Announcements

- HW13 due Wednesday

- Scheme project due Monday

- Scheme contest deadline extended to Friday
CPU Performance
Performance of individual CPU cores has largely stagnated in recent years.
CPU Performance

Performance of individual CPU cores has largely stagnated in recent years. Graph of CPU clock frequency, an important component in CPU performance:
Performance of individual CPU cores has largely stagnated in recent years.

Graph of CPU clock frequency, an important component in CPU performance:

[Graph of CPU clock frequency]

http://cpudb.stanford.edu
Parallelism
Parallelism

Applications must be *parallelized* in order to run faster.
Parallelism

Applications must be *parallelized* in order to run faster

- Waiting for a faster CPU core is no longer an option
Parallelism

Applications must be *parallelized* in order run faster

• Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:
Parallelism

Applications must be *parallelized* in order run faster
• Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:
• When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
Parallelism

Applications must be *parallelized* in order to run faster

- Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program
Parallelism

Applications must be *parallelized* in order run faster

- Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program

But not all problems can be solved efficiently using functional programming
Parallelism

Applications must be *parallelized* in order run faster
• Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:
• When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
• Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program

But not all problems can be solved efficiently using functional programming

Today: the easy case of parallelism, using only pure functions
Parallelism

Applications must be *parallelized* in order run faster

- Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program

But not all problems can be solved efficiently using functional programming

Today: the easy case of parallelism, using only pure functions

- Specifically, we will look at *MapReduce*, a framework for such computations
Parallelism

Applications must be *parallelized* in order to run faster
- Waiting for a faster CPU core is no longer an option

Parallelism is easy in functional programming:
- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program

But not all problems can be solved efficiently using functional programming

Today: the easy case of parallelism, using only pure functions
- Specifically, we will look at *MapReduce*, a framework for such computations

Next time: the hard case, where shared data is required
MapReduce
MapReduce is a *framework* for batch processing of Big Data
MapReduce

MapReduce is a \textit{framework} for batch processing of Big Data

What does that mean?
MapReduce

MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
MapReduce

MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
MapReduce

MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
- **Big Data**: A buzzword used to describe data sets so large that they reveal facts about the world via statistical analysis
MapReduce

MapReduce is a framework for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
- **Big Data**: A buzzword used to describe data sets so large that they reveal facts about the world via statistical analysis

The MapReduce idea:
MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
- **Big Data**: A buzzword used to describe data sets so large that they reveal facts about the world via statistical analysis

The MapReduce idea:

- Data sets are too big to be analyzed by one machine
MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
- **Big Data**: A buzzword used to describe data sets so large that they reveal facts about the world via statistical analysis

The MapReduce idea:

- Data sets are too big to be analyzed by one machine
- When using multiple machines, systems issues abound
MapReduce is a *framework* for batch processing of Big Data

What does that mean?

- **Framework**: A system used by programmers to build applications
- **Batch processing**: All the data is available at the outset, and results aren't used until processing completes
- **Big Data**: A buzzword used to describe data sets so large that they reveal facts about the world via statistical analysis

The MapReduce idea:

- Data sets are too big to be analyzed by one machine
- When using multiple machines, systems issues abound
- Pure functions enable an abstraction barrier between data processing logic and distributed system administration
Systems research enables the development of applications by defining and implementing abstractions:
Systems research enables the development of applications by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
Systems research enables the development of applications by defining and implementing abstractions:

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

- **Networks** provide a simple, robust data transfer interface to constantly evolving communications infrastructure
Systems research enables the development of applications by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware
- **Networks** provide a simple, robust data transfer interface to constantly evolving communications infrastructure
- **Databases** provide a declarative interface to software that stores and retrieves information efficiently
Systems research enables the development of applications by defining and implementing abstractions:

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

Systems research enables the development of applications by defining and implementing abstractions:

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

• **Networks** provide a simple, robust data transfer interface to constantly evolving communications infrastructure

• **Databases** provide a declarative interface to software that stores and retrieves information efficiently

• **Distributed systems** provide a single-entity-level interface to a cluster of multiple machines

A unifying property of effective systems:
Systems research enables the development of applications by defining and implementing abstractions:

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

• **Networks** provide a simple, robust data transfer interface to constantly evolving communications infrastructure

• **Databases** provide a declarative interface to software that stores and retrieves information efficiently

• **Distributed systems** provide a single-entity-level interface to a cluster of multiple machines

A unifying property of effective systems:

Hide *complexity*, but retain *flexibility*
Essential features of the Unix operating system (and variants):
The Unix Operating System

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

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

standard input ➔ process ➔ standard output

Text input ➔ Text output
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
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

The *standard streams* in a Unix-like operating system are conceptually similar to Python iterators.
Python Programs in a Unix Environment
Python Programs in a Unix Environment

The built-in `input` function reads a line from *standard input*
Python Programs in a Unix Environment

The built-in `input` function reads a line from *standard input*

The built-in `print` function writes a line to *standard output*
Python Programs in a Unix Environment

The built-in `input` function reads a line from standard input.

