EDM2016 Tutorial

Massively Scalable EDM with Spark

Tristan Nixon

Institute for Intelligent Systems

University of Memphis

t.nixon@memphis.edu

What is Spark?

- Apache Spark is a fast and general engine for large-scale data processing. (Using computing clusters)
- **<u>The</u>** pre-eminent platform for developing BigData processing applications
  - Used by most big internet companies (Facebook, Yahoo, LinkedIn, etc.)
  - But not Google…
  - But accessible to small teams, too!

What is Spark?

- OpenSource (Apache license)
- Runs on lots of cluster platforms
  - Also in local mode!
- Interacts with lots of popular systems
- Many APIs
  - SQL
  - Streaming
  - Machine-Learning
  - Graph processing
  - more coming all the time…

How Big is Big?

- Yesterday’s “Big” data is today’s data
- Too big to store/process on a single conventional computer
  - Billions of rows / records
  - Terabytes of data
- How to handle this?
  - Distribute storage, processing across machines

Distributed Systems

- Advantages:
  - Scalability - workload is partitioned
  - Resiliency - fault tolerant
    - Faults do not become failures
- Drawbacks:
  - Complexity!!!!
    - Setup, configuration, development, management, maintenance
  - Need tools, frameworks to manage this complexity

High Performance Computing

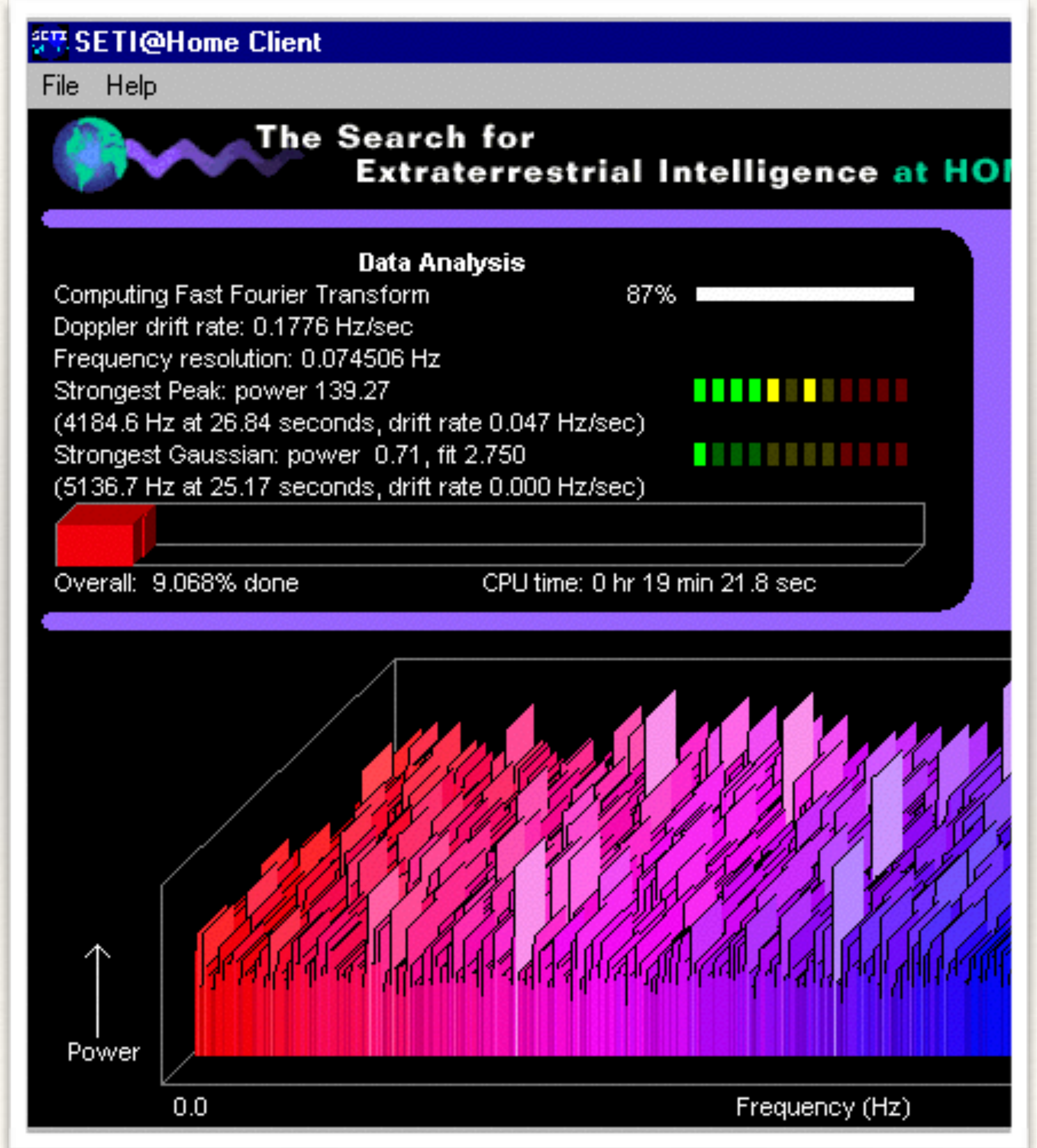
- Super computing
- Cluster computing
- Grid computing
- Different types of HPC for different kinds of problems:
  - What is the scale of your data?
  - How tightly coupled are your operations?
- Yields different architectures:
  - Speed of interconnection between processing units

Super Computing

- Great many processors (100K+)
- Connected by very fast custom buses / networking
- Often with shared/pooled memory, storage
- Highly coupled computations
  - Detailed physics simulations
- Expensive custom hardware
- Highly reliable task completion

Grid Computing

- Thousands / millions of home computers
- Opportunistic computation when available
- Slow / unreliable home internet connections
- Completely isolated memory, storage
- Basic home PC hardware
- Good for moderately intensive computation on completely independent data chunks
- Not good for aggregating / summarizing across chunks
- No guarantees / control over task completion



Cluster computing

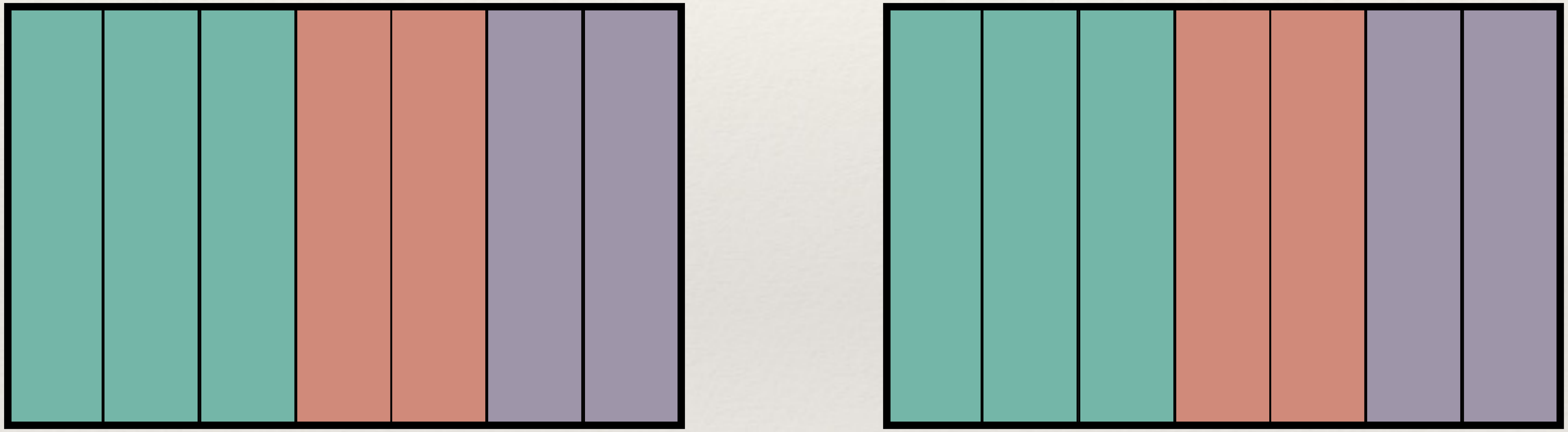
This one is just right!

- 100s-1,000s of server-grade machines
- Higher end commodity hardware
- Datacenter speeds (Gigabit ethernet+)
- Independent memory, pooled storage
- Good moderately coupled processing of big data!
- Reasonably high guarantees of task completion

Cluster Computing

Like RAID, but for servers!

