61A Lecture 35

Announcements

Unix

Computer Systems

Computer Systems

Systems research enables application development by defining and implementing abstractions:

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure
- **Databases** provide a declarative interface to complex software that stores and retrieves information efficiently

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure
- **Databases** provide a declarative interface to complex software that stores and retrieves information efficiently
- **Distributed systems** provide a unified interface to a cluster of multiple machines

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure
- **Databases** provide a declarative interface to complex software that stores and retrieves information efficiently
- **Distributed systems** provide a unified interface to a cluster of multiple machines

A unifying property of effective systems:

Computer Systems

Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure
- **Databases** provide a declarative interface to complex software that stores and retrieves information efficiently
- **Distributed systems** provide a unified interface to a cluster of multiple machines

A unifying property of effective systems:

Hide complexity, but retain flexibility

Example: The Unix Operating System

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

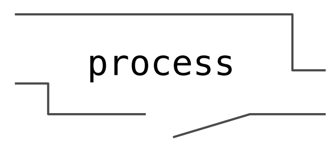
“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.

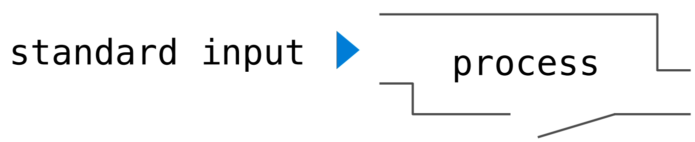


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.

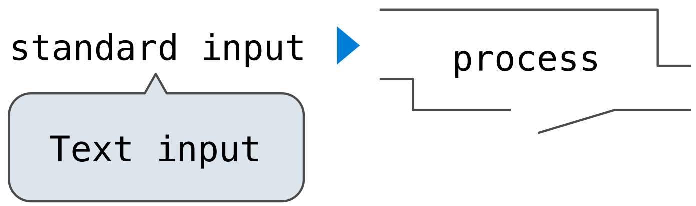


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.

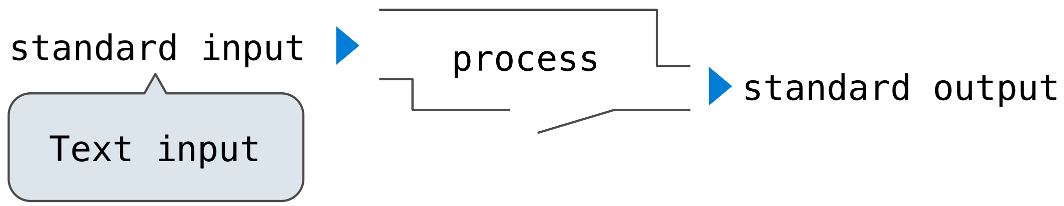


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.

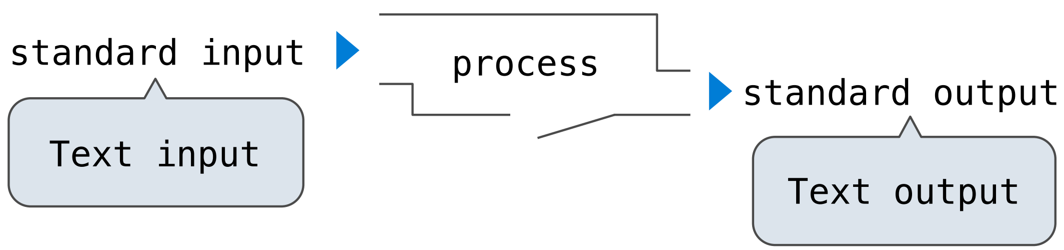


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

"We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way," Doug McIlroy in 1964.

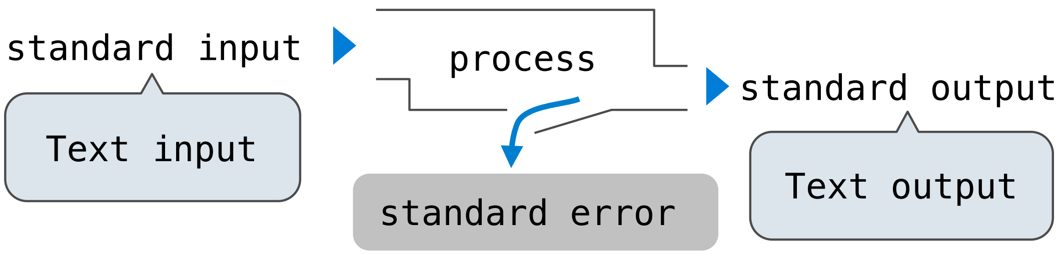


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

"We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way," Doug McIlroy in 1964.

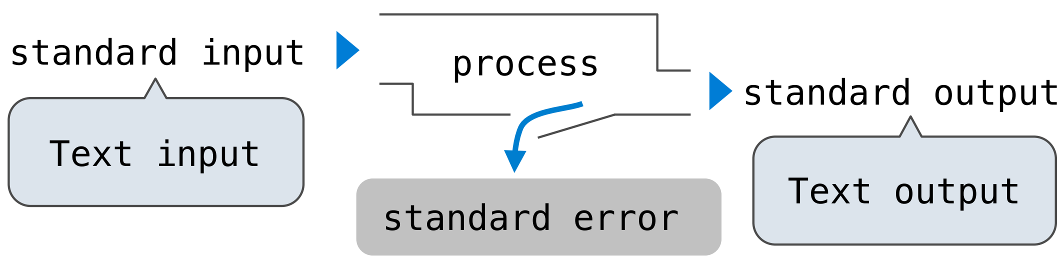


Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

"We should have some ways of coupling programs like [a] garden hose – screw in another segment when it becomes necessary to massage data in another way," Doug McIlroy in 1964.



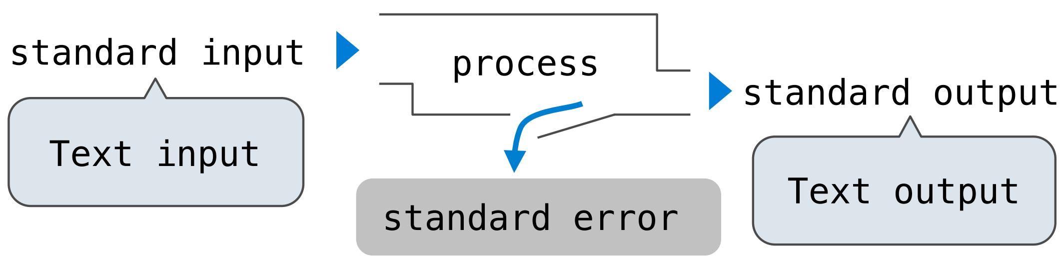
The standard streams in a Unix-like operating system are similar to Python iterators

Example: The Unix Operating System

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware
- **Multi-Tasking:** Many processes run concurrently on a machine
- **Plain Text:** Data is stored and shared in text format
- **Modularity:** Small tools are composed flexibly via pipes

“We should have some ways of coupling programs like [a] garden hose — screw in another segment when it becomes necessary to massage data in another way,” Doug McIlroy in 1964.



