15-440 Distributed Systems Recitation 12

Tamim Jabban

MapReduce Applications

- Image Processing
- Sorting

MapReduce Applications

- Image Processing
- Sorting

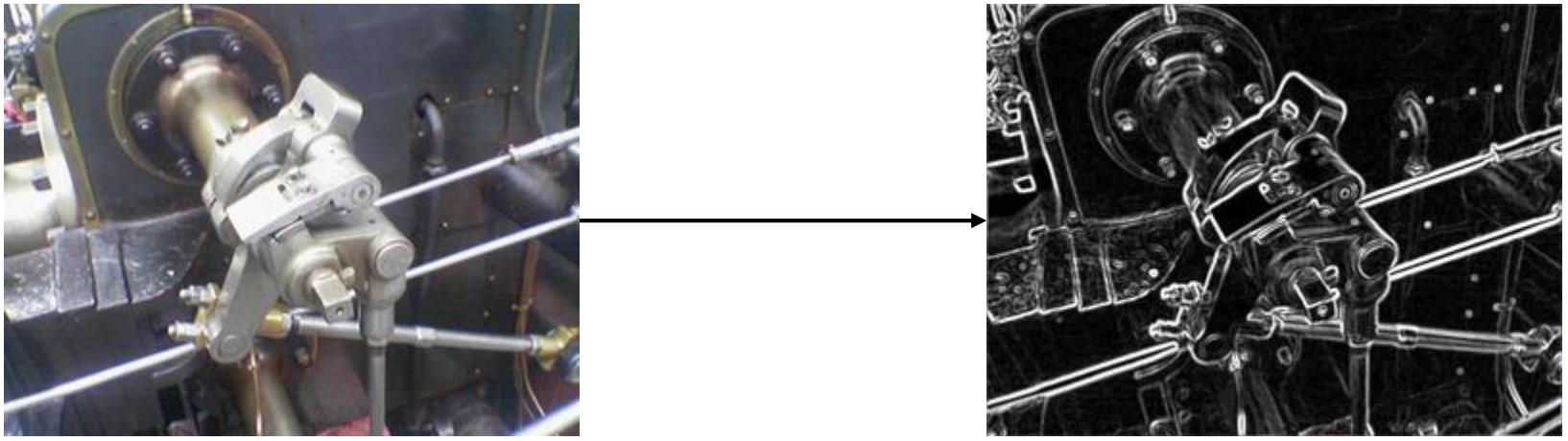
Image Processing

- Image processing involves the **application of some operation** on an Image
  - Images are typically represented as a 2D-Matrix of values (Binary, Grayscale or RGB/CYMK color values)
  - Operations on images are typically *matrix operations*.
- Image Processing can be a **computationally intensive process**!