RAID Mirroring



Cluster Computing

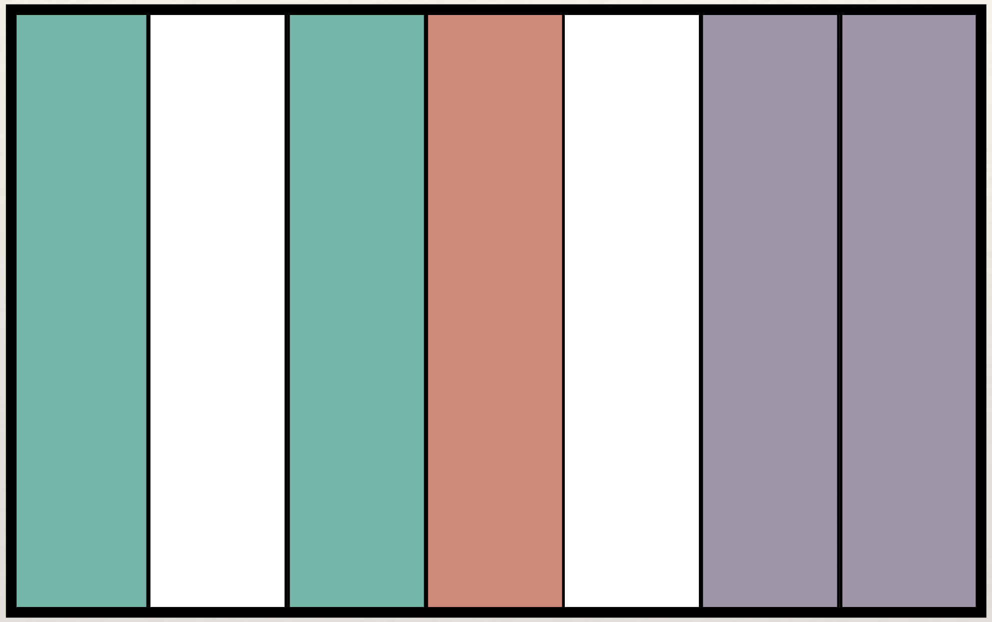
Like RAID, but for servers!

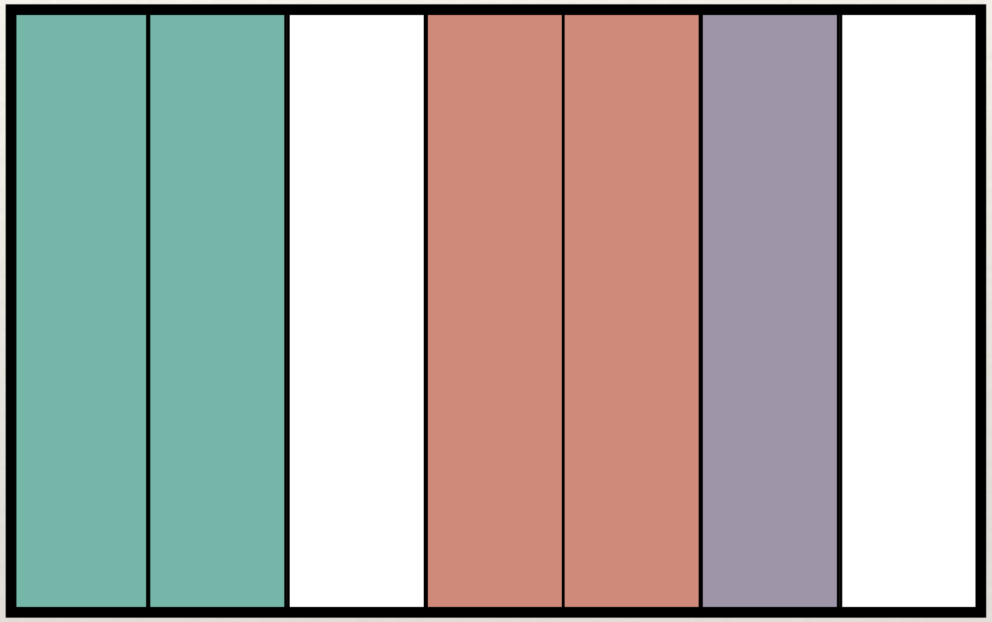
RAID Striping

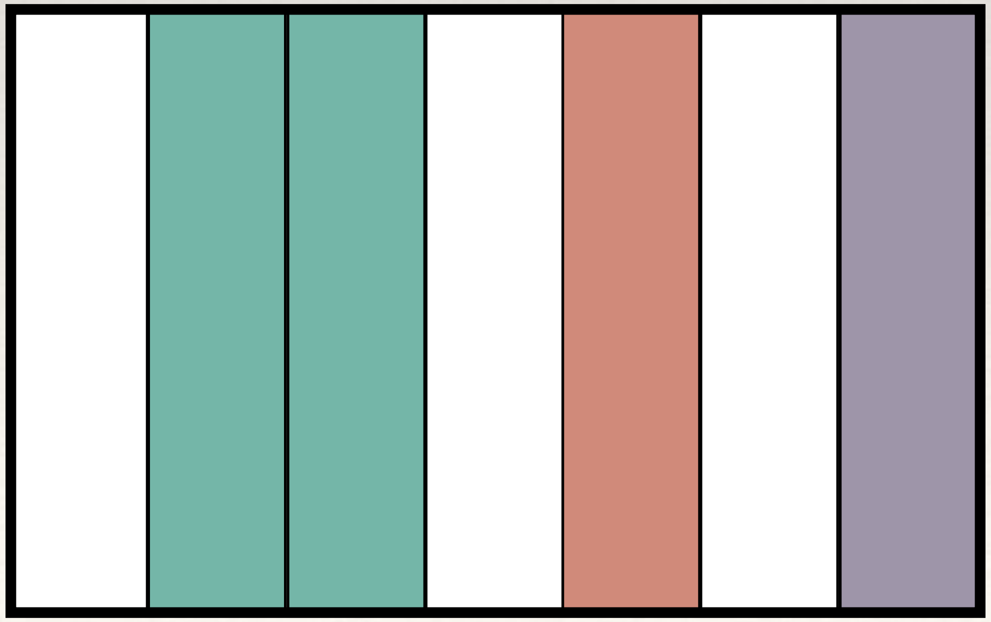
Cluster Computing

Like RAID, but for servers!

