) Communication: Moving key-value pairs

Ultimate Goal: wall-clock Time.

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 gigabits per sec;
- HD read: 50-150 gigabytes per sec
- Even reading from disk to memory typically takes longer than operating on the data.

Communication Cost Model

How to assess performance?

Communication Cost = input size +
(sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 **gigabits** per sec;
- HD read: 50-150 **gigabytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.

Ultimate Goal: wall-clock Time

Communication Cost Model

How to assess performance?

Communication Cost = input size +
(sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 **gigabits** per sec;
HD read: 50-150 **gigabytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.
- Output from reducer ignored because it's either small (finished summarizing data) or being passed to another mapreduce job.

Example: Natural Join

R_1, R_2 : Relations (Tables)

Communication Cost = input size +
(sum of size of all map-to-reducer files)

$$= |R| + |S| + (|R| + |S|)$$

$$= O(|R| + |S|)$$

Exercise:

Calculate Communication Cost for “Matrix Multiplication with One MapReduce Step”
(see MMDS section 2.3.10)

Last Notes

- Performance Refinements:
 - Backup tasks (aka speculative tasks)
 - Schedule multiple copies of tasks when close to the end to mitigate certain nodes running slow.
 - Combiners (like word count version 2)
 - Do some reducing from within map before passing to reduce
 - Reduces communication cost
 - Override partition hash function
E.g. instead of `hash(ur1)` use `hash(hostname(ur1))`