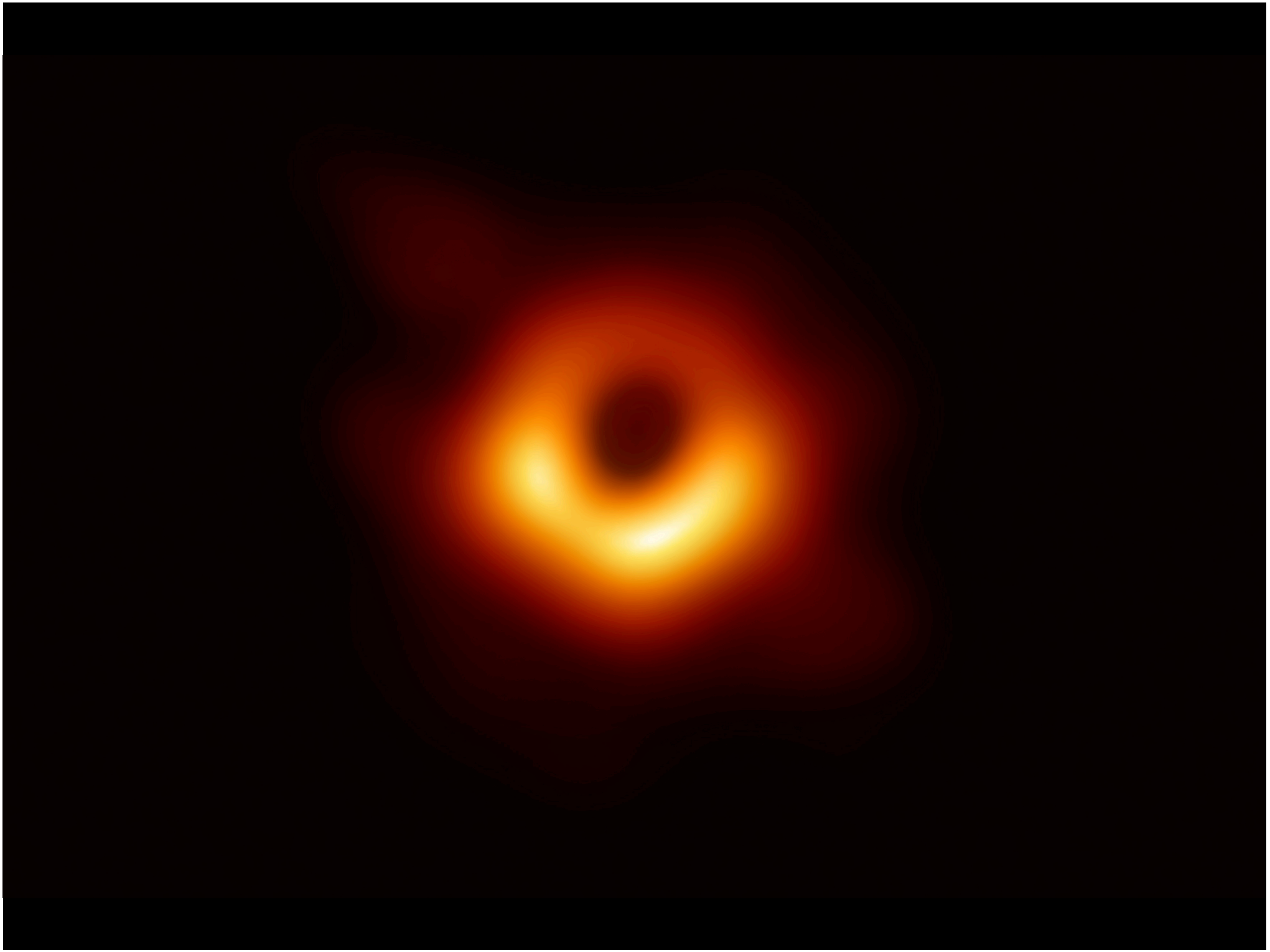
ETH zürich

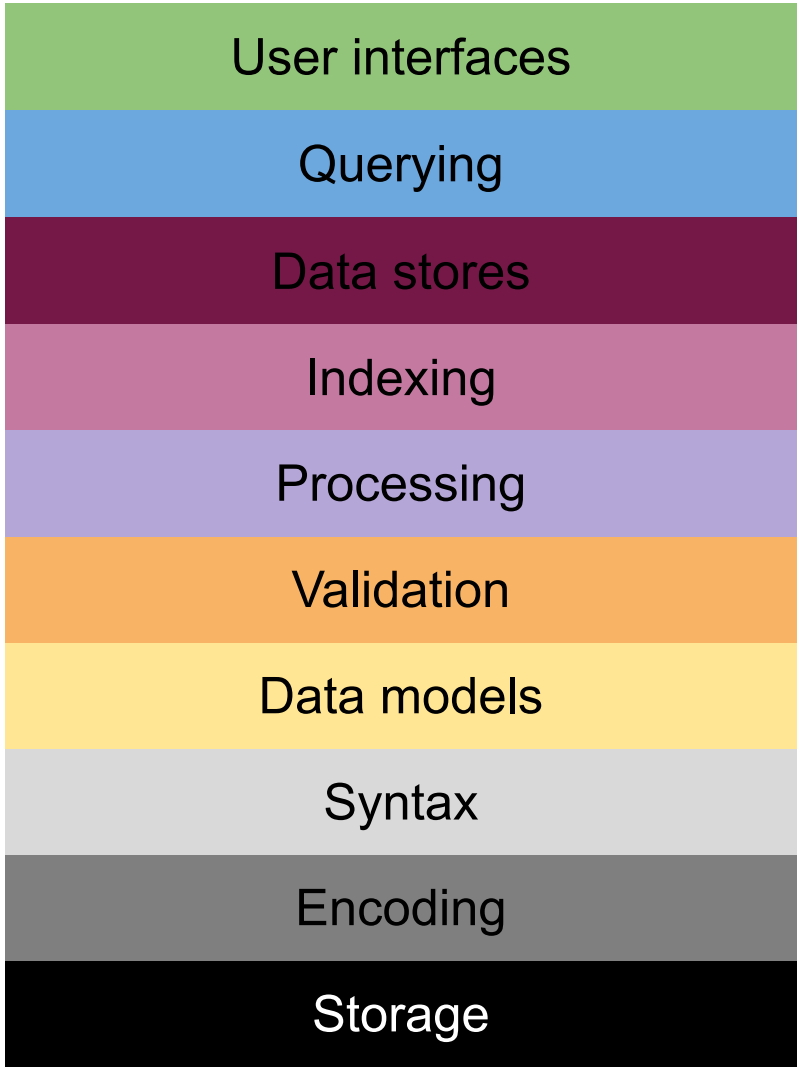
Ghislain Fourny

Big Data for Engineers Spring 2020

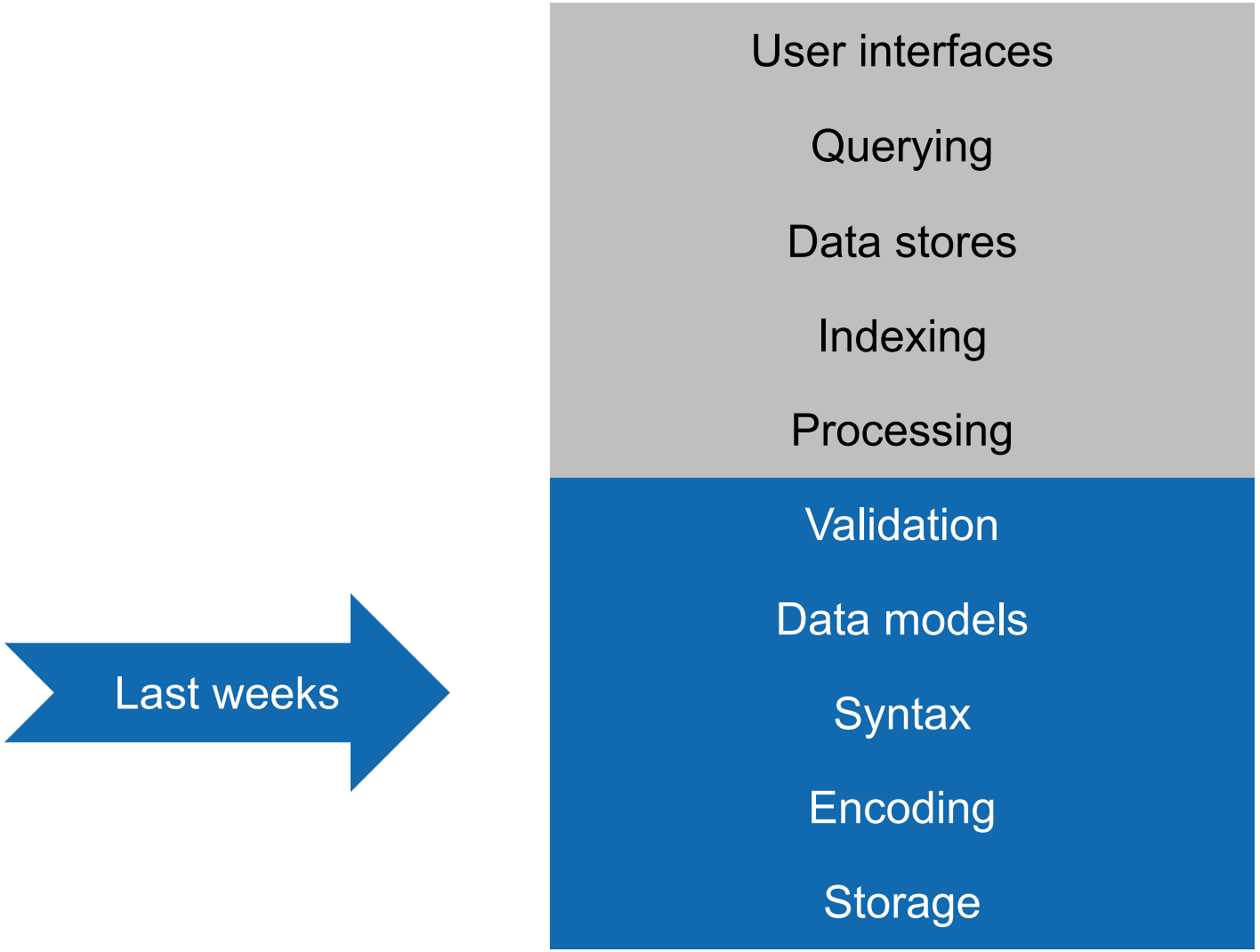
8. Distributed Computations I: MapReduce



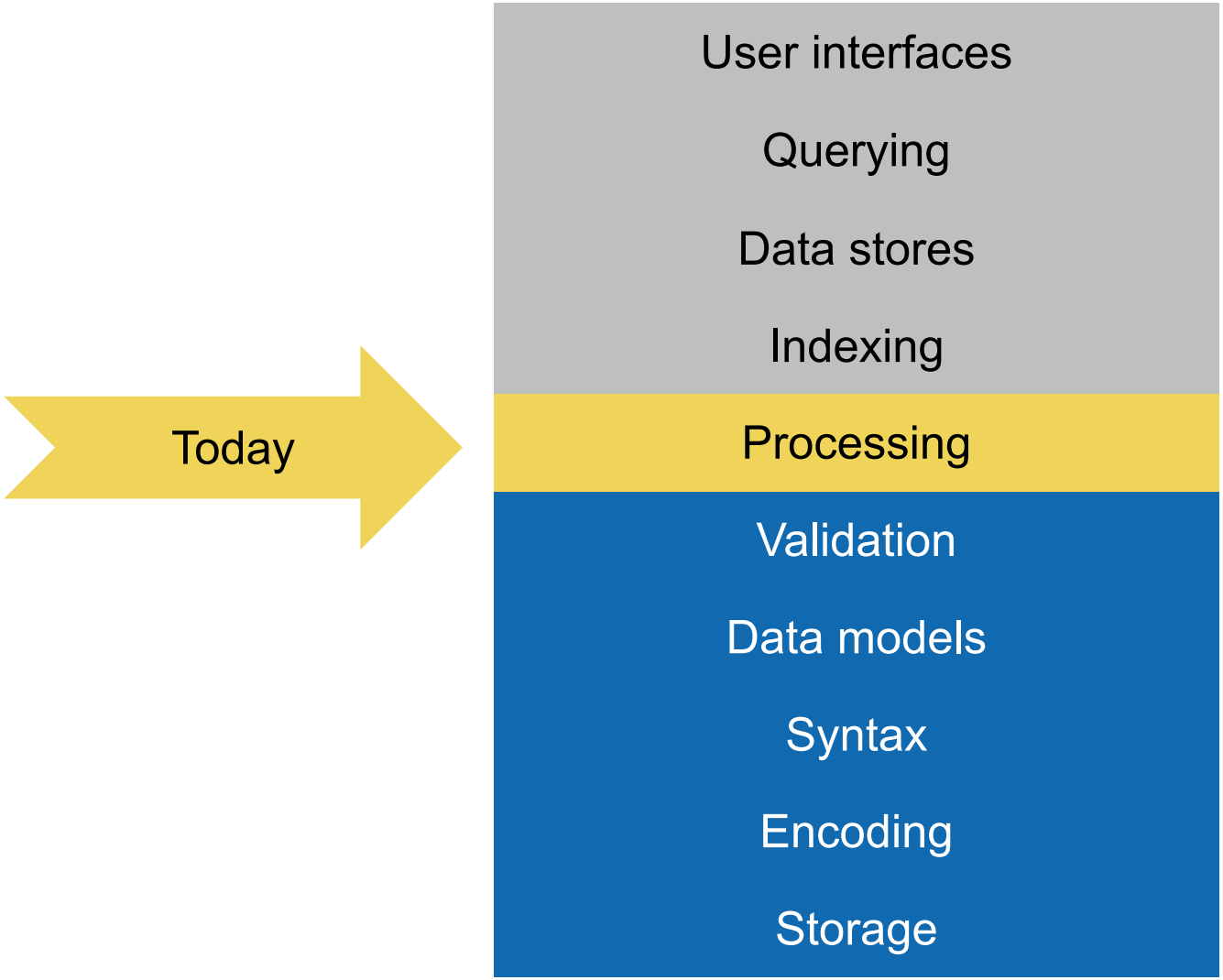
Data Technology Stack



Where we are


User interfaces
Querying
Data stores
Indexing
Processing
Validation
Data models
Syntax
Encoding
Storage
Last weeks

Where we are


Today
User interfaces
Querying
Data stores
Indexing
Processing
Validation
Data models
Syntax
Encoding
Storage

Let's begin with a field experiment

400+ Pokemons, 10 different

How many of each?

 ?

 ?

 ?

 ?

 ?

 ?

 ?

 ?

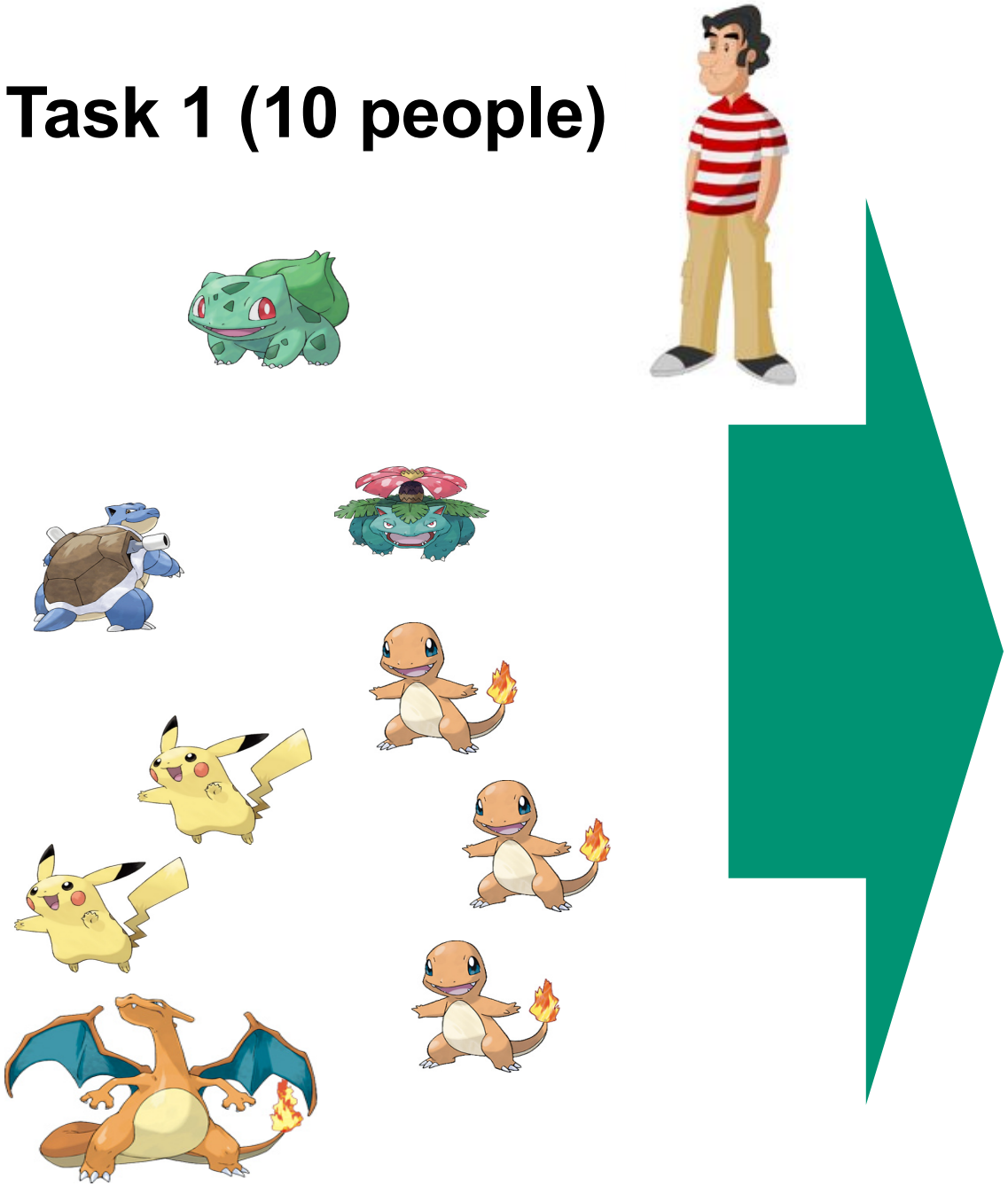
 ?

 ?

400 distributed to many volunteers

Task 1 (10 people)

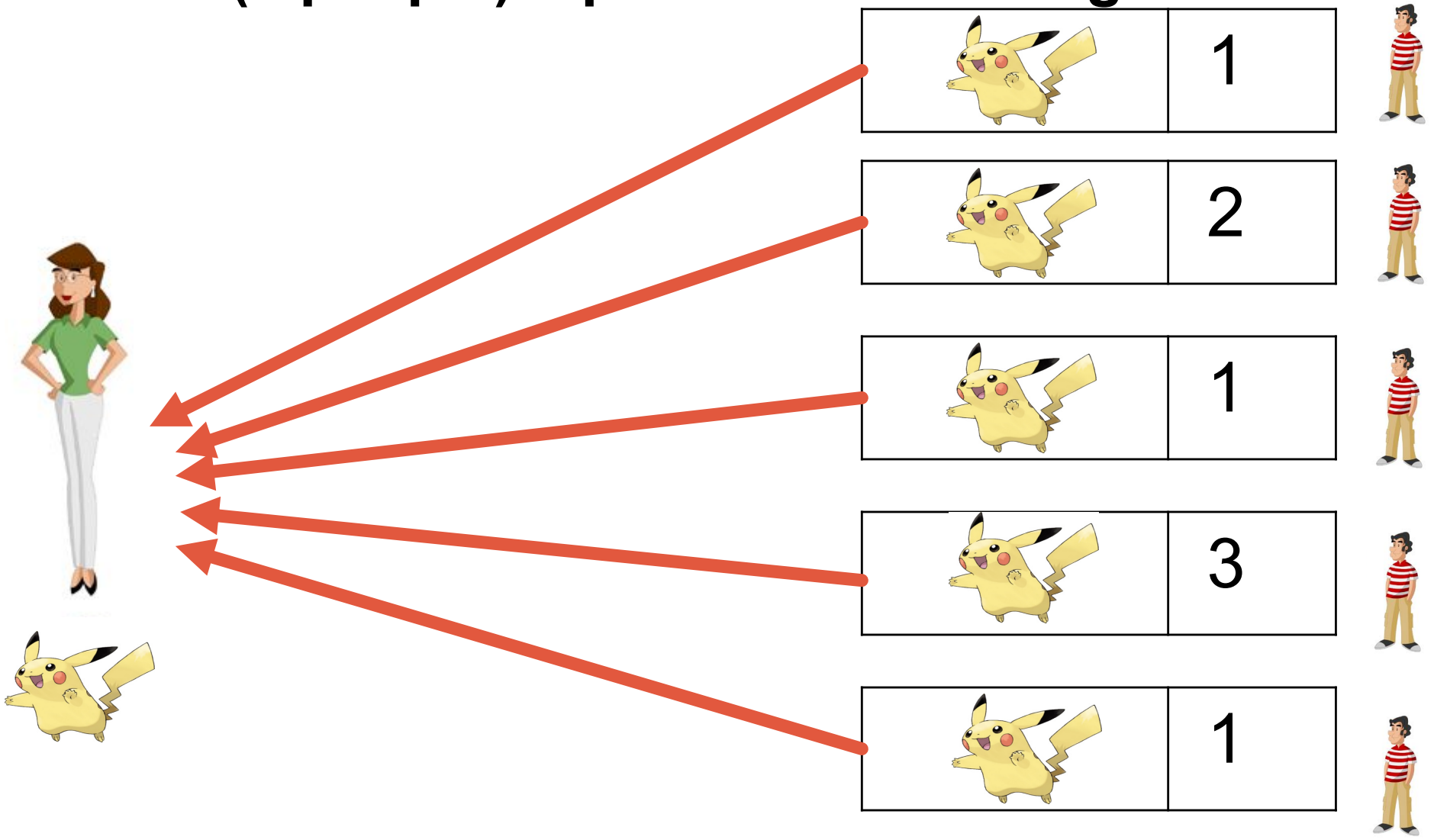
Task 1 (10 people)



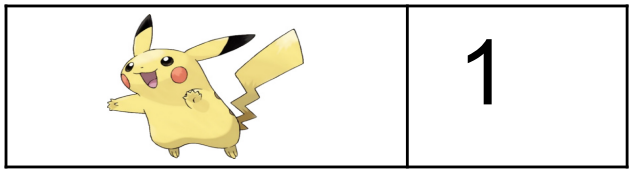
	1
	3
	1
	2
	1
	1

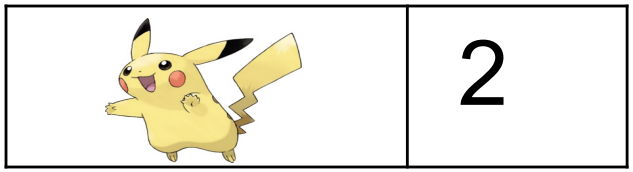
Task 2 (8 people)

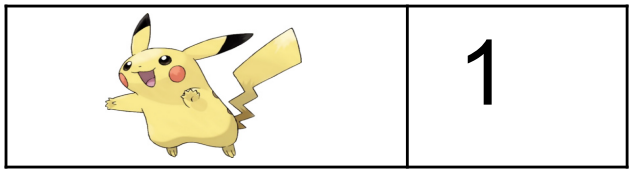
Task 2 (8 people) – part 1 aka "The big mess"

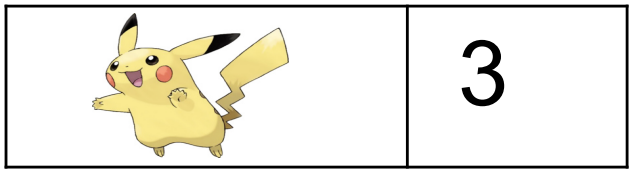


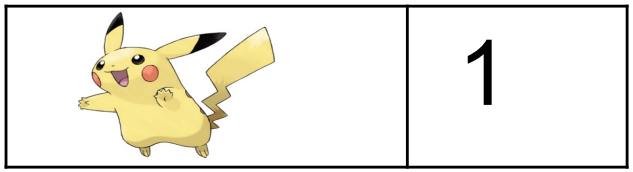
Task 2 (8 people) – part 2

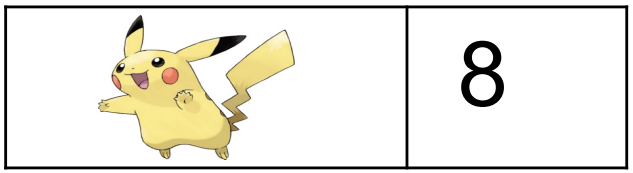




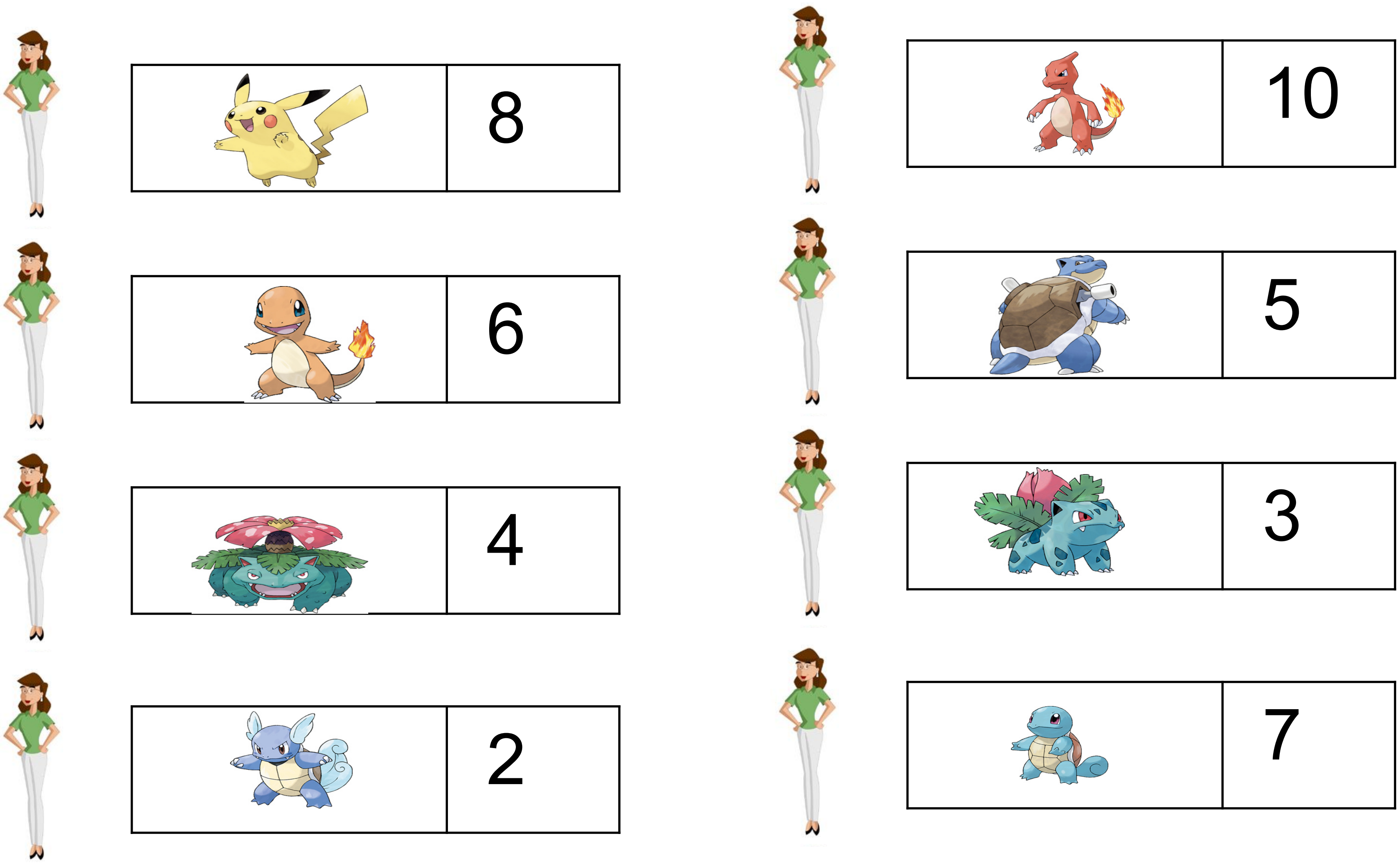








Final summary



Let's go!

So far, we have...

Storage as file system (HDFS)

HDFS: Text files

Billions of lines

```
In the year 1878 I took my degree of Doctor of Medicine of the
University of London, and proceeded to Netley to go through
the course prescribed for surgeons in the army. Having
completed my studies there, I was duly attached to the Fifth
Northumberland Fusiliers as Assistant Surgeon. The regiment
was stationed in India at the time, and before I could join
it, the second Afghan war had broken out. On landing at
Bombay, I learned that my corps had advanced through the
passes, and was already deep in the enemy's country. I
followed, however, with many other officers who were in the
same situation as myself, and succeeded in reaching Candahar
in safety, where I found my regiment, and at once entered upon
my new duties.
```

HDFS: CSV files

Billions of lines

```
Year,Date,Duration,Guest
2019,2019-04-08,00:17:44,Strassburg
2018,2018-04-16,00:20:31,BS
2017,2017-04-24,00:09:56,GL
2016,2016-04-18,00:43:34,LU
...
```

HDFS: "Raw" files

Billions of lines

```
20192019-04-0800:17:44FR
20182018-04-1600:20:31BS
20172017-04-2400:09:56GL
20162016-04-1800:43:34LU
...
```

HDFS: key-values

Billions of lines

```
2019 00:17:44.000000
2018 00:20:31.000000
2017 00:09:56.000000
2016 00:43:34.000000
...
```

HDFS: JSON Lines

Billions of lines

```
{ "year": 2019, "date": "2019-04-08", "duration": "00:17:44", "canton": "St" }
{ "year": 2018, "date": "2018-04-16", "duration": "00:20:31", "canton": "BS" }
{ "year": 2017, "date": "2017-04-24", "duration": "00:09:56", "canton": "GL" }
{ "year": 2016, "date": "2016-04-18", "duration": "00:43:34", "canton": "LU" }
...
...
{ "year": 1923, "duration": null }
```

HDFS: XML Lines

Billions of lines

```
<occurrence><year>2019</year><date>2019-04-08</date><duration>P17M44S</duration><canton>S</canton></occurrence>
<occurrence><year>2018</year><date>2018-04-16</date><duration>P20M31S</duration><canton>BS</canton></occurrence>
<occurrence><year>2017</year><date>2017-04-24</date><duration>P09M56S</duration><canton>GL</canton></occurrence>
<occurrence><year>2016</year><date>2016-04-18</date><duration>P43M34S</duration><canton>LU</canton></occurrence>
<occurrence><year>1923</year><duration xsi:nil="true"/></occurrence>
```

Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="occurrence">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="year" type="xs:integer"/>
    <xs:element name="date" type="xs:date" minOccurs="0"/>
    <xs:element name="duration" type="xs:duration" nillable="true"/>
    <xs:element name="canton" type="xs:string" minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

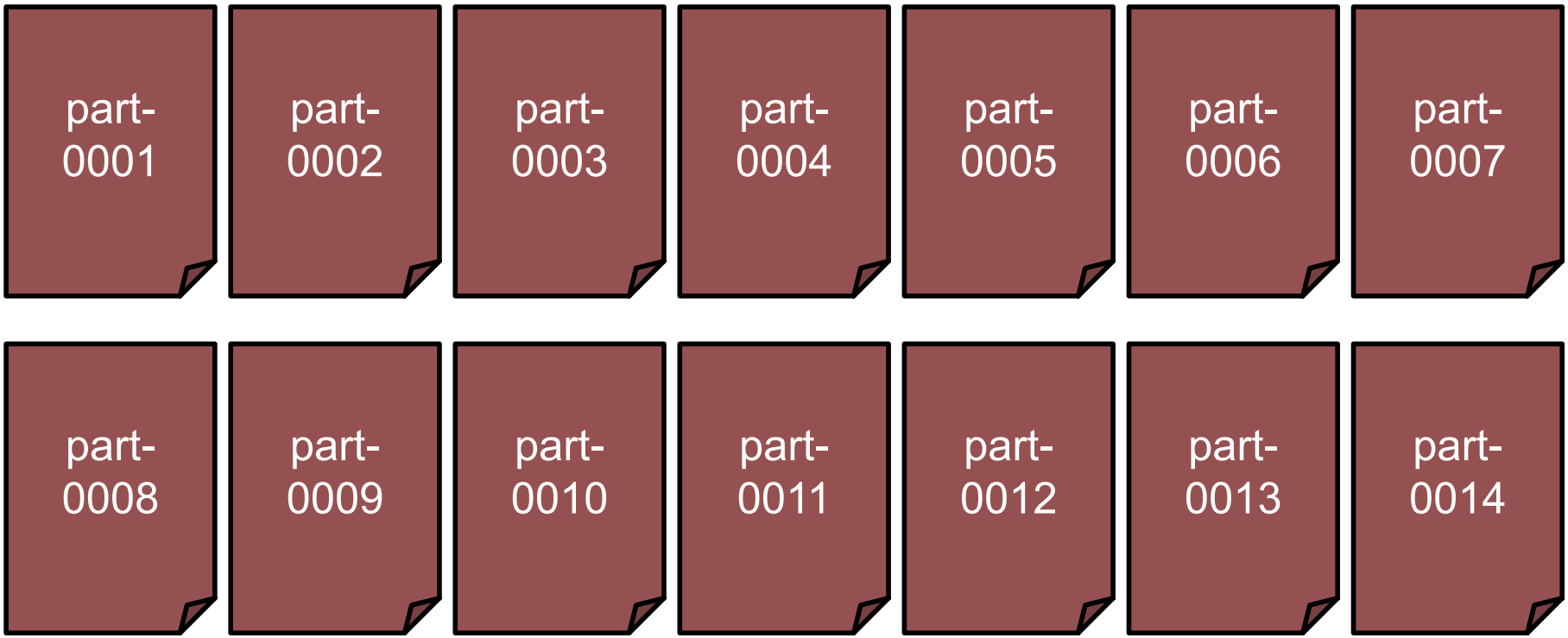
HDFS: Binary Files

Sequence files

Hadoop binary format

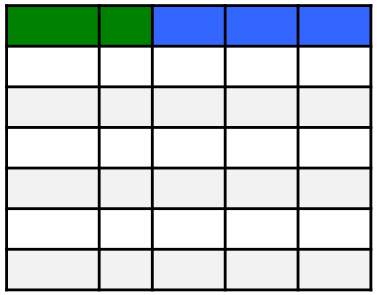
Stores generic key-values

HDFS: Common Scenario



Plenty of files (usually within the same directory)

So far, we have...



Storage as tables (HBase)

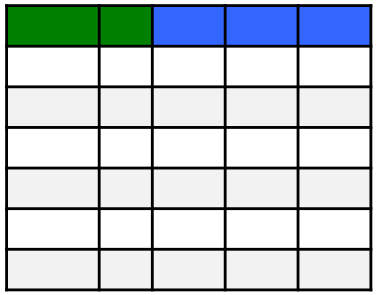
Storage as file system (HDFS)

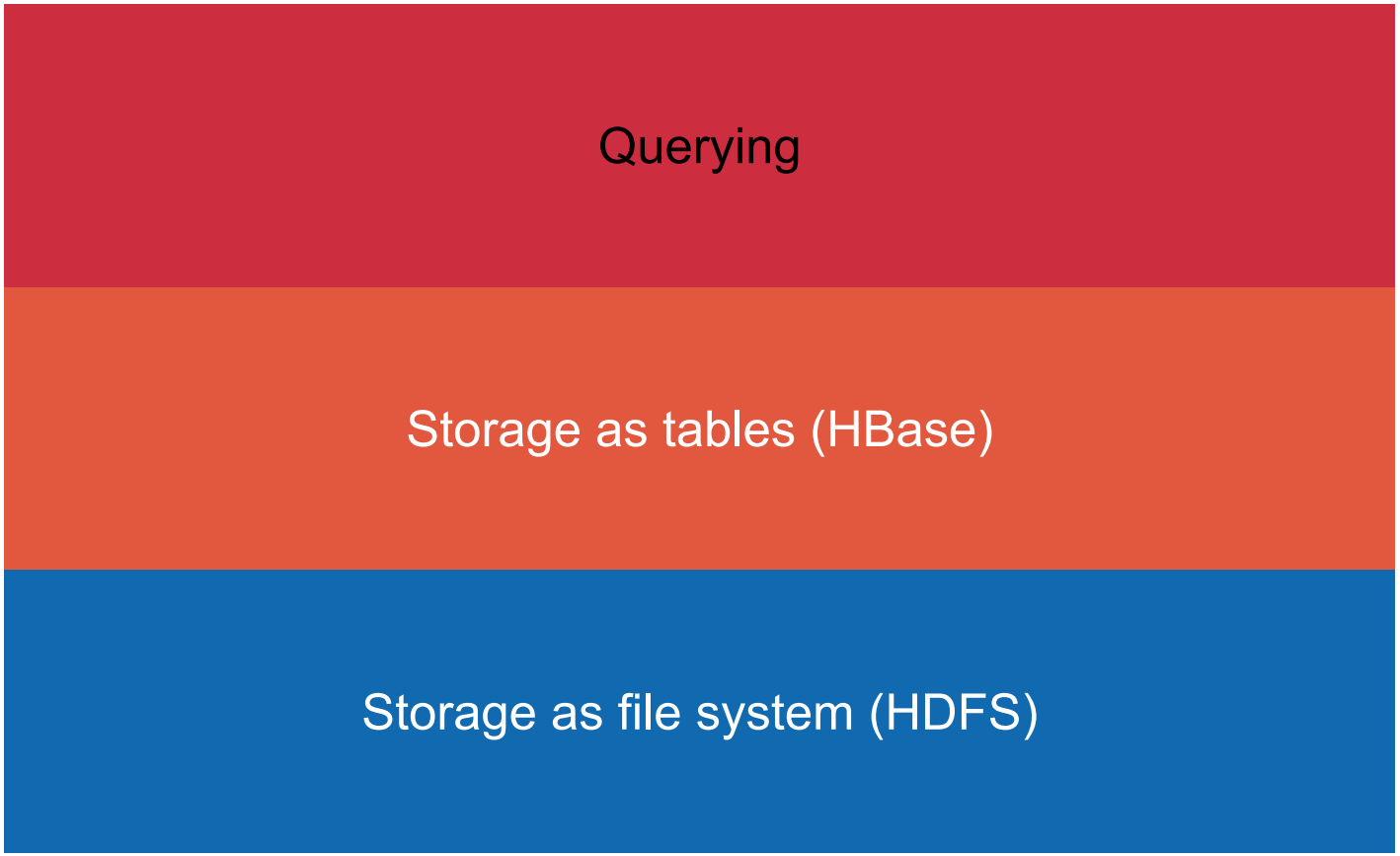
Tables

Billions of rows

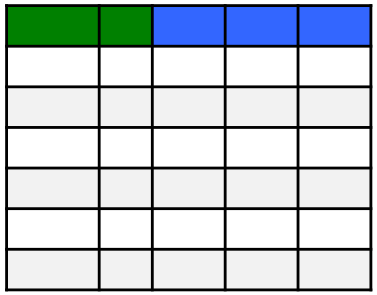
Year	Date	Duration	Canton
2019	2019-04-08	17:44	Strassburg
2018	2018-04-16	20:31	BS
2017	2017-04-24	09:56	GL
2016	2016-04-18	43:34	LU

