



Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

Week 1: Introduction (1/2)
January 5, 2016

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at <http://lintool.github.io/bigdata-2016w/>



This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details

Questions, questions, questions...

Who am I?

What is big data?

Why big data?

What is this course about?

IIT



From the Ivory Tower...



... to building sh*t that works



UNIVERSITY OF
WATERLOO



... and back.



Big Data



Processes 20 PB a day (2008)
Crawls 20B web pages a day (2012)
Search index is 100+ PB (5/2014)
Bigtable serves 2+ EB, 600M QPS (5/2014)



400B pages, 10+
PB (2/2014)



19 Hadoop clusters: 600
PB, 40k servers (9/2015)



Hadoop: 10K nodes, 150K
cores, 150 PB (4/2014)

300 PB data in Hive +
600 TB/day (4/2014)



S3: 2T objects, 1.1M request/
second (4/2013)

640K ought to be
enough for anybody.



150 PB on 50k+ servers
running 15k apps (6/2011)

LHC: ~15 PB a year



LSST: 6-10 PB a year
(~2020)

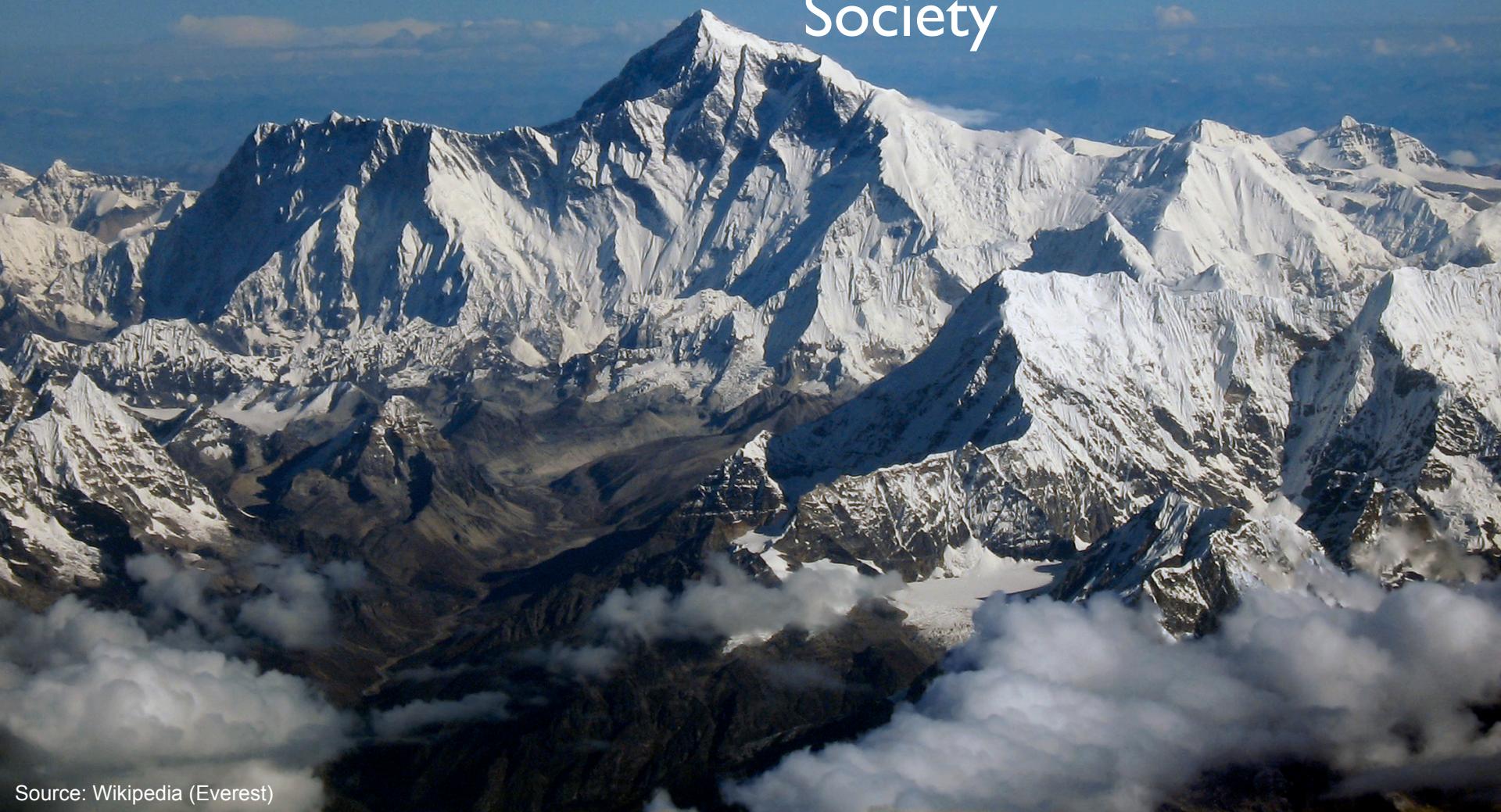


SKA: 0.3 – 1.5 EB
per year (~2020)

How much data?

Why big data?

Science
Engineering
Commerce
Society





Science

Emergence of the 4th Paradigm

Data-intensive e-Science

Engineering

The unreasonable effectiveness of data
Search, recommendation, prediction, ...





Know thy customers

Data → Insights → Competitive advantages

Commerce

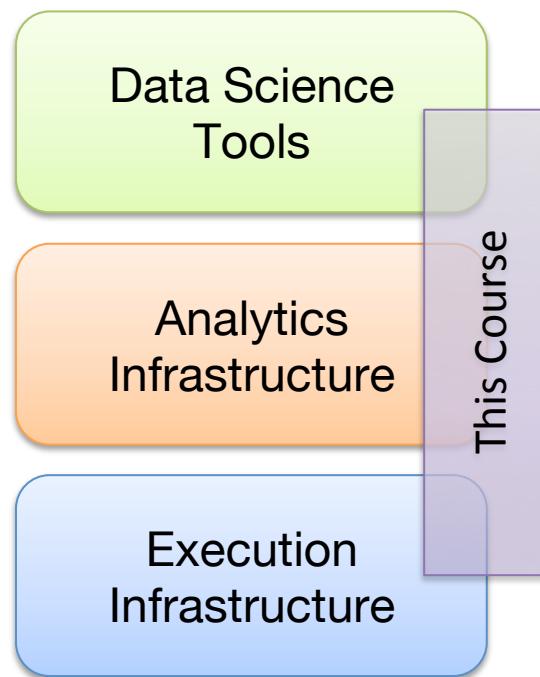
A wide-angle photograph of a massive outdoor gathering at night. In the foreground, a dense crowd of people is visible, many holding up small lights or flags. A stage is set up in the lower right, featuring a podium and several American flags. The background shows a city skyline with numerous lit-up skyscrapers, including one with a prominent 'USA' sign. The overall atmosphere is one of a major public event or rally.

