Part II

# MapReduce patterns

# Data Organization Patterns

# Data Organization Patterns

- Are used to reorganize/split in subsets the input data
  - Binning (not covered in these slides)
  - Shuffling
- The output of an application based on an organization pattern is usually the input of another application(s)

# Data Organization Patterns

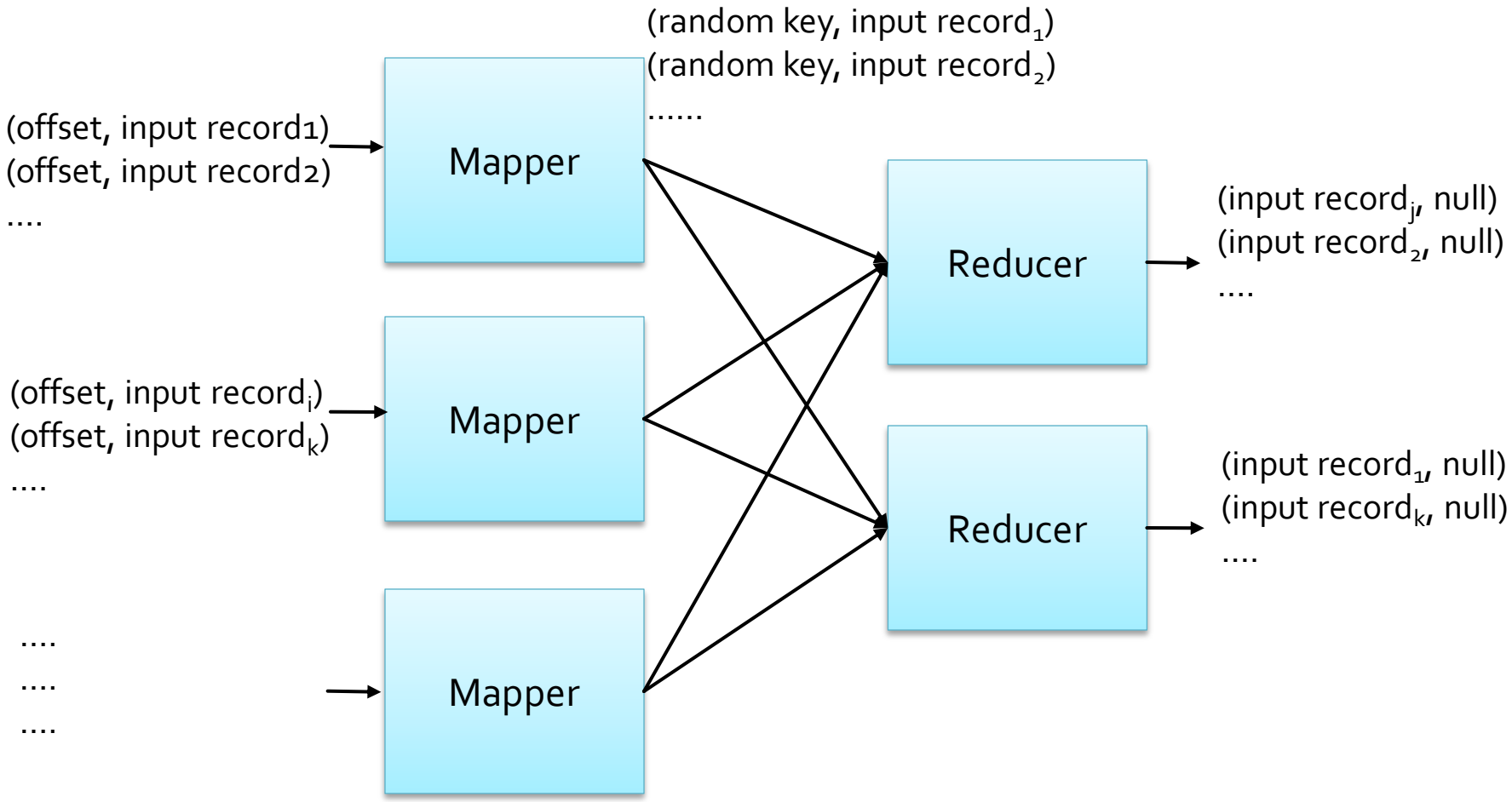
# Shuffling

- Goal
  - Randomize the order of the data (records)
- Motivation
  - Randomize the order of the data
    - For anonymization reasons
    - For selecting a subset of random data (records)

