Map-Reduce Design Patterns

Big Data “Pipelines”

- Data pipeline is a chain (DAG) of map-reduce jobs

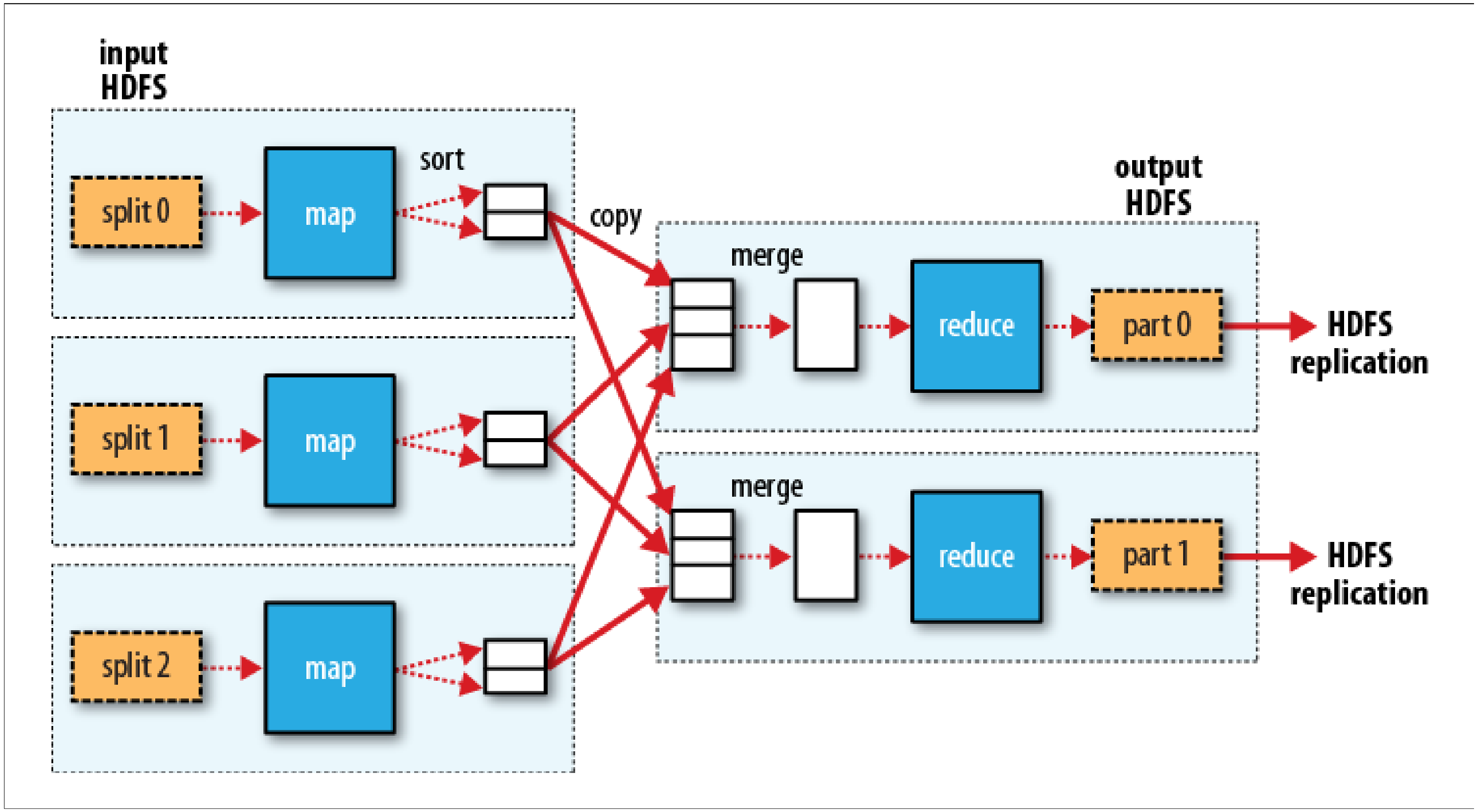
- Map-Reduce (Hadoop) helps:
  - Distribute computation across many machines
  - Maximize performance
    - Minimize I/O to disk, minimize transfers across the network
  - Combine the results of distributed computation
  - Recover from failures

