
Google™
11
I/O

App Engine MapReduce

Mike Aizatsky
11 May 2011

Hashtags: **#io2011 #AppEngine**
Feedback: **http://goo.gl/SnV2i**

Agenda

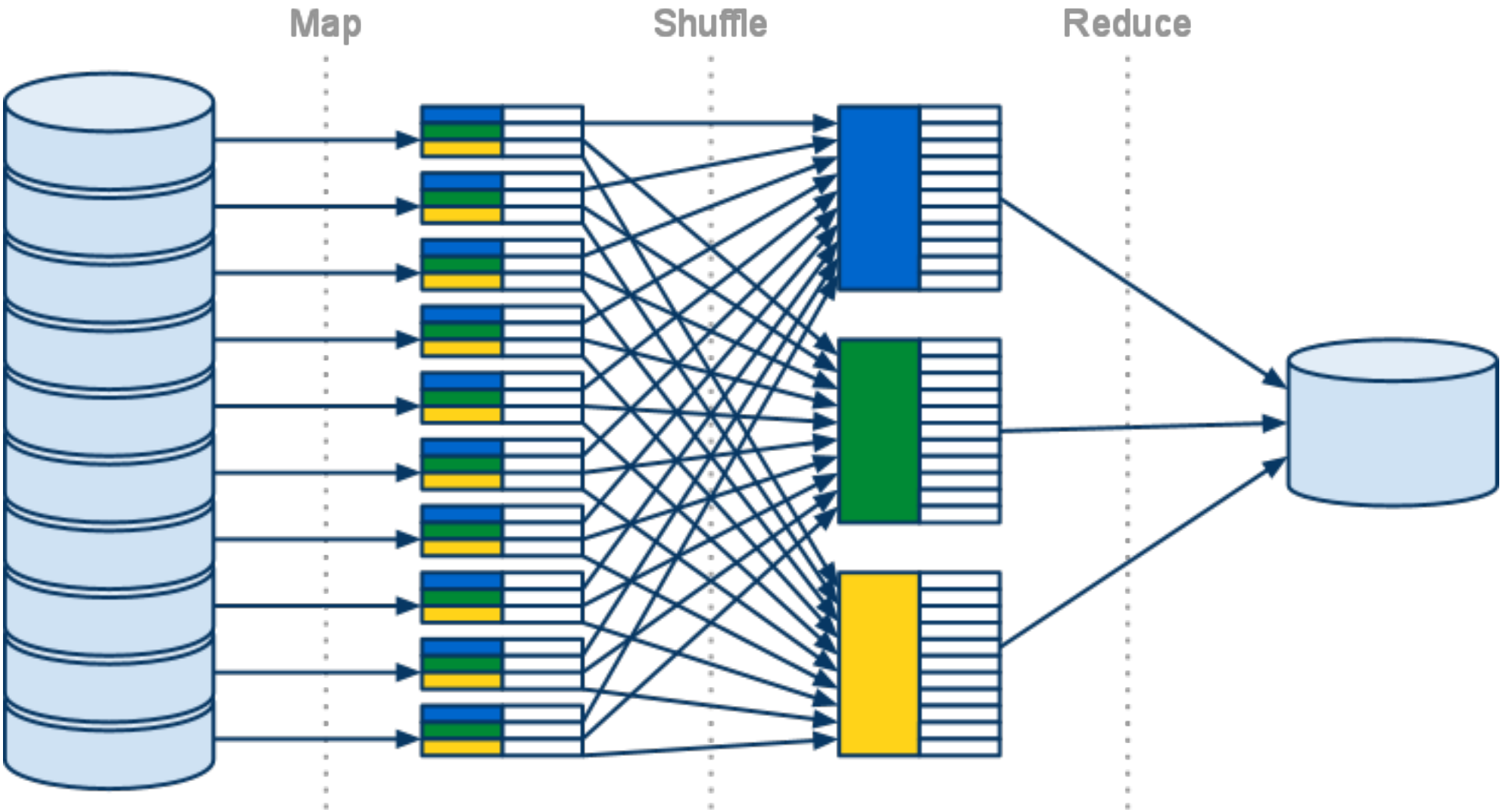
- MapReduce Computational Model
- Mapper library
- Announcement
- Technical bits:
  - Files API
  - User-space shuffling
- MapReduce & Pipeline API
- Examples and Demos