Original Image

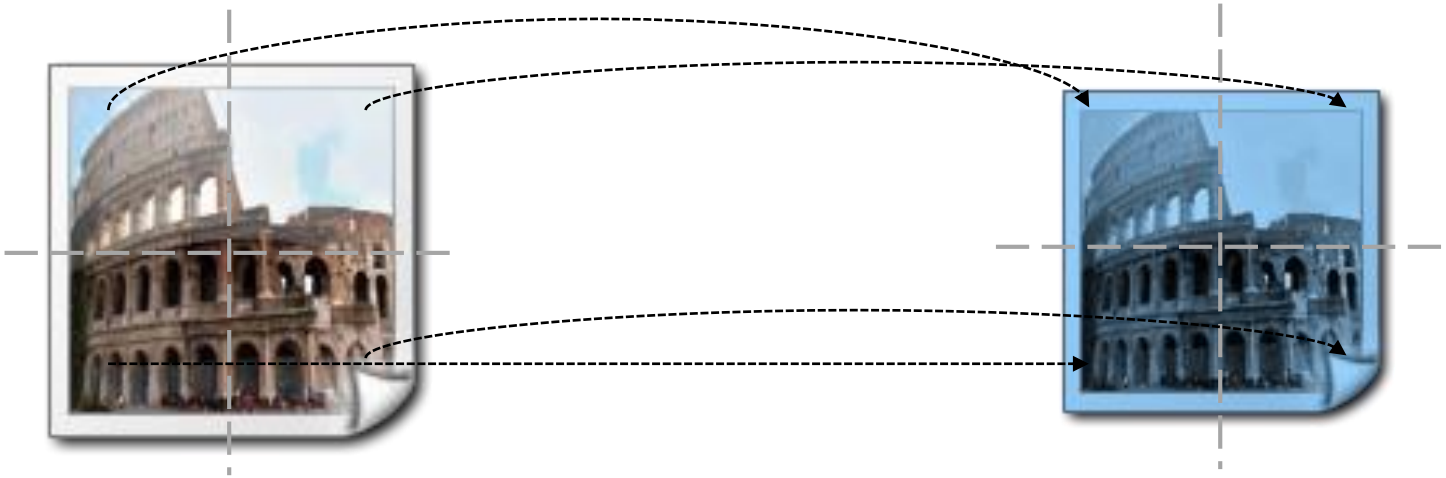
Processed Image

Ex: Sobel Edge Detection



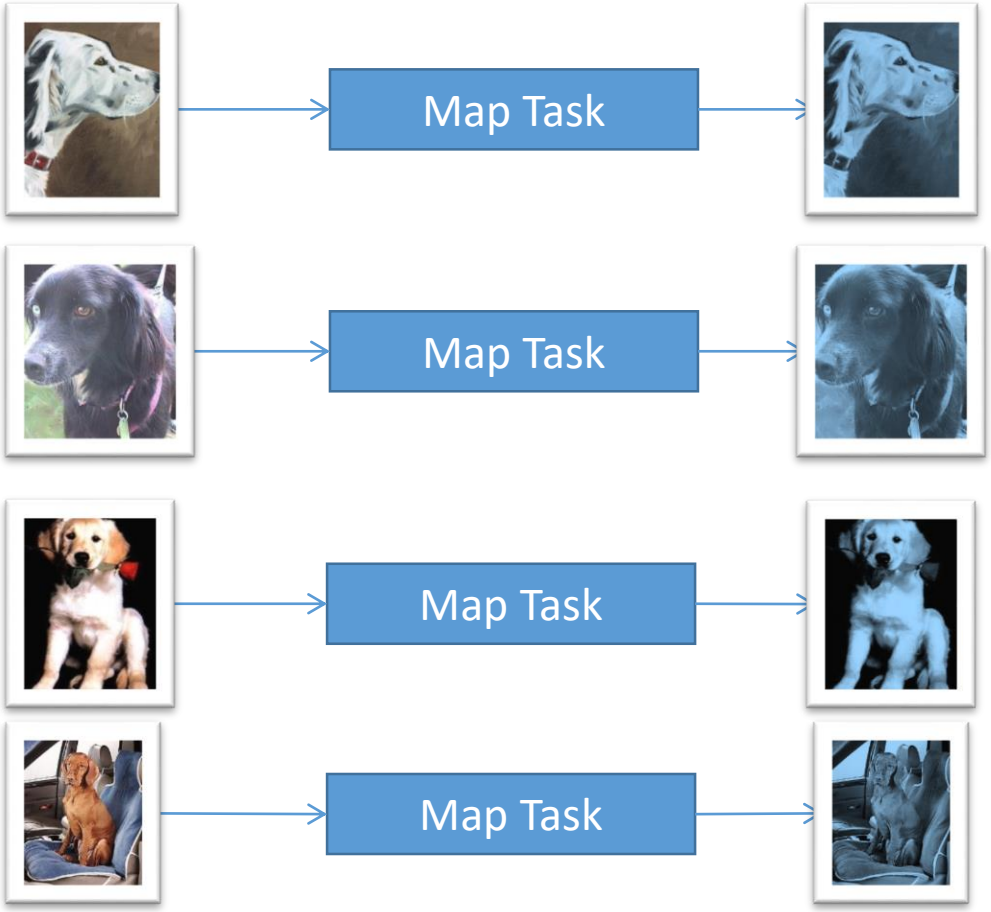
