CSC 369

Introduction to Distributed Computing

Filtering

Projection
Transformation

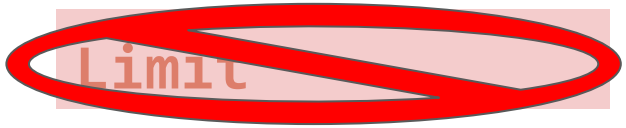
Grouping

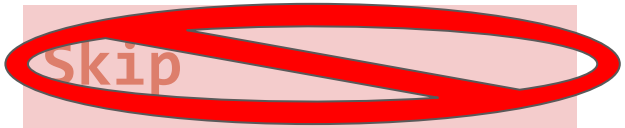
Aggregation

Join

Sort

Ungrouping
Unwinding

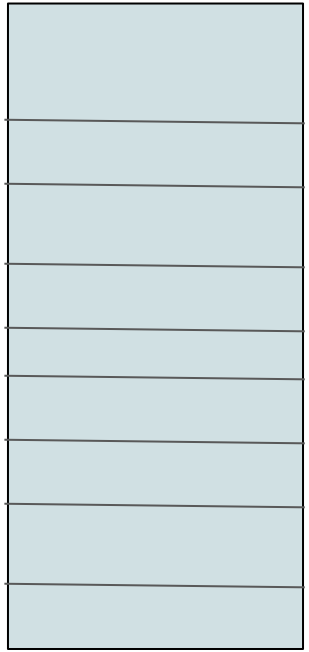




Sample

Join in MapReduce

R



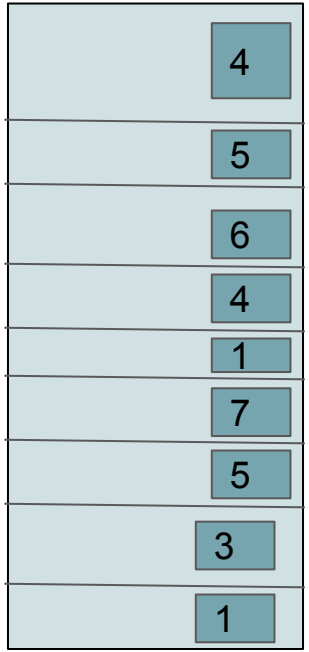
S

Join condition

```
R.x == S.y
```

Join in MapReduce

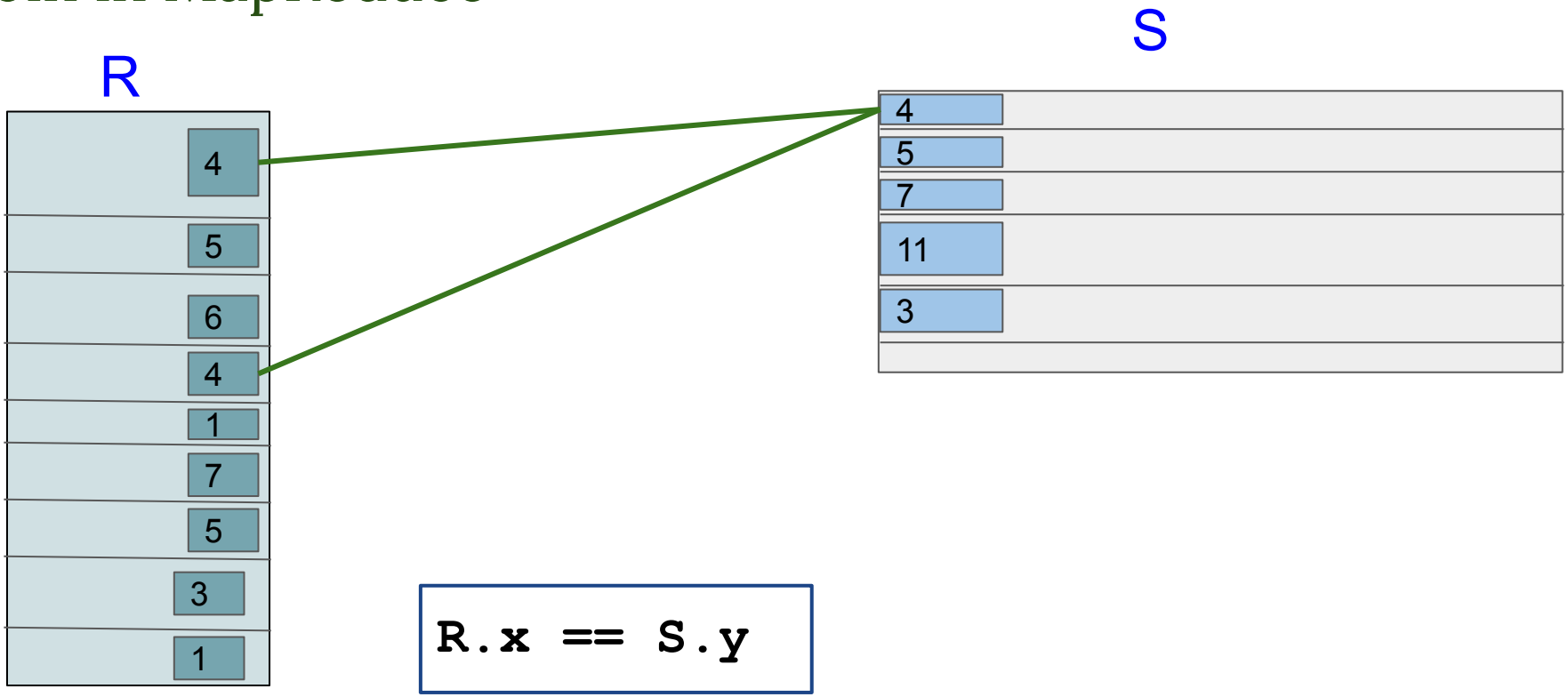
R



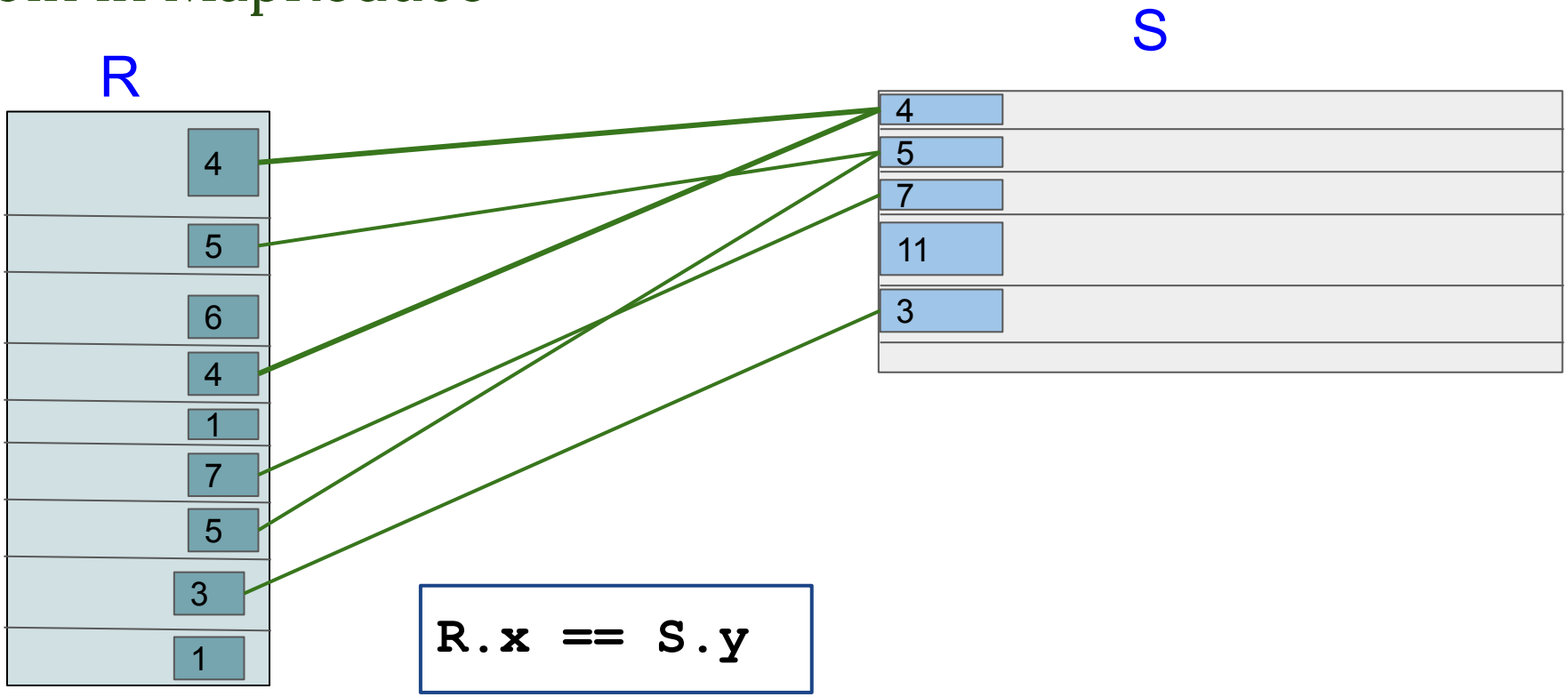
S


```
R.x == S.y