Big Data "Pipelines"

- When designing a map-reduce program …
  - You must fit your problem into the map-reduce programming model
    - Map function
    - Reduce function
- Map-reduce design patterns
  - Standard patterns for common data pipeline processing

MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide



MapReduce Design Patterns

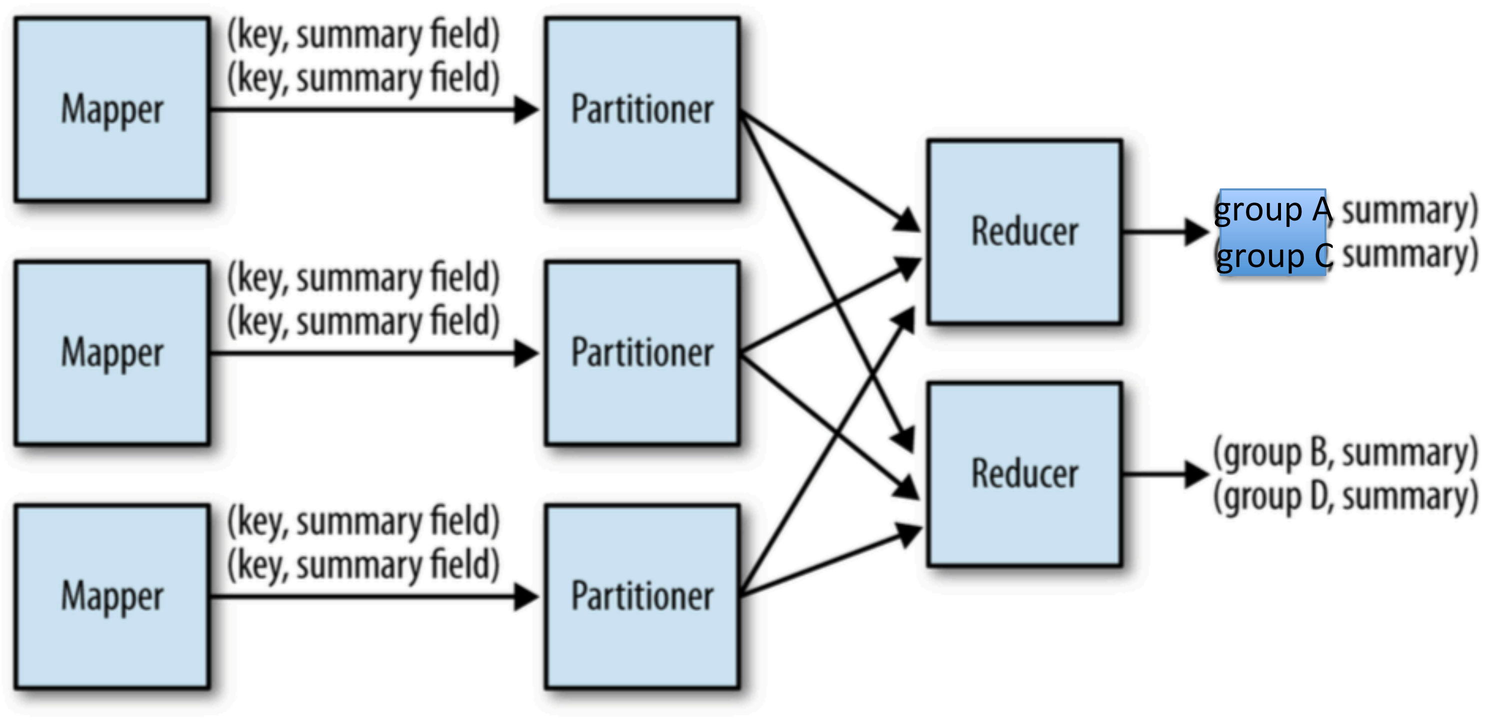
- Summarization
- Filtering
- Data Organization
  - Partitioning/binning, sorting, shuffle
- Joins
  - Merging data sets
- Meta-patterns
  - Optimizing map-reduce chains (data pipelines)