How to use MapReduce?

- Parallelize operations *within* an image



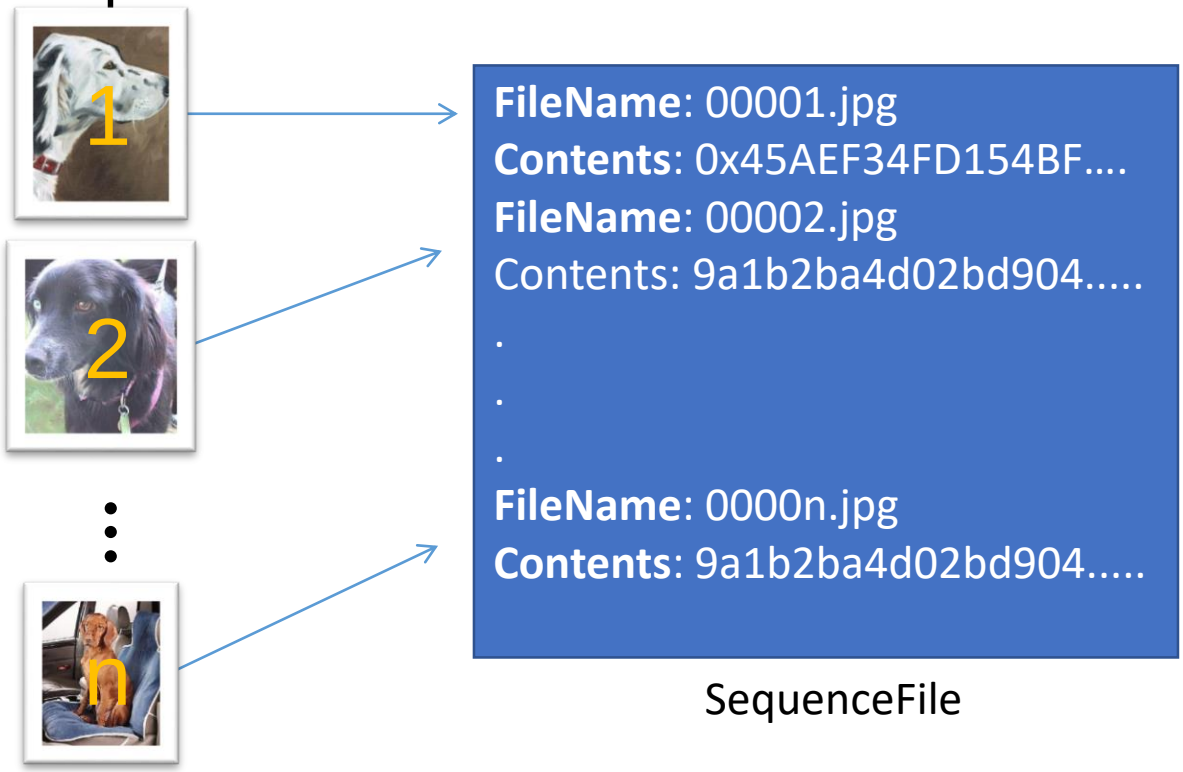
- Parallelize operations *across multiple* images