```

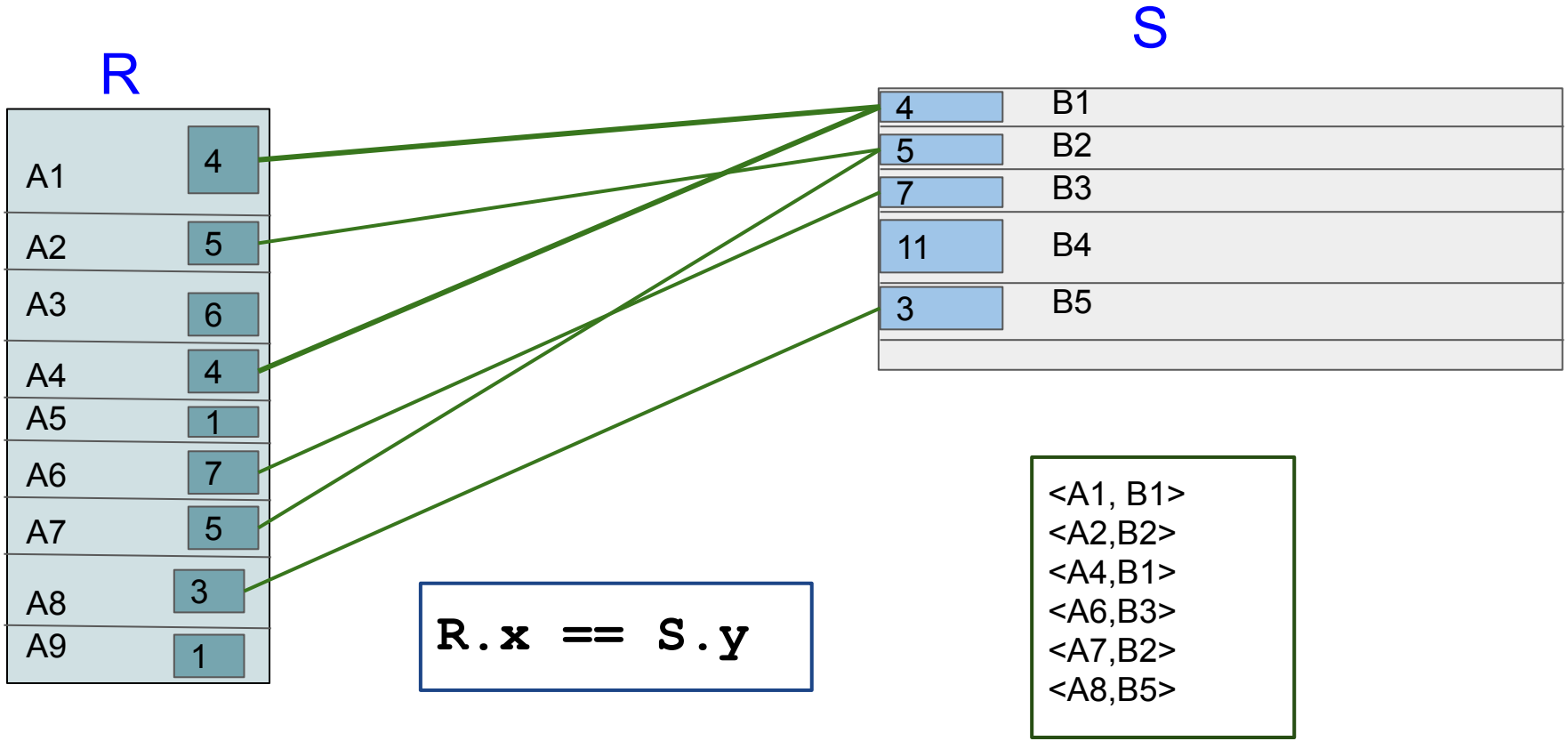
Join in MapReduce



Join in MapReduce



Join in MapReduce



Join in MapReduce

Straightforward implementation: nested loop

```
for each t in R do
    for each s in S do
        If t.x == s.y then emit(<t,s>)
```

Join in MapReduce

Straightforward implementation: nested loop

```
for each t in R do
    for each s in S do
        If t.x == s.y then emit(<t,s>)
```

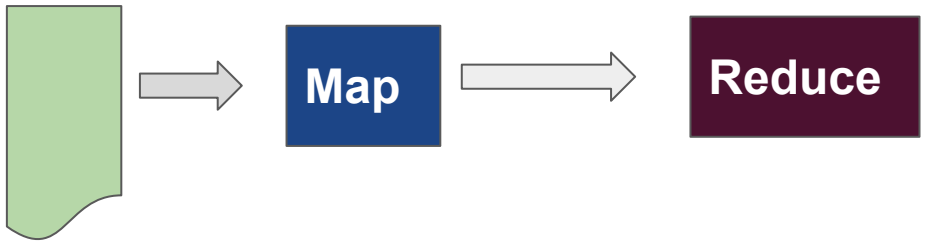
Need to turn this into distributed computation