Society

Humans as social sensors

Computational social science

What is this course about?

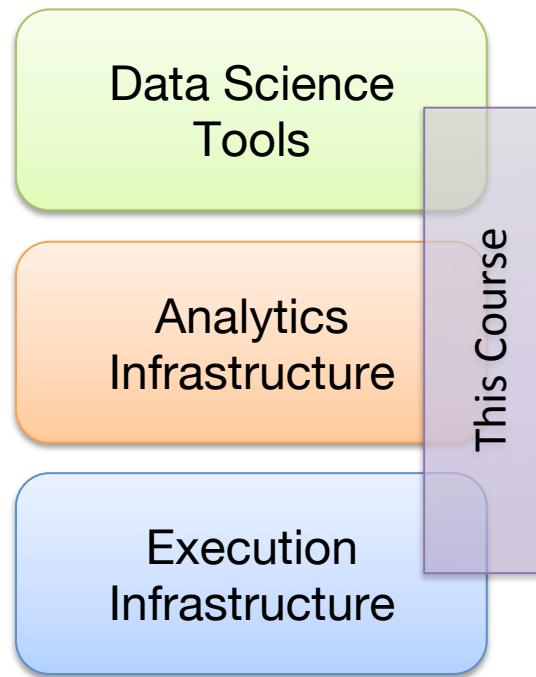


“big data stack”

Buzzwords

data analytics, business intelligence, OLAP, ETL, data warehouses and data lakes

MapReduce, Spark, noSQL, Flink, Pig, Hive, Dryad, Pregel, Giraph, Storm



Text: frequency estimation, language models, inverted indexes

Graphs: graph traversals, random walks (PageRank)

Relational data: SQL, joins, column stores

Data mining: hashing, clustering (k -means), classification, recommendations

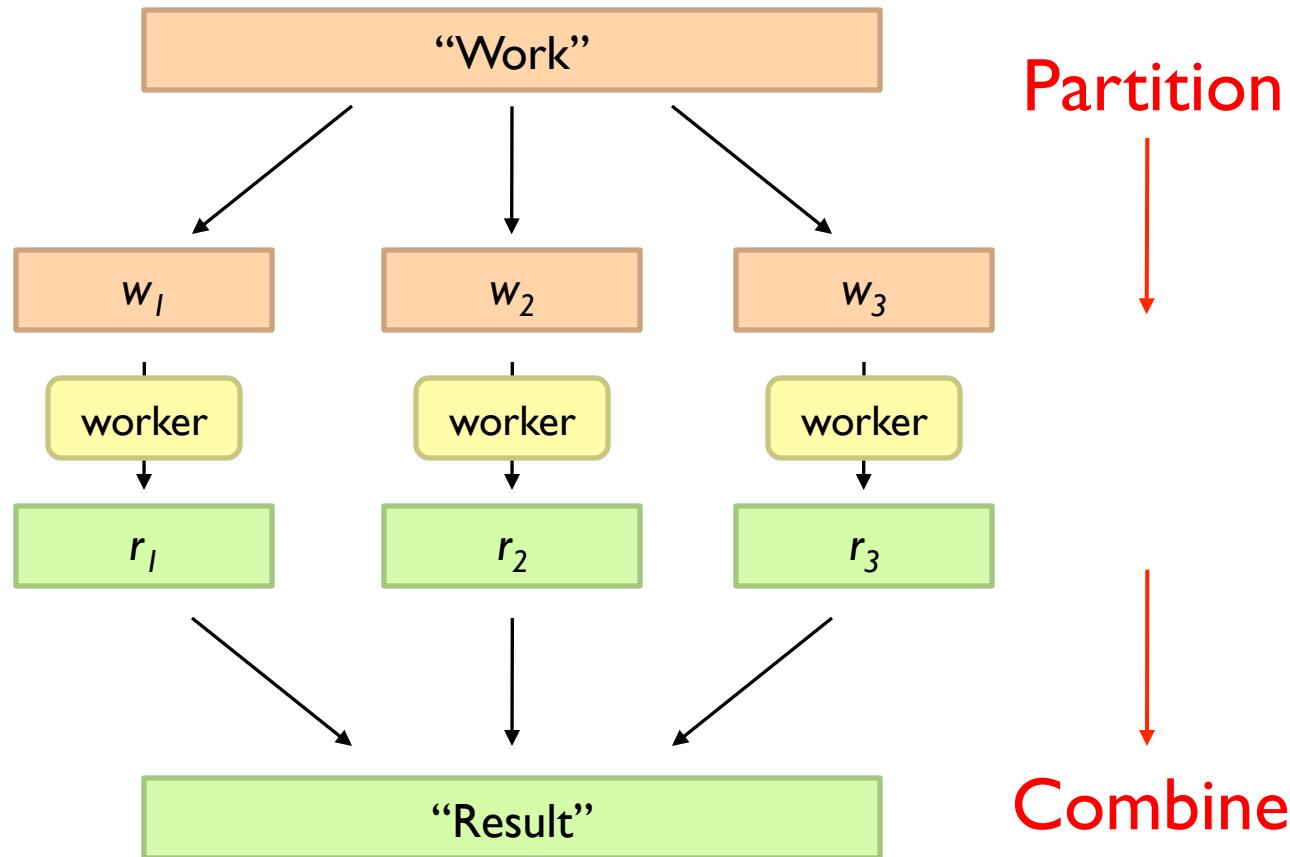
Streams: probabilistic data structures (Bloom filters, CMS, HLL counters)

This course focuses on algorithm design and “thinking at scale”

A wide-angle photograph of a massive server room. The space is filled with rows upon rows of server racks, their front panels glowing with various colors like blue, green, and yellow. A complex network of white and grey pipes and cables hangs from the ceiling in a dense web. The ceiling itself is a dark steel truss structure. The floor is made of large, light-colored tiles. The overall atmosphere is one of a high-tech, industrial facility.