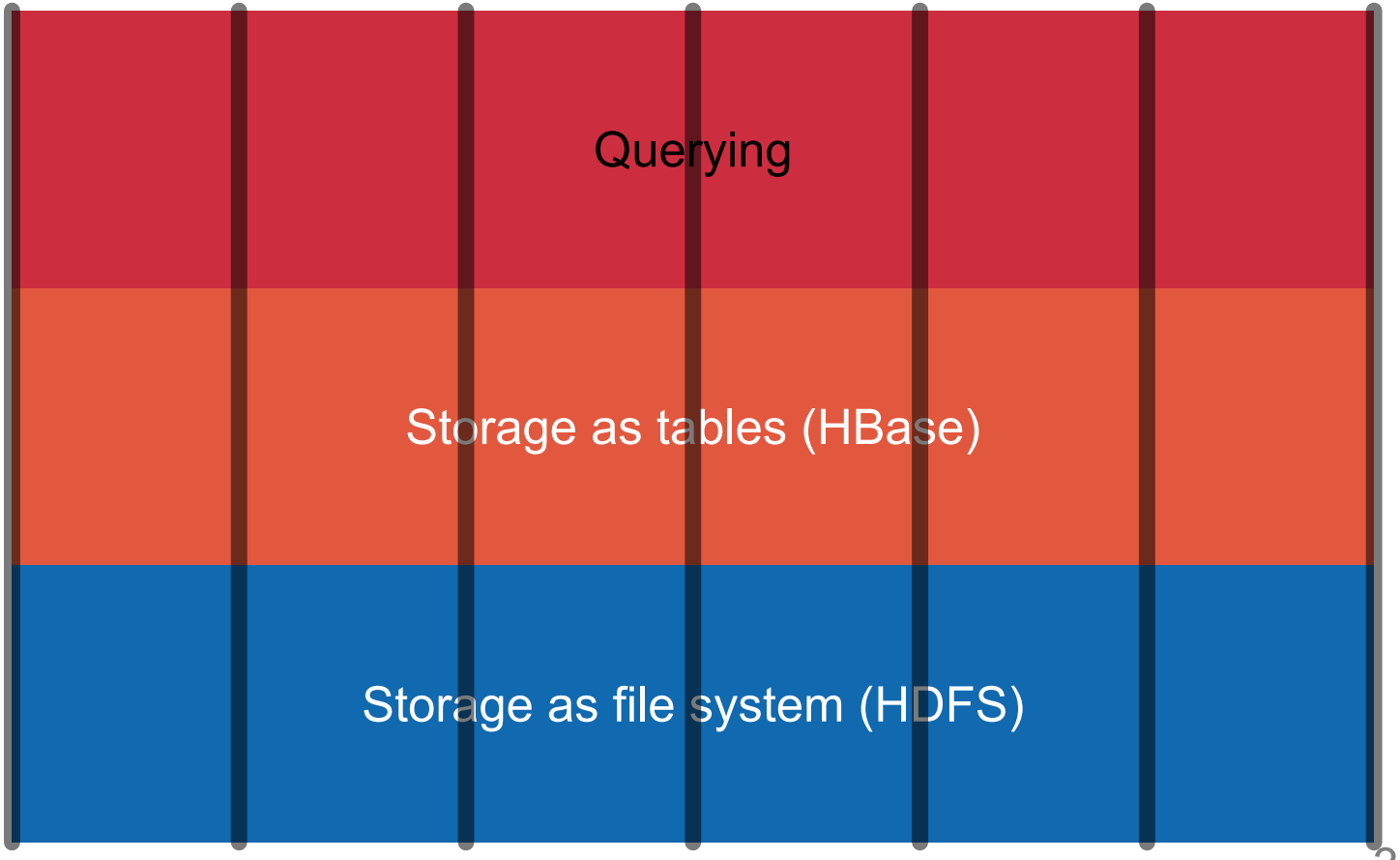
Data is only useful if we can query it





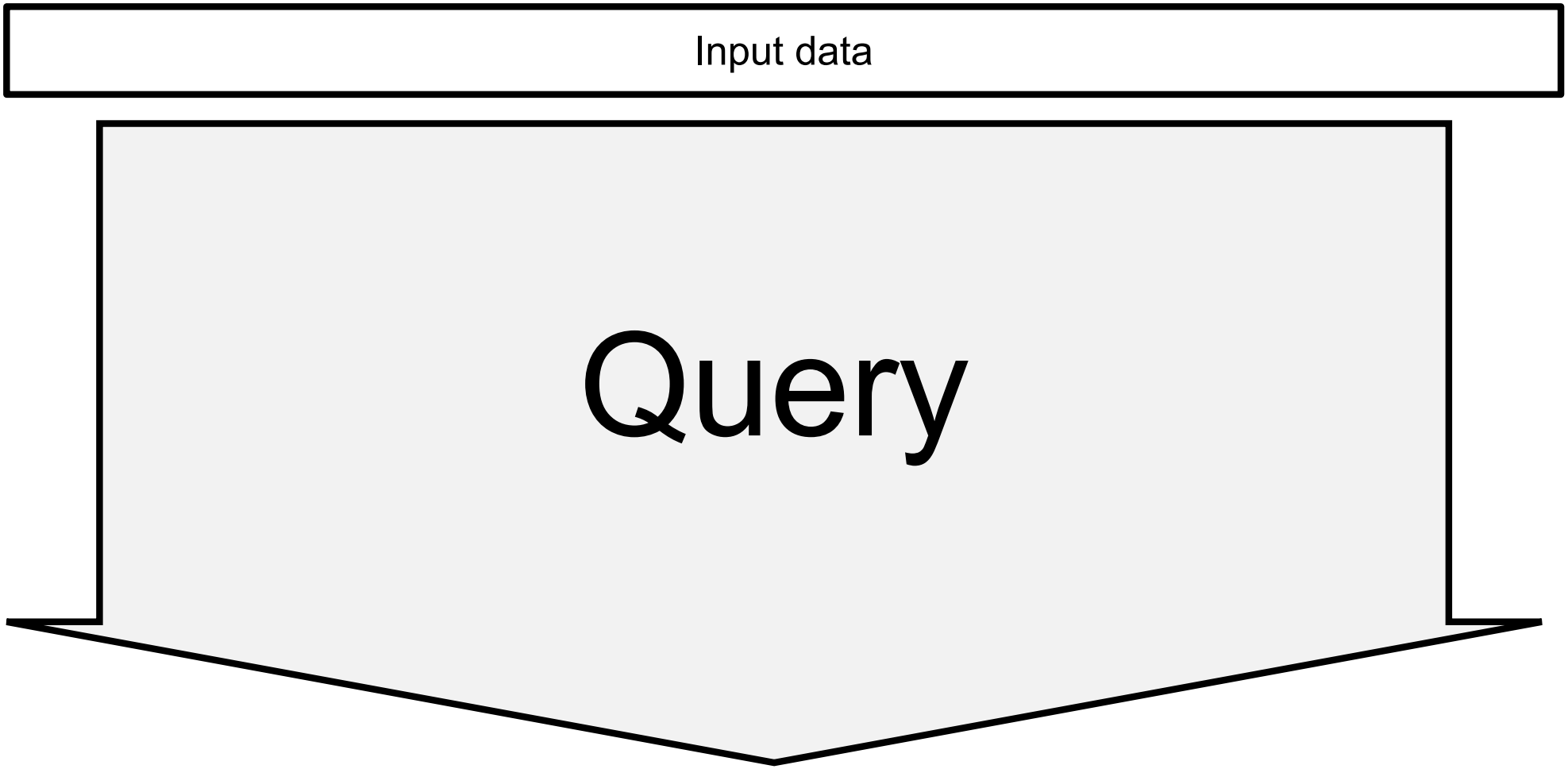
... in parallel



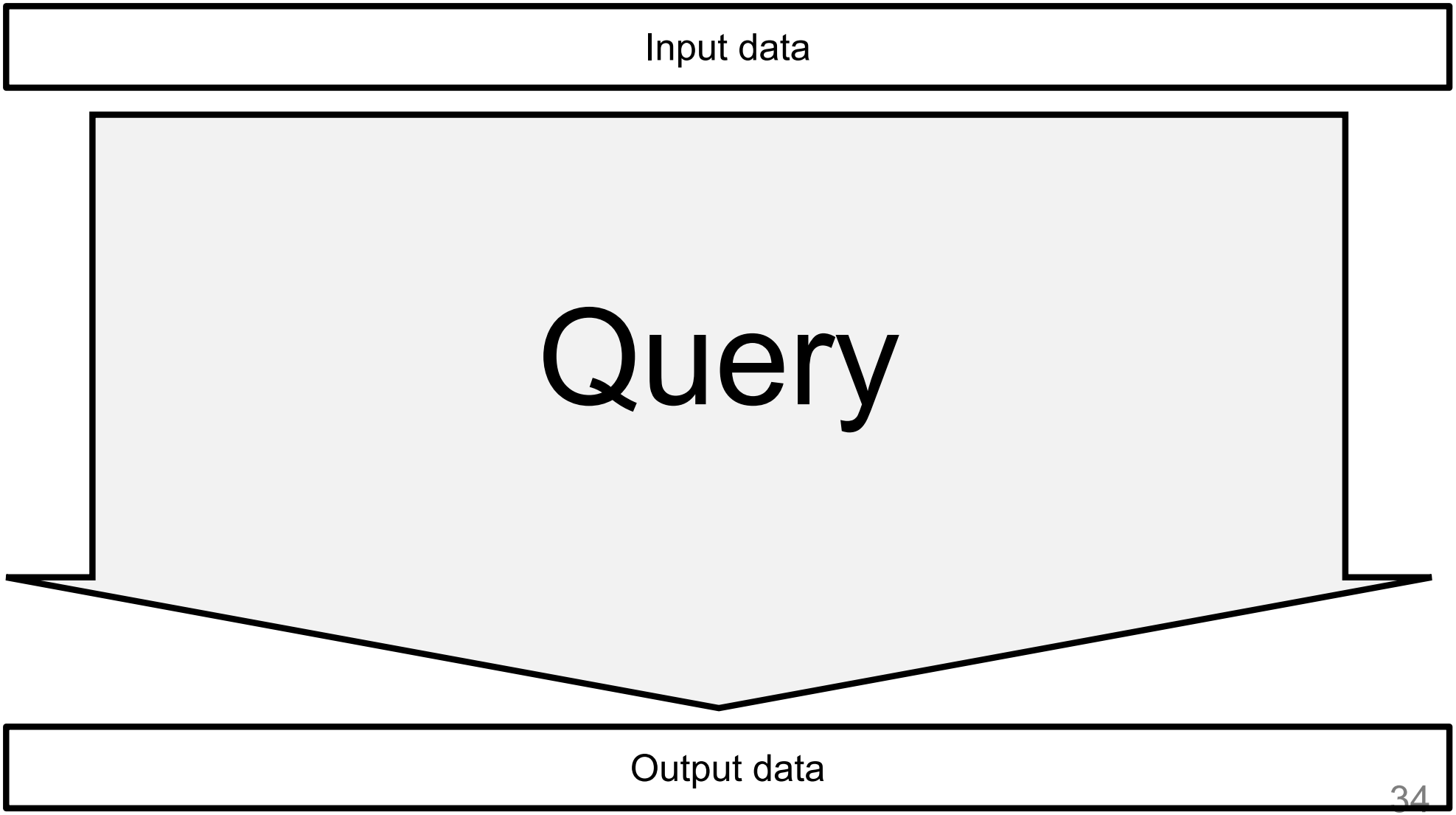

Querying
Storage as tables (HBase)
Storage as file system (HDFS)

Data Processing

Data Processing

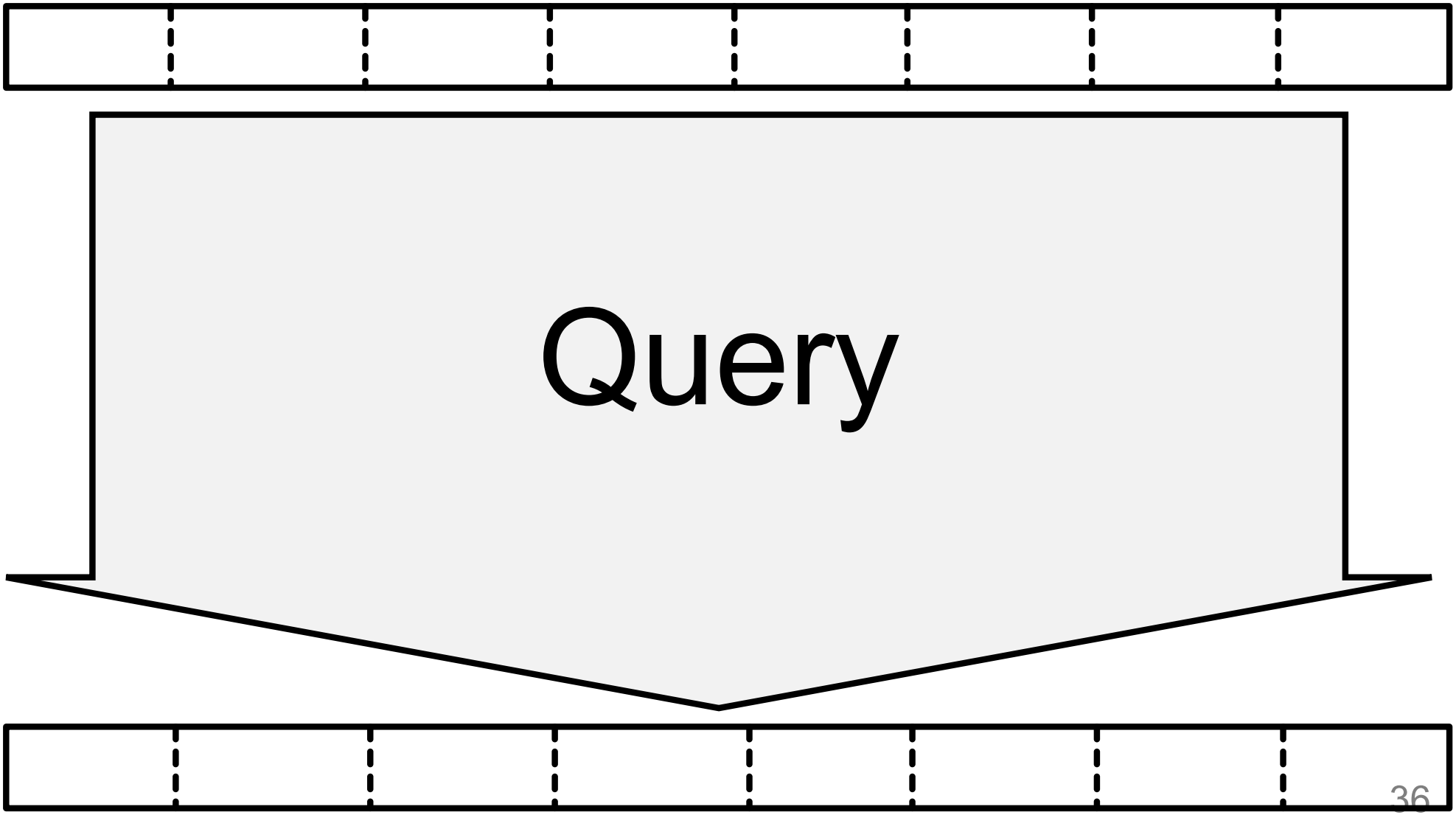


Data Processing

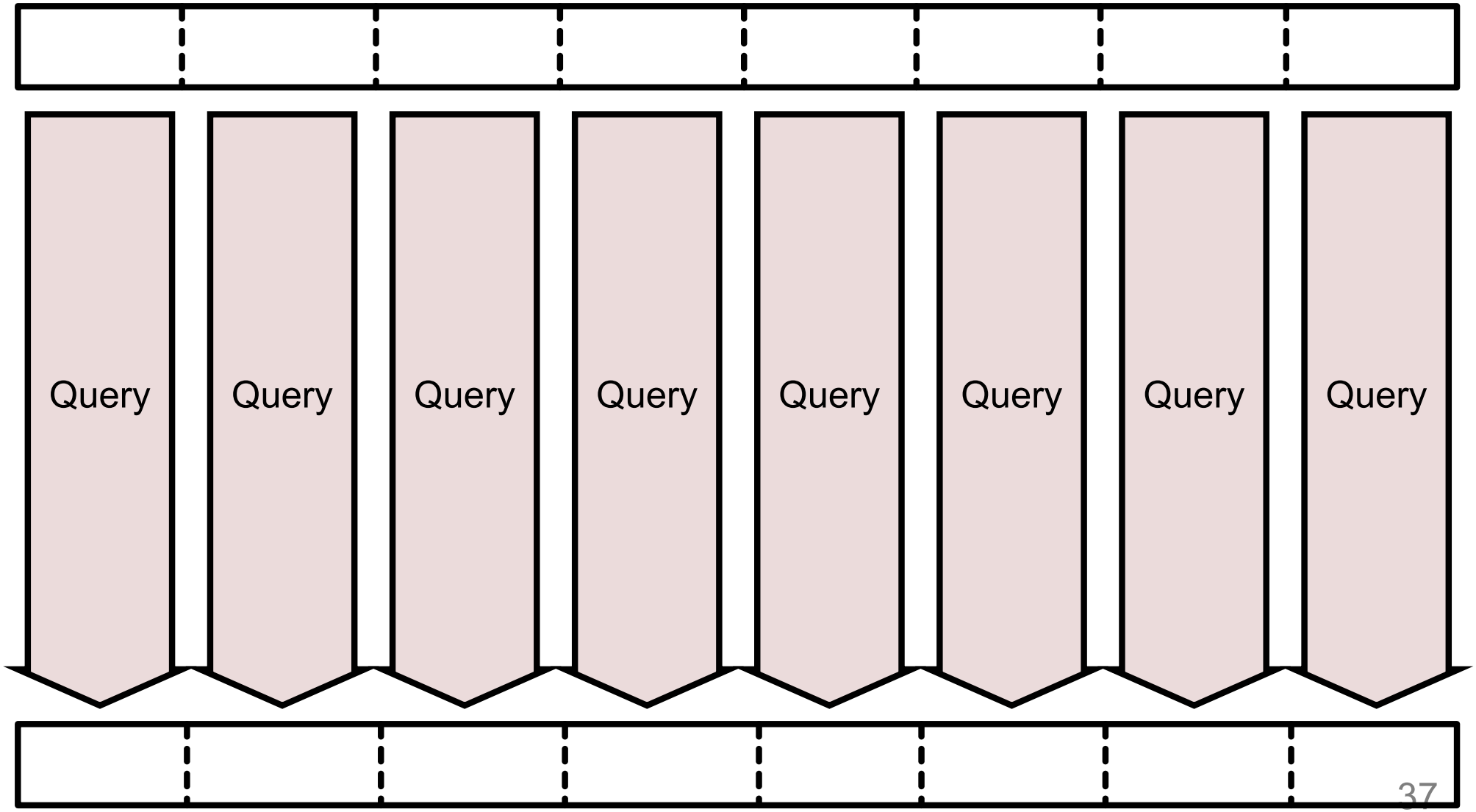


MapReduce

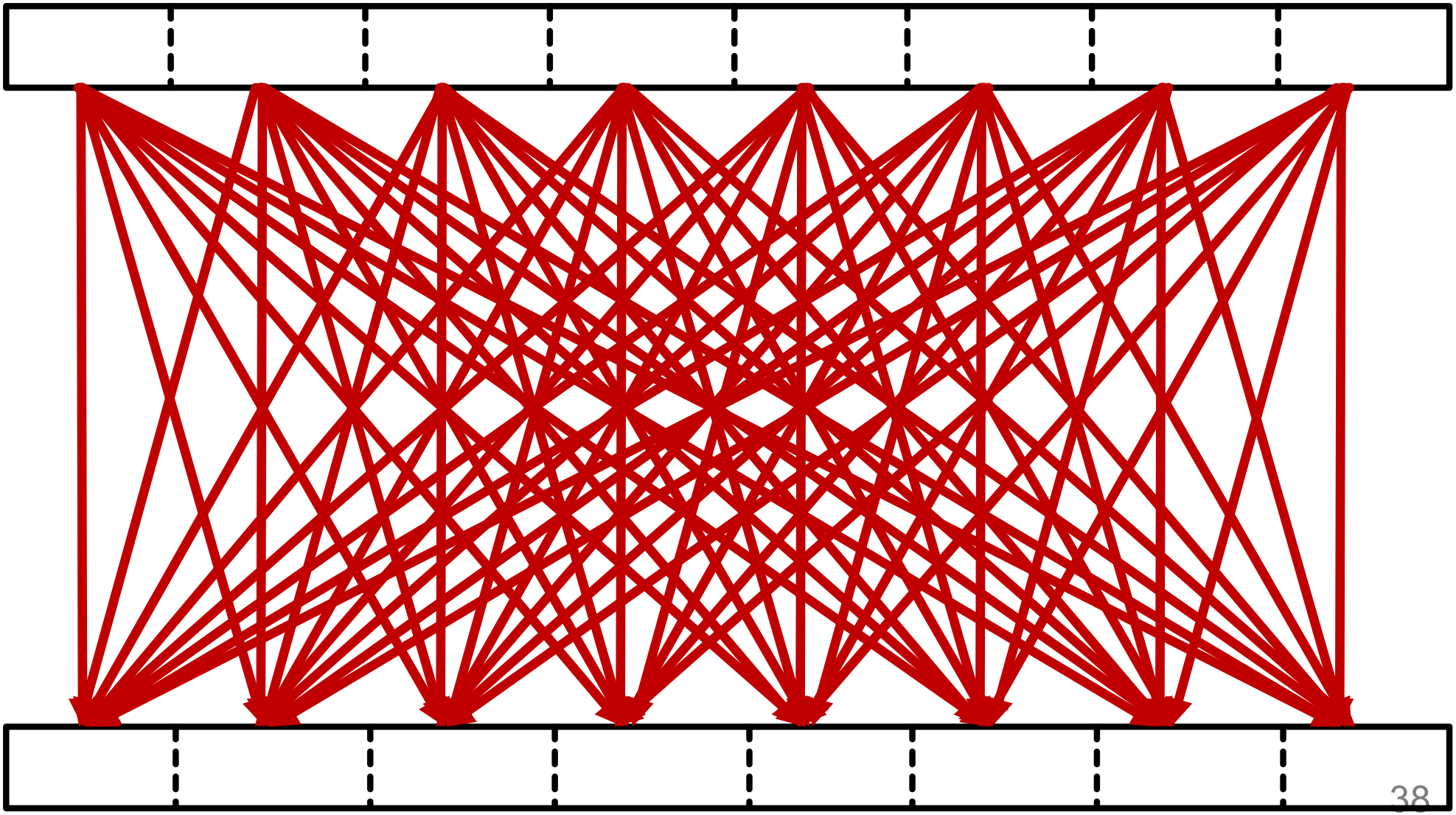
Data Processing: data comes in chunks



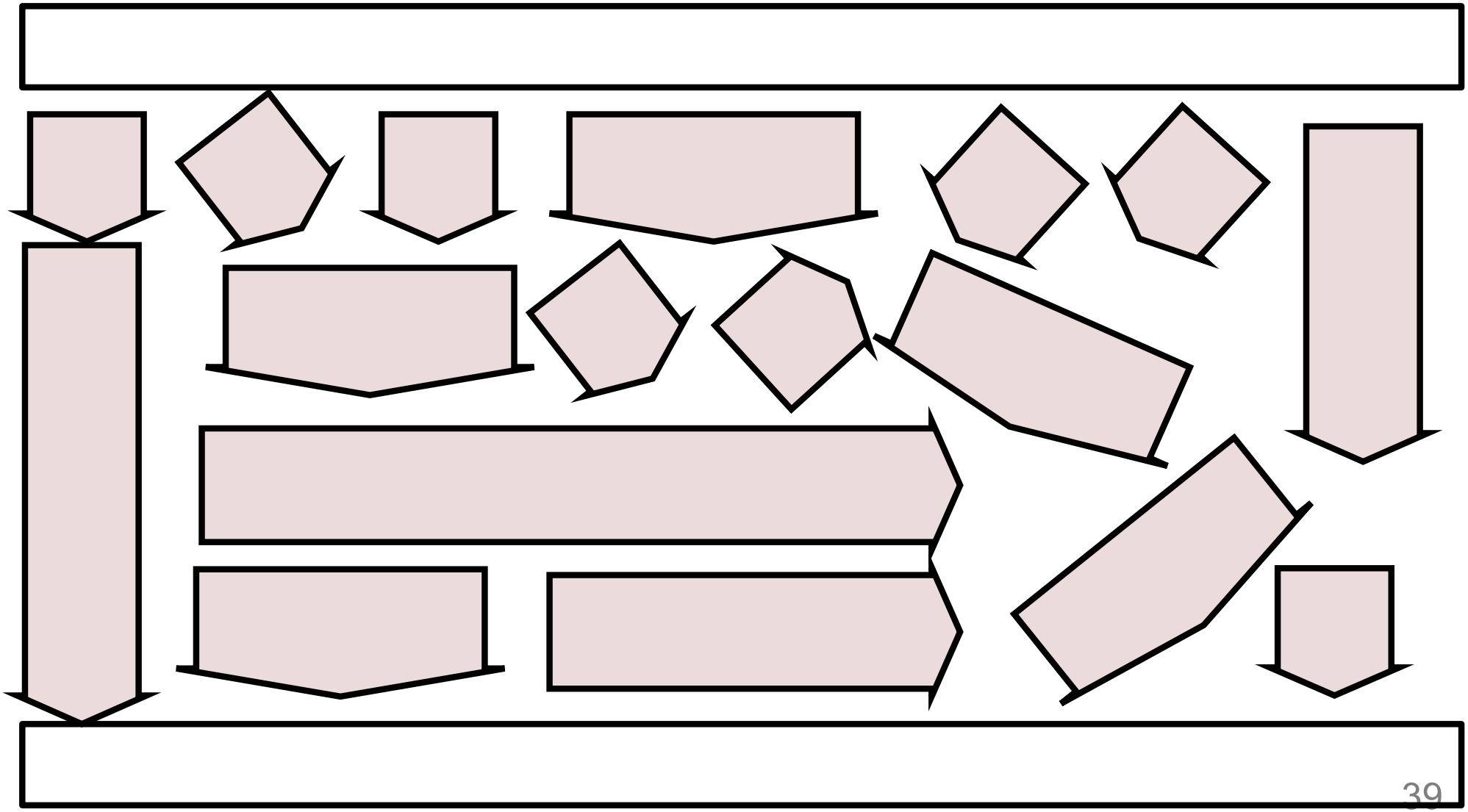
Data Processing: the ideal case



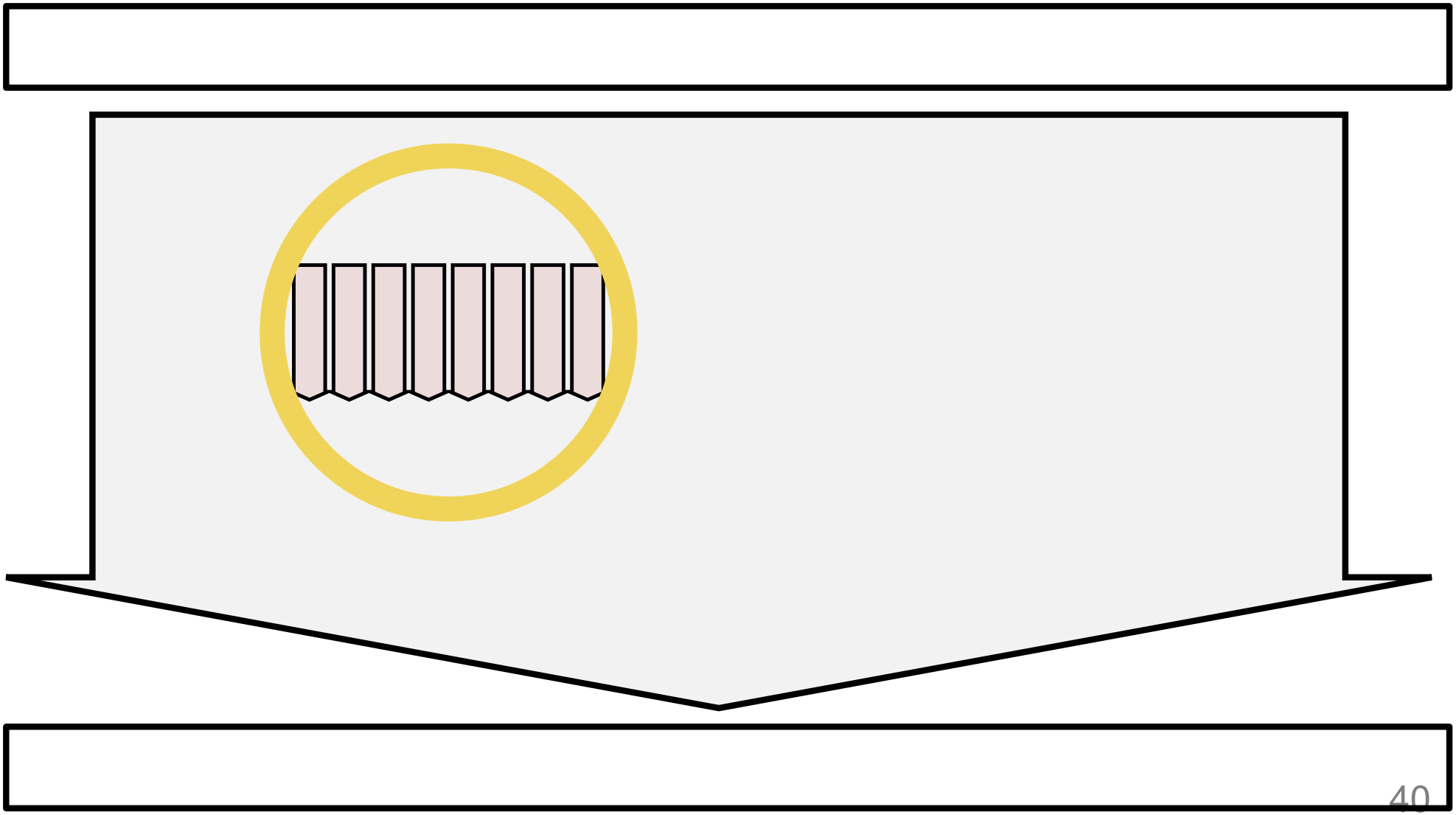
Data Processing: the worst case



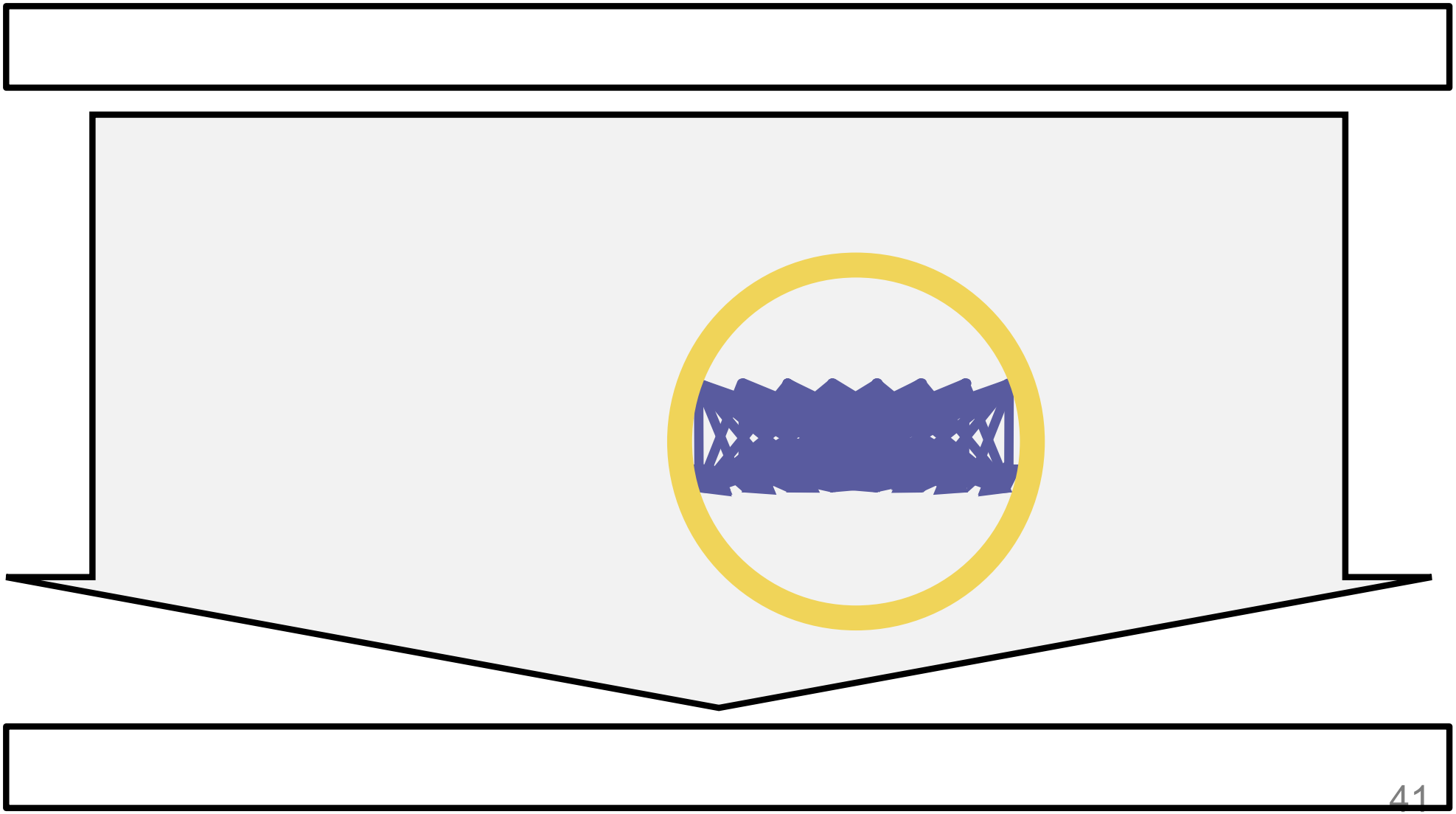
Data Processing: the typical case



Data Processing: Map here...