Join in MapReduce

Technical Issue To Solve: **work with multiple files**

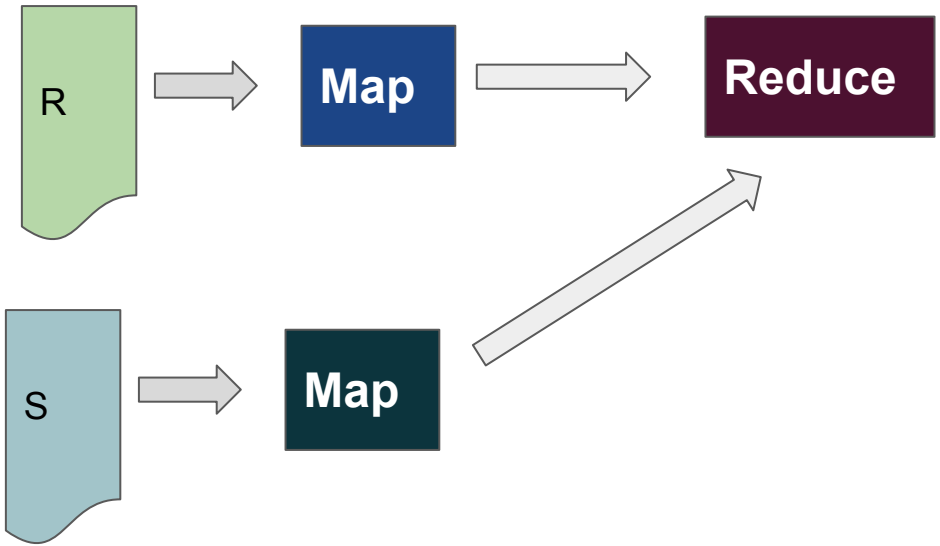
Join in MapReduce

Technical Issue To Solve: **work with multiple files**



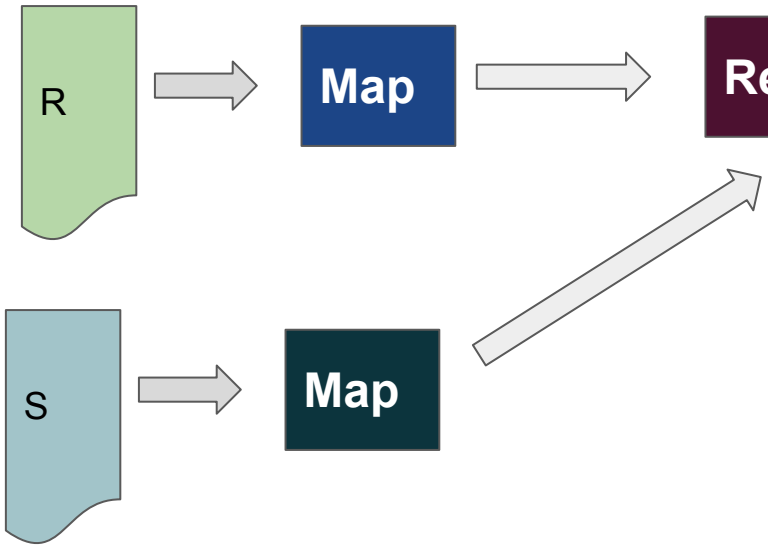
Join in MapReduce

Technical Issue To Solve: **work with multiple files**



Join in MapReduce

Technical Issue To Solve: **work with multiple files**



Multiple
Join Approaches

MultipleInputs: Reduce-side Join

Distributed Cache: Map-side Join

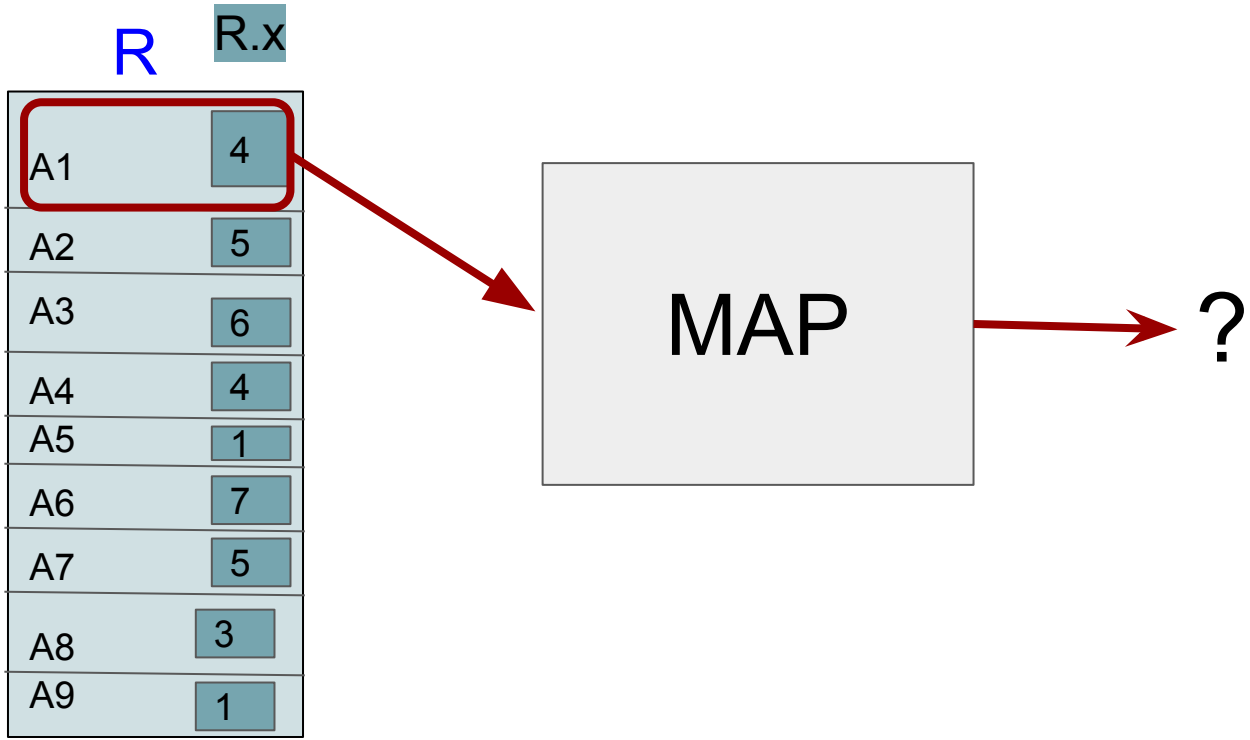
Reduce-side Join

Idea: process two sources independently, “in parallel”

