Lecture 26: Parallelism

Brian Hou
August 4, 2016

Announcements

Announcements

- Project 4 is due tomorrow (8/5)

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)

Announcements

- Project 4 is due tomorrow (8/5)
    - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
    - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9
- **Potluck II** on 8/10! 5–8pm (or later) in Wozniak Lounge

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9
- **Potluck II** on 8/10! 5–8pm (or later) in Wozniak Lounge
  - Bring food and board games!

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9
- **Potluck II** on 8/10! 5–8pm (or later) in Wozniak Lounge
  - Bring food and board games!
- Homework 10 will be due 8/9

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11-12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5-8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9
- **Potluck II** on 8/10! 5-8pm (or later) in Wozniak Lounge
  - Bring food and board games!
- Homework 10 will be due 8/9
- Homework 11 and 12 will be due 8/10 and 8/12

Announcements

- Project 4 is due tomorrow (8/5)
  - Submit by today for 1 EC point
- Final Review tomorrow (8/5) from 11–12:30pm in 2050 VLSB
  - Final Exam on Friday (8/12) from 5–8pm in 155 Dwinelle
- Ants composition revisions due Saturday (8/6)
- Scheme Recursive Art Contest is open! Submissions due 8/9
- **Potluck II** on 8/10! 5–8pm (or later) in Wozniak Lounge
  - Bring food and board games!
- Homework 10 will be due 8/9
- Homework 11 and 12 will be due 8/10 and 8/12
  - Last two of the three extra credit surveys

Roadmap

- Introduction
- Functions
- Data
- Mutability
- Objects
- Interpretation
- Paradigms
- Applications

Roadmap

Introduction

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications

- This week (Paradigms), the goals are:

Roadmap

- Introduction
- Functions
- Data
- Mutability
- Objects
- Interpretation
- Paradigms
- Applications

- This week (Paradigms), the goals are:
  - To study examples of paradigms that are very different from what we have seen so far

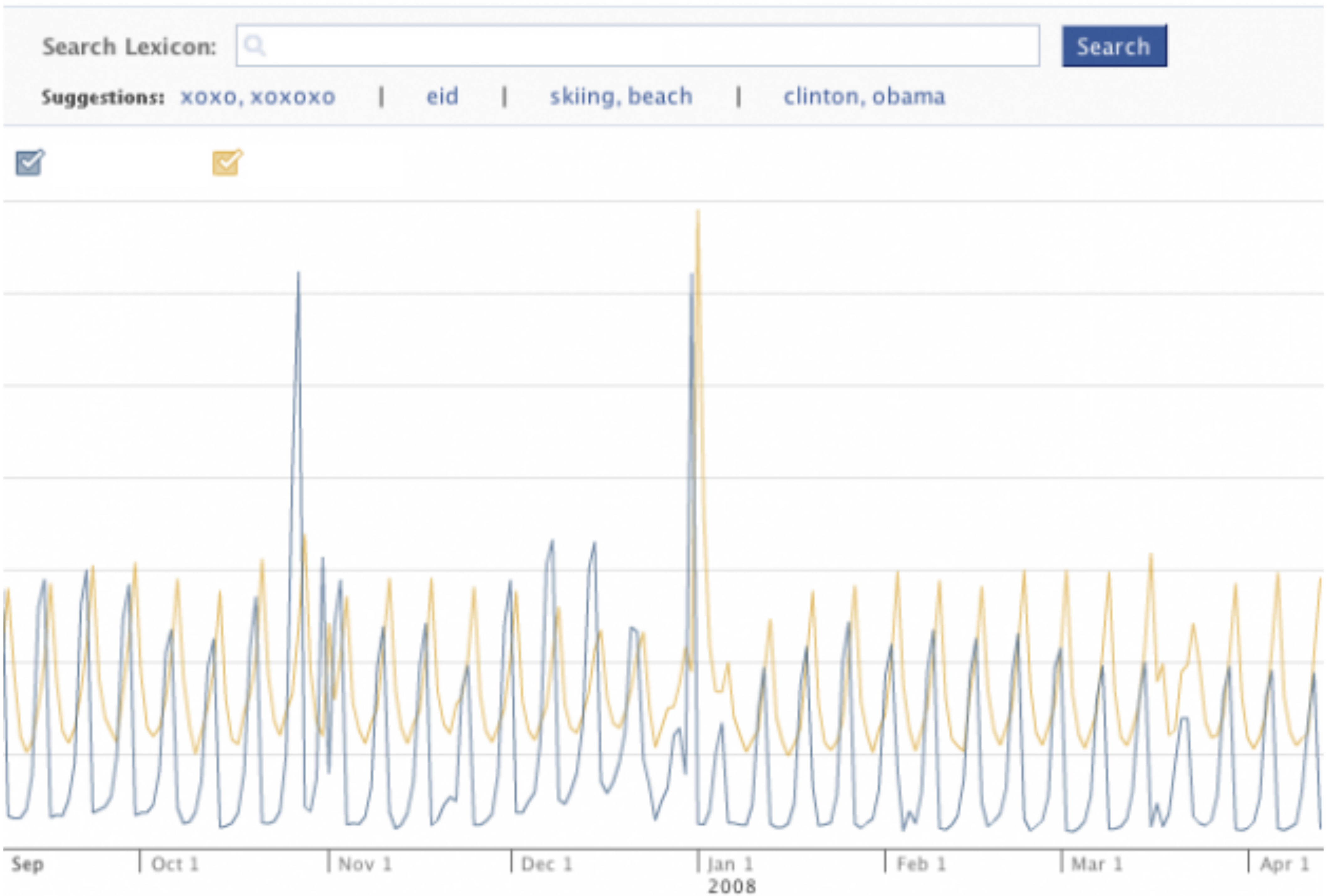
Roadmap

- Introduction
- Functions
- Data
- Mutability
- Objects
- Interpretation
- Paradigms
- Applications

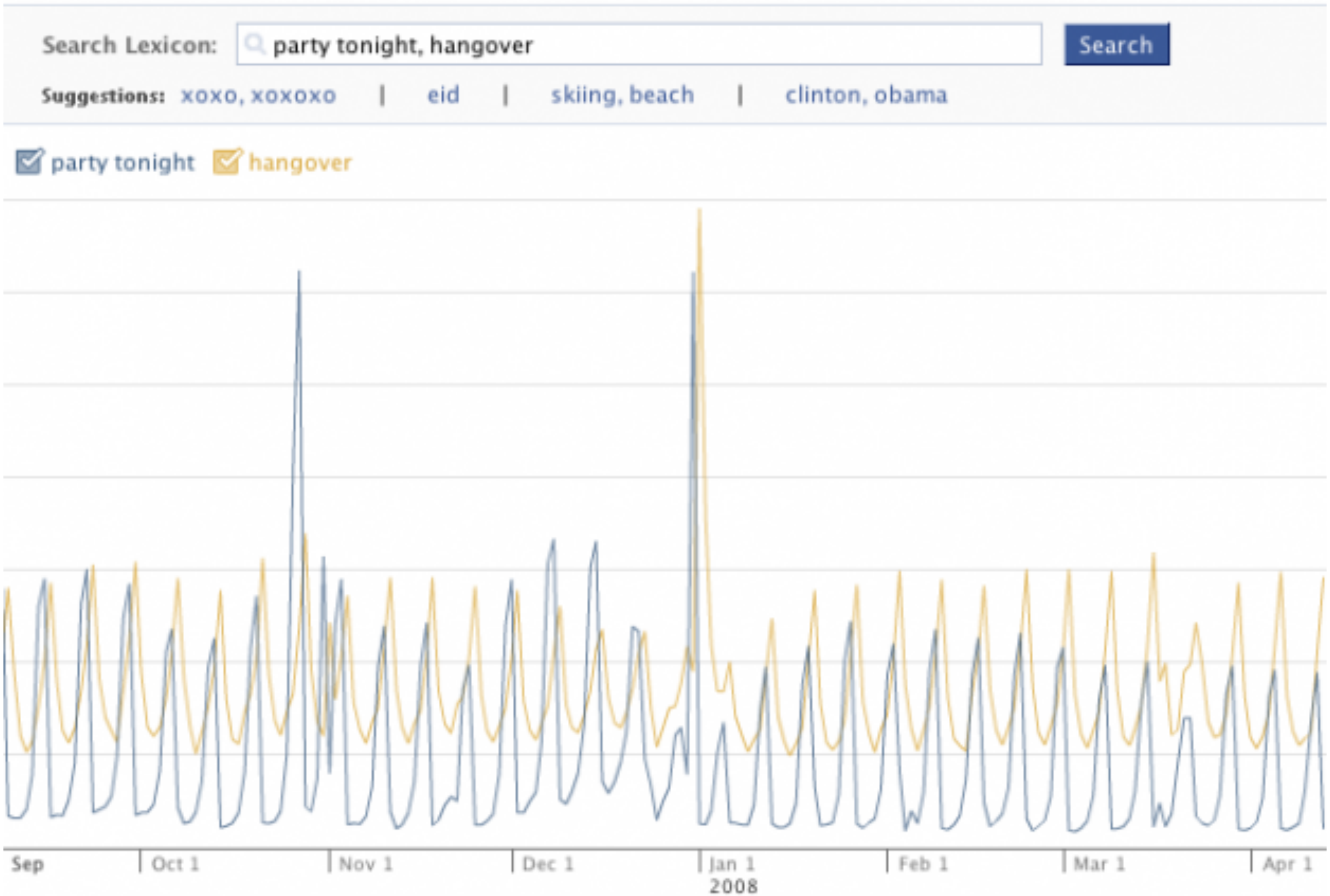
- This week (Paradigms), the goals are:
  - To study examples of paradigms that are very different from what we have seen so far
  - To expand our definition of what counts as programming

