CS 378 – Big Data Programming

Lecture 13

Join Patterns

Review

- Assignment 6 – Reduce-side join
  - User session and impression data

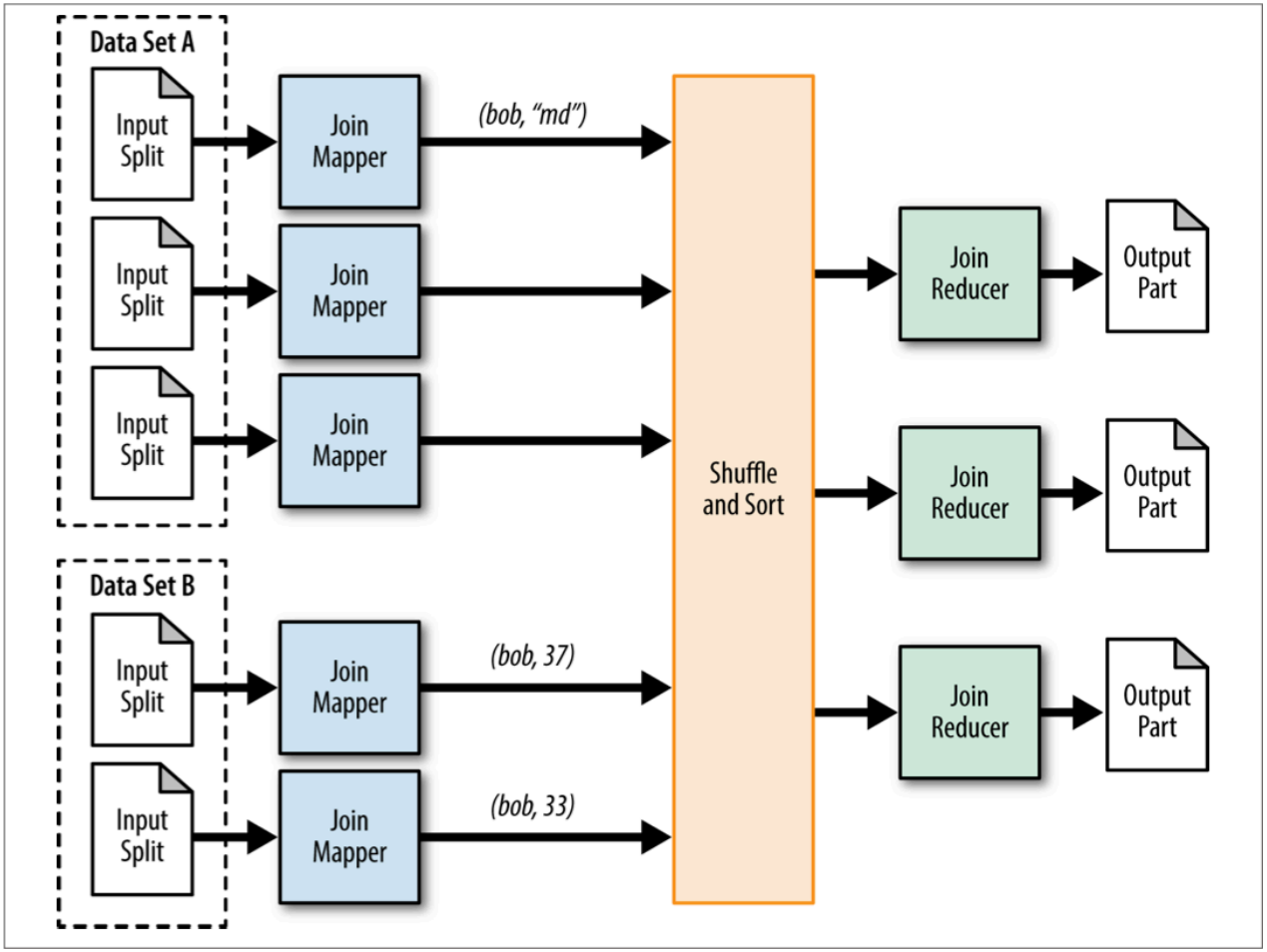
- Questions/issues?