Naïve Implementation

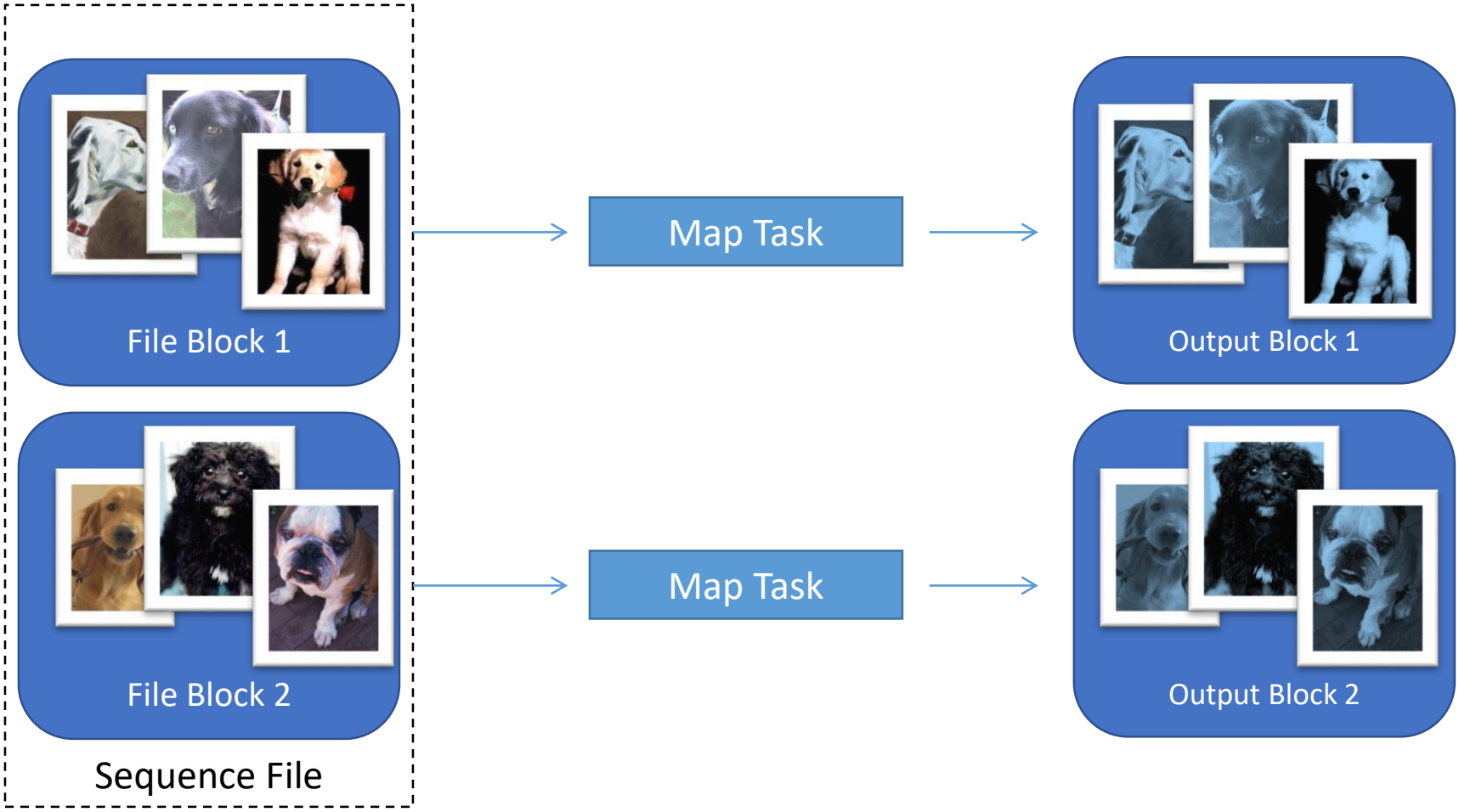


Naïve Implementation

- Large Number of Mappers
- HDFS is inefficient for small files.
  - Use `SequenceFile`!