Big Data

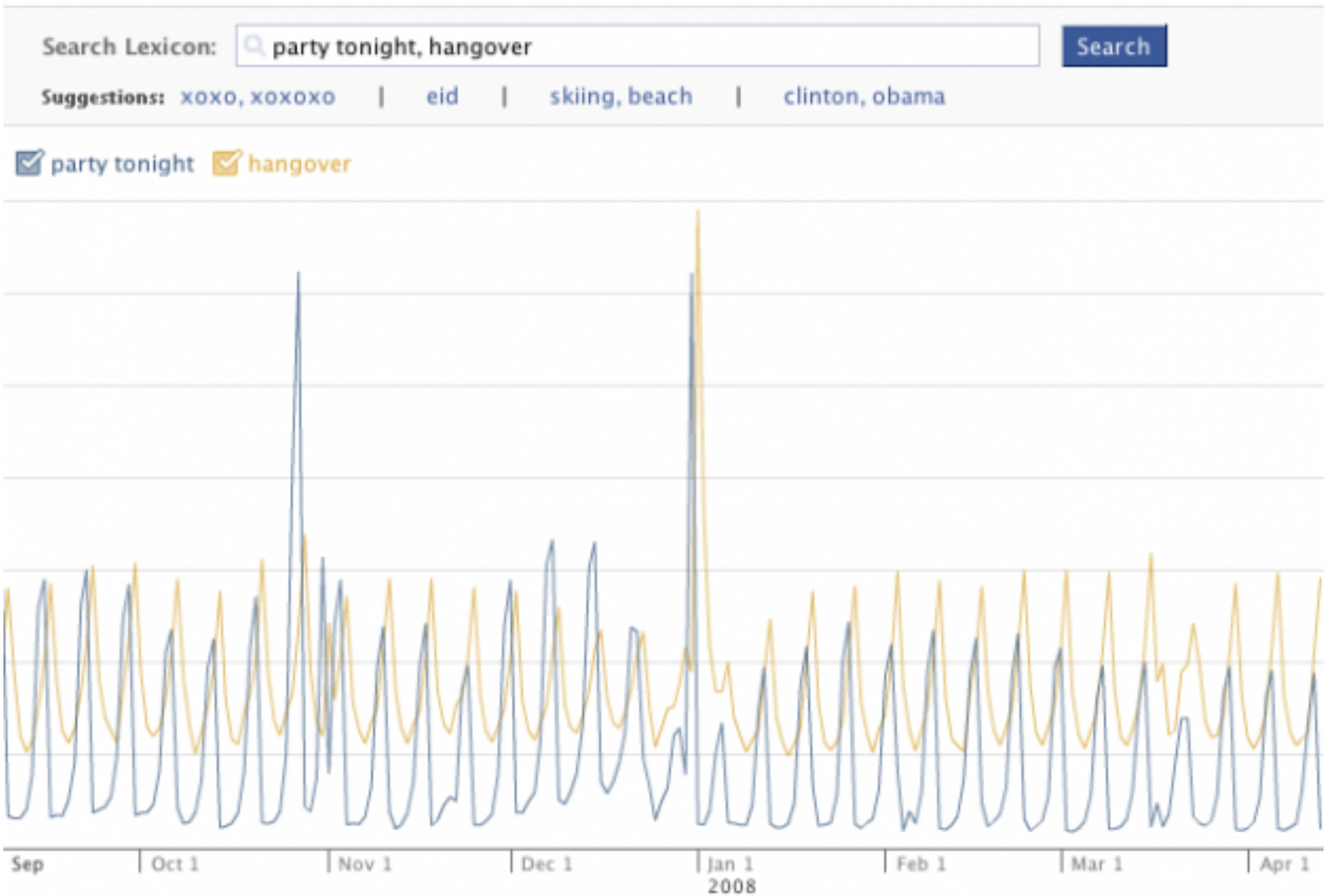
Facebook Lexicon



Facebook Lexicon



Facebook Lexicon



(discontinued)

Examples of Big Data

Examples from Anthony Joseph

Examples of Big Data

- There's a lot of data out there!

Examples from Anthony Joseph

Examples of Big Data

- There's a lot of data out there!
  - Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)

Examples from Anthony Joseph

Examples of Big Data

- There's a lot of data out there!
  - Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)
  - 1,000 genomes project: 200 Terabytes

Examples from Anthony Joseph

Examples of Big Data

- There's a lot of data out there!
  - 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

Examples of Big Data

- There's a lot of data out there!
  - Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)
  - 1,000 genomes project: 200 Terabytes
  - Google web index: 10+ Petabytes (10,000,000 Gigabytes!!)
- These datasets are too large to fit on a single computer

Examples from Anthony Joseph

Examples of Big Data

- There's a lot of data out there!
    - Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)
    - 1,000 genomes project: 200 Terabytes
    - Google web index: 10+ Petabytes (10,000,000 Gigabytes!!)
- These datasets are too large to fit on a single computer
- Reading 1 Terabyte from disk: 3 hours (100 MB per second)

Examples from Anthony Joseph

Distributed Algorithms

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment

Computers for Big Data

Computers for Big Data

- Typical hardware for big data applications:

Computers for Big Data

- Typical hardware for big data applications:
  - Consumer-grade hard disks and processors

Computers for Big Data

- Typical hardware for big data applications:
  - Consumer-grade hard disks and processors
  - Independent computers are stored in racks

Computers for Big Data

- Typical hardware for big data applications:
  - Consumer-grade hard disks and processors
  - Independent computers are stored in racks

Facebook datacenter (2014)