# Shuffling - structure

- Mappers
  - Emit one (key, value) for each input record
    - key is a random key (i.e., a random number)
    - value is the input record
- Reducers
  - Emit one (key, value) pair for each value in [list-of-values] of the input (key, [list-of-values]) pair

# Shuffling - structure



# Metapatterns

# Metapatterns

- Are used to organize the workflow of a complex application executing many jobs
  - Job Chaining

# Metapatterns

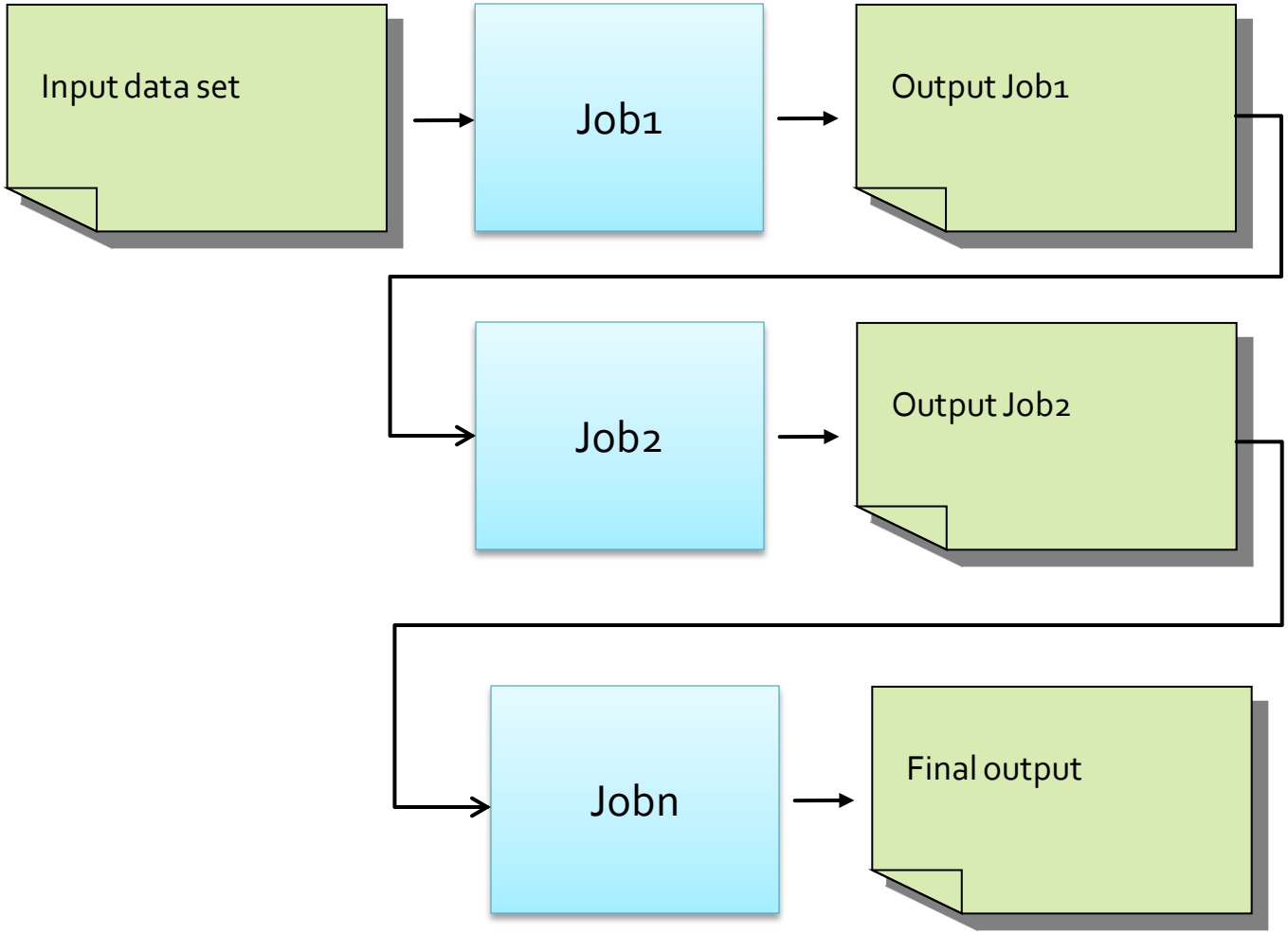
# Job Chaining

- Goal
  - Execute a sequence of jobs (synchronizing them)
- Intent
  - Manage the workflow of complex applications based on many phases (iterations)
    - Each phase is associated with a different MapReduce Job (i.e., one sub-application)
    - The output of a phase is the input of the next one
- Motivation
  - Real application are usually based  on many phases

# Job Chaining - structure

- The (single) Driver
  - Contains the workflow of the application
  - Executes the jobs in the proper order
- Mappers, reducers, and combiners
  - Each phase of the complex application is implement by a MapReduce Job
    - i.e., it is associated with a mapper, a reducer (and a combiner if it is useful)

# Job Chaining - structure



# Complex workflow

- More complex workflows, which execute jobs in parallel, can also be implemented
- However, the synchronization of the jobs become more complex

# Join Patterns

# Join Patterns

- Are use to implement the join operators of the relational algebra (i.e., the join operators of traditional relational databases)
  - Reduce side join
  - Map side join

# Join Patterns

- We will focus on the natural join
- However, the pattern is analogous for the other types of joins (theta-, semi-, outer-join)

# Join Patterns

# Reduce side natural join

- Goal
  - Join the content of two relations (i.e., relational tables)
    - Both tables are large
- Motivation
  - The join operation is useful in many applications

# Reduce side natural join - structure

- There are two mapper classes
  - One mapper class for each table
- Mappers
  - Emit one (key, value) pair for each input record
    - Key is the value of the common attribute(s)
    - Value is the concatenation of the name of the table of the current record and the content of the current record

# Reduce side natural join - structure

- Suppose you want to join the following tables
  - ***Users*** with schema userid, name, surname
  - ***Likes*** with schema userid, movieGenre
- The record
  - userid=u1, name=Paolo, surname=Garza of the Users table will generate the pair
    - (userid=u1, "Users:name=Paolo,surname=Garza")
- While the record
  - userid=u1, movieGenre=horror of the Likes table will generate the pair
  - (userid=u1, "Likes:movieGenre=horror")