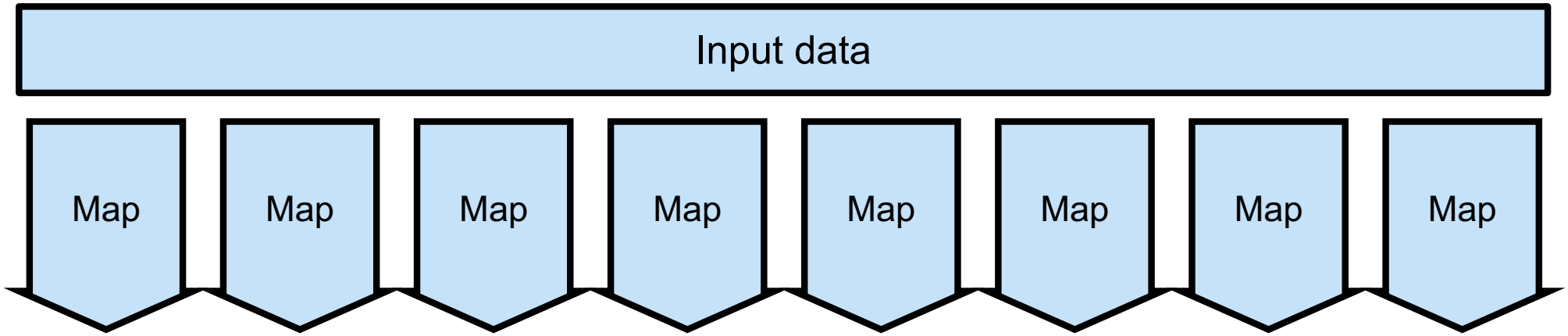


Data Processing: ... and shuffle there

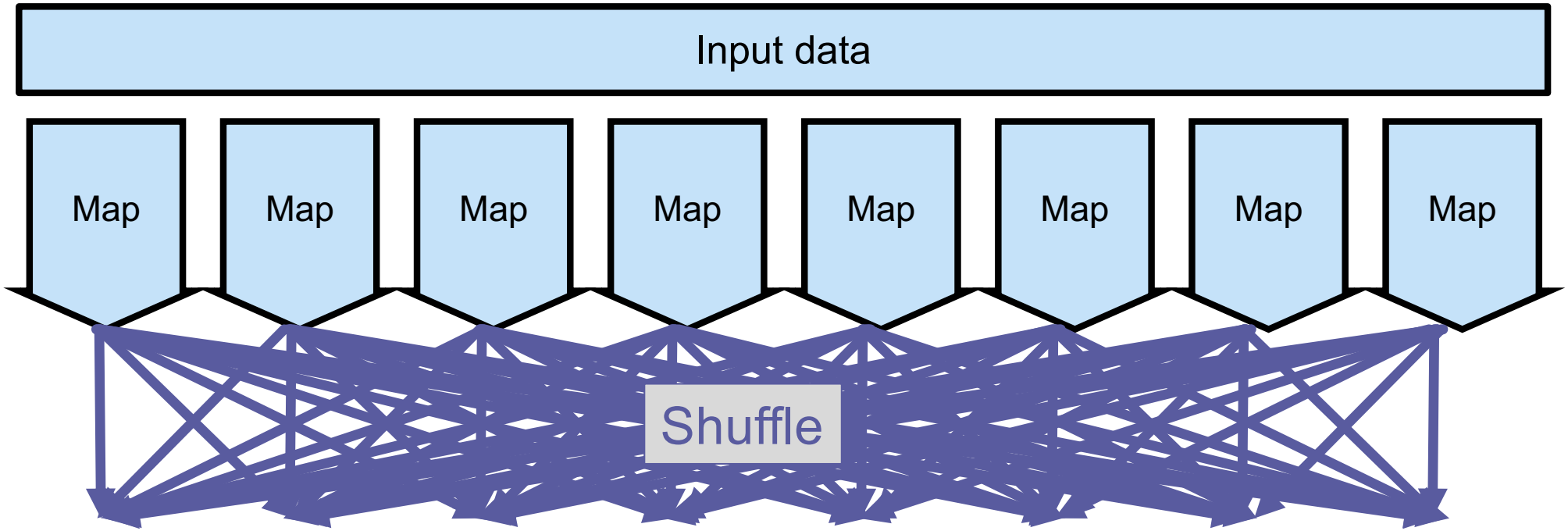


A common and useful sub-case: MapReduce

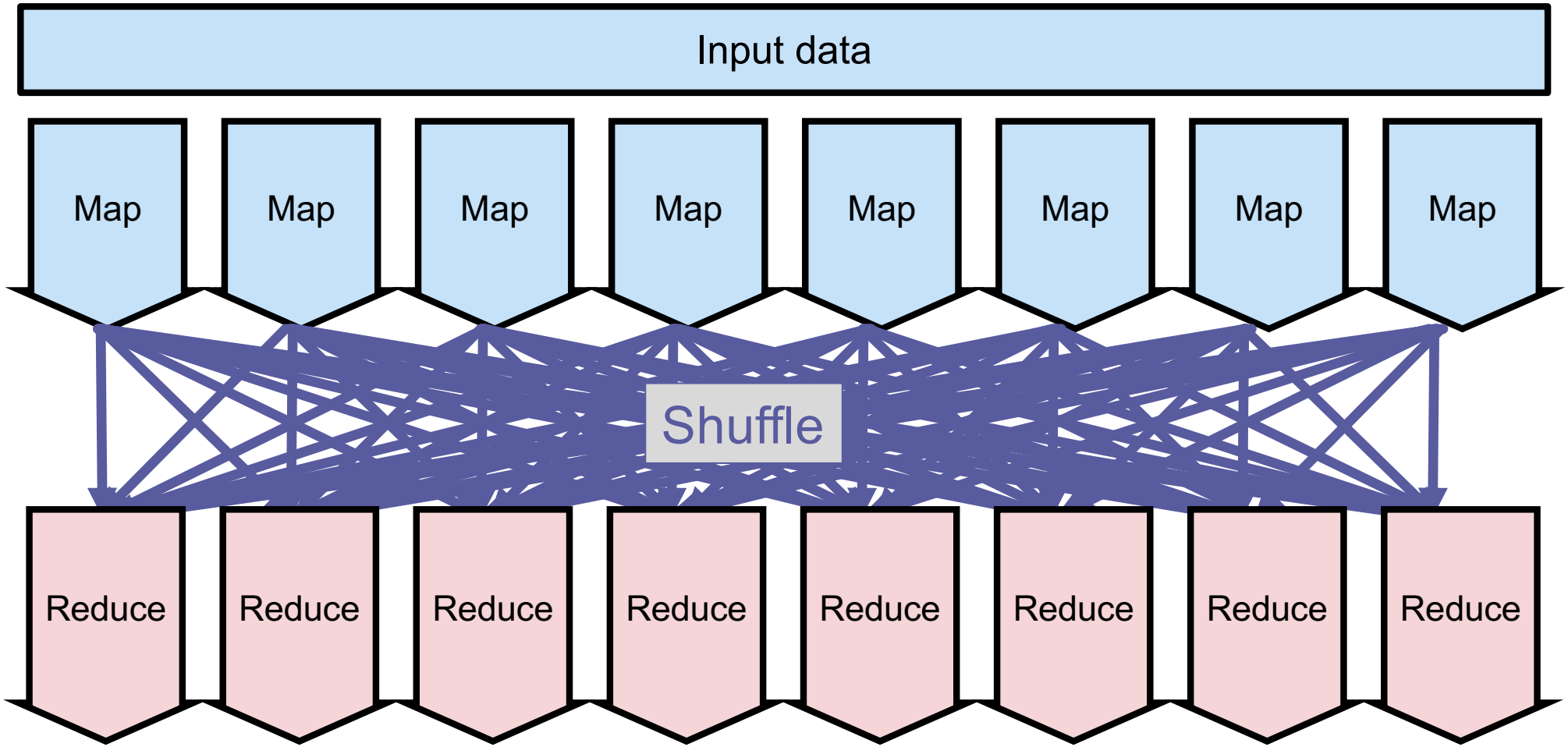
A common and useful sub-case: MapReduce



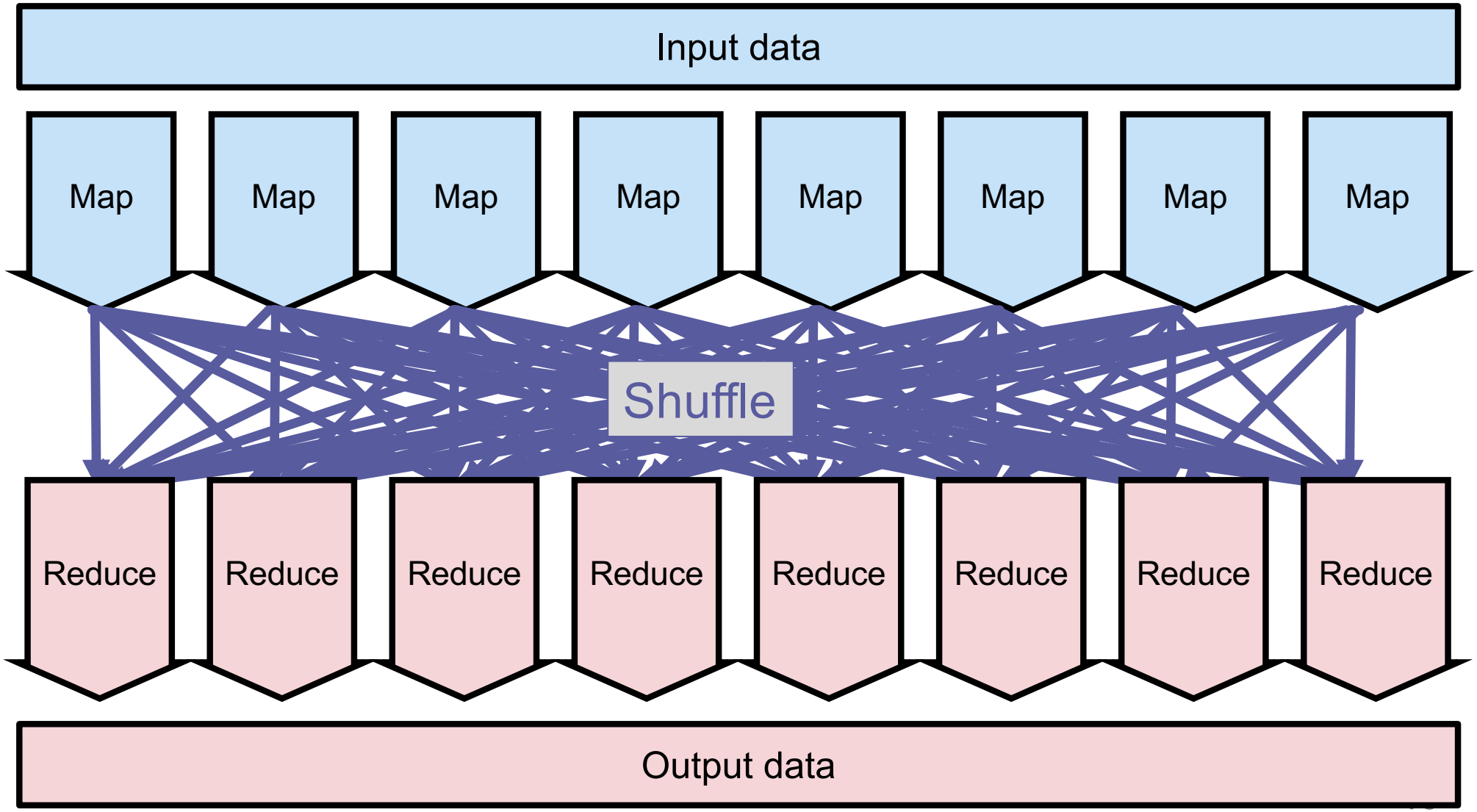
A common and useful sub-case: MapReduce



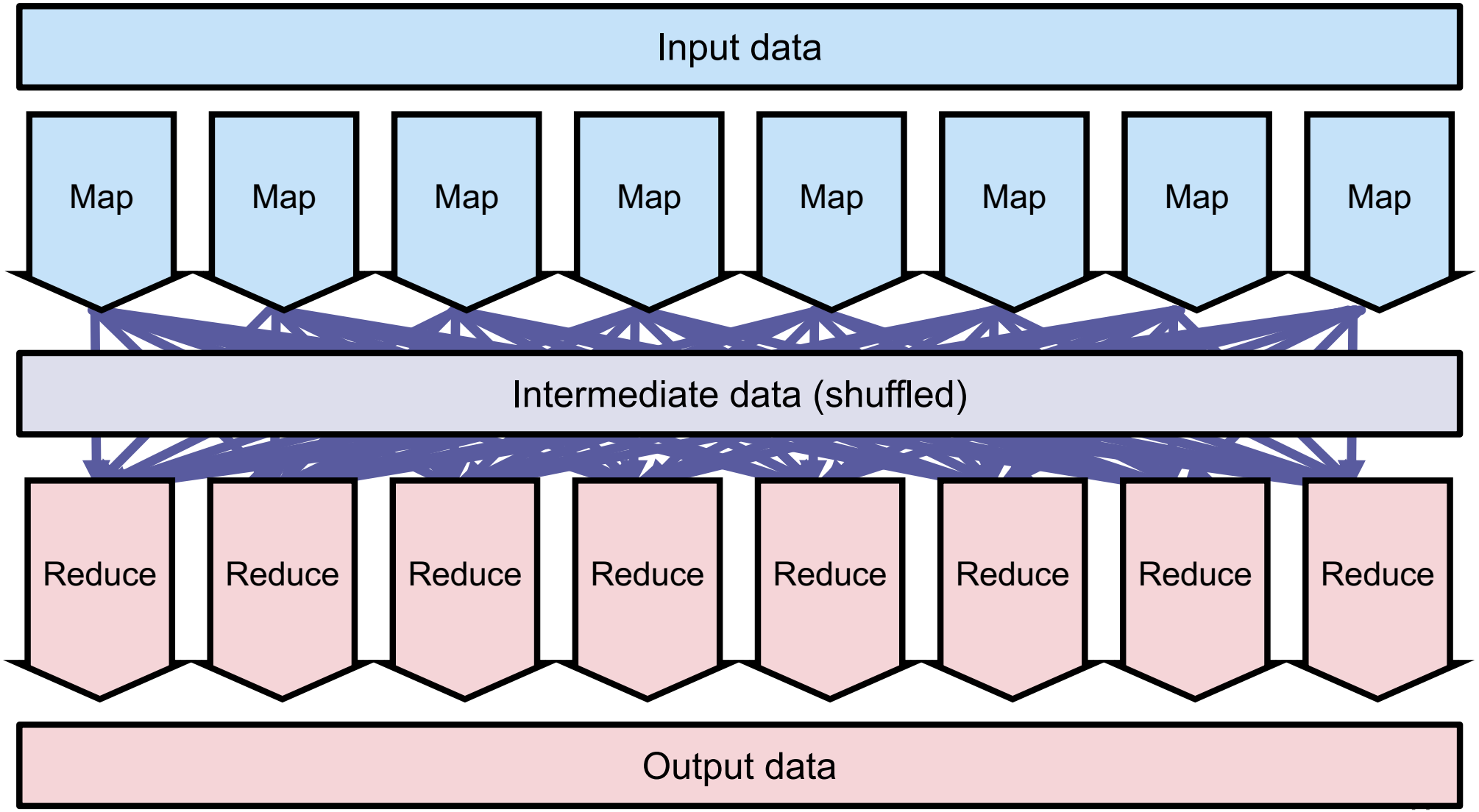
A common and useful sub-case: MapReduce



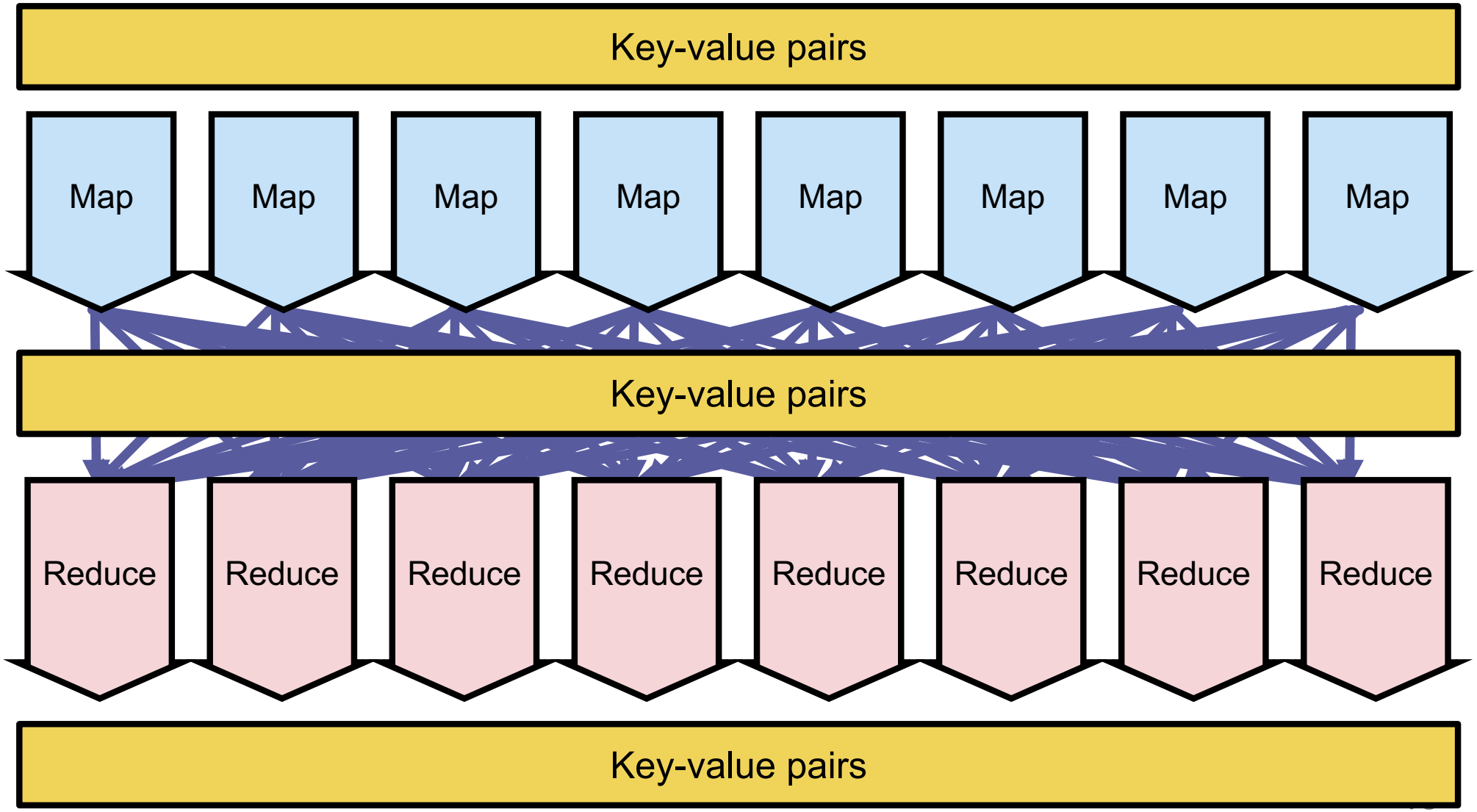
A common and useful sub-case: MapReduce



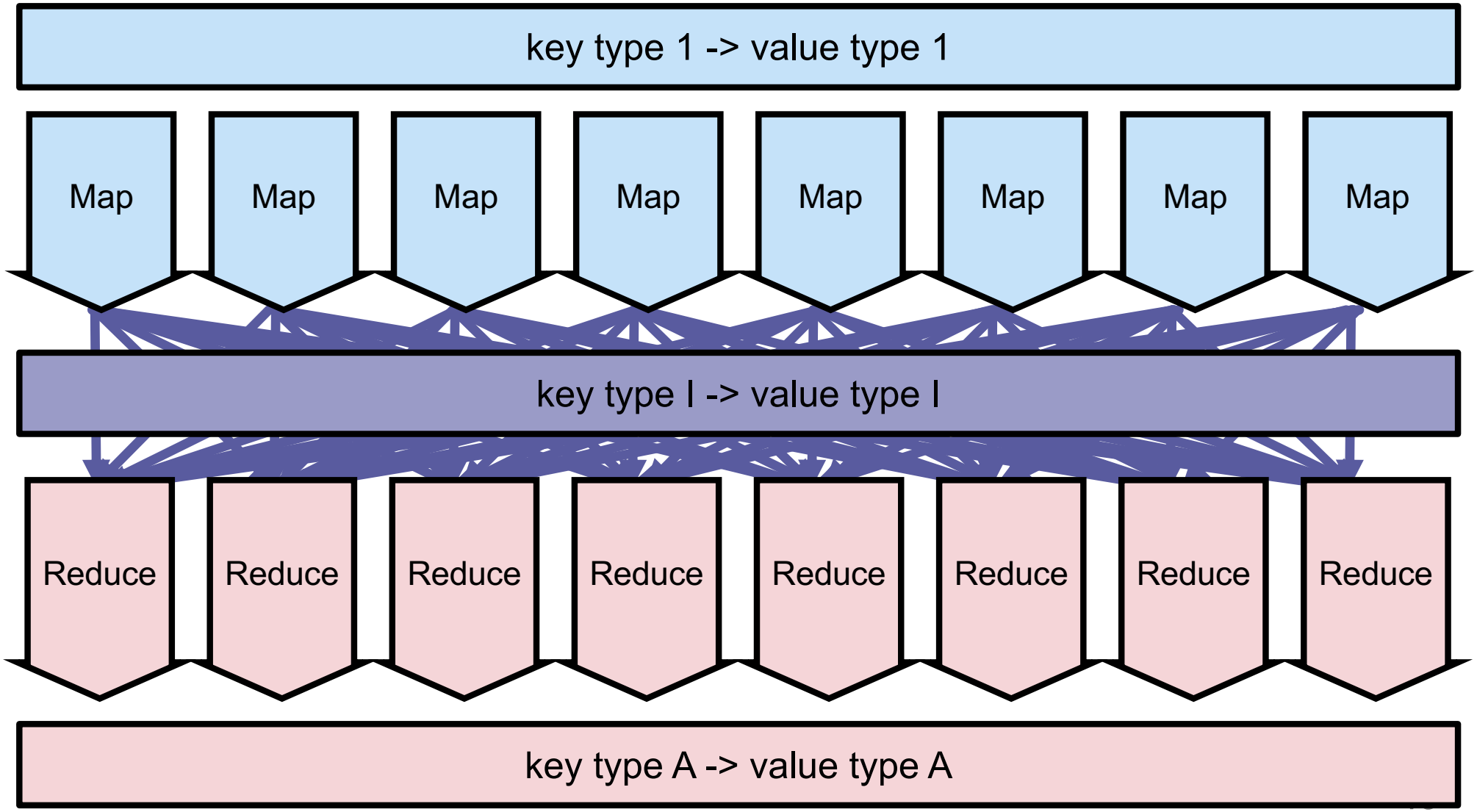
Data Processing: Data Model



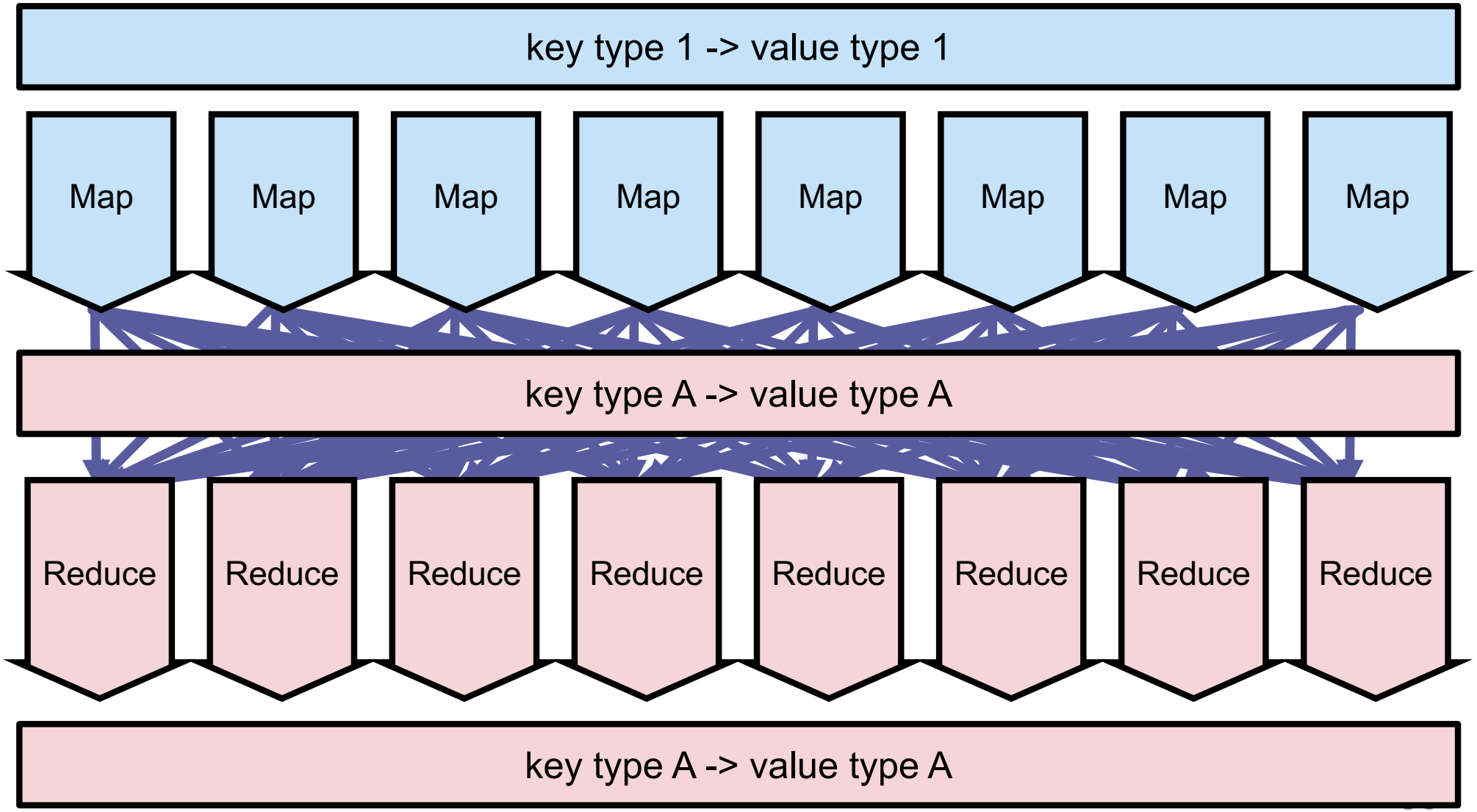
Data Processing: Data Shape



Data Processing: Data Types



Data Processing: Most often



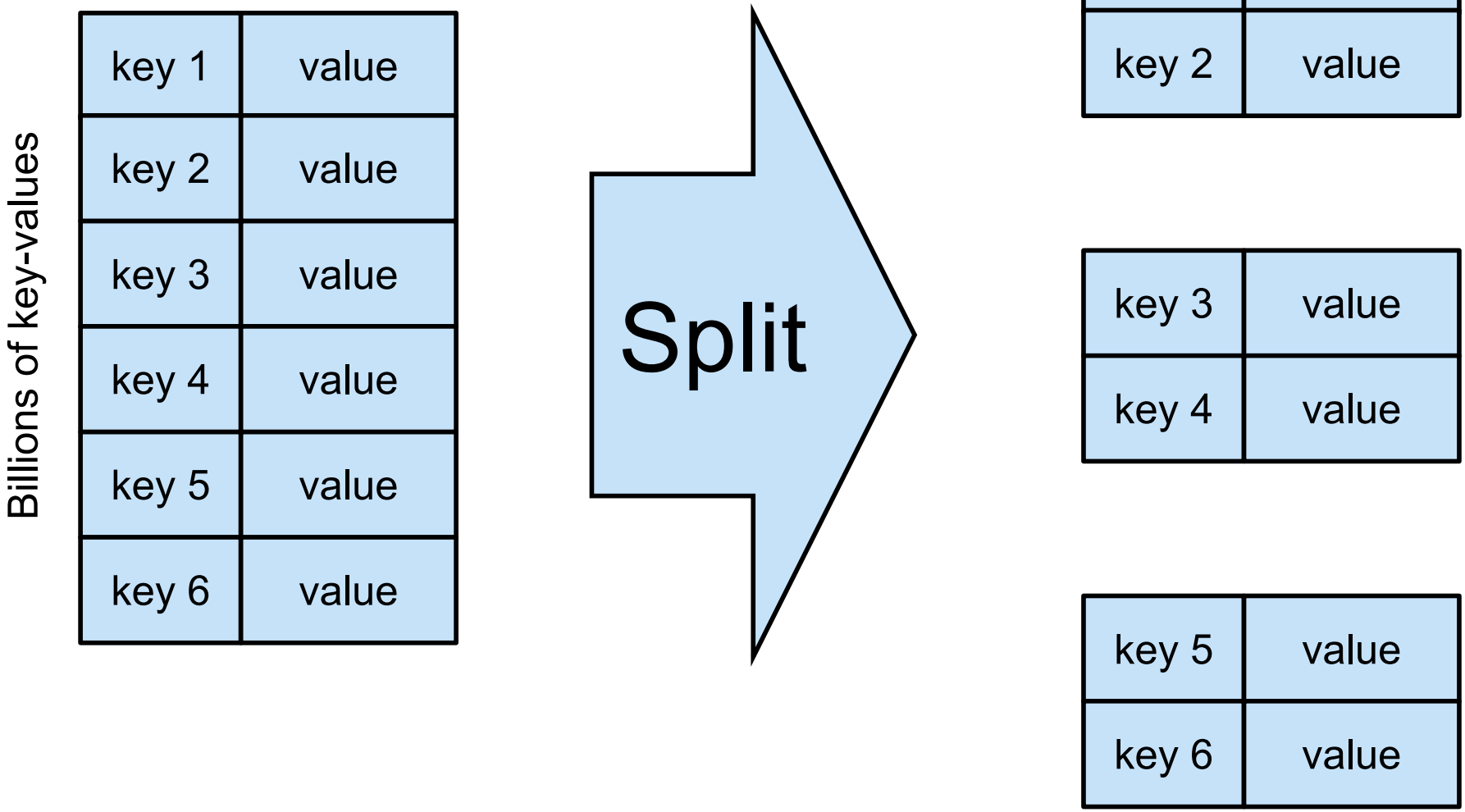
Logical Walkthrough

Splitting

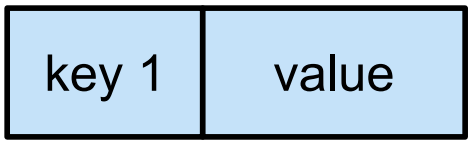
Billions of key-values

key 1	value
key 2	value
key 3	value
key 4	value
key 5	value
key 6	value

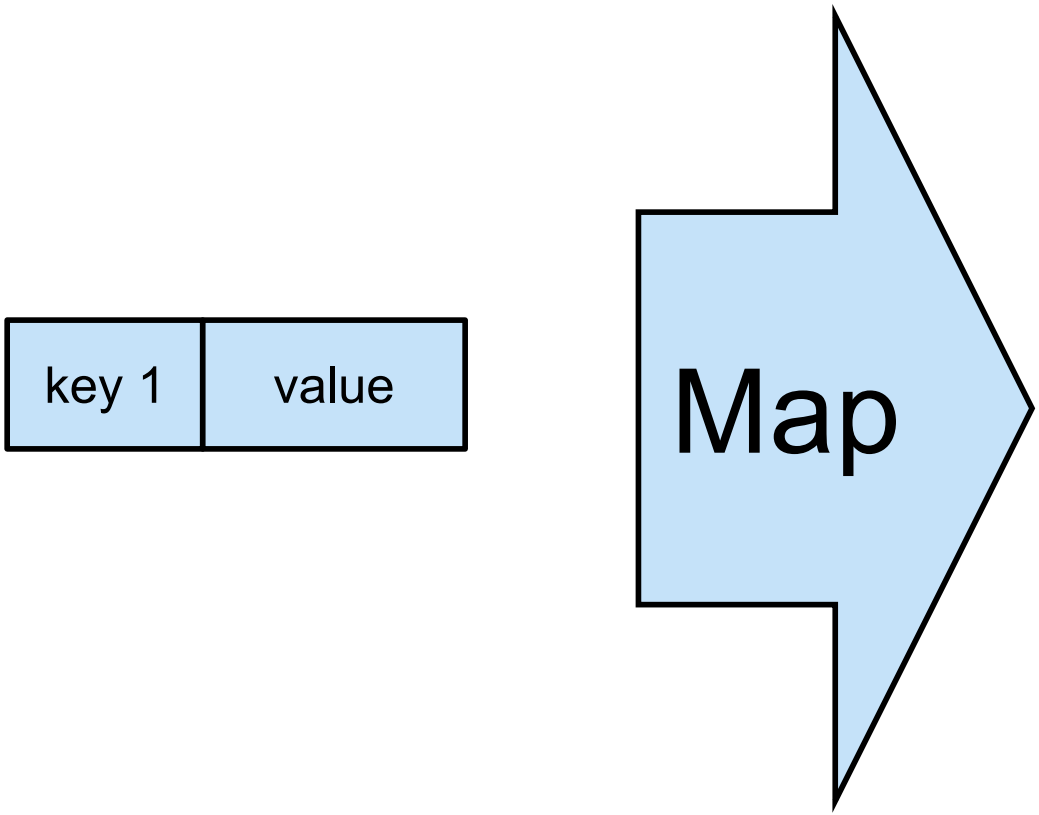
Splitting


Billions of key-values
key 1
value
key 2
value
key 3
value
key 4
value
key 5
value
key 6
value
Split
key 1
value
key 2
value
key 3
value
key 4
value
key 5
value
key 6
value

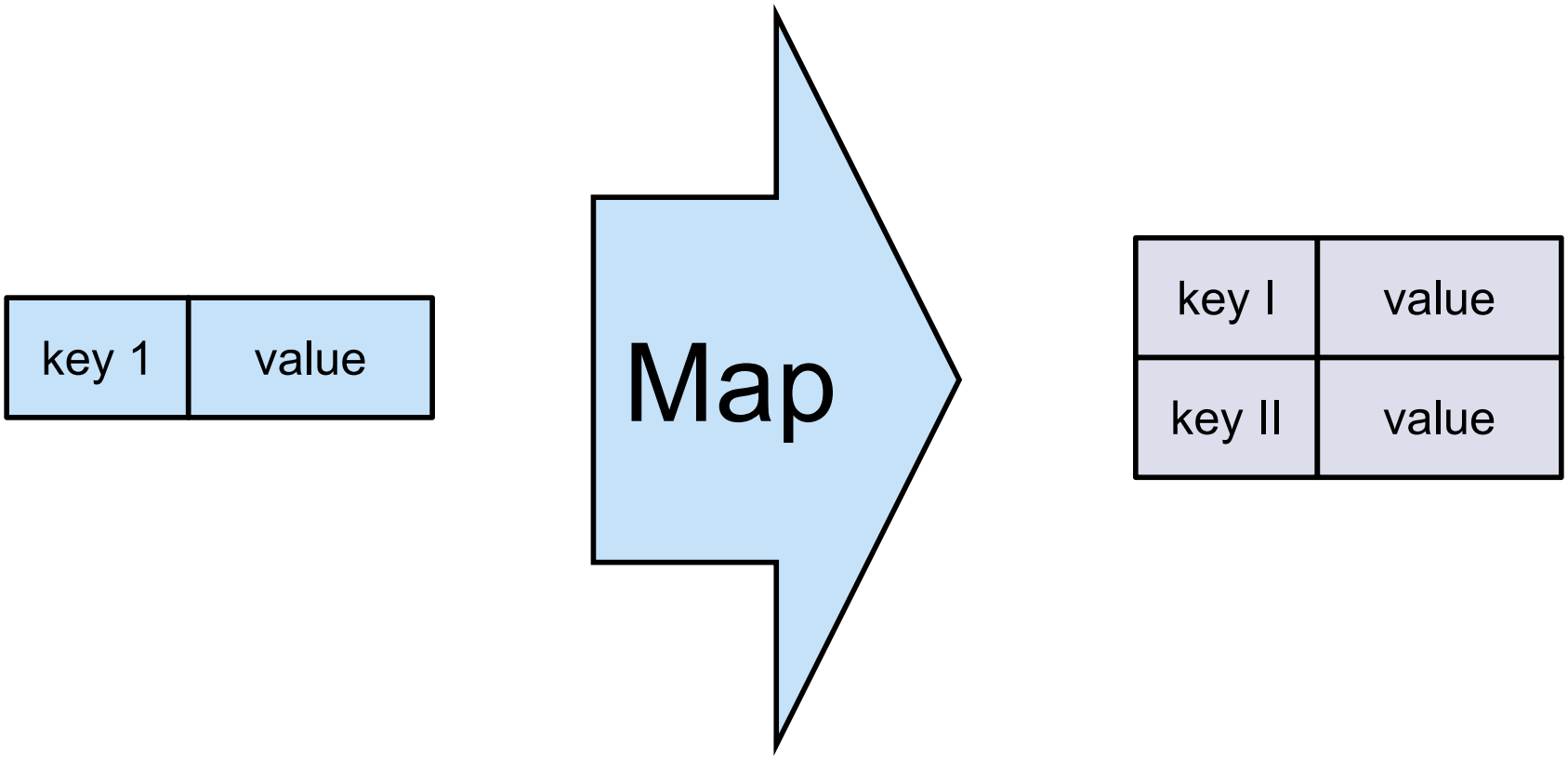
Mapping function



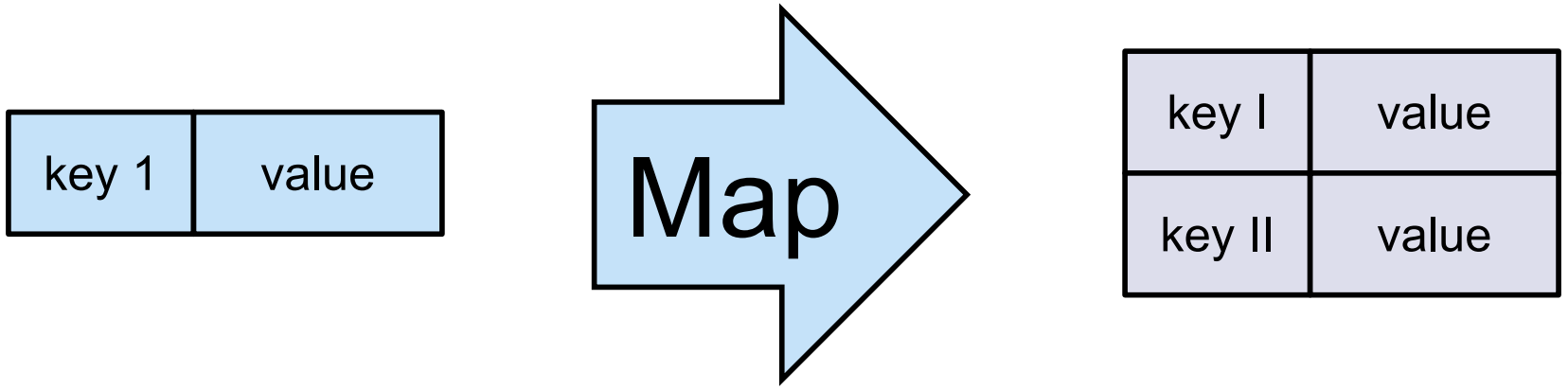
Mapping function



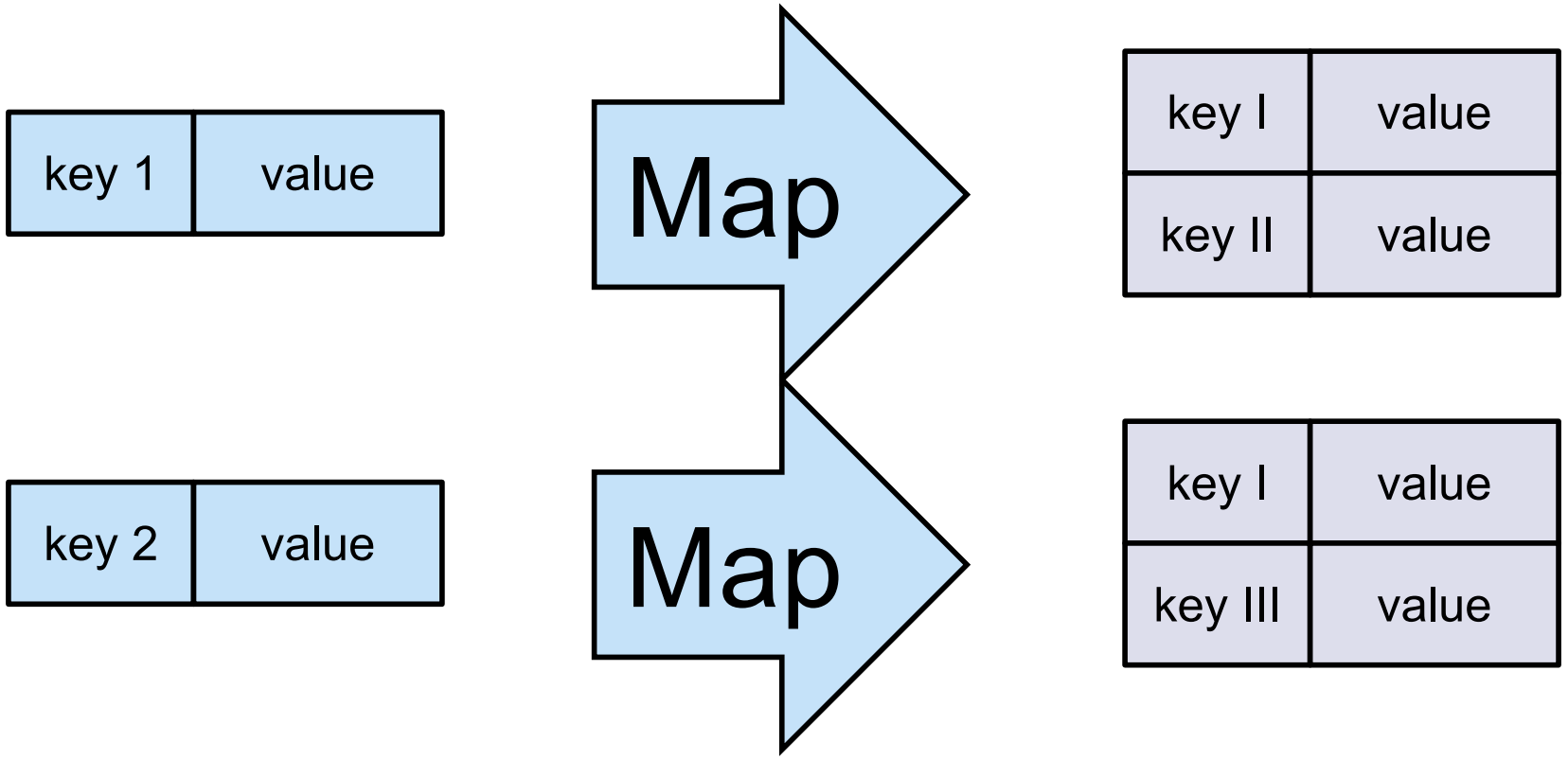
Mapping function



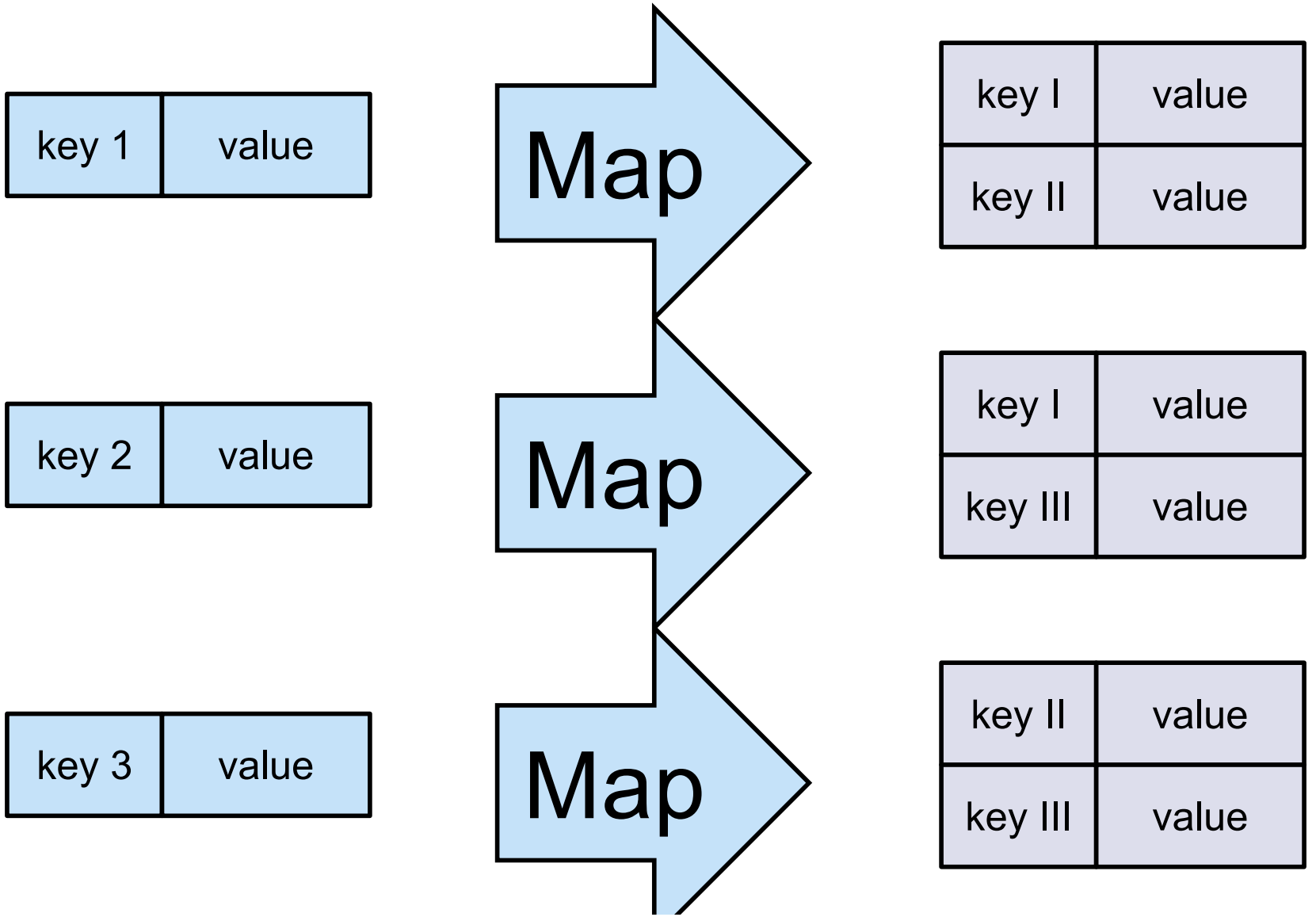
Mapping function... in parallel



Mapping function... in parallel



Mapping function... in parallel



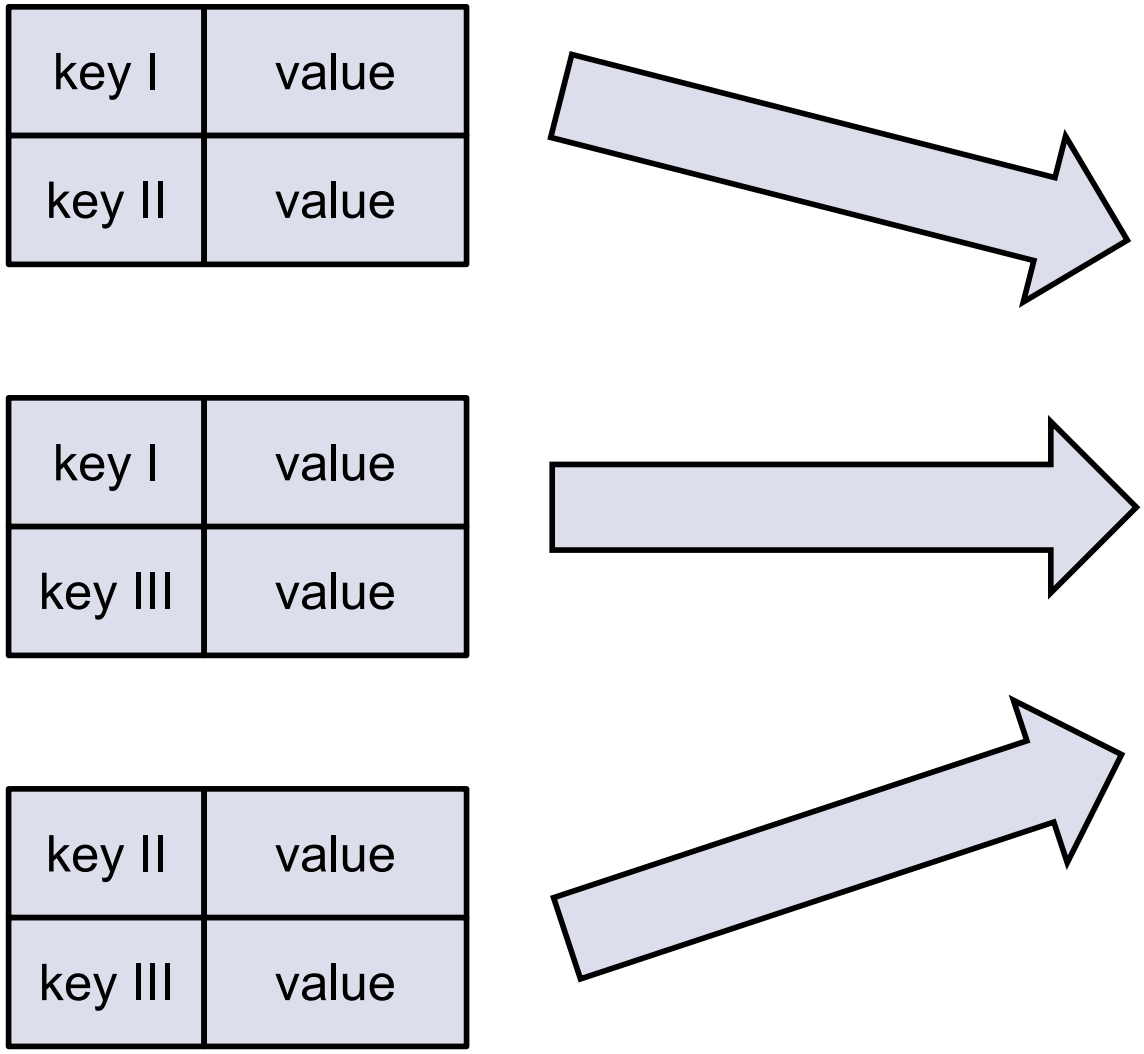
Put it all together

key I	value
key II	value

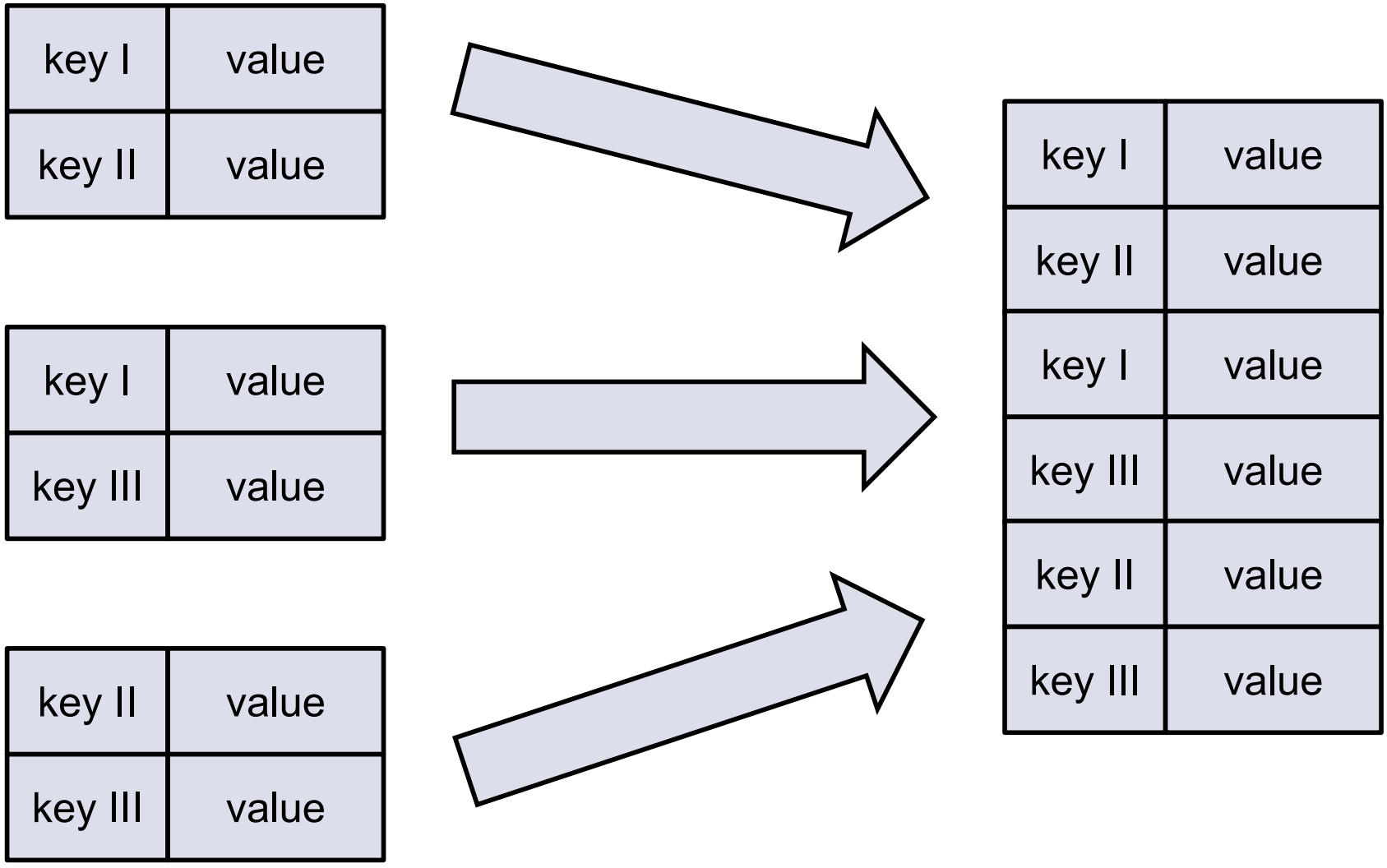
key I	value
key III	value

key II	value
key III	value

Put it all together



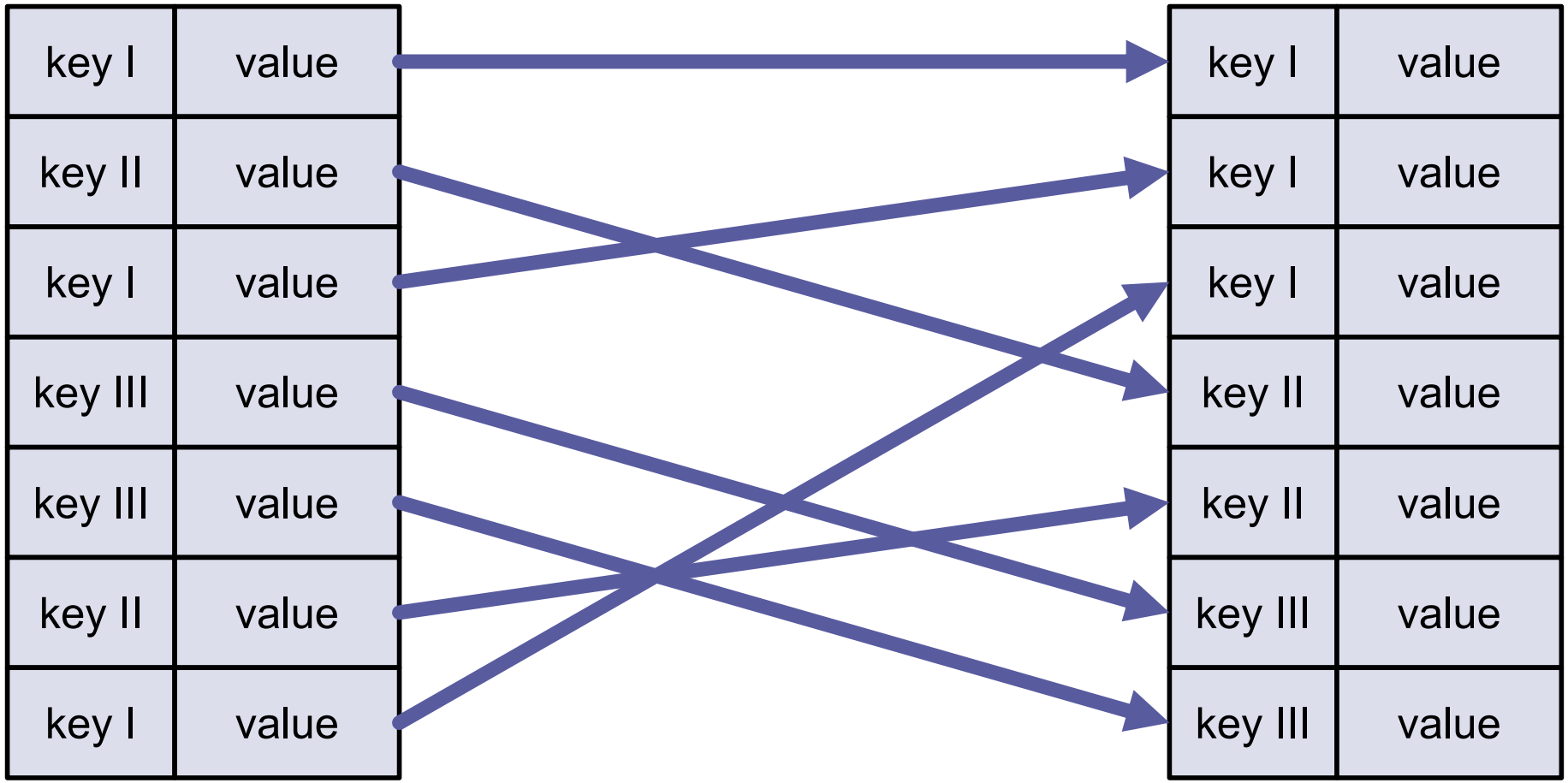
Put it all together



Sort by key

key I	value
key II	value
key I	value
key III	value
key III	value
key II	value
key I	value

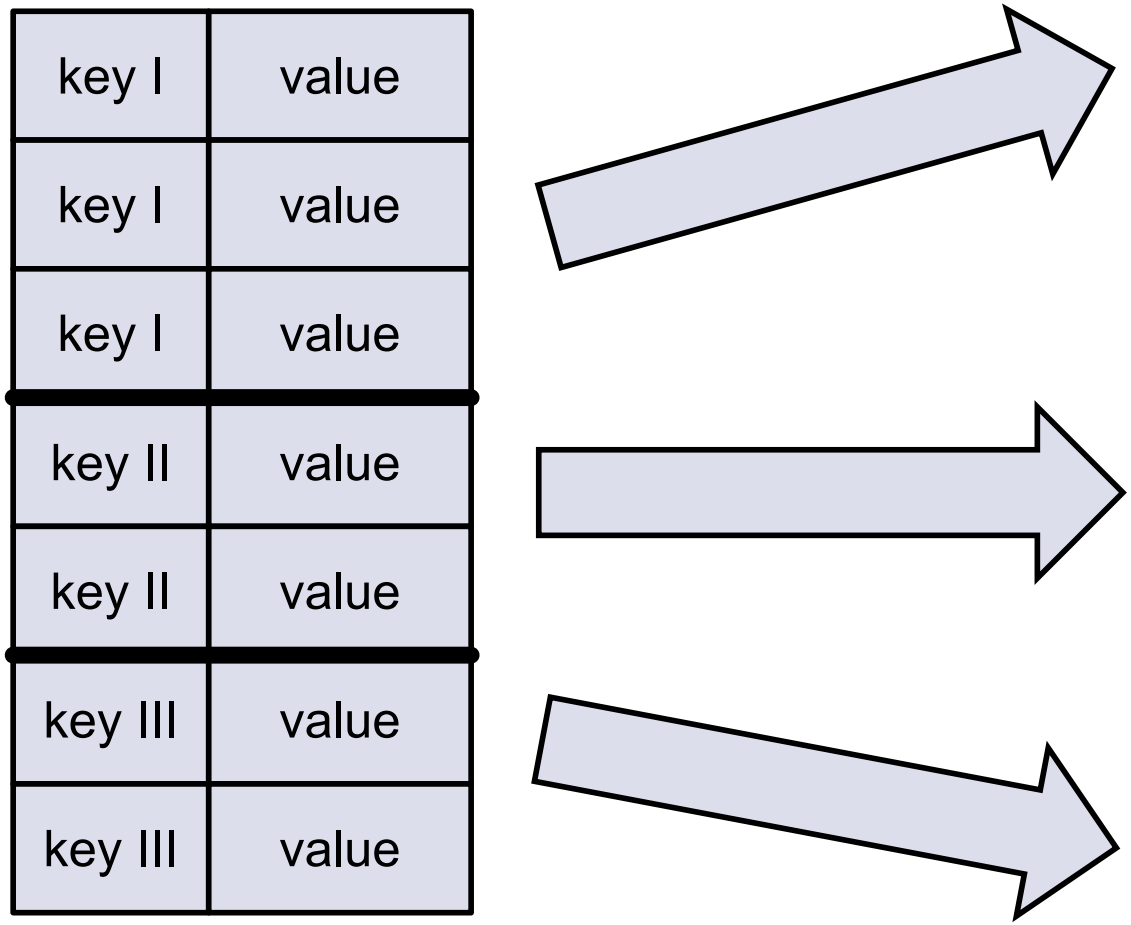
Sort by key


key I
value
key II
value
key I
value
key III
value
key III
value
key II
value
key I
value
key I
value
key I
value
key II
value
key II
value
key III
value
key III
value