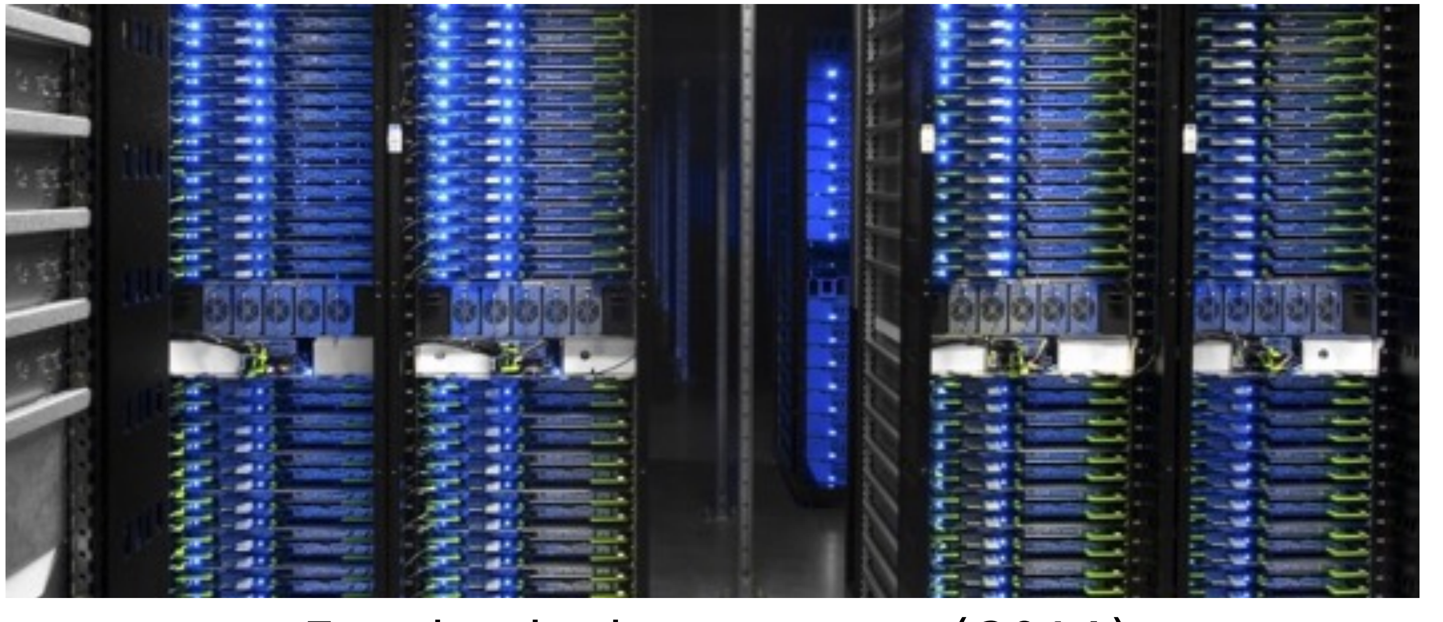
Computers for Big Data

- Typical hardware for big data applications:
  - Consumer-grade hard disks and processors
  - Independent computers are stored in racks
- Concerns: heat, power, monitoring, networking

Facebook datacenter (2014)

Computers for Big Data

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



Facebook datacenter (2014)

Distributed Algorithms

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment
- Machines and networks occasionally fail!

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment
- Machines and networks occasionally fail!
  - Lost work must be recomputed

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment
- Machines and networks occasionally fail!
  - Lost work must be recomputed
- Slow workers should be detected and their task should be given to a different worker

Distributed Algorithms

- If data can't be stored on a single machine, then our programs can't run on a single machine
- Therefore, we need to develop *distributed algorithms* to distribute and coordinate work between worker machines
- Machines can communicate, but perform computations in their own isolated environment
- Machines and networks occasionally fail!
  - Lost work must be recomputed
- Slow workers should be detected and their task should be given to a different worker
- This is getting complicated...

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
  - Developed right here at Berkeley in 2010!

Apache Spark

- Apache Spark is a data processing system that provides a simple interface for large data
  - Developed right here at Berkeley in 2010!
- 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
  - Developed right here at Berkeley in 2010!
- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common sequence operations: map, filter, reduce

Apache Spark

- Apache Spark is a data processing system that provides a simple interface for large data
  - Developed right here at Berkeley in 2010!
- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common sequence operations: map, filter, reduce
  - 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
  - Developed right here at Berkeley in 2010!
- A Resilient Distributed Dataset (RDD) is a collection of values or key–value pairs
- Supports common sequence operations: map, filter, reduce
  - These operations can be performed on RDDs that are partitioned across machines
- Idea: Working with distributed data is complicated. **Use abstraction** to hide the fact that the data is distributed!

Apache Spark Execution Model

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions
  - Transformations: Create a new RDD from an existing RDD

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions
  - Transformations: Create a new RDD from an existing RDD
  - Actions: Summarize RDD into one value (e.g. sum, take)

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions
  - Transformations: Create a new RDD from an existing RDD
  - Actions: Summarize RDD into one value (e.g. sum, take)
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions
  - Transformations: Create a new RDD from an existing RDD
  - Actions: Summarize RDD into one value (e.g. sum, take)
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out
- Worker nodes perform computation and communicate values to each other

Apache Spark Execution Model

- An RDD is distributed in partitions to *worker nodes*
- A *driver program* defines transformations and actions
  - Transformations: Create a new RDD from an existing RDD
  - Actions: Summarize RDD into one value (e.g. sum, take)
- A *cluster manager* assigns tasks to individual *worker nodes* to carry them out
- Worker nodes perform computation and communicate values to each other
- Final results are communicated back to the driver program

The Last Words of Shakespeare

The Last Words of Shakespeare

- A *driver program* defines transformations and actions
- A *cluster manager* assigns tasks to individual *worker nodes*
- Worker nodes perform computation and communicate values to each other

The Last Words of Shakespeare

- A *driver program* defines transformations and actions
- A *cluster manager* assigns tasks to individual *worker nodes*
- Worker nodes perform computation and communicate values to each other

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 .
```

The Last Words of Shakespeare

- A *driver program* defines transformations and actions
- A *cluster manager* assigns tasks to individual *worker nodes*
- Worker nodes perform computation and communicate values to each other

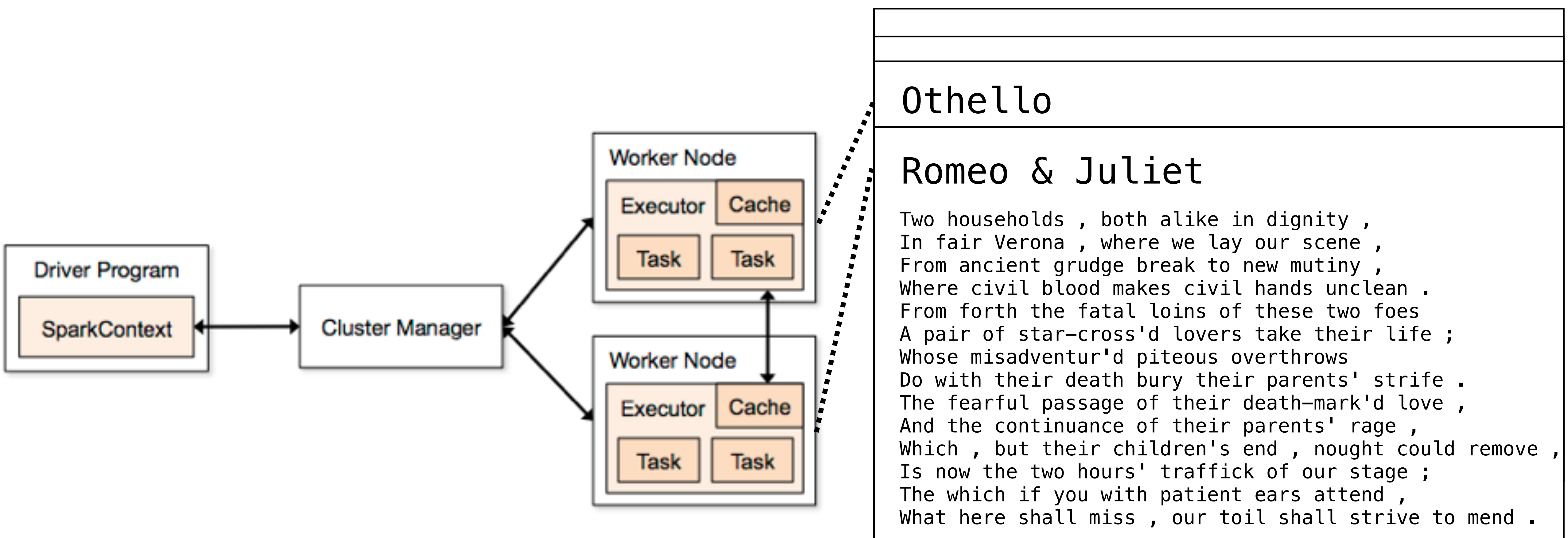
Othello

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 .
```

The Last Words of Shakespeare

- A *driver program* defines transformations and actions
- A *cluster manager* assigns tasks to individual *worker nodes*
- Worker nodes perform computation and communicate values to each other



The Last Words of Shakespeare

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager

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

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file

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

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
```

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file
- The sortBy transformation and take action are methods

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
```

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file
- The sortBy transformation and take action are methods