Tackling Big Data

Divide and Conquer



Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

What's the common theme of all of these problems?

Common Theme?

- Parallelization problems arise from:
 - Communication between workers (e.g., to exchange state)
 - Access to shared resources (e.g., data)
- Thus, we need a synchronization mechanism

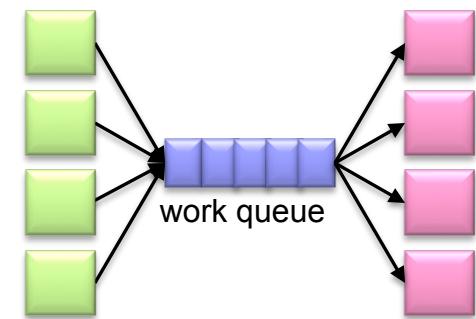
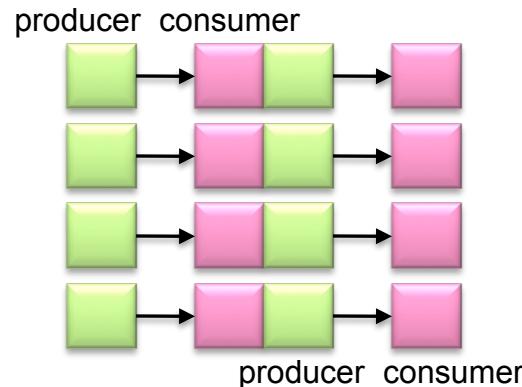
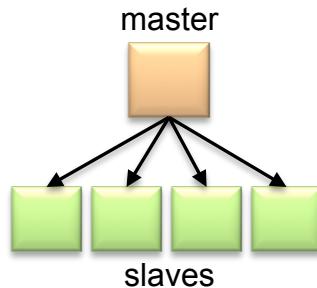
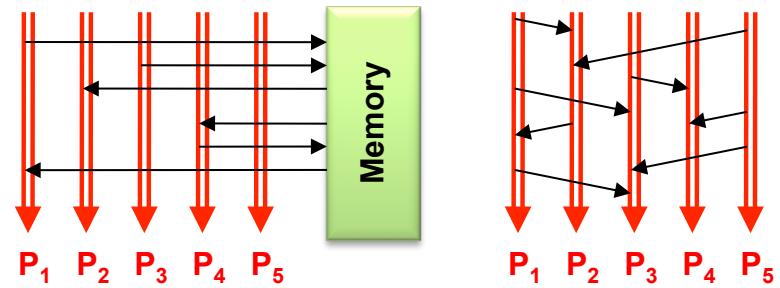


Managing Multiple Workers

- Difficult because
 - We don't know the order in which workers run
 - We don't know when workers interrupt each other
 - We don't know when workers need to communicate partial results
 - We don't know the order in which workers access shared data
- Thus, we need:
 - Semaphores (lock, unlock)
 - Conditional variables (wait, notify, broadcast)
 - Barriers
- Still, lots of problems:
 - Deadlock, livelock, race conditions...
 - Dining philosophers, sleeping barbers, cigarette smokers...
- Moral of the story: be careful!

Current Tools

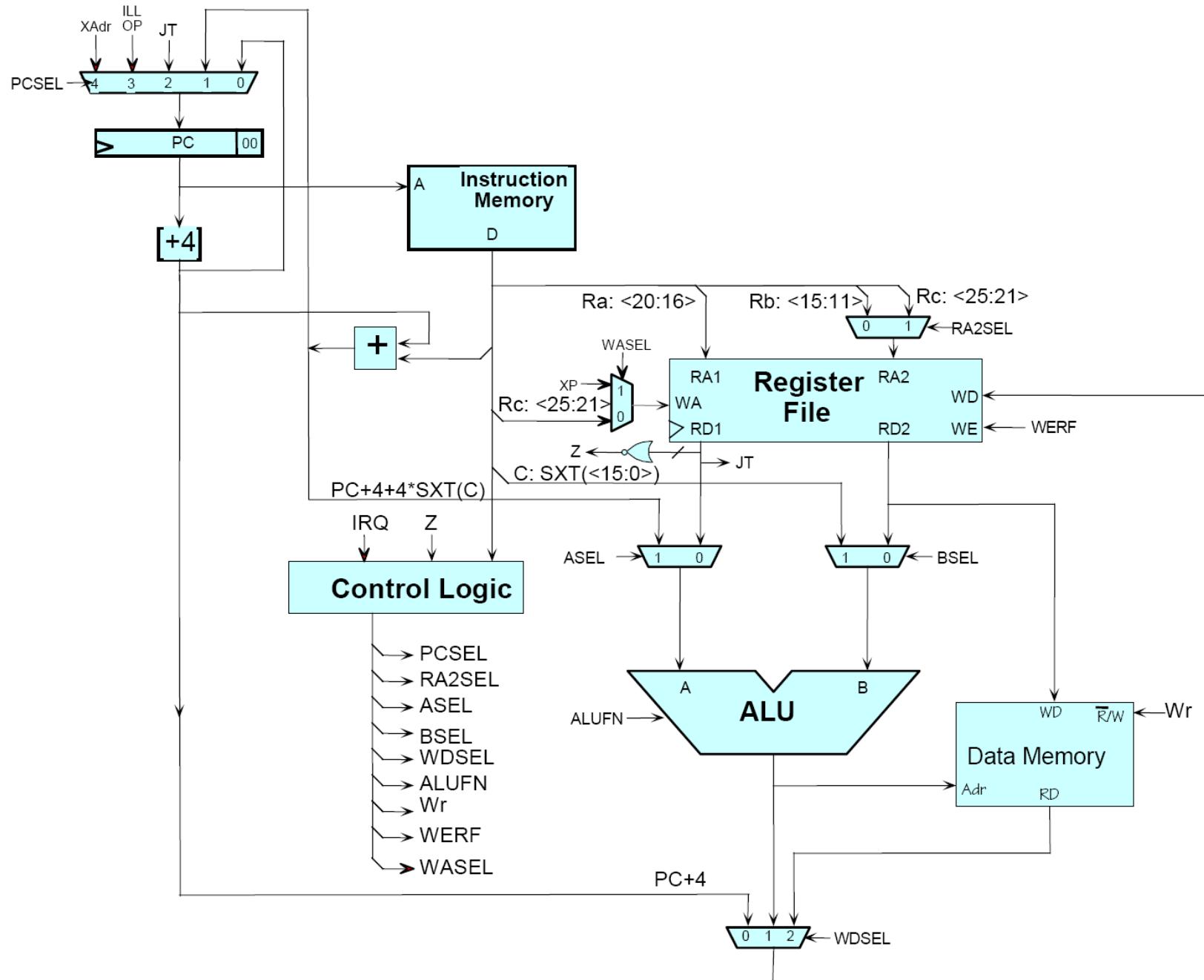
- Programming models
 - Shared memory (pthreads)
 - Message passing (MPI)
- Design Patterns
 - Master-slaves
 - Producer-consumer flows
 - Shared work queues