Merge records in reduce

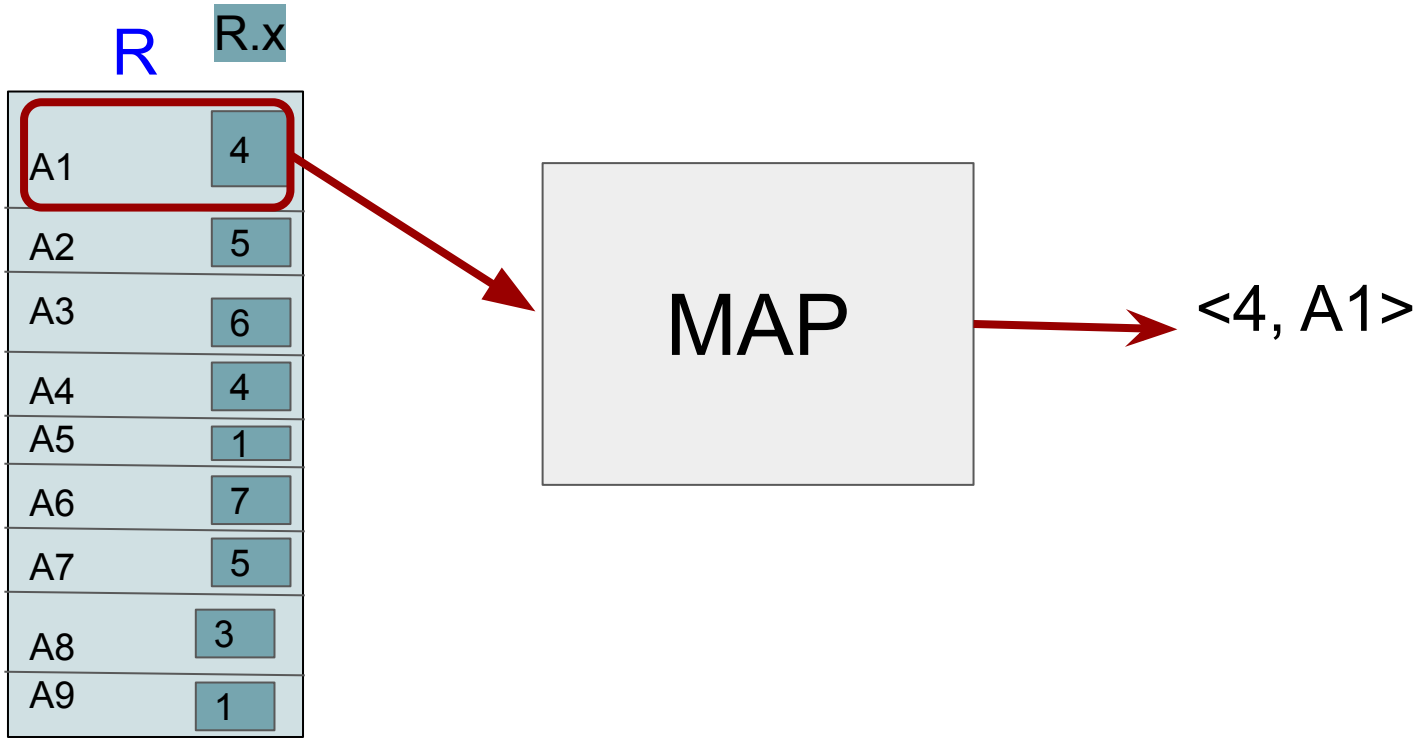
Reduce-side Join

```
R.x == S.y
```



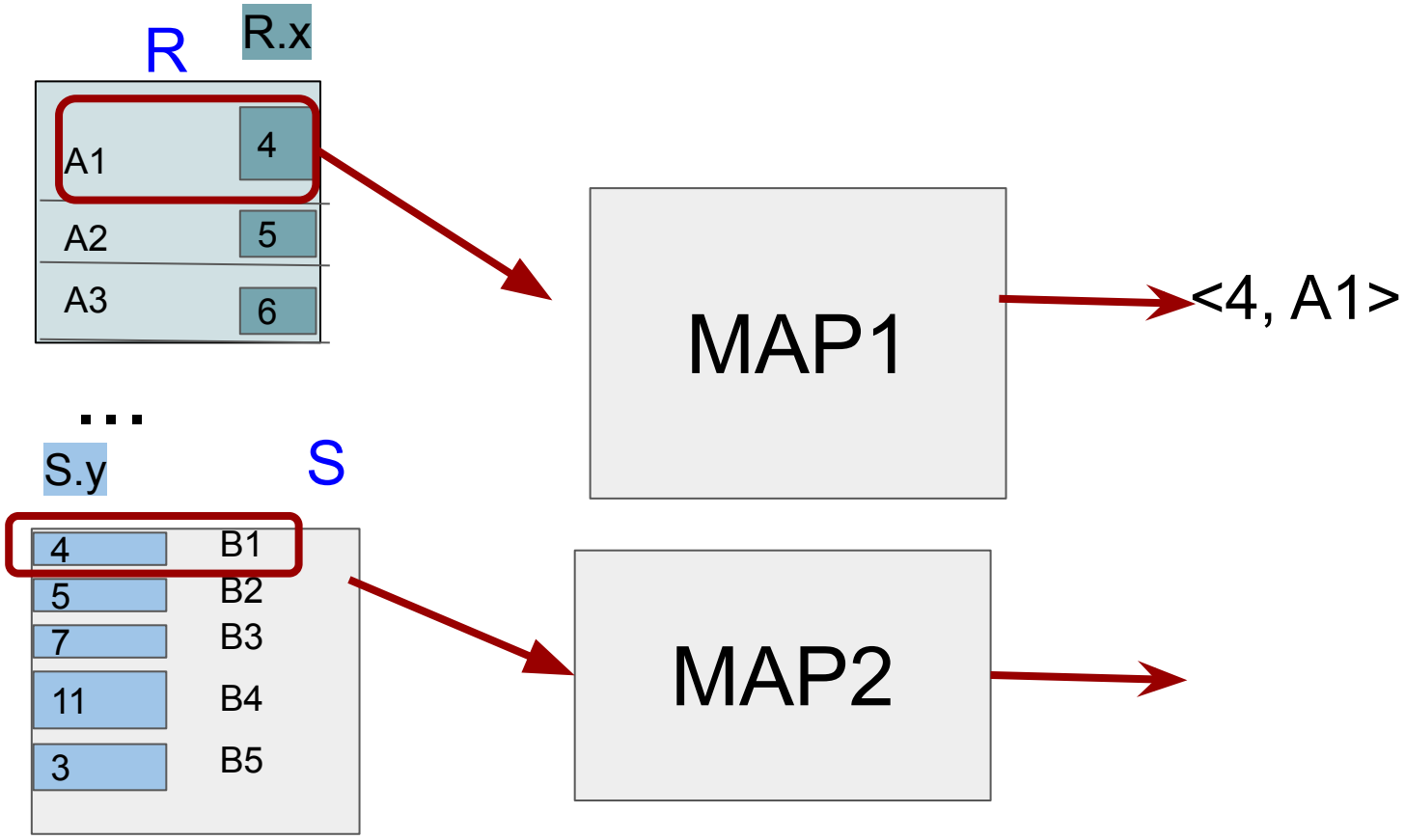
Reduce-side Join

```
R.x == S.y
```



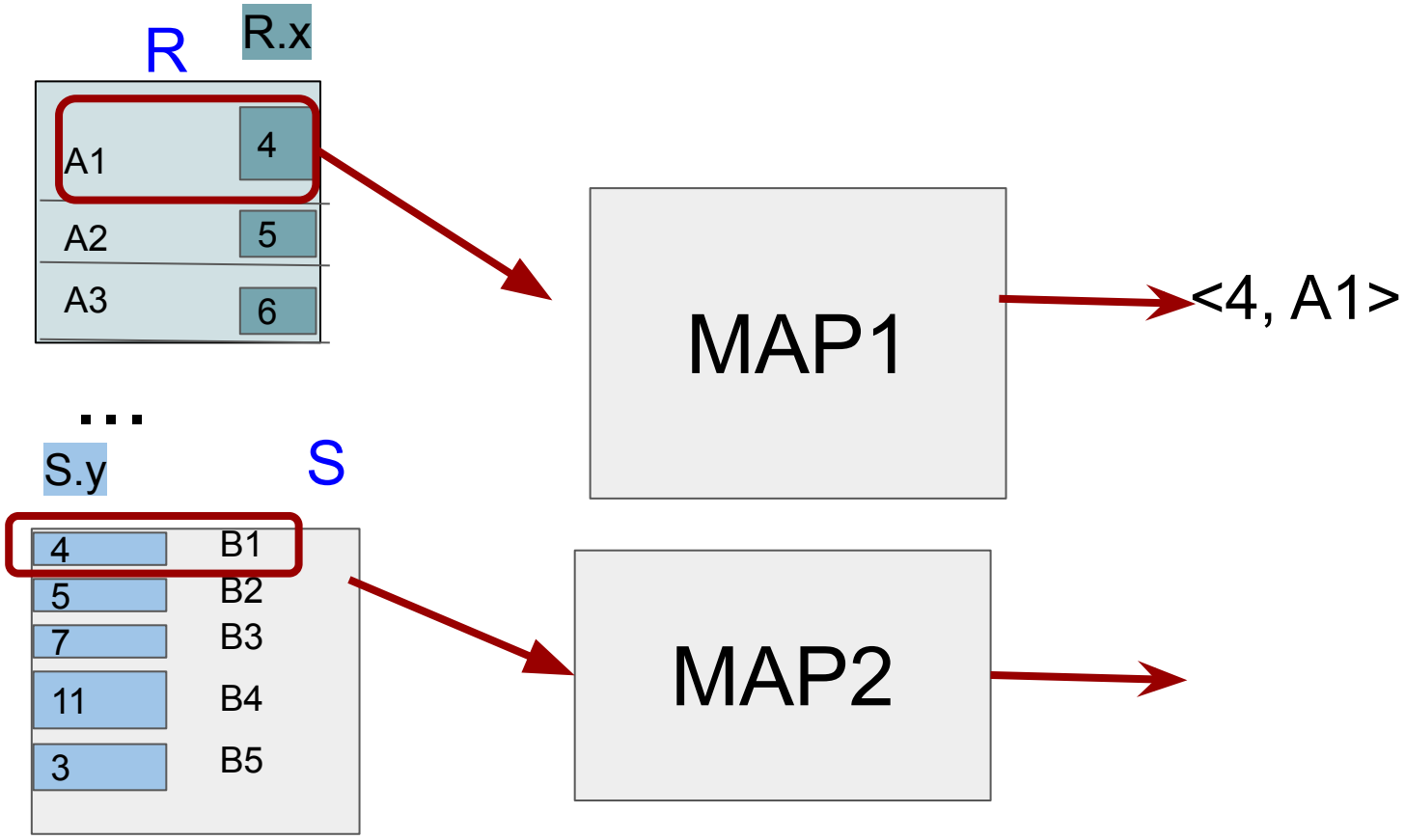
Reduce-side Join

```
R.x == S.y
```



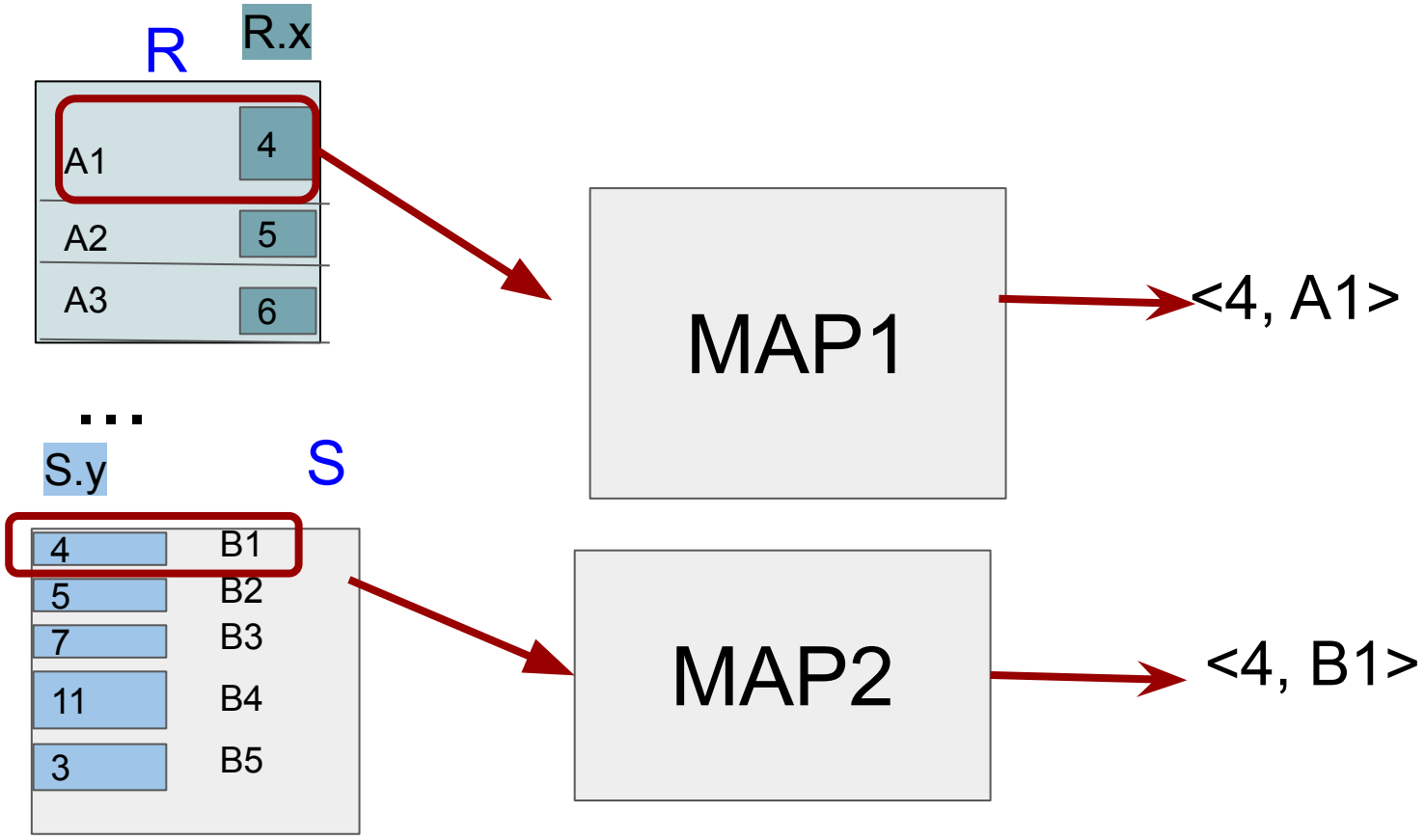
Reduce-side Join

```
R.x == S.y
```



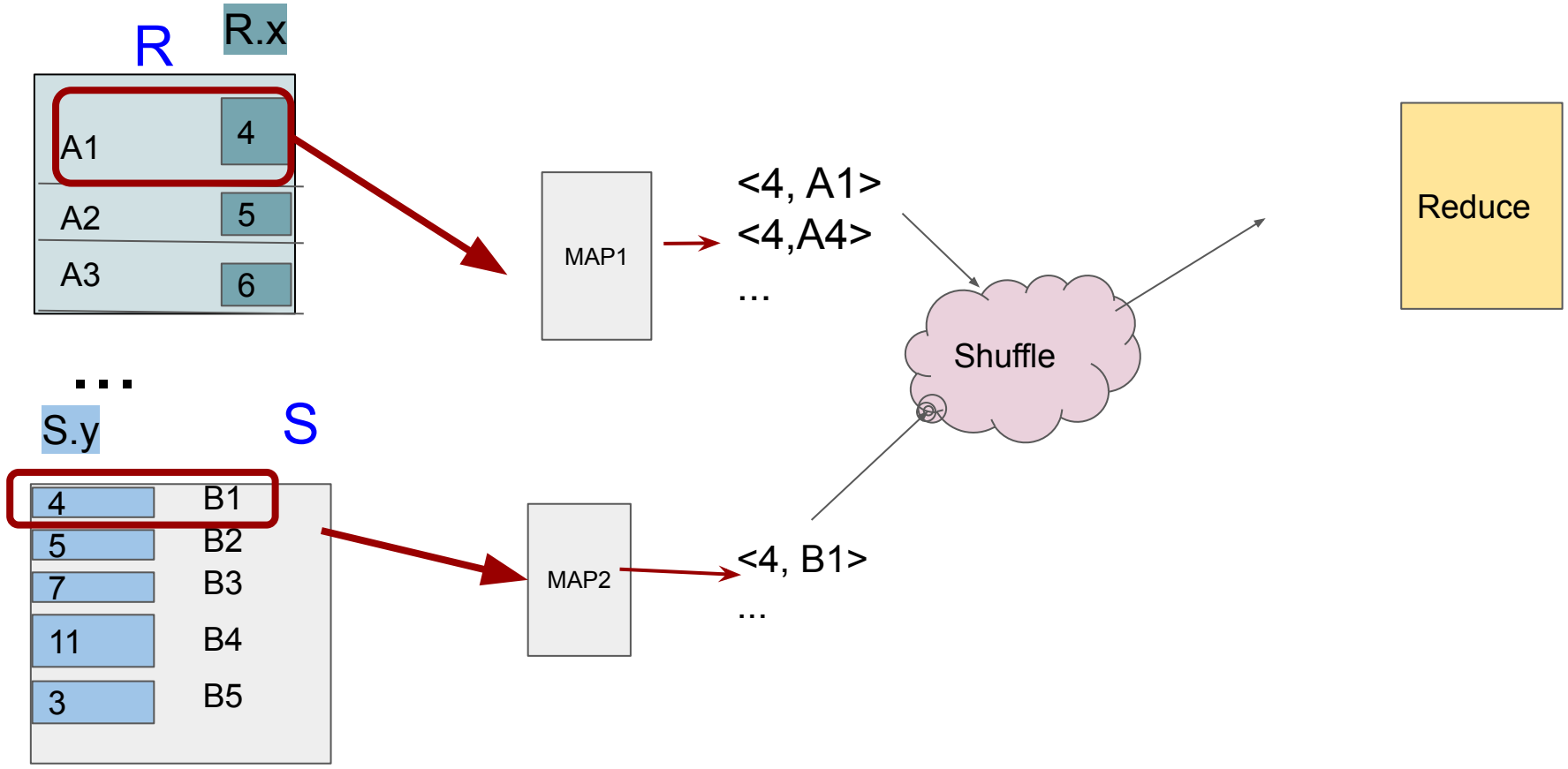
Reduce-side Join

```
R.x == S.y
```



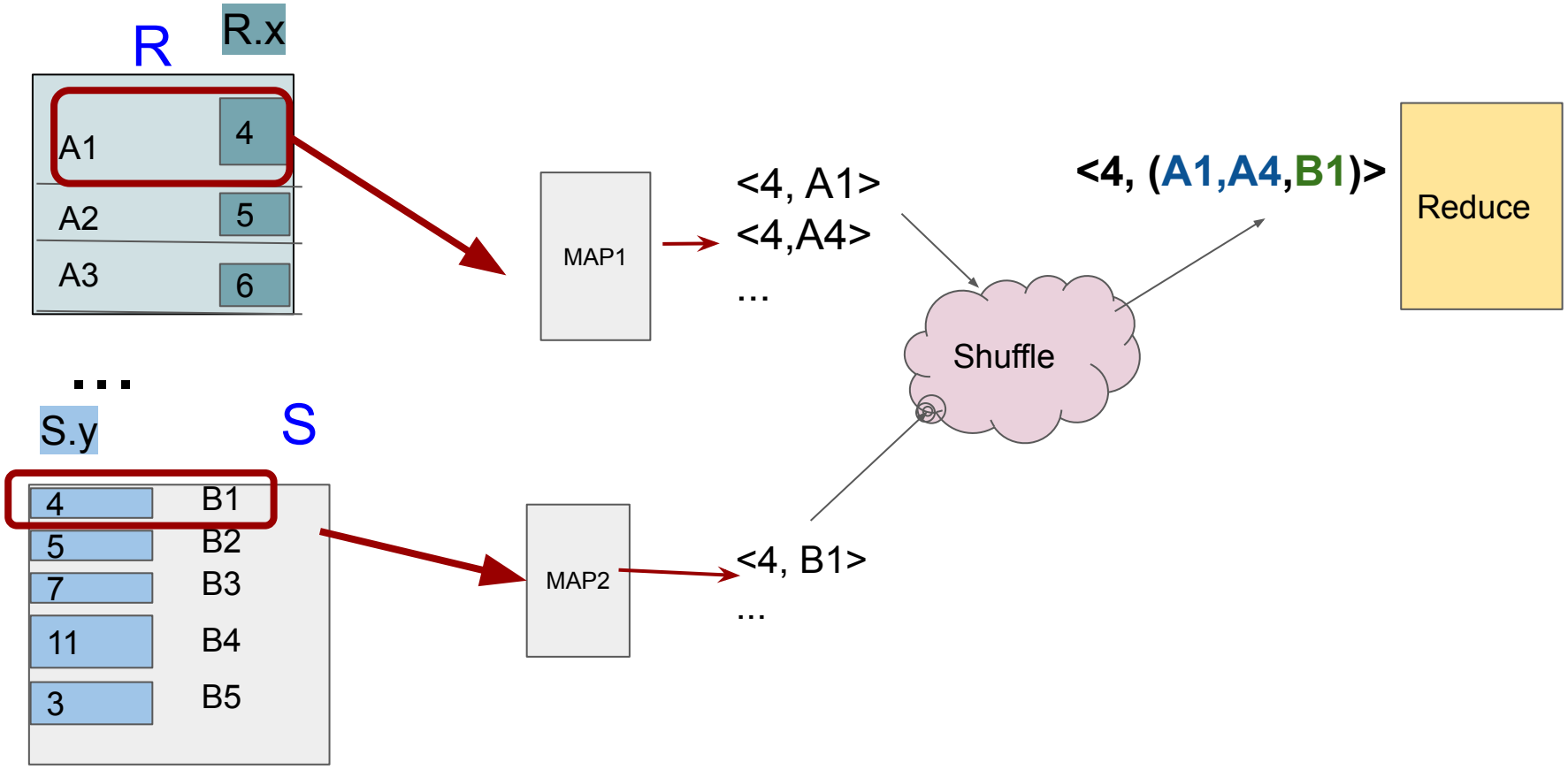
Reduce-side Join

```
R.x == S.y
```



Reduce-side Join

R.x == S.y



Reduce-side Join

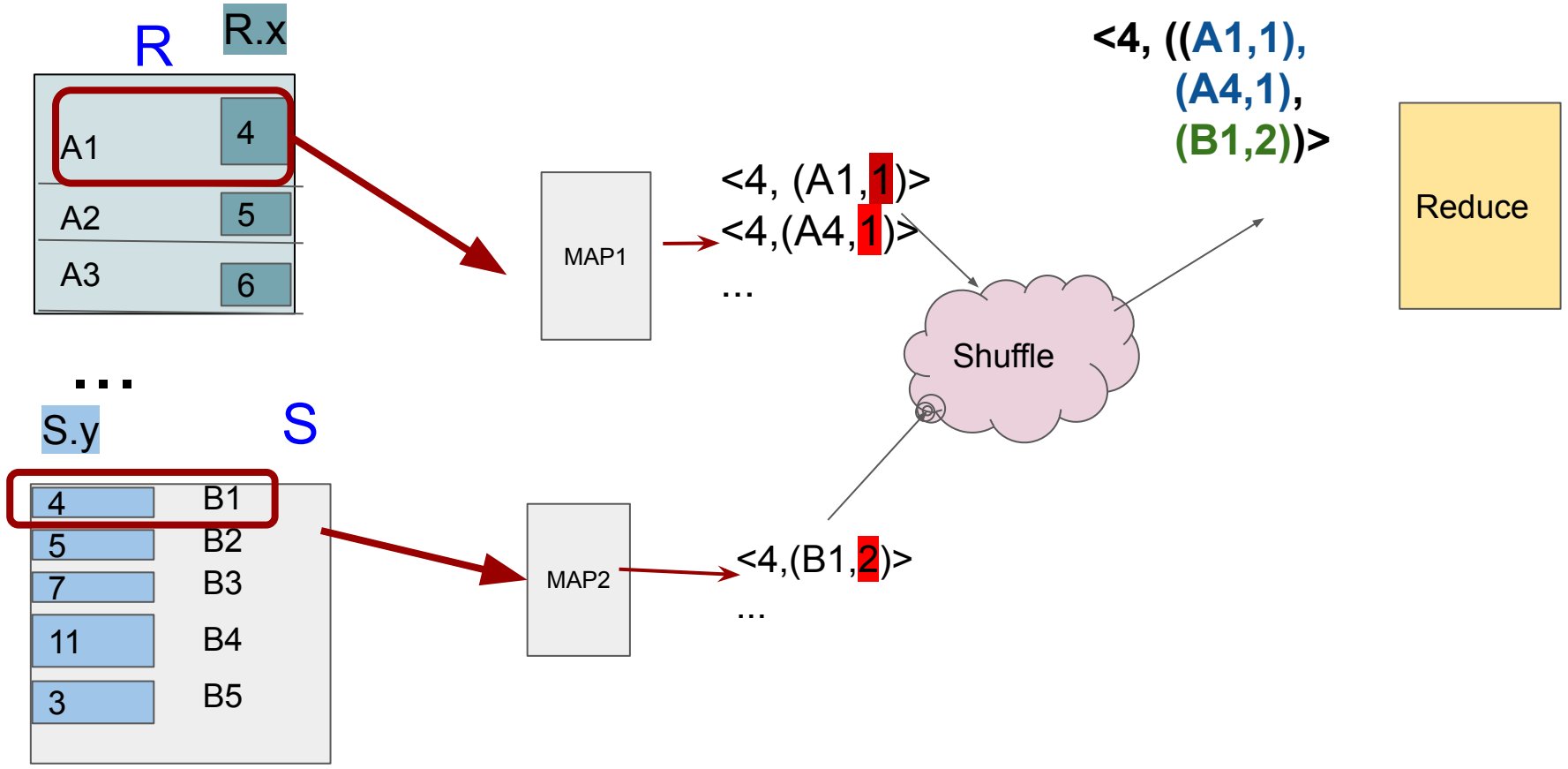
<4, (A1,A4,B1)>

Need a Trick!!!

Include Source Information into the output value of Map

Reduce-side Join

`R.x == S.y`



Reduce-Side Join: Maps

```
mapR(key, value)
    newKey = value.x
    newValue={“source”:1,
              “Content”:value}