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
>>> shakes.sortBy(lambda line: line, False)
```

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file
- The sortBy transformation and take action are methods

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
>>> shakes.sortBy(lambda line: line, False)
...       .take(2)
```

The Last Words of Shakespeare

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file
- The sortBy transformation and take action are methods

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
>>> shakes.sortBy(lambda line: line, False)
...       .take(2)
['you shall...', 'yet, a...']
```

The Last Words of Shakespeare (demo)

- A SparkContext gives access to the cluster manager
- An RDD can be constructed from the lines of a text file
- The sortBy transformation and take action are methods

```
>>> sc
<pyspark.context.SparkContext ...>
>>> shakes = sc.textFile('shakespeare.txt')
>>> shakes.sortBy(lambda line: line, False)
...       .take(2)
['you shall...', 'yet, a...']
```

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
- **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
- **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
- **Monitoring:** Will my job finish before dinner?!?
  - The cluster manager provides a web-based interface describing jobs
- **Abstraction!**

MapReduce

MapReduce Applications

MapReduce Applications

- An important early distributed processing system was MapReduce, published by Google in 2004

MapReduce Applications

- An important early distributed processing system was MapReduce, published by Google in 2004
- Simple structure that happened to capture many common data processing tasks

MapReduce Applications

- An important early distributed processing system was MapReduce, published by Google in 2004
- Simple 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, published by Google in 2004
- Simple 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, published by Google in 2004
- Simple 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: All the values for a key are processed as a sequence (reduce)

MapReduce Applications

- An important early distributed processing system was MapReduce, published by Google in 2004
- Simple 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: All the values for a key are processed as a sequence (reduce)
- Early applications: indexing web pages, computing PageRank

MapReduce Evaluation Model

MapReduce Evaluation Model

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

MapReduce Evaluation Model

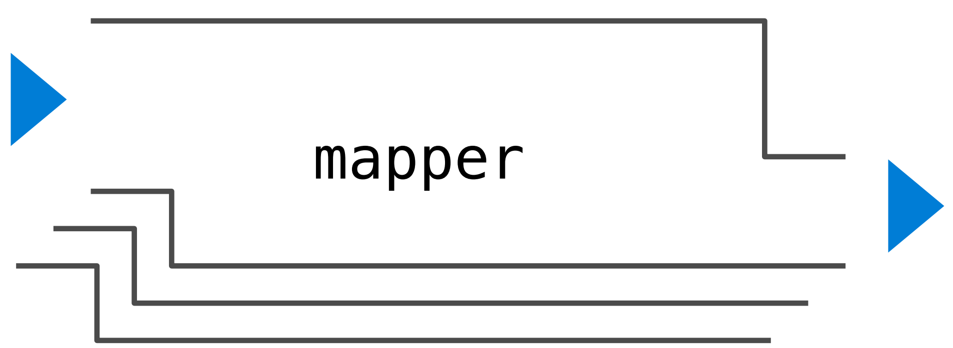
- Map step: Apply a mapper function to all inputs, emitting intermediate key-value pairs

Google MapReduce
Is a Big Data framework
For batch processing

MapReduce Evaluation Model

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

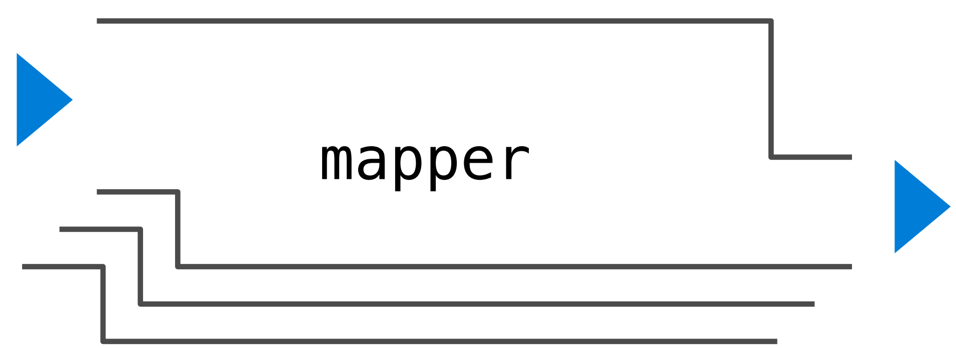
Google MapReduce