Better Implementation



Ex: Sobel Edge Detection

```
public static class ImagePMapper extends Mapper<Text,
       BytesWritable, Text, BytesWritable>{

    //Sobel Kernels
    float[] xKernel = {
      -1.0f, 0.0f, 1.0f,
      -2.0f, 0.0f, 2.0f,
      -1.0f, 0.0f, 1.0f
    };

    float[] yKernel = {
      -1.0f, -2.0f, -1.0f,
       0.0f,  0.0f,  0.0f,
       1.0f,  2.0f,  1.0f
    };
```

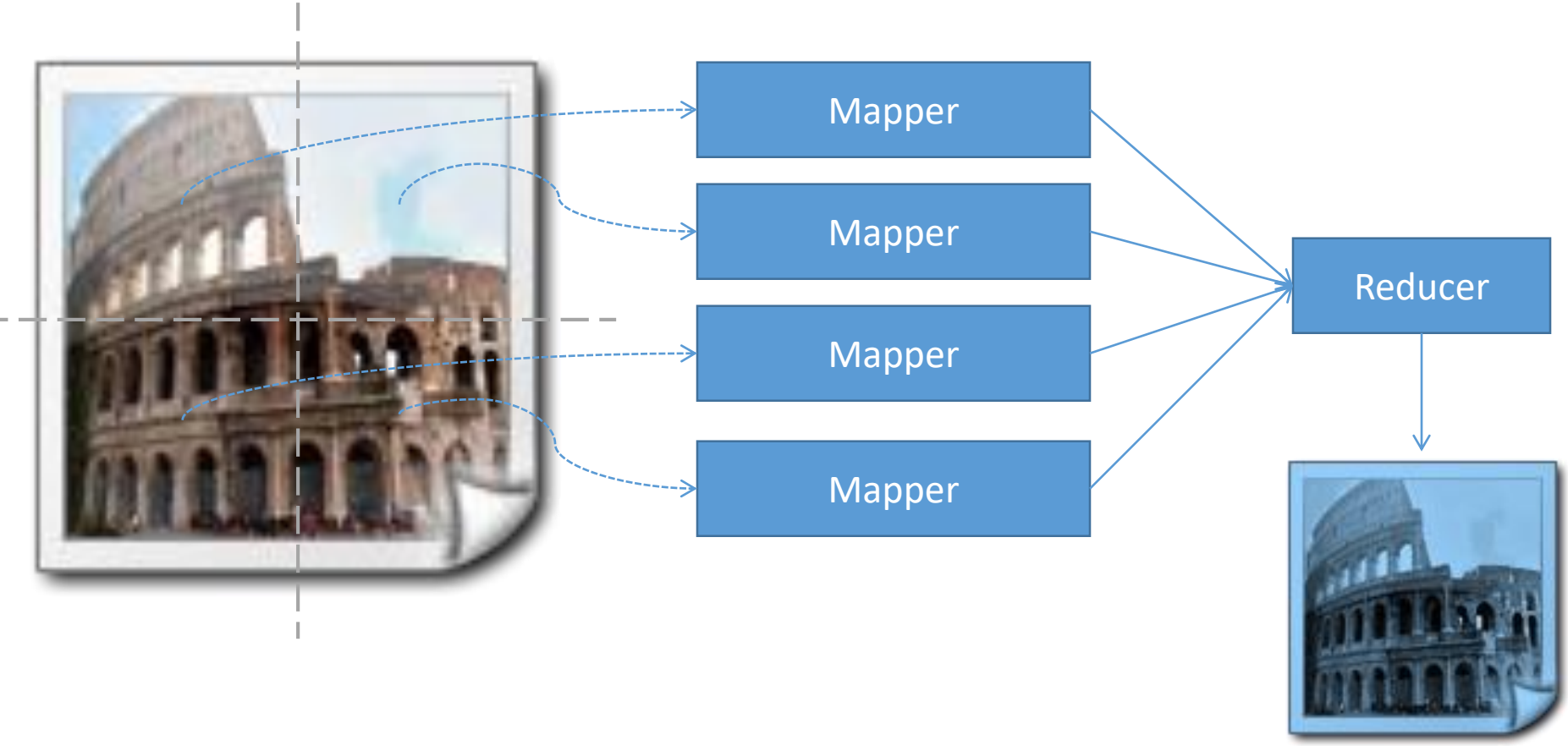
Ex: Sobel Edge Detection

```
public void map(Text key, BytesWritable value,
    Context context) throws IOException, InterruptedException{

 //Read Image
 InputStream in = new ByteArrayInputStream(value.getBytes());
 BufferedImage bImageFromConvert = ImageIO.read(in);

 //Perform Sobel Operations on Image
 ConvolveOp blurX = new ConvolveOp(new Kernel(3, 3, xKernel));
 BufferedImage x = blurX.filter(bImageFromConvert, null);

 ConvolveOp blurY = new ConvolveOp(new Kernel(3, 3, yKernel));
 BufferedImage y = blurY.filter(x, null);
```