The standard streams in a Unix-like operating system are similar to Python iterators

(Demo)

```
cd ww/assets/slides && ls *.pdf | cut -f 1 -d - | sort -r | uniq -c
```

Python Programs in a Unix Environment

Python Programs in a Unix Environment

The **`sys.stdin`** and **`sys.stdout`** values provide access to the Unix standard streams as files

Python Programs in a Unix Environment

The **sys.stdin** and **sys.stdout** values provide access to the Unix standard streams as files

A Python file has an interface that supports iteration, **read**, and **write** methods

Python Programs in a Unix Environment

The **sys.stdin** and **sys.stdout** values provide access to the Unix standard streams as files

A Python file has an interface that supports iteration, **read**, and **write** methods

Using these "files" takes advantage of the operating system text processing abstraction

Python Programs in a Unix Environment

The **`sys.stdin`** and **`sys.stdout`** values provide access to the Unix standard streams as files

A Python file has an interface that supports iteration, **`read`**, and **`write`** methods

Using these "files" takes advantage of the operating system text processing abstraction

The **`input`** and **`print`** functions also read from standard input and write to standard output

Python Programs in a Unix Environment

The **sys.stdin** and **sys.stdout** values provide access to the Unix standard streams as files

A Python file has an interface that supports iteration, **read**, and **write** methods

Using these "files" takes advantage of the operating system text processing abstraction

The **input** and **print** functions also read from standard input and write to standard output

(Demo)

Big Data

Big Data Examples

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

Facebook datacenter (2014)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

  Consumer-grade hard disks and processors

Facebook datacenter (2014)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

- Consumer-grade hard disks and processors
- Independent computers are stored in racks

Facebook datacenter (2014)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

- Consumer-grade hard disks and processors
- Independent computers are stored in racks
- Concerns: networking, heat, power, monitoring

Facebook datacenter (2014)

Examples from Anthony Joseph

Big Data Examples

Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

1,000 genomes project: 200 Terabytes

Google web index: 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

- Consumer-grade hard disks and processors
- Independent computers are stored in racks
- Concerns: networking, heat, power, monitoring
- When using many computers, some will fail!

Facebook datacenter (2014)

Examples from Anthony Joseph

Apache Spark

Apache Spark

Apache Spark

```
Apache Spark is a data processing system that provides a simple interface for large data
```

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**
- Supports common sequence operations: **map**, **filter**, **reduce**

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**
- Supports common sequence operations: **map**, **filter**, **reduce**
- Supports common database operations: **join**, **union**, **intersection**

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**
- Supports common sequence operations: **map**, **filter**, **reduce**
- Supports common database operations: **join**, **union**, **intersection**

All of these operations can be performed on RDDs that are partitioned across machines

Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**
- Supports common sequence operations: **map**, **filter**, **reduce**
- Supports common database operations: **join**, **union**, **intersection**

All of these operations can be performed on RDDs that are partitioned across machines

Romeo & Juliet

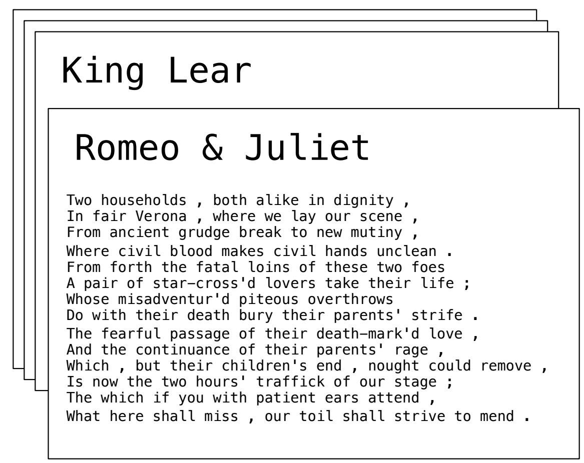
```
Two households , both alike in dignity ,
In fair Verona , where we lay our scene ,
From ancient grudge break to new mutiny ,
Where civil blood makes civil hands unclean .
From forth the fatal loins of these two foes
A pair of star-cross'd lovers take their life ;
Whose misadventur'd piteous overthrows
Do with their death bury their parents' strife .
The fearful passage of their death-mark'd love ,
And the continuance of their parents' rage ,
Which , but their children's end , nought could remove ,
Is now the two hours' traffick of our stage ;
The which if you with patient ears attend ,
What here shall miss , our toil shall strive to mend .
```

Apache Spark

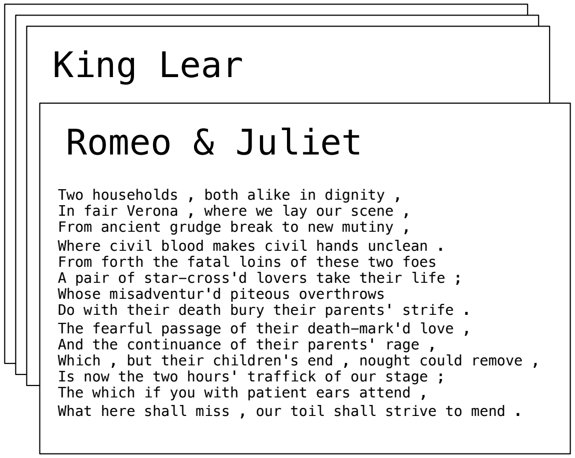
Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: **sort**, **distinct** (**uniq** in UNIX), **count**, **pipe**
- Supports common sequence operations: **map**, **filter**, **reduce**
- Supports common database operations: **join**, **union**, **intersection**

All of these operations can be performed on RDDs that are partitioned across machines

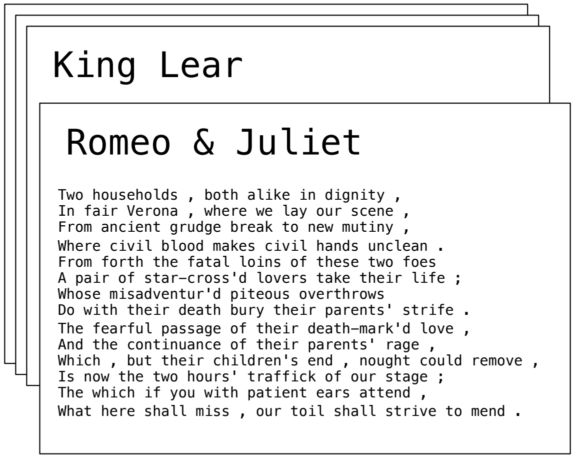


Apache Spark Execution Model



Apache Spark Execution Model

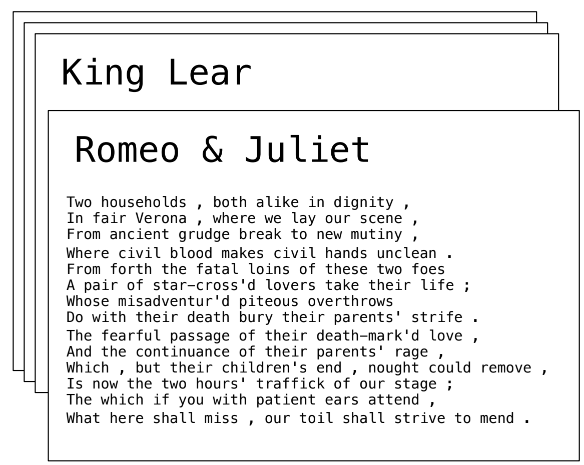
Processing is defined centrally but executed remotely



Apache Spark Execution Model

Processing is defined centrally but executed remotely

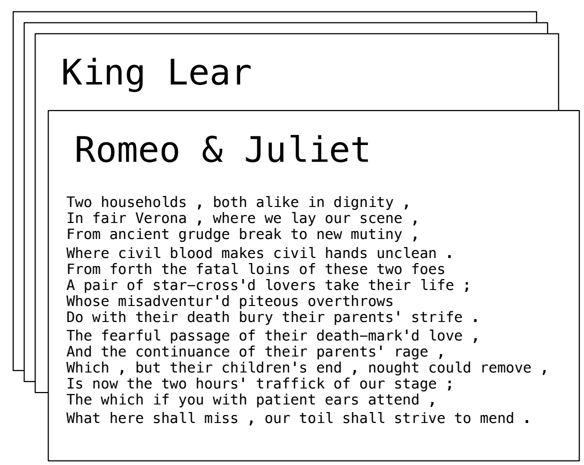
- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*