MapReduce Computational Model

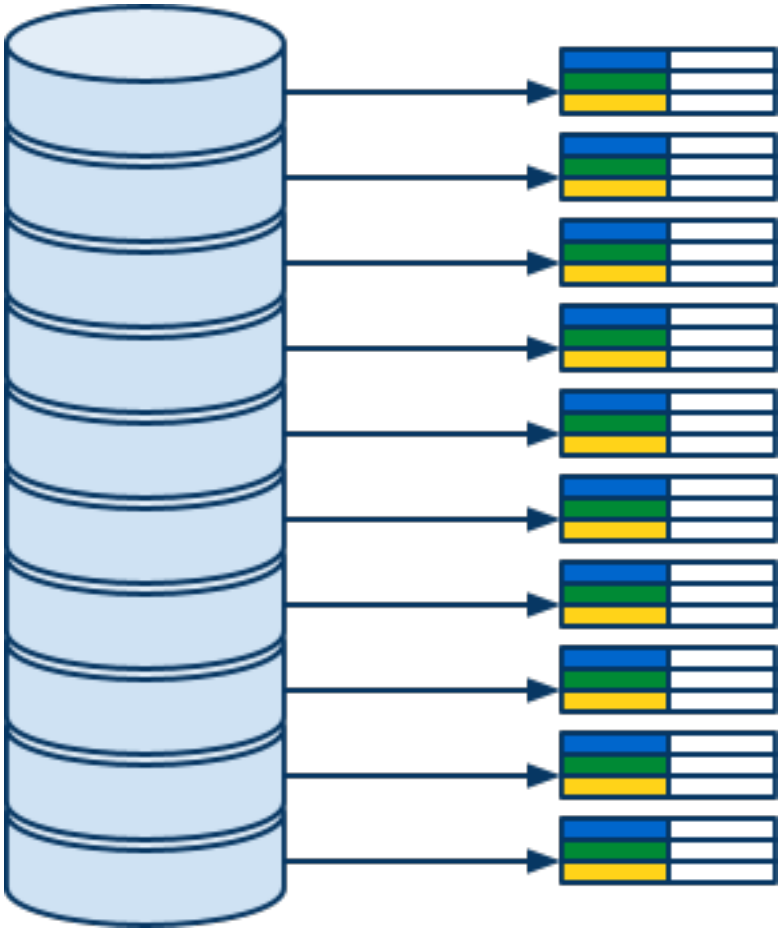
MapReduce

- A model to do efficient distributed computing over large data sets.
- Used at Google for years
- Every project uses MapReduce!