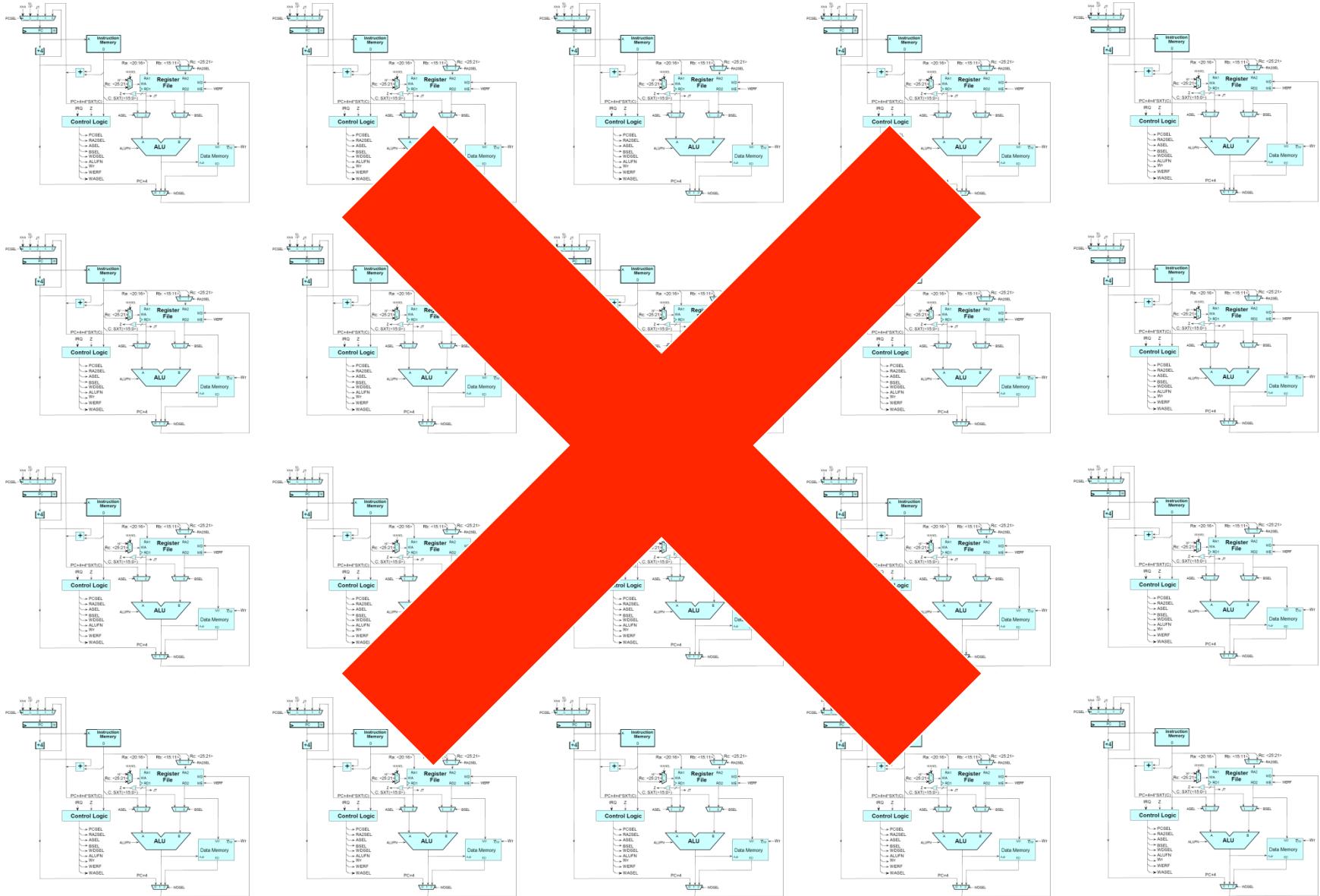


Where the rubber meets the road

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
 - At the scale of datacenters and across datacenters
 - In the presence of failures
 - In terms of multiple interacting services
- Not to mention debugging...
- The reality:
 - Lots of one-off solutions, custom code
 - Write your own dedicated library, then program with it
 - Burden on the programmer to explicitly manage everything







An aerial photograph of a large datacenter complex during sunset. The sky is a vibrant orange and yellow. In the foreground, there are several large white industrial buildings, parking lots, and rows of white shipping containers. A major highway runs through the middle ground. The background shows a vast, green, agricultural landscape stretching to a distant horizon under a hazy sky.

The datacenter *is* the computer!

The datacenter is the computer

- It's all about the right level of abstraction
 - Moving beyond the von Neumann architecture
 - What's the “instruction set” of the datacenter computer?
- Hide system-level details from the developers
 - No more race conditions, lock contention, etc.
 - No need to explicitly worry about reliability, fault tolerance, etc.
- Separating the *what* from the *how*
 - Developer specifies the computation that needs to be performed
 - Execution framework (“runtime”) handles actual execution

MapReduce is the first instantiation of this idea...