Apache Spark Execution Model

Processing is defined centrally but executed remotely

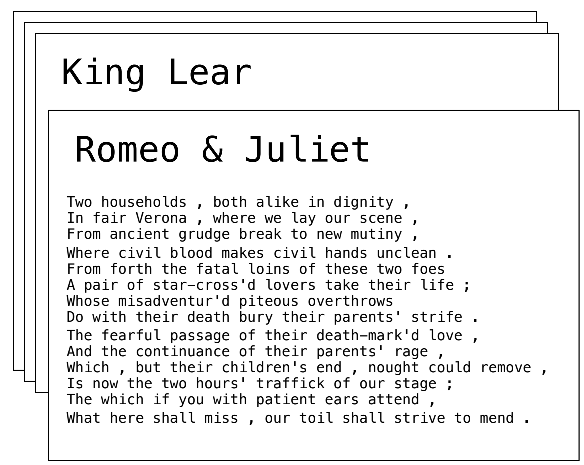
- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions on an RDD



Apache Spark Execution Model

Processing is defined centrally but executed remotely

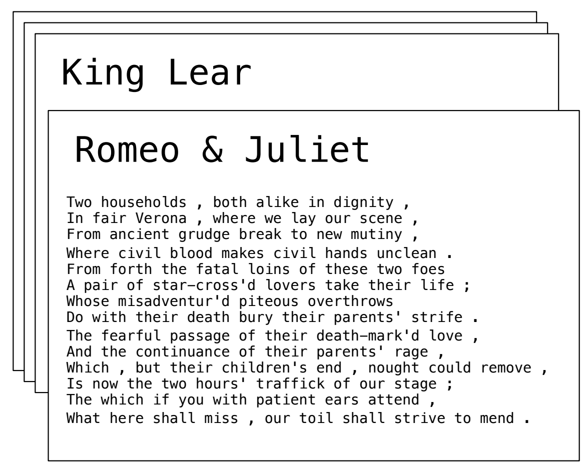
- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions on an RDD
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out



Apache Spark Execution Model

Processing is defined centrally but executed remotely

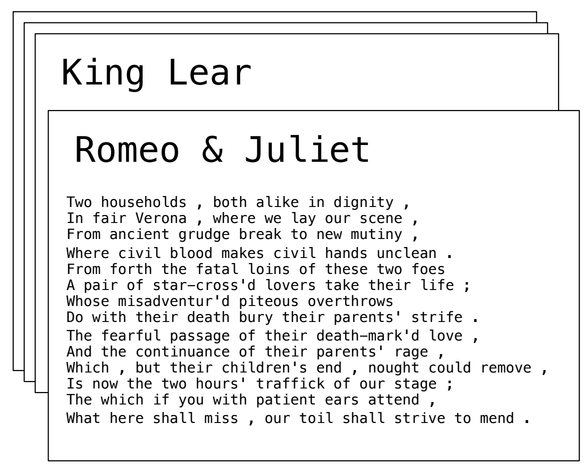
- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions on an RDD
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out
- Worker nodes perform computation & communicate values to each other



Apache Spark Execution Model

Processing is defined centrally but executed remotely

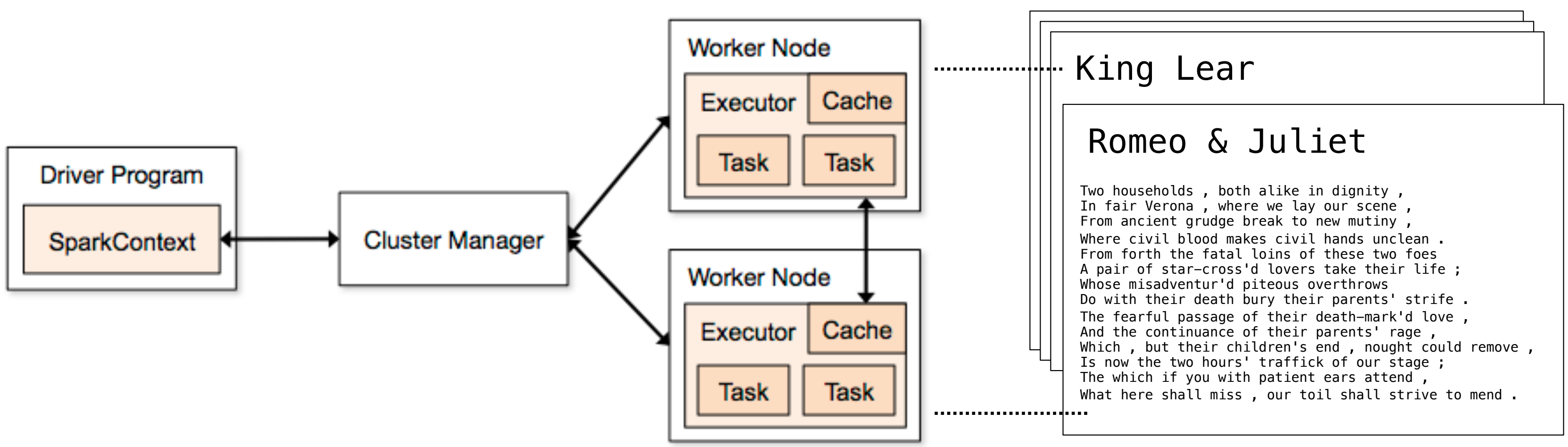
- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions on an RDD
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out
- Worker nodes perform computation & communicate values to each other
- Final results are communicated back to the driver program



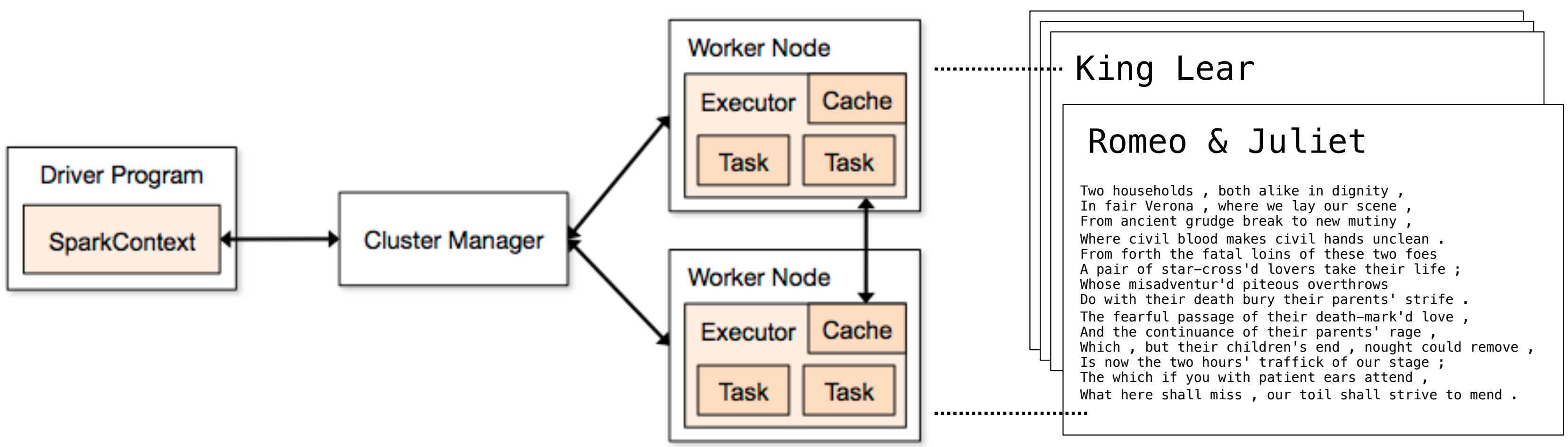
Apache Spark Execution Model

Processing is defined centrally but executed remotely

- A Resilient Distributed Dataset (RDD) is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions on an RDD
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out
- Worker nodes perform computation & communicate values to each other
- Final results are communicated back to the driver program