Join Patterns

- Suppose we only wanted sessions with submits
  - In practice, a small % of sessions have submits
- In our current implementation, we can't identify these sessions until we "reduce" them
- How could we avoid transferring all the impressions for no-submit sessions from mappers to reducers?
  - Mappers would need to know which log entries to ignore

Reduce Side Join - Data Flow

Figure 5-1 from MapReduce Design Patterns



Join Patterns

- Could we tell each mapper which userIds to accept?
- First we'll need to get that info to each mapper
  - Somehow we'll need to get some info to all mappers
  - A list of userIds?
- We still have an issue if that list is too large to hold in memory

DistributedCache

- The Hadoop class: `DistributedCache`
- Allows us to specify files that are distributed to the local file system of each task (mapper or reducer)
- What do we do about the file/data size?
  - Could still be too large to hold in memory

DistributedCache

- In the driver code (`run()` method)
  - Get the file name from the command line
  - Tell Hadoop about this file
  - Name(s) conveyed in the configuration object

```
Path userIdsPath = new Path(args[1]);
FileStatus[] files =
   FileSystem.getConf().listStatus(userIdsPath);
DistributedCache.addCacheFile(
   files[0].getPath().toUri(), conf);
```

DistributedCache

- In the mapper code (`setup()` method)
  - `setup()` method called once for each mapper
  - Get the file name from the configuration
  - Load info from the file(s)

```
URI[] files = DistributedCache.getCacheFiles(
    context.getConfiguration());
```

Join Patterns

- Review: Suppose we want to join many sources, only one of which is large
  - User sessions (large)
  - Map from cities to DMA (demographic marketing area)
  - …
- This is called a *replicated* join
  - All the small files will be replicated to all machines

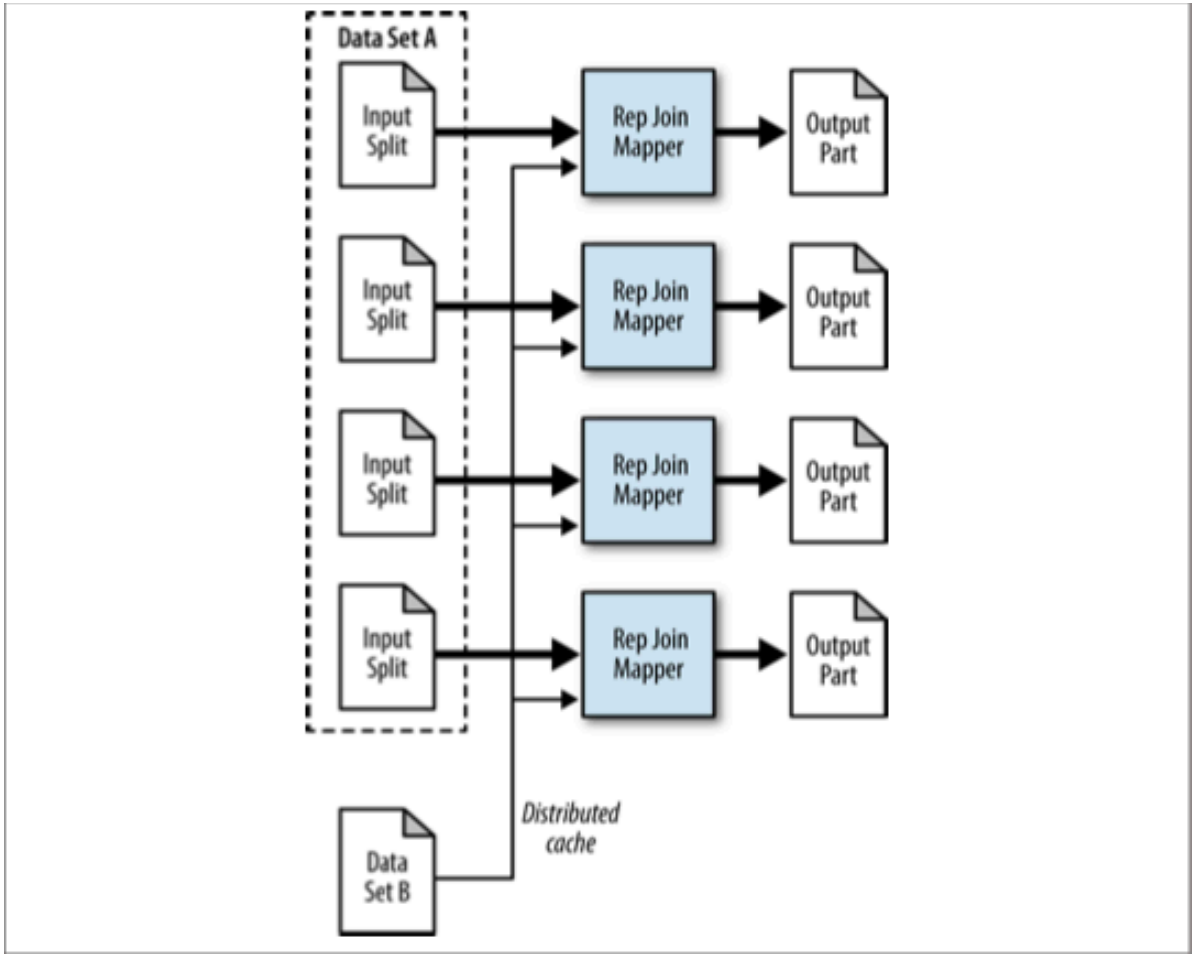
Replicated Join

- Can be done completely in mappers
  - No need for sort, shuffle, or reduce
  - Files are replicated with `DistributedCache`