MapReduce

A wide-angle photograph of a massive server room, likely a Google data center. The room is filled with floor-to-ceiling server racks, their front panels glowing with various colors (blue, yellow, green) from integrated LED lights. A complex network of grey metal walkways and support structures spans the entire space, with stairs leading up to different levels. The ceiling is a dark, multi-layered steel truss structure with recessed lighting. The overall atmosphere is cool and industrial, with a strong blue tint from the artificial lighting.

Typical Big Data Problem

- Iterate over a large number of records

Map Extract something of interest from each

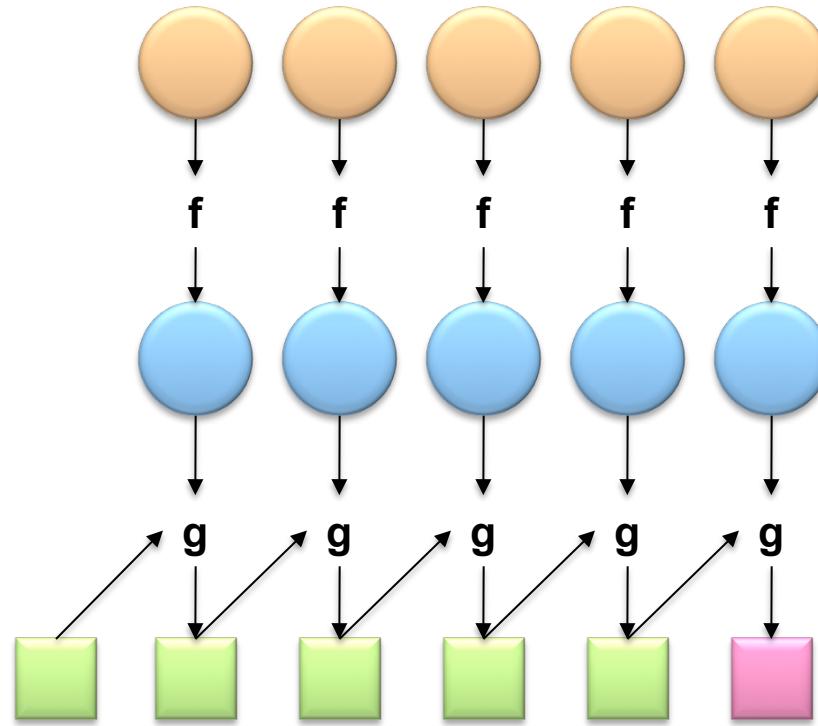
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Reduce

Key idea: provide a functional abstraction for these two operations

Roots in Functional Programming

Map



Fold

```
scala> val t = Array(1, 2, 3, 4, 5)
t: Array[Int] = Array(1, 2, 3, 4, 5)
```

```
scala> t.map(n => n*n)
res0: Array[Int] = Array(1, 4, 9, 16, 25)
```

```
scala> t.map(n => n*n).foldLeft(0)((m, n) => m + n)
res1: Int = 55
```

**Functional programming +
distributed computing!**

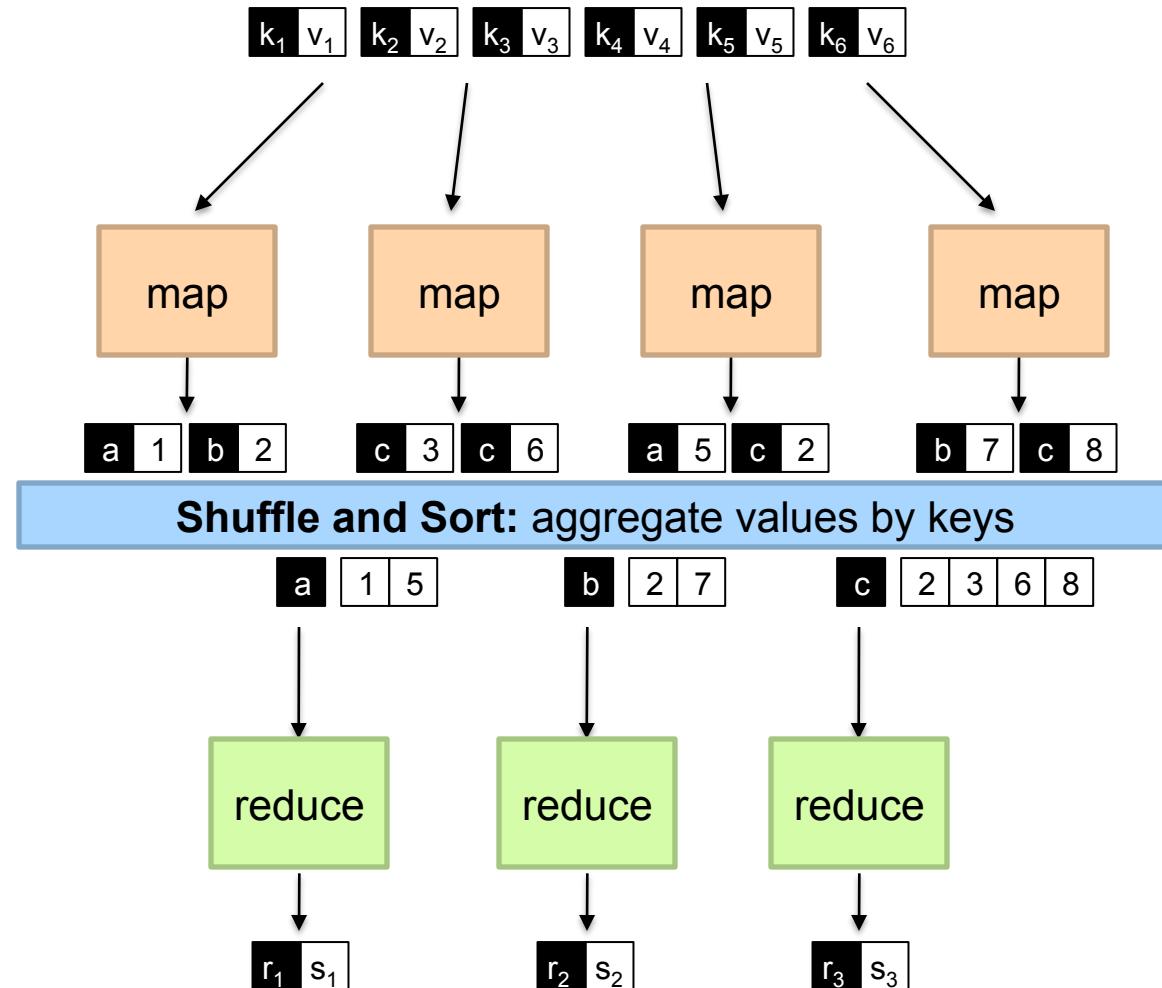
MapReduce

- Programmers specify two functions:

map (k_1, v_1) \rightarrow [k_2, v_2]

reduce ($k_2, [v_2]$) \rightarrow [k_3, v_3]