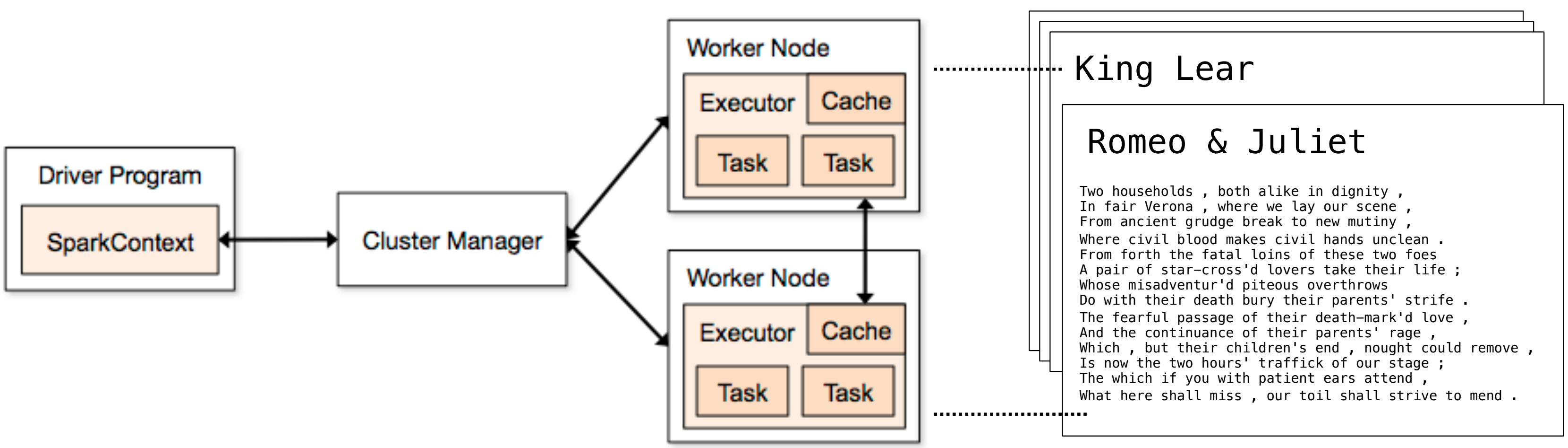


Apache Spark Interface



Apache Spark Interface

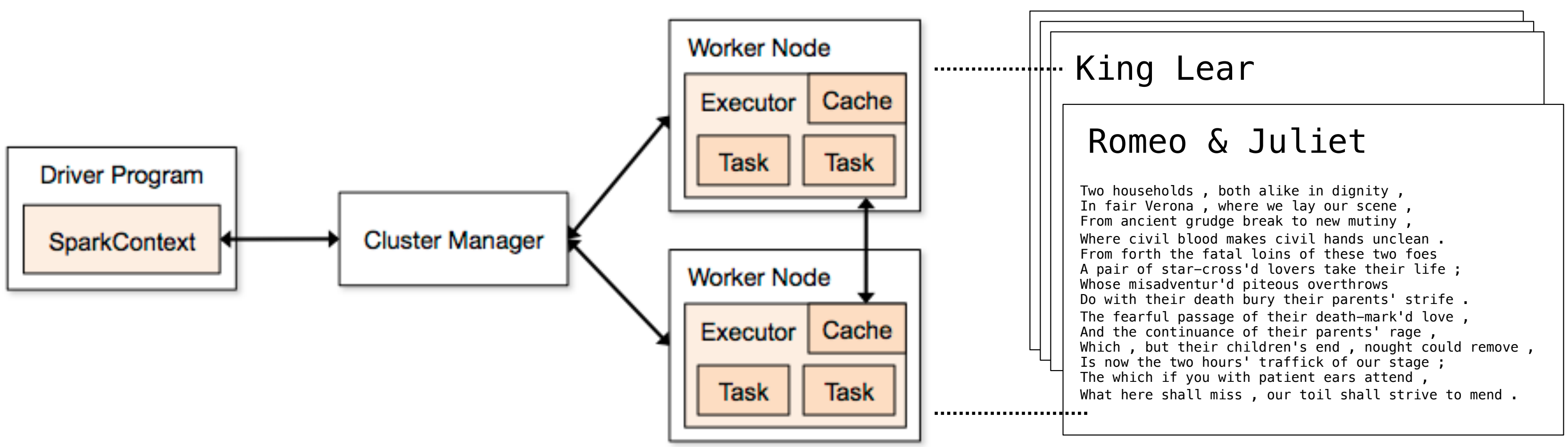
```
The Last Words of Shakespeare (Demo)
```



Apache Spark Interface

The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

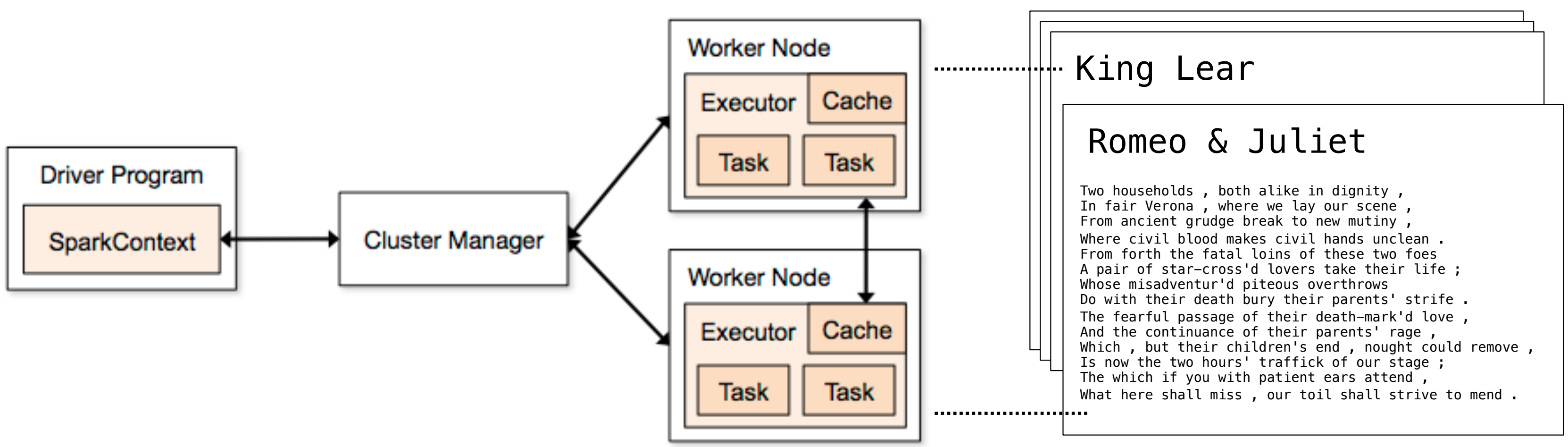


Apache Spark Interface

The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

```
>>> sc
<pyspark.context.SparkContext ...>
```