- Restrictions:
  - All but one of the inputs must fit in memory
  - 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 …

Composite Join

- These conditions might exist for data from other mapReduce jobs where:

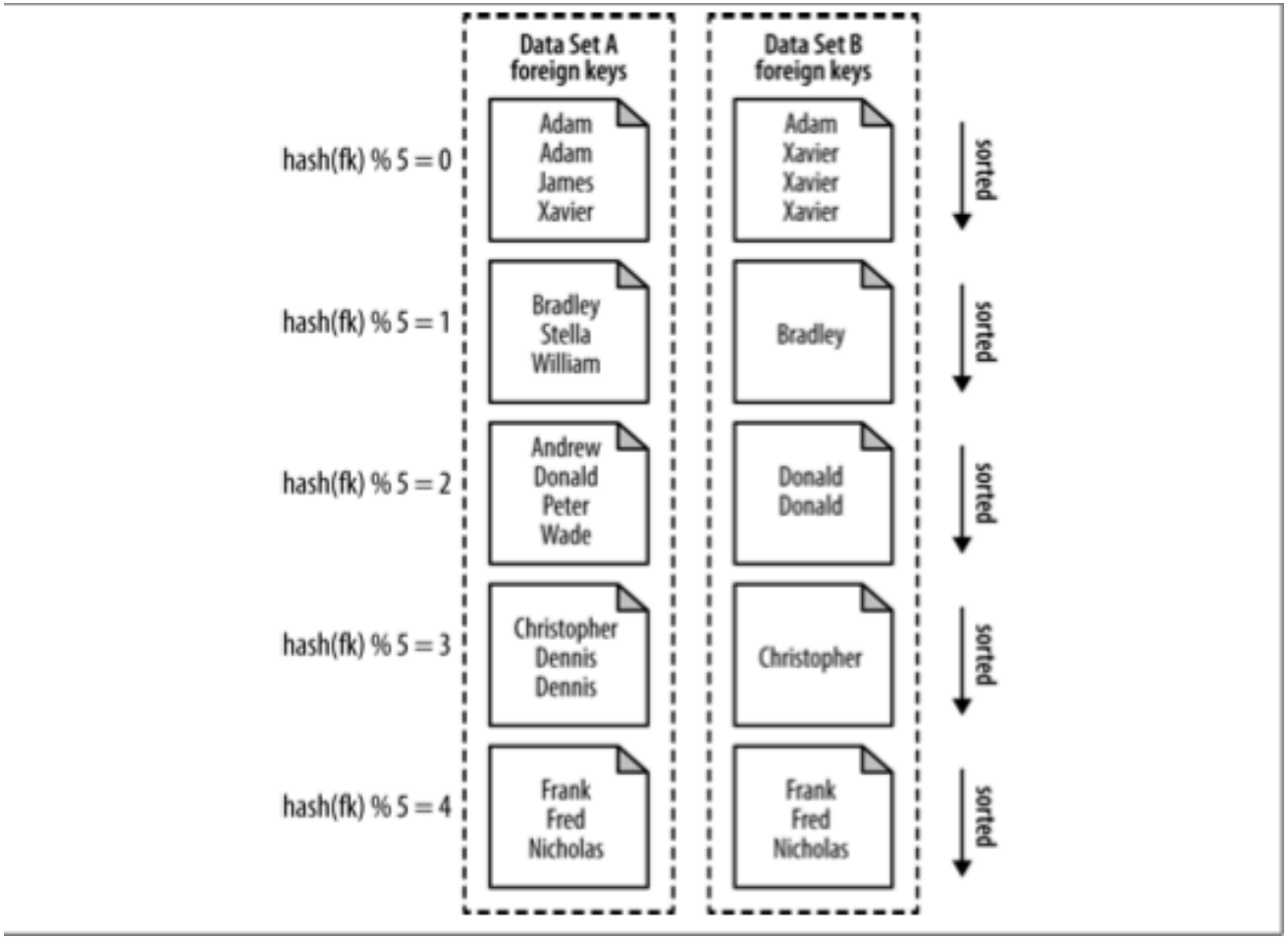
- The jobs had the same # of reducers
  - Recall that input data sets must be partitioned in same way
- The jobs had the same foreign key
- Output files aren't splittable

Composite Join

- If all those conditions are true, this join works
  - Map-side only, so it's efficient if we can use it.

- If you find that you are preparing and formatting the data only to be able to use composite join
- It's probably not worth it.
- Just use a reduce-side join.

Composite Join – Data



Composite Join – Data Flow