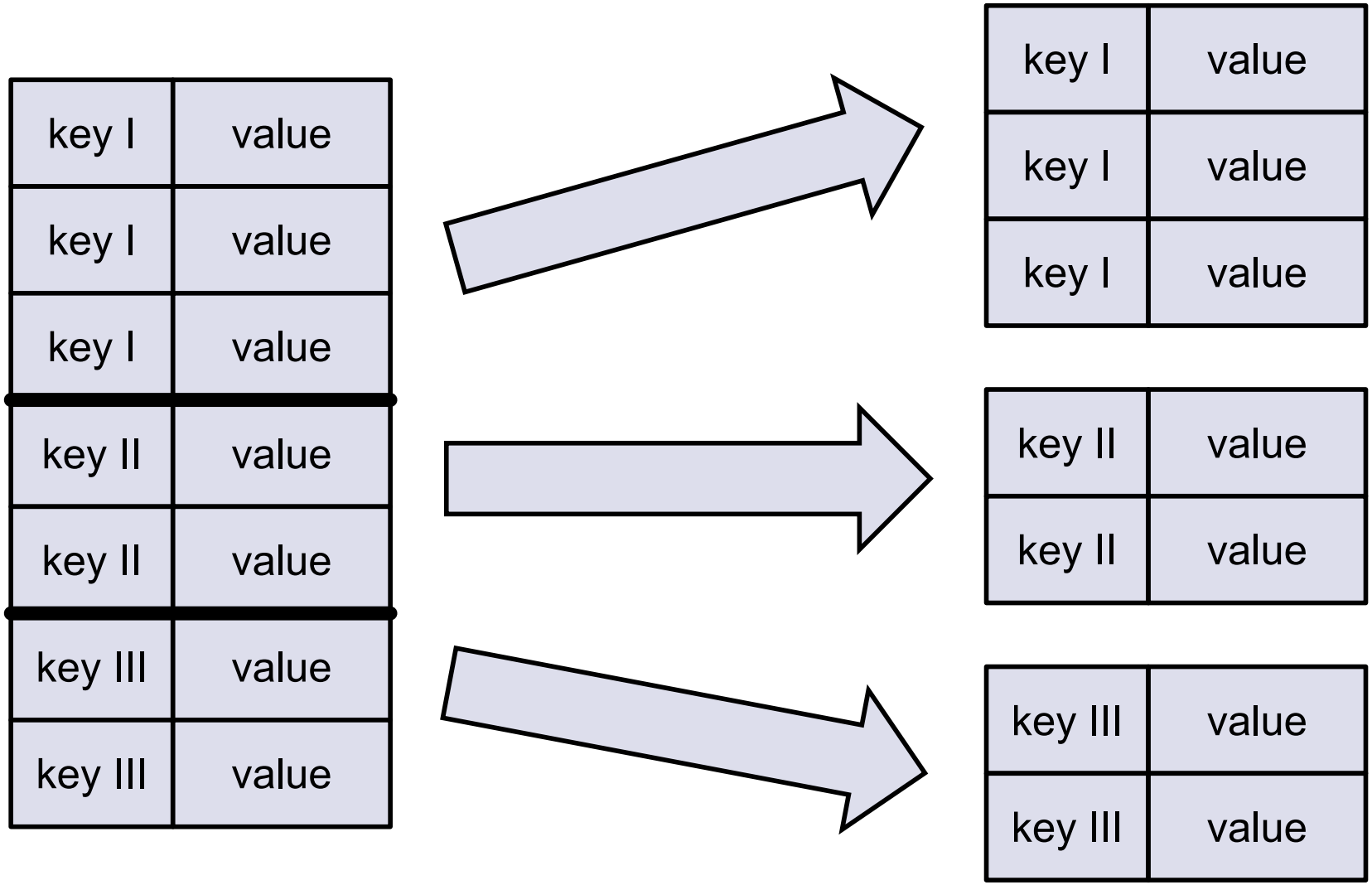
Partition

key I	value
key I	value
key I	value
key II	value
key II	value
key III	value
key III	value

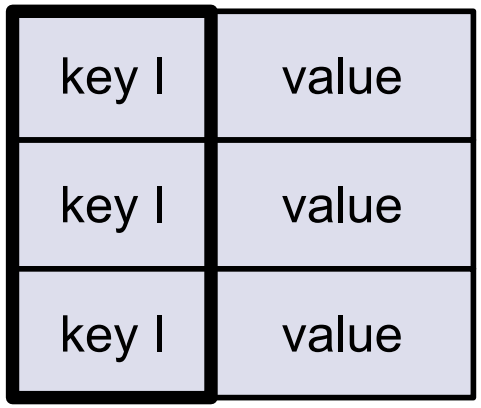
Partition


key I
value
key I
value
key I
value
key II
value
key II
value
key III
value
key III
value

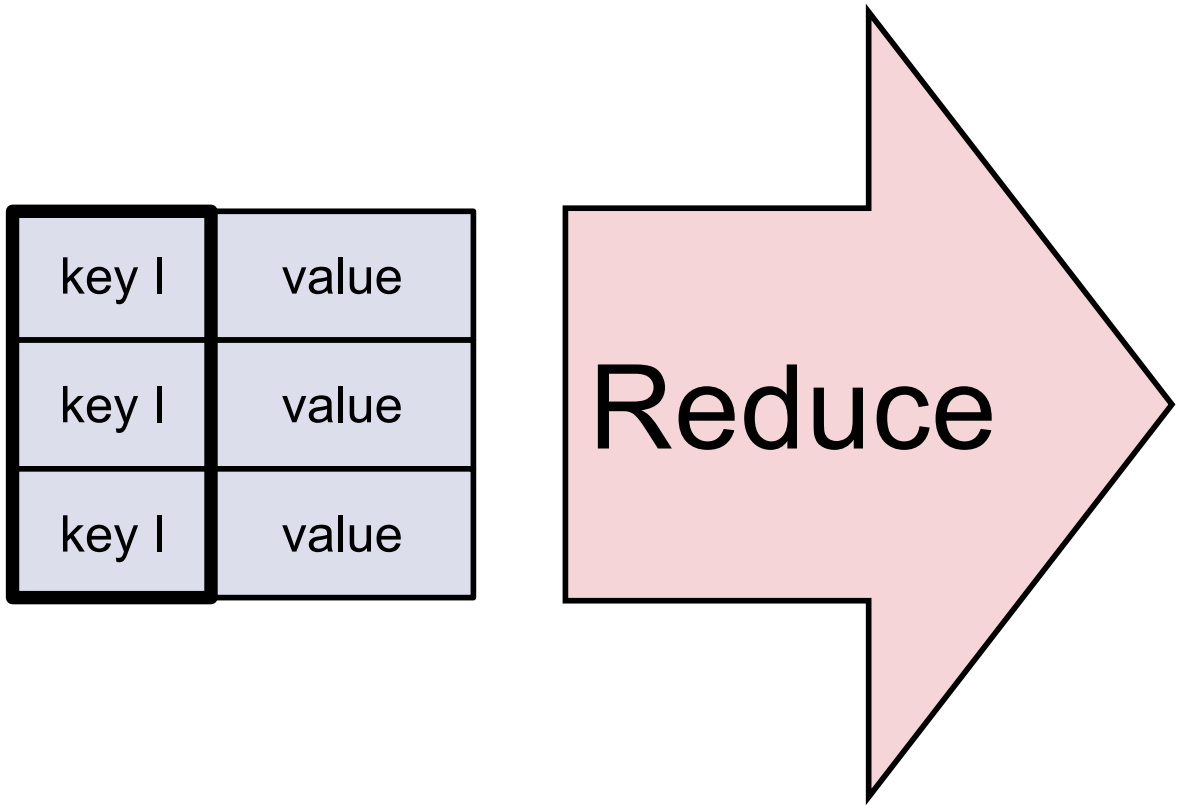
Partition


key I
value
key I
value
key I
value
key II
value
key II
value
key III
value
key III
value
key I
value
key I
value
key I
value
key II
value
key II
value
key III
value
key III
value

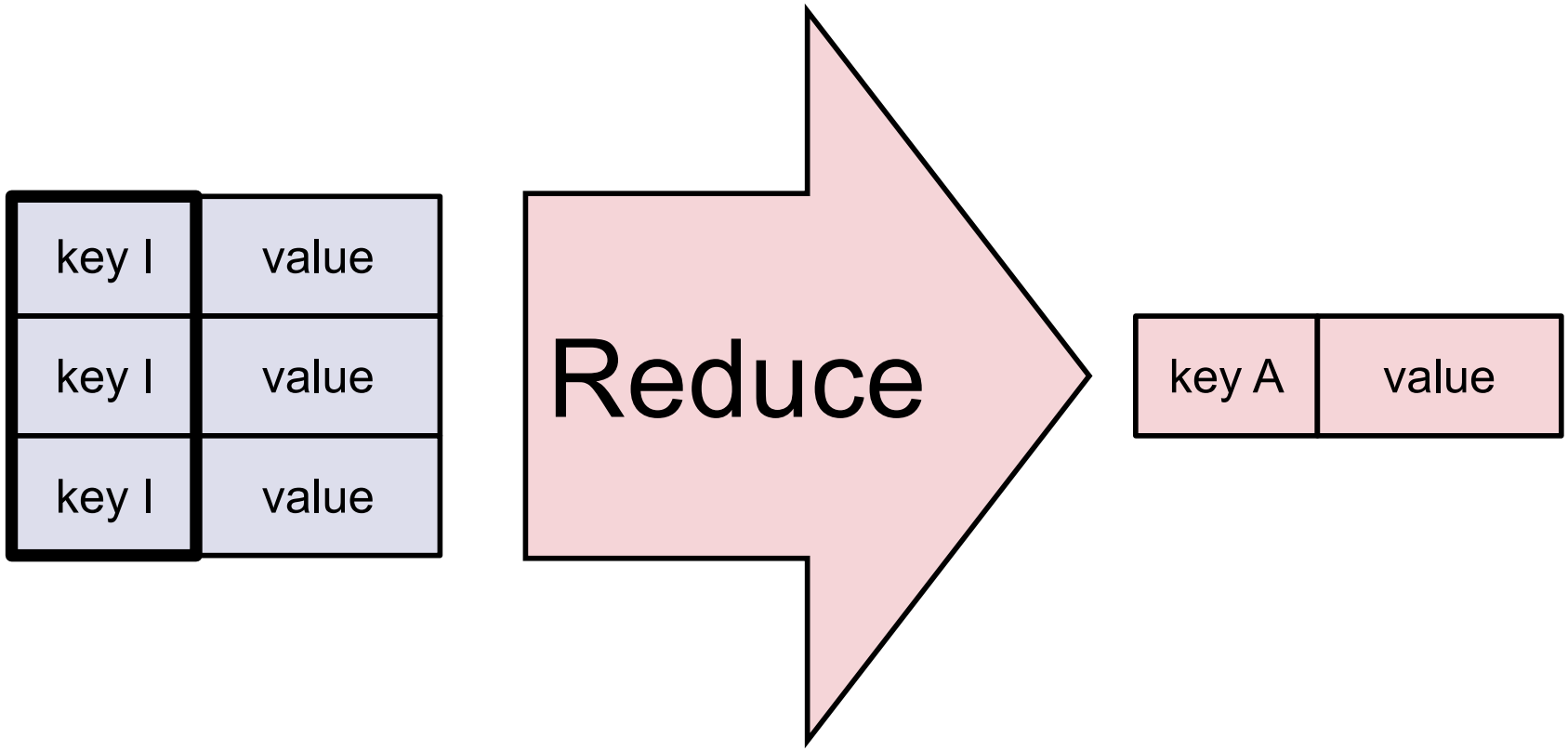
Reduce function



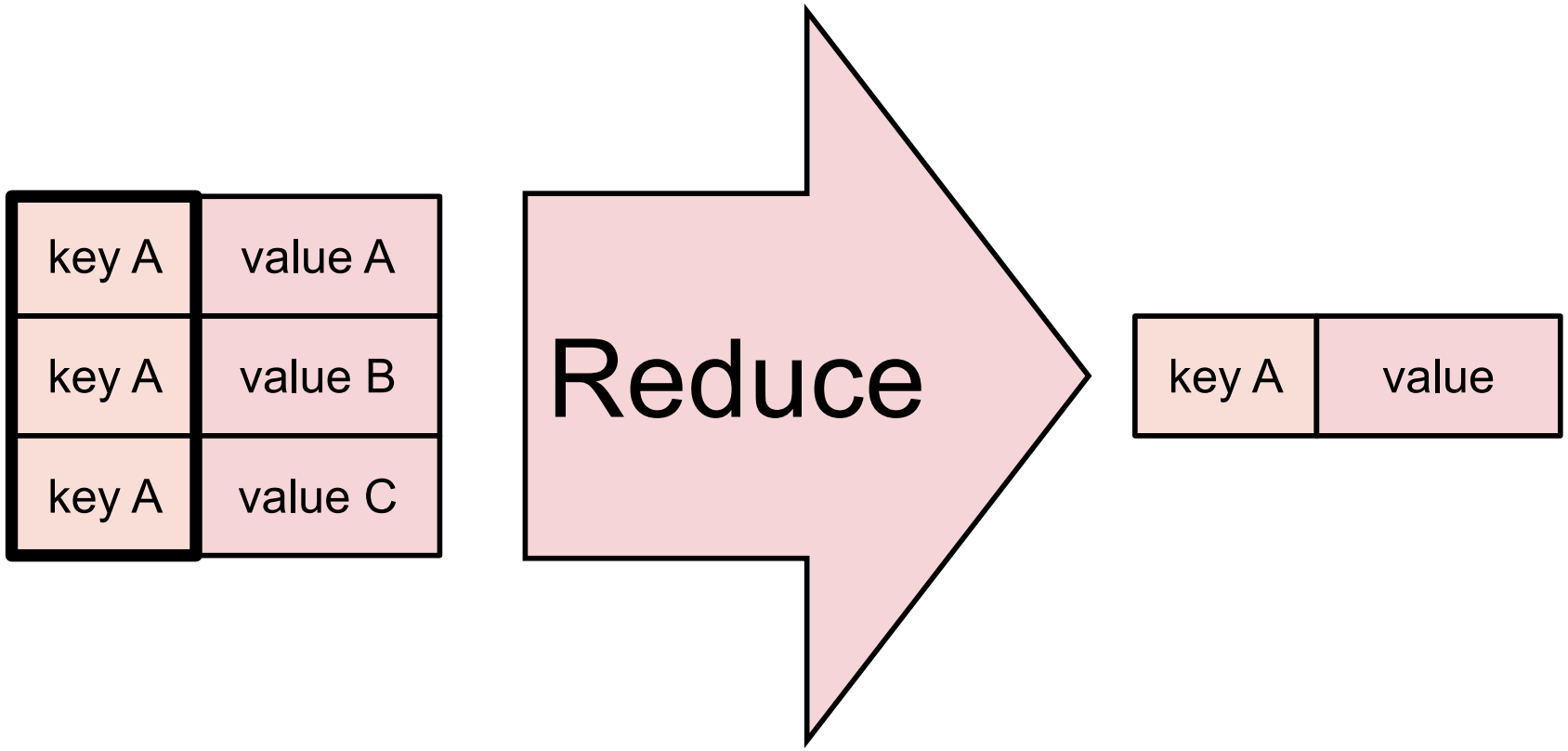
Reduce function



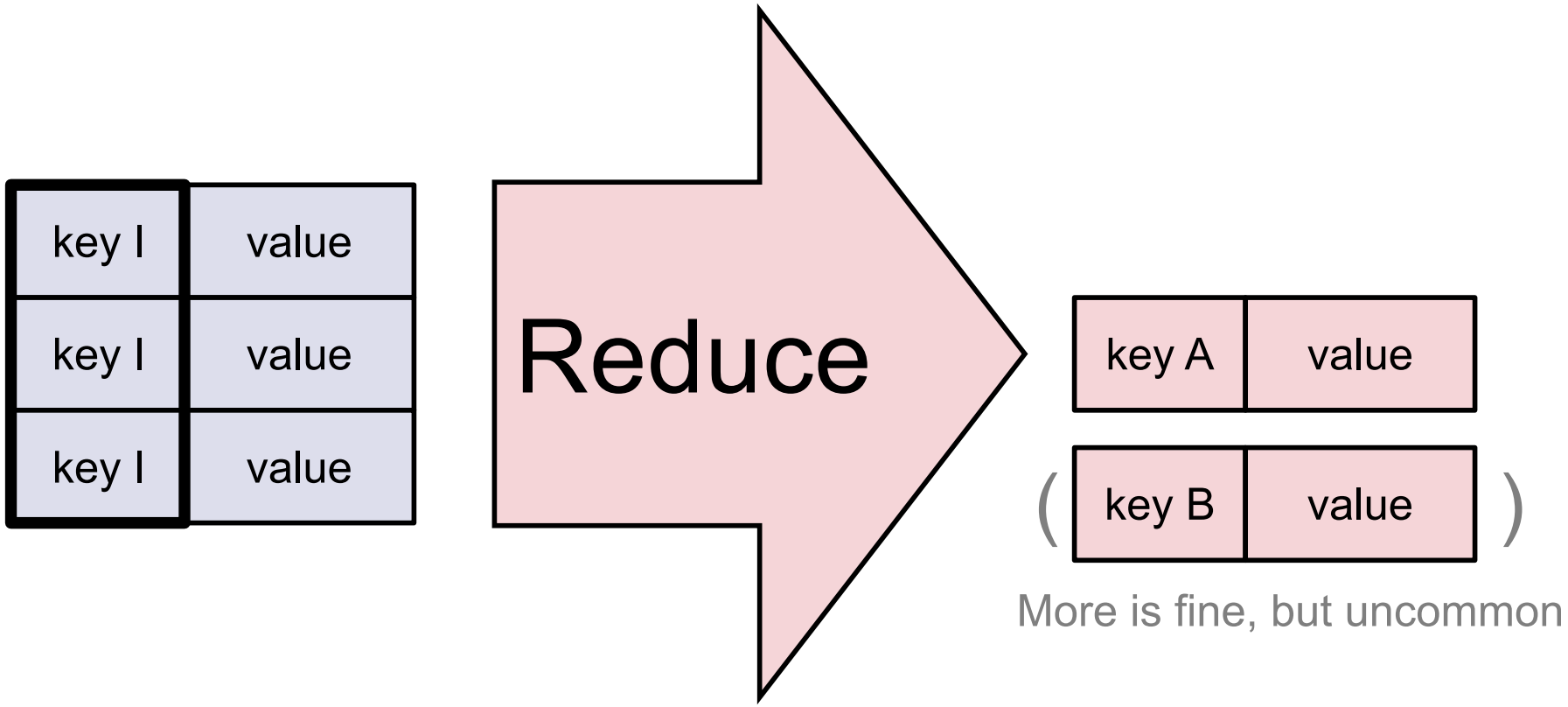
Reduce function



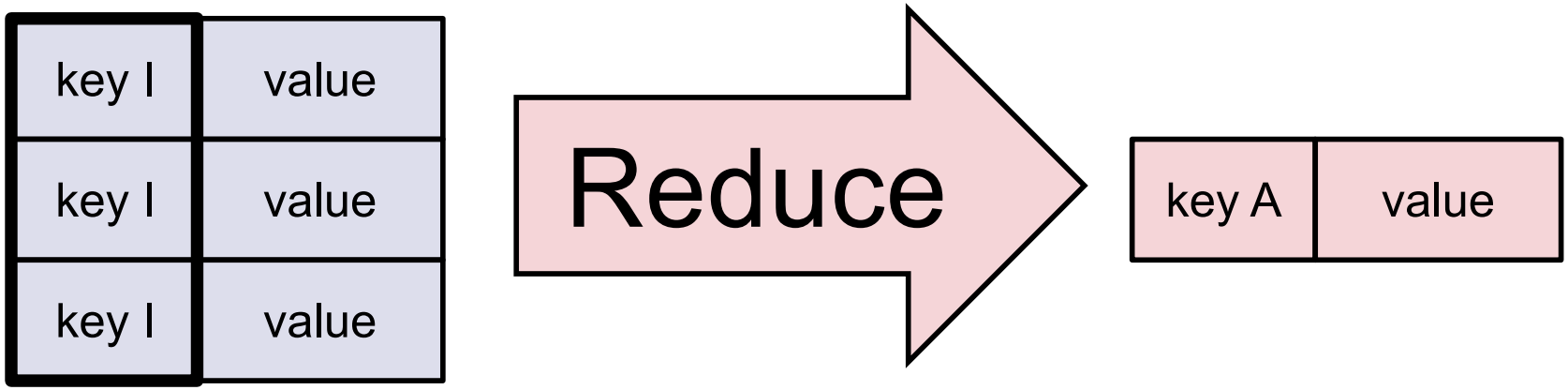
Reduce function (with identical key sets)



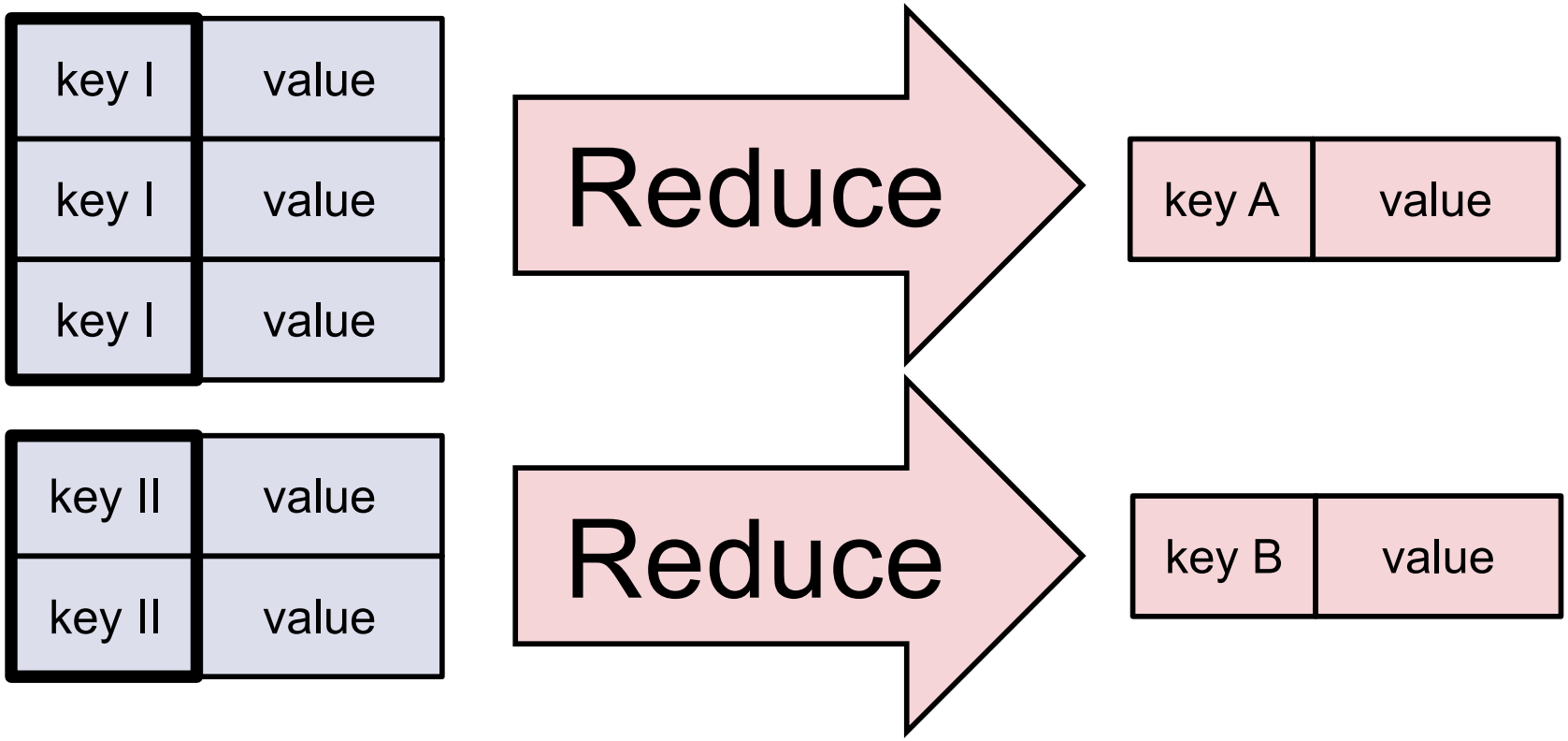
Reduce function (most generic)