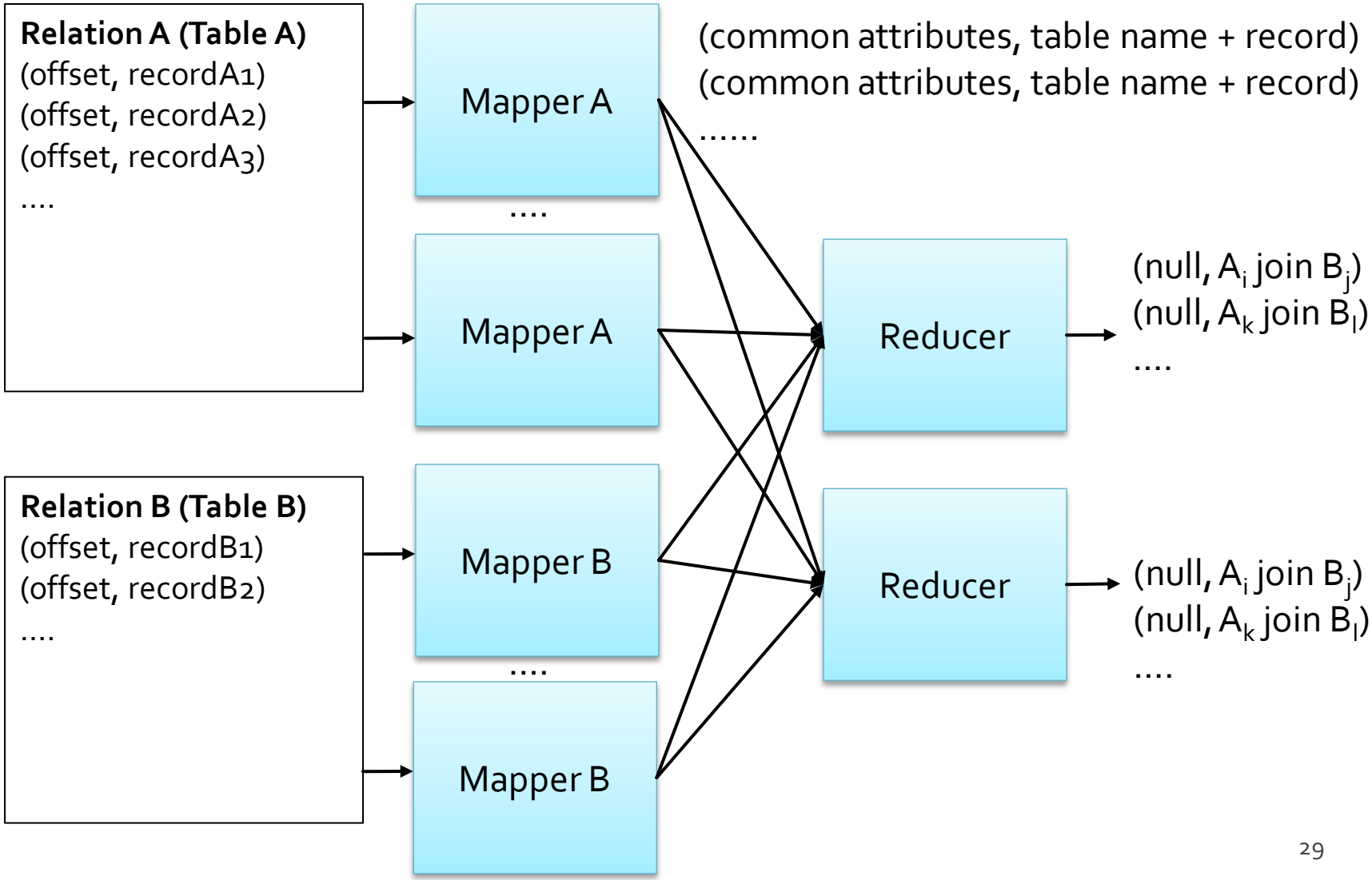
# Reduce side natural join - structure

- Reducers
  - Iterate over the values associated with each key (value of the common attributes) and compute the "local natural join" for the current key
    - Generate a copy for each pair of values such that one record is a record of the first table and the other is the record of the other table

# Reduce side natural join - structure

- For instance, the (key, [list of values]) pair
  - (userid=u1,["User:name=Paolo,surname=Garza", "Likes:movieGenre=horror", "Likes:movieGenre=adventure"] will generate the following output (key,value) pairs
    - (userid=u1,"name=Paolo,surname=Garza, genre=horror")
    - (userid=u1,"name=Paolo,surname=Garza, genre=adventure")

# Reduce side natural join - structure



# Join Patterns

# Map side natural join

- Goal
  - Join the content of two relations (i.e., relational tables)
    - One table is large
    - The other is small enough to be completely loaded in main memory
- Motivation
  - The join operation is useful in many applications and frequently one of the two tables is small

