Map Reduce

CSCI 333
Spring 2019

Jeannie Sez

“Why am I teaching file systems and you teaching MapReduce? Obviously we didn’t coordinate. Tell them to take 339 and they’ll read those papers again”

Logistics

Midterm

- I said “SATF” but meant “SSTF” and graded that way. If you answered correctly using either scheduling algorithm, you should get full credit.
  - Option 1: stop by my office
  - Option 2: submit your midterm with your final and indicate you want your question re-graded

Final Project

- Questions?

Last Class

Google File System

- Remove the single-server bottleneck
- Record-append
- 3-way replicate

Storage foundation on which MapReduce runs.

This Class

Map, reduce, (reuse, recycle)

- The problem
  - Examples
- The model
- Fault tolerance
- The straggler problem
- Moving data vs. moving computation

When Reading a Paper

Look at authors

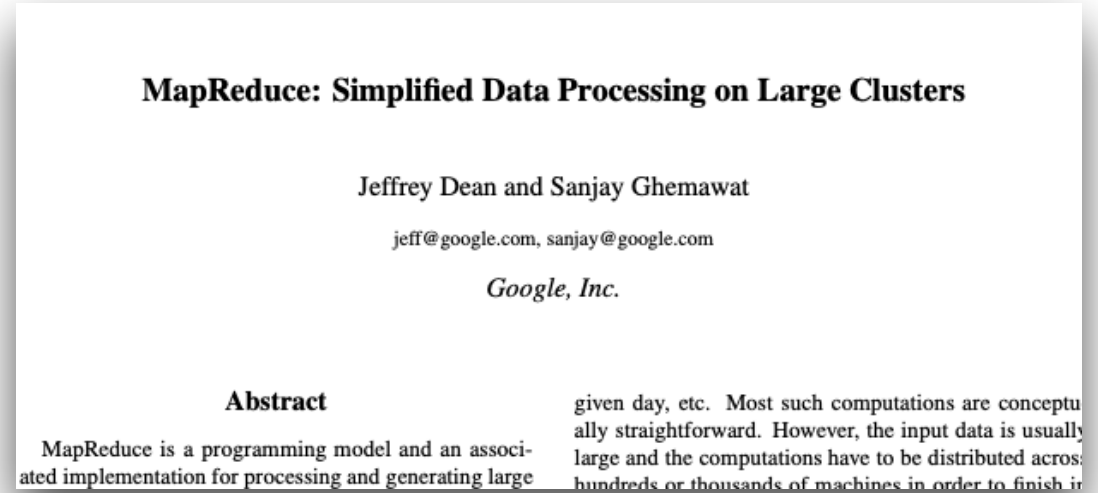
Look at institution

Look at past/future research

Look at publication venue

These things will give you insight into the

- motivations
- perspectives
- agendas
- resources



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish

Think: Are there things that they are promoting? Hiding? Building towards?

Why?

Thought Experiment

What is it that Google actually does?

- Sells ads

How do they sell ads?

- NLP on your emails, harvesting GPS data, etc. (in general by creeping on our personal lives)

But what does the average person mean when they use “Google” as a verb?

- Search!

Reverse Indexes

World-wide-web is a graph of webpages

- URI -> content (set of words)

Reverse index does the opposite

- word -> set of URIs

How would you implement a reverse index?

The Problem

Hundreds of special-purpose computations per day that

- Consume data distributed over thousands of machines
- Can be parallelized, and must be in order to finish in a reasonable timeframe

Challenges that each computation must solve:

- Parallelization
- Fault tolerance
- Data distribution
- Load balancing

Want one computation model that can use to abstract away these concerns

The Model

Map Reduce uses a functional model

- User supplied **map** function
  - *key-value pair* -> **set** of *key-value pairs*
- User supplied **reduce** function
  - **set** of all *key-value pairs* with a given key -> *key-value pair*

- The system applies the **map** function to each key-value pair, yielding a set of intermediate key-value pairs
- The system gathers all intermediate key-value pairs, and for each unique key, calls **reduce** on the set of key-value pairs with that key

Example: Word Frequency

Pseudo code (section 2.1):

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
     EmitIntermediate(w, “1”);

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
     result += ParseInt(v);