The built-in `print` function writes a line to standard output.

The values `sys.stdin` and `sys.stdout` also provide access to the Unix standard streams as "files".
Python Programs in a Unix Environment

The built-in `input` function reads a line from *standard input*

The built-in `print` function writes a line to *standard output*

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

A Python "file" is an interface that supports iteration, read, and write methods
Python Programs in a Unix Environment

The built-in `input` function reads a line from standard input.

The built-in `print` function writes a line to standard output.

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

A Python "file" is an interface that supports iteration, read, and write methods.

Using these "files" takes advantage of the operating system standard stream abstraction.
MapReduce Evaluation Model
**MapReduce Evaluation Model**

**Map phase**: Apply a *mapper* function to inputs, emitting a set of *intermediate* key-value pairs.
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

Google MapReduce
Is a Big Data framework
For batch processing
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

Google MapReduce
Is a Big Data framework
For batch processing
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more *key-value pairs* per input

Google MapReduce
Is a Big Data framework
For batch processing

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more *key-value pairs* per input
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per 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 inputs, emitting a set of intermediate key-value pairs

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

---

Google MapReduce
Is a Big Data framework
For batch processing

mapper

<table>
<thead>
<tr>
<th>o</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
</tbody>
</table>
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

---

Google MapReduce
Is a Big Data framework
For batch processing

mapper

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
**MapReduce Evaluation Model**

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

---

Google MapReduce
Is a Big Data framework
For batch processing

mapper

| o: 2  |
| a: 1  |
| u: 1  |
| e: 3  |
| i: 1  |
| a: 4  |
| e: 1  |
| o: 1  |
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

Google MapReduce

Is a Big Data framework

For batch processing

---

mapper

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

- i: 1
- a: 4
- e: 1
- o: 1

- a: 1
- o: 2
- e: 1
- i: 1
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

**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 inputs, emitting a set of intermediate key-value pairs

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

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

- The *reducer* takes an iterator over key-value pairs
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

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

- The *reducer* takes an iterator over key-value pairs
- All pairs with a given key are consecutive
MapReduce Evaluation Model

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

- The *mapper* takes an iterator over inputs, such as text lines
- The *mapper* yields zero or more key-value pairs per input

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

- The *reducer* takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The *reducer* yields 0 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

- The *reducer* takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The *reducer* yields 0 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

- The *reducer* takes an iterator over *key-value pairs*
- All pairs with a given key are consecutive
- The *reducer* yields 0 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

• The reducer takes an iterator over key-value pairs
• All pairs with a given key are consecutive
• The reducer yields 0 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

- The reducer takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The reducer yields 0 or more values, each associated with that intermediate key
Reduce phase: For each intermediate key, apply a reducer function to accumulate all values associated with that key

- The reducer takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The reducer yields 0 or more values, each associated with that intermediate key
MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing

mapper

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

- The reducer takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The reducer yields 0 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

- The reducer takes an iterator over key-value pairs
- All pairs with a given key are consecutive
- The reducer yields 0 or more values, each associated with that intermediate key
Above-the-Line: Execution Model

Below-the-Line: Parallel Execution

http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0008.html
A "task" is a Unix process running on a machine

http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0008.html
A "task" is a Unix process running on a machine.
MapReduce Assumptions
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key.
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key.

**Benefits** of functional programming:
Constraints on the mapper and reducer:

• The mapper must be equivalent to applying a deterministic pure function to each input independently

• The reducer must be equivalent to applying a deterministic pure function to the sequence of values for each key

Benefits of functional programming:

• When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key.

**Benefits** of functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel.
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program.
MapReduce Assumptions

**Constraints** on the mapper and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key.

**Benefits** of functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel.
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program.

In MapReduce, these functional programming ideas allow:
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:

- The *mapper* must be equivalent to applying a deterministic pure function to each input independently.
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key.

**Benefits** of functional programming:

- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel.
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program.