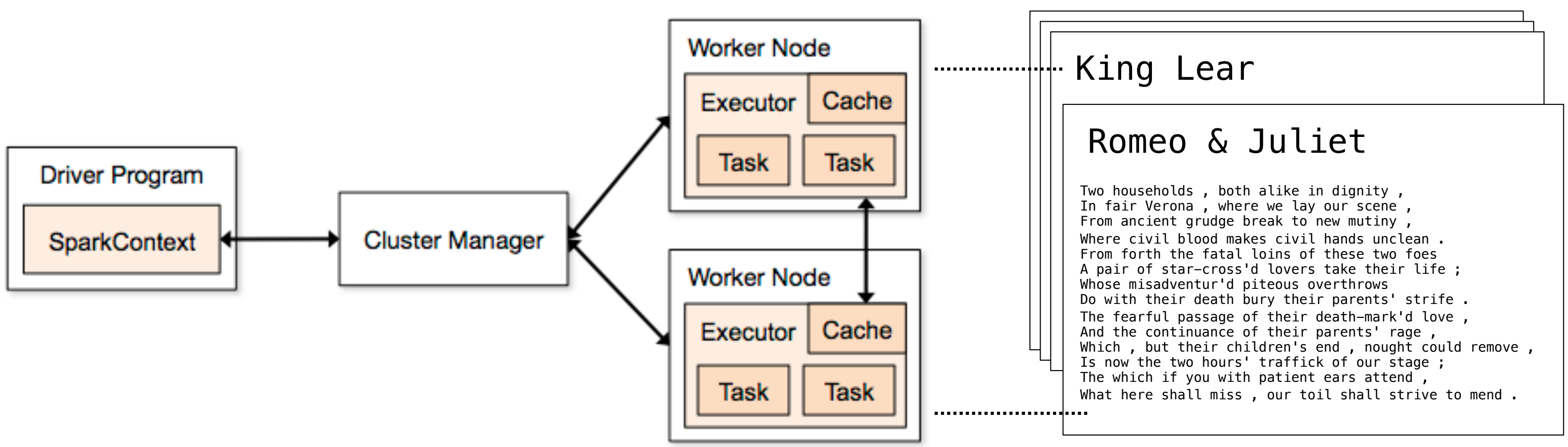
Apache Spark Interface

The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

A RDD can be constructed from the lines of a text file

```
>>> sc
<pyspark.context.SparkContext ...>
```



Apache Spark Interface

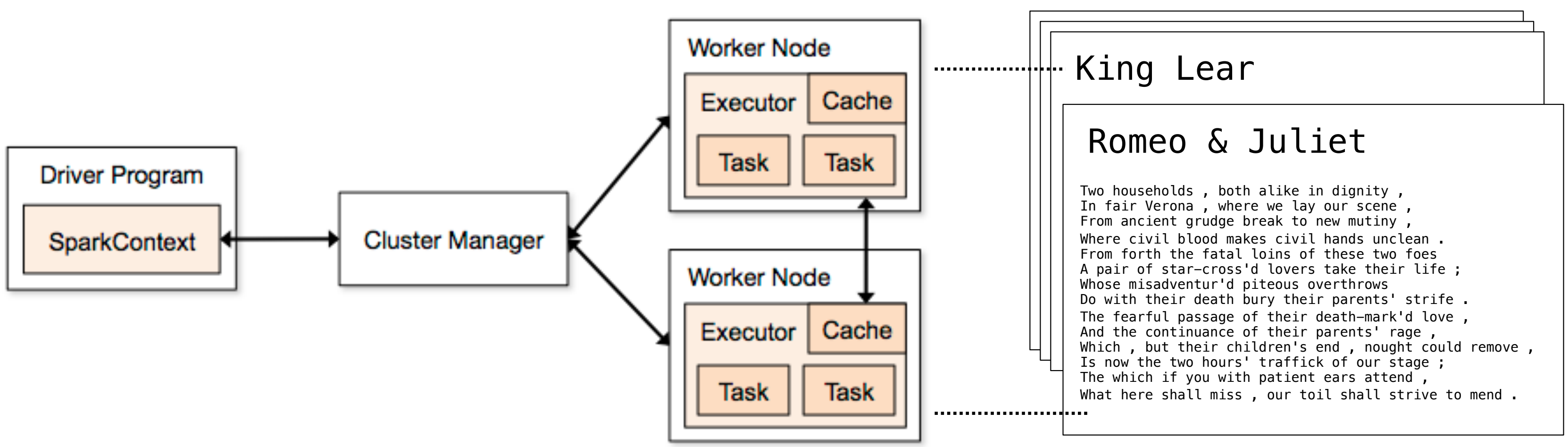
The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

```
>>> sc
<pyspark.context.SparkContext ...>
```

A RDD can be constructed from the lines of a text file

```
>>> x = sc.textFile('shakespeare.txt')
```



Apache Spark Interface

The Last Words of Shakespeare (Demo)

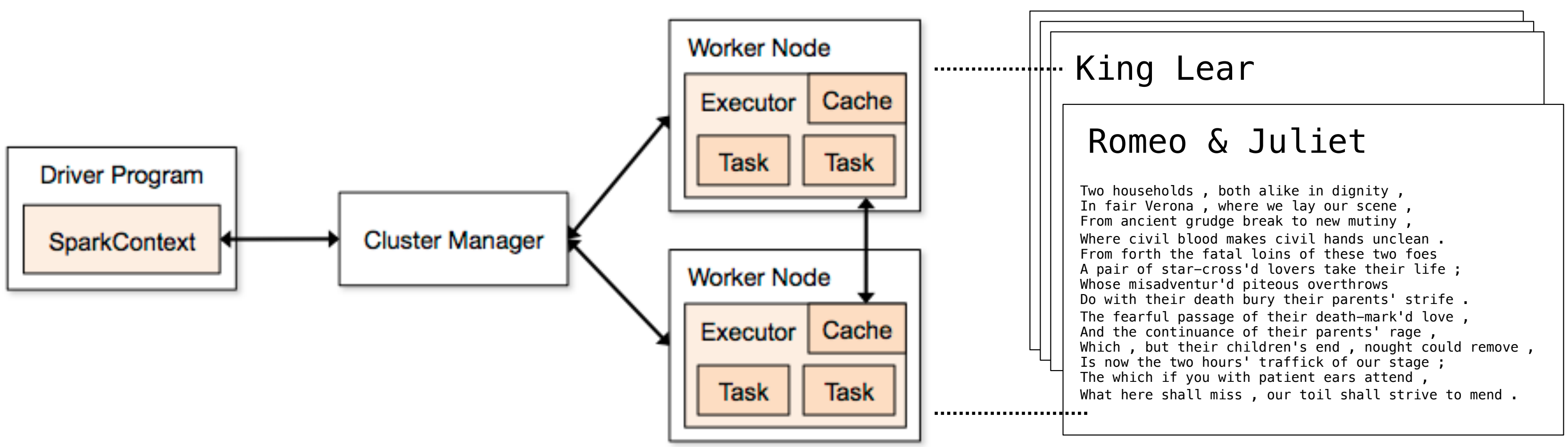
A **SparkContext** gives access to the cluster manager

```
>>> sc
<pyspark.context.SparkContext ...>
```

A RDD can be constructed from the lines of a text file

```
>>> x = sc.textFile('shakespeare.txt')
```