  Emit(AsString(result));
```

Emits each word plus an associated “**count**” (1 here; duplicates possible)

Aggregates all counts for individual words and sums the entries.

Design

Input data is distributed across multiple systems

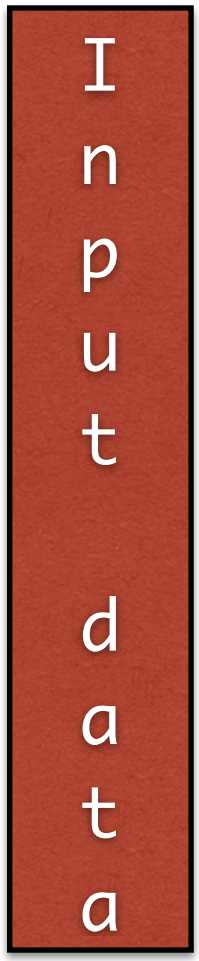
- Input data is divided into **M** (even) splits
- System schedules a mapper to run on each of the **M** splits

Intermediate (pre-reduce) data is distributed across multiple systems

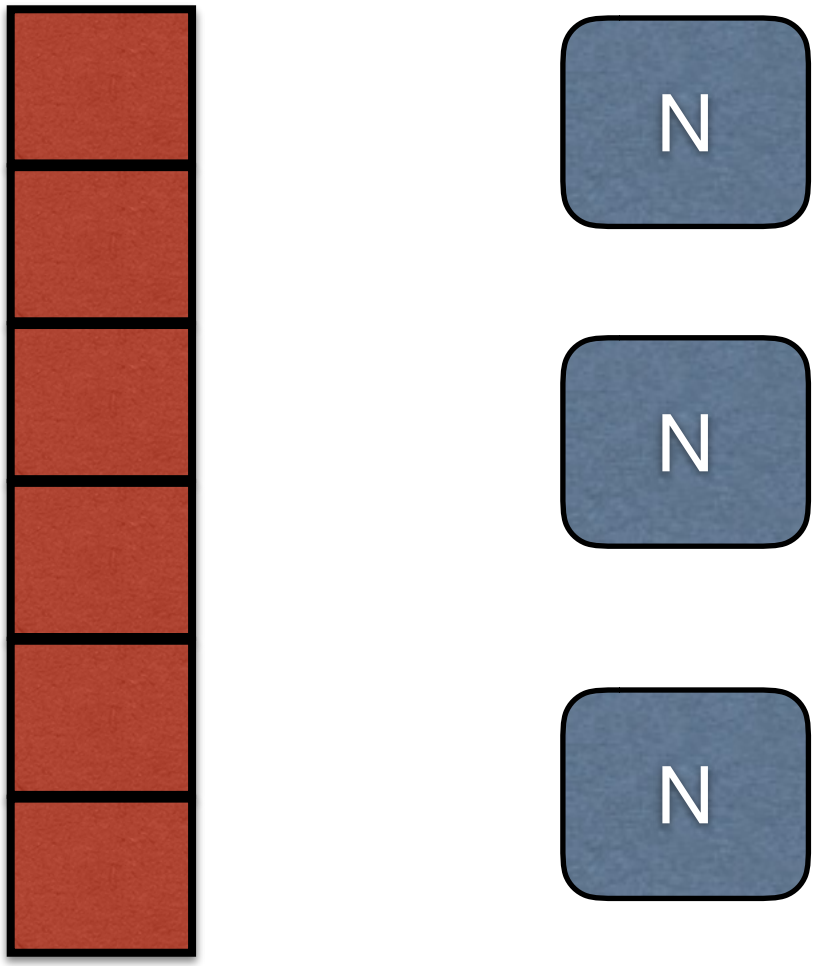
- Users provide a "partitioning" function (e.g., `hash(key) mod R`) that is used to distribute the mapper outputs
- System schedules a reducer on each of the **R** pieces of the intermediate outputs