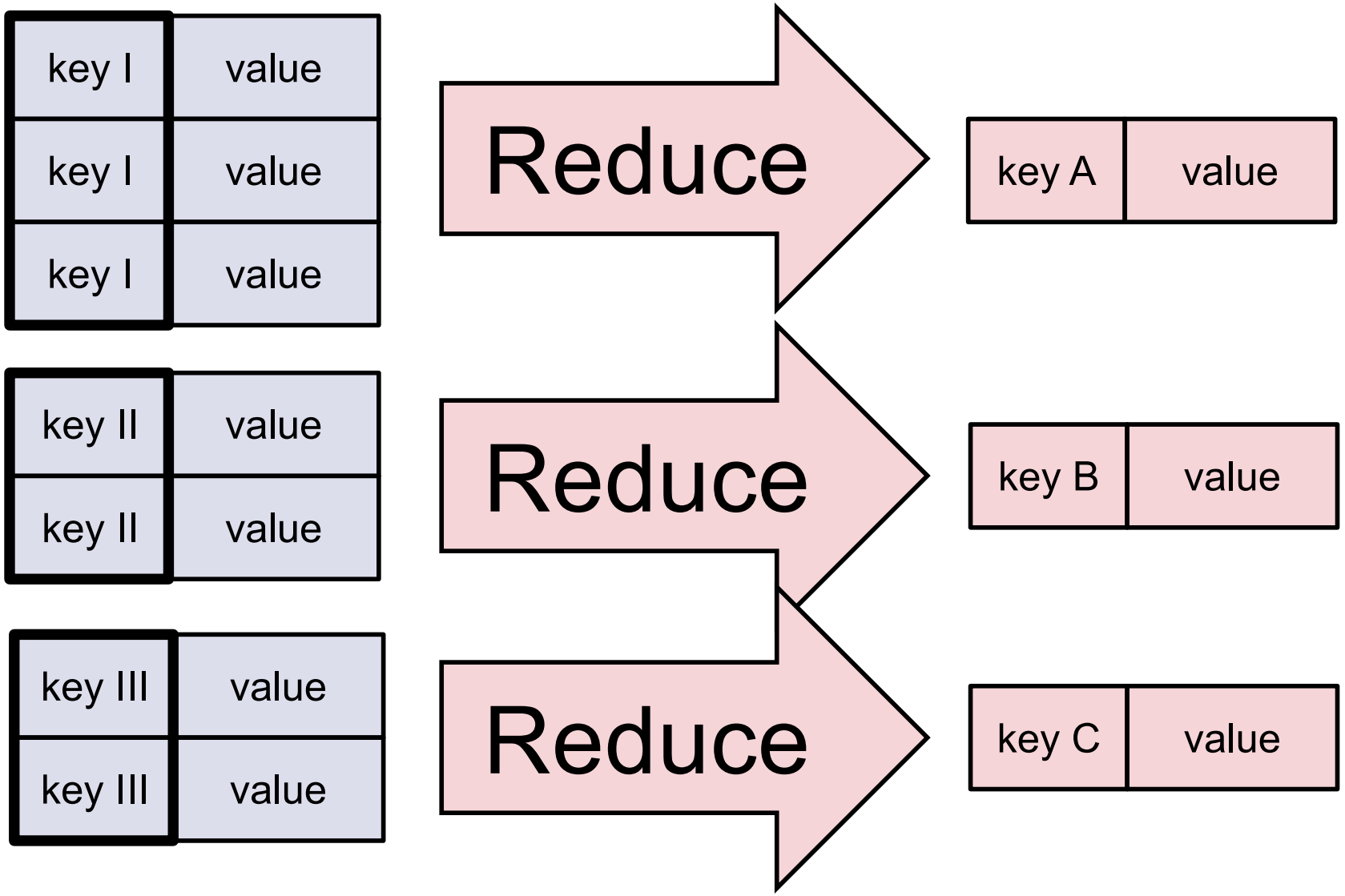
Reduce function... in parallel



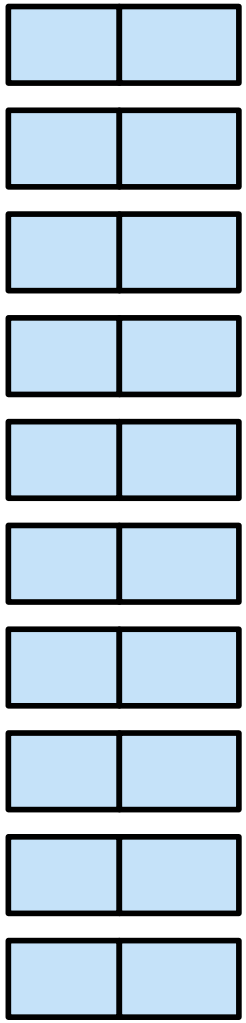
Reduce function... in parallel



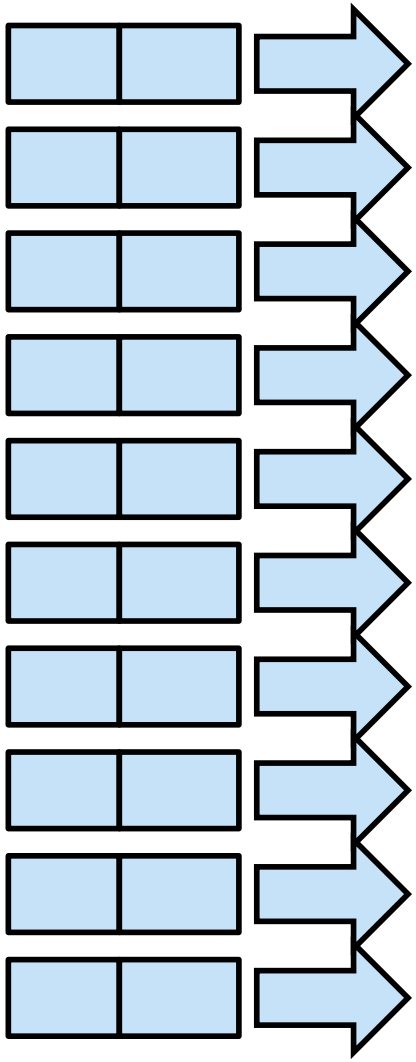
Reduce function... in parallel



Overall

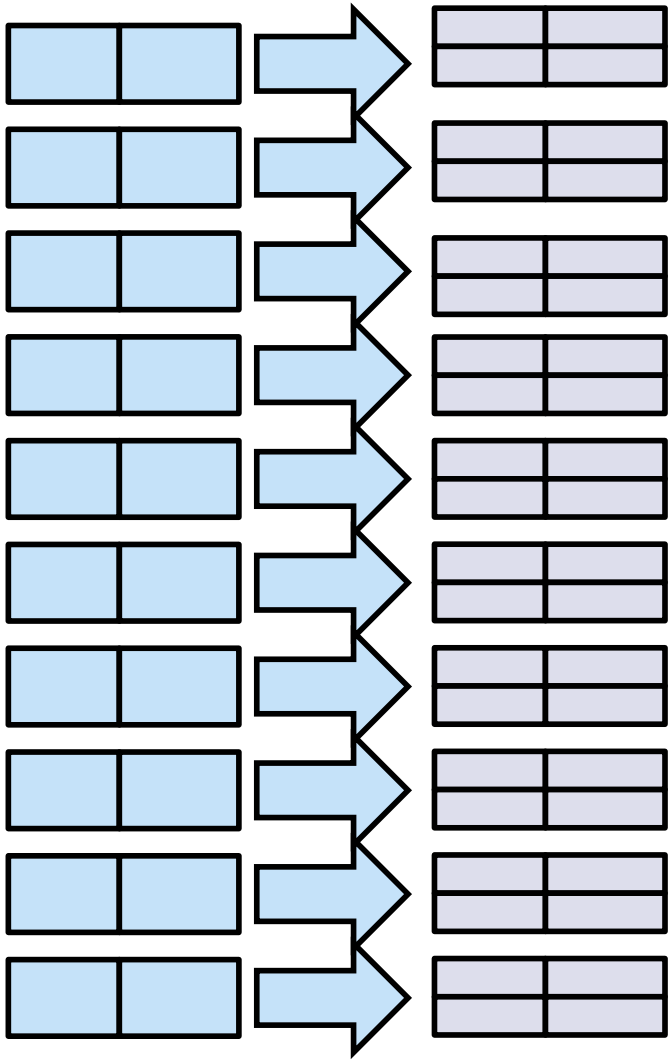


Overall



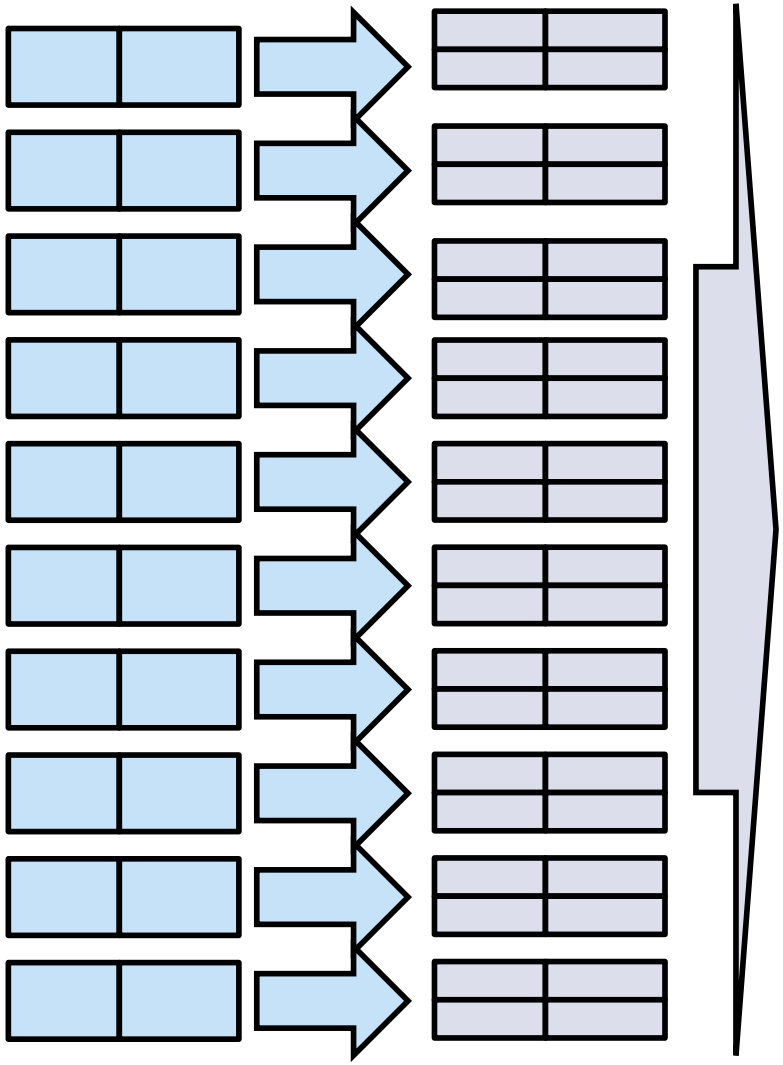
Map

Overall

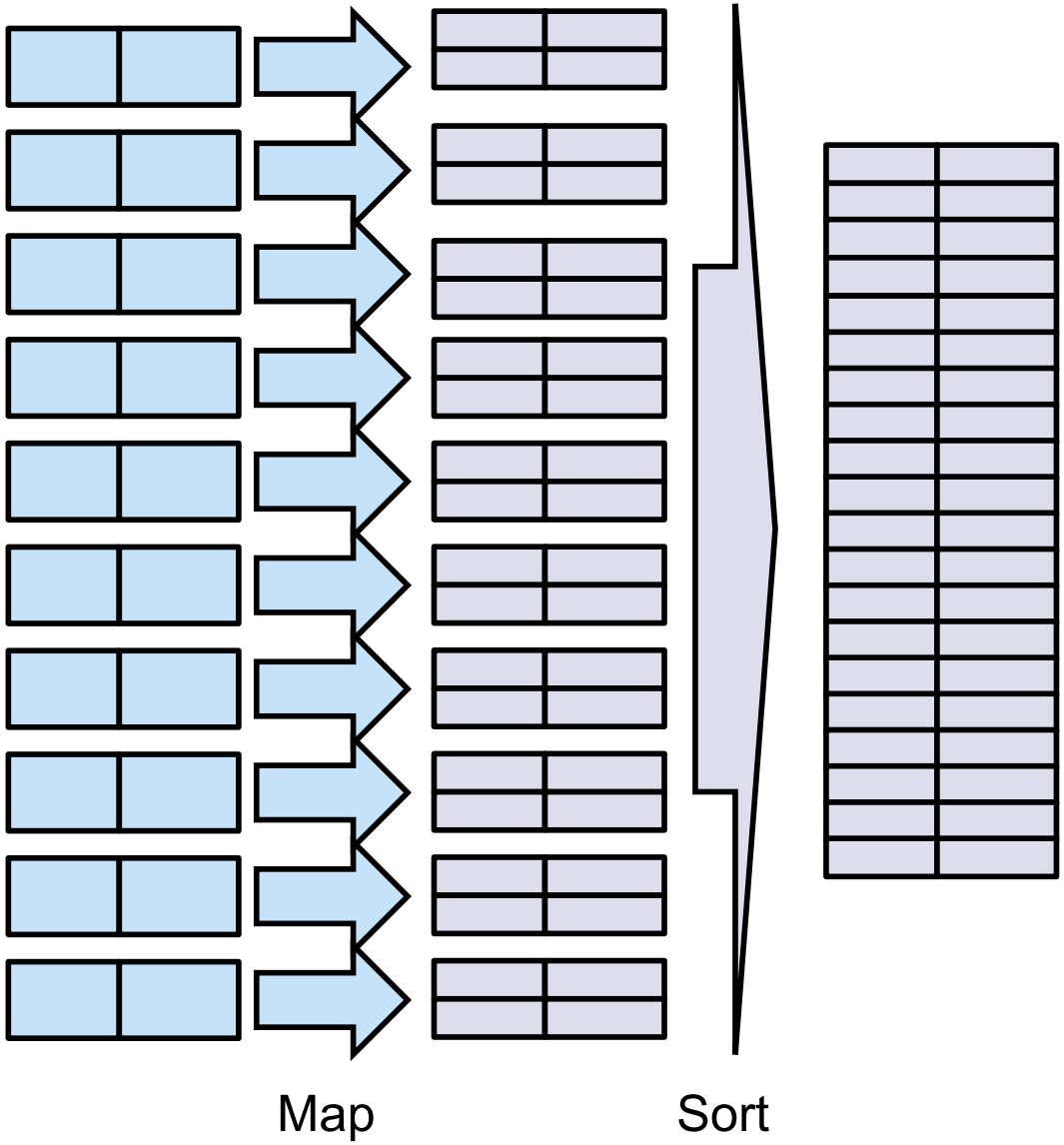


Map

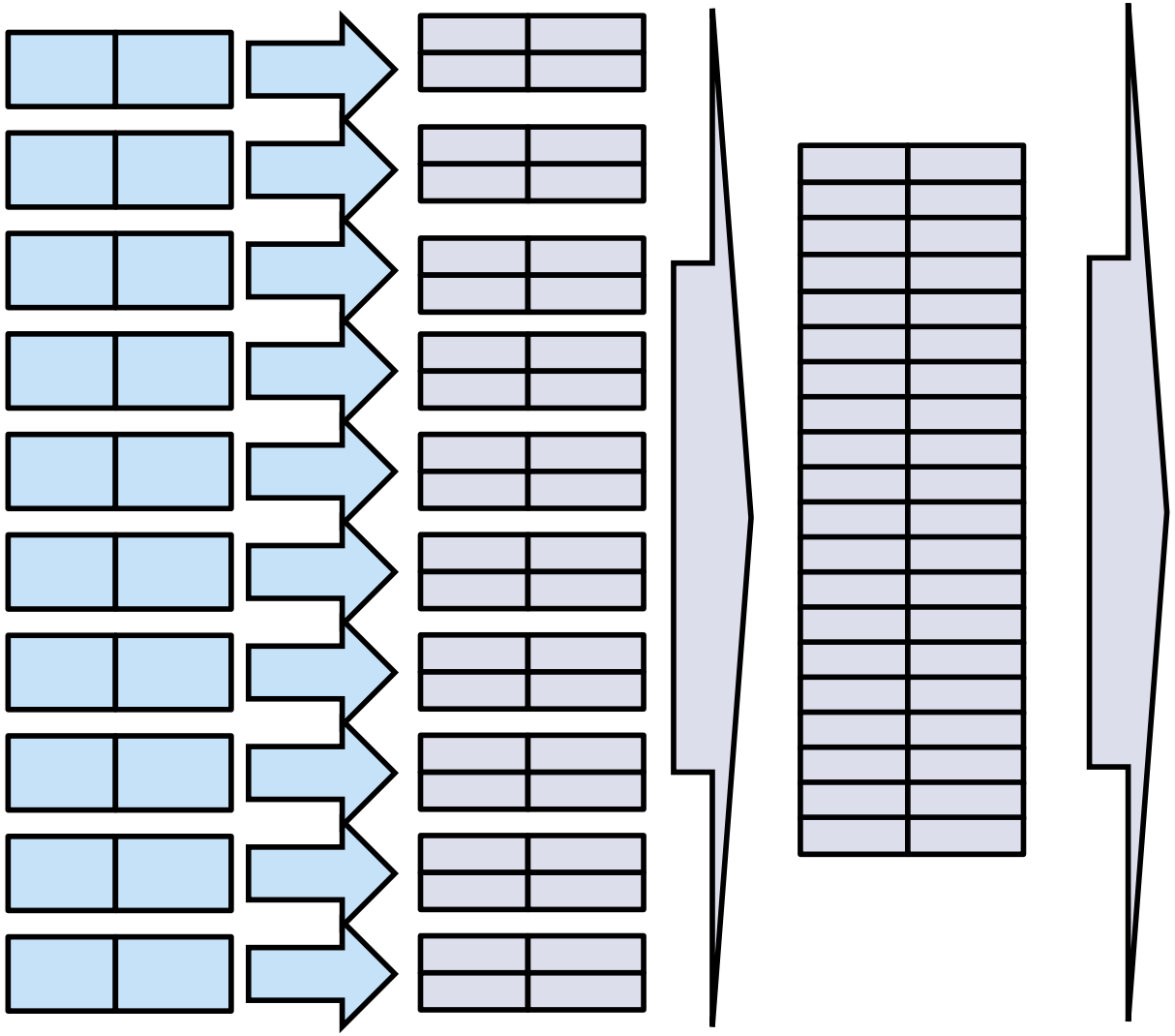
Overall



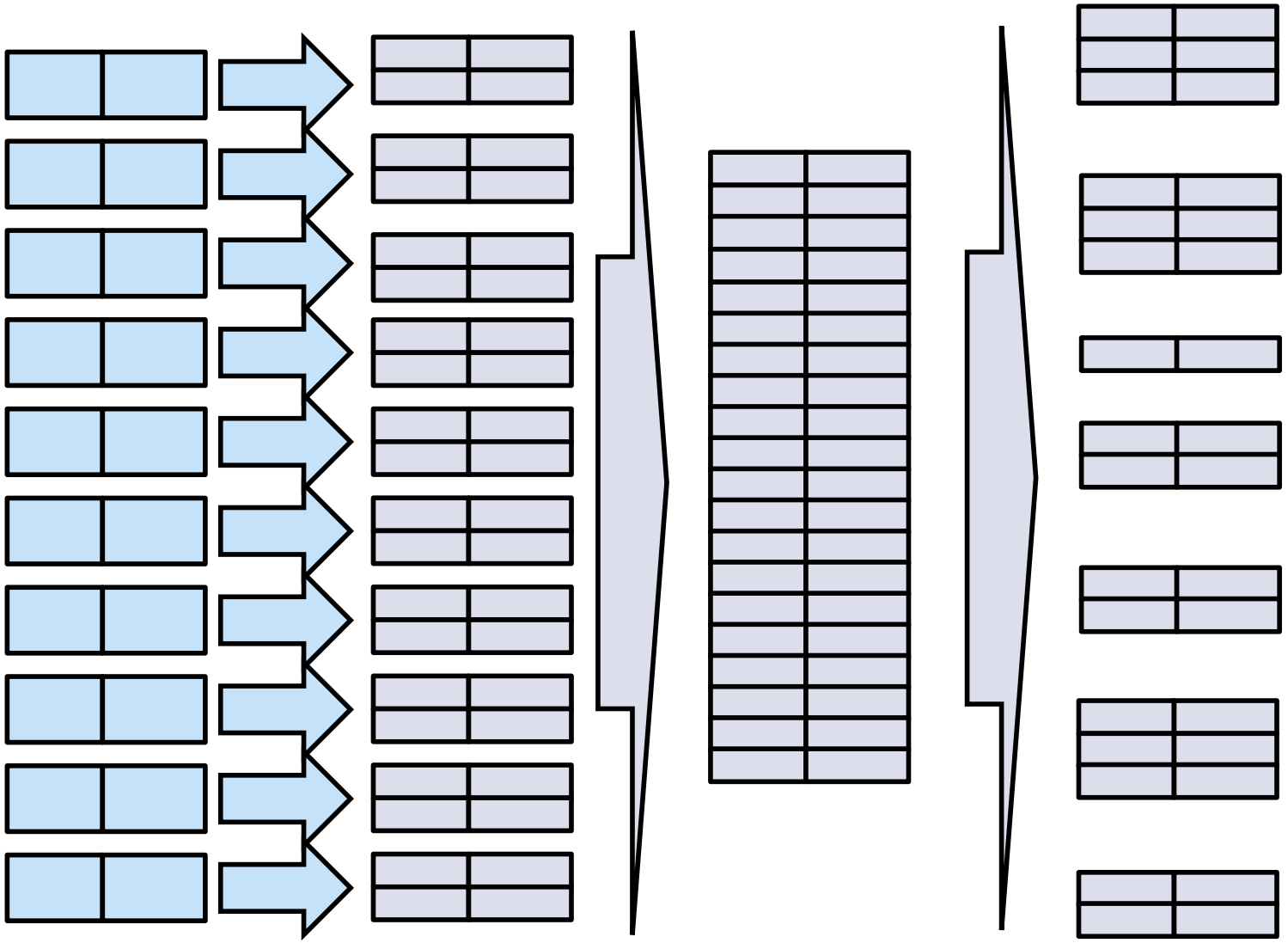
Overall



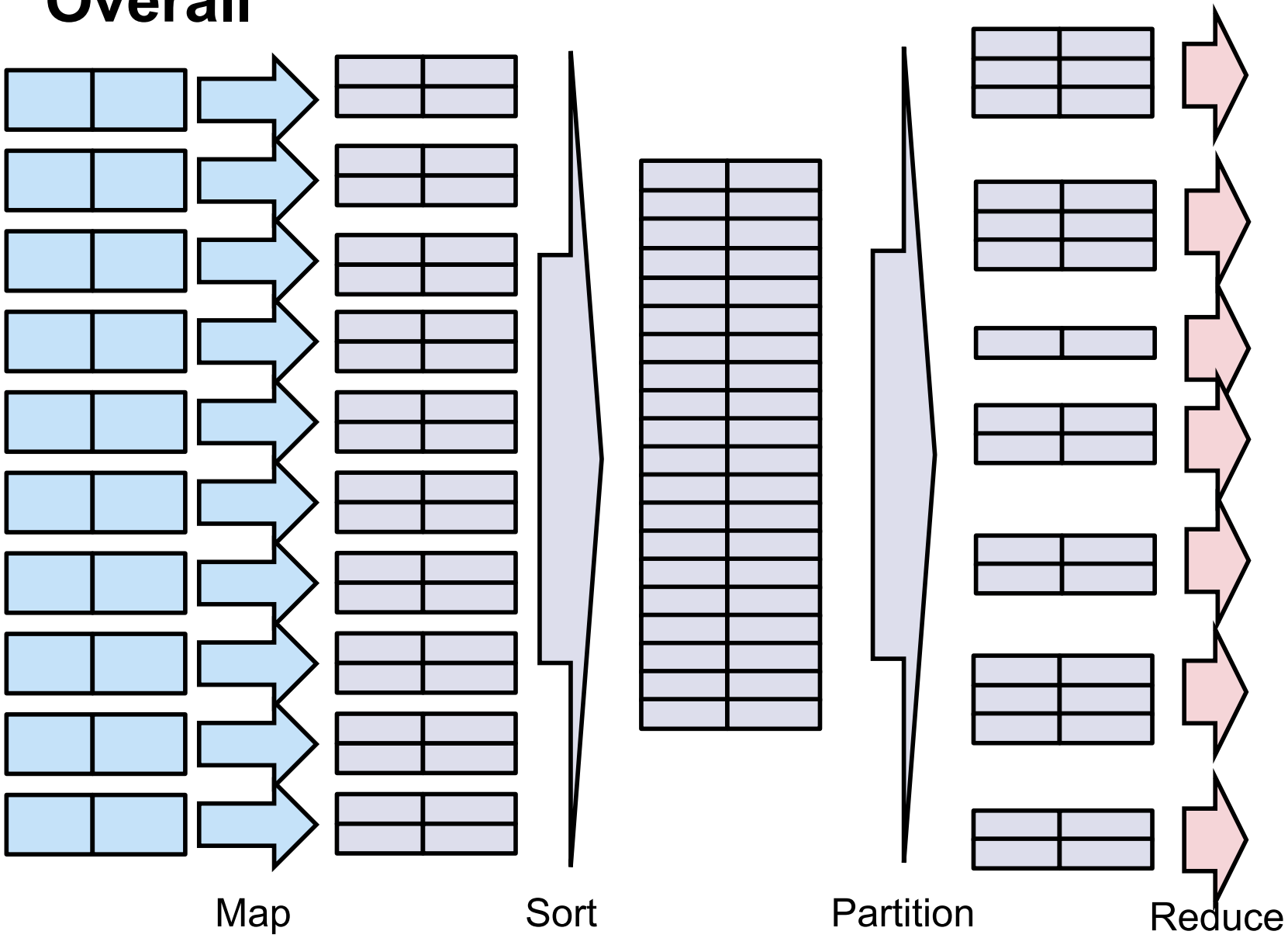
Overall



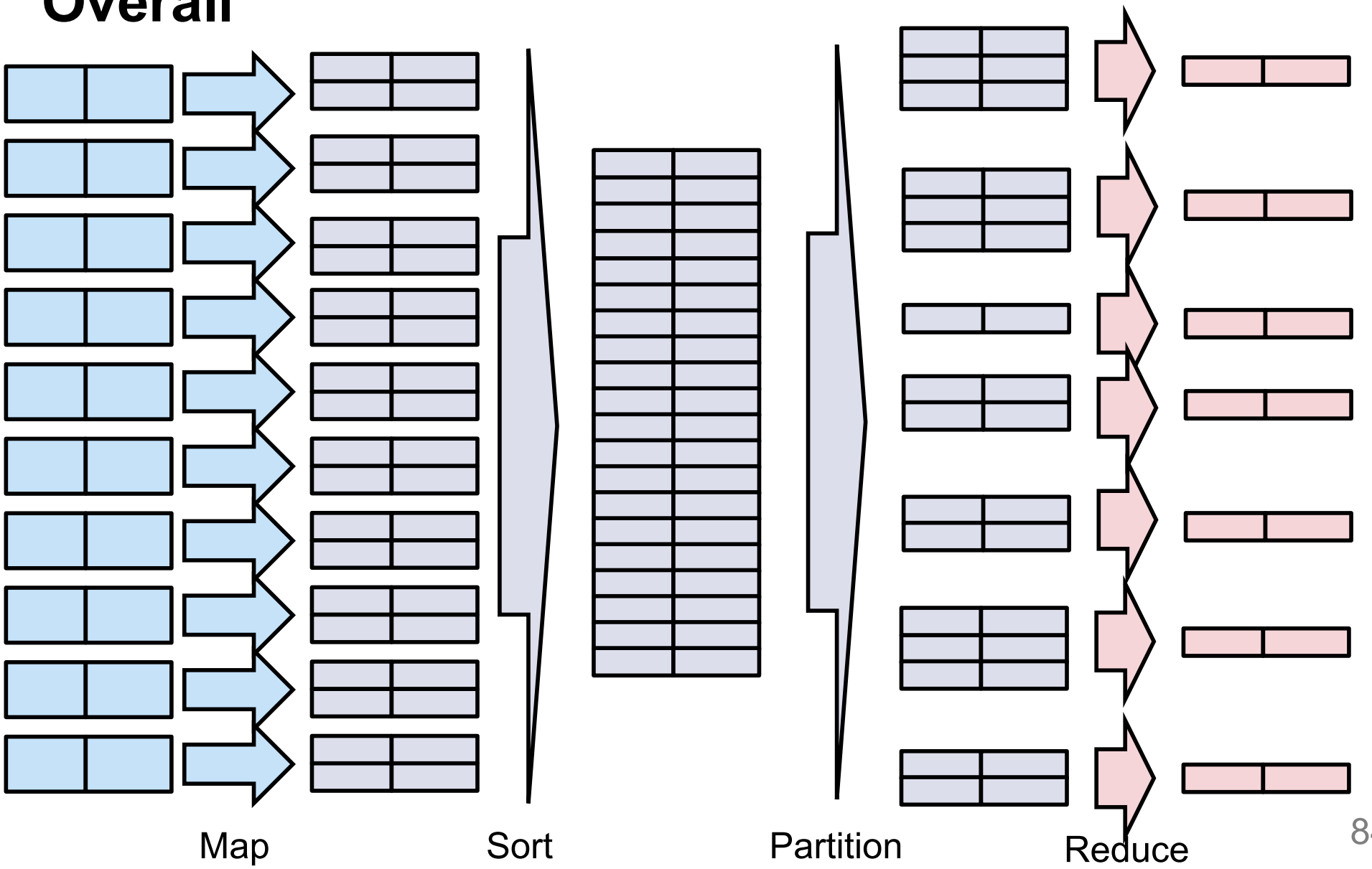
Overall



Overall

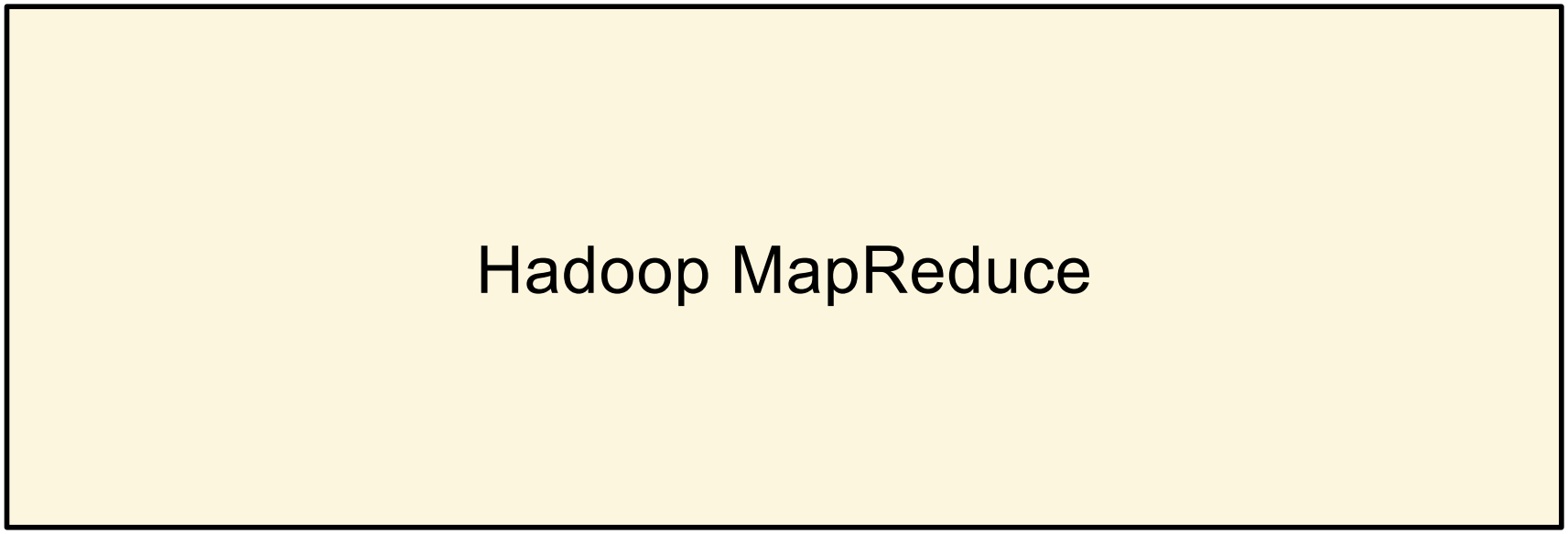


Overall

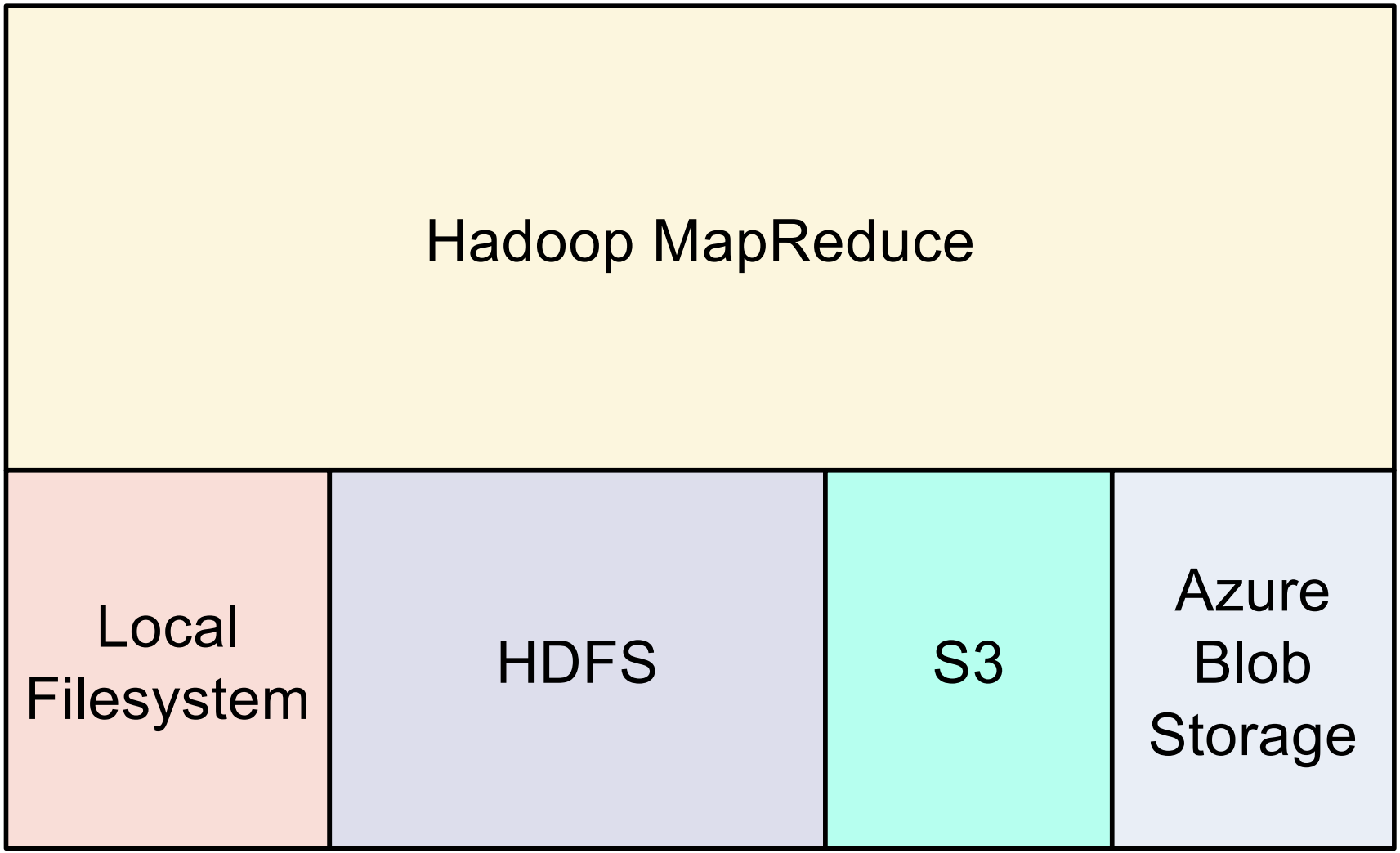


Architecture

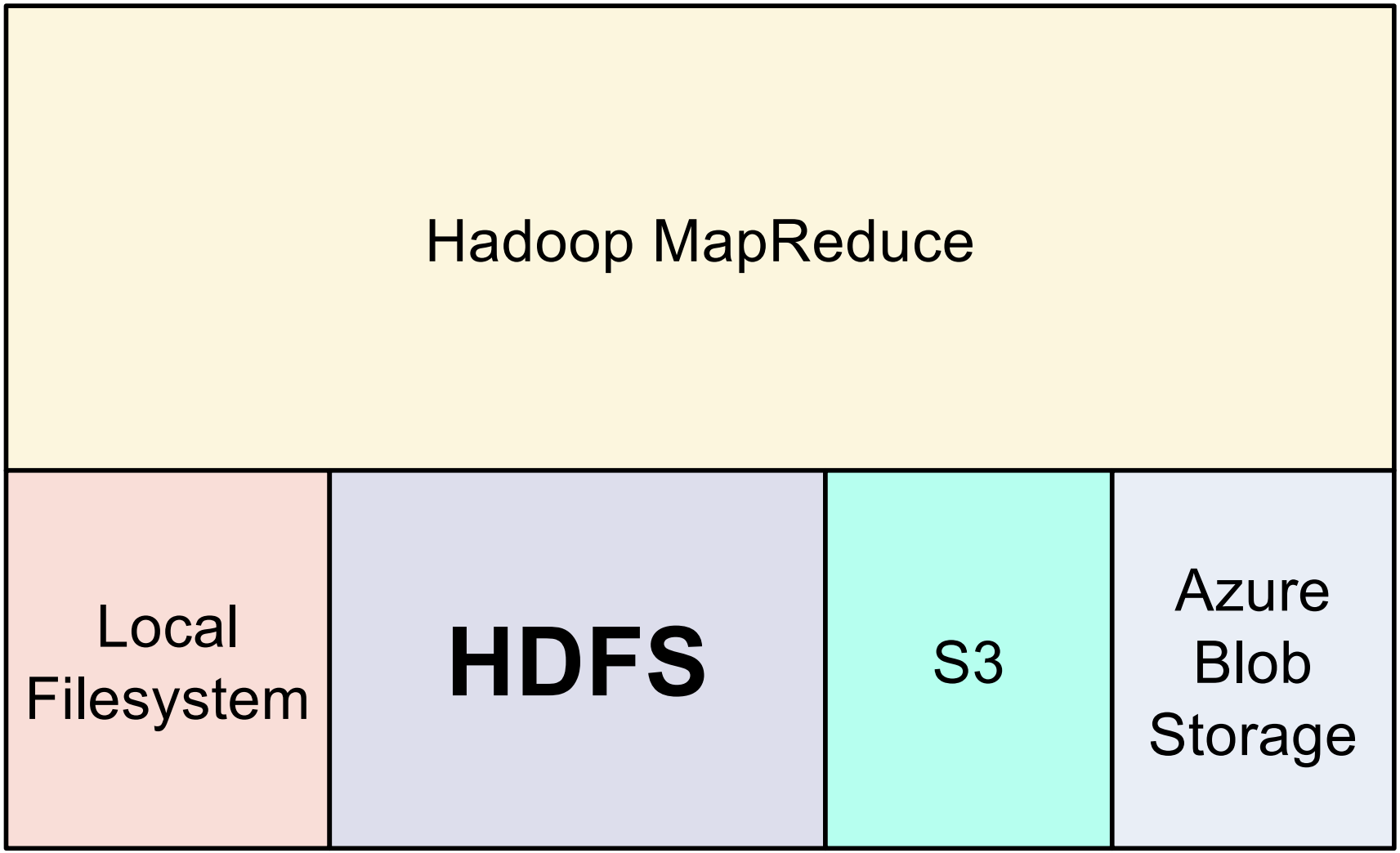
Possible storage layers



Possible storage layers



Possible storage layers



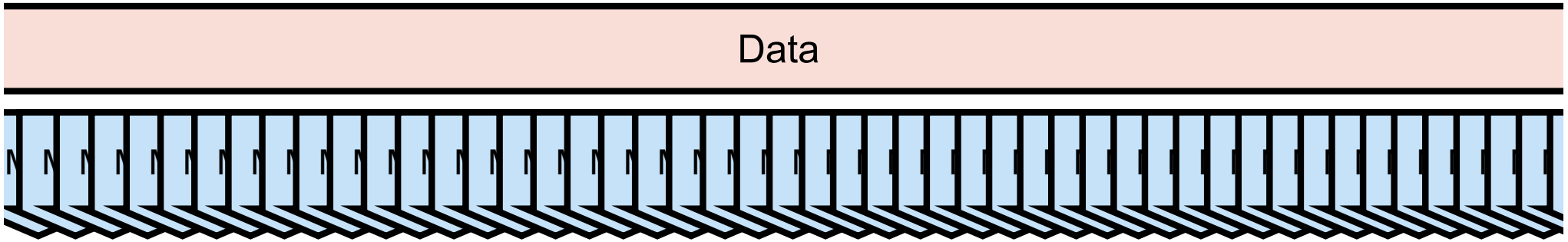
Hadoop MapReduce: Numbers

Several TBs of data

Data

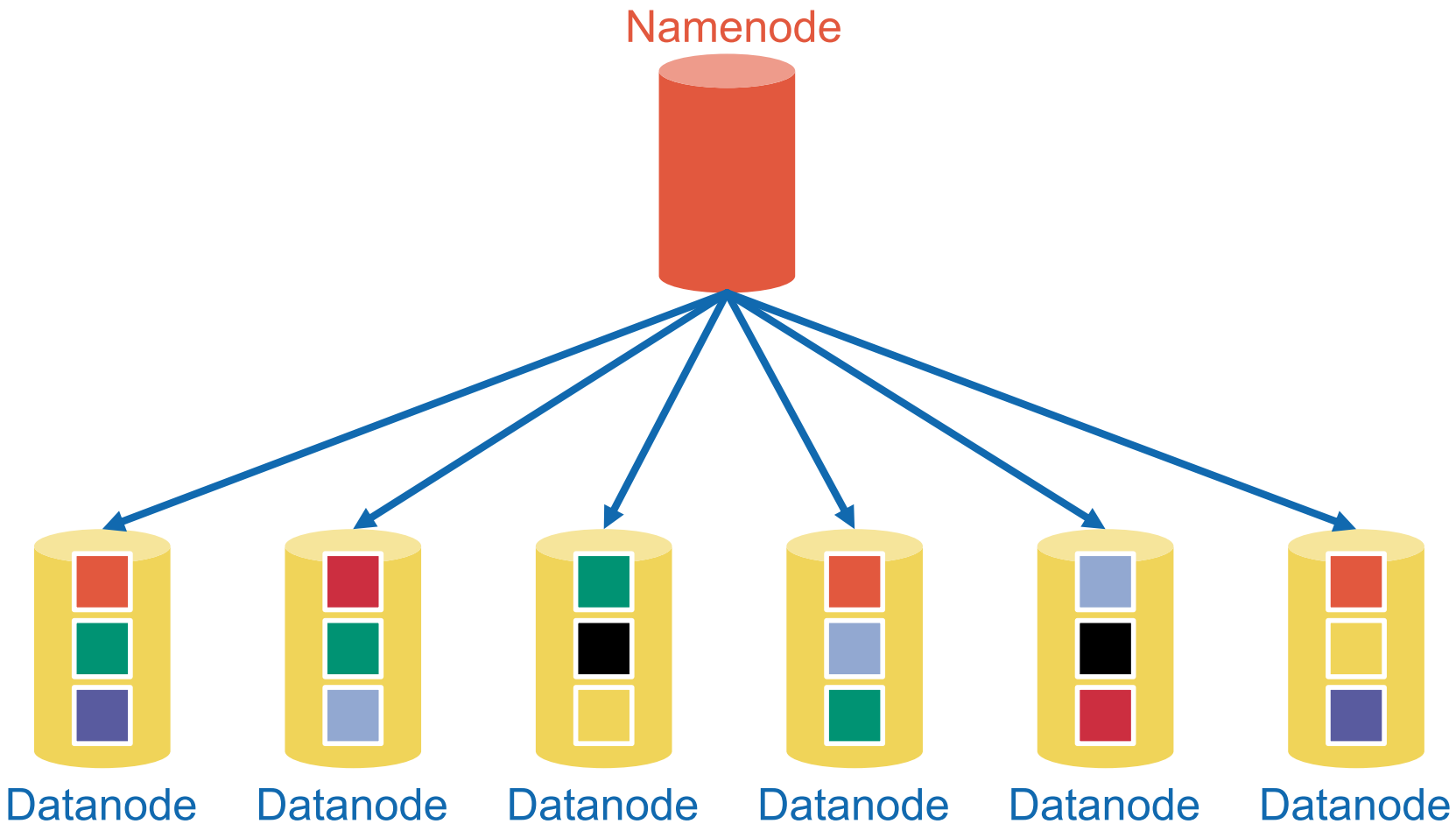
Hadoop MapReduce: Numbers

Several TBs of data

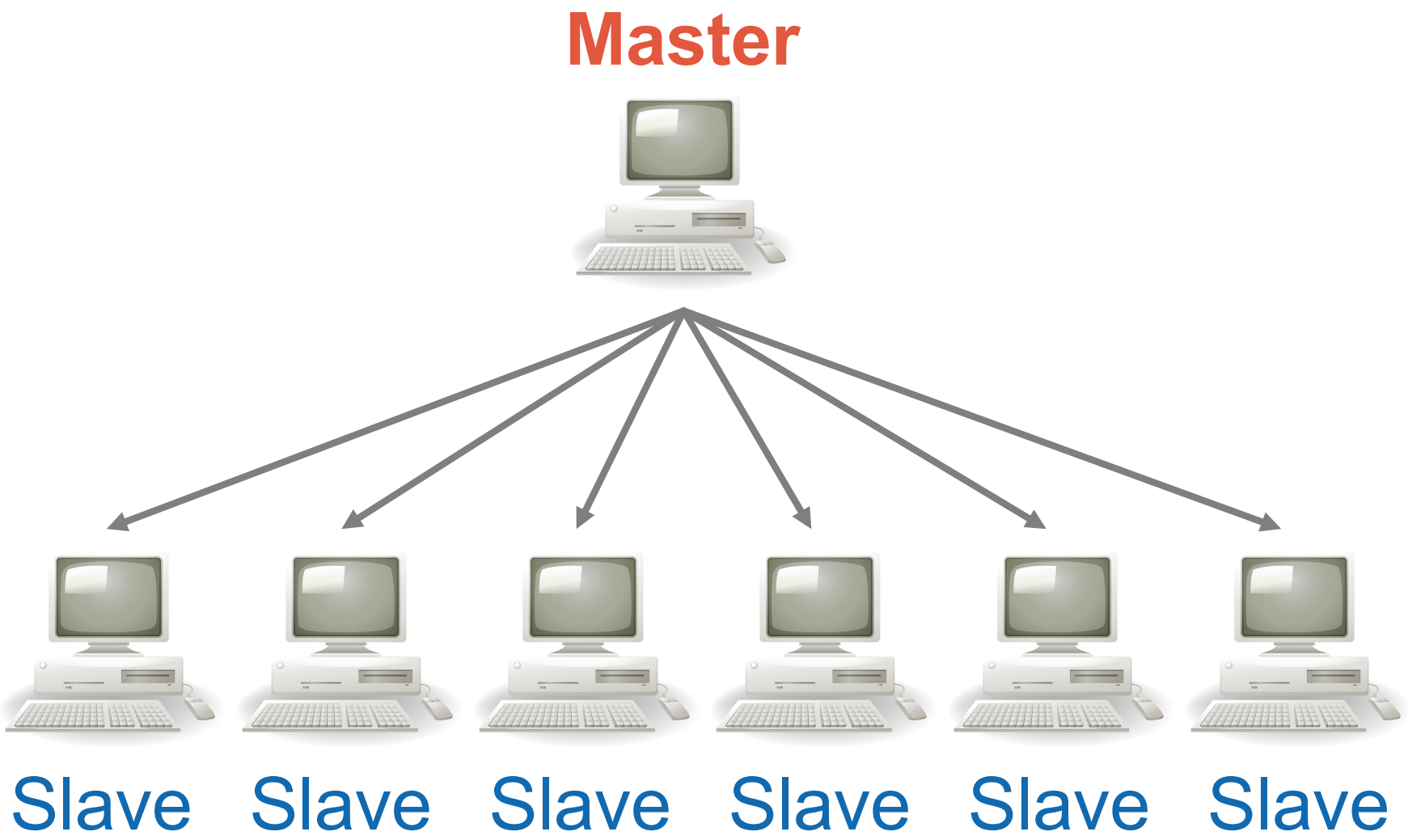


1000s of nodes

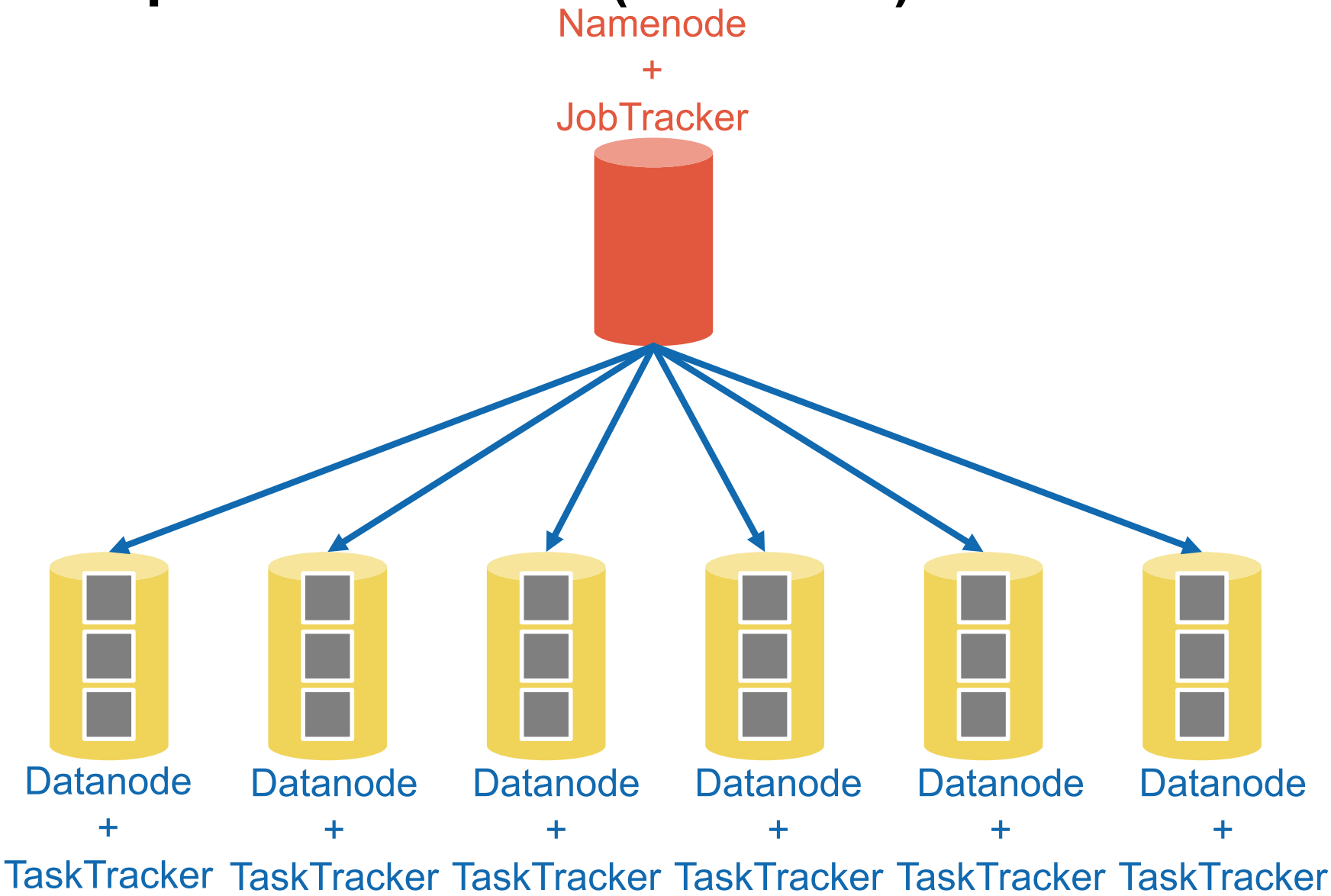
Hadoop infrastructure (version 1)



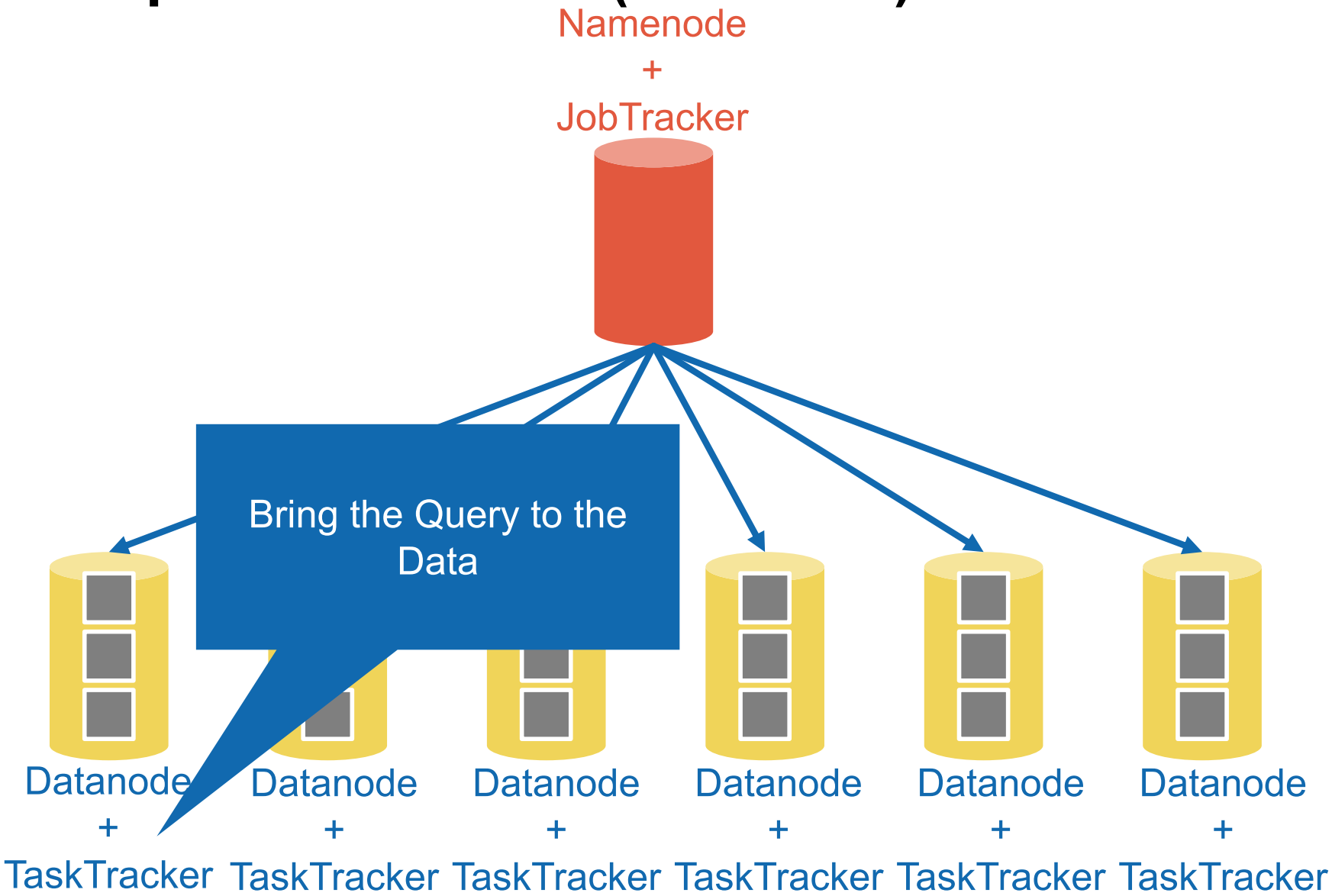
Master-slave architecture



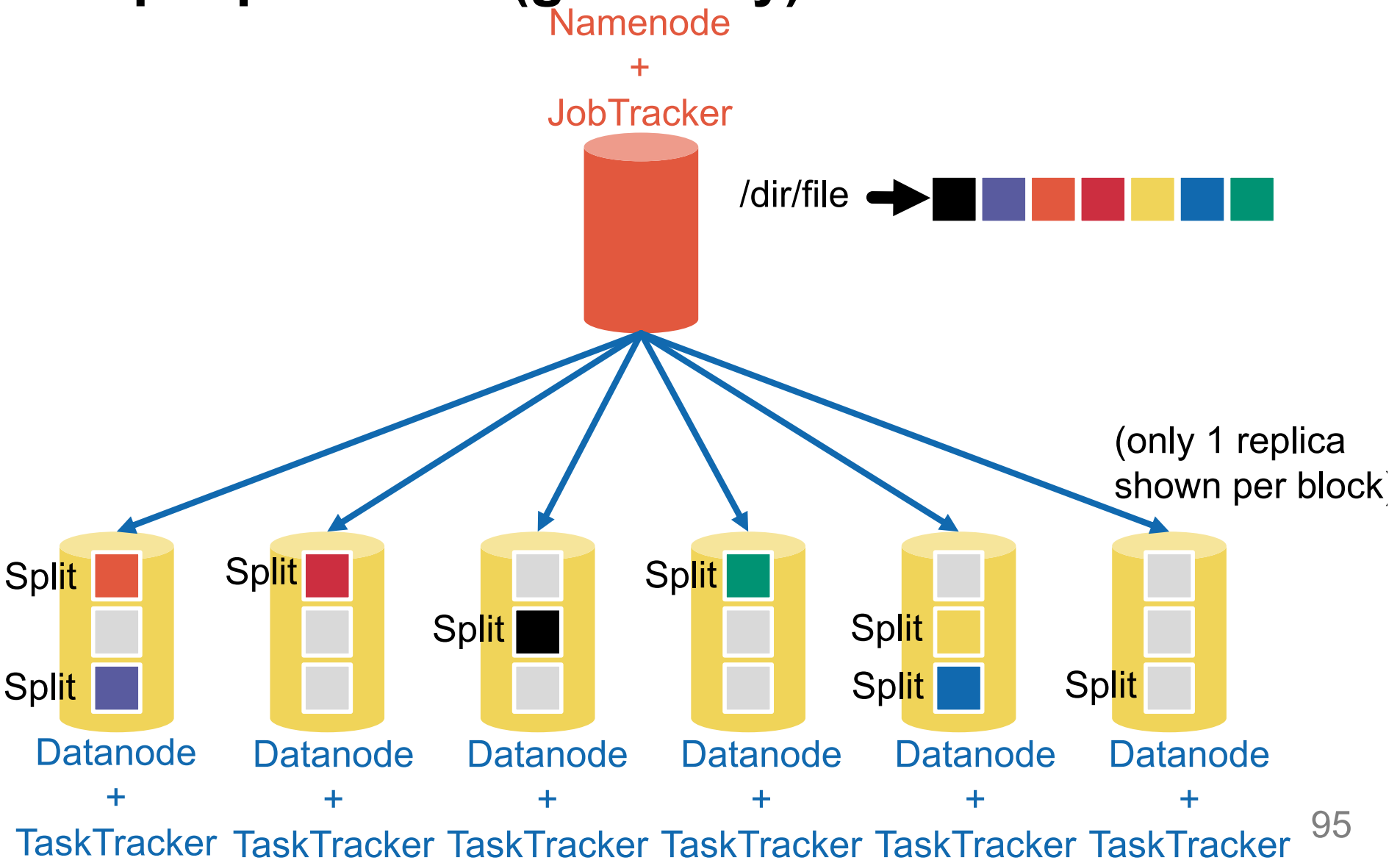
Hadoop infrastructure (version 1)



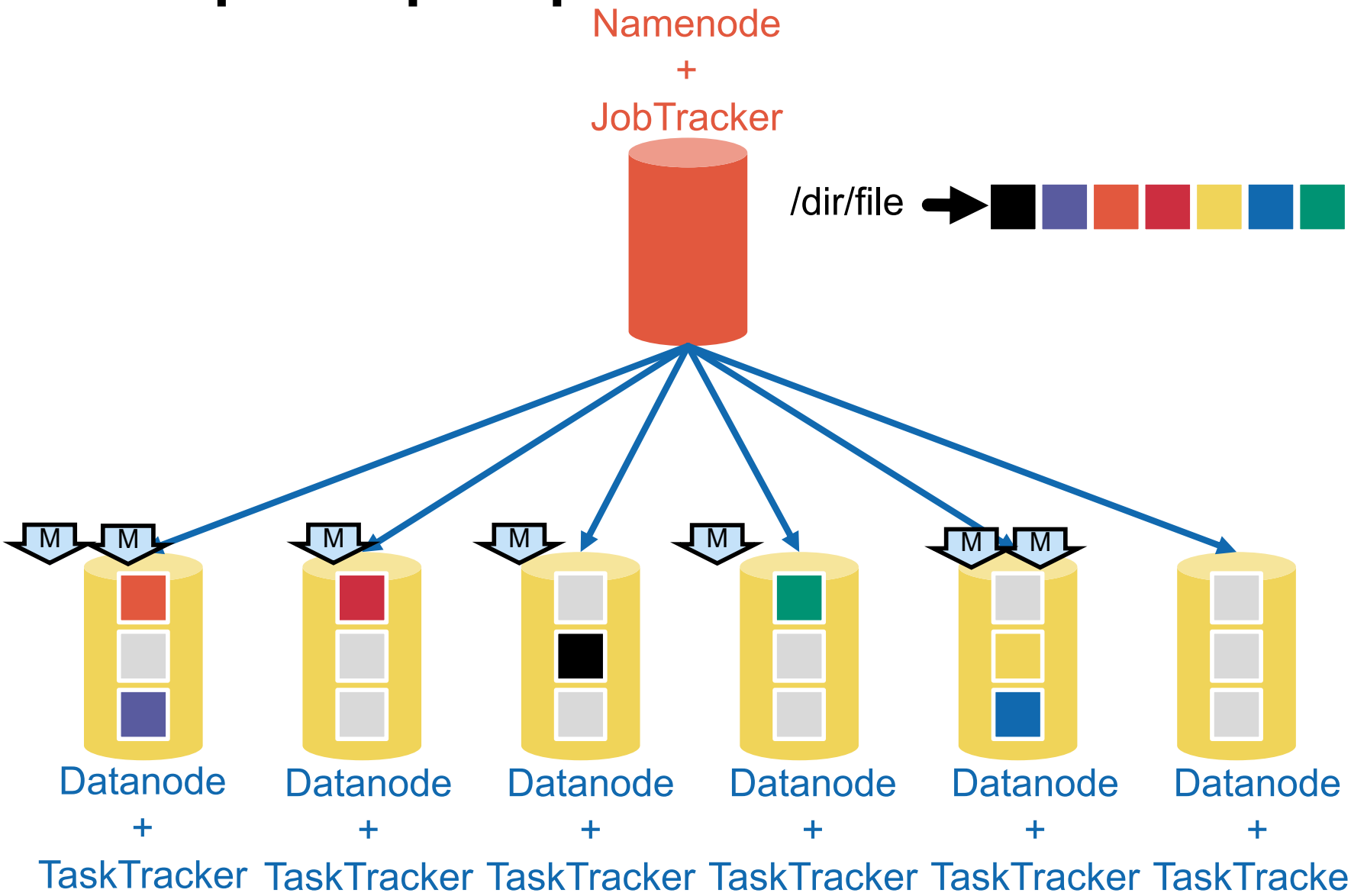
Hadoop infrastructure (version 1)