    emit(newKey,newValue)
```

Reduce-Side Join: Maps

Encode the source

```
mapR(key, value)
    newKey = value.x
    newValue={"source":1,
              "Content":value}
    emit(newKey,newValue)
```

```
mapS(key, value)
    newKey = value.y
    newValue={"source":2,
              "Content":value}
    emit(newKey,newValue)
```

Reduce-Side Join: Maps + Reduce

```
mapR(key, value)
    newKey = value.x
    newValue={"source":1,
              "Content":value}
    emit(newKey,newValue)
```

```
mapS(key, value)
    newKey = value.y
    newValue={"source":2,
              "Content":value}
    emit(newKey,newValue)
```

```
reduce(key, Iterable values)
    fromR = []
    fromS = []
    for v in values do
        if v.source == 1 then
            fromR.append(v)
        else
            fromS.append(v)
  end for
```

Step 1: break input list apart by source

Reduce-Side Join: Maps + Reduce

```
mapR(key, value)
    newKey = value.x
    newValue={"source":1,
              "Content":value}
    emit(newKey,newValue)
```

```
mapS(key, value)
    newKey = value.y
    newValue={"source":2,
              "Content":value}
    emit(newKey,newValue)
```

```
reduce(key, Iterable values)
    fromR = []
    fromS = []
    for v in values do
        if v.source == 1 then
            fromR.append(v)
        else
            fromS.append(v)
    end for
```

Step 1: break input list apart by source

Reduce-Side Join: Maps + Reduce

```
mapR(key, value)
    newKey = value.x
    newValue={"source":1,
              "Content":value}
    emit(newKey,newValue)
```

```
mapS(key, value)
    newKey = value.y
    newValue={"source":2,
              "Content":value}