Summarization

- Counting things is a common map-reduce task
  - Word count was a simple example
  - Min, max, mean, median, variance, …
- By making the "things" being counted keys, MapReduce is doing much of the work for us
  - Hadoop sorts and groups data by key
- In WordCount, the words counted are the keys

Summarization

Figure 2.4, Map Reduce Design Patterns (edited)



Summarization

- Simple and useful pattern
- Mappers do local counts, reducers sum up
- Combiners are very useful here
- Usually collecting multiple statistics

Inverted Index

- For an inverted index that represents which documents an individual word appears in:
- What is the final output?
  - Key: word
  - Value: list of documents the word appears in
- Given our data set of document(s)
  - What should the mapper do?
  - What should the reducer do?
  - Can we use a combiner?

Filtering Patterns

- For filtering, we’re not changing the data
- We interested in finding subsets of the data
  - Examine the data in detail
  - “Search”

- Some common basic filtering uses
  - grep
  - Random sample
  - Score records on some criterion, apply a threshold
  - Data cleansing

Design Pattern

- Structured to hierarchical design pattern

- Data sources linked by some foreign key
- Data is structured and row based
  - For example, from databases
- Data is semi-structured and event based
  - Web logs

- Example: User session

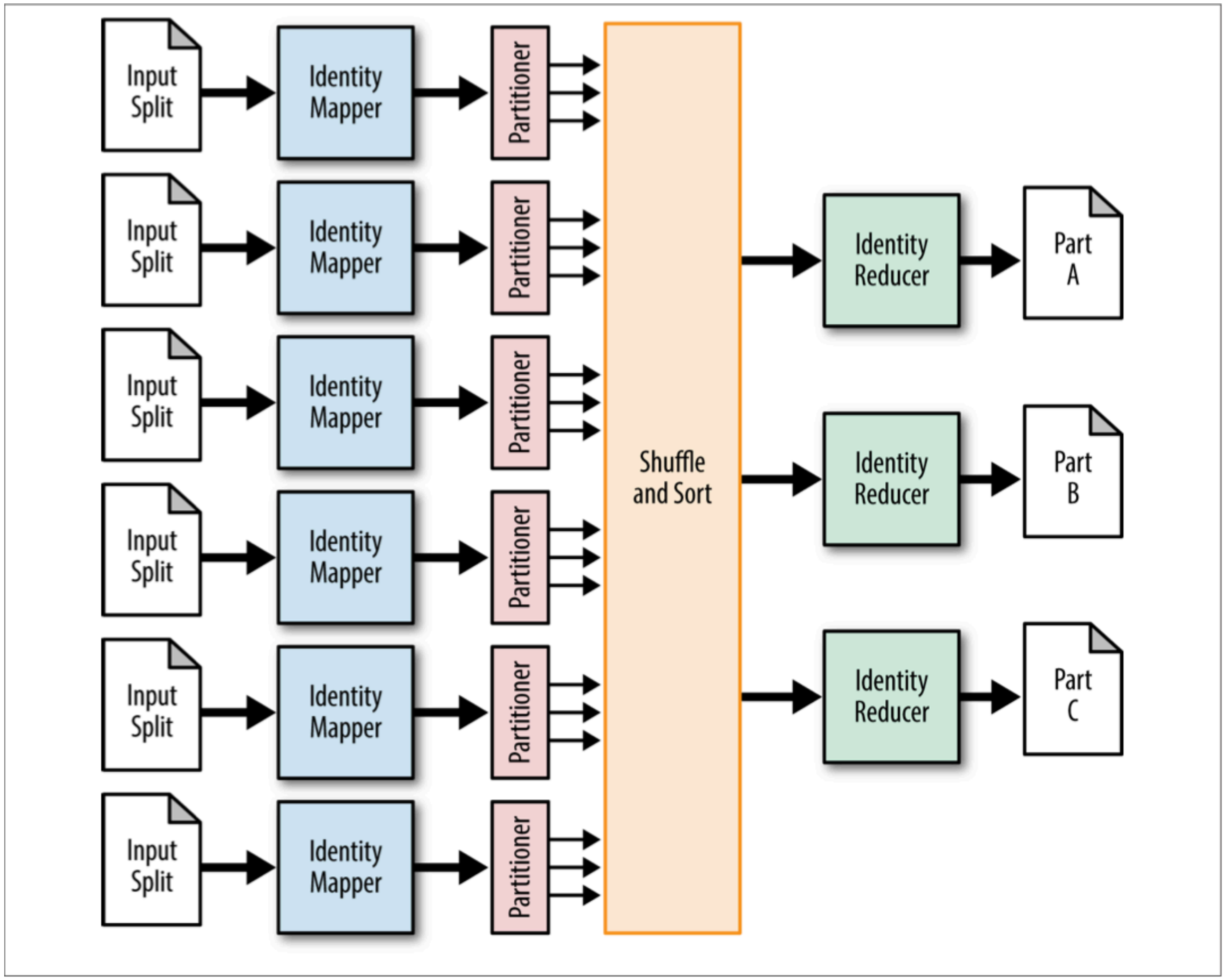
Partitioning

- Organize “similar” records into partitions

- Why?
  - Future jobs will only focus on subsets of the data
- Partitioning schemes:
  - Time: hour, day, week, month, year
  - Geography: ZIP, DMA, state, time zone, country
  - Data source: web site
  - Data type