- All values with the same key are sent to the same reducer
- The execution framework handles everything else...



MapReduce

- Programmers specify two functions:
map $(k, v) \rightarrow \langle k', v' \rangle^*$
reduce $(k', v') \rightarrow \langle k', v' \rangle^*$
 - All values with the same key are sent to the same reducer
- The execution framework handles everything else...

What's “everything else”?

MapReduce “Runtime”

- Handles scheduling
 - Assigns workers to map and reduce tasks
- Handles “data distribution”
 - Moves processes to data
- Handles synchronization
 - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
 - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

MapReduce

- Programmers specify two functions:

map (k, v) $\rightarrow \langle k', v' \rangle^*$

reduce (k', v') $\rightarrow \langle k', v' \rangle^*$

- All values with the same key are reduced together

- The execution framework handles everything else...

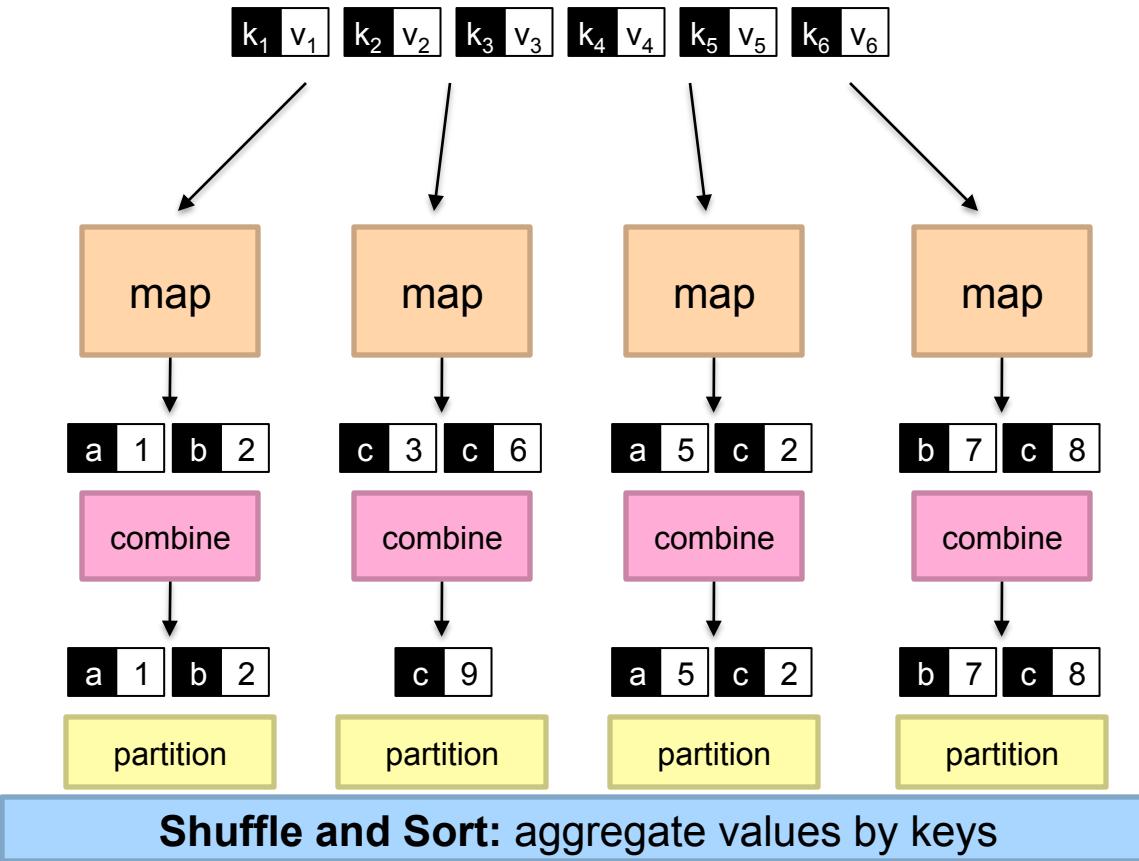
- Not quite...usually, programmers also specify:

partition (k' , number of partitions) \rightarrow partition for k'

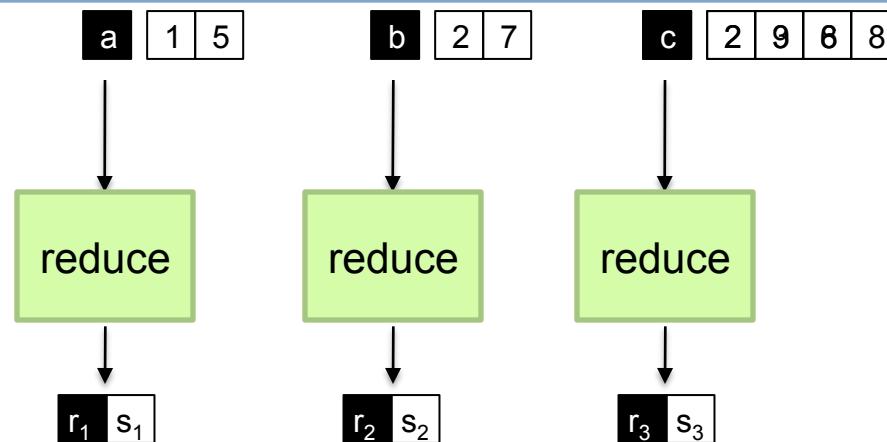
- Often a simple hash of the key, e.g., $\text{hash}(k') \bmod n$
- Divides up key space for parallel reduce operations

combine (k', v') $\rightarrow \langle k', v' \rangle^*$

- Mini-reducers that run in memory after the map phase
- Used as an optimization to reduce network traffic



Shuffle and Sort: aggregate values by keys



Two more details...