The **sortBy** transformation and **take** action are methods



Apache Spark Interface

The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

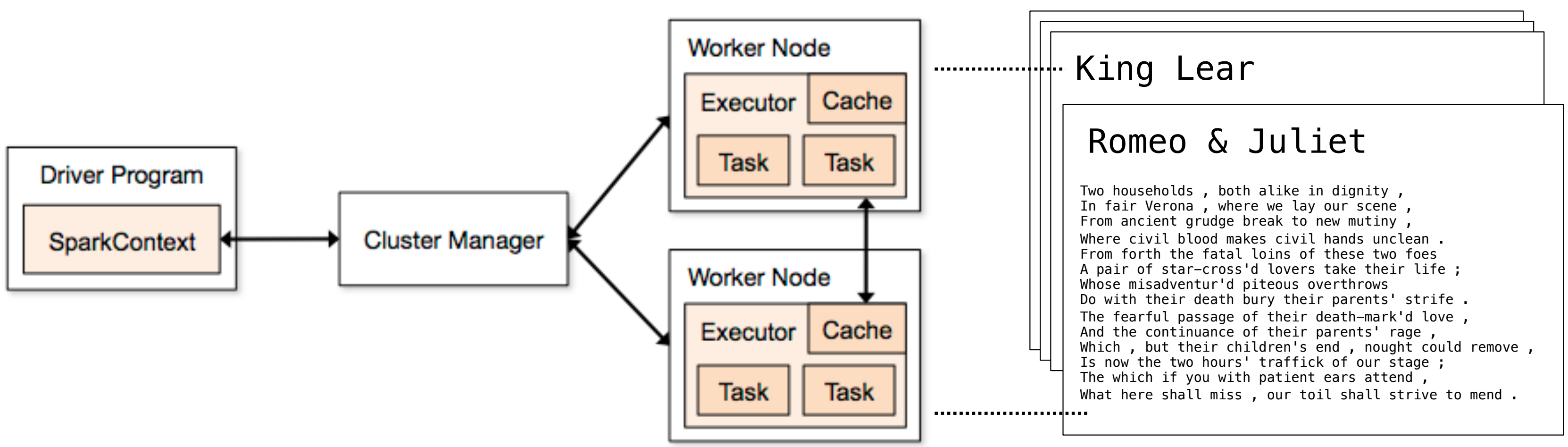
```
>>> sc
<pyspark.context.SparkContext ...>
```

A RDD can be constructed from the lines of a text file

```
>>> x = sc.textFile(                 )
```

The **sortBy** transformation and **take** action are methods

```
>>> x.sortBy(lambda s: s, False).take(2)
['you shall ...', 'yet , a ...']
```



Apache Spark Interface

The Last Words of Shakespeare (Demo)

A **SparkContext** gives access to the cluster manager

```
>>> sc
<pyspark.context.SparkContext ...>
```

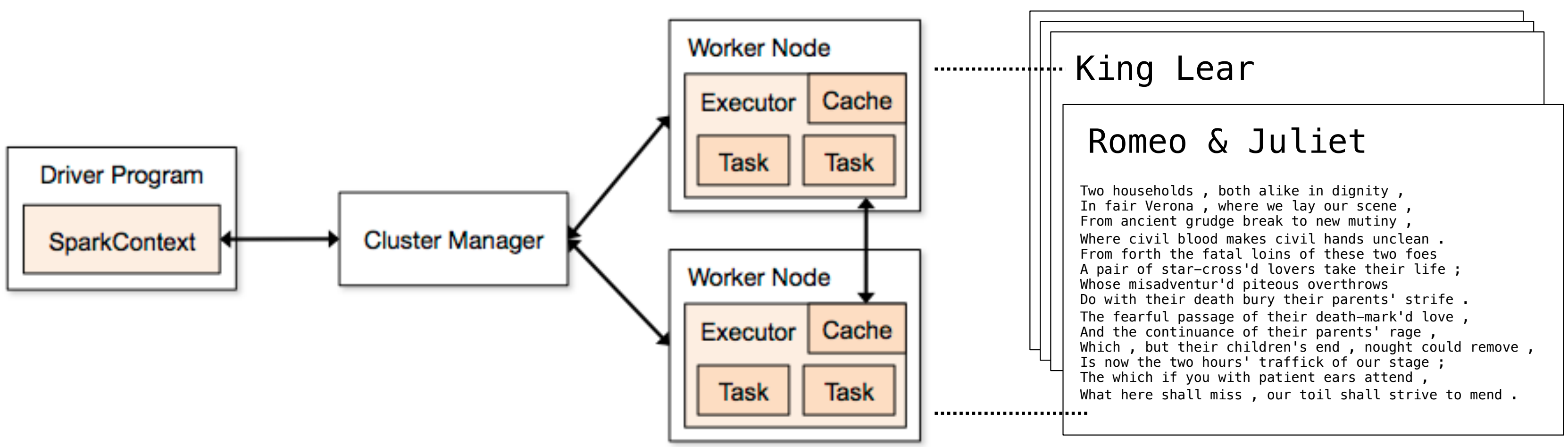
A RDD can be constructed from the lines of a text file

```
>>> x = sc.textFile(                  )
```

The **sortBy** transformation and **take** action are methods

```
>>> x.sortBy(lambda s: s, False).take(2)
['you shall ...', 'yet , a ...']
```

(Demo)



What Does Apache Spark Provide?

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive

- The cluster manager tries to schedule computation on the machines that hold the data to be processed

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive

- The cluster manager tries to schedule computation on the machines that hold the data to be processed

Monitoring: Will my job finish before dinner?!?

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive

- The cluster manager tries to schedule computation on the machines that hold the data to be processed

Monitoring: Will my job finish before dinner?!?

- The cluster manager provides a web-based interface describing jobs

What Does Apache Spark Provide?

Fault tolerance: A machine or hard drive might crash

- The cluster manager automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded

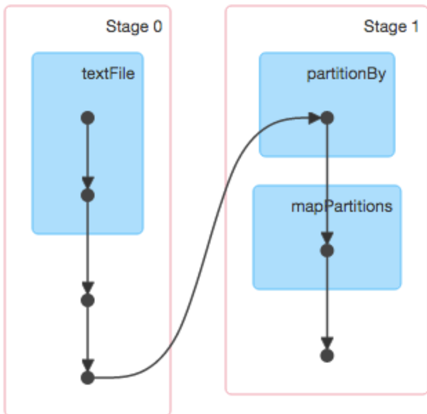
- The cluster manager can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive

- The cluster manager tries to schedule computation on the machines that hold the data to be processed

Monitoring: Will my job finish before dinner?!?

- The cluster manager provides a web-based interface describing jobs



MapReduce

MapReduce Applications

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

- Step 1: Each element in an input collection produces zero or more key-value pairs (map)

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

- Step 1: Each element in an input collection produces zero or more key-value pairs (map)
- Step 2: All key-value pairs that share a key are aggregated together (shuffle)

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

- Step 1: Each element in an input collection produces zero or more key-value pairs (map)
- Step 2: All key-value pairs that share a key are aggregated together (shuffle)
- Step 3: The values for a key are processed as a sequence (reduce)

MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

- Step 1: Each element in an input collection produces zero or more key-value pairs (map)
- Step 2: All key-value pairs that share a key are aggregated together (shuffle)
- Step 3: The values for a key are processed as a sequence (reduce)

Early applications: indexing web pages, training language models, & computing PageRank

MapReduce Evaluation Model

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

```
Google MapReduce
Is a Big Data framework
For batch processing
```

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

```
Google MapReduce
Is a Big Data framework
For batch processing
```

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

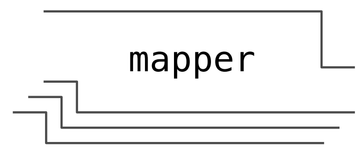
- The mapper yields zero or more key-value pairs for each input

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

```
Google MapReduce
Is a Big Data framework
For batch processing
```