Mixed RAID Modes





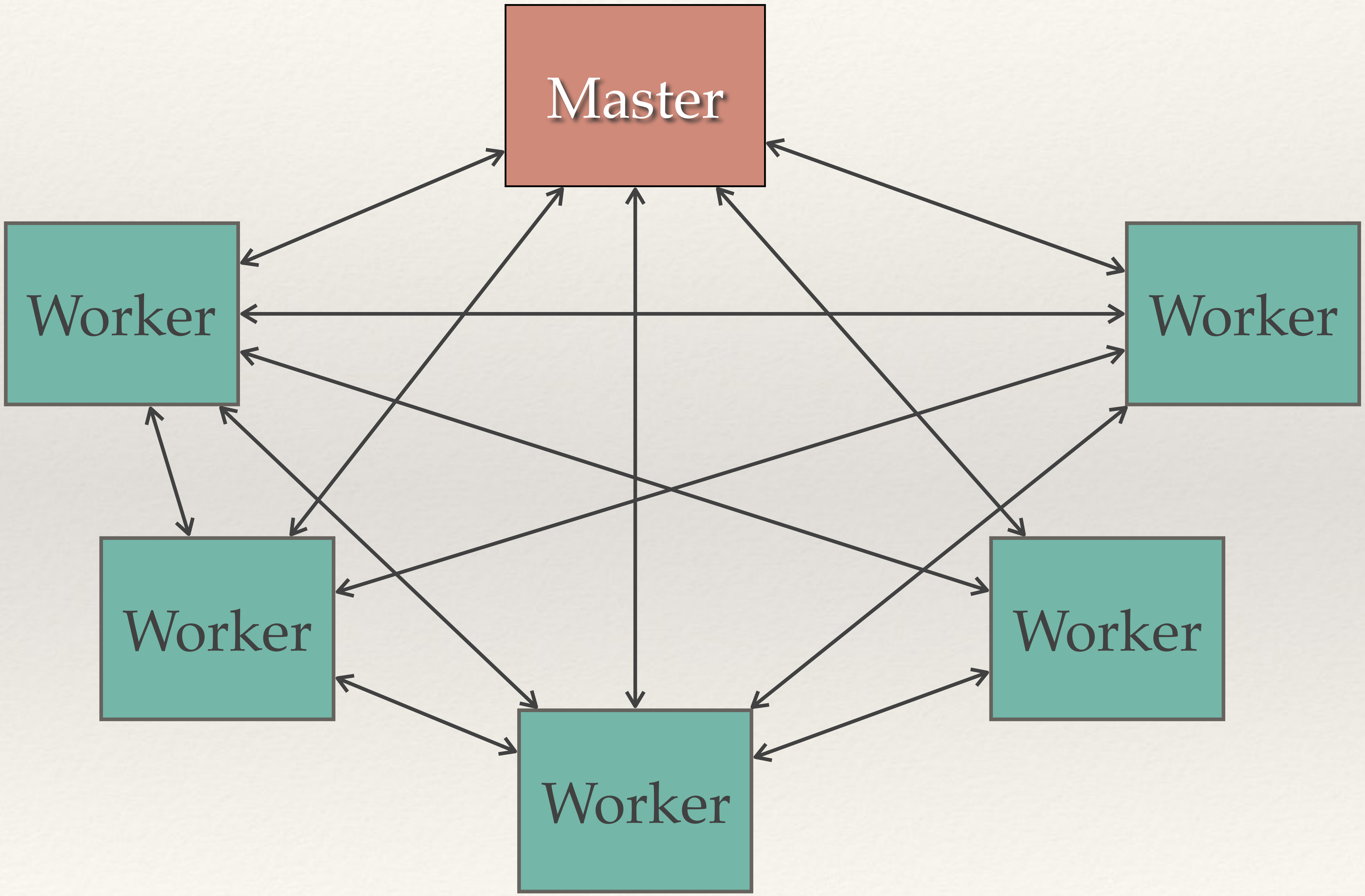


Cluster Computing

Like RAID, but for servers!

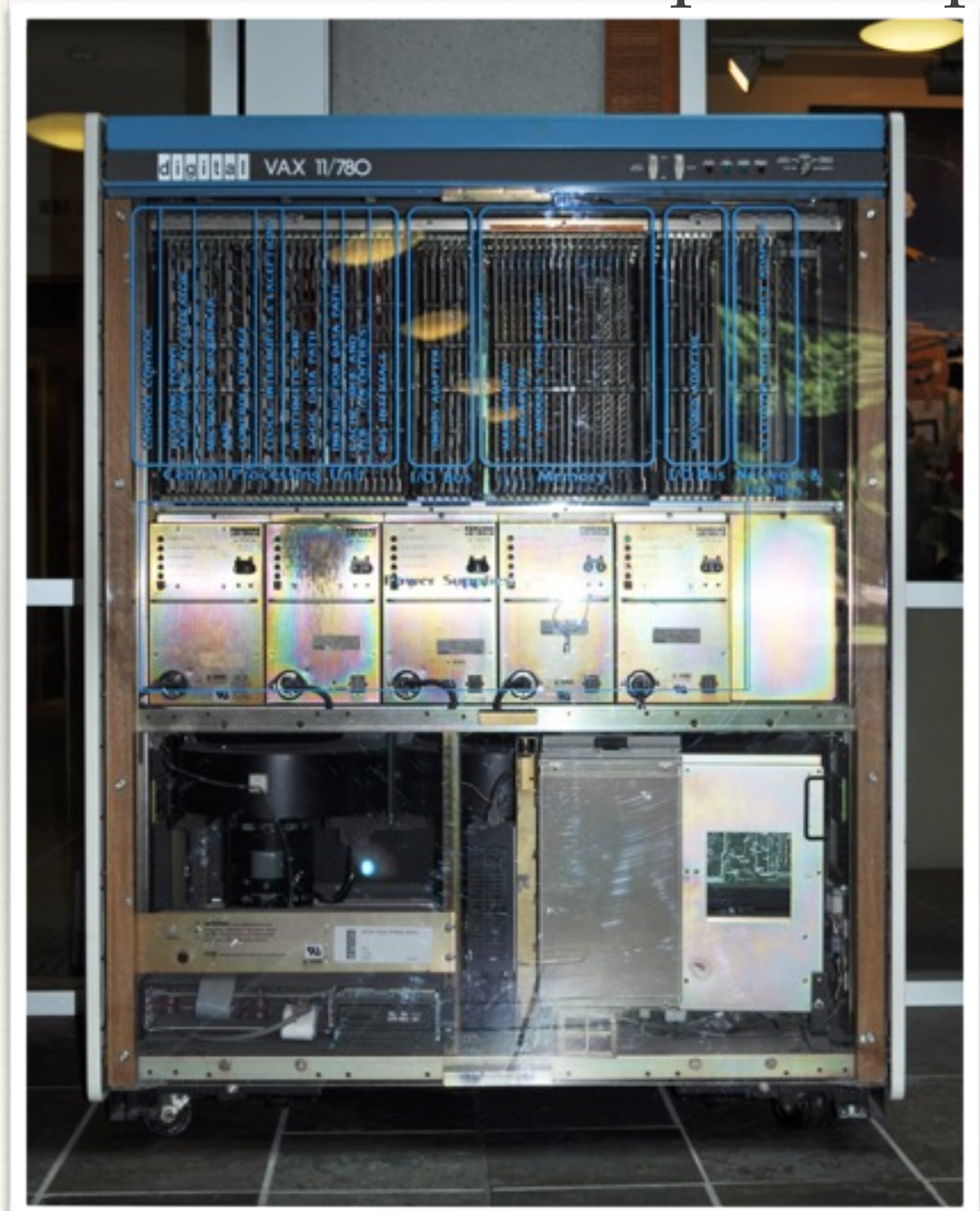
- Distributed, redundant storage of data
- Distributed, redundant computation on that data
- Scalability & Resiliency of both Storage & Computation!
- Computation makes use of data-locality!
  - Do the processing where the data is
  - Minimizes network lag
- But sometimes cluster synchronization is unavoidable

Cluster Topology



History of Clusters

'60s-'80s: Old-school supercomputers



History of Clusters

1995: Beowulf!

History of Clusters

Imagine a Beowulf cluster of these!

History of Clusters

Imagine a Beowulf cluster of these!

History of Clusters

2003-2006: The Google Papers

- Researchers at Google publish 3 seminal papers:
    1. 2003: Google File System (GFS)
    2. 2004: Map-Reduce
    3. 2006: BigTable

History of Clusters

2006: Yahoo! wants one too!

- Researchers working on similar problems go to work at Yahoo!
- They implement the Google systems, but in Java:
- Hadoop!
  - HDFS = GFS
  - Hadoop MapReduce
  - HBase = BigTable
- By 2008, all Yahoo! searches run on Hadoop

History of Clusters

2010-today: the floodgates open

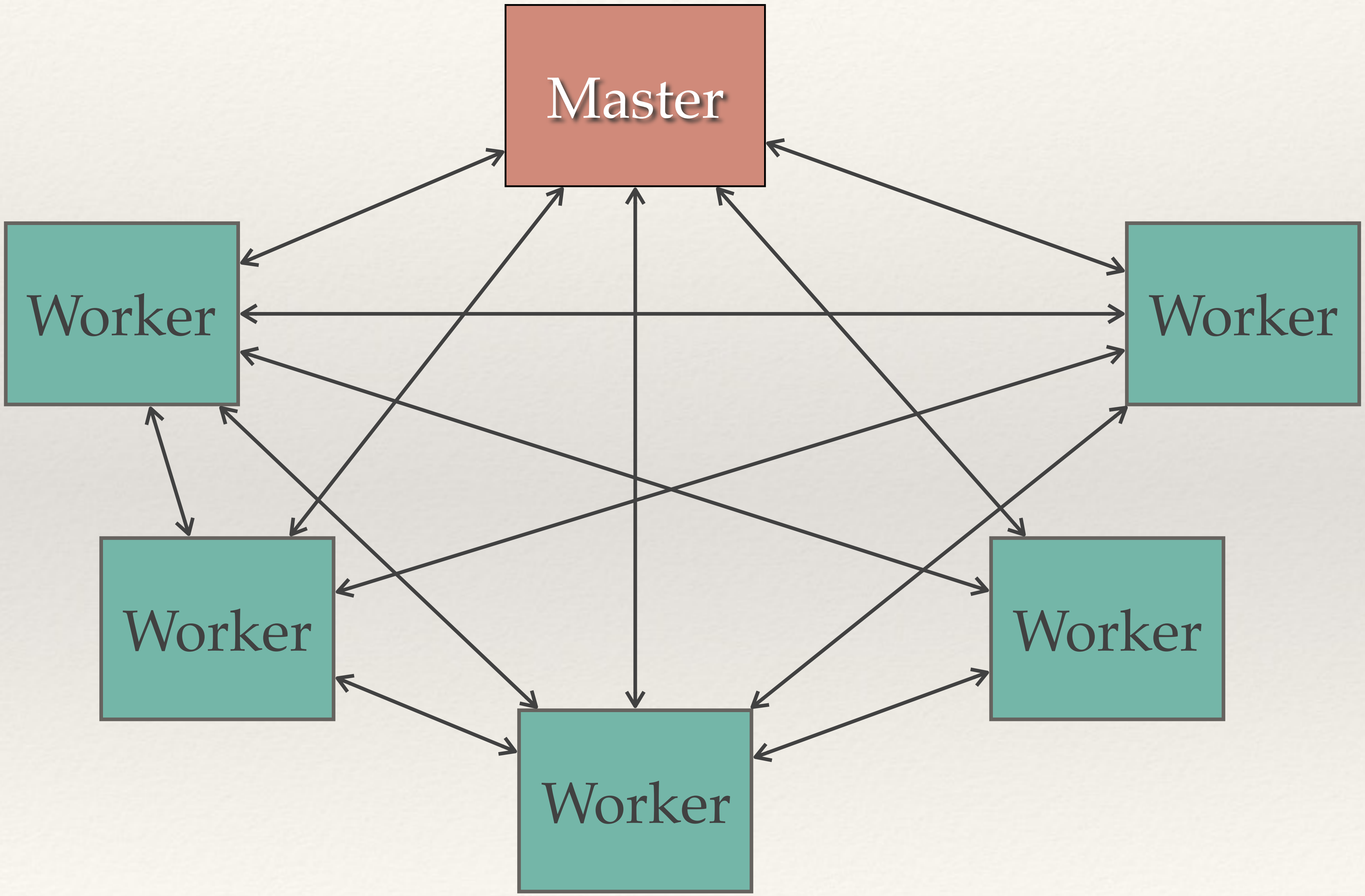
- New startups to support, augment Hadoop
- Lots of new tools, applications
- 2011: Hadoop 1.0
- 2013: Hadoop 2.2
  - Yarn!
- 2014: Spark 1.0
- 2016: Spark 1.6