Data Flow

Figure 4-2 from MapReduce Design Patterns



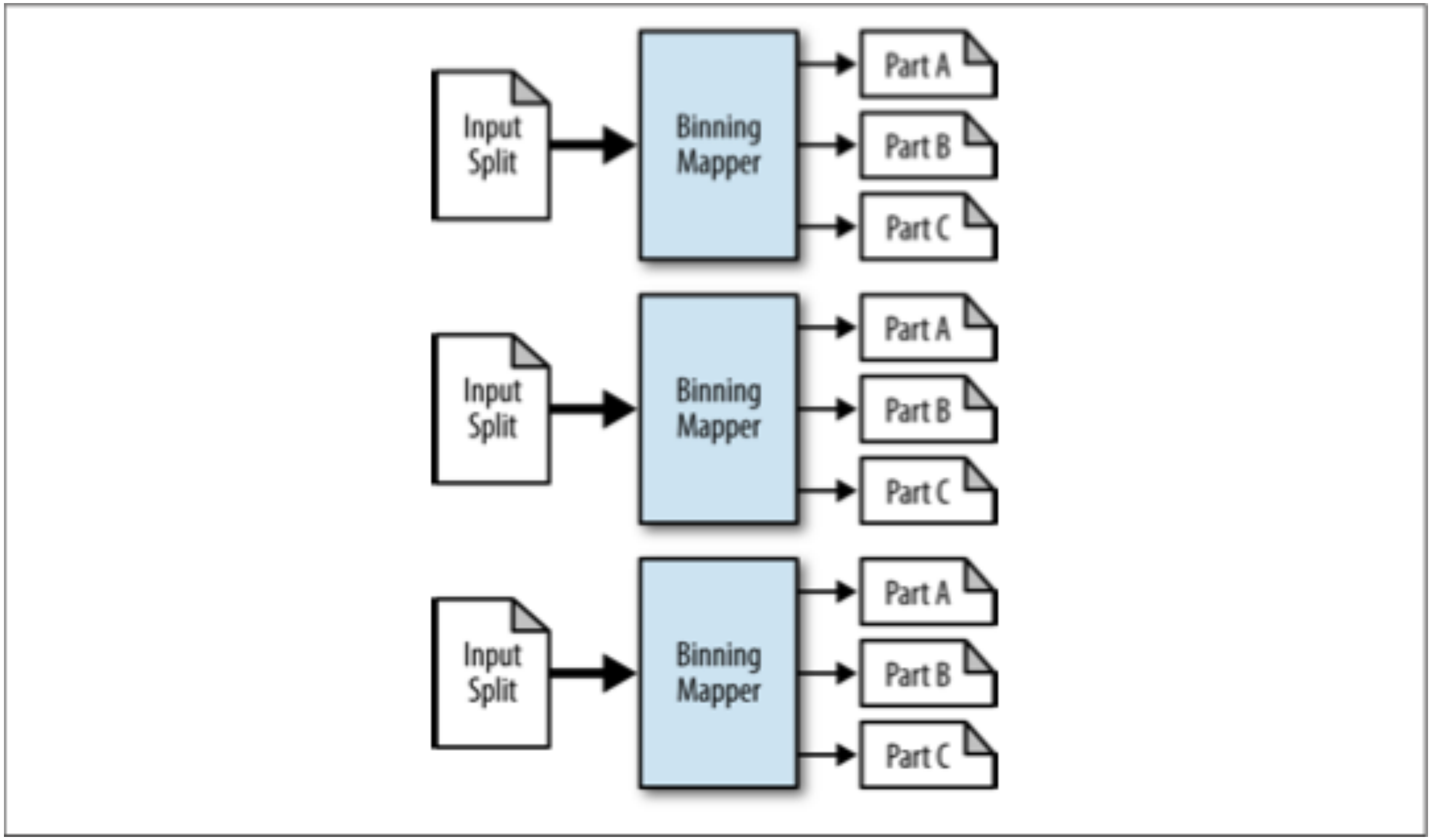
Binning

- Similar to partitioning
  - Want to organize output into categories
  - Map-only pattern (# reduce tasks set to 0)

- Mapper output written to output directories
- Uses `MultipleOutputs` class
  - Call `write()` on `MultipleOutputs`, not `Context`
  - For each category, each mapper writes a file
  - Expensive if many mappers and many categories

Binning Data Flow

Figure 4-3 from MapReduce Design Patterns



Shuffle

- Want to distribute output randomly
- Mapper generates a random key for each output
- If you want to reuse a mapper, you could add a partitioner that generates a random partition #
  - Mapper code is then unchanged
- Reducer can sort based on some other random key
  - Further shuffling the data (input order now gone)

Shuffle – Why Do This?

- Random sampling
- Randomly select subset of the data (downsample)
- Multiple random subsets for
  - Model generation and testing – cross validation
  - Train on 80%, test on 20%, for 5-fold cross validation

- Anonymizing data (example from the textbook)
  - Replace PII with a random key

Join Patterns

- It is almost always the case that our “big data” is coming from multiple sources
  - Web logs (of different types)
  - Databases (RDBMS, column-oriented DB, NoSQL, …)
  - Key/value store (redis, …)
  - …

- And we need to combine/integrate this data for
  - Reporting, analysis, BI, ETL, …

Join Patterns

- We're familiar with "join" in RDBMS (using SQL)
- A join combines data from two or more data sets
  - Based on a field or set of fields – the *foreign key*
  - In RDBMS, the *foreign key* field matches values in the column of another table
  - Effectively a cross-reference between two or more tables