Result of computation is located in R output files

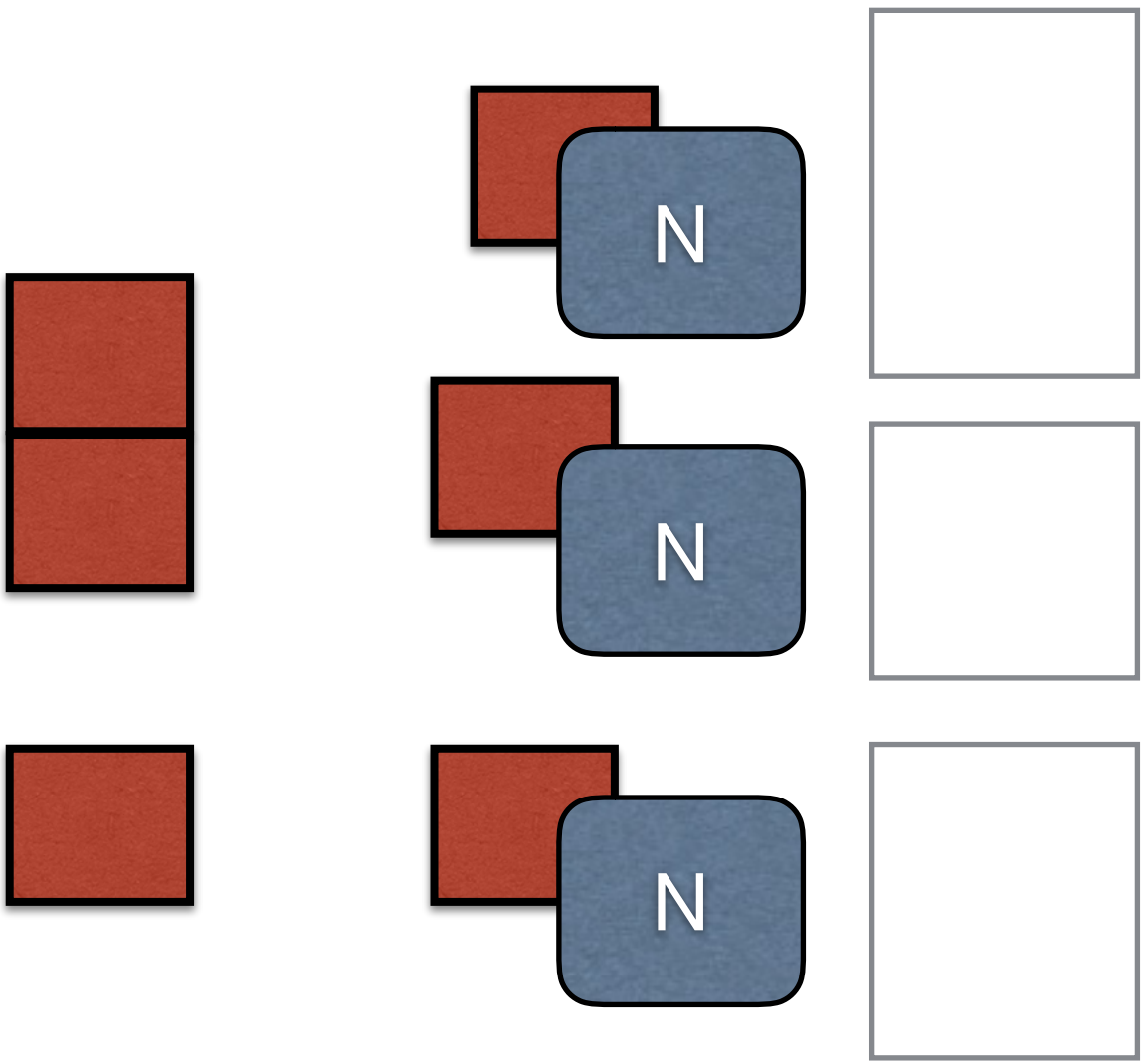
Map Reduce



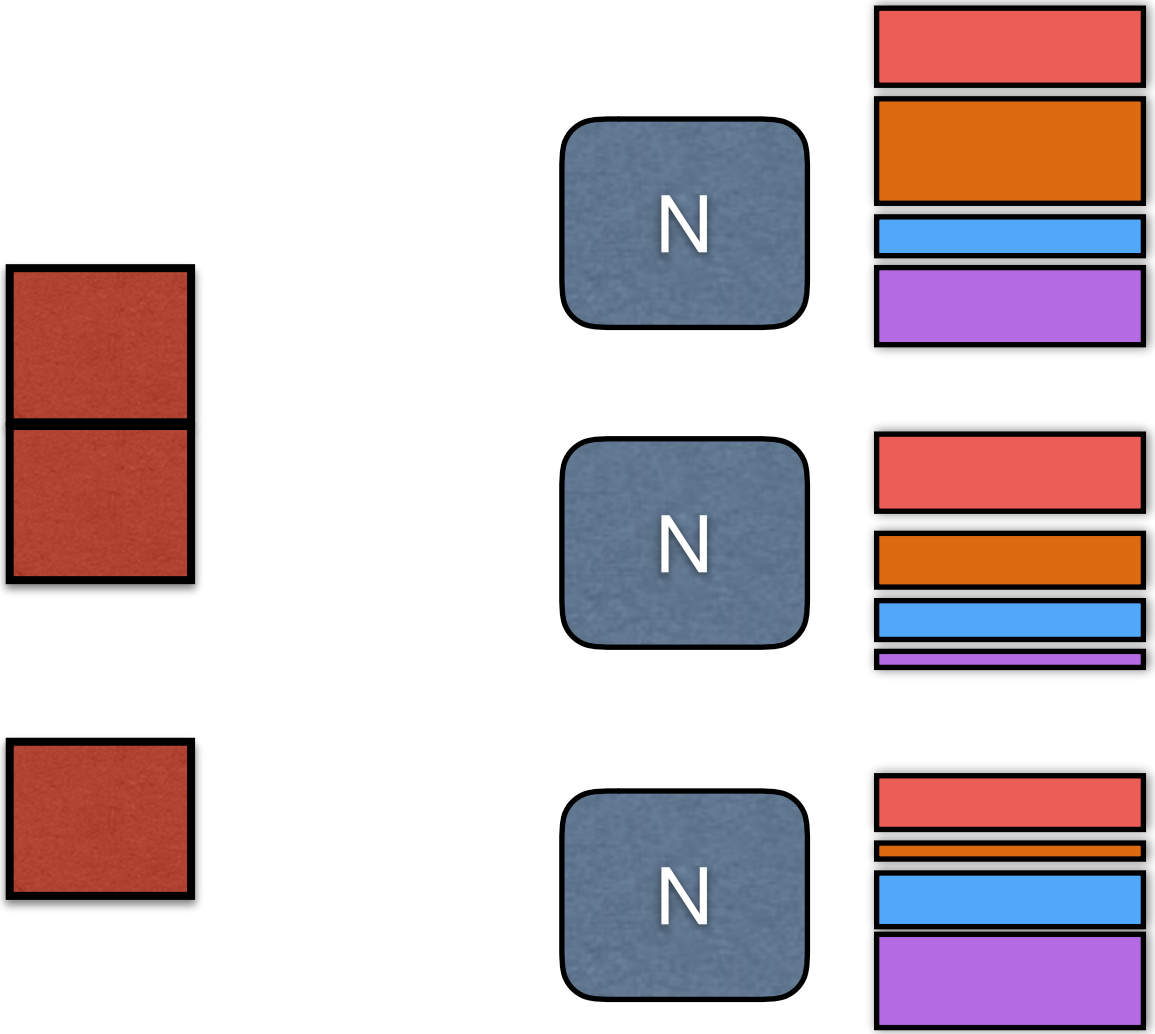
Map Reduce



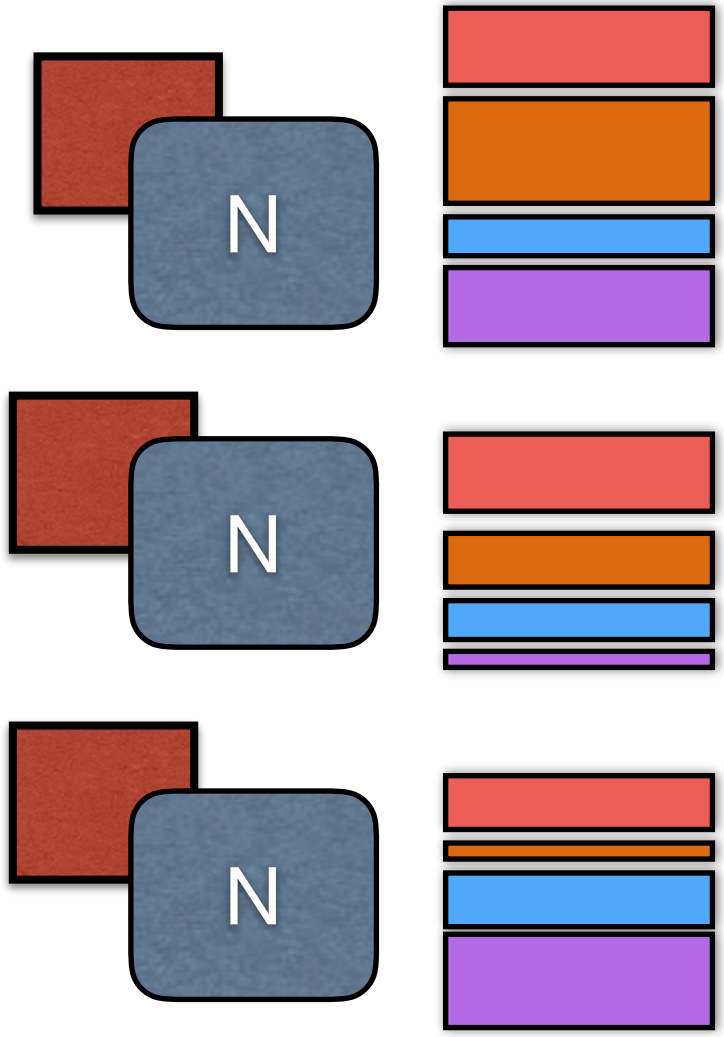
Map Reduce



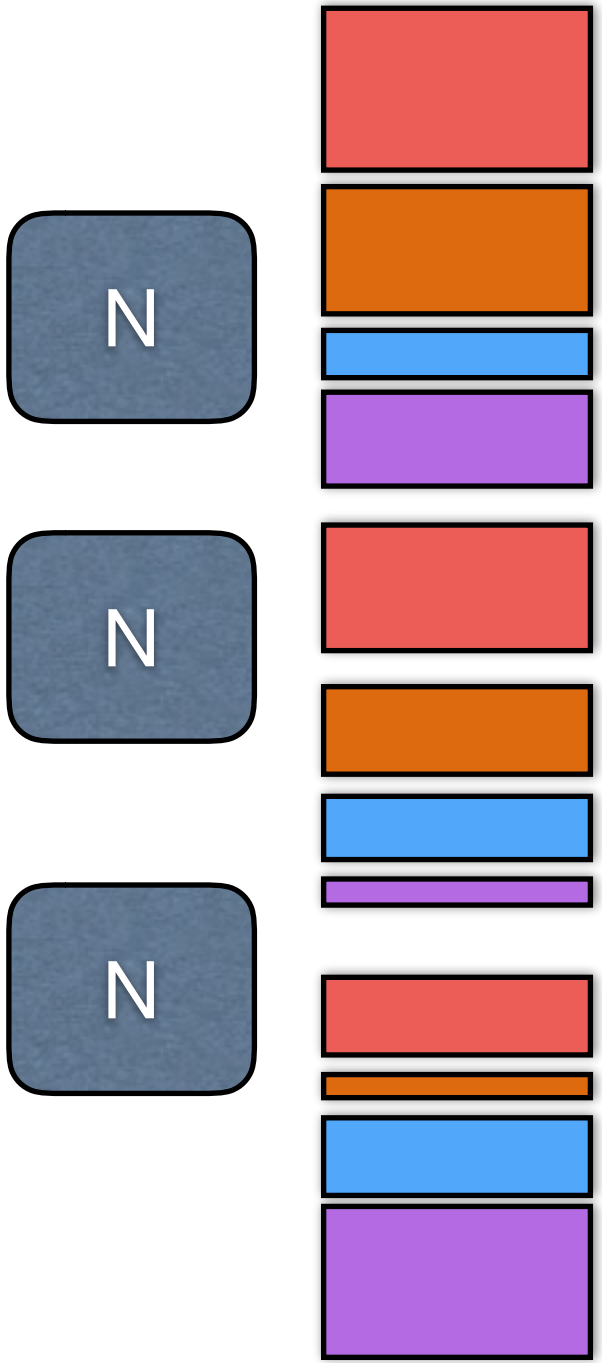
Map Reduce



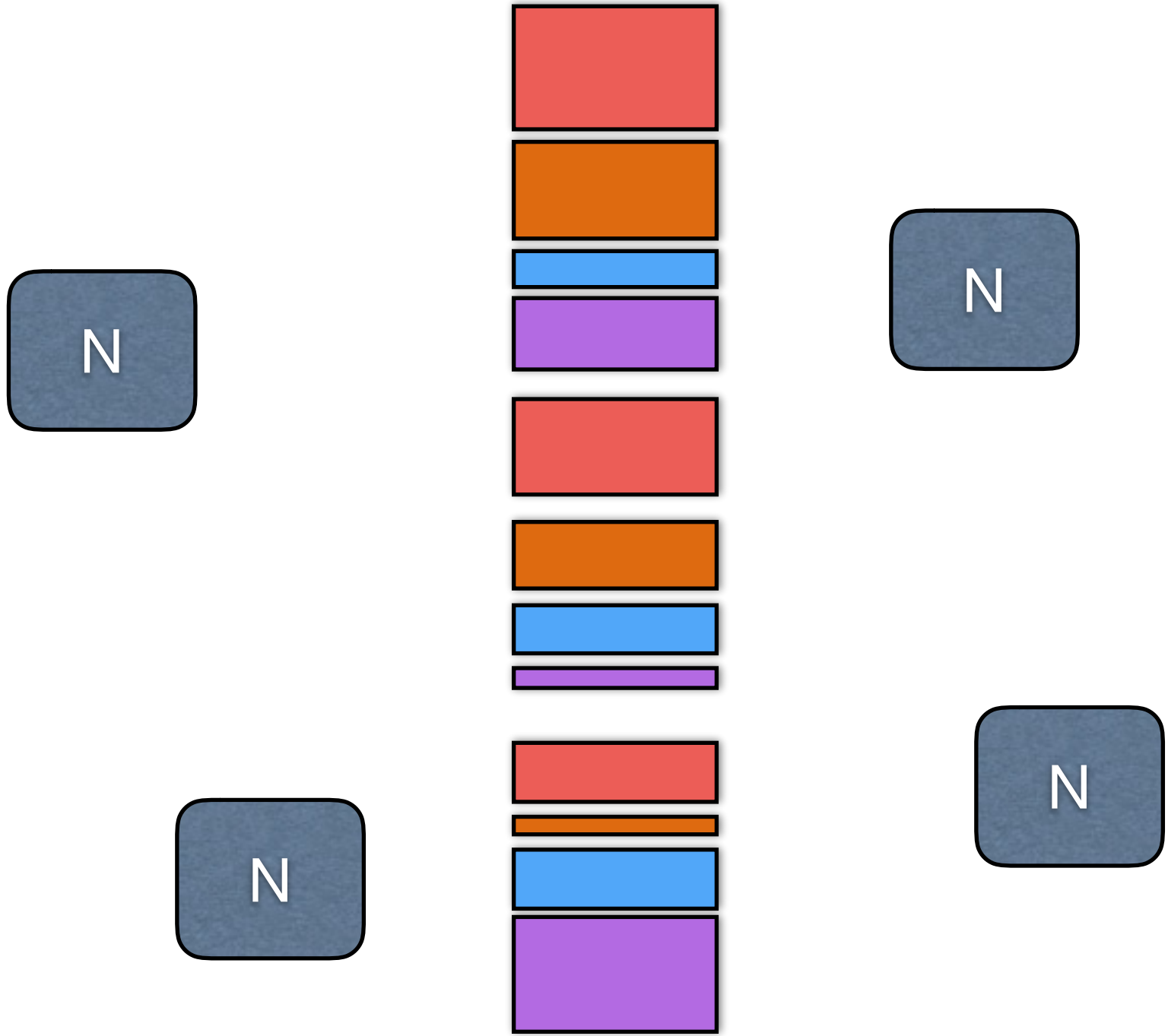
Map Reduce



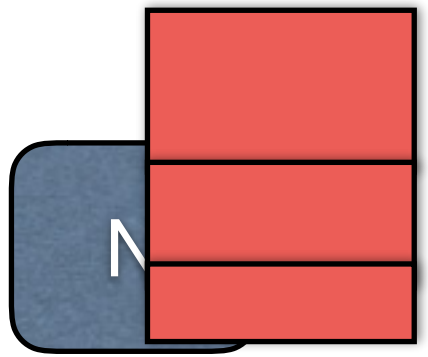