- Barrier between map and reduce phases
 - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
 - No enforced ordering across reducers

“Hello World”: Word Count

Map(String docid, String text):

for each word w in text:

 Emit(w, 1);

Reduce(String term, Iterator<Int> values):

 int sum = 0;

 for each v in values:

 sum += v;

 Emit(term, sum);

MapReduce can refer to...

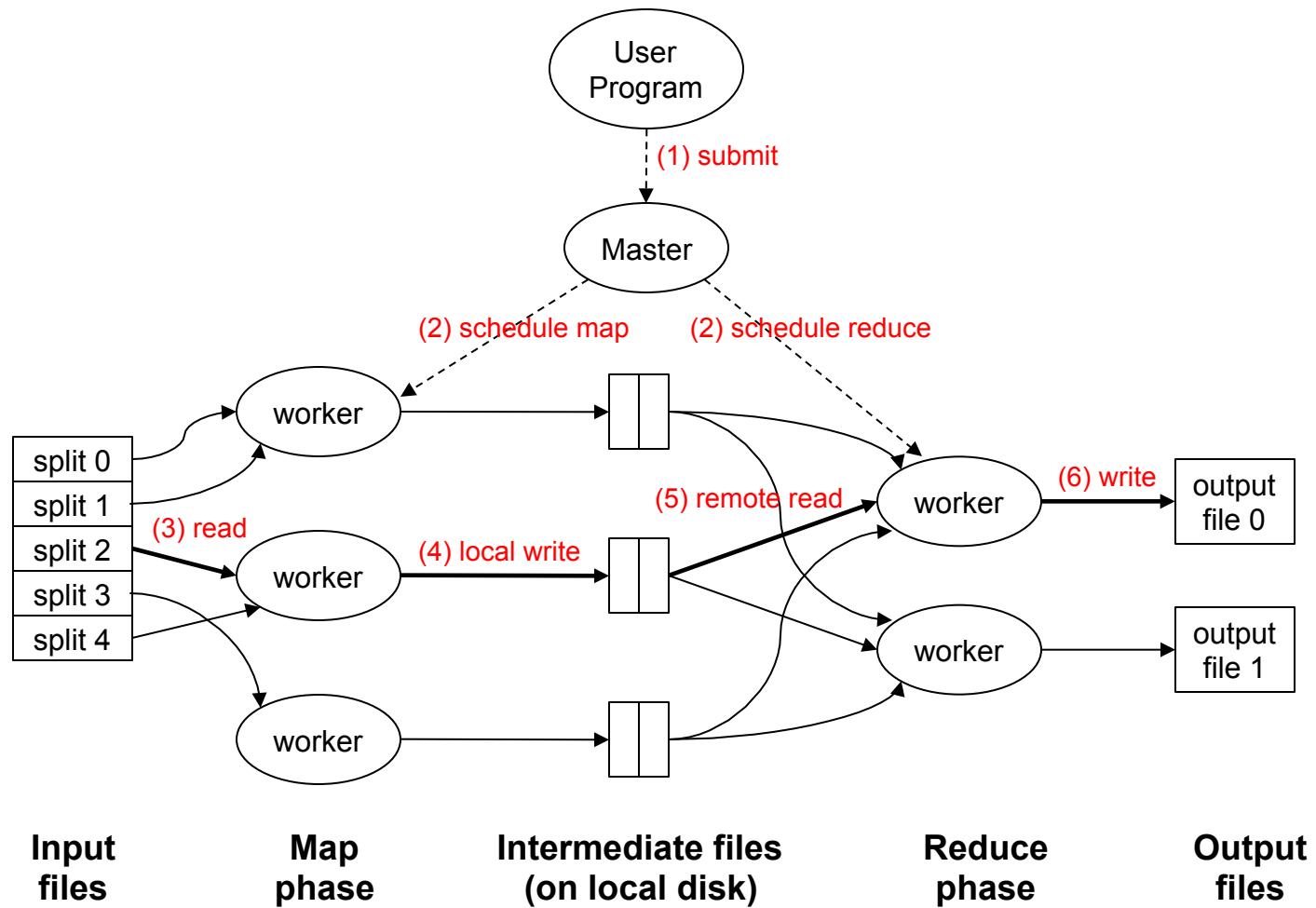
- The programming model
- The execution framework (aka “runtime”)
- The specific implementation

Usage is usually clear from context!

MapReduce Implementations

- Google has a proprietary implementation in C++
 - Bindings in Java, Python
- Hadoop provides an open-source implementation in Java
 - Development led by Yahoo, now an Apache project
 - Used in production at Yahoo, Facebook, Twitter, LinkedIn, Netflix, ...
 - Large and expanding software ecosystem
 - Potential point of confusion: Hadoop is more than MapReduce today
- Lots of custom research implementations





We'll discuss physical execution in detail later...

A grid of approximately 100 small wooden stick figures arranged in 10 rows and 10 columns. Each figure has a small wooden head, thin arms, and legs made of twigs. They are dressed in colorful, triangular wooden skirts in various colors including yellow, orange, red, maroon, pink, purple, blue, green, and light green. The figures are positioned with their arms raised and legs spread, resembling a group of dancing or celebrating figures.