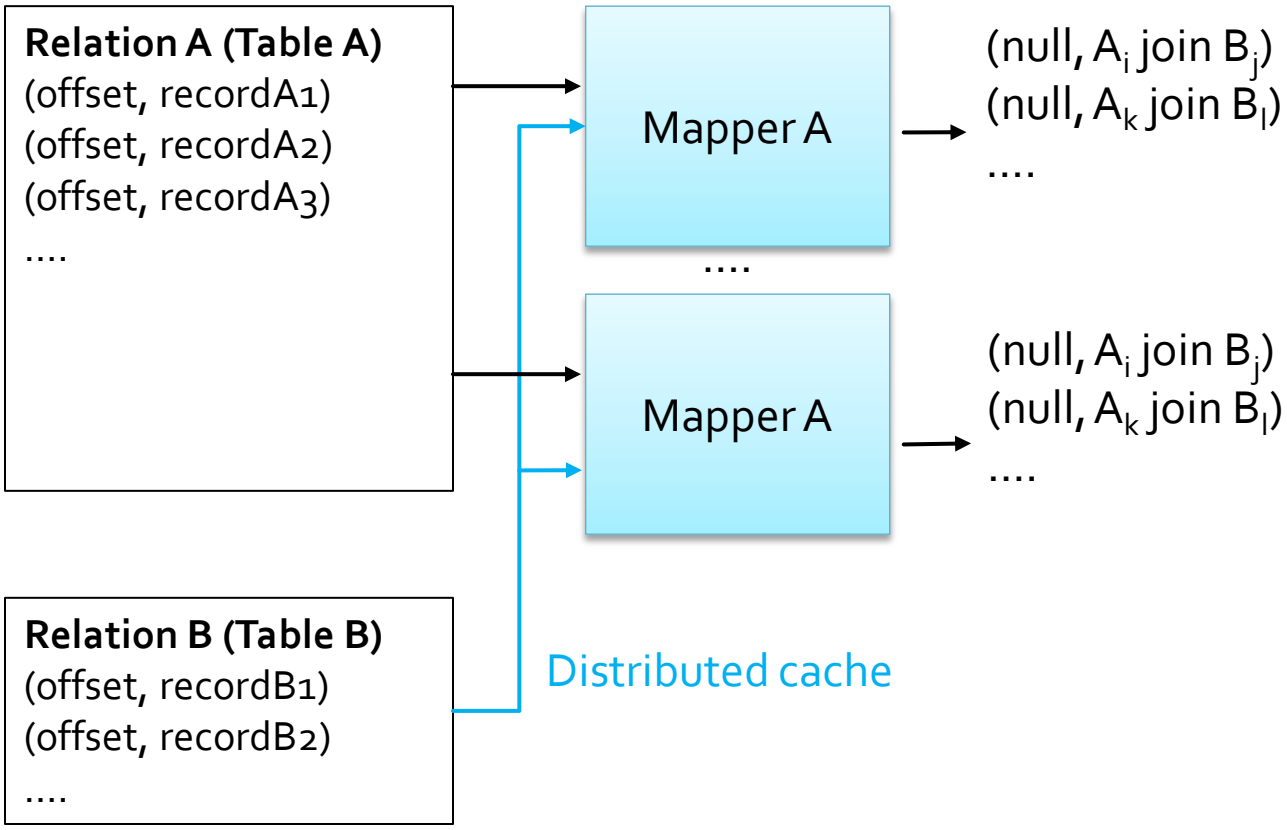
# Map side natural join - structure

- Map-only job
- Mapper class
  - Processes the content of the large table
    - Receives one input (key,value) pair for each record of the large table and joins it with the "small" table
- The distributed cache approach is used to "provide" a copy of the small table to all mappers

# Map side natural join - structure

- Each mapper
  - Performs the "local natural join" between the current record (of the large table) it is processing and the records of the small table (that is in the distributed cache)
    - The content of the small table (file) is loaded in the main memory of each mapper during the execution of its setup method

# Map side natural join - structure



# Join Patterns

# Theta-join, Semi-join, Outer-join

- The SQL language is characterized by many types of joins
  - Theta-join
  - Semi-join
  - Outer-join
- The same patterns used for implementing the natural join can be used also for the other SQL joins
  - The "local join" in the reducer of the reduce side natural join (in the mapper of the map side natural join) is substituted with the type of join of interest (theta-, semi-, or outer-join)