MapReduce Computational Model

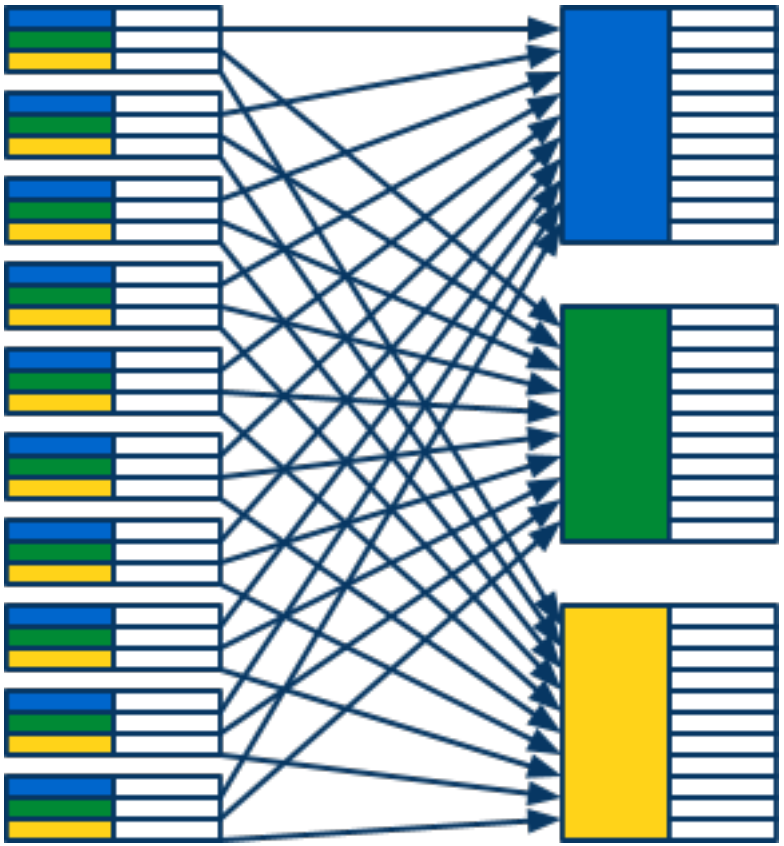


Map



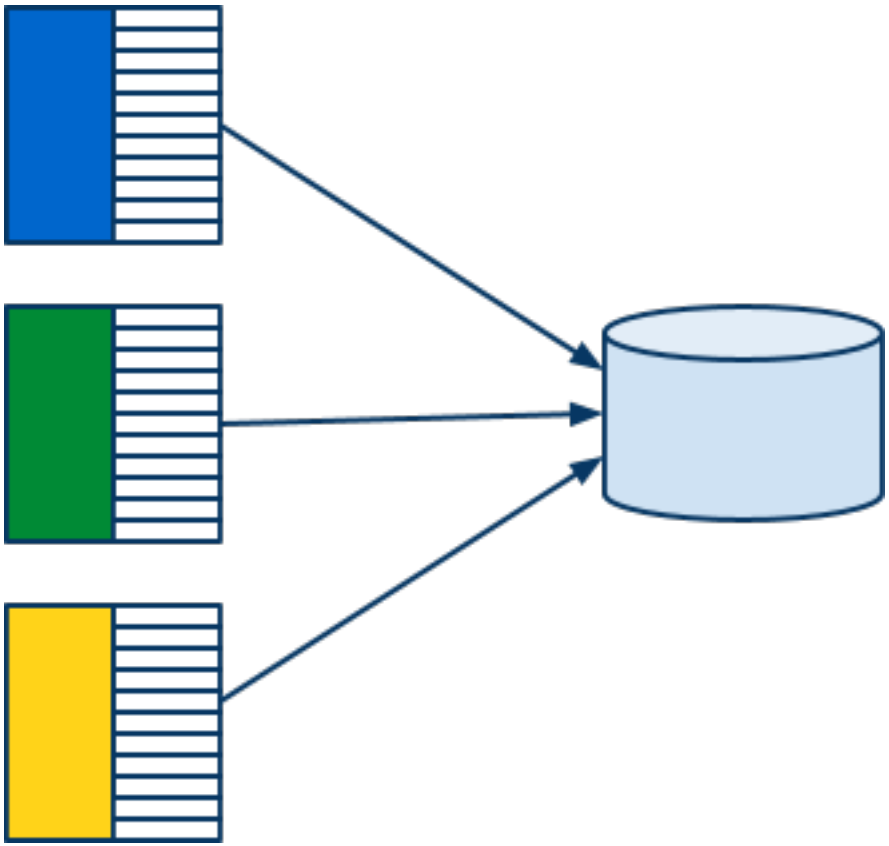
- Input: user data
- Output: (key, value) pairs
- User code