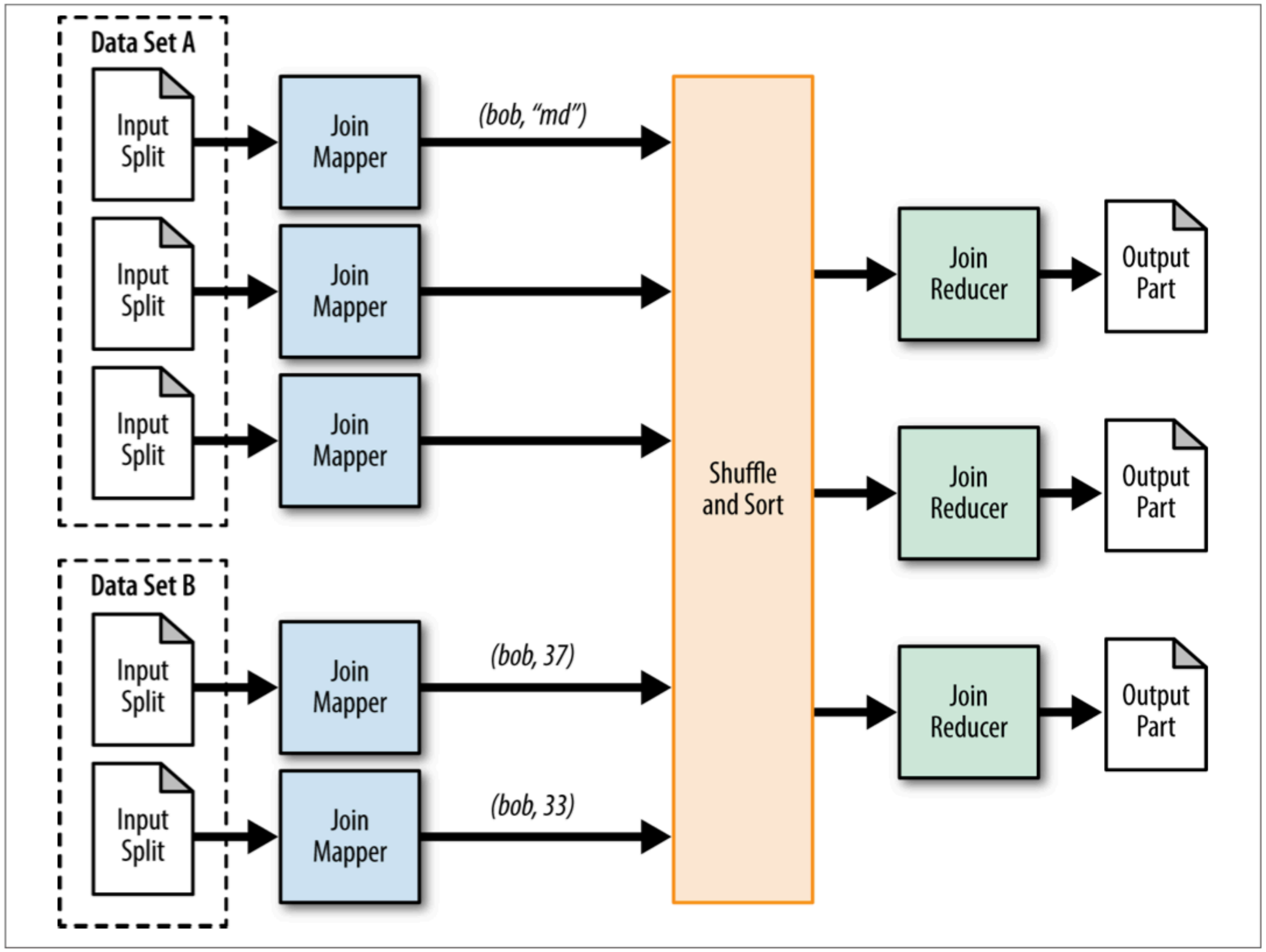
Reduce Side Join

- The simplest implementation of mapReduce join is the *reduce side join*

- Reduce side join can accomplish any of the joins we discussed
  - Inner join, full outer join, left outer join, right outer join
  - Anti-join, full Cartesian product
- Requires that all data be sent over the network to reducers
- Can join as many data sets as you need (need a common key)