Map Reduce

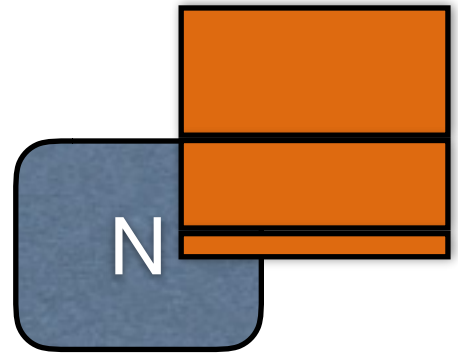


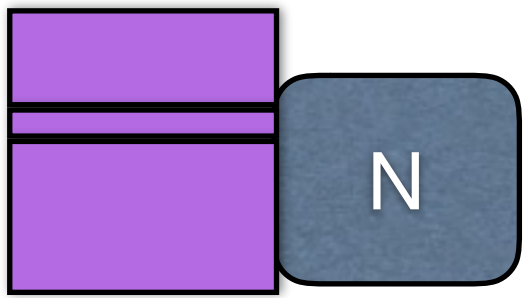
Map Reduce

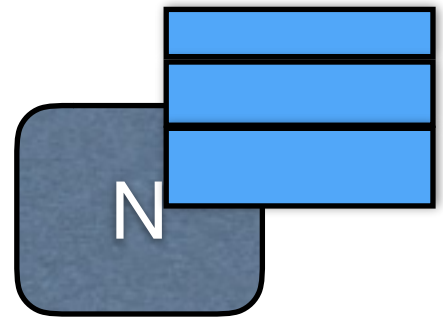


Map Reduce

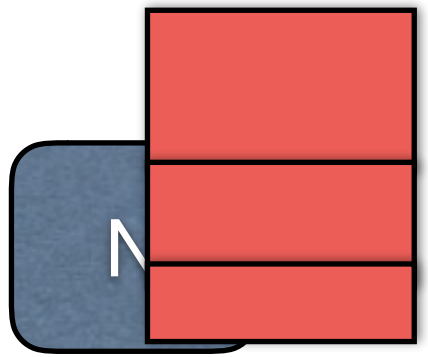


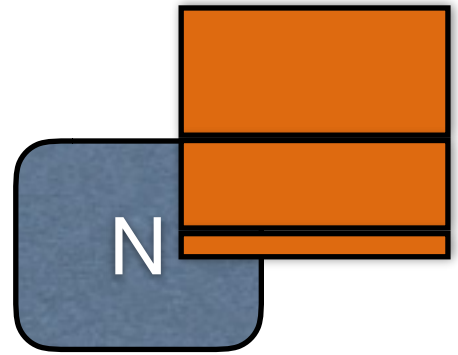


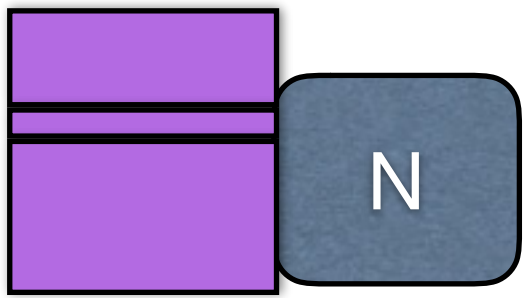


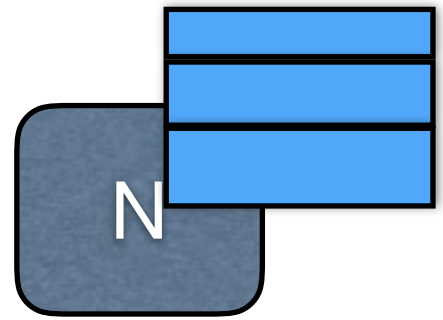


Map Reduce

