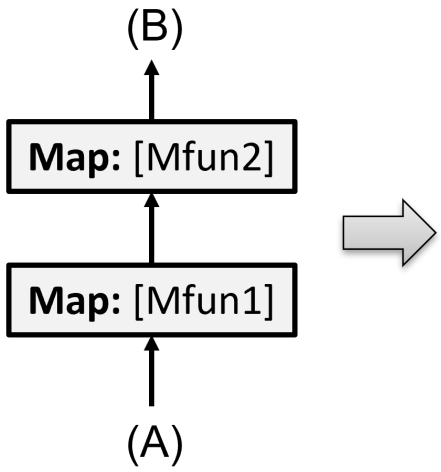
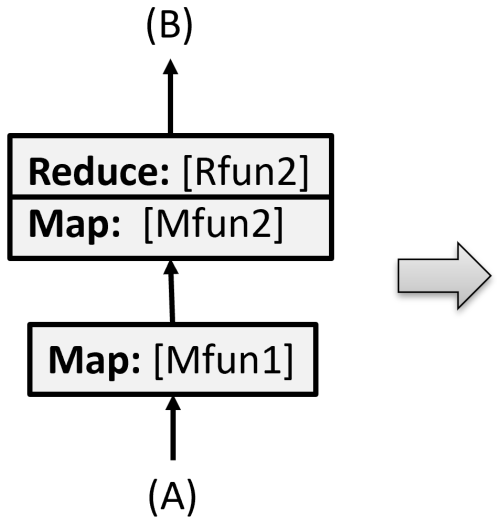
MDS Self-Study
Winter Term 2024/25
Optimization of MapReduce workflows

This unit is based on the book chapter "Chain Folding" from the O'Reilly book "MapReduce Design Patterns" by Donald Miner and Adam Shook.

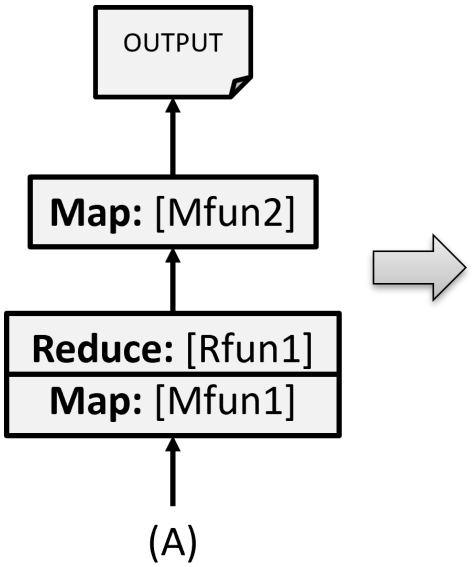
1. The book mentions a number of patterns for chain folding. One says: If multiple Map phases are adjacent, merge them into one phase.

   Complete the following sketches of how this pattern may be applied:





2. Another pattern says: If a job ends with a Map phase (combined or otherwise), push that phase into the Reducer right before it. Complete the following sketch of how this pattern may be applied:



3. What are the benefits of applying these patterns?

4. We consider the following SQL query.

```
SELECT DISTINCT p.name
FROM Eats e, Person p
WHERE e.name = p.name AND p.age <= 18
```

We assume that the following input file `Person.txt` resides in HDFS:

```
Person  {"Person.name": "Amy", "Person.age": 16, "Person.gender": "female"}
Person  {"Person.name": "Ben", "Person.age": 21, "Person.gender": "male"}
Person  {"Person.name": "Cal", "Person.age": 33, "Person.gender": "male"}
Person  {"Person.name": "Dan", "Person.age": 13, "Person.gender": "male"}
Person  {"Person.name": "Eli", "Person.age": 45, "Person.gender": "male"}
Person  {"Person.name": "Fay", "Person.age": 21, "Person.gender": "female"}
Person  {"Person.name": "Gus", "Person.age": 24, "Person.gender": "male"}
Person  {"Person.name": "Hil", "Person.age": 30, "Person.gender": "female"}
Person  {"Person.name": "Ian", "Person.age": 18, "Person.gender": "male"}
```

Further, there is the file `Eats.txt`:

```
Eats    {"Eats.name": "Amy", "Eats.pizza": "mushroom"}
Eats    {"Eats.name": "Amy", "Eats.pizza": "pepperoni"}
Eats    {"Eats.name": "Ben", "Eats.pizza": "cheese"}
Eats    {"Eats.name": "Ben", "Eats.pizza": "pepperoni"}
Eats    {"Eats.name": "Cal", "Eats.pizza": "supreme"}
Eats    {"Eats.name": "Dan", "Eats.pizza": "cheese"}
Eats    {"Eats.name": "Dan", "Eats.pizza": "mushroom"}
Eats    {"Eats.name": "Dan", "Eats.pizza": "pepperoni"}
Eats    {"Eats.name": "Dan", "Eats.pizza": "sausage"}
Eats    {"Eats.name": "Dan", "Eats.pizza": "supreme"}
Eats    {"Eats.name": "Eli", "Eats.pizza": "cheese"}
Eats    {"Eats.name": "Eli", "Eats.pizza": "supreme"}
Eats    {"Eats.name": "Fay", "Eats.pizza": "mushroom"}
Eats    {"Eats.name": "Gus", "Eats.pizza": "cheese"}
Eats    {"Eats.name": "Gus", "Eats.pizza": "mushroom"}
Eats    {"Eats.name": "Gus", "Eats.pizza": "supreme"}
Eats    {"Eats.name": "Hil", "Eats.pizza": "cheese"}
Eats    {"Eats.name": "Hil", "Eats.pizza": "supreme"}
Eats    {"Eats.name": "Ian", "Eats.pizza": "pepperoni"}
Eats    {"Eats.name": "Ian", "Eats.pizza": "supreme"}
```

Perform the following steps:

1. Canonical translation into relational algebra, just like in Milestone 1.
2. Logical optimization, just like in Milestone 2: Selection pushing and joins.
3. Translation into a physical query plan with MapReduce-based operators, like in Milestone 3. Use the graphical notation familiar from questions 1 and 2 above.
4. Chain folding according to the patterns discussed earlier.
5. Compare the number of tuples written into HDFS when evaluating the physical query plans before and after chain folding. Discuss *briefly.*
6. Can the physical plan be folded any further?

5. We consider the following SQL query:

```
SELECT *
FROM (SELECT gender, age, count(*) as count
      FROM Person
      GROUP BY gender, age) T
WHERE T.count > 1
```

Perform the same steps as in the exercise before.