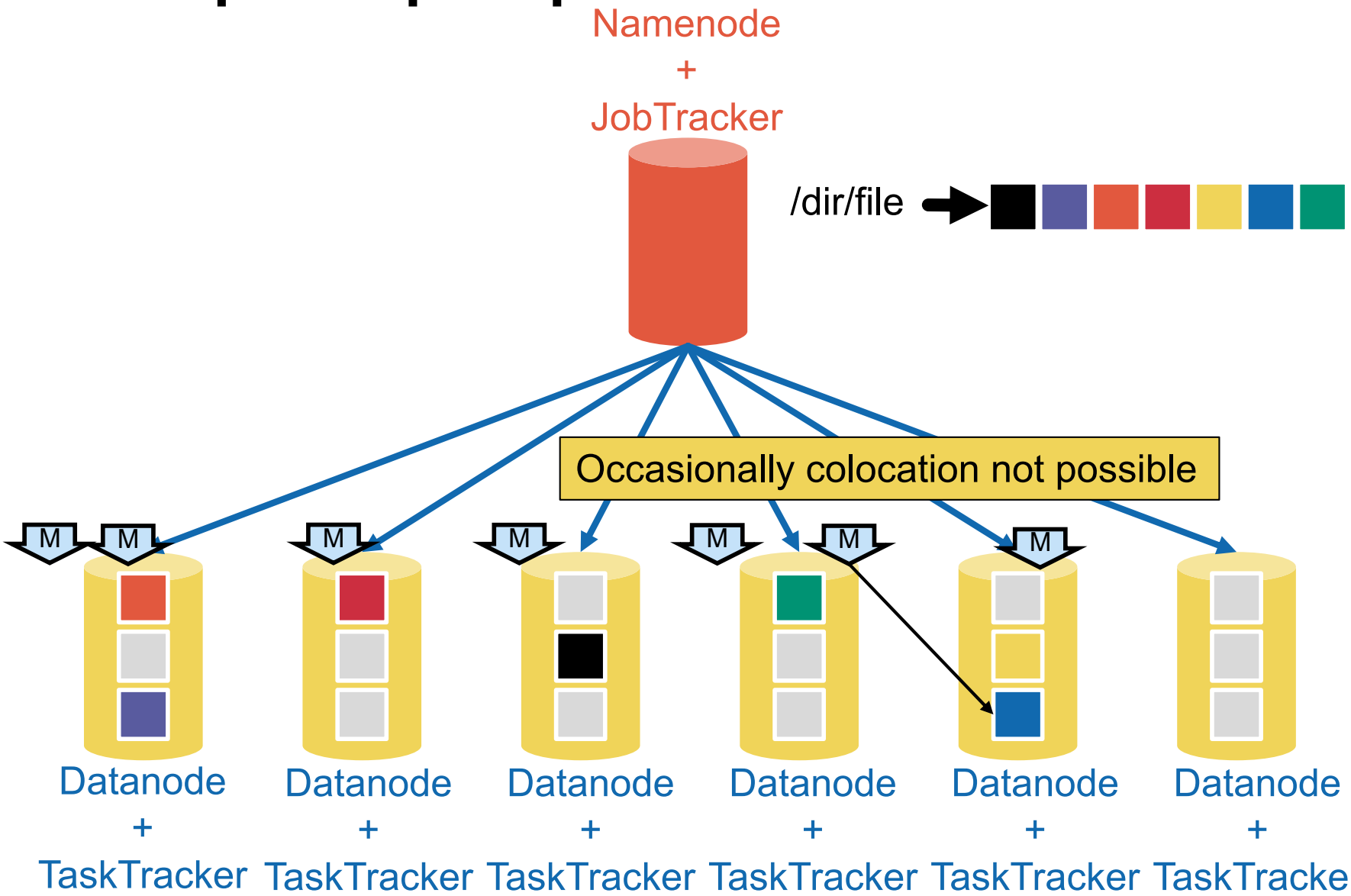
One split per block (generally)



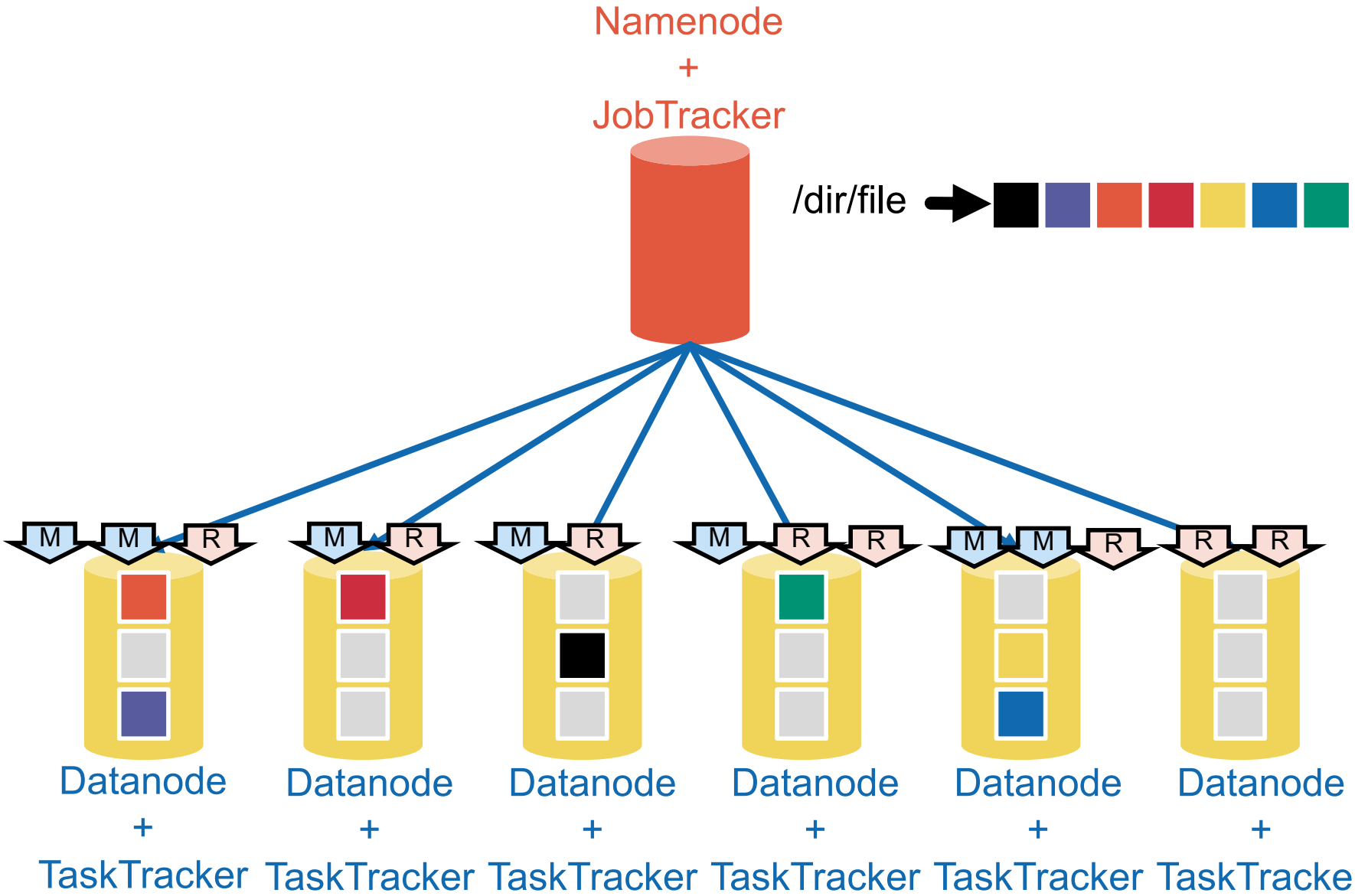
One map task per split



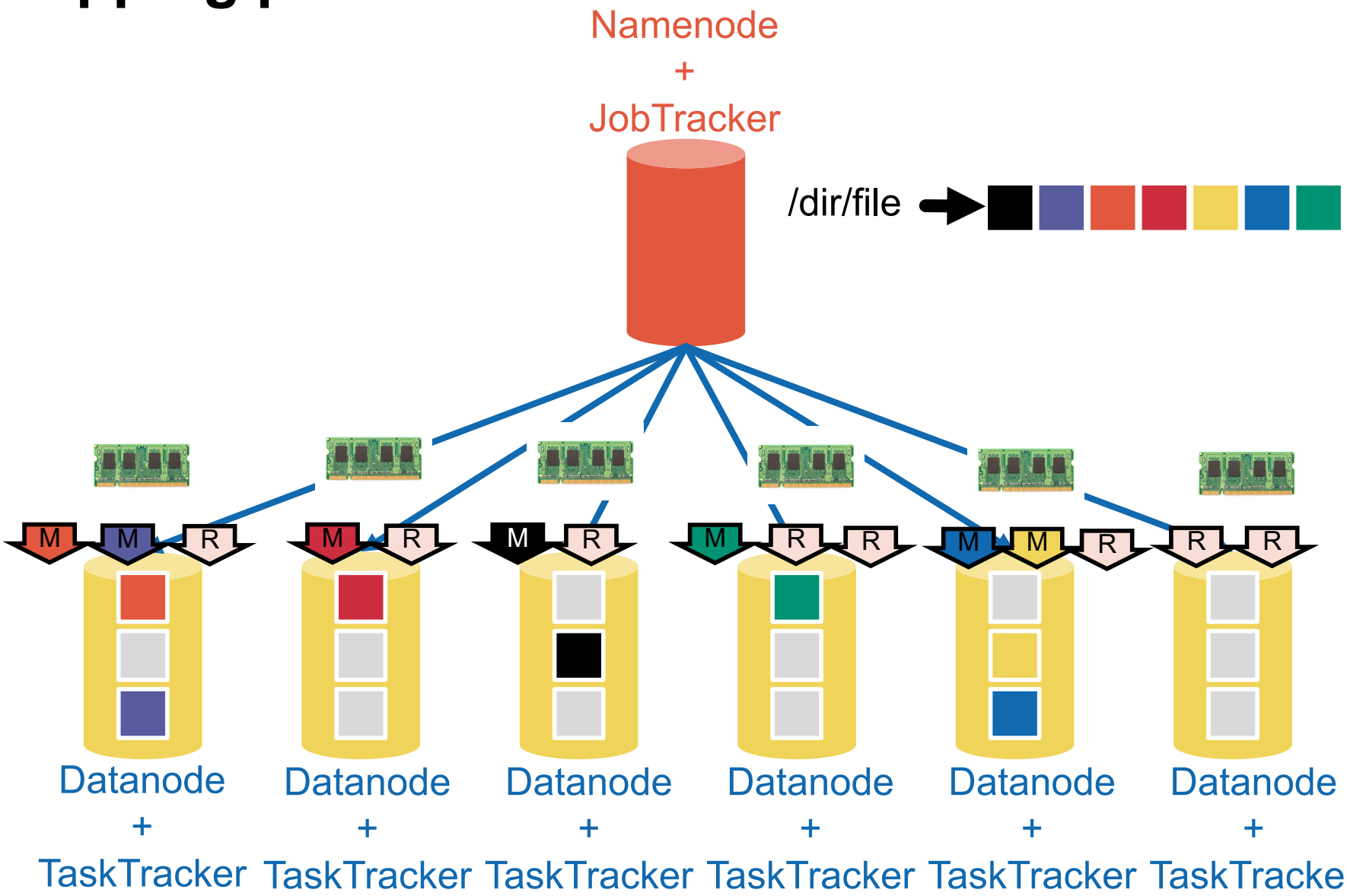
One map task per split



Reduce tasks

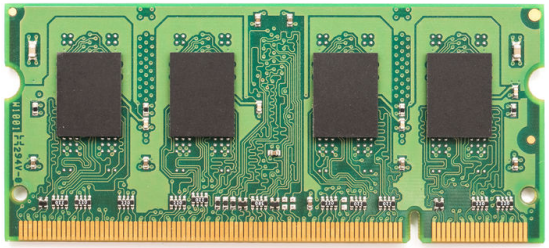


Mapping phase

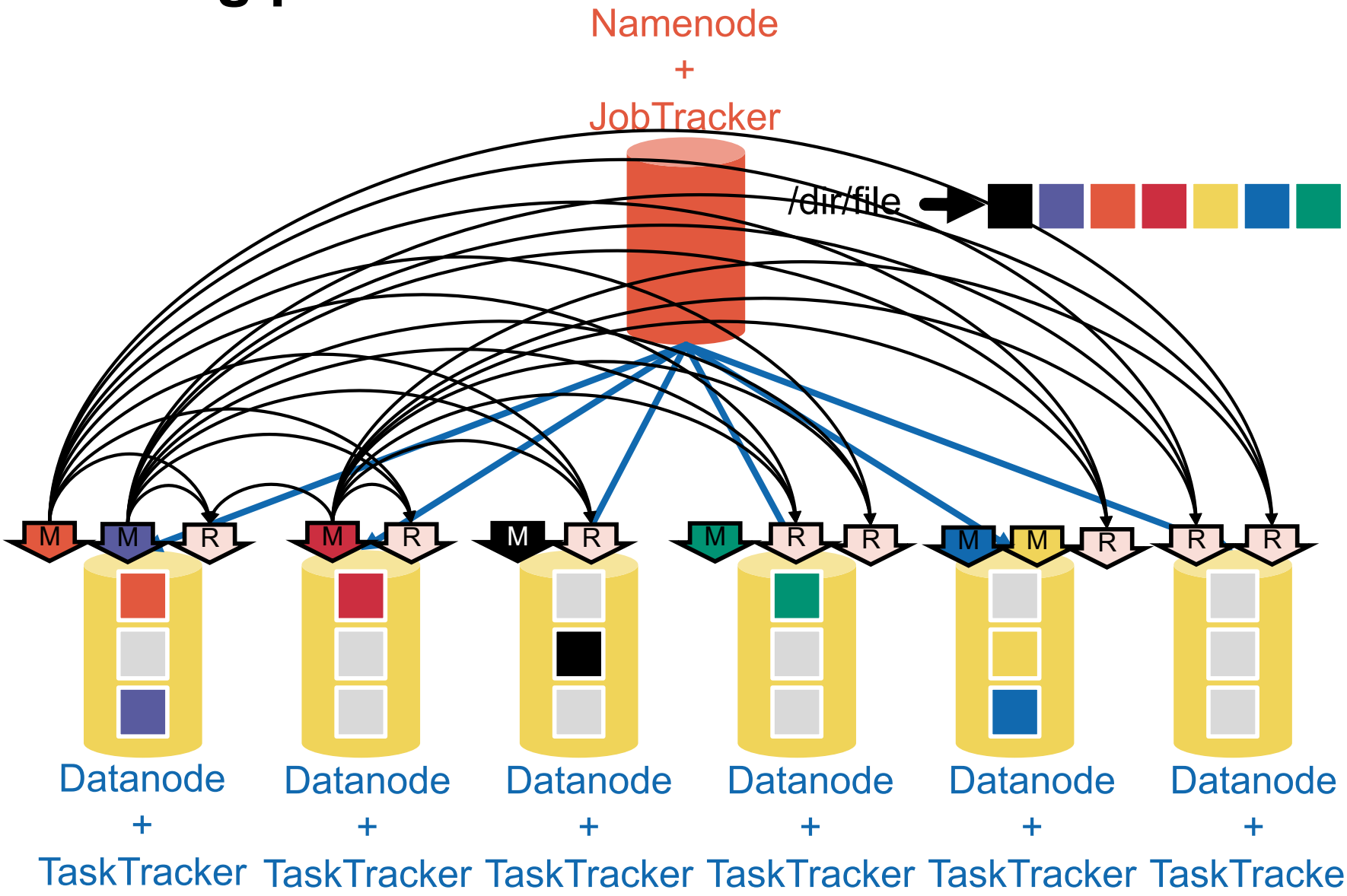


Spilling to disk

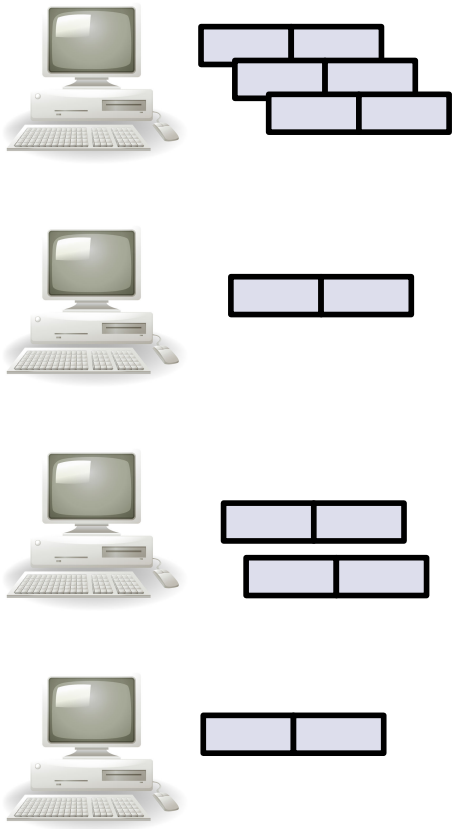
Key-value pairs are spilled to disk if necessary



Shuffling phase



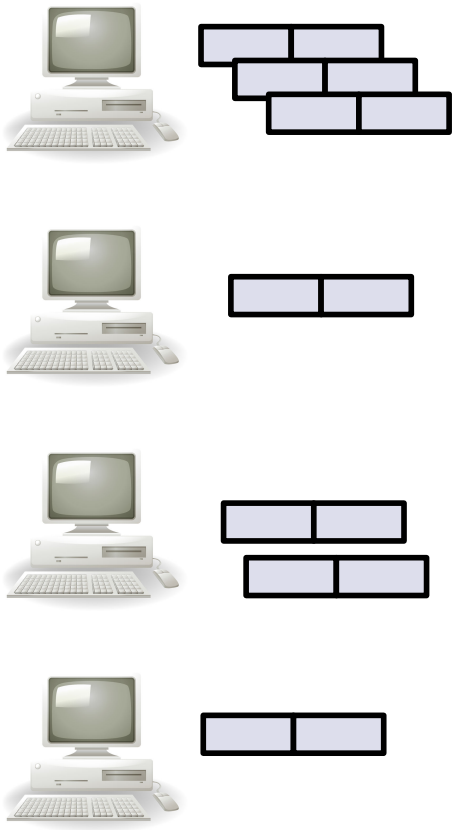
Shuffling phase



Mappers

Reducer

Shuffling phase

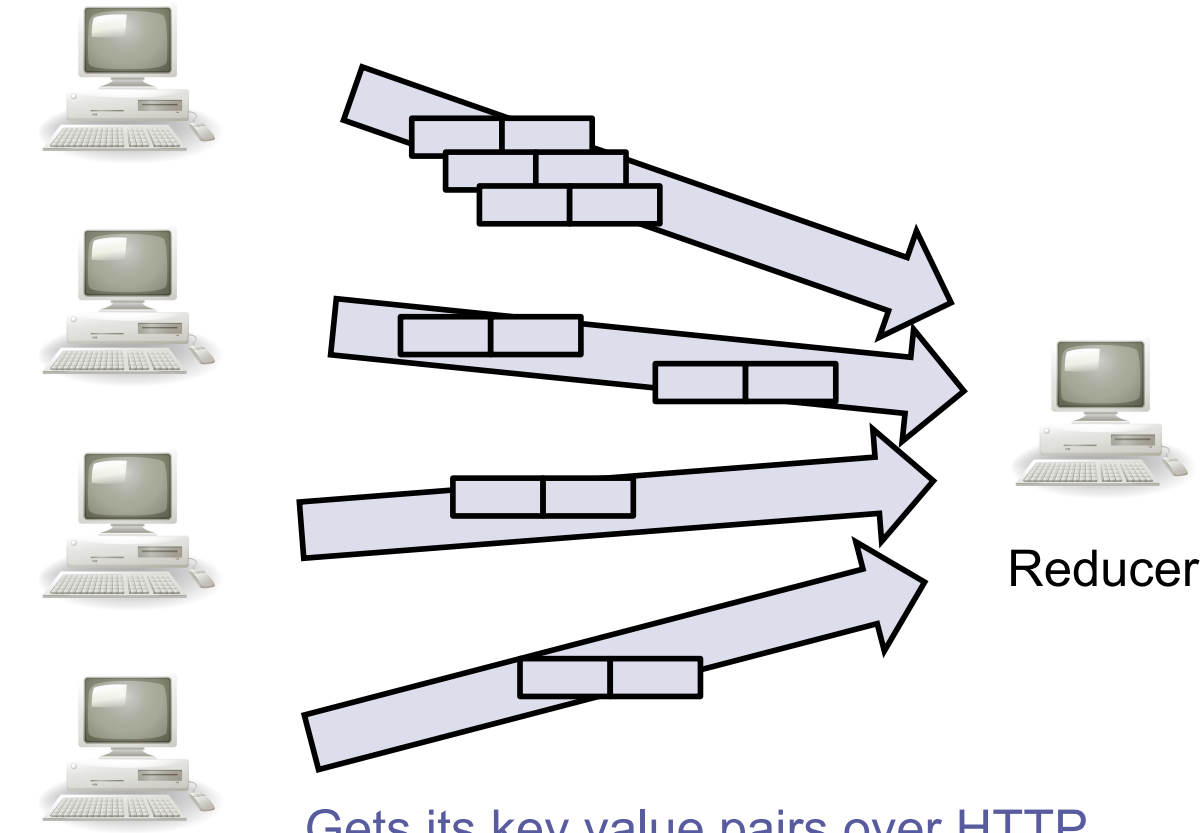


Reducer

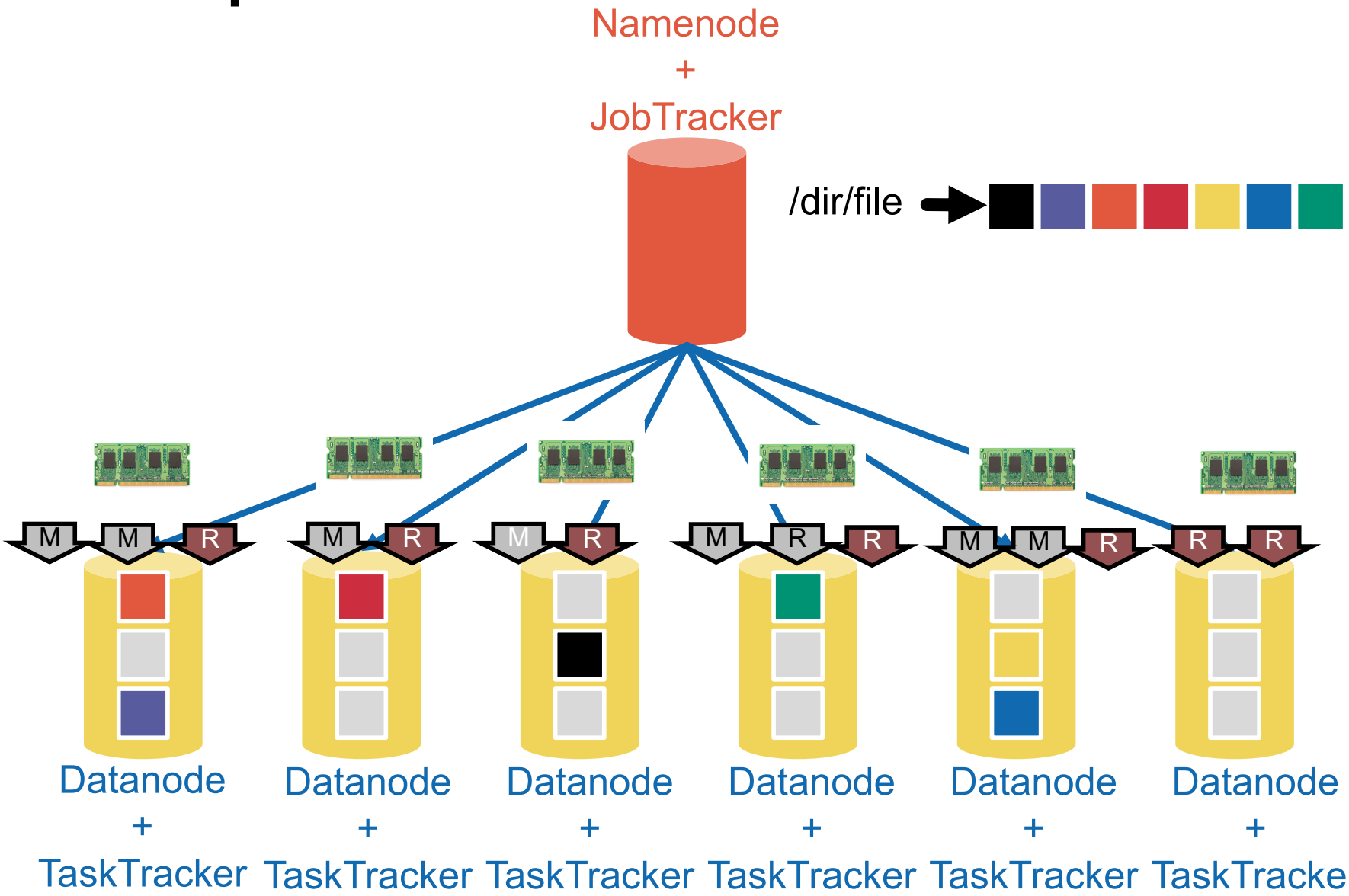
Mappers

Each mapper sorts its output key-value pairs

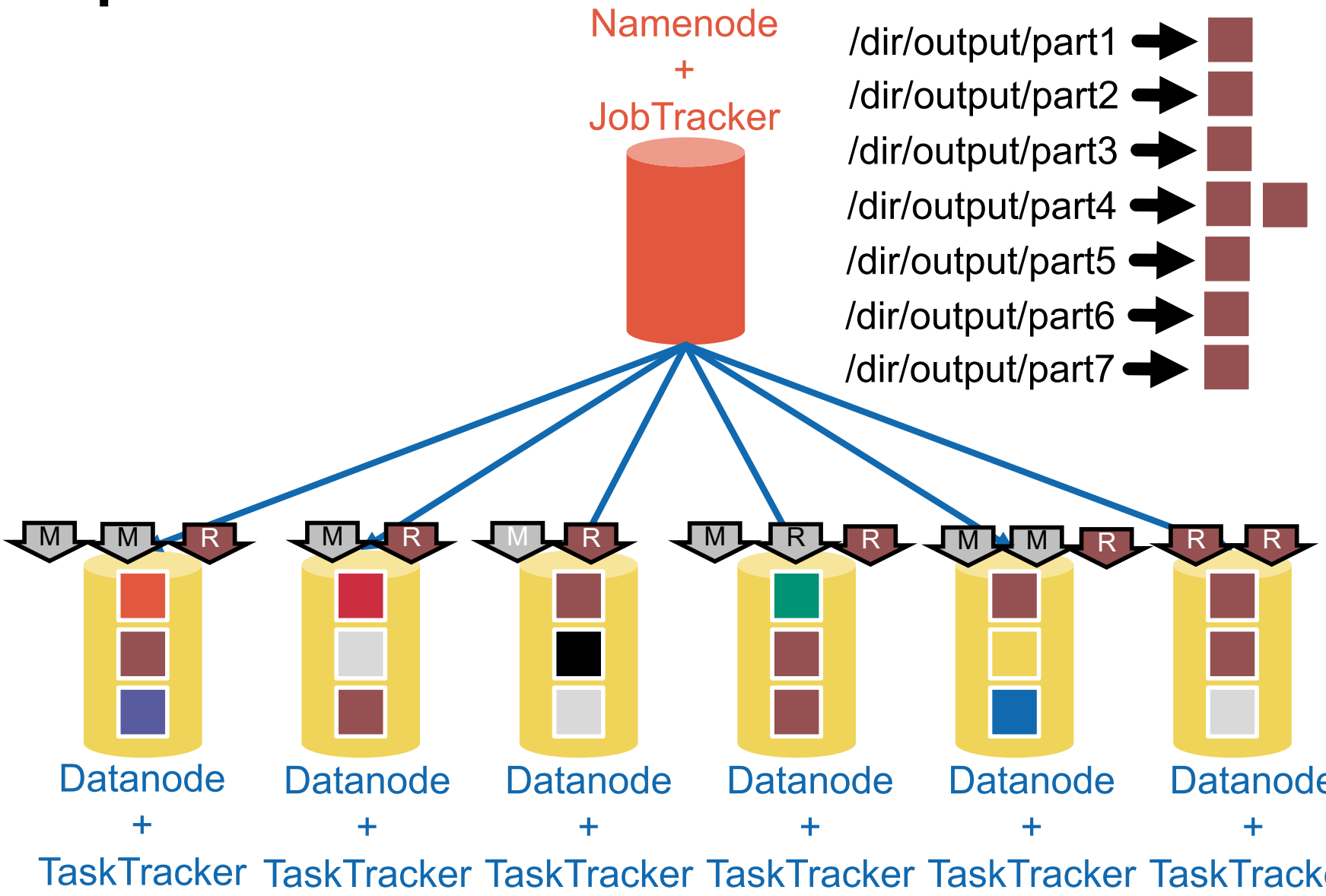
Shuffling phase



Reduce phase



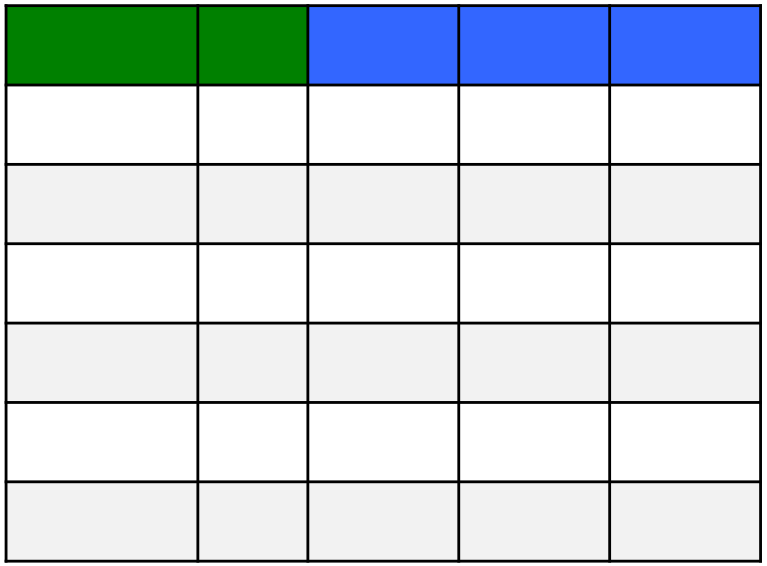
Output to disk



Input/Output formats

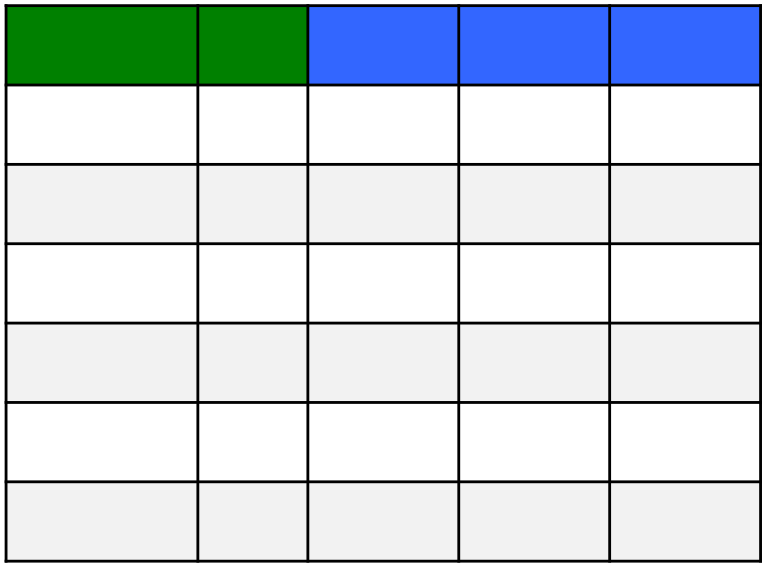
Input and output formats

Input and output formats



From/to tables

Input and output formats

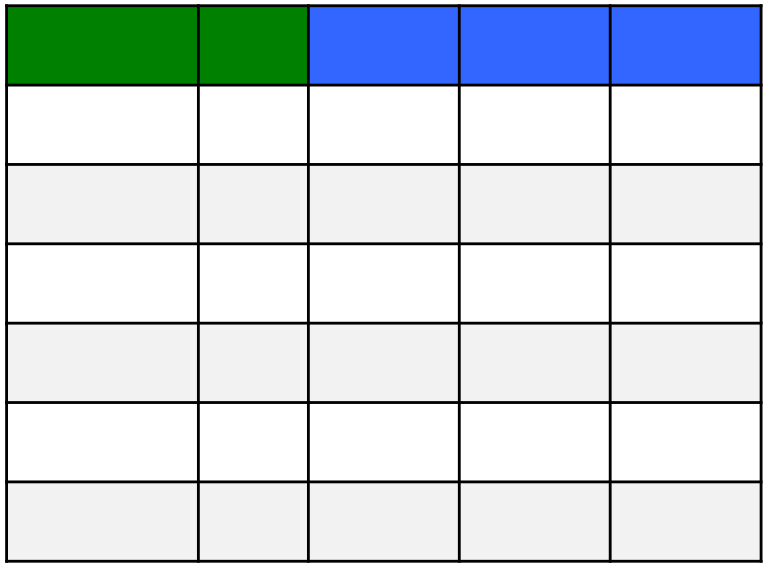


From/to tables

From/to files

Formats: tabular

Formats: tabular



RDBMS

Formats: tabular

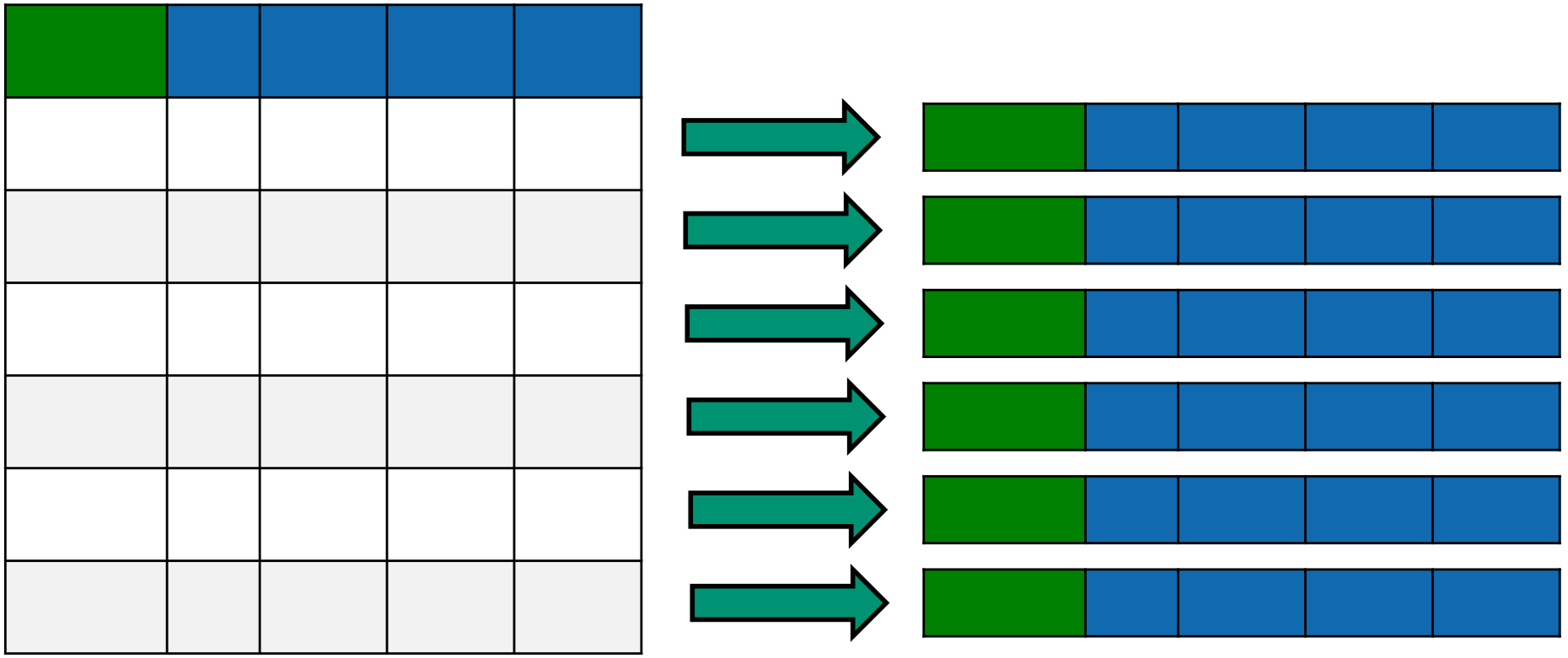
RDBMS

Row ID					
000					
002					
0A1					
1E0					
22A					
4A2					

HBase

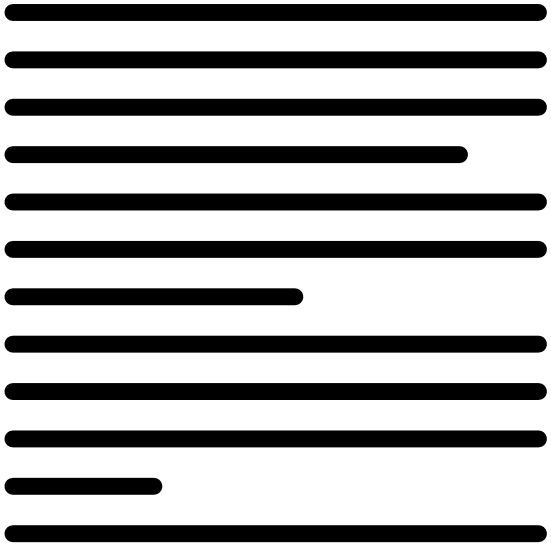
Formats: tabular

Formats: tabular



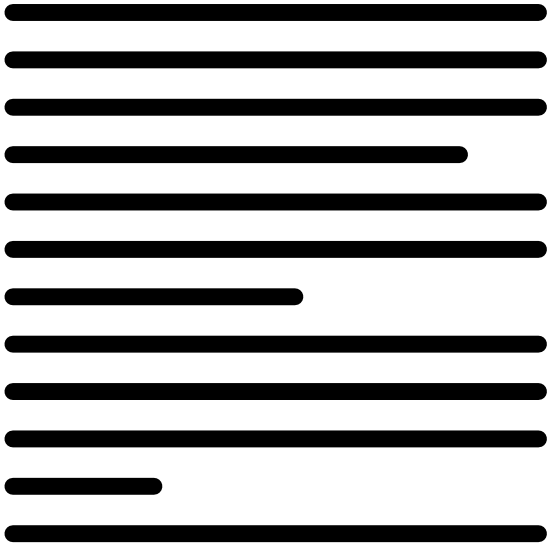
Formats: files (e.g., from HDFS)

Formats: files (e.g., from HDFS)



Text

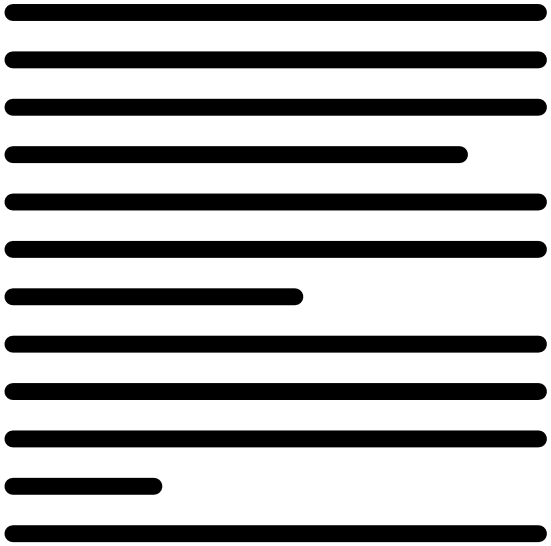
Formats: files (e.g., from HDFS)



Text

KeyValue

Formats: files (e.g., from HDFS)



Text

KeyValue

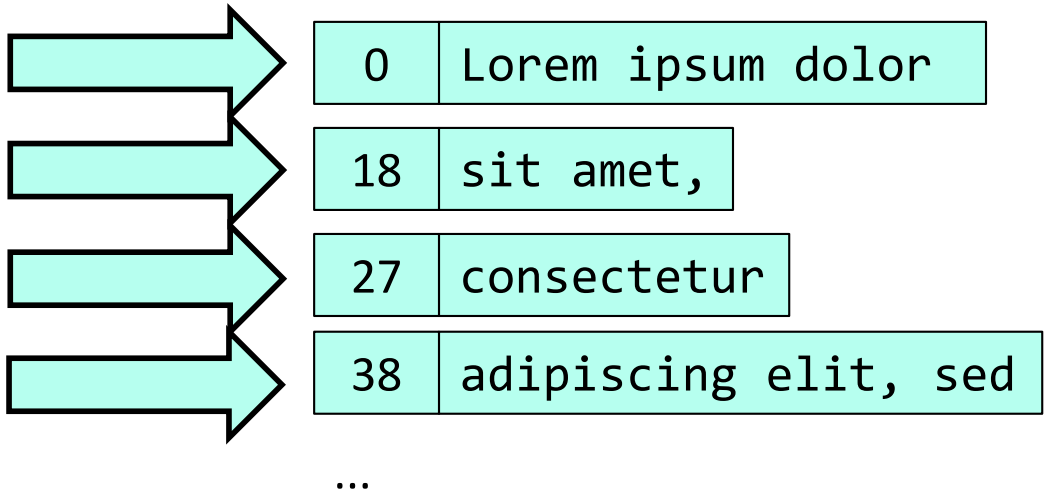
SequenceFile

Text files

Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
...

Text files

Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
...

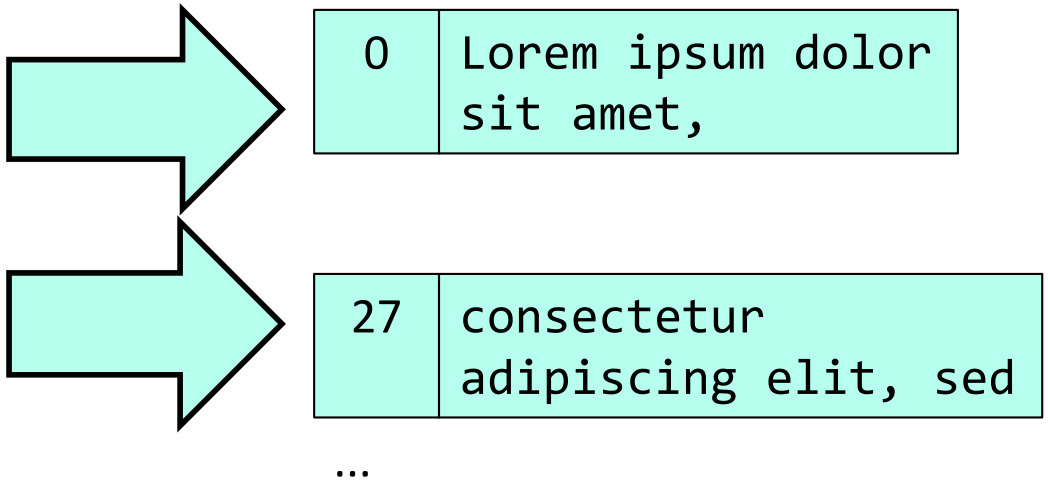


Text files: NLine

Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
...