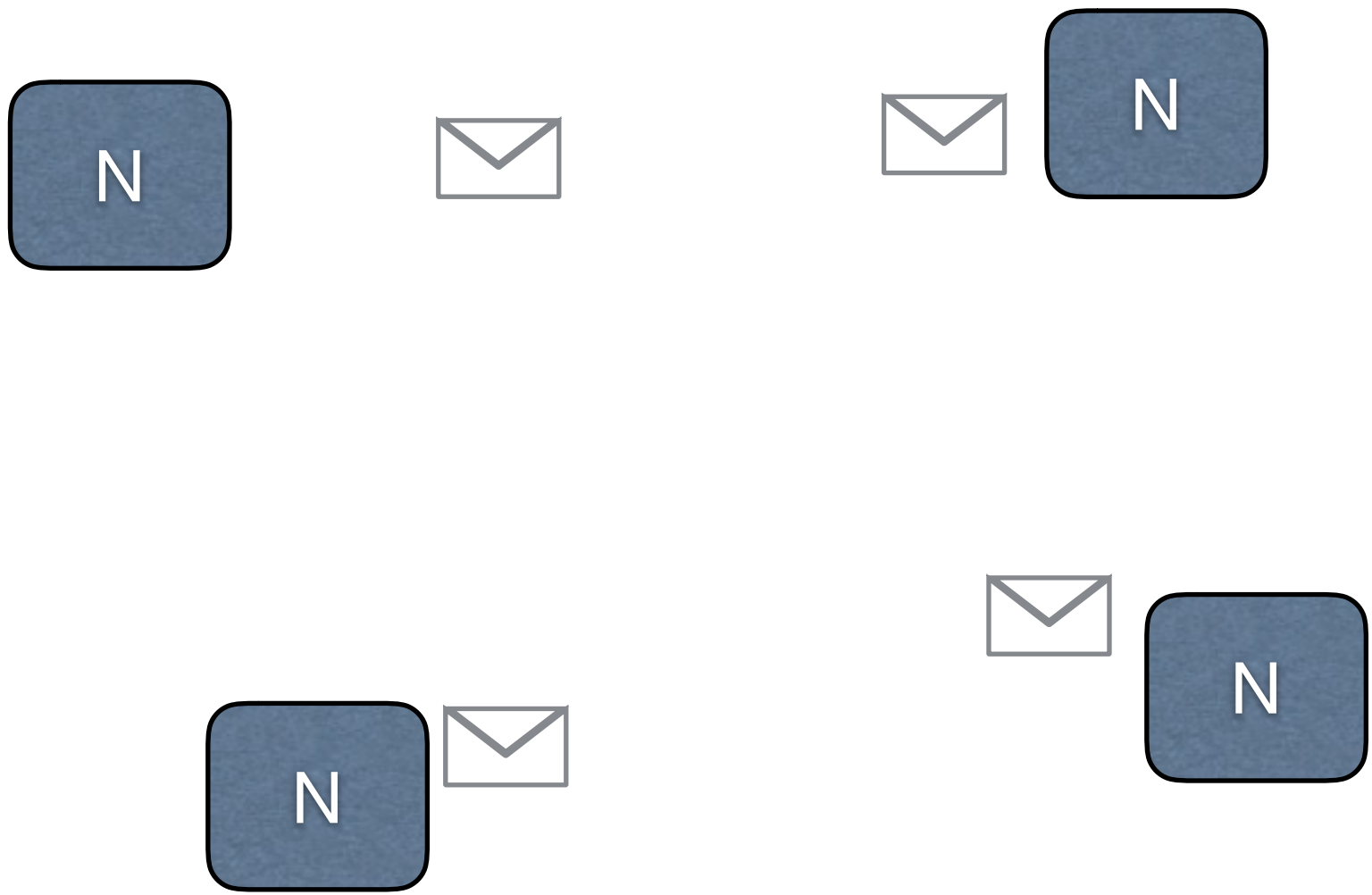








Map Reduce



Map Reduce

N

N

N

N

Other Considerations

Fault Tolerance

- Functional model makes this easy:
  - Schedule again on another node!
  - Caveat: deterministic functions, otherwise may get different results

Stragglers

- What if you have a few slow machines?
  - When near end of the run, reschedule all remaining tasks
  - Use first version of task that returns

Moving data vs. moving computation

- Expensive to copy large amounts of data around
  - GFS replicates data!
  - Scheduler tries as hard as possible to locate mappers/reducers where the data lives

Course Recap

Layers

- Storage stack is many layers deep
  - Software
  - Hardware
- Abstraction lets us drop/replace components without altering surrounding stack
- Specialization lets us optimize for specifics

Tradeoffs

- Many! Understand them and write better code!
- Avoid heuristics when theory can give you provable guarantees

Whether you use or develop storage systems, understanding their behavior will speed your apps.