Ex: Sobel Edge Detection

```
    //Create Output ByteStream
    BytesWritable outputBytes = new BytesWritable();
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ImageIO.write( y, format, baos );
    baos.flush();
    byte[] imageInByte = baos.toByteArray();
    baos.close();
    outputBytes.set(imageInByte, 0, imageInByte.length);

    //Send output key,value pairs.
    context.write(key, outputBytes);
    }
}
```

Strategy for Larger Images



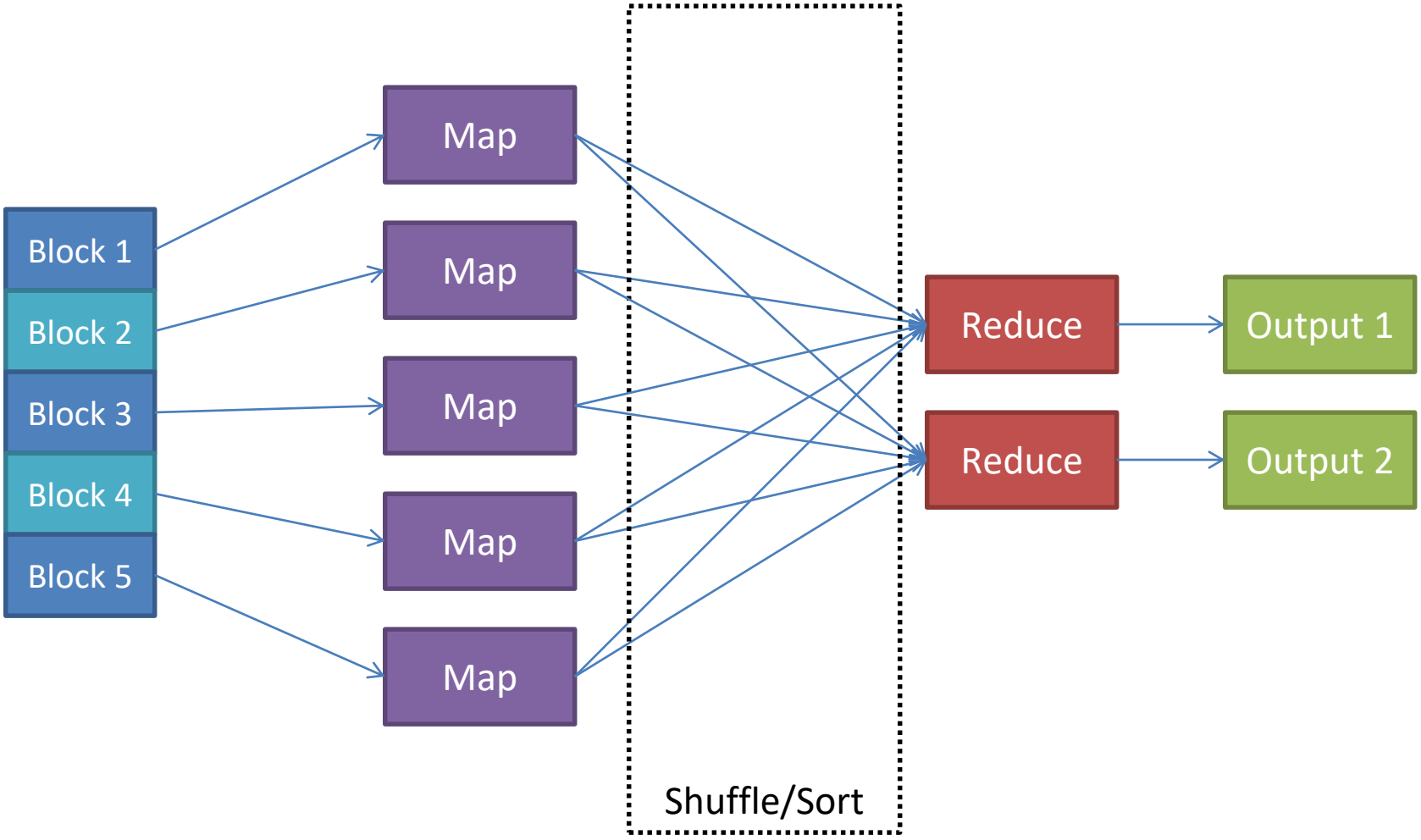
MapReduce Applications

- Image Processing
- Sorting

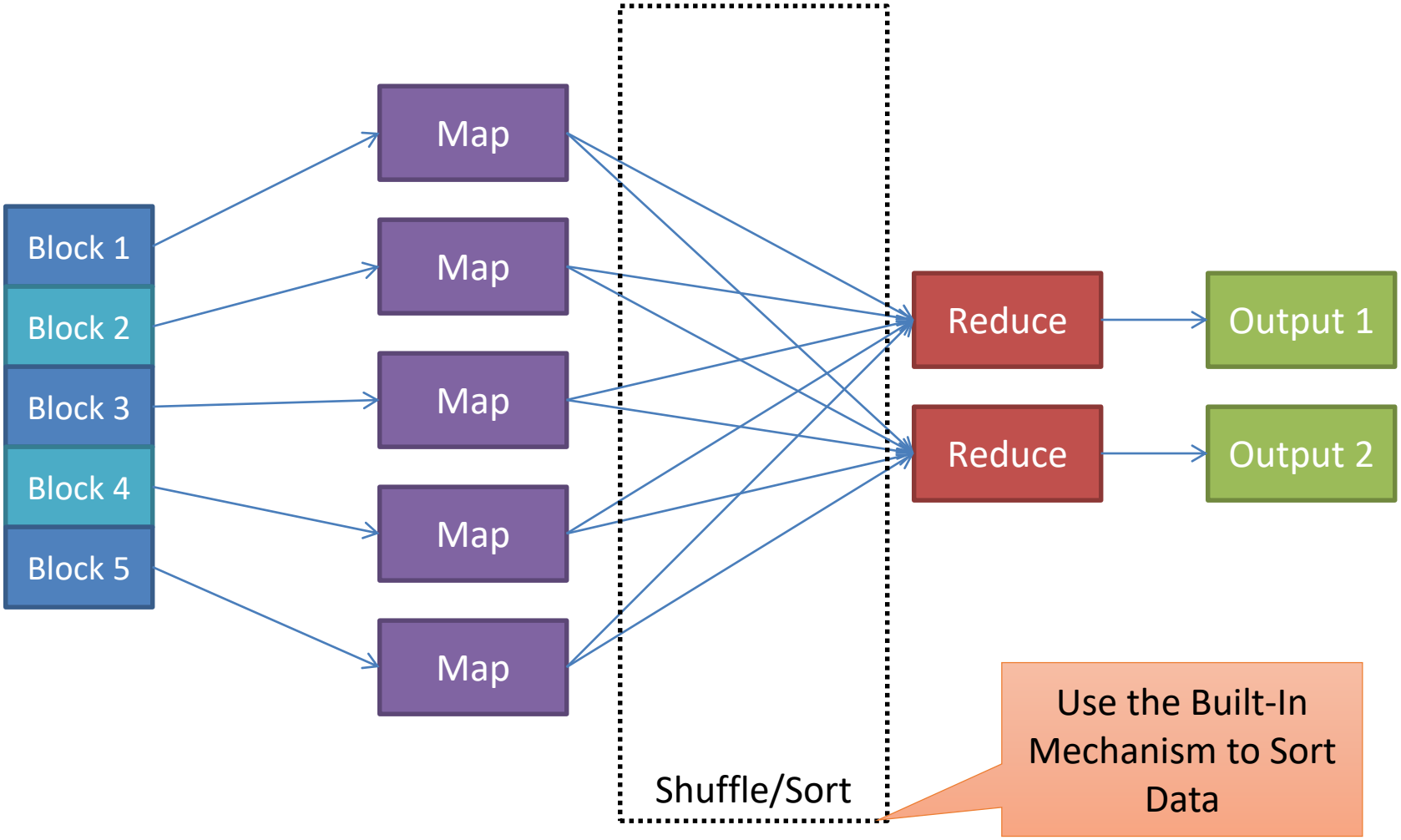
MapReduce Applications

- Image Processing
- Sorting

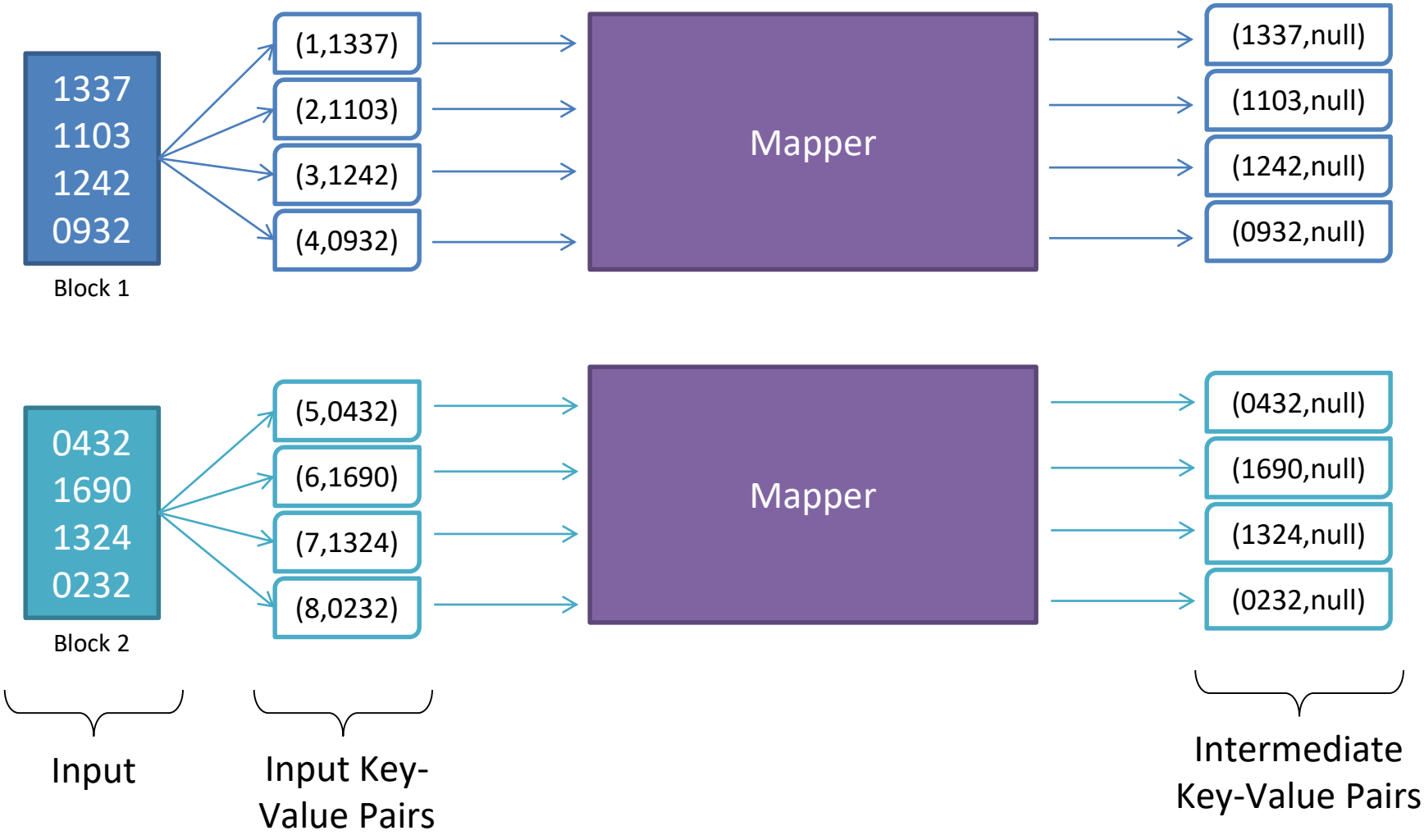
Sorting


Block 1
Block 2
Block 3
Block 4
Block 5
Map
Map
Map
Map
Map
Shuffle/Sort
Reduce
Reduce
Output 1
Output 2