Enough talk, show me the code!

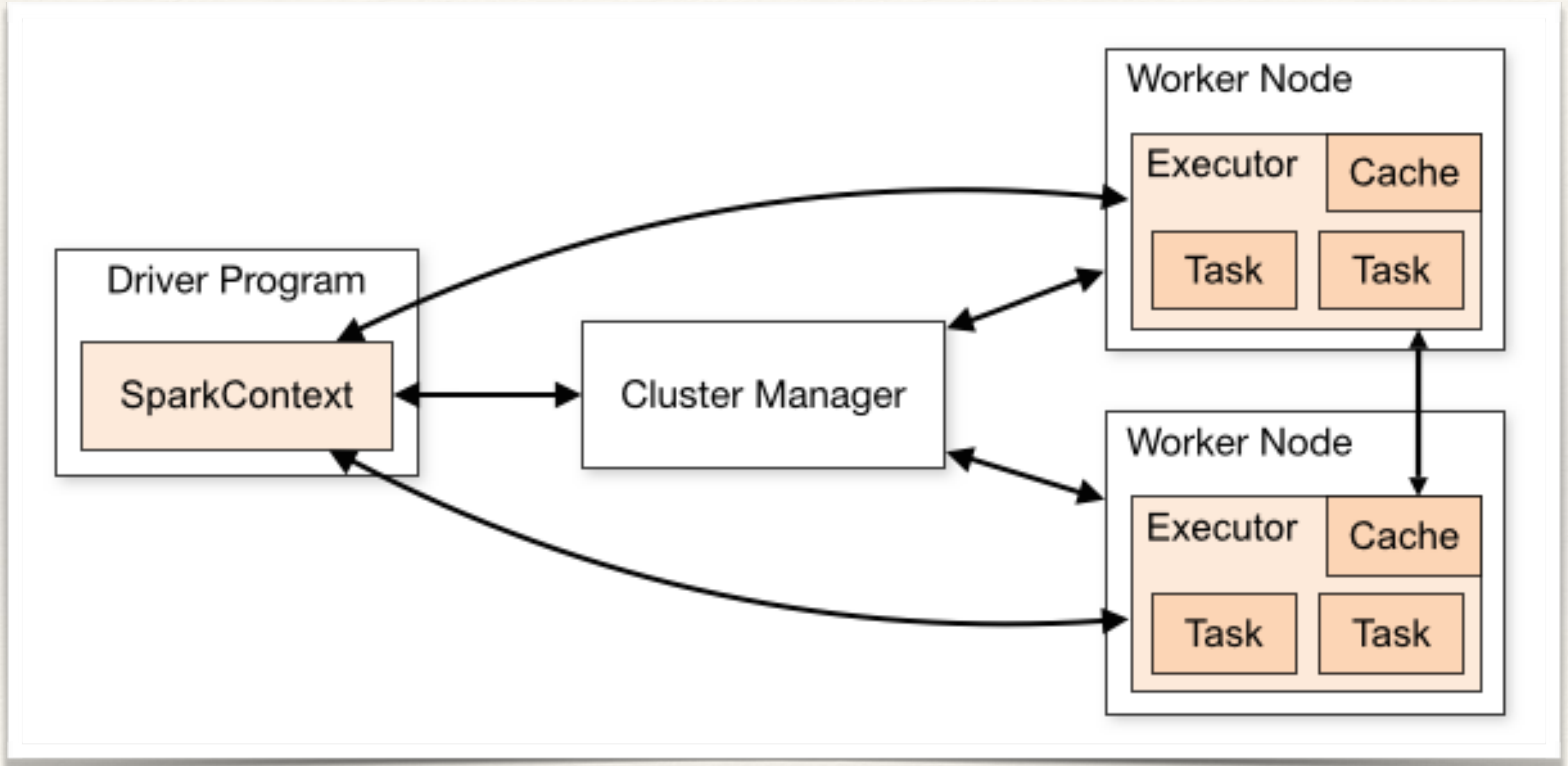
Exercise 1:

Map-Reduce
Word Count

Cluster Topology



Spark Topology



RDD: Resilient Distributed Dataset

- Central construct in Spark
- Logically like a List-type collection
- Internally partitioned
- Partitions distributed across cluster
- Partitioning scheme may vary based on source data
- Certain transformations cause shuffling, repartitioning

Item 1 Item 2 Item 3	Partition 1
Item 4 Item 5 Item 6 Item 7	Partition 2
Item 8 Item 9 Item 10	Partition 3

Transformations

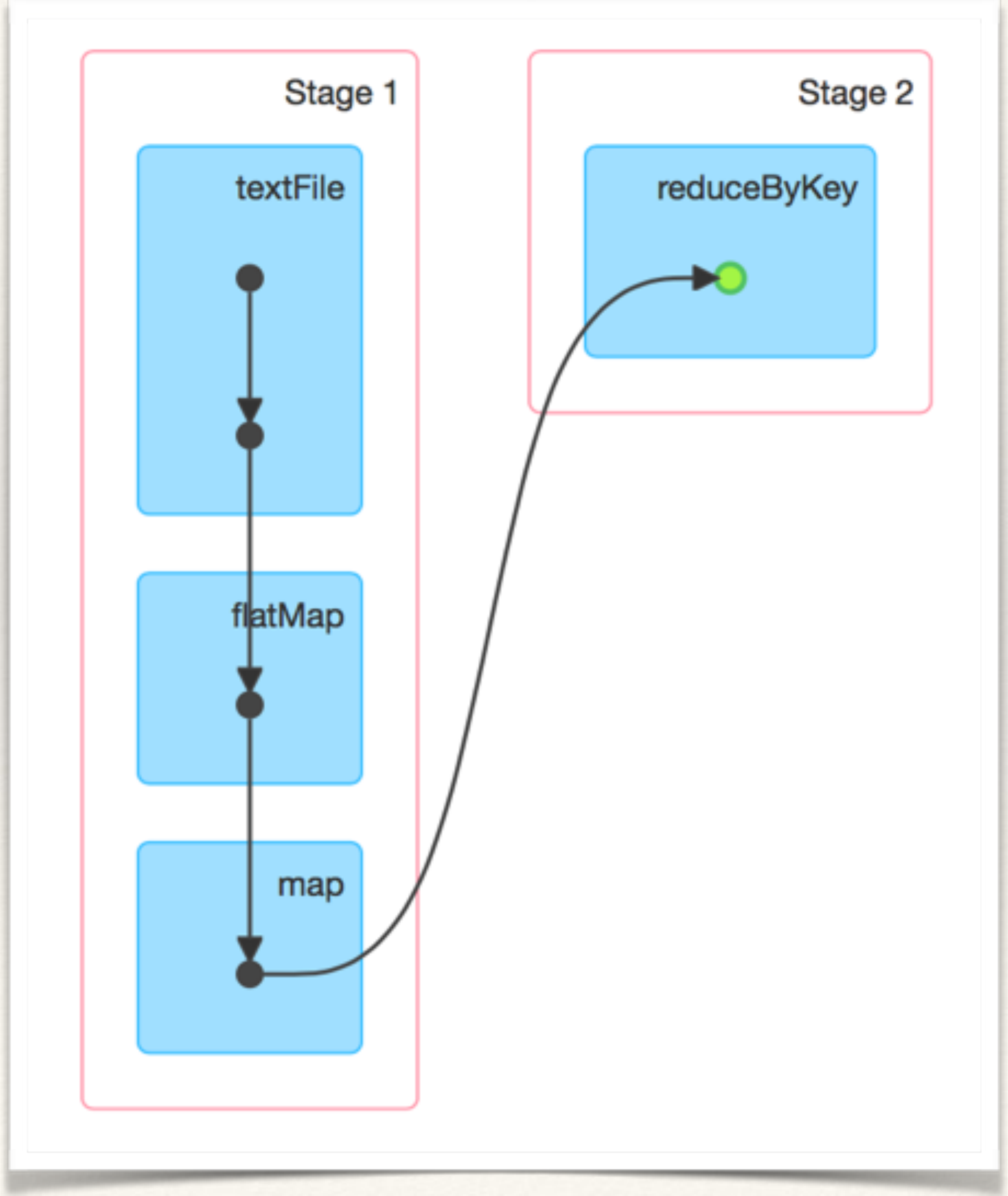
- Act to transform data in an RDD
- Returns RDD (of new schema)
- Are chained together to for a Directed Acyclic Graph (DAG) of transformations
- Underlying transformation does **NOT** take place when RDD is defined
    - Type inference does return modified RDD