Shuffle



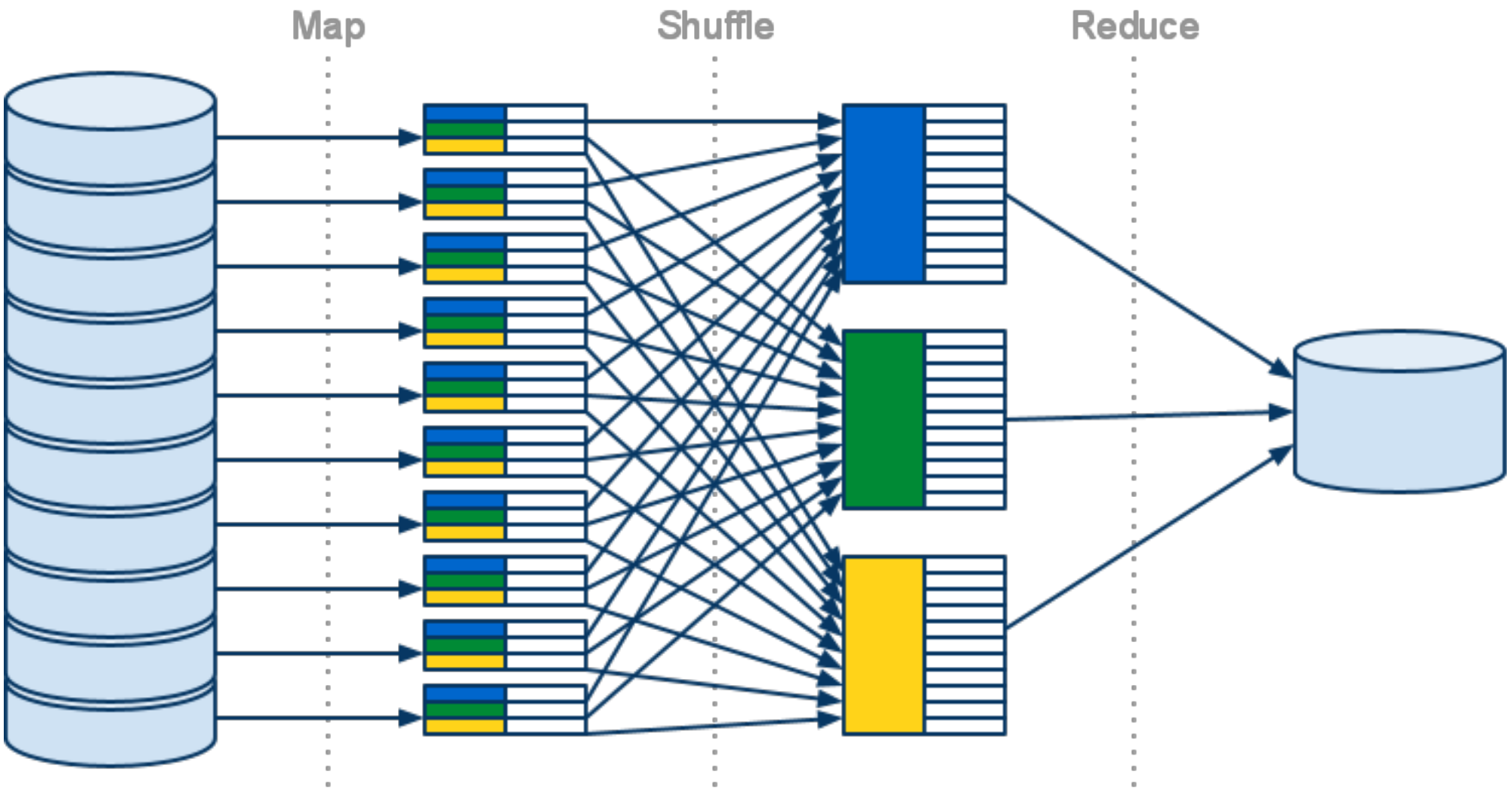
- Collates value with the same key
- Input: (key, value) pairs
- Output: (key, [value]) pairs
- No user code

Reduce



- Input: (key, [value]) pairs
- Output: user data
- User code

MapReduce Computational Model



Common App Engine Approach

- Take what works for us at Google
- Give it to people

App Engine & Google's MapReduce

- Additional scaling dimension:
  - Lots and lots of applications
  - Many of them will run MapReduce at the same time
- Isolation: application shouldn't influence performance of the other

App Engine & Google's MapReduce

- Rate limiting: you don't want to burn all day's resources in 15min and kill your online traffic
- Very slow execution: free apps want to go really slow, staying under their resource limint
- Protection: from malicious App Engine users

Mapper

Mapper Library

- Released at Google I/O 2010
- Heavily used by developers outside and inside Google (admin console, new indexer pipeline, etc.)
- Has seen lots of improvements since

Mapper Library Improvements

- Control API - start your jobs programmatically (and transactionally)
- Custom mutation pools - batch work between map function calls
- Namespaces support - iterate over data in different namespaces or over namespaces themselves
- Better sharding with scatter indices
- And more!

Mapper => MapReduce?