In MapReduce, these functional programming ideas allow:

- Consistent results, however computation is partitioned.
MapReduce Assumptions

**Constraints** on the *mapper* and reducer:
- The *mapper* must be equivalent to applying a deterministic pure function to each input independently
- The *reducer* must be equivalent to applying a deterministic pure function to the sequence of values for each key

**Benefits** of functional programming:
- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or *vice versa*) without changing the program

In MapReduce, these functional programming ideas allow:
- Consistent results, however computation is partitioned
- Re-computation and caching of results, as needed
Python Example of a MapReduce Application
Python Example of a MapReduce Application

The *mapper* and *reducer* are both self-contained Python programs.
Python Example of a MapReduce Application

The *mapper* and *reducer* are both self-contained Python programs

- Read from *standard input* and write to *standard output*!
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs

- Read from standard input and write to standard output!

Mapper
The *mapper* and *reducer* are both self-contained Python programs

- Read from *standard input* and write to *standard output*!

```python
def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)
```

**Mapper**
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs
• Read from standard input and write to standard output!

Mapper

#!/usr/bin/env python3

import sys
from ucb import main
from mapreduce import emit

def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)
The mapper and reducer are both self-contained Python programs
• Read from standard input and write to standard output!

```python
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit

def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)
```
Python Example of a MapReduce Application

The *mapper* and *reducer* are both self-contained Python programs

- Read from *standard input* and write to *standard output*!

**Mapper**

```
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit

def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)
```

Tell Unix: this is Python

The `emit` function outputs a key and value as a line of text to standard output
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs

- Read from standard input and write to standard output!

Mapper

```
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit

def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)

for line in sys.stdin:
    emit_vowels(line)
```

Tell Unix: this is Python

The `emit` function outputs a key and value as a line of text to standard output
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs

- Read from *standard input* and write to *standard output*!

```python
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit

def emit_vowels(line):
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)

for line in sys.stdin:
    emit_vowels(line)
```

**Mapper**

Tell Unix: this is Python

The `emit` function outputs a key and value as a line of text to standard output

Mapper inputs are lines of text provided to standard input
Python Example of a MapReduce Application

The *mapper* and *reducer* are both self-contained Python programs

- Read from *standard input* and write to *standard output*!

Reducer
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs
• Read from standard input and write to standard output!

Reducer

#!/usr/bin/env python3

import sys
from ucb import main
from mapreduce import emit, group_values_by_key
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs
• Read from standard input and write to standard output!

Reducer

#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit, group_values_by_key

Takes and returns iterators
The *mapper* and *reducer* are both self-contained Python programs

- Read from *standard input* and write to *standard output*!

**Reducer**

```python
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit, group_values_by_key
```

**Input**: lines of text representing key-value pairs, grouped by key

**Output**: Iterator over (key, value_iterator) pairs that give all values for each key
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs

- Read from *standard input* and write to *standard output*

Reducer

```python
#!/usr/bin/env python3
import sys
from ucb import main
from mapreduce import emit, group_values_by_key

for key, value_iterator in group_values_by_key(sys.stdin):
    emit(key, sum(value_iterator))
```

**Input:** lines of text representing key-value pairs, grouped by key

**Output:** Iterator over (key, value_iterator) pairs that give all values for each key
What the MapReduce Framework Provides
What the MapReduce Framework Provides

Fault tolerance: A machine or hard drive might crash
What the MapReduce Framework Provides

Fault tolerance: A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks
What the MapReduce Framework Provides

**Fault tolerance:** A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed:** Some machine might be slow because it's overloaded
What the MapReduce Framework Provides

**Fault tolerance:** A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed:** Some machine might be slow because it's overloaded
- The framework can run multiple copies of a task and keep the result of the one that finishes first
What the MapReduce Framework Provides

**Fault tolerance:** A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed:** Some machine might be slow because it's overloaded
- The framework can run multiple copies of a task and keep the result of the one that finishes first

**Network locality:** Data transfer is expensive
What the MapReduce Framework Provides

**Fault tolerance:** A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed:** Some machine might be slow because it's overloaded
- The framework can run multiple copies of a task and keep the result of the one that finishes first

**Network locality:** Data transfer is expensive
- The framework tries to schedule map tasks on the machines that hold the data to be processed
What the MapReduce Framework Provides

**Fault tolerance:** A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed:** Some machine might be slow because it's overloaded
- The framework can run multiple copies of a task and keep the result of the one that finishes first

**Network locality:** Data transfer is expensive
- The framework tries to schedule map tasks on the machines that hold the data to be processed

**Monitoring:** Will my job finish before dinner?!?
What the MapReduce Framework Provides

**Fault tolerance**: A machine or hard drive might crash
- The MapReduce framework automatically re-runs failed tasks

**Speed**: Some machine might be slow because it's overloaded
- The framework can run multiple copies of a task and keep the result of the one that finishes first

**Network locality**: Data transfer is expensive
- The framework tries to schedule map tasks on the machines that hold the data to be processed

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