Actions

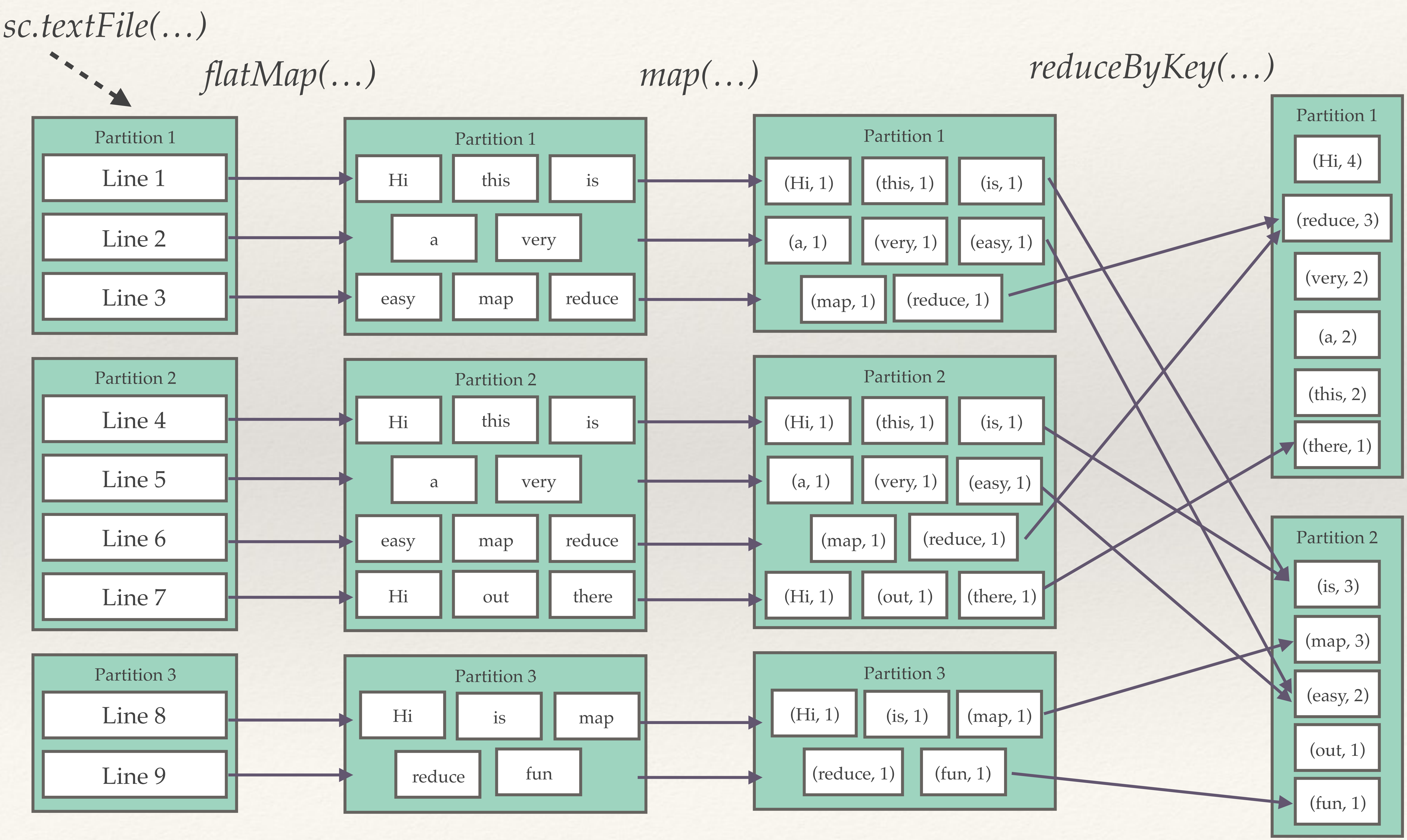
- Actions trigger execution of entire DAG
- Spark engine can optimize execution of transforms in DAG into stages, tasks across nodes
- Multiple actions will re-execution shared DAG segments
  - Can cache results of common segments in memory
  - Or persist them to disk, if very large

Map-Reduce Example

http://localhost:4040



Map-Reduce Example



Enough talk, show me the code!

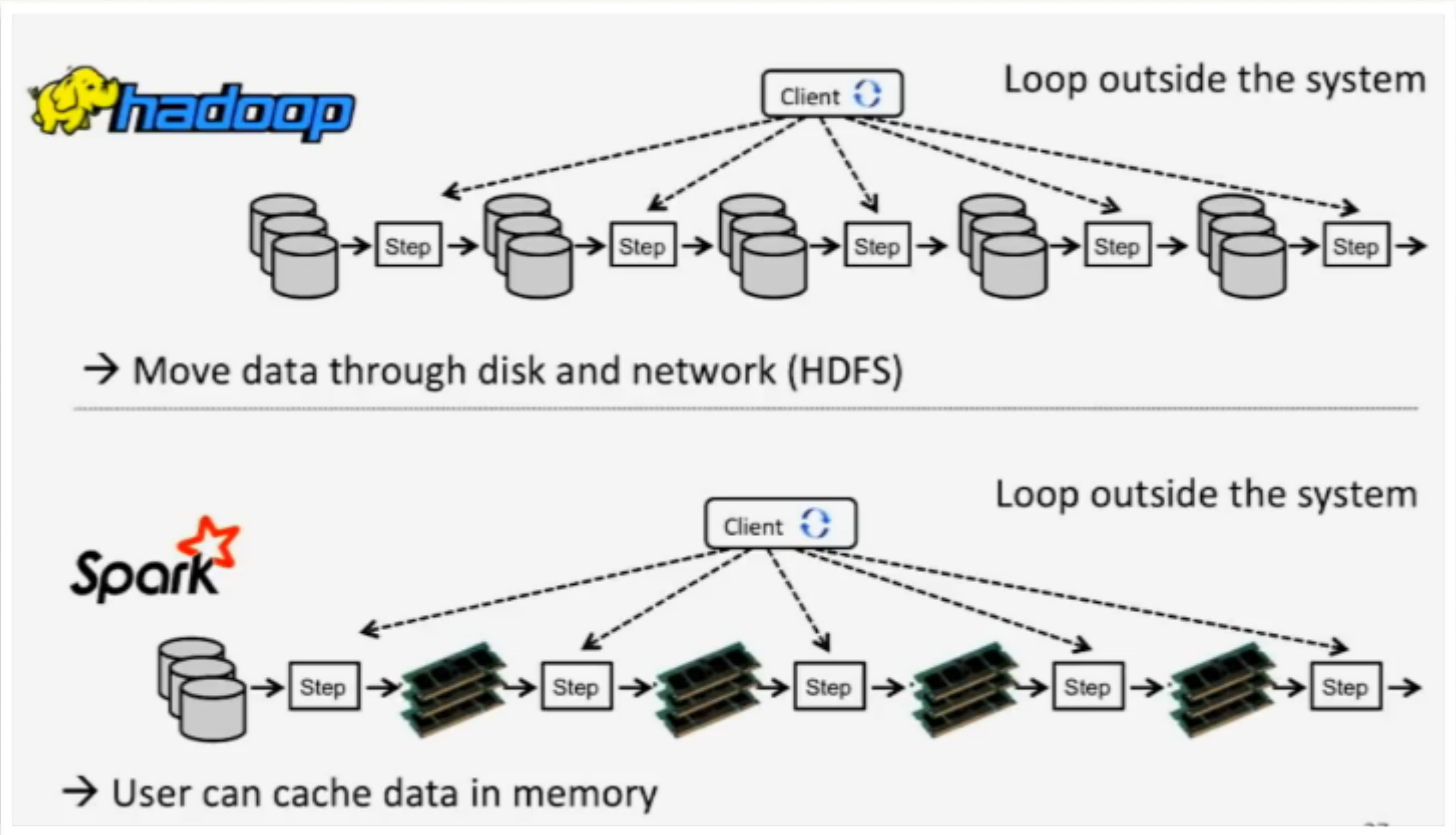
Exercise 2:

Word Frequency

Spark vs. Hadoop Map-Reduce

- Hadoop MR only has 2 functional verbs:
  - Map
  - Reduce
- Spark offers full featured functional programming environment
- Hadoop requires network synch. and saving to FS between each stage
- By analyzing DAG, Spark engine can optimize number of operations done consecutively in memory without network synchronization
- Spark is up to 100x faster than Hadoop MR

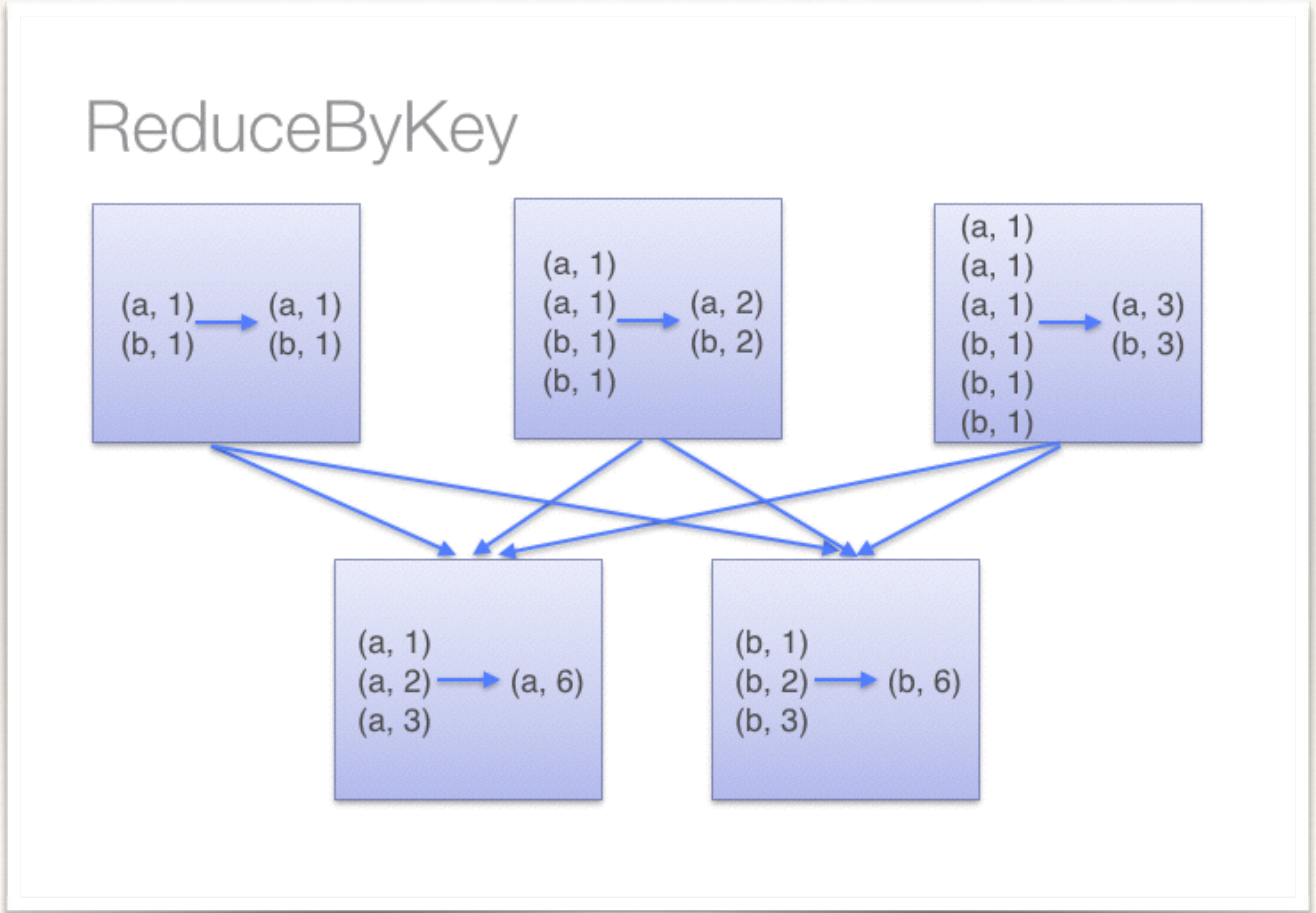
Spark vs. Hadoop Map-Reduce



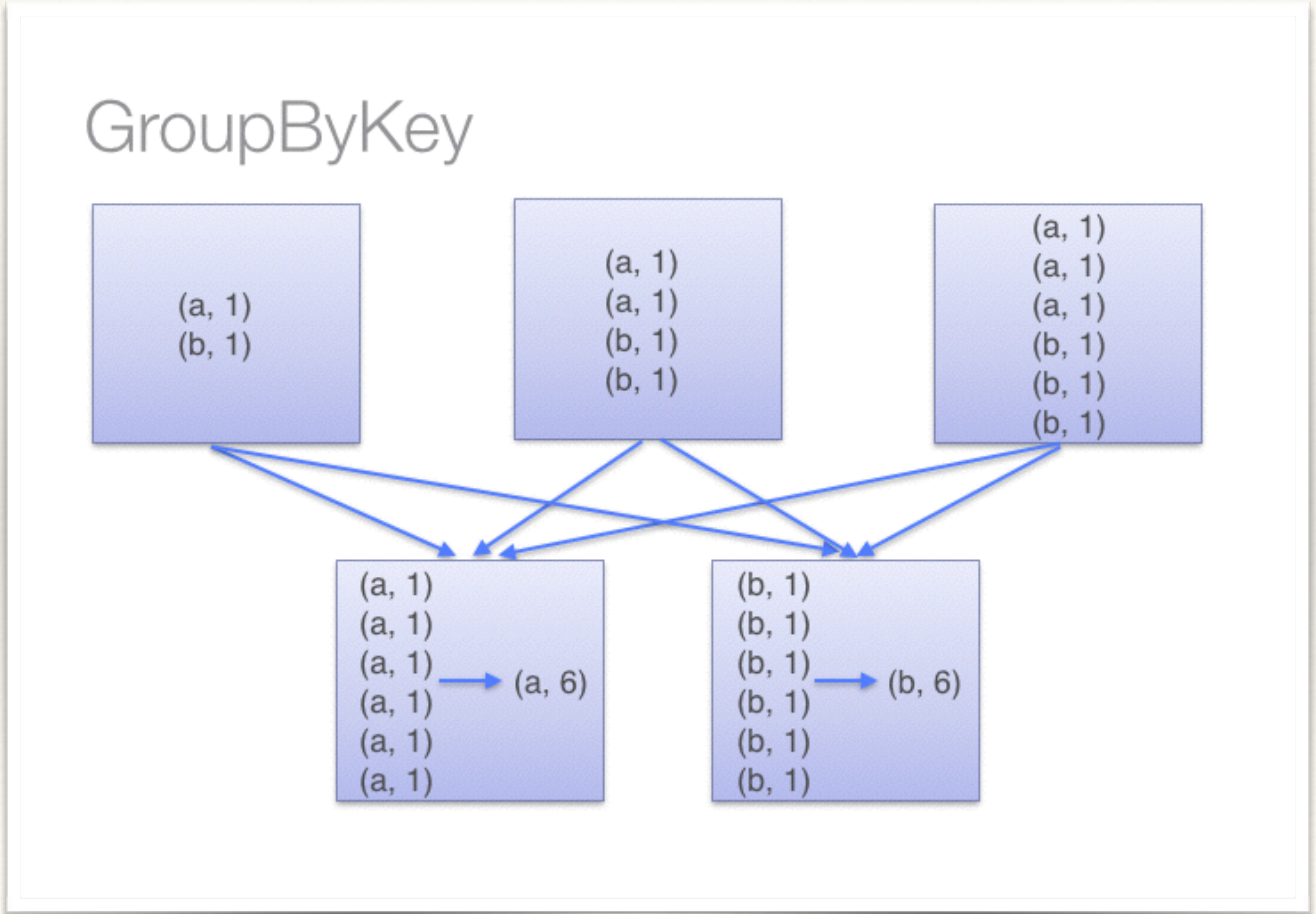
Know your transformations!