- Storage system for intermediate data:
  - Files API, released in 1.4.3 (March 2011)
- Shuffler
- Lots of glue code

Launching Shuffler Functionality

- In-memory, user-space, task-driven shuffle for small (100Mb) datasets.
- Trusted testers access to big shuffler.
- All the integration pieces needed to run your own mapreduce jobs are part of Mapper library.
- Mapper library => Mapreduce library!
- Python today, Java soon.

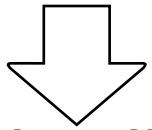
http://mapreduce.appspot.com

Examples

Example 1: Word Count

```
# Map
def map(line):
 for w in clean(line).split():
    yield (w, '')
```

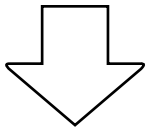
Zed's dead, baby, Zed's dead!



('zed's', ''), ('dead', ''), ('baby', ''), ('zed's', ''), ('dead', '')

```
# Reduce
def reduce(key, values):
 yield (key, len(values))
```

('zed's', ['', '']), ('dead', ['', '']), ('baby', [''])



('zed's', 2), ('dead', 2'), ('baby', 1)

Demo

Example 2: Inverse Index

```
# Map
def map(line, filename):
 for w in clean(line).split():
    yield (w, filename)
```

```
# Reduce
def reduce(key, values):
 yield (key, list(set(values)))
```

Demo

MapReduce: Technical Bits

Technical Bits

- Files API: solution to MapReduce storage problem
- User-space shuffler

Files API

Mapreduce Storage

- Mapreduce jobs generate lots of intermediate data.
- Datastore: expensive, 1MB entity limit
- Blobstore: read-only
- Memcache: small, volatile

Files API

- Familiar, files-like interface to various virtual file systems.
- Released in 1.4.3, integrated with Mapper library.
- Considered to be a low-level API.

Files API

- Files have two states: writable and readable.
- Start in writable. Moved to readable by "finalization".
- Can't read writable, can't write to readable.
- Write is append-only, atomic and fully serializable between concurrent clients.
- Concrete filesystems might have their own reliability constraints and/or additional APIs.

Blobstore Filesystem

- Write directly to blobstore.
- Files can be >2G.
- Finalized files are durable.
- Writable files are not (just restart your MapReduce)
- Can fetch a blob key for finalized files and use blobstore api.

Blobstore Filesystem Python Example

```
from google.appengine.api import files
from __future__ import with_statement

# Create the file.
file_name = files.blobstore.create()
# Open the file for append.
with files.open(file_name, 'a') as f:
 f.write('data')

# All data is in. Finalize the file.
files.finalize(file_name)
```

Blobstore Filesystem Python Example

```
# Open the file for read.
with files.open(file_name, 'r') as f:
 data = f.read(4)

# Fetch blobkey for blobstore api.
blob_key = files.blobstore.get_blob_key(file_name)
```

Mapper Integration

```
# mapreduce.yaml
...
mapper:
output_writer: mapreduce.output_writers.
BlobstoreOutputWriters

# Handler function
def map(entity):
 yield entity.to_csv_line() + '\n'
```

Low-level Features

- Exclusive locks: files can be opened exclusively by a single client only.
- Sequence keys: each write can have a "sequence key" attached. Our backends make sure that they only increase.

Future Plans

- "Tempfile" file system: much faster, much cheaper, but not durable, several days of storage only (geared specifically towards MapReduce)
- Integrations with other Google storage technologies and other reliability guarantees