    emit(newKey,newValue)
```

```
reduce(key, Iterable values)
  fromR = []
  fromS = []
  for v in values do
     if v.source == 1 then
        fromR.append(v)
     else
        fromS.append(v)
  end for
  for r in fromR do
    for s in fromS do
        emit(key, (r,s))
```

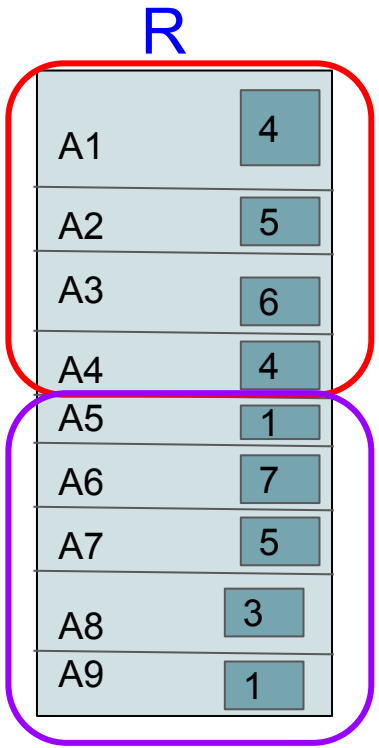
Step 2: emit cartesian product of records from R cross records from S

Reduce Side Join Demo (UserMessages)

Reduce-Side Joins

Advantages	**Disadvantages**
Combine two sources -- any size Sorting by join attribute	(R.x == S.y) = Empty Set If one user is **extremely** chatty -- most of the work could be performed by 1 reducer Sorting by join attribute Imbalance in size of inputs

Map-Side Join

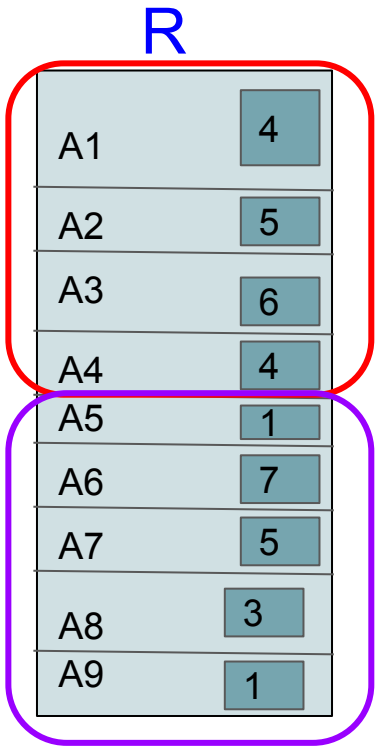


Assumptions:

R is LARGE

S is SMALL (fits main memory)

Map-Side Join



Assumptions:

R is LARGE

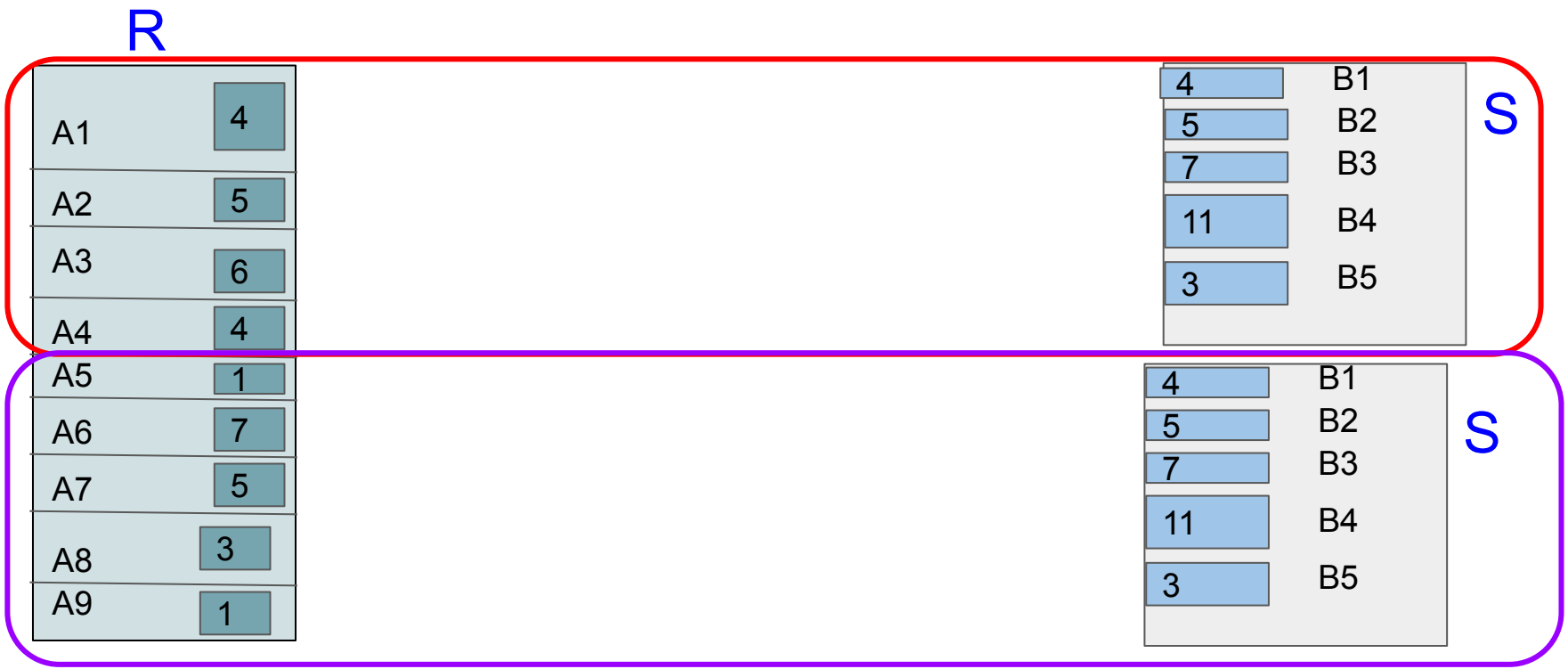
S is SMALL (fits in main memory)

Idea:

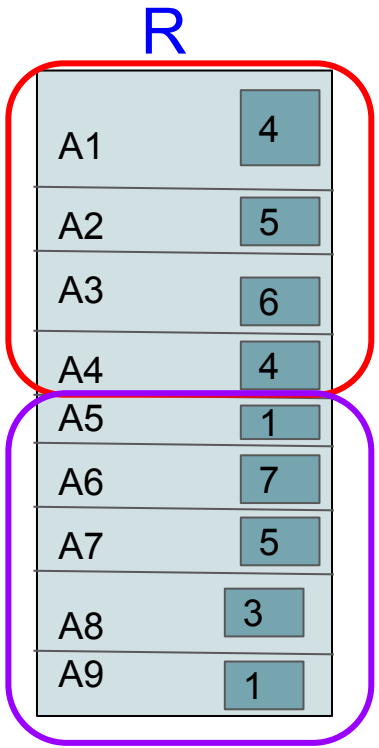
Split R

Load S onto every compute node

Map-Side Join



Map-Side Join



Assumptions:

R is LARGE

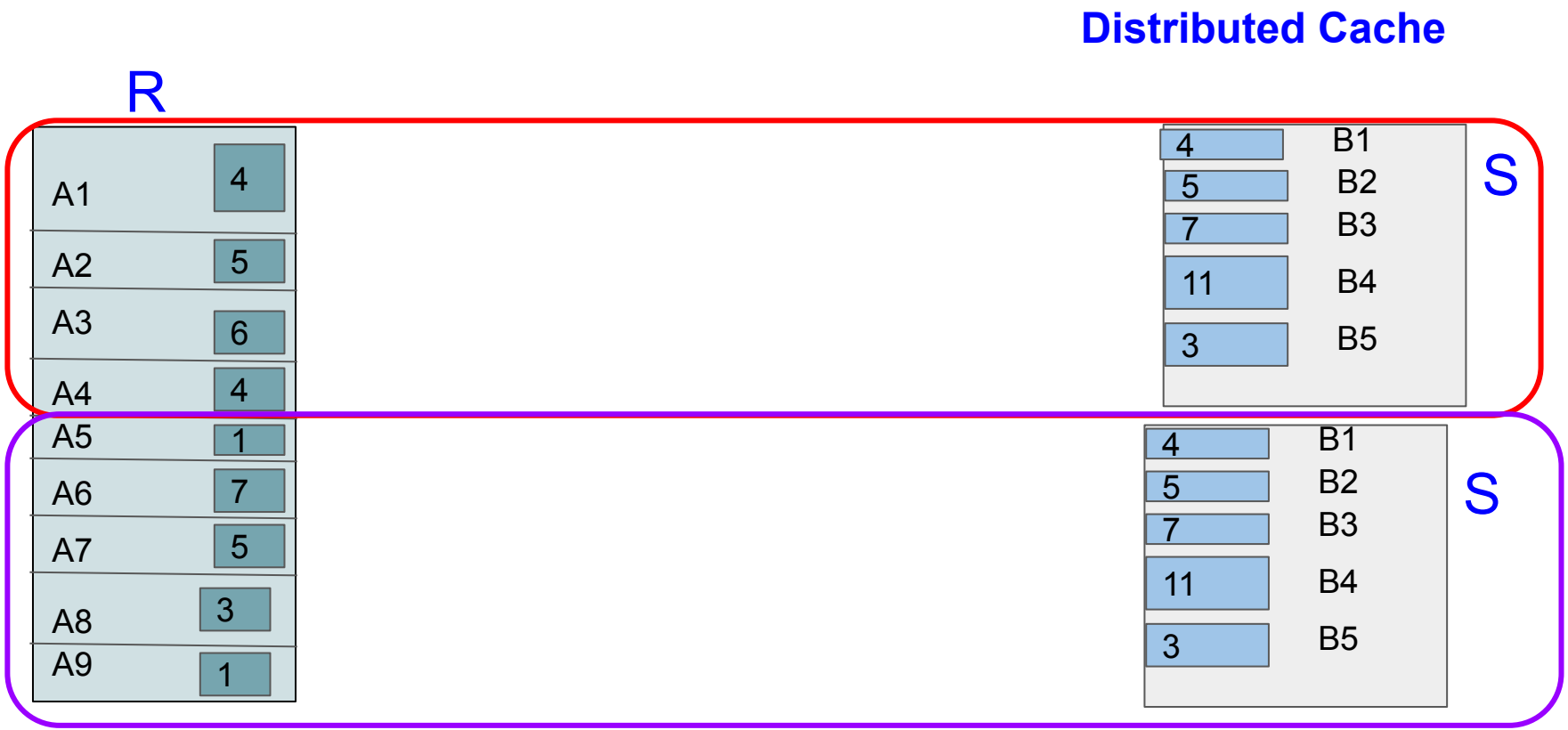
S is SMALL (fits main memory)

Idea:

Split R

Load S onto every compute node

Map-Side Join



Map-side Join Mapper:

```
hashMap dCache     // distributed cache is an instance
                   // variable in Mapper class

setup(cacheFile):
   // upload cache file into a convenient data structure
   file = open(cacheFile, "read")

   for each line in cacheFile do
     record = parse(line)     // use whatever parsing needed
     dCache.insert(record)
    end for
```

Map-Side Join Mapper:

```
hashMap dCache      // distributed cache is an instance
                    // variable in Mapper class

map(key, value):    // map gets input parameters from the
                    // other file

   // naive loop implementation, improve with hash or sort
   for  record in dCache do
      if  join condition on the pair (value, record) is true
      then emit(null, (value, record))
```

Map-Side Join Mapper:

```
reduce(key, Iterable values):

 for record in values do
   emit(null,values)
 end for
```

Map-Side Joins

Advantages	Disadvantages
Just one mapper to implement No extra sorting effort	Not suitable when both inputs are large No "free" sort