Text files: NLine

Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
…

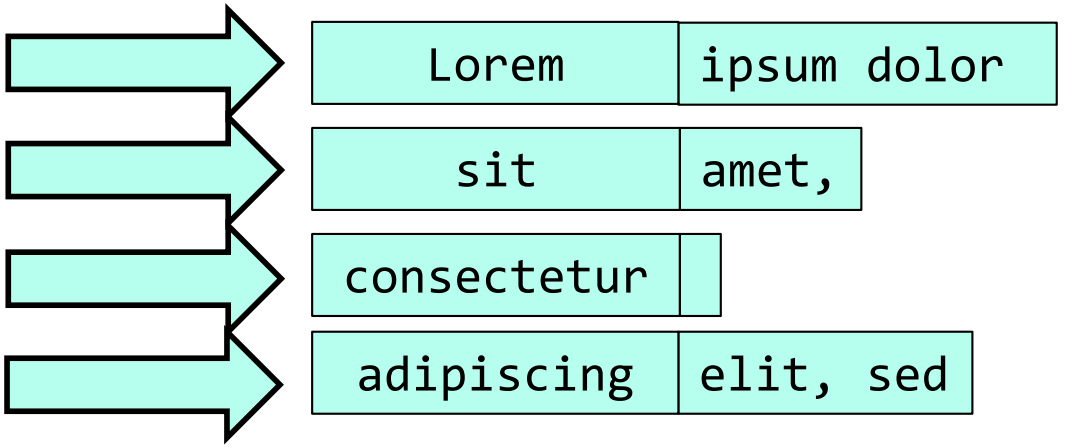


Key-Value

Lorem●ipsum dolor
sit●amet,
consectetur
adipiscing●elit, sed
…

Key-Value

Lorem●ipsum dolor
sit●amet,
consectetur
adipiscing●elit, sed
…



Sequence files

Hadoop binary format

Stores generic key-values

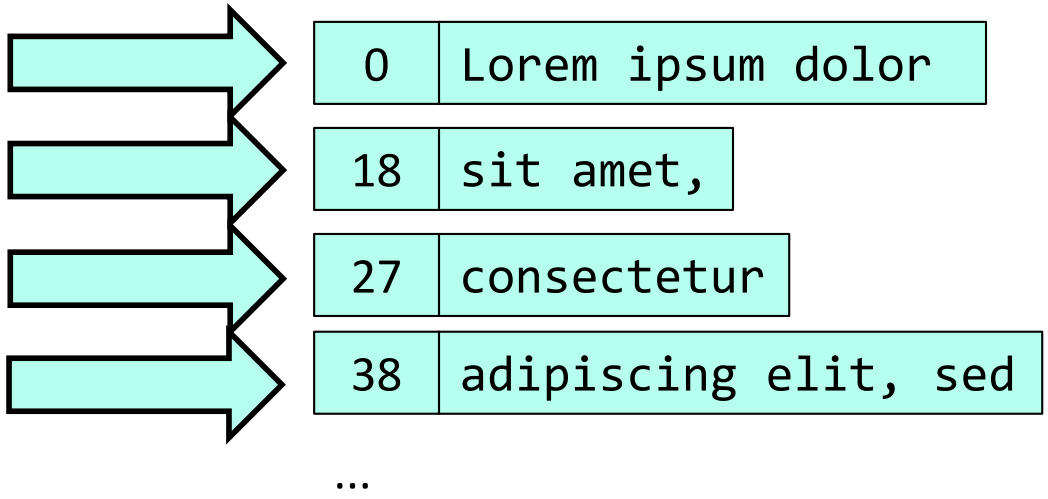
Sequence files

Hadoop binary format

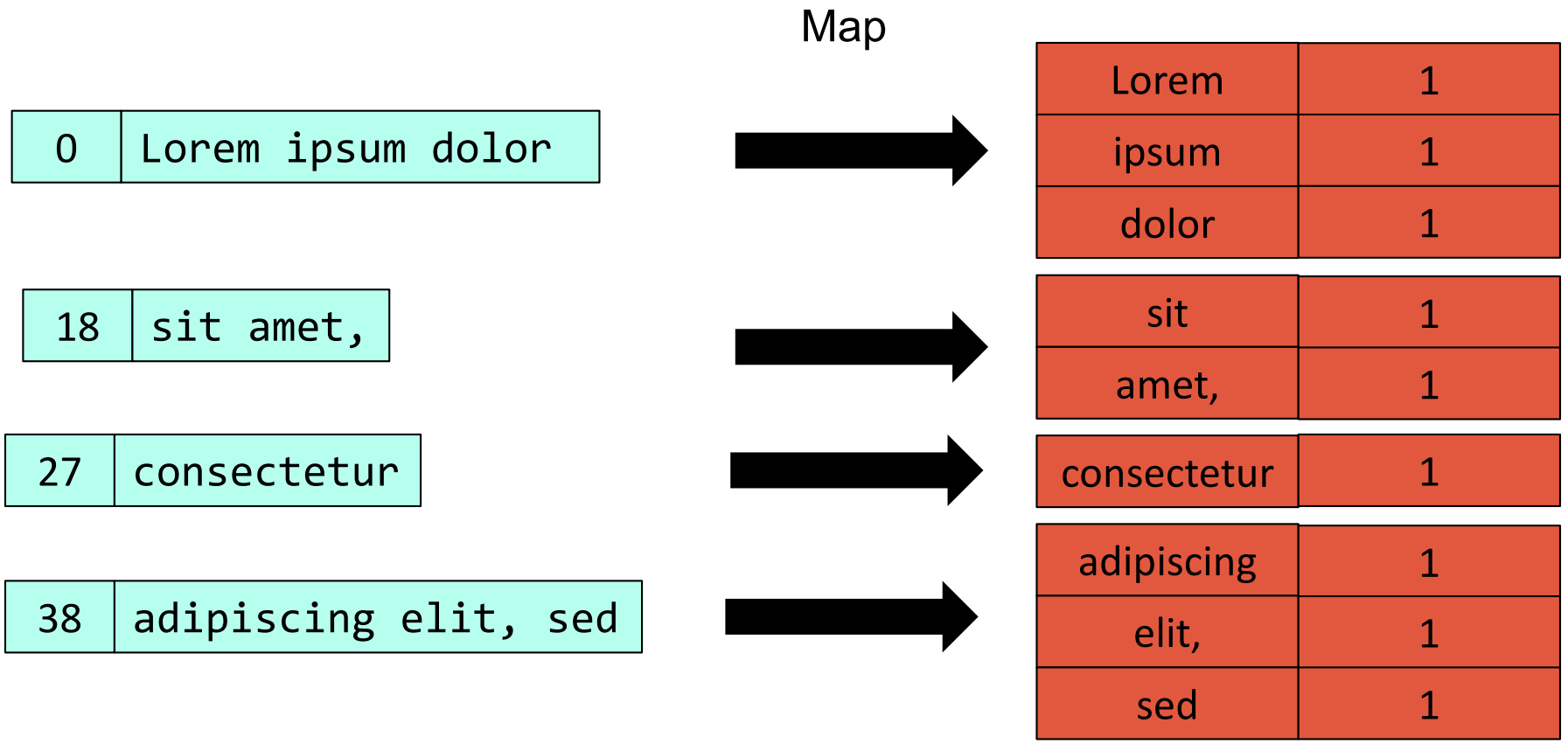
Stores generic key-values

Example: counting words

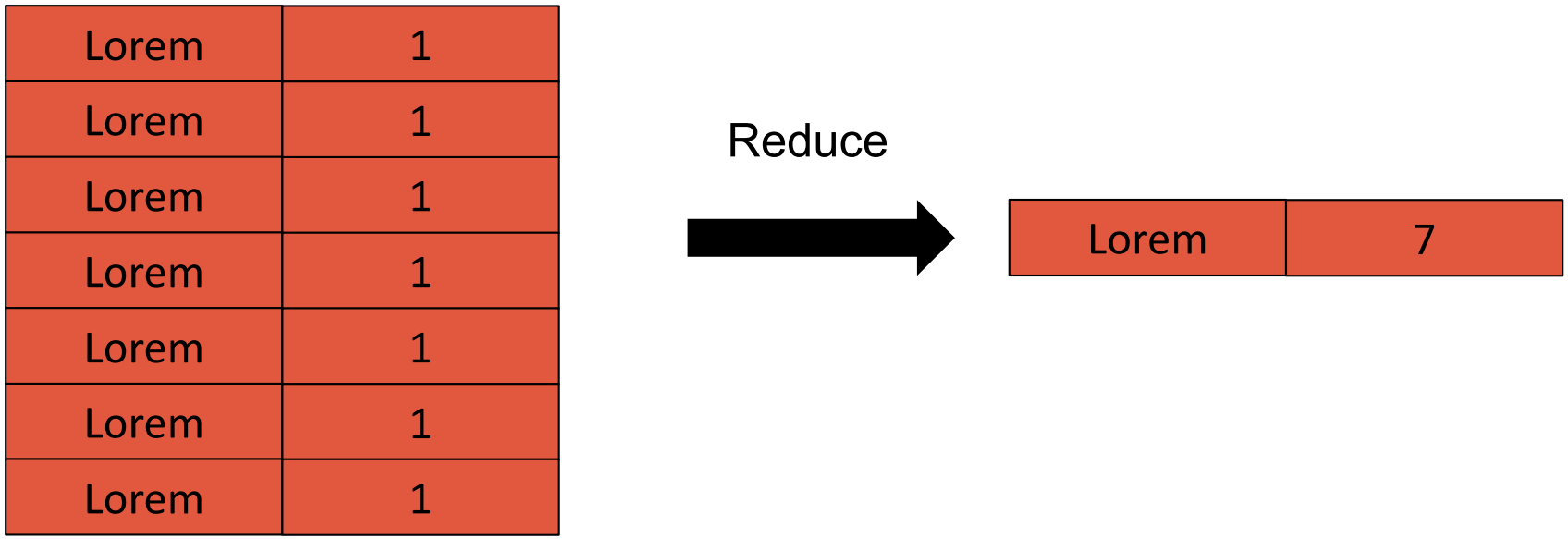
Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
...



Example: counting words



Example: counting words



Example: counting words

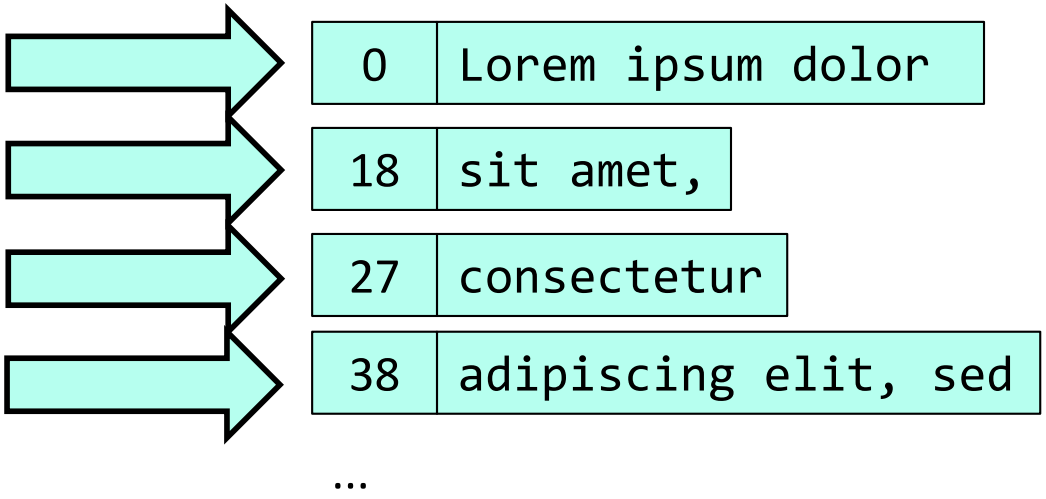
Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1

Combine

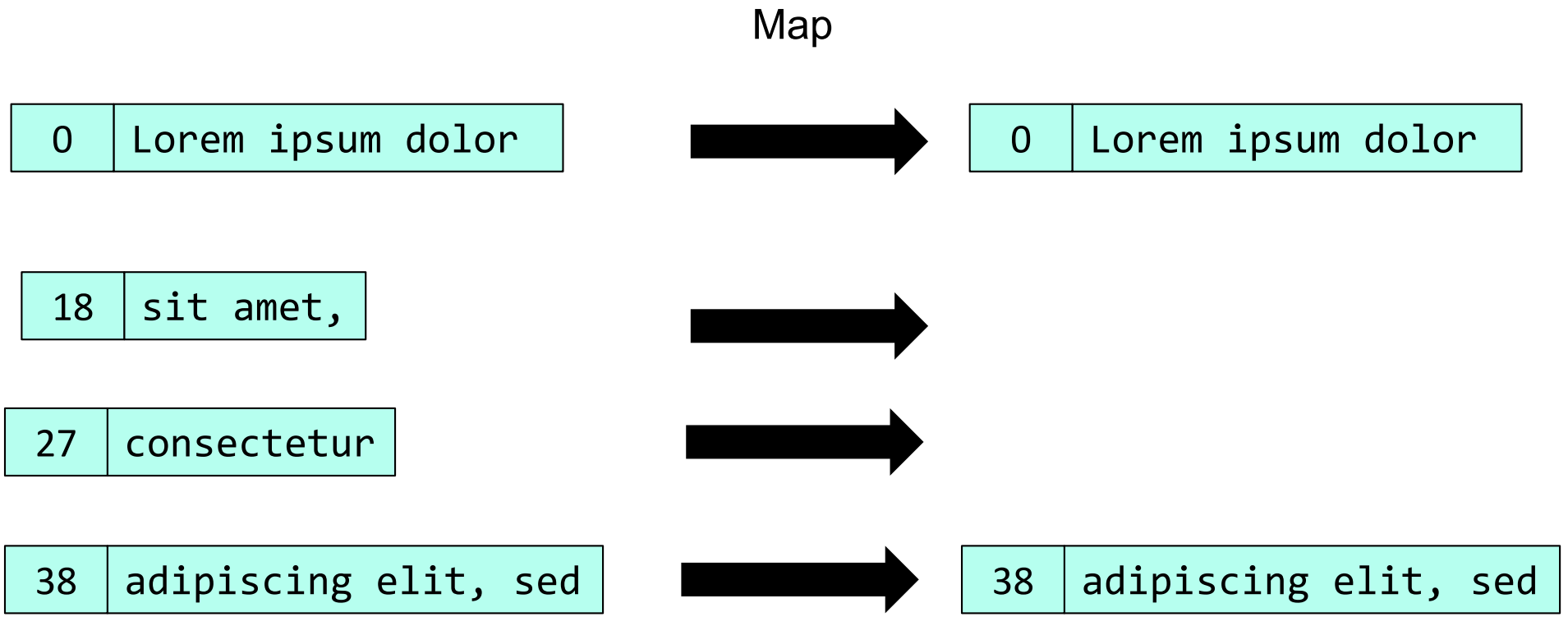
Lorem	7

Example: filtering lines

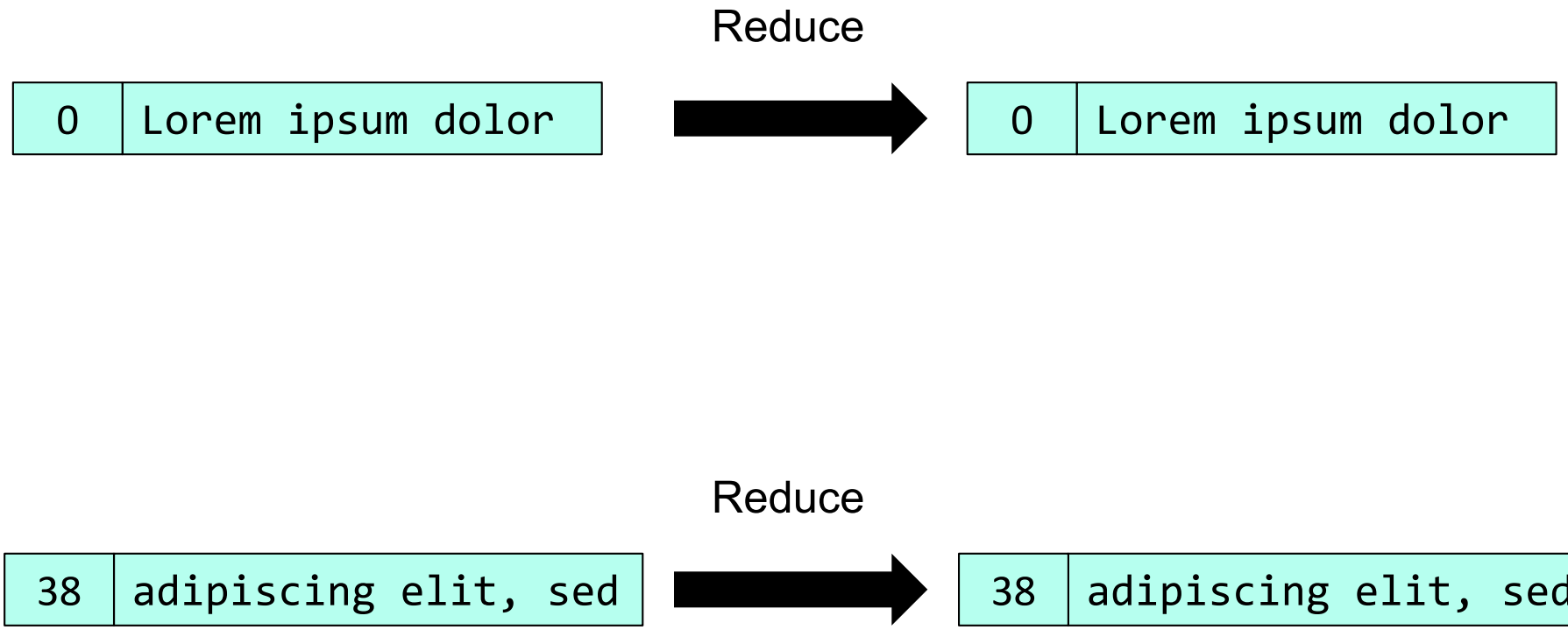
Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
...



Example: filtering lines



Example: filtering lines



Example: projecting

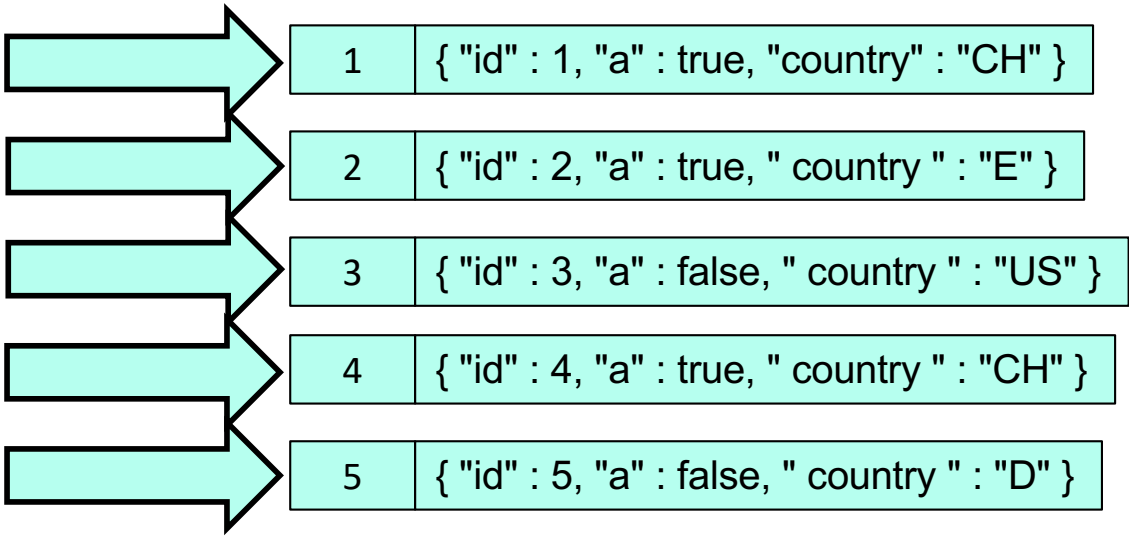
{ "id" : 1, "a" : true, "country" : "CH" }

{ "id" : 2, "a" : true, " country " : "E" }

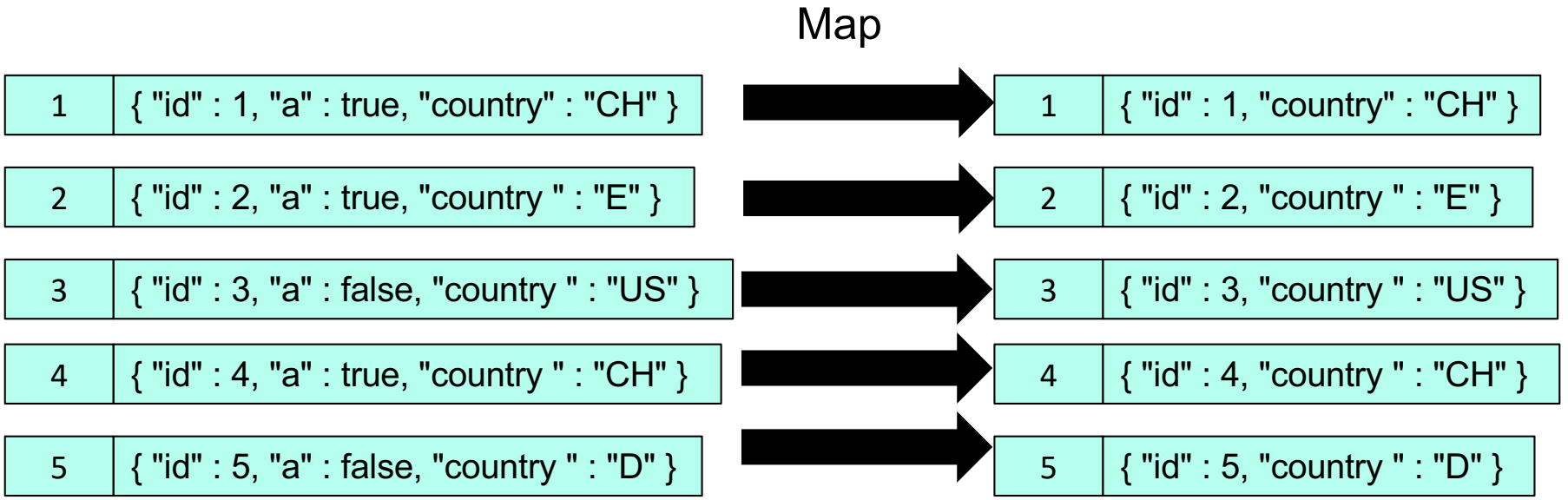
{ "id" : 3, "a" : false, " country " : "US" }

{ "id" : 4, "a" : true, " country " : "CH" }

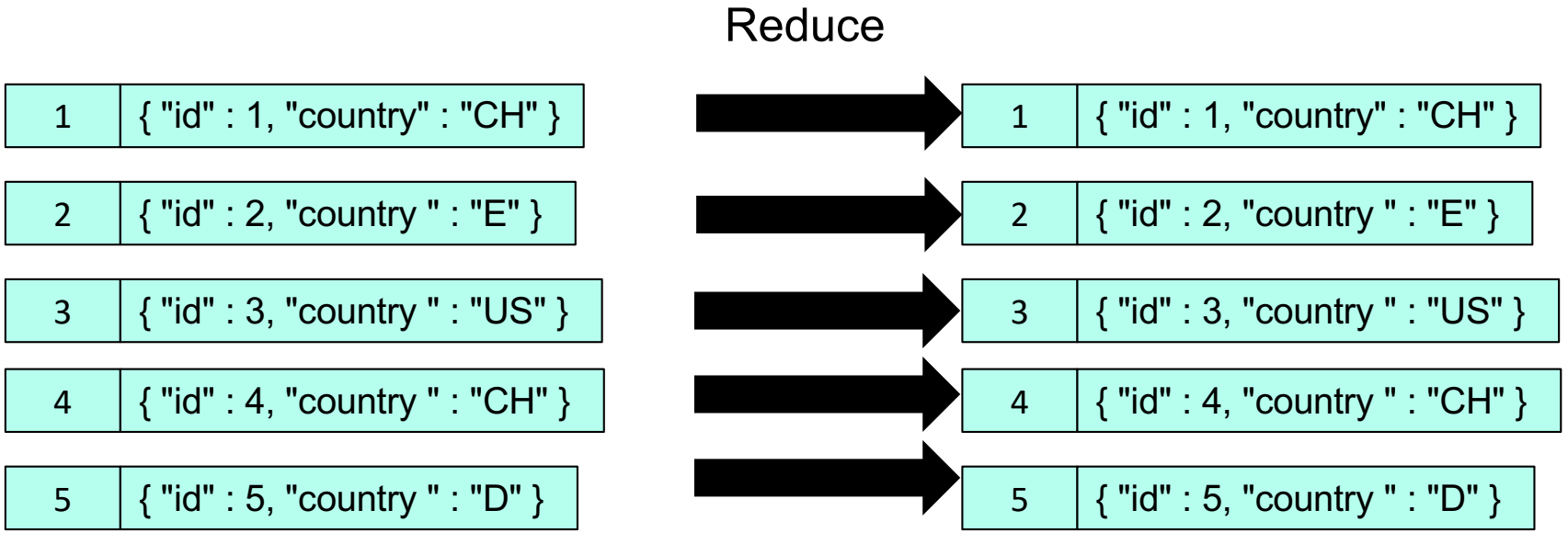
{ "id" : 5, "a" : false, " country " : "D" }



Example: projecting

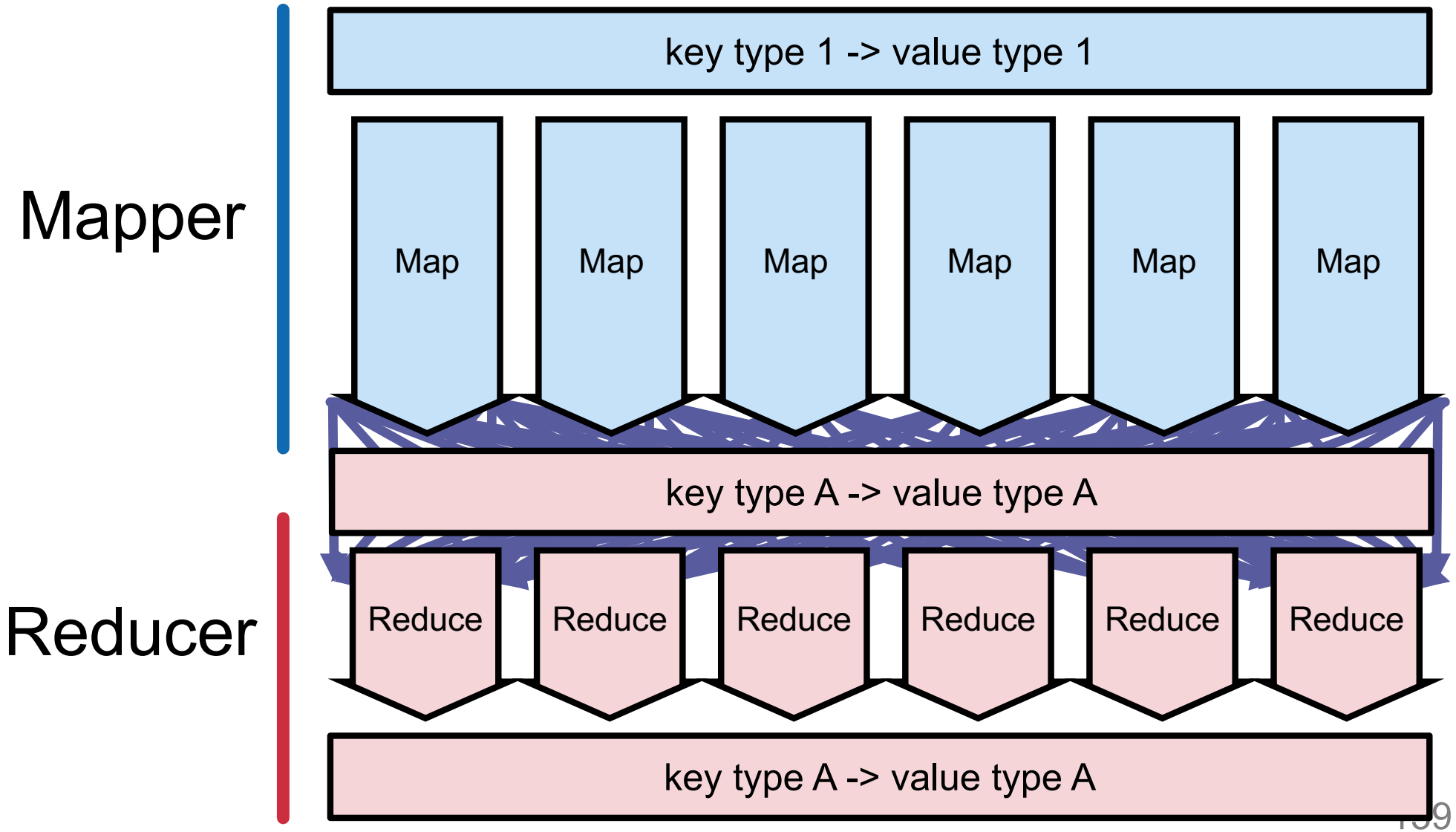


Example: projecting

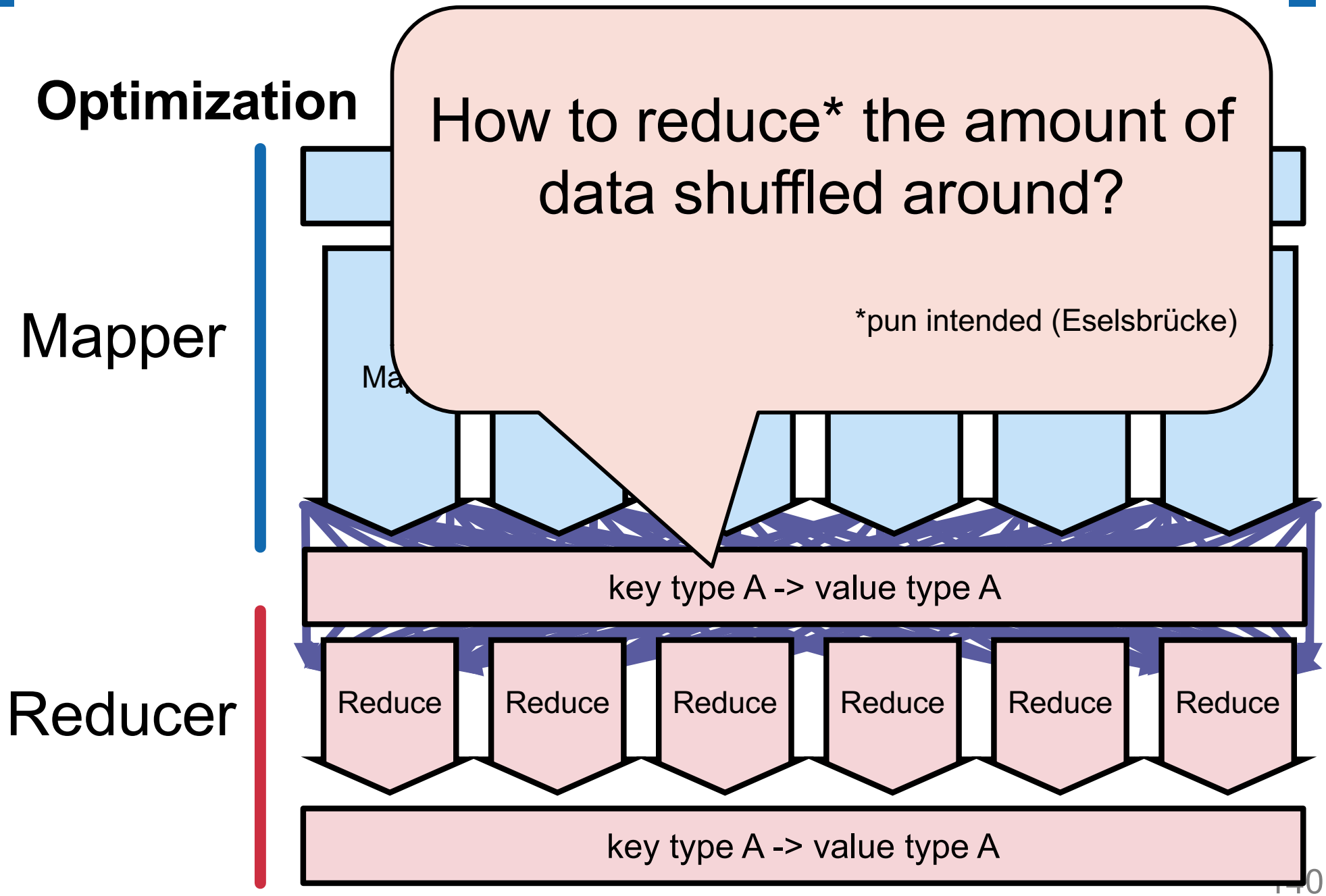


Optimization

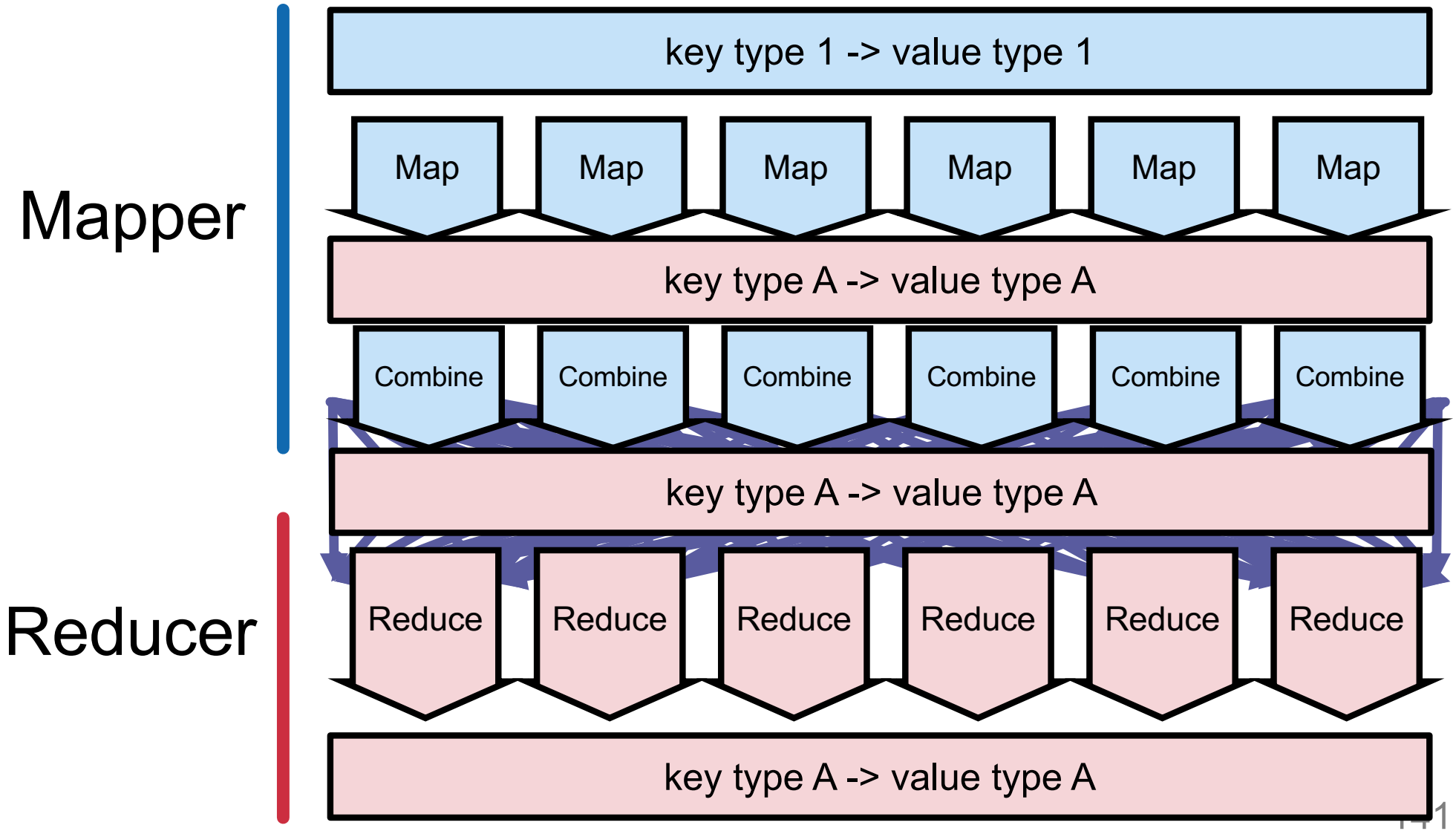
Optimization



Optimization

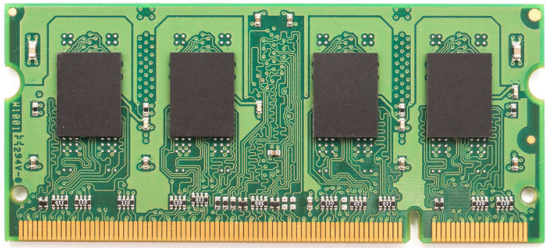


Optimization: Combine



Spilling to disk

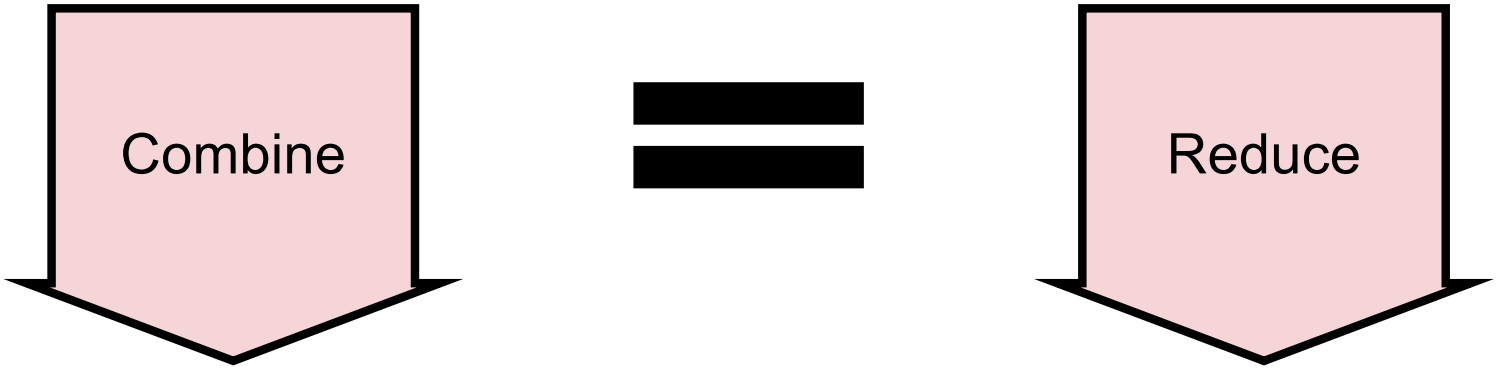
Combine
when spilled
to Disk



Combine: the 90% case

Combine: the 90% case

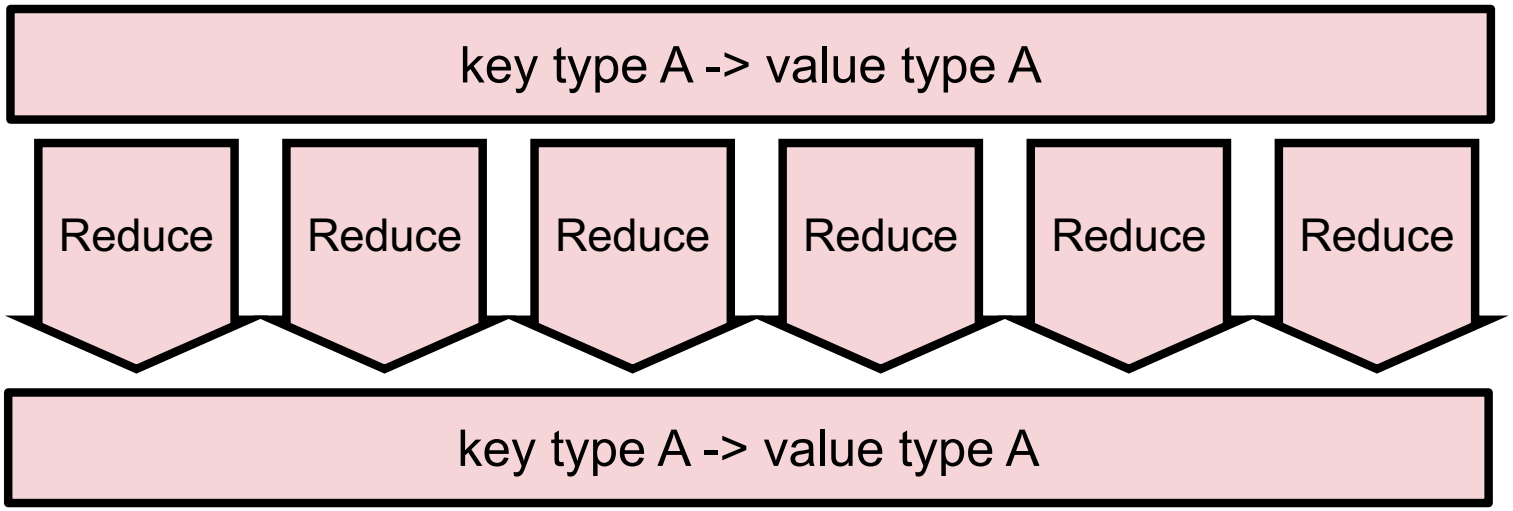
Often, the **combine** function is identical to the **reduce** function.



Disclaimer: there are assumptions

Combine=Reduce: Assumption 1

Key/Value types must be **identical** for reduce input and output.