User-Space Shuffler

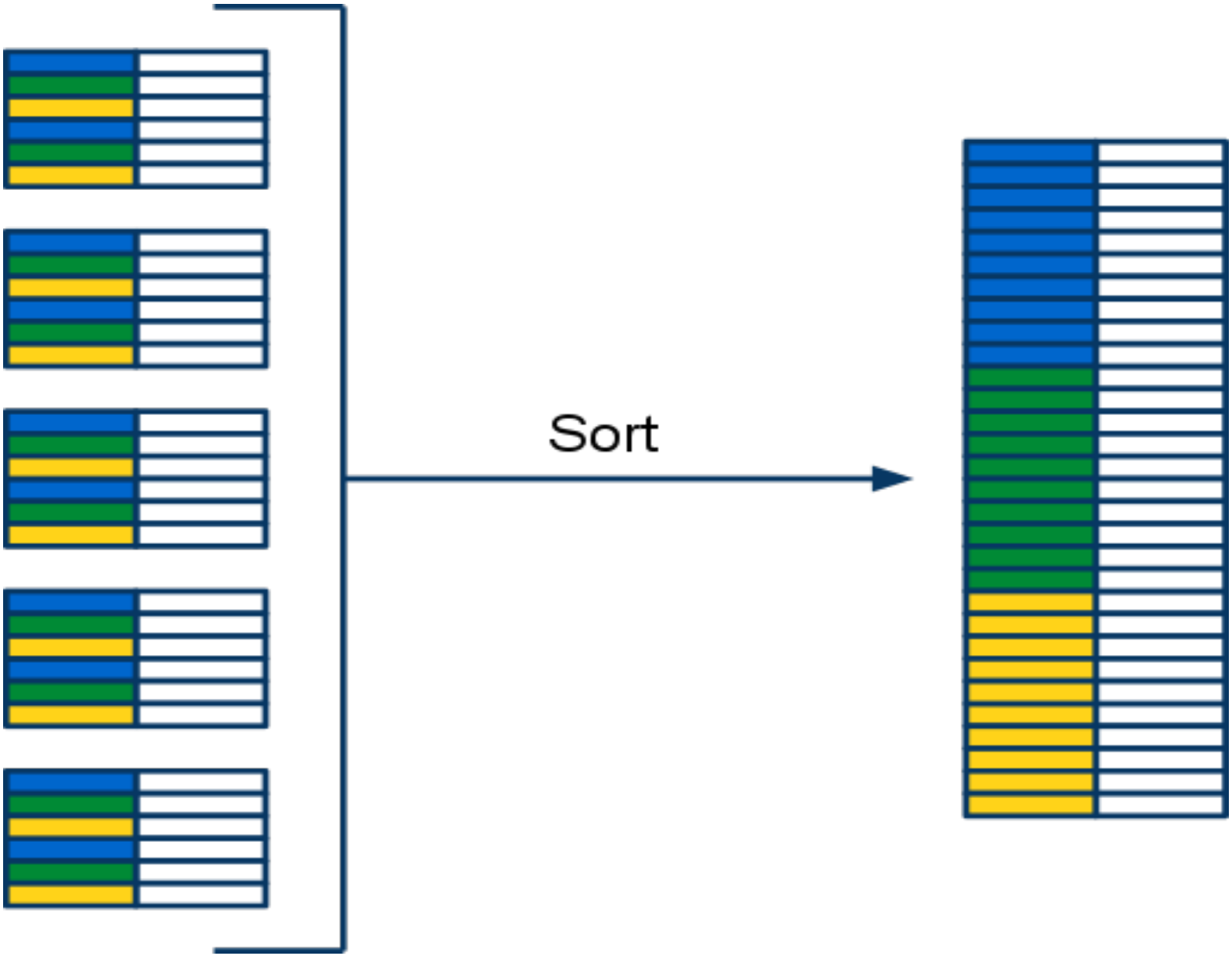
User-Space Shuffler

- Consolidates values for the same key together.
- [(key, value)] => [(key, [value])]
- Should be reasonably fast, scalable and efficient.
- User-space: full source code, no new AppEngine components.