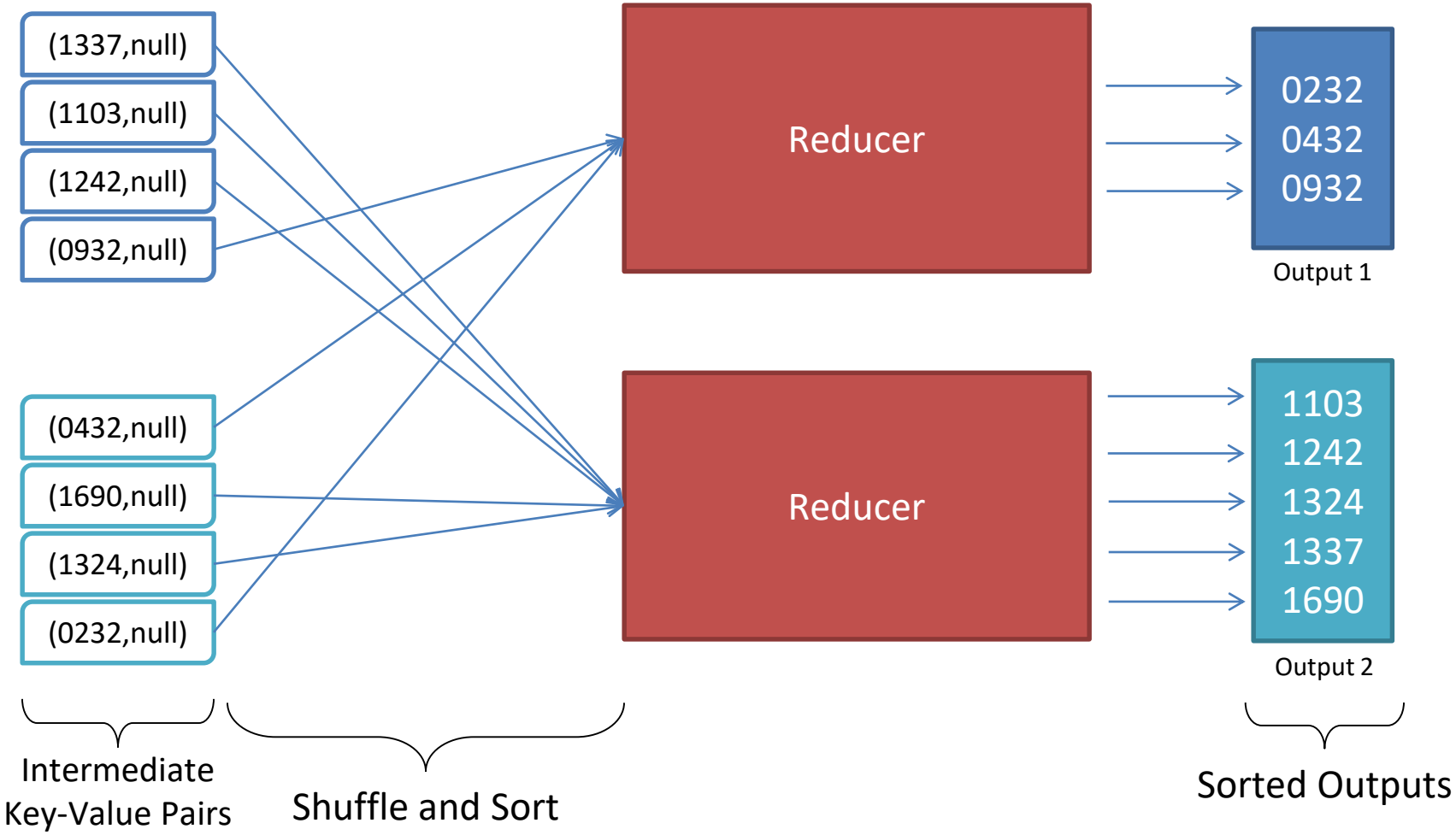
Sorting


Block 1
Block 2
Block 3
Block 4
Block 5
Map
Map
Map
Map
Map
Reduce
Reduce
Output 1
Output 2
Shuffle/Sort
Use the Built-In Mechanism to Sort Data

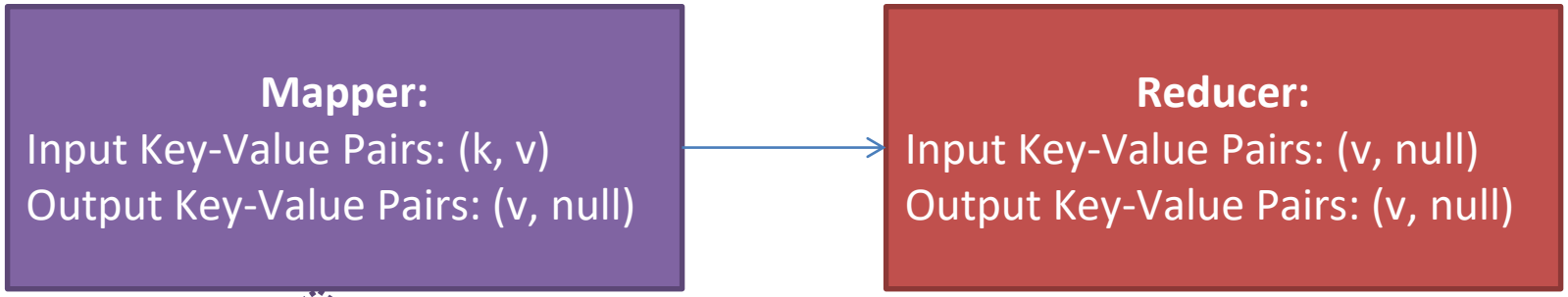
Mapper


1337
1103
1242
0932
Block 1
(1,1337)
(2,1103)
(3,1242)
(4,0932)
Mapper
(1337,null)
(1103,null)
(1242,null)
(0932,null)
0432
1690
1324
0232
Block 2
(5,0432)
(6,1690)
(7,1324)
(8,0232)
Mapper
(0432,null)
(1690,null)
(1324,null)
(0232,null)
Input
Input Key-Value Pairs
Intermediate Key-Value Pairs

Reducer


(1337,null)
(1103,null)
(1242,null)
(0932,null)
(0432,null)
(1690,null)
(1324,null)
(0232,null)
Reducer
Reducer
0232
0432
0932
Output 1
1103
1242
1324
1337
1690
Output 2
Intermediate Key-Value Pairs
Shuffle and Sort
Sorted Outputs

MapReduce Program




```
public void map(Object key, Text value, Context context)
throws Exception
{
//Parse the value, if required.

context.write(value, null);

}
```

MapReduce Program

Mapper:
Input Key-Value Pairs: (k, v)
Output Key-Value Pairs: (v, null)

Reducer:
Input Key-Value Pairs: (v, null)
Output Key-Value Pairs: (v, null)

```
public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException
{

      while(values.iterator().hasNext())
      {
            context.write(key, values.iterator().next());
      }

}
```