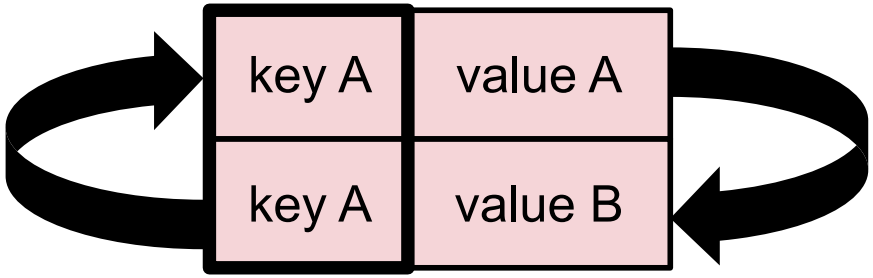


Combine=Reduce : Assumption 2

Combine=Reduce : Assumption 2

Reduce function must be

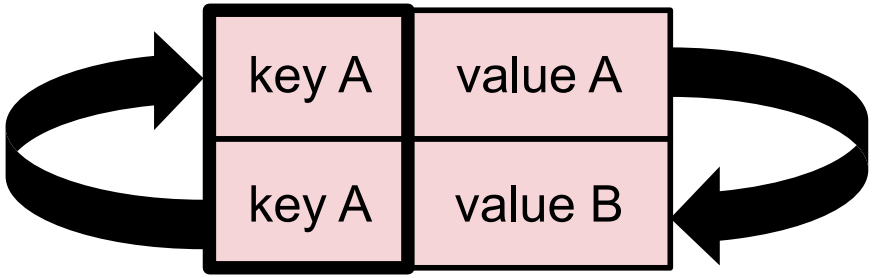
Commutative



Combine=Reduce : Assumption 2

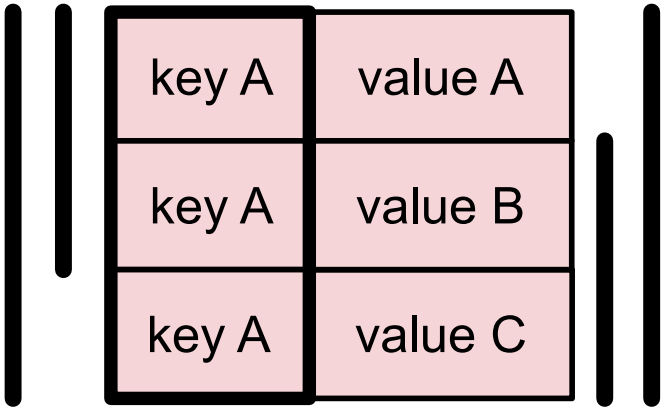
Reduce function must be

Commutative

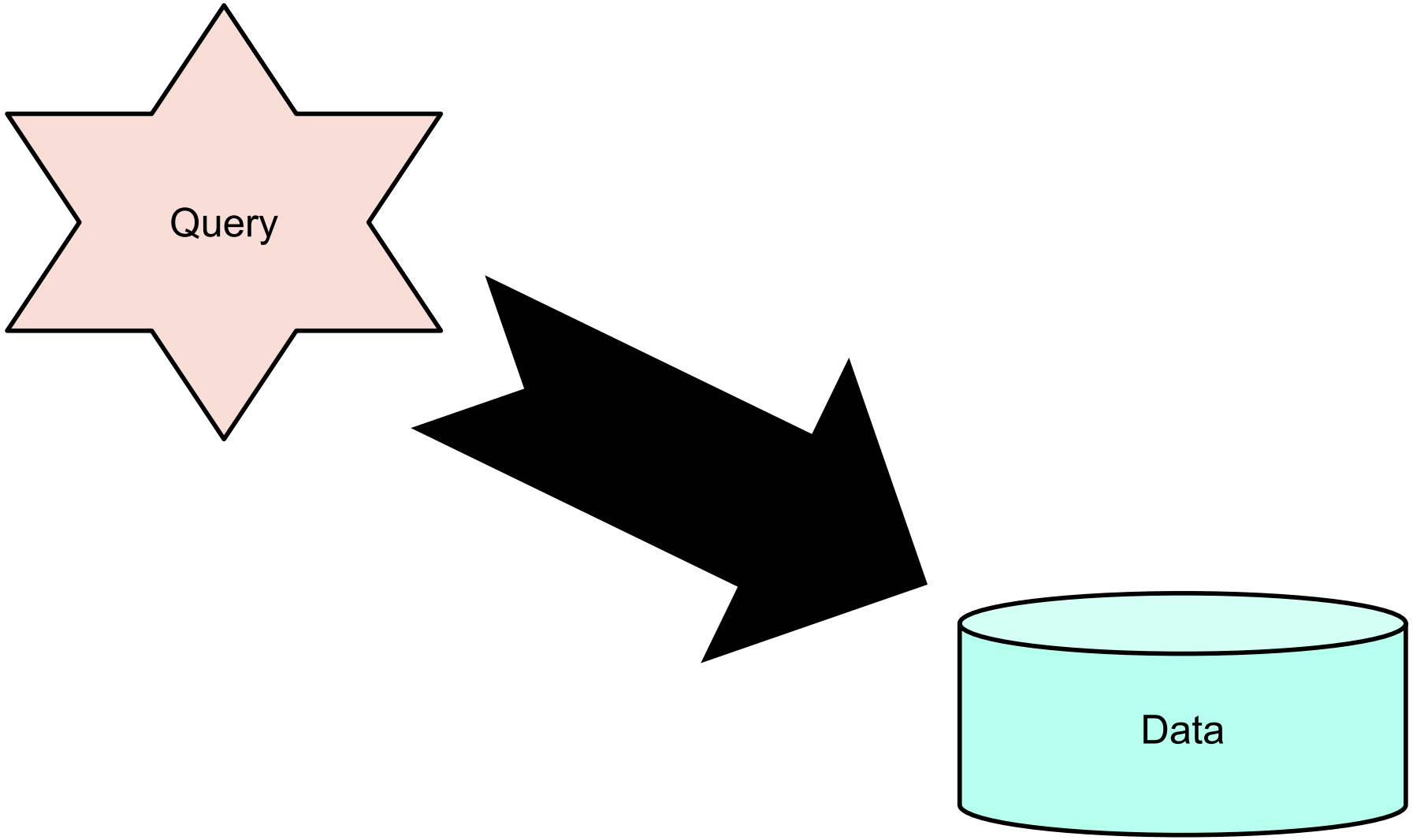


and

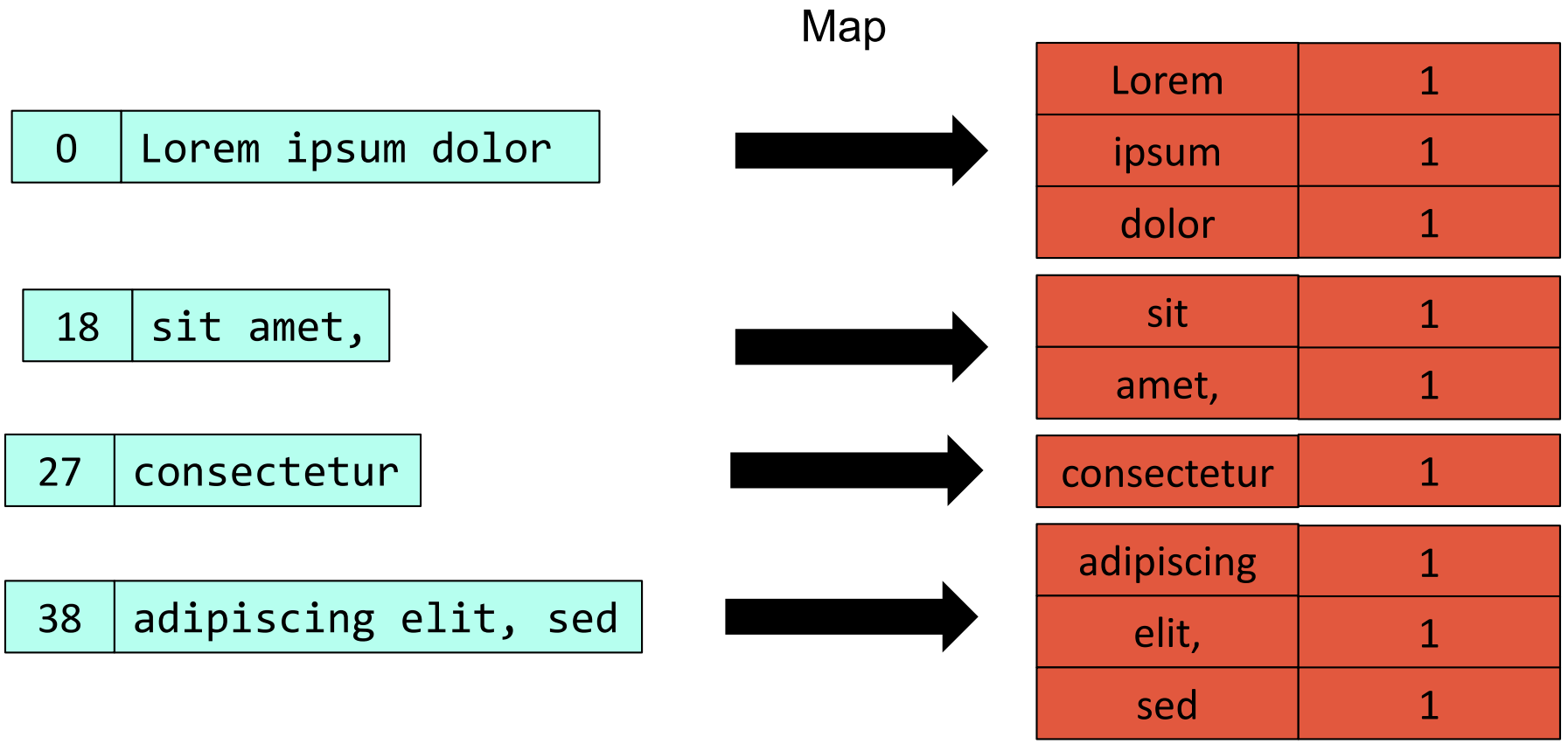
Associative



Optimization: Bring the Query to the Data



Example: counting words



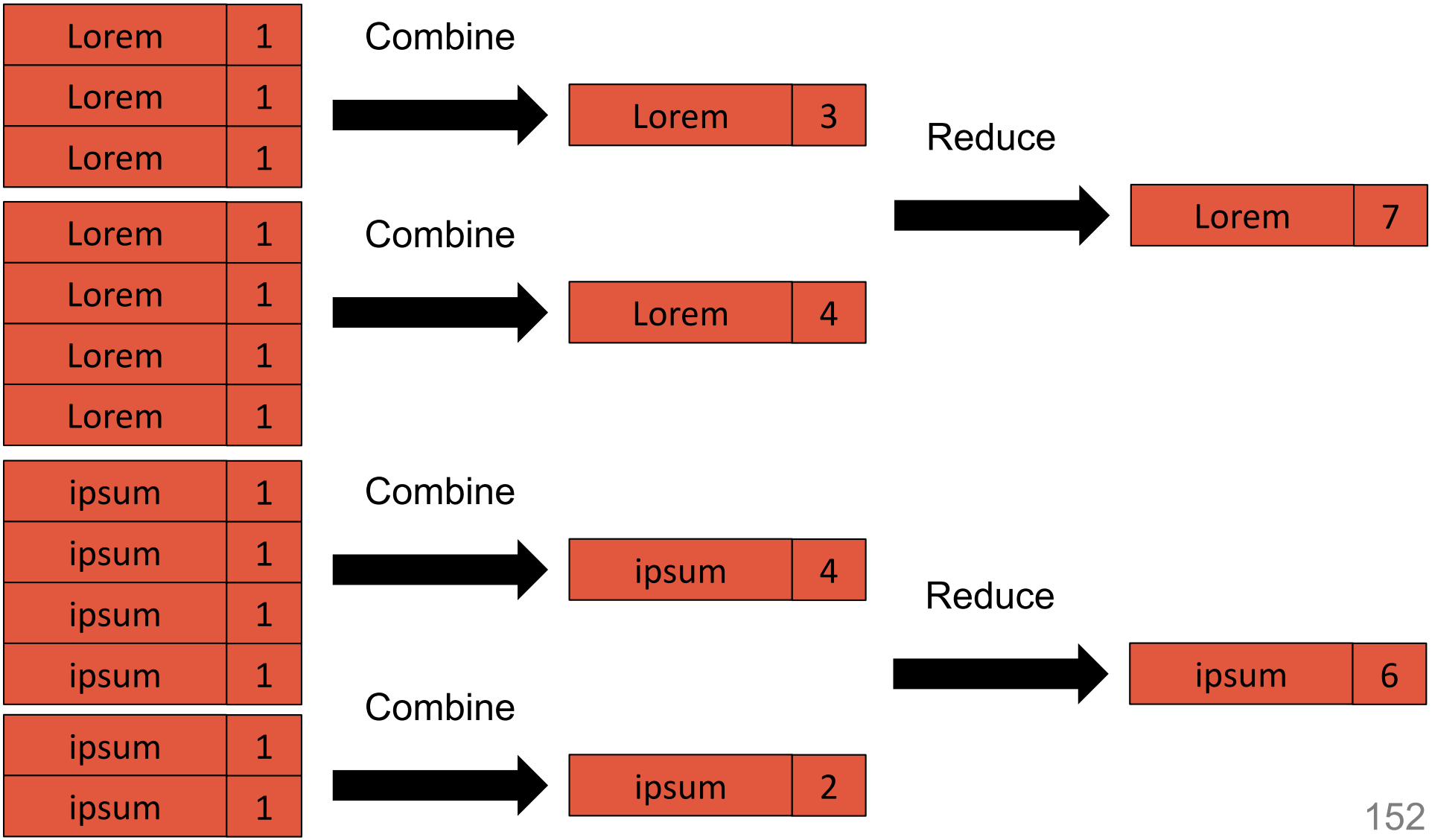
Example: counting words

Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1
Lorem	1

Lorem	7

- Commutative
- Associative

Example: counting words

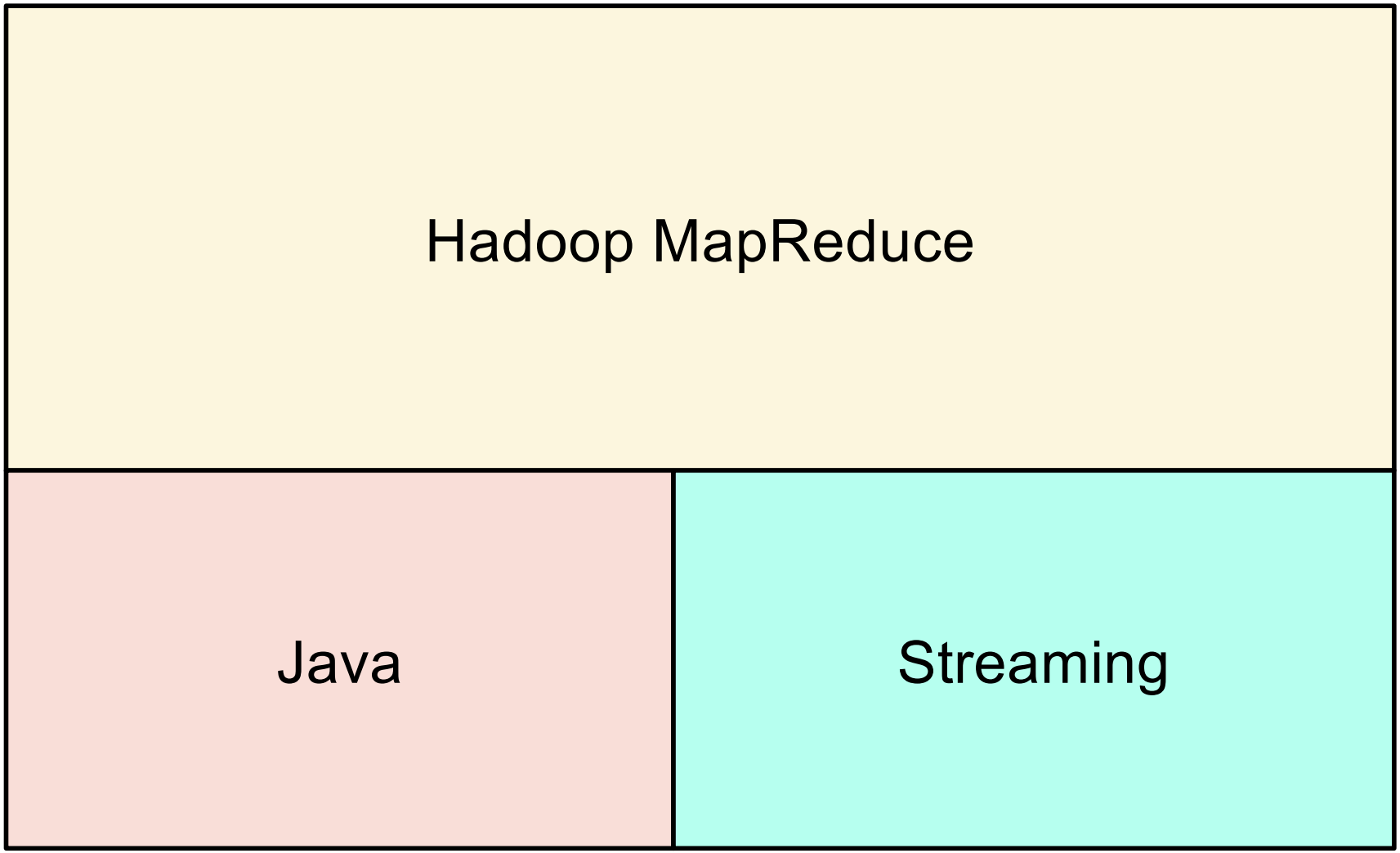


MapReduce: the APIs

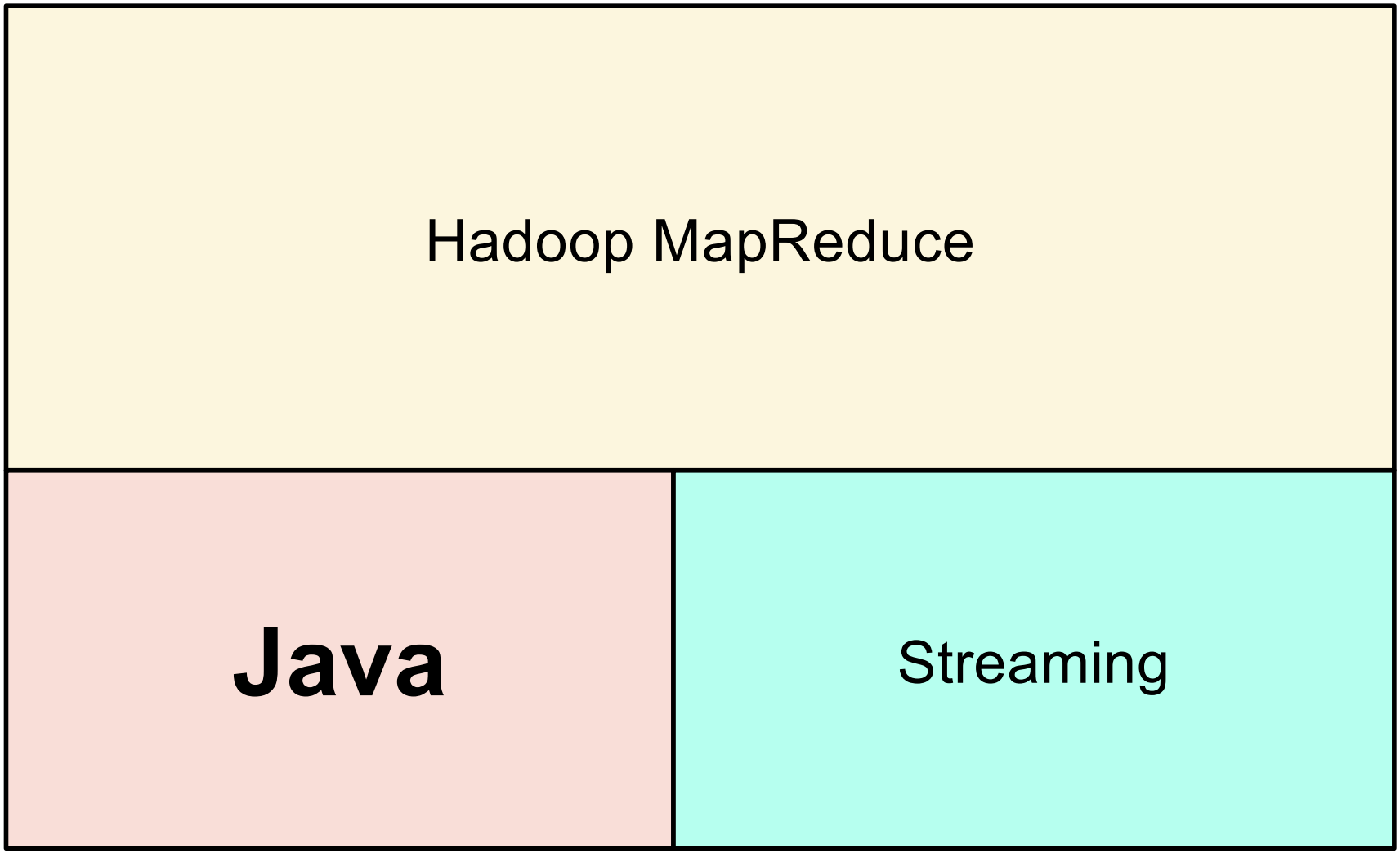
Supported frameworks

Hadoop MapReduce

Supported frameworks



Supported frameworks



Java API: Mapper

```
import org.apache.hadoop.mapreduce.Mapper;

public class MyOwnMapper extends Mapper<K1, V1, K2, V2>{

}
```

Java API: Mapper

```
import org.apache.hadoop.mapreduce.Mapper;

public class MyOwnMapper extends Mapper<K1, V1, K2, V2>{

  public void map(K1 key, V1 value, Context context)
    throws IOException, InterruptedException
  {




  }

}
```

Java API: Mapper

```
import org.apache.hadoop.mapreduce.Mapper;

public class MyOwnMapper extends Mapper<K1, V1, K2, V2>{

  public void map(K1 key, V1 value, Context context)
    throws IOException, InterruptedException
  {
    ...
    K2 new-key = ...
    V2 new-value = ...
    context.write(new-key, new-value);
    ...
  }

}
```

Java API: Reducer

```
import org.apache.hadoop.mapreduce.Reducer;

public class MyOwnReducer extends Reducer<K2, V2, K3, V3>{

}
```

Java API: Reducer

```
import org.apache.hadoop.mapreduce.Reducer;

public class MyOwnReducer extends Reducer<K2, V2, K3, V3>{

  public void reduce
    (K2 key, Iterable<V2> values, Context context)
    throws IOException, InterruptedException
  {









  }

}
```

Java API: Reducer

```
import org.apache.hadoop.mapreduce.Reducer;

public class MyOwnReducer extends Reducer<K2, V2, K3, V3>{

  public void reduce
    (K2 key, Iterable<V2> values, Context context)
    throws IOException, InterruptedException
  {
    ...
    K3 new-key = ...
    V3 new-value = ...
    context.write(new-key, new-value);
    ...
  }

}
```

Java API: Job

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {









}
```

Java API: Job

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");









}
```

Java API: Job

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");

     job.setMapperClass(MyOwnMapper.class);
     job.setReducerClass(MyOwnReducer.class);




}
```

Java API: Job

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");

     job.setMapperClass(MyOwnMapper.class);
     job.setReducerClass(MyOwnReducer.class);

     FileInputFormat.addInputPath(job, ...);
     FileOutputFormat.setOutputPath(job, ...);


}
```

Java API: Job

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");

     job.setMapperClass(MyOwnMapper.class);
     job.setReducerClass(MyOwnReducer.class);

     FileInputFormat.addInputPath(job, ...);
     FileOutputFormat.setOutputPath(job, ...);

     System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Java API: Combiner (=Reducer)

```
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");

     job.setMapperClass(MyOwnMapper.class);
     job.setCombinerClass(MyOwnReducer.class);
     job.setReducerClass(MyOwnReducer.class);

     FileInputFormat.addInputPath(job, ...);
     FileOutputFormat.setOutputPath(job, ...);

     System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Java API: InputFormat classes

InputFormat

- DBInputFormat — RDBMS
- TableInputFormat — HBase
- **FileInputFormat**
  - KeyValueTextInputFormat — Key value file
  - SequenceFileInputFormat — Sequence file
  - TextInputFormat — Text
  - FixedLengthInputFormat
  - NLineInputFormat

Java API: OutputFormat classes

OutputFormat

- DBOutputFormat — RDBMS
- TableOutputFormat — HBase
- **FileoutputFormat**
  - SequenceFileOutputFormat — Sequence file
  - TextOutputFormat — Text
  - MapFileOutputFormat

Tricky point: blocks and splits

Tasks

Task

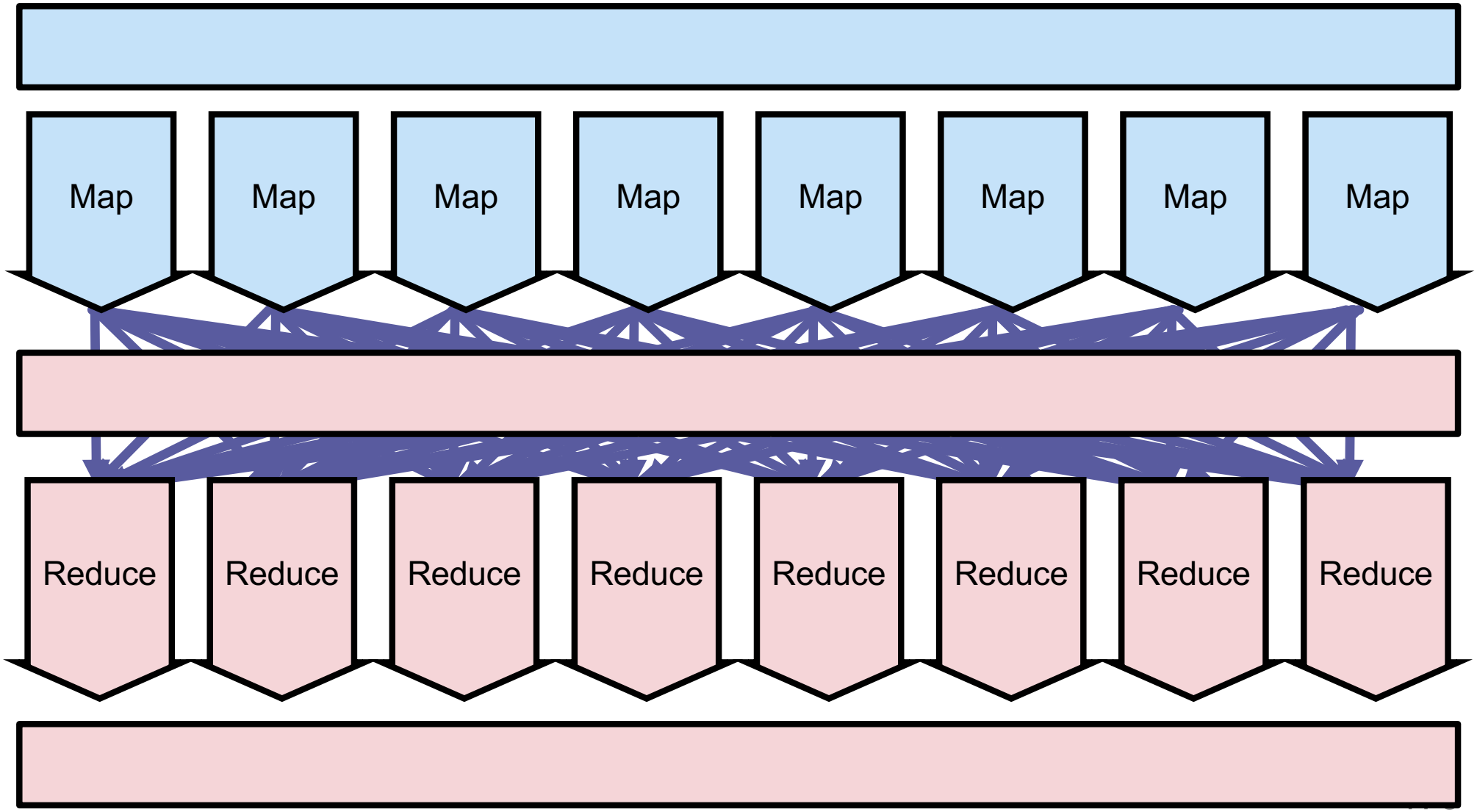
=

Map

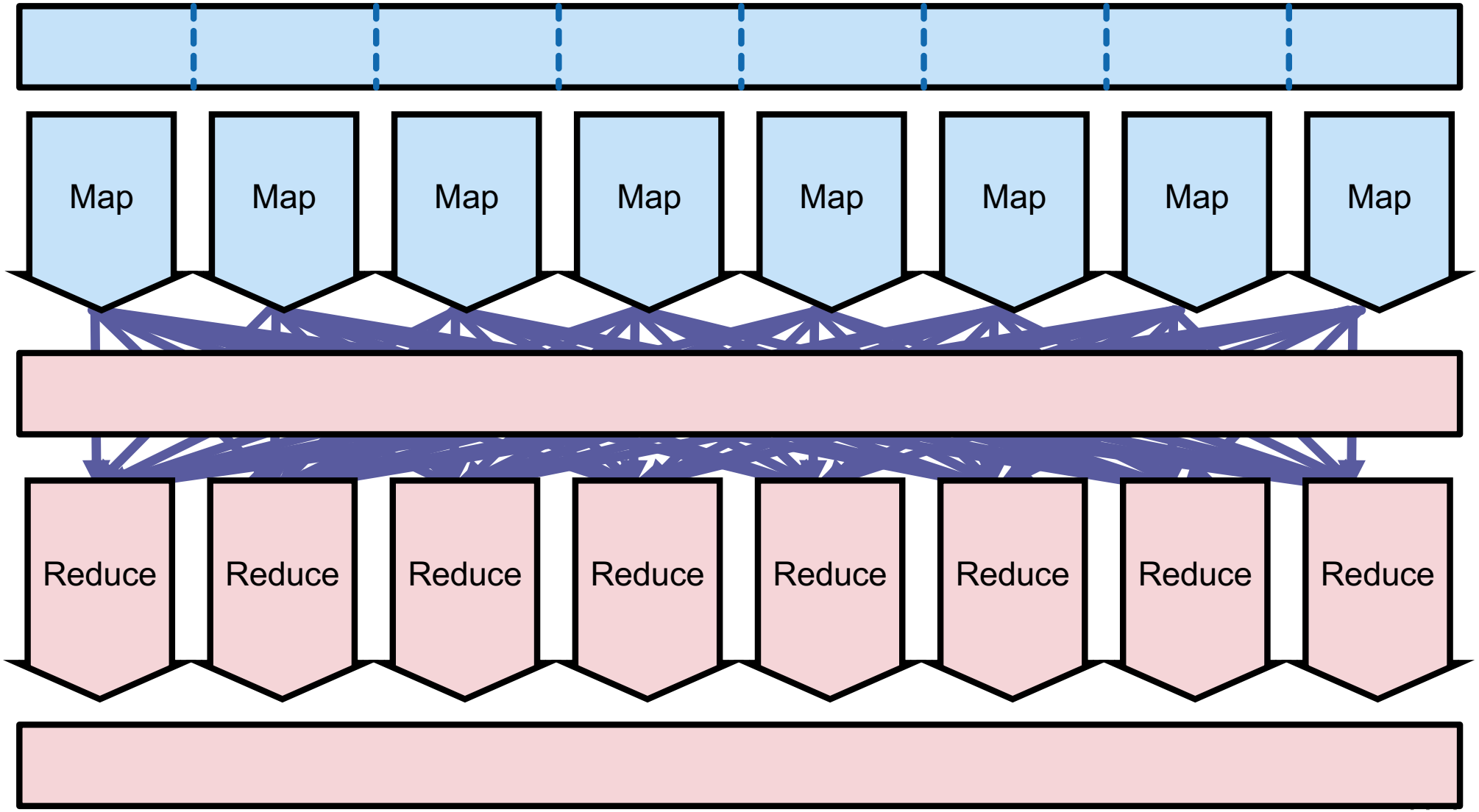
or

Reduce

Splits

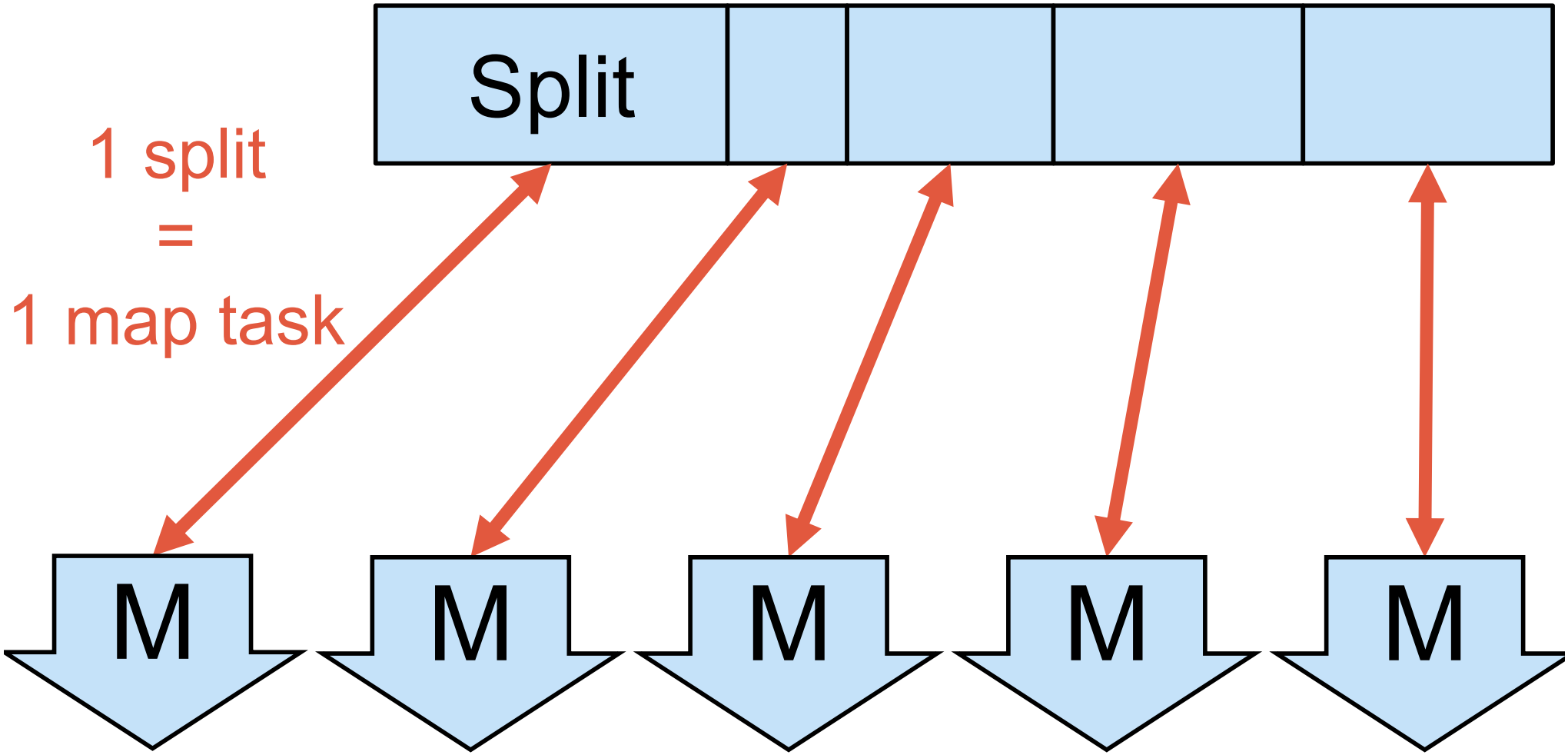

Map
Map
Map
Map
Map
Map
Map
Map
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce

Splits

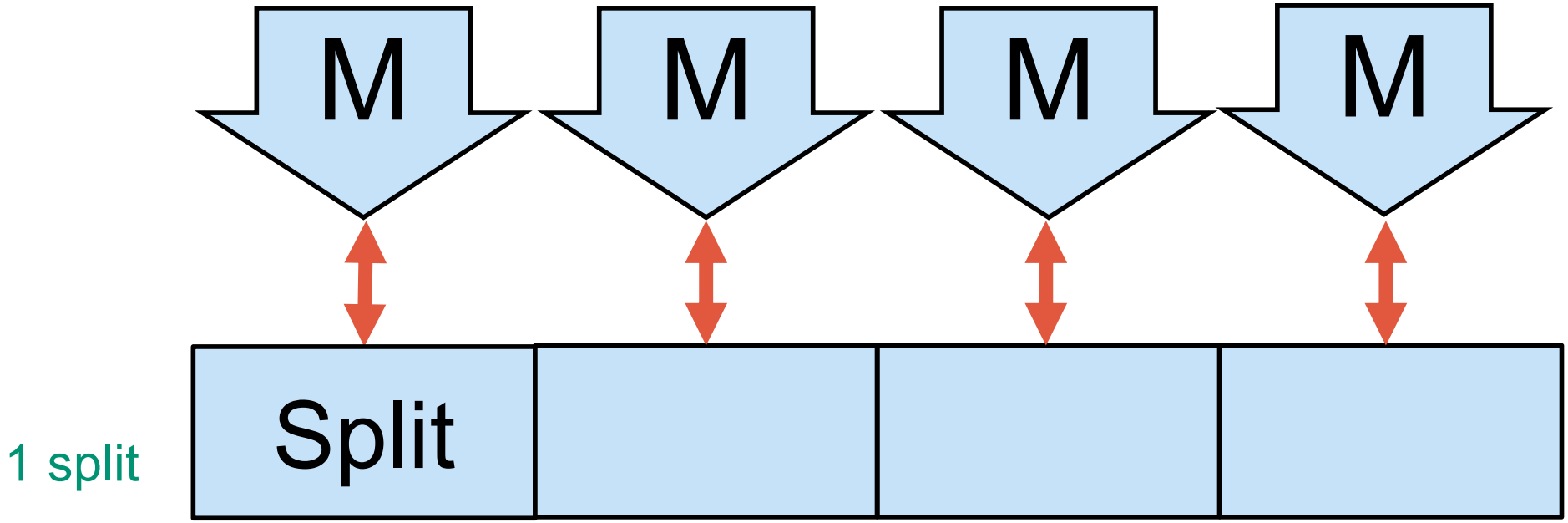

Map
Map
Map
Map
Map
Map
Map
Map
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce

Splits vs. map tasks

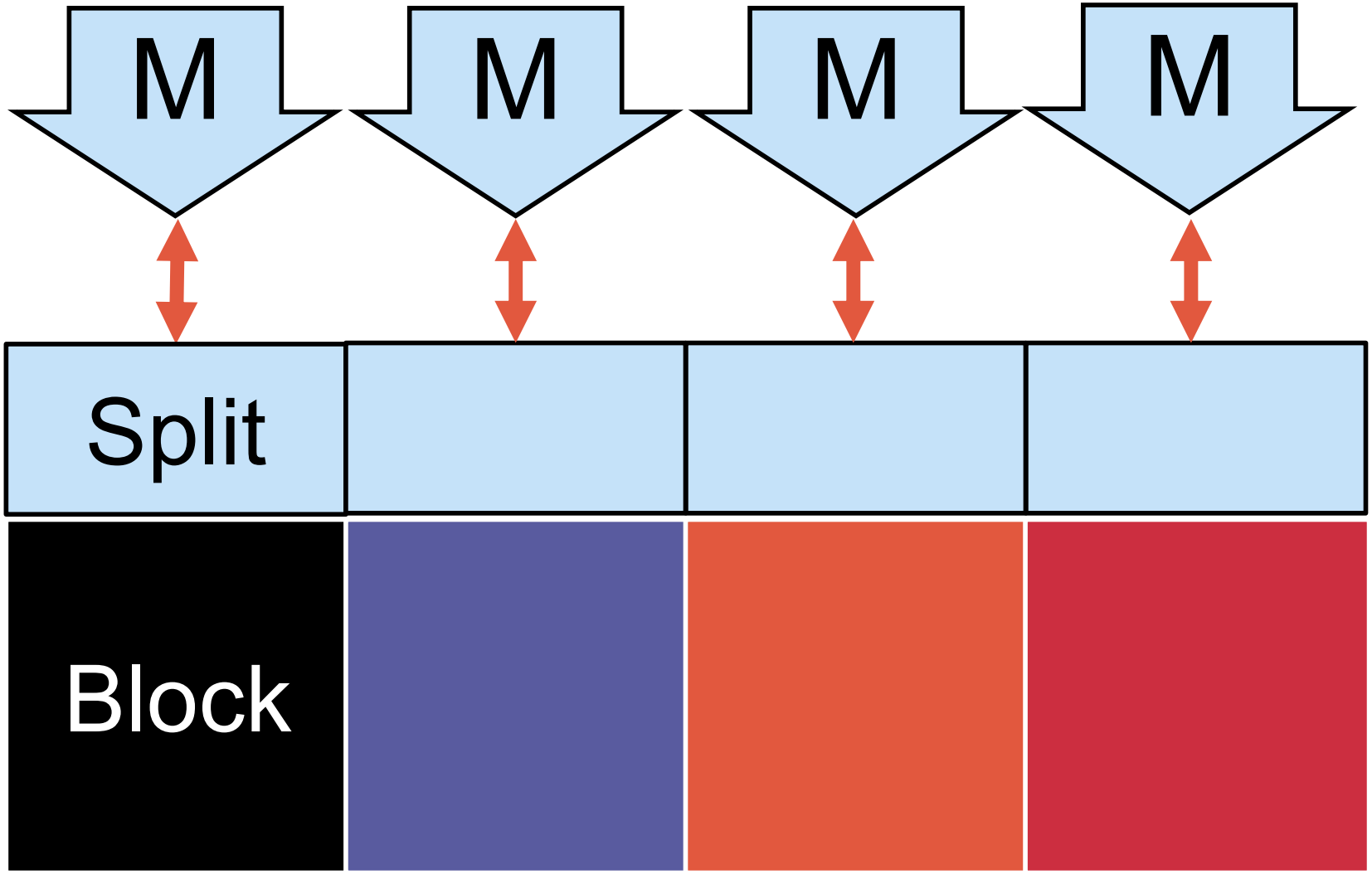
Splits vs. map tasks



In practice



In practice



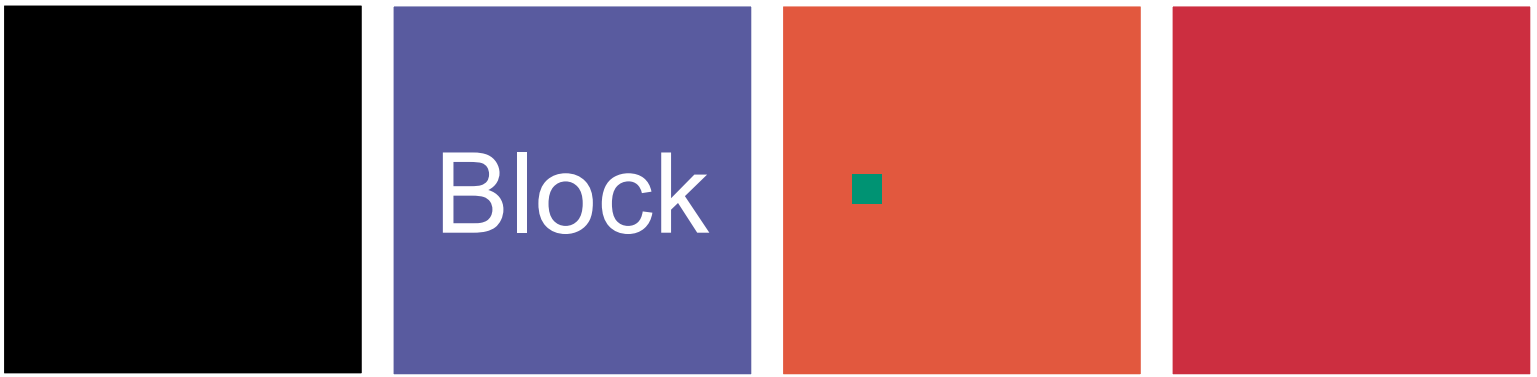
1 split
=
1 block

(subject to min and max size)

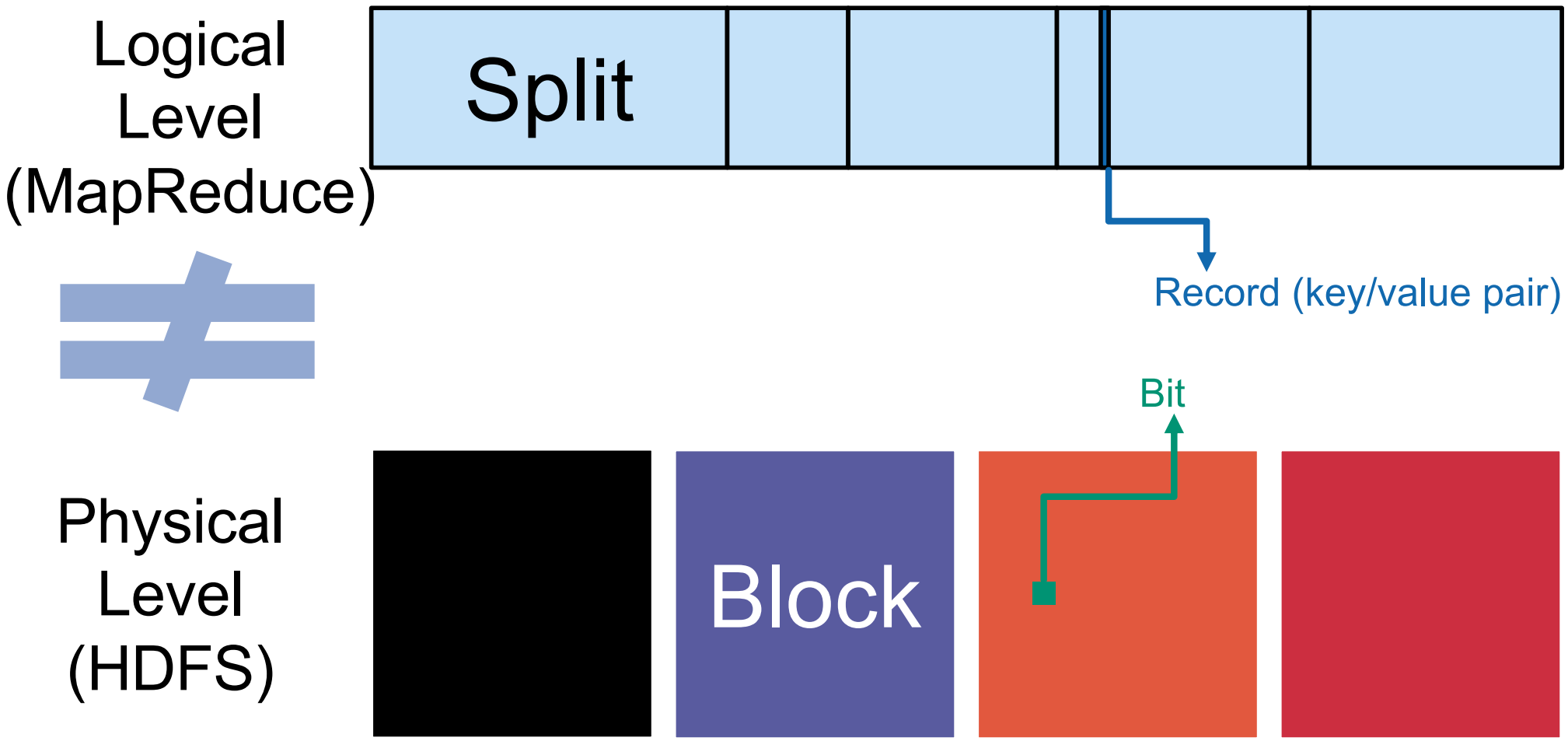
Splits vs. blocks: possible confusion

Logical Level (MapReduce)

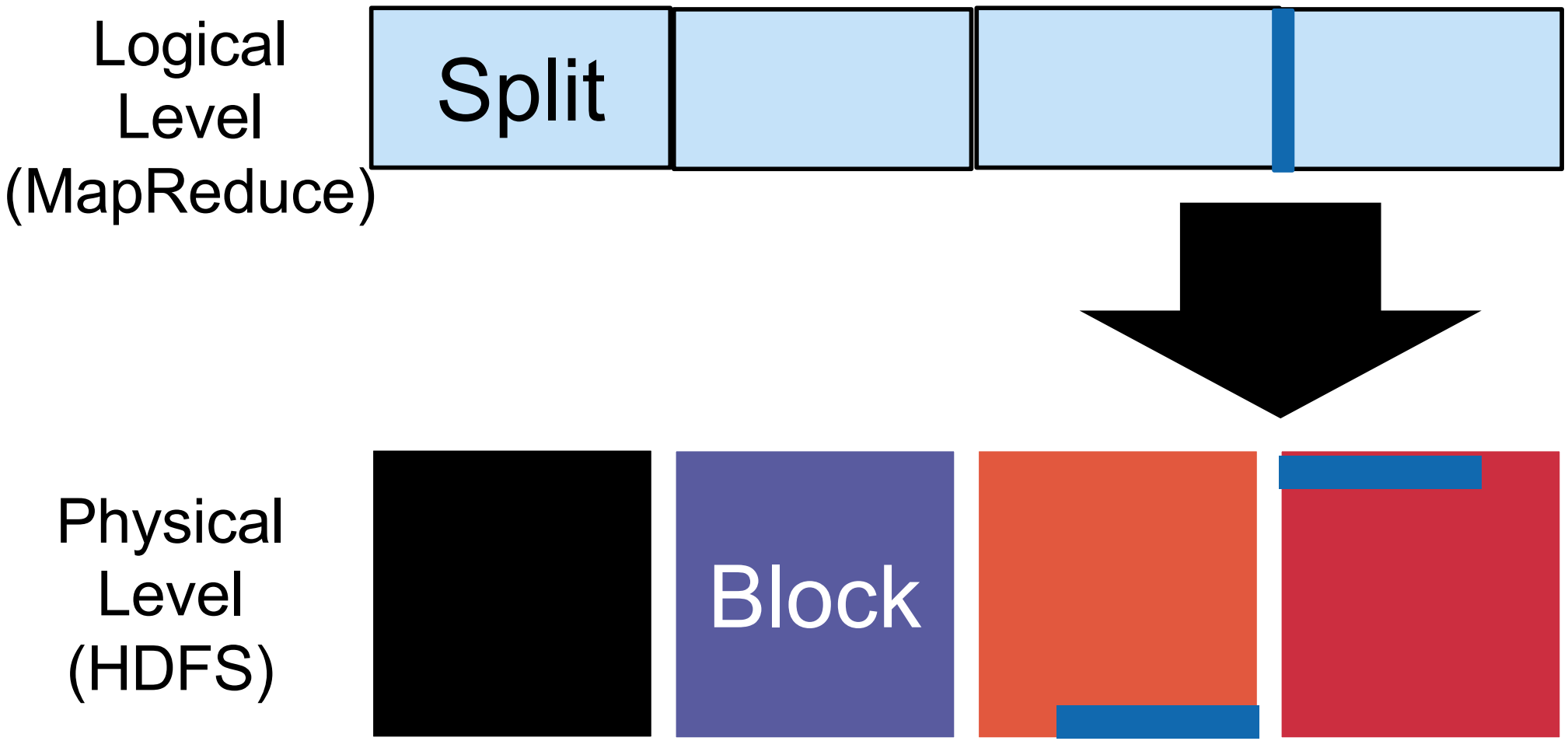
Physical Level (HDFS)