Take 1

- Load all data into memory
- Sort
- Read sorted array

Take 1



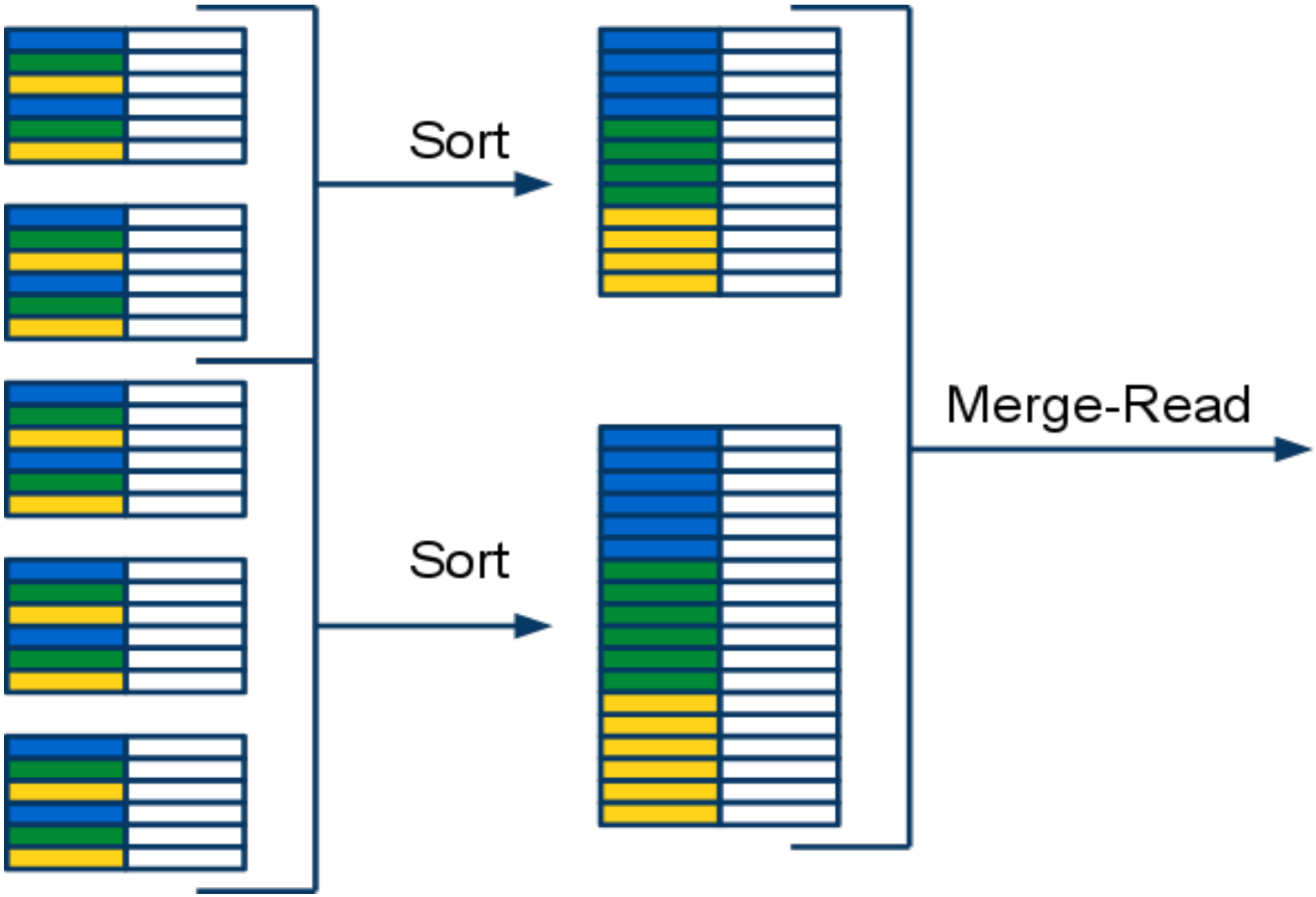
Take 1 Properties

- Memory-bound
- No parallelism

Take 2

- Sort chunks of data and store them back to Files API
- Merge-sort all chunks (or merge-read)