Course Administrivia

My Expectations

- Your background:
 - Pre-reqs: CS 341, CS 350, (CS 348)
 - Comfortable in Java and Scala
 - Know how to use Git
 - Reasonable “command-line”-fu skills
 - Experience in compiling, patching, and installing open source software
 - Good debugging skills
- You are:
 - Genuinely interested in the topic
 - Be prepared to put in the time
 - Comfortable with the uncertainty and unpredictability that comes with cutting-edge, immature software

Course Design

- Course website: <https://www.student.cs.uwaterloo.ca/~cs489/>
Redirects to: <http://lintool.github.io/bigdata-2016w/>
- Piazza: <https://piazza.com/uwaterloo.ca/winter2016/cs489698/>
- Bespin: <http://bespin.io/>
- This course focuses on algorithm design and “thinking at scale”
 - Not the “mechanics” (API, command-line invocations, et.)
 - You’re expected to pick up MapReduce/Spark with minimal help
- Components of the final grade:
 - Eight individual assignments: mix of MapReduce (Java) and Spark (Scala)
 - Final exam
 - CS 698: additional final project

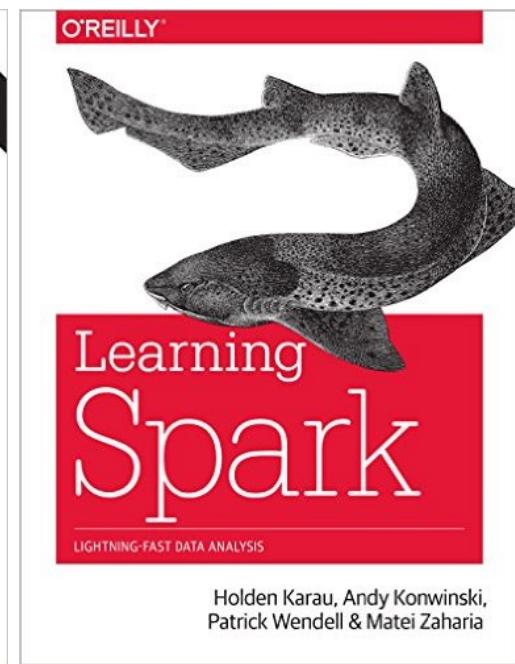
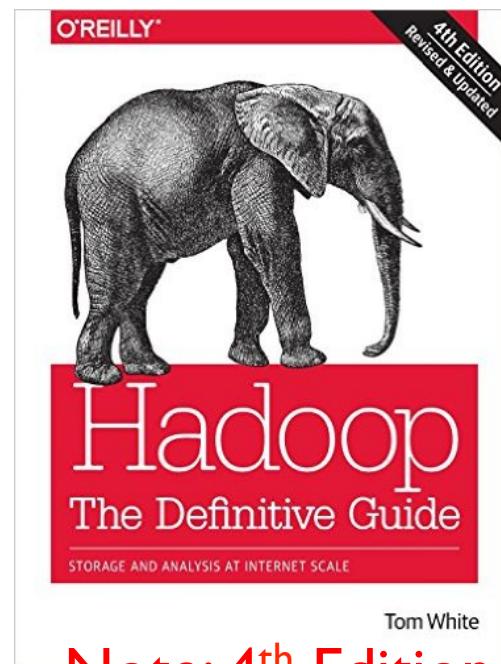
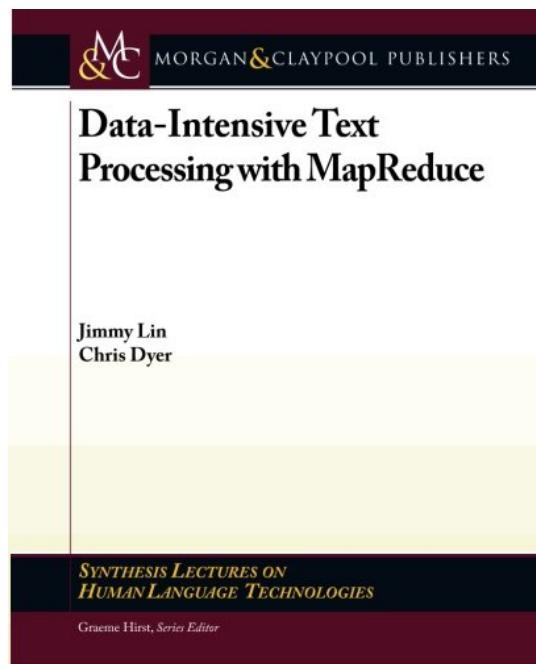
Assignment Mechanics

- We'll be using private GitHub repos for assignments
 - Complete your assignments, push to GitHub
 - We'll pull your repos at the deadline and grade
- Note late policy (details on course homepage)
 - Late by up to 24 hours: 25% reduction in grade
 - Late 24-48 hours: 50% reduction in grade
 - Late by more than 48 hours: not accepted
 - By assumption, we'll pull and grade at deadline – if you want us to hold off, you must let us know

Important: Register for (free) GitHub educational account!
https://education.github.com/discount_requests/new

Course Materials

One (required) textbook +
Two (optional but recommended) books +
Additional readings from research papers as appropriate



Note: 4th Edition

(optional but recommended)

MapReduce/Spark Environments

- Linux Student CS Environment
 - Everything is set up for you, just follow instructions
 - We'll make sure everything works
- Local installations
 - Install all software components on your own machine
 - Requires at least 4GB RAM + 10s GB disk for a good experience
 - Works fine on Mac and Linux, YMMV on Windows
 - We'll provide basic instructions, but not technical support
- Altiscale: Hadoop-as-a-Service
 - You'll be provided an account later

See “Software” page in course homepage for details:
<http://lintool.github.io/bigdata-2016w/software.html>

Be Prepared...



“Hadoop Zen”

- Parts of the ecosystem are still immature
 - We've come a long way since 2007, but still far to go...
 - Bugs, undocumented “features”, inexplicable behavior, etc.
 - Different versions = major pain
- Don't get frustrated (take a deep breath)...
 - Those W\$*#T@F! moments
- Be patient...
 - We will inevitably encounter “situations” along the way
- Be flexible...
 - We will have to be creative in workarounds
- Be constructive...
 - Tell me how I can make everyone's experience better

“Hadoop Zen”



A photograph of a traditional Japanese rock garden. In the foreground, there is a gravel path with several large, dark, angular stones. Beyond the gravel, a small pond contains more stones and is surrounded by various types of greenery, including low-lying bushes and larger trees with autumn-colored leaves. In the background, there are traditional wooden buildings with tiled roofs.

Questions?

To Do:

1. Bookmark course homepage
2. Get on Piazza
3. Register for GitHub educational account