Is a Big Data framework
For batch processing



MapReduce Evaluation Model

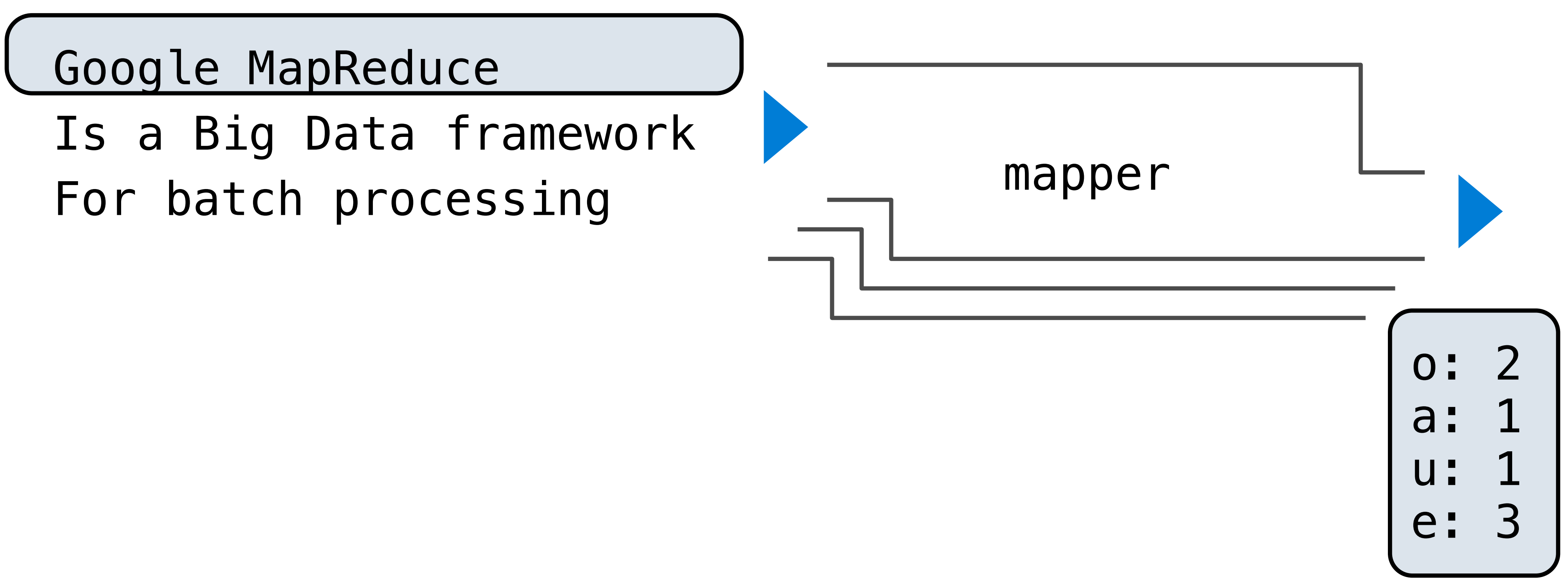
- Map step: Apply a mapper function to all inputs, emitting intermediate key-value pairs

Google MapReduce
Is a Big Data framework
For batch processing



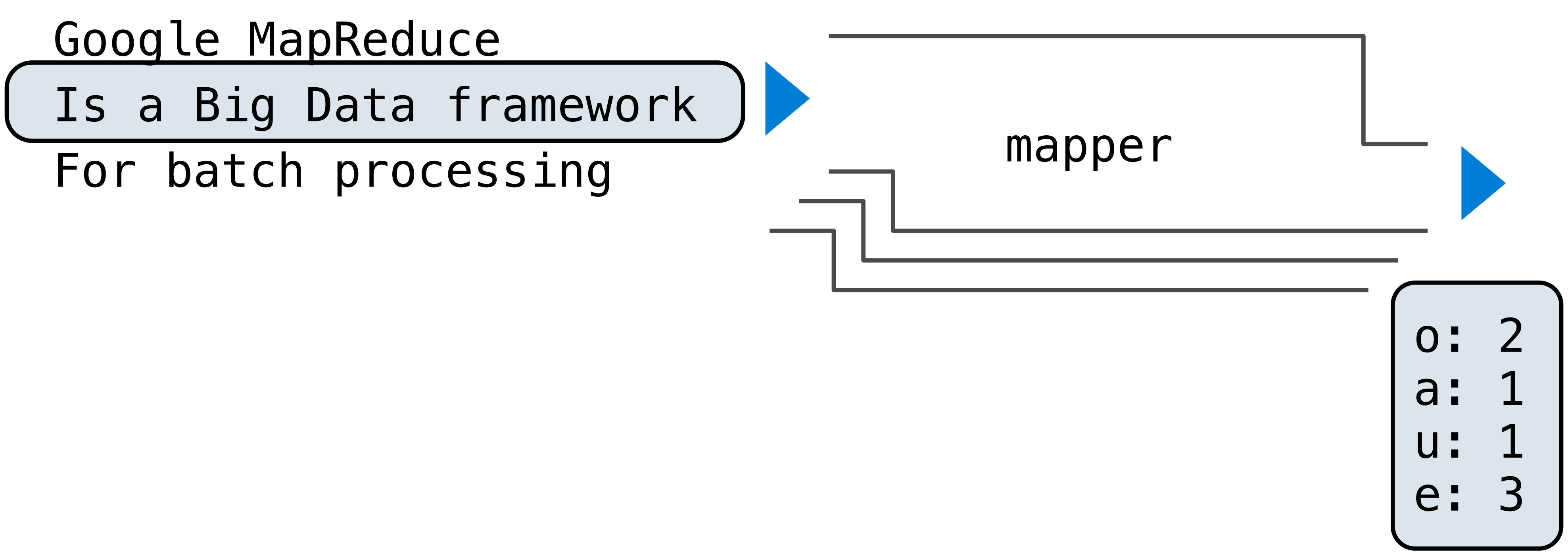
MapReduce Evaluation Model

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



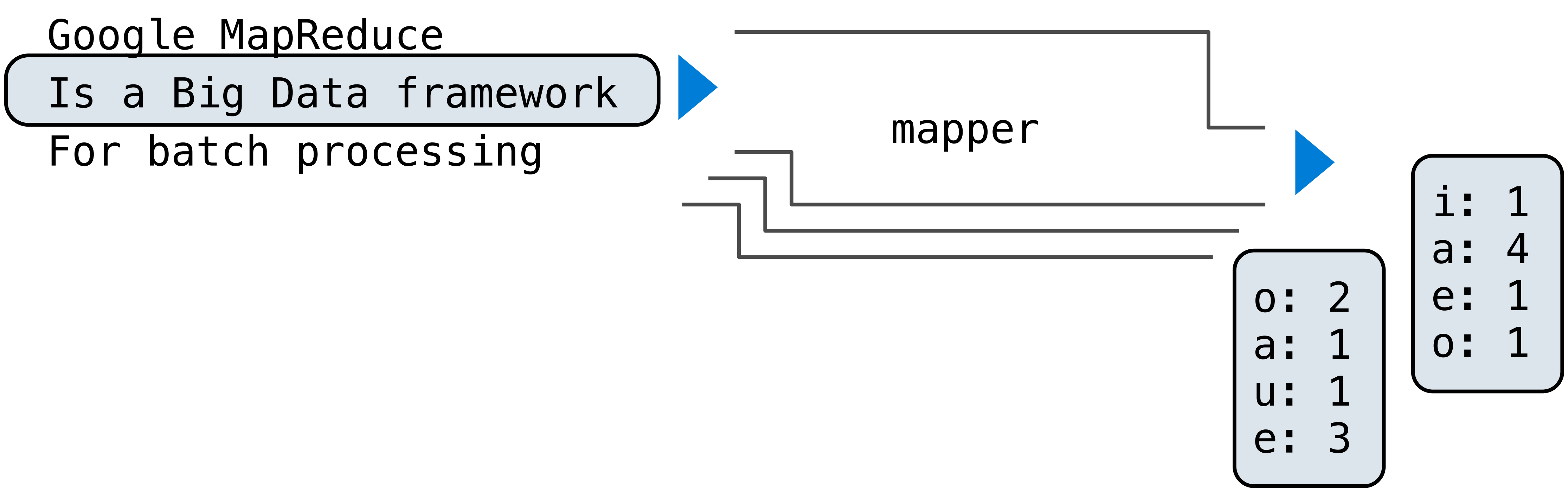
MapReduce Evaluation Model

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



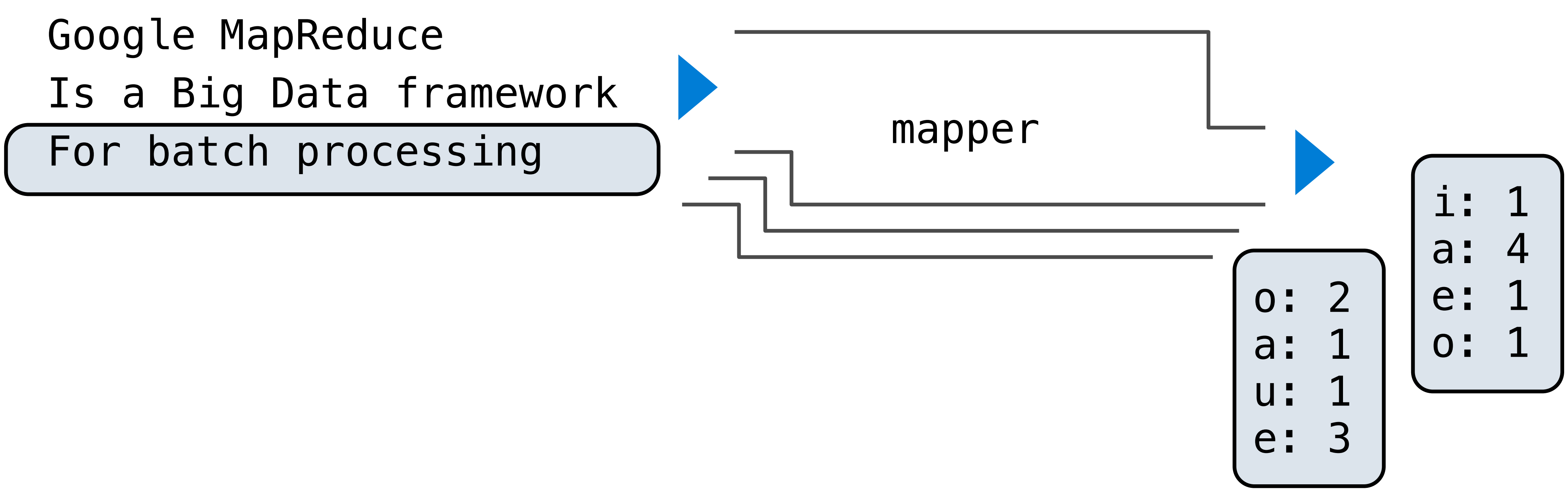
MapReduce Evaluation Model

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



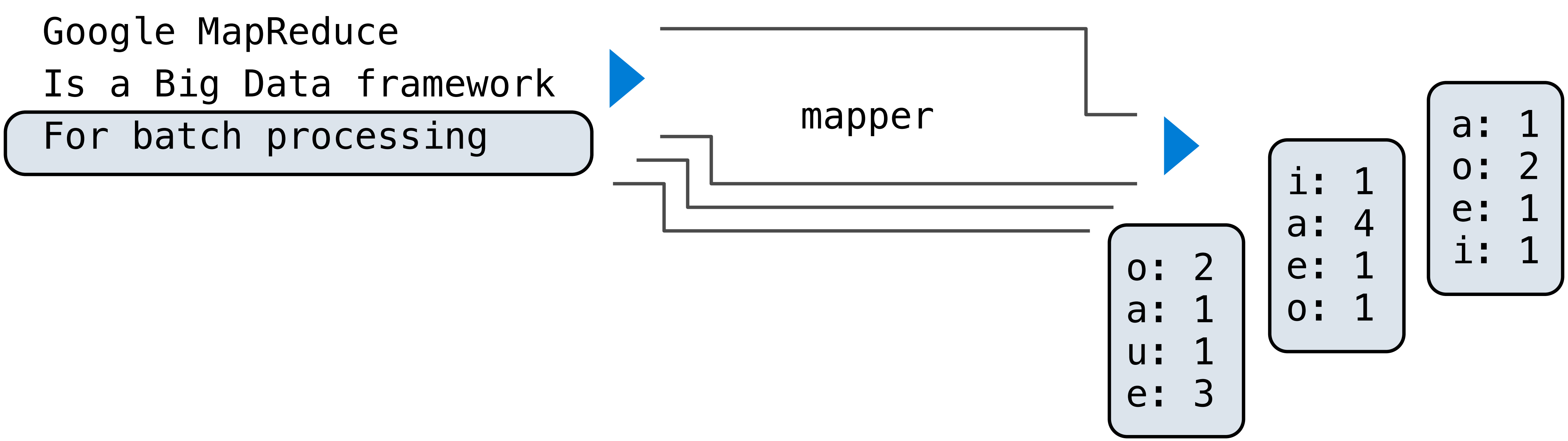
MapReduce Evaluation Model

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



MapReduce Evaluation Model

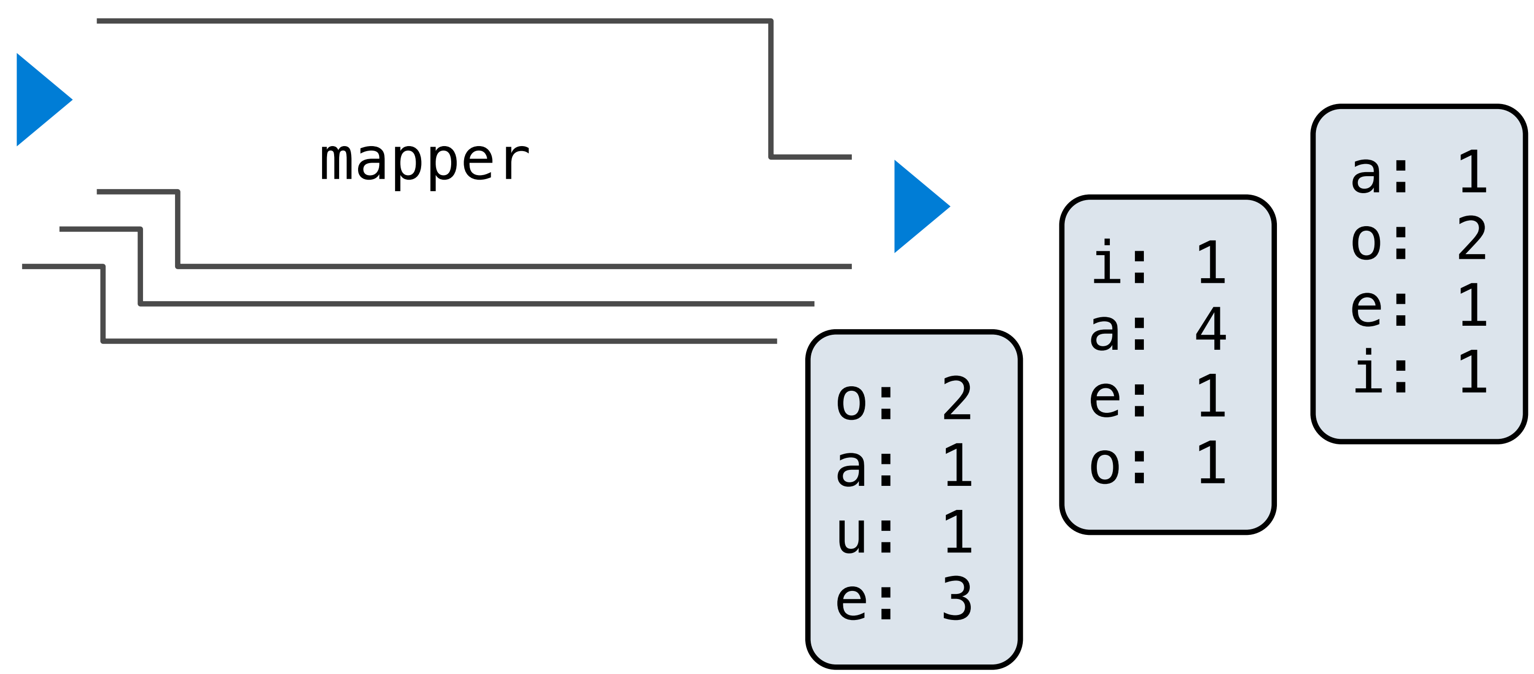
- Map step: Apply a mapper function to all inputs, emitting intermediate key-value pairs



MapReduce Evaluation Model

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

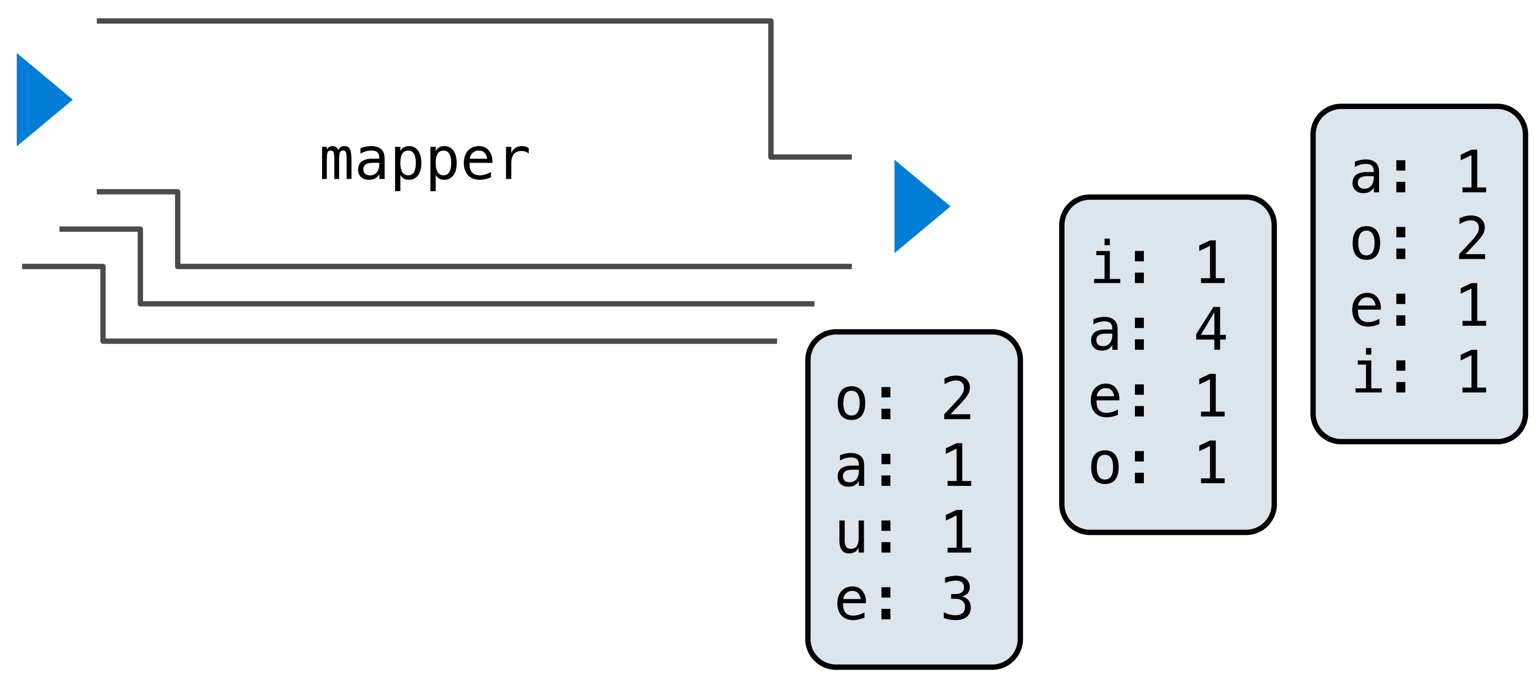
Google MapReduce
Is a Big Data framework
For batch processing



MapReduce Evaluation Model

- Map step: Apply a mapper function to all inputs, emitting intermediate key-value pairs
- Reduce step: For each intermediate key, apply a reducer function to accumulate all values associated with that key

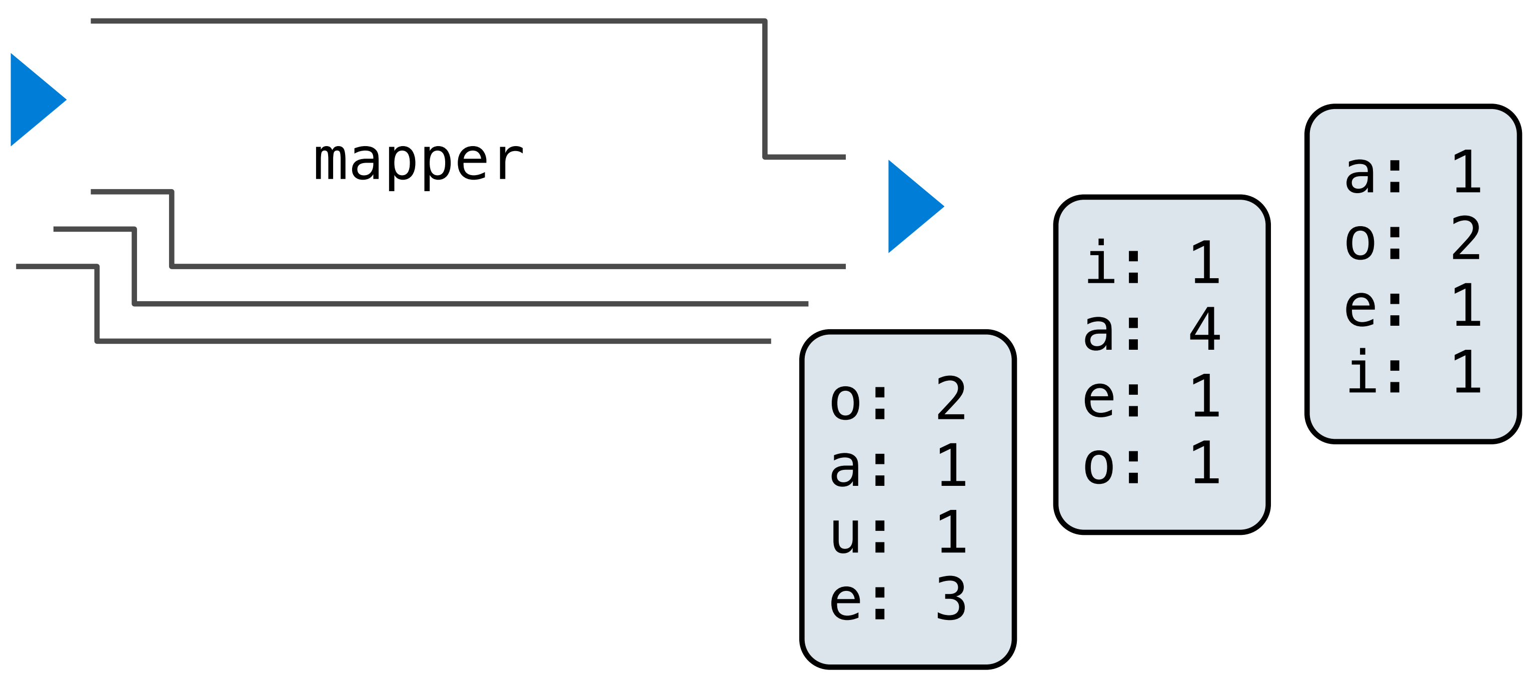
Google MapReduce
Is a Big Data framework
For batch processing