Take 2



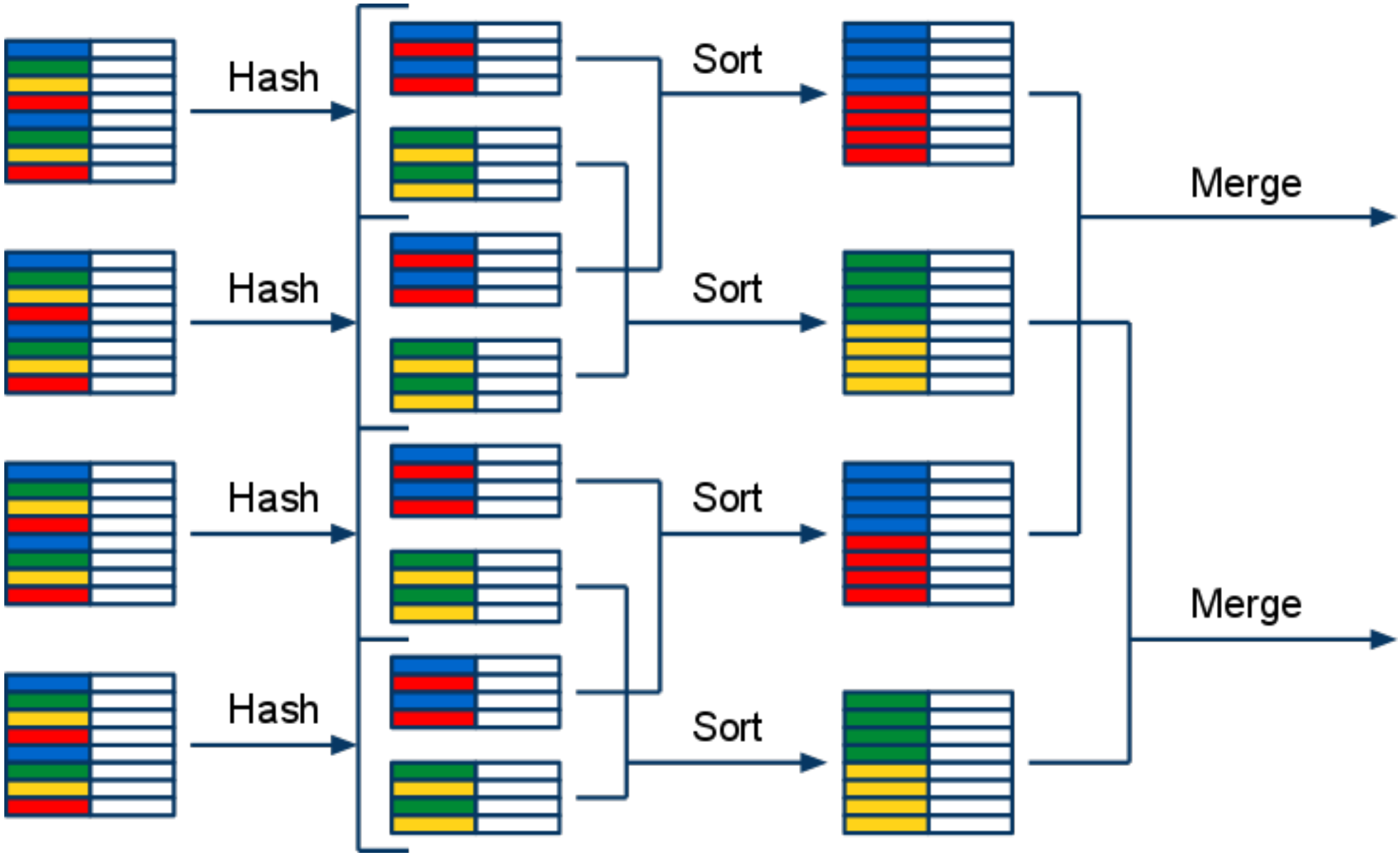
Take 2 Properties

- No longer memory-bound
- Sorting is parallel
- Merge phase is not parallel
- Difficult (and slow) to read from too many files

Take 3

- Shard mapper output by key hash code
- Sort each shard into chunks
- Merge-read each shard

Take 3



Take 3 Properties

- No longer memory-bound
- Sorting is parallel
- Merge phase is now parallel
- This is the shuffler we release today.

MapReduce & Pipeline

Pipeline API

- New API to chain complex work together.
- A glue which holds Mapper + Shuffler + Reducer together.
- MapReduce library is fully integrated with Pipeline.
- For in-depth look visit "Large-scale Data Analysis Using the App Engine Pipeline API" talk later today.

More Complex Example

Example 3: Distinguishing Phrases

```
# Map
def map(text, filename):
 for words in ngrams(text):
    yield (words, filename)

# Reduce
def reduce(key, values):
if len(values) < 10:
return
for filename, count in count_occurences(values):
if count > len(values) / 2:
yield (key, filename)
```

Demo

Summary

- Small & Medium MapReduce jobs can be run by anyone today!
- Contact us for getting access to Large MapReduce jobs.

http://mapreduce.appspot.com

Questions?

Hashtags: **#io2011 #AppEngine**
Feedback: **http://goo.gl/SnV2i**


Google™
11
I/O