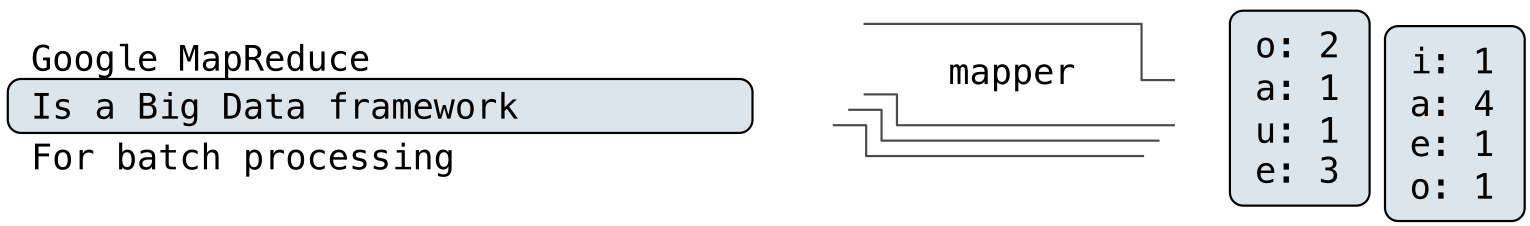



```
o: 2
a: 1
u: 1
e: 3
```

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

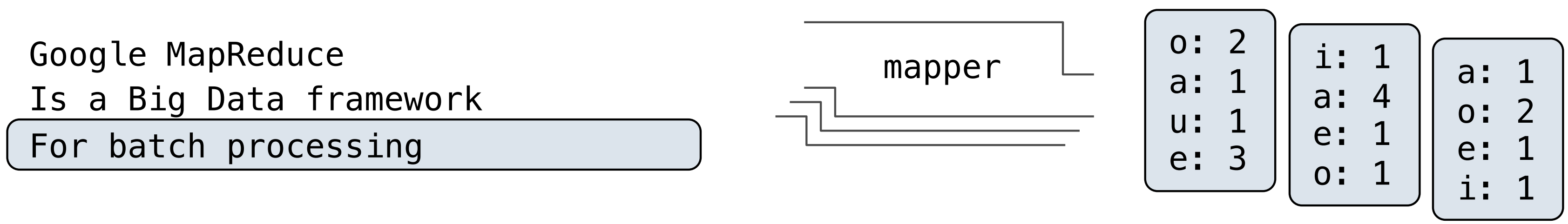
- The mapper yields zero or more key-value pairs for each input



MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

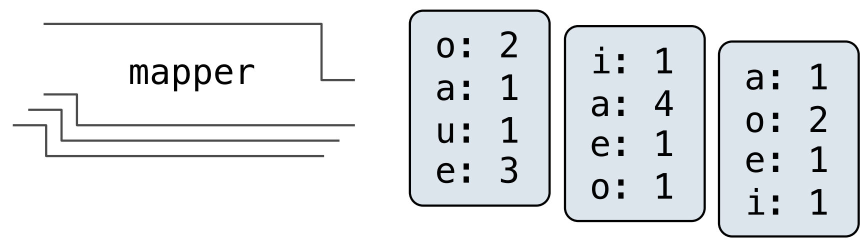


MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

Google MapReduce
Is a Big Data framework
For batch processing

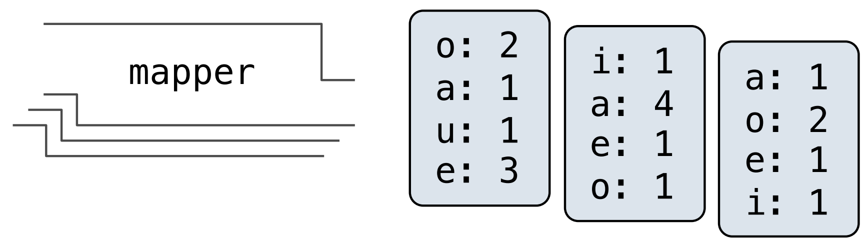


MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

Google MapReduce
Is a Big Data framework
For batch processing



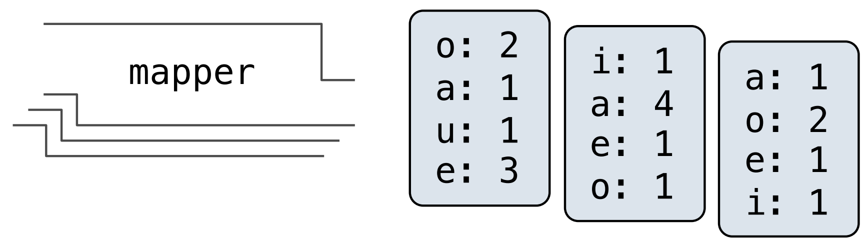
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

Google MapReduce
Is a Big Data framework
For batch processing



Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

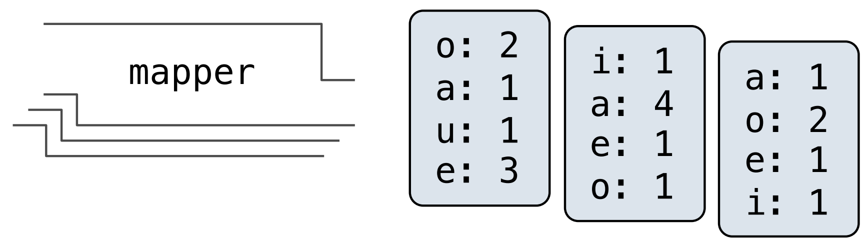
- All key-value pairs with the same key are processed together

MapReduce Evaluation Model

Map phase: Apply a *mapper* function to all inputs, emitting intermediate key-value pairs

- The mapper yields zero or more key-value pairs for each input

Google MapReduce
Is a Big Data framework
For batch processing

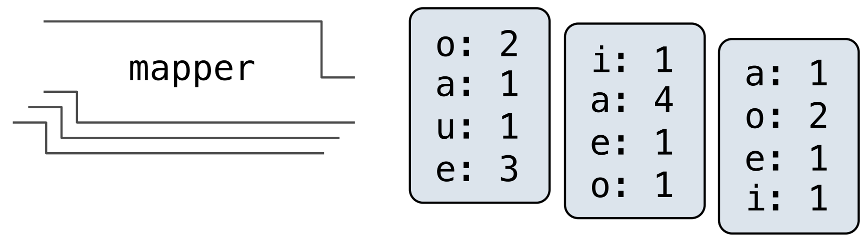


Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key

MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key

MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing


```
o: 2    i: 1    a: 1
a: 1    a: 4    o: 2
u: 1    e: 1    e: 1
e: 3    o: 1    i: 1
```

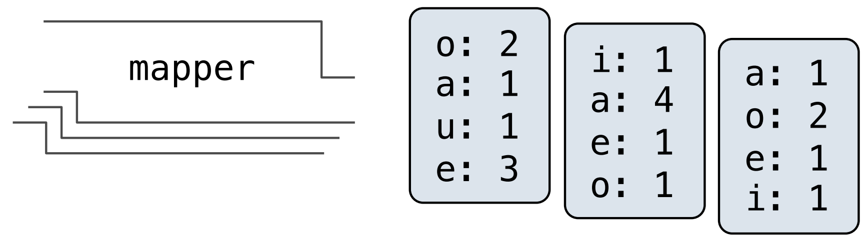
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key

```
a: 4
a: 1
a: 1
e: 1
e: 3
e: 1
...
```

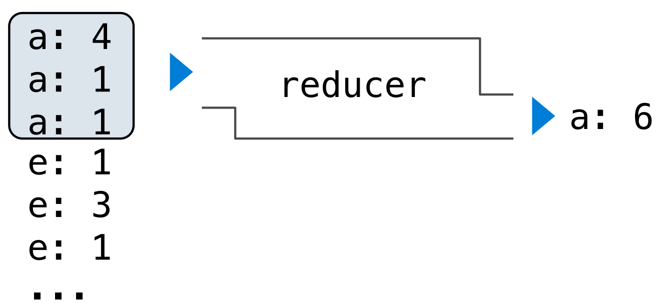
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



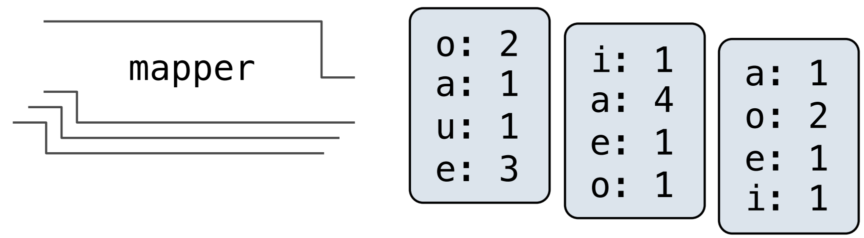
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