MapReduce Evaluation Model

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

Google MapReduce
Is a Big Data framework
For batch processing



MapReduce Evaluation Model

MapReduce Evaluation Model

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

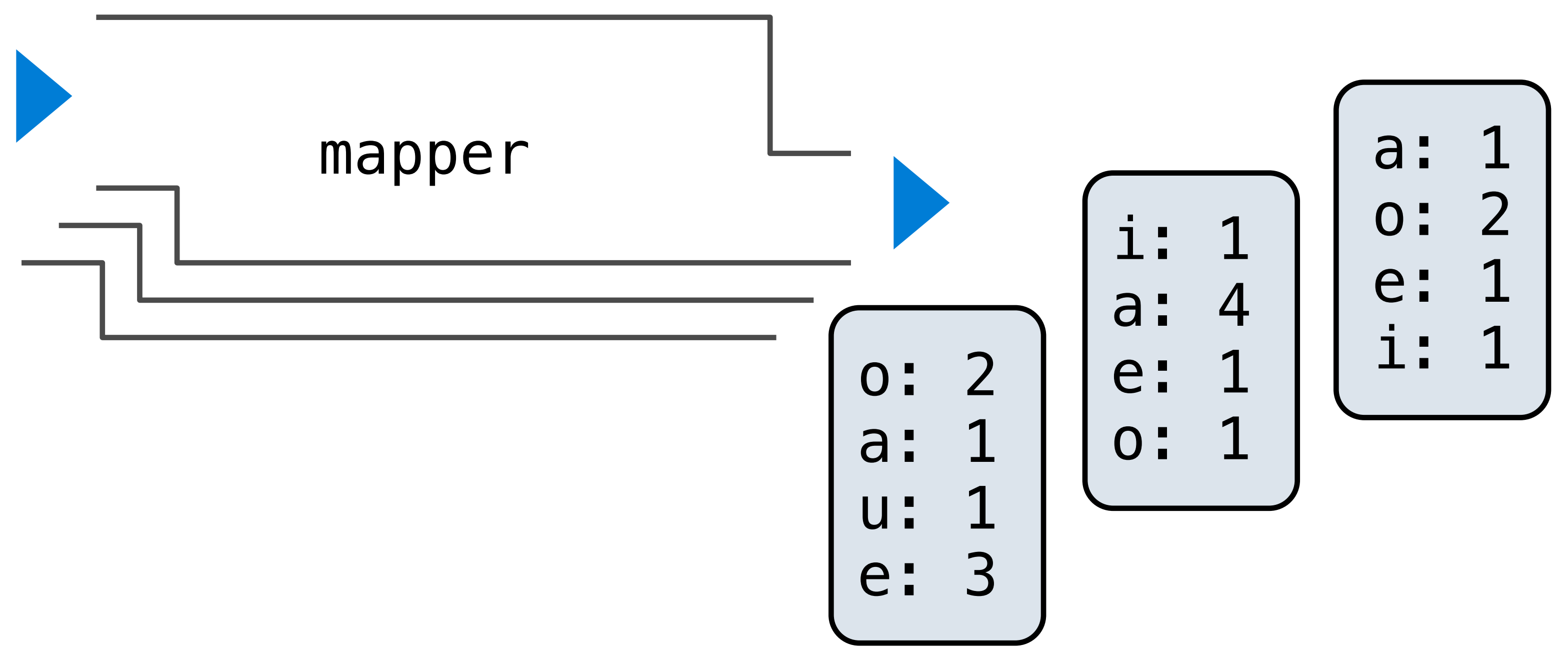
MapReduce Evaluation Model

- Reduce step: 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

- Reduce step: 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

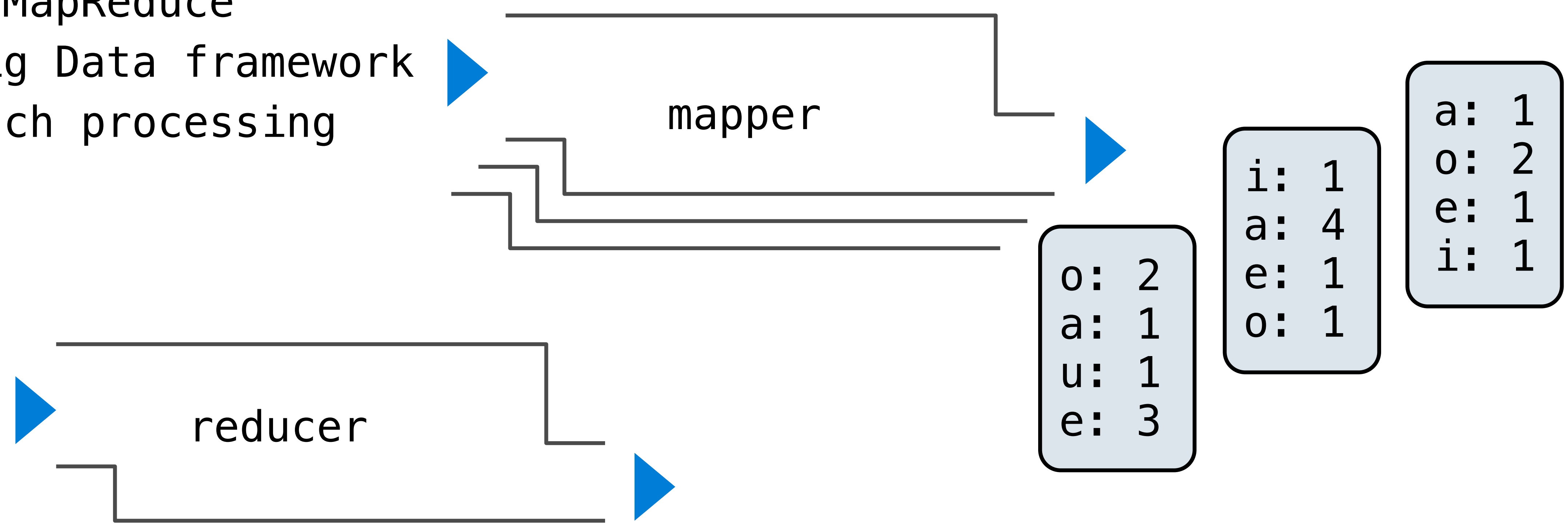
Google MapReduce
Is a Big Data framework
For batch processing



MapReduce Evaluation Model

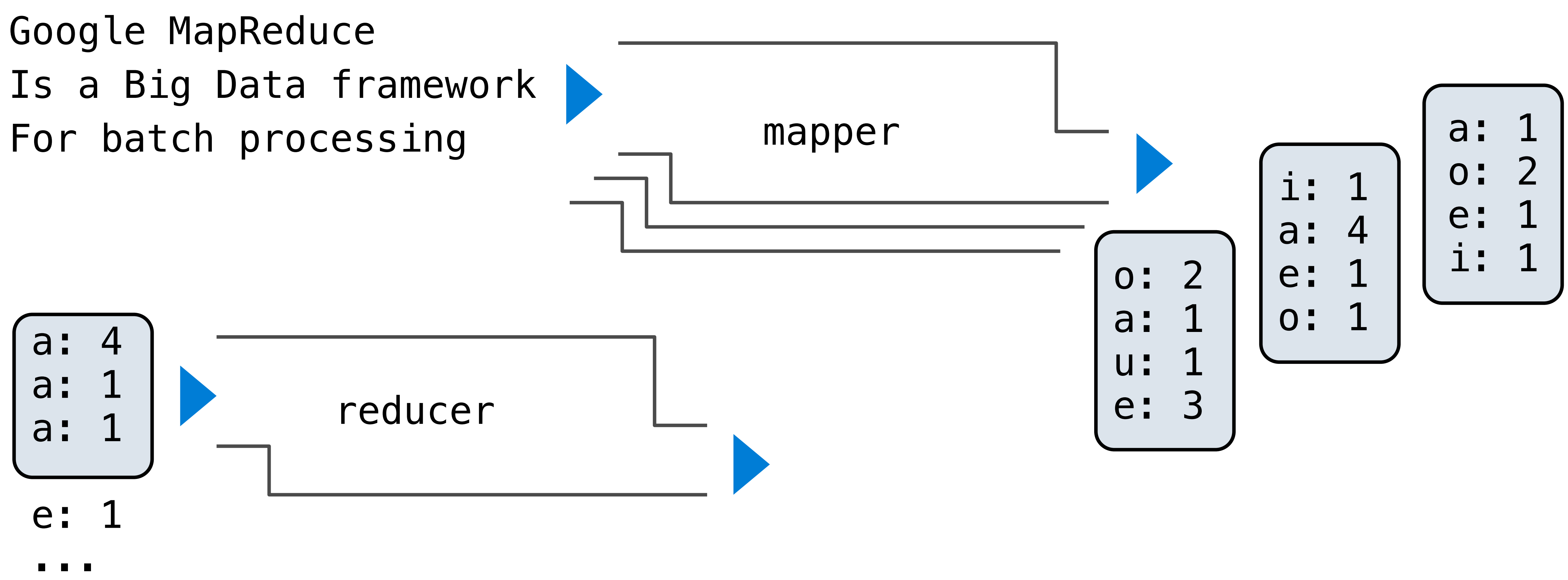
- Reduce step: For each intermediate key, apply a reducer function to accumulate all values associated with that key
  - All key-value pairs with the same key are processed together

Google MapReduce
Is a Big Data framework
For batch processing



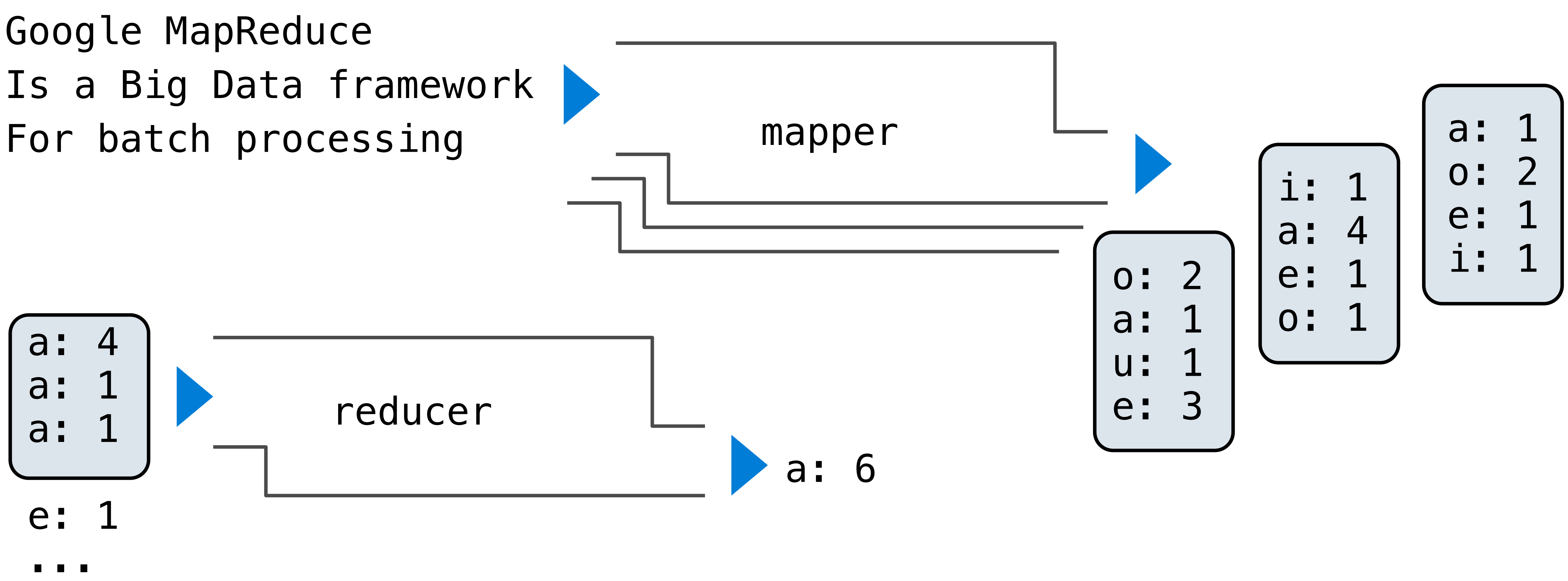
MapReduce Evaluation Model

- Reduce step: 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

- Reduce step: 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 on Apache Spark

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

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

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

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

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

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression

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

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

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

Call Expression **Data**

`data.flatMap(fn)`

`data.reduceByKey(fn)`

MapReduce on Apache Spark

Key-value pairs are just two-element Python tuples

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

MapReduce 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 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 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 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 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 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 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 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 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 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 on Apache Spark (demo)

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

Summary

Summary

- Some problems are too big for one computer to solve!

Summary

- Some problems are too big for one computer to solve!
- However, distributed programming comes with its own issues

Summary

- Some problems are too big for one computer to solve!
- However, distributed programming comes with its own issues
- We can use abstractions (such as Apache Spark) to manage some of the complexity that is inevitable when running programs on many machines