- Different transforms have different characteristics of network synchronization (shuffle & sort)
- Using the wrong one can slow down your code
- Example: avoid groupByKey

Avoid groupByKey



Avoid groupByKey



Partitioning matters!

- Effective partitioning of your data can also maximize the parallelization you get.
- Often the default partitioning is fine, but sometimes you want to provide your own
- Sometimes you want to do a shuffle before a compute-intensive operation
  - Imposes network cost
  - But can be faster if data unequally distributed before op
  - e.g. Sentence parsing:
    - unequal distribution of sentences/document
    - re-partitioning before parsing can balance load across workers

Enough talk, show me the code!

Exercise 3:

Submitting a Spark Application

Deployment Options

- local: development, testing
- standalone cluster: testing, occasional use
  - AWS EC2 utility
- Hadoop: production clusters, multi-use, multi-user data infrastructure
  - On your own hardware
  - Rented from AWS (EMR)
- Mesos: ???

Enough talk, show me the code!

Exercise 4:

DataShop import

Spark DataFrames

- Similar to DataFrames in R, Python
  - Uses those constructs in SparkR, PySpark
  - Has column schema (names, types, etc.)
- R DataFrames are column-oriented (list of columns)
- Spark DataFrames are row-oriented
  - Actually RDD[Row]
  - Rows have same schema as DataFrame
  - Want to partition and distribute Rows over cluster

Spark DataFrames

- Can run SQL queries on DataFrames
- Backed by different data sources:
  - lines in CSV
  - JSON entities
  - Tables in SQL / NoSQL database
- Engine executes reads/scans/queries as needed upon task execution (actions)

Enough talk, show me the code!

Exercise 5:

File layout

File Layout

- Each partition saved independently
- Would be physically located on different nodes in cluster
- Visible through distributed file-system utilities
- Drives initial layout of partitions on read

Parquet Format

- Developed specifically to be efficient on clusters
- Columnar storage!
  - More efficient for aggregation than row-oriented
- Involves all sorts of segmentation & compression tricks

Let's build a Model, already!

AFM

A Logistic Regression model of student performance

Additive Factors Model

$$p_{ij} = \Pr(Y_{ij} = 1 \mid \theta_i, \boldsymbol{\beta}, \boldsymbol{\gamma}) = \frac{\exp(\theta_i + \sum_{k=1}^{K} q_{jk}\beta_k + \sum_{k=1}^{K} q_{jk}\gamma_k T_{ik})}{1 + \exp(\theta_i + \sum_{k=1}^{K} q_{jk}\beta_k + \sum_{k=1}^{K} q_{jk}\gamma_k T_{ik})} \qquad (1)$$

Where

Y_{ij} = the response of student i on item j

θ_i = coefficient for proficiency of student i

β_k = coefficient for difficulty of skill k

γ_k = coefficient for the learning rate of skill k

T_{ik} = the number of practice opportunities student i has had on the skill k

q_{jk} = 1 if item j uses skill k; 0 otherwise

K = the total number of skills in the Q-matrix

Hao Cen, Ken Koedinger, Brian Junker (2006)

Enough talk, show me the code!

Exercise 6:

Train AFM

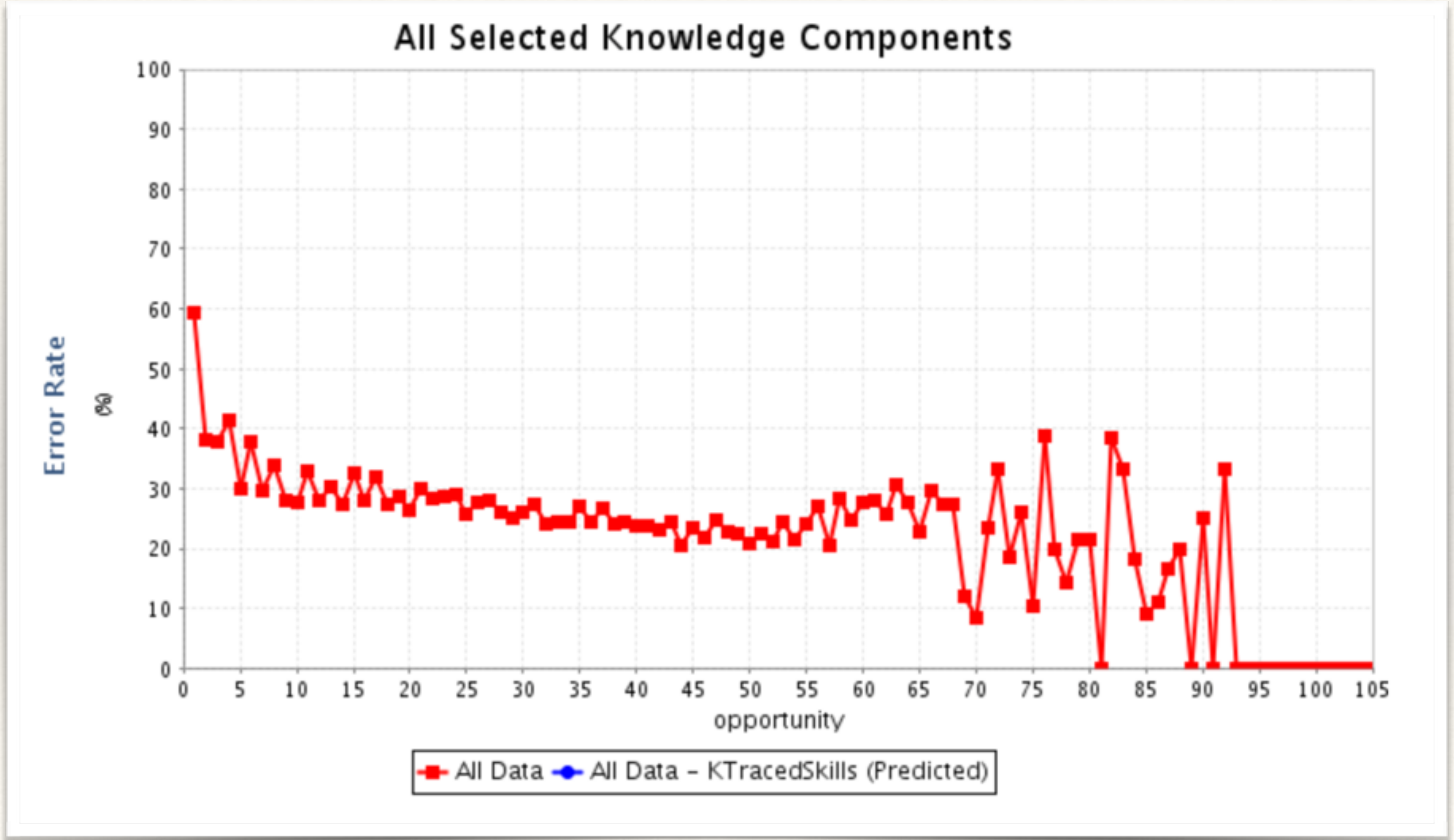
Aggregation

- Produces a single row per group
  - … per distinct tuple of group by columns
- Follows pattern:
  - groupBy(*columns*).agg(*functions*)
- Many built-in aggregation functions (as per SQL)
  - *org.apache.spark.sql.functions*
- Can build your own custom aggregators, but
  - Not so straight forward as defining a UDF
  - Requires thinking through different parts of the aggregation lifecycle:
    - when updating with new rows
    - when merging with another running aggregation instance (from another node)

Windowing

- Not everything can be done with aggregation!
- Sometimes you want a measure per row, but calculated over a group
- Windowing!
    - Not available in all SQL databases
- Follows pattern:
    - *function*.over(Window.partitionBy(*cols*).orderBy(*cols*))

Learning Curves



Enough talk, show me the code!