Reduce Side Join - Data Flow

Figure 5-1 from MapReduce Design Patterns



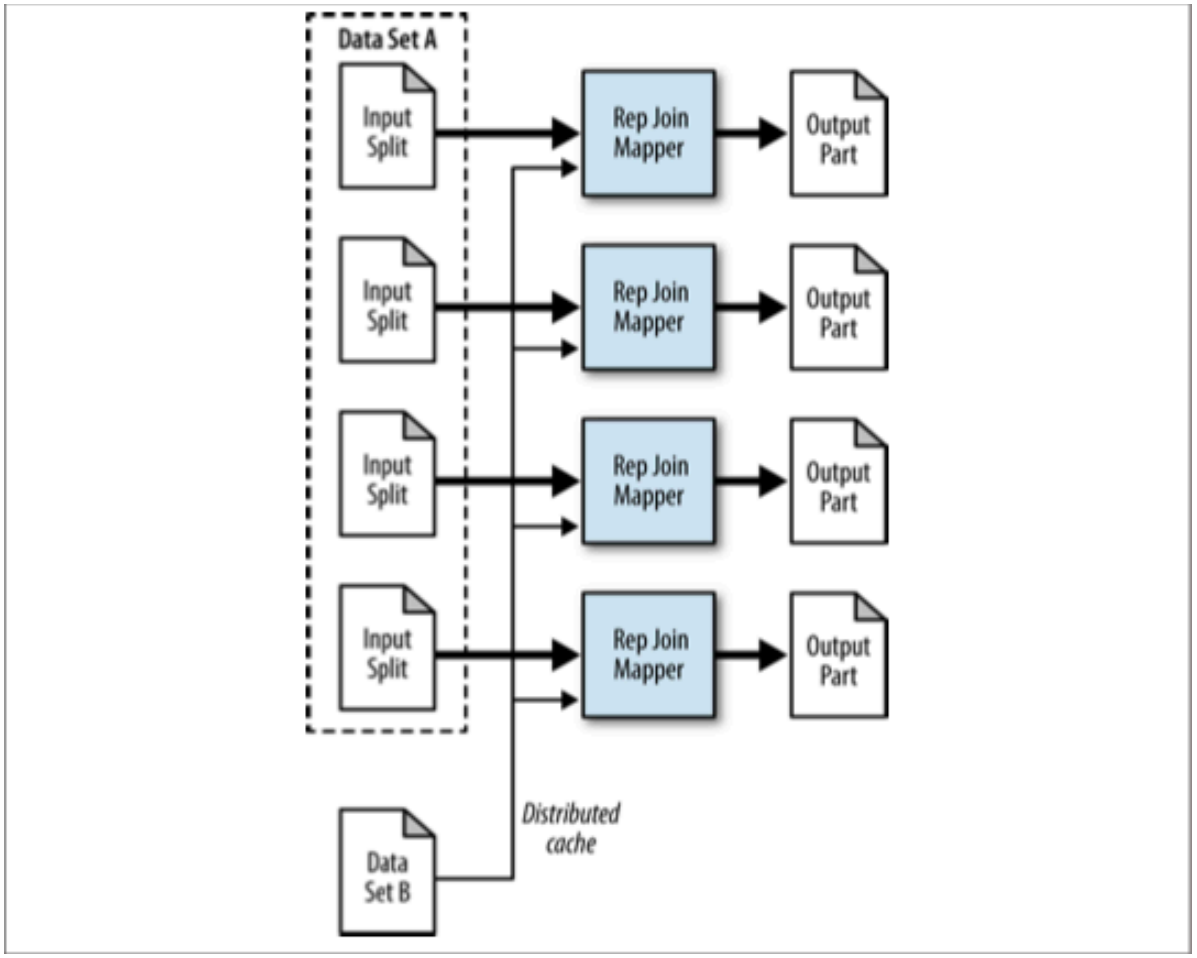
Replicated Join

- Suppose we want to join many sources, only one of which is large
  - Replicate join can be done completely in mappers
  - No need for sort, shuffle, or reduce

- Restrictions:
  - All but one of the inputs must fit in memory
  - Small files are replicated with `DistributedCache`
  - Can only accomplish an inner join, or
  - A left outer join where the large data source is “left” part

Replicated Join - Data Flow

Figure 5-2 from MapReduce Design Patterns



Join Patterns

- OK, so replicated join was interesting, but more than one of my data sources is large.
- Is there a way to do a map-side join in this case?
- Or is reduce-side join my only option?

- If we organize the input data in a specific way,
- We can do this on the map-side.

Composite Join

- Hadoop class `CompositeInputFormat`

- Restricted to inner, or full outer join
- Input data sets must have the same # of partitions
  - Each input partition must be sorted by key
  - All records for a particular key must be in the same partition

- Seems pretty restrictive, but actually comes up a lot

Resources for Hadoop

- *Hadoop: The Definitive Guide, 3rd Edition*, by Tom White
  - O'Reilly Media
  - Print ISBN: 978-1-4493-1152-0 | ISBN 10: 1-4493-1152-0
  - Ebook ISBN: 978-1-4493-1151-3 | ISBN 10: 1-4493-1151-2
- *MapReduce Design Patterns*, by Donald Miner and Adam Shook
  - O'Reilly Media
  - Print ISBN: 978-1-4493-2717-0 | ISBN 10: 1-4493-2717-6
  - Ebook ISBN: 978-1-4493-4197-8 | ISBN 10: 1-4493-4197-7

- http://hadoop.apache.org/
- Several vendors provide Hadoop distributions
- Amazon Web Services – ElasticMapReduce