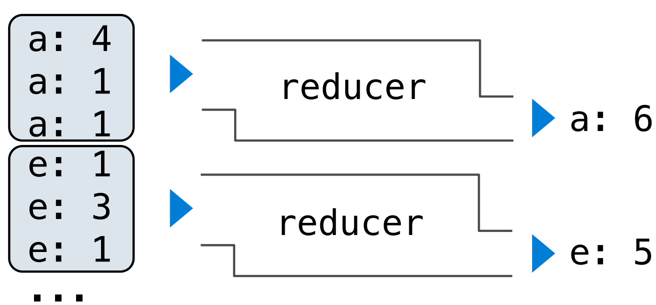
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



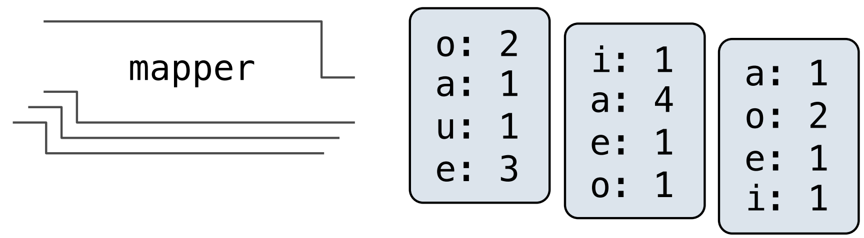
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



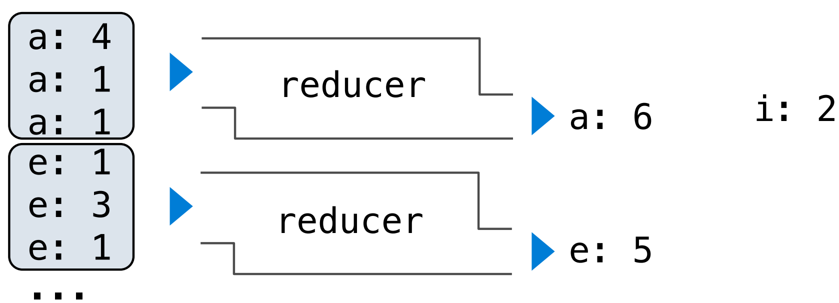
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



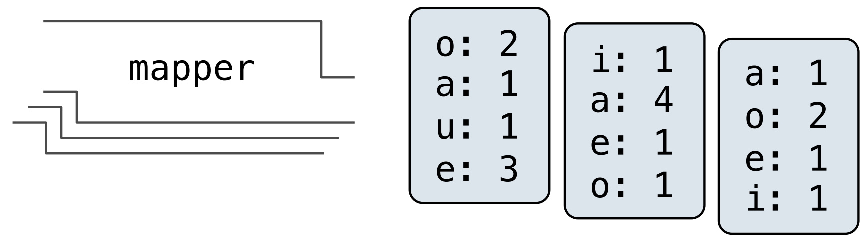
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



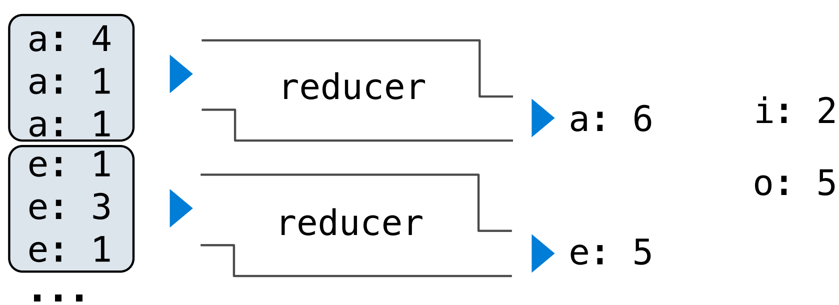
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



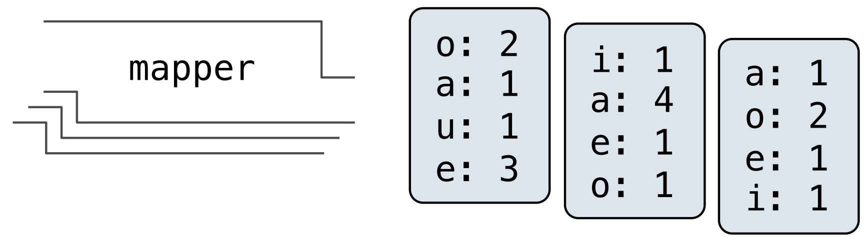
Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



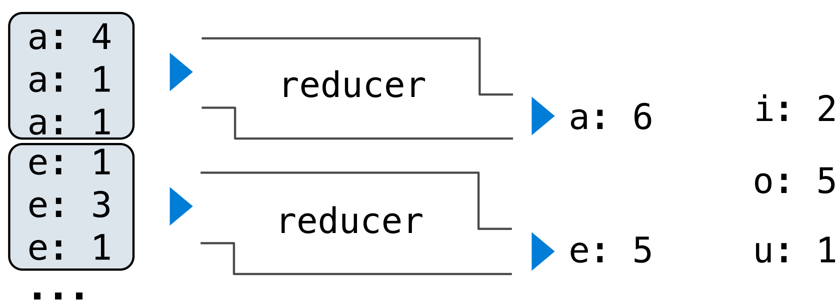
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



MapReduce Applications on Apache Spark

MapReduce Applications on Apache Spark

```
Key-value pairs are just two-element Python tuples
```

MapReduce Applications on Apache Spark

```
Key-value pairs are just two-element Python tuples
```

```
data.flatMap(fn)
```

MapReduce Applications on Apache Spark

```
Key-value pairs are just two-element Python tuples
```

```
data.flatMap(fn)
```

```
data.reduceByKey(fn)
```

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression

```
data.flatMap(fn)
```

```
data.reduceByKey(fn)
```

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**
`data.flatMap(fn)`	
`data.reduceByKey(fn)`	

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**
`data.flatMap(fn)`		
`data.reduceByKey(fn)`		

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	Data	fn Input	fn Output
data.**flatMap**(fn)			
data.**reduceByKey**(fn)			

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	Data	fn Input	fn Output	Result
`data.flatMap(fn)`				
`data.reduceByKey(fn)`				

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	Data	fn Input	fn Output	Result
`data.flatMap(fn)`	Values			
`data.reduceByKey(fn)`				

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	Data	fn Input	fn Output	Result
`data.flatMap(fn)`	Values	One value		
`data.reduceByKey(fn)`				

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	Data	fn Input	fn Output	Result
`data.flatMap(fn)`	Values	One value	Zero or more key-value pairs	
`data.reduceByKey(fn)`				

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
`data.flatMap(fn)`	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
`data.reduceByKey(fn)`				

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
`data.flatMap(fn)`	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
`data.reduceByKey(fn)`	Key-value pairs			

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
data.**flatMap**(fn)	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
data.**reduceByKey**(fn)	Key-value pairs	Two values		

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
data.**flatMap**(fn)	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
data.**reduceByKey**(fn)	Key-value pairs	Two values	One value	

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
`data.flatMap(fn)`	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
`data.reduceByKey(fn)`	Key-value pairs	Two values	One value	One key-value pair for each unique key

MapReduce Applications on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression	**Data**	**fn Input**	**fn Output**	**Result**
data.**flatMap**(fn)	Values	One value	Zero or more key-value pairs	All key-value pairs returned by calls to fn
data.**reduceByKey**(fn)	Key-value pairs	Two values	One value	One key-value pair for each unique key

(Demo)