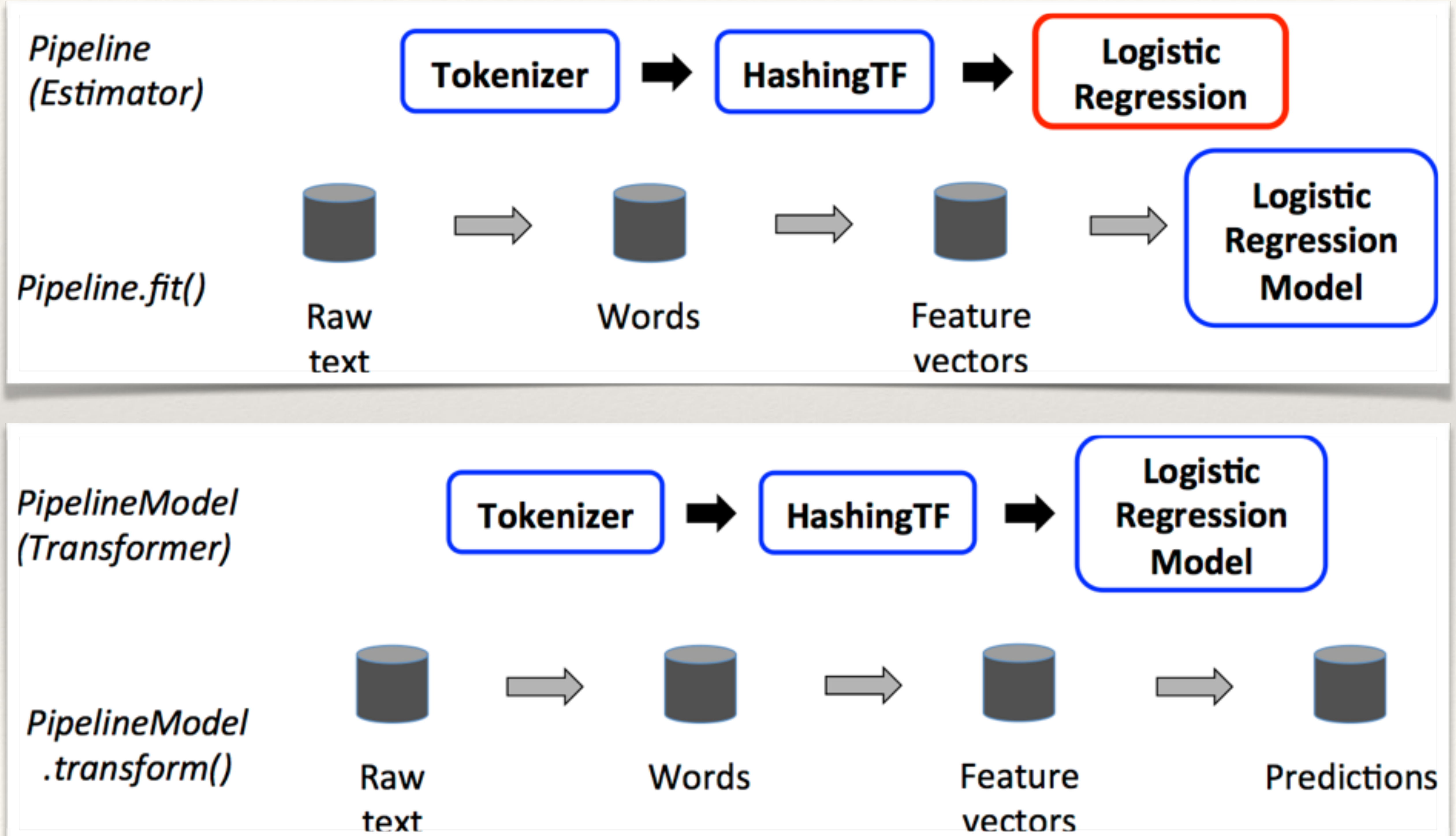
Exercise 7:

Build Learning Curve

Machine Learning API

- Transformers:
  - Change a DataFrame
  - Typically add columns, computing new values from other columns
- Models:
  - Special kind of Transformer: computes predictions
- Estimators:
  - Fits (& outputs) a model
- Pipelines:
  - Transformers, estimators can be chained together to form Pipelines
  - Trigger each “Stage” in order
  - Are special Estimators - produce fitted models
    - Pipeline models will also trigger all stages in order

Machine Learning API

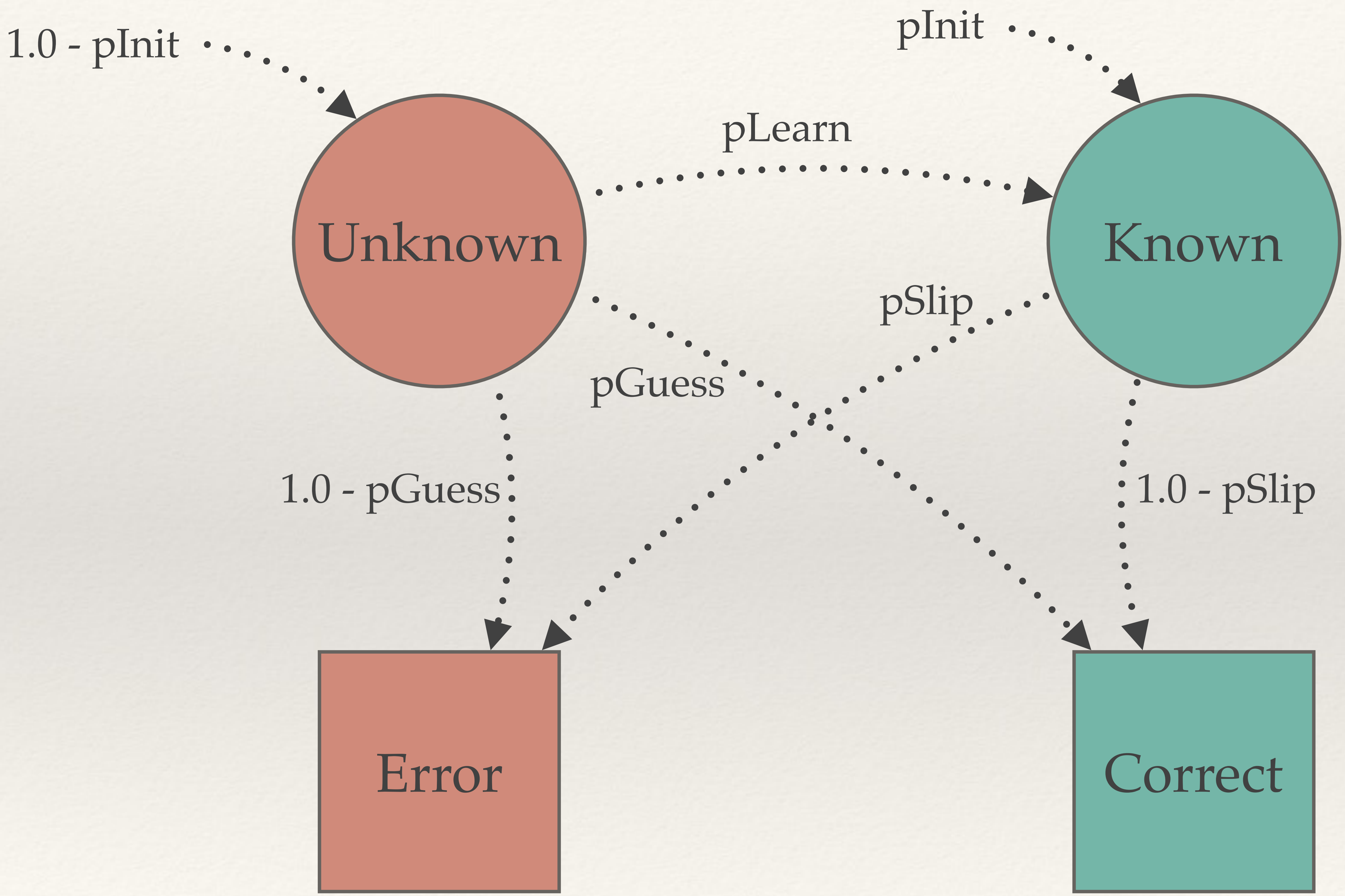


Enough talk, show me the code!

Exercise 8:

Other Features to include in Model?

Bayesian Knowledge Tracing



Corbett, Anderson (1995)

Enough talk, show me the code!

Exercise 9:

BKT GridSearch

What cannot be parallelized

- Some relations are fundamentally sequential!
- Cannot parallelize steps within these sequences
- Can parallelize across independent sequences!
  - Can train individual students in parallel
  - Fitting model-per-student
  - Must still synchronize after every iteration to average models together

Enough talk, show me the code!

Exercise 10:

Try some other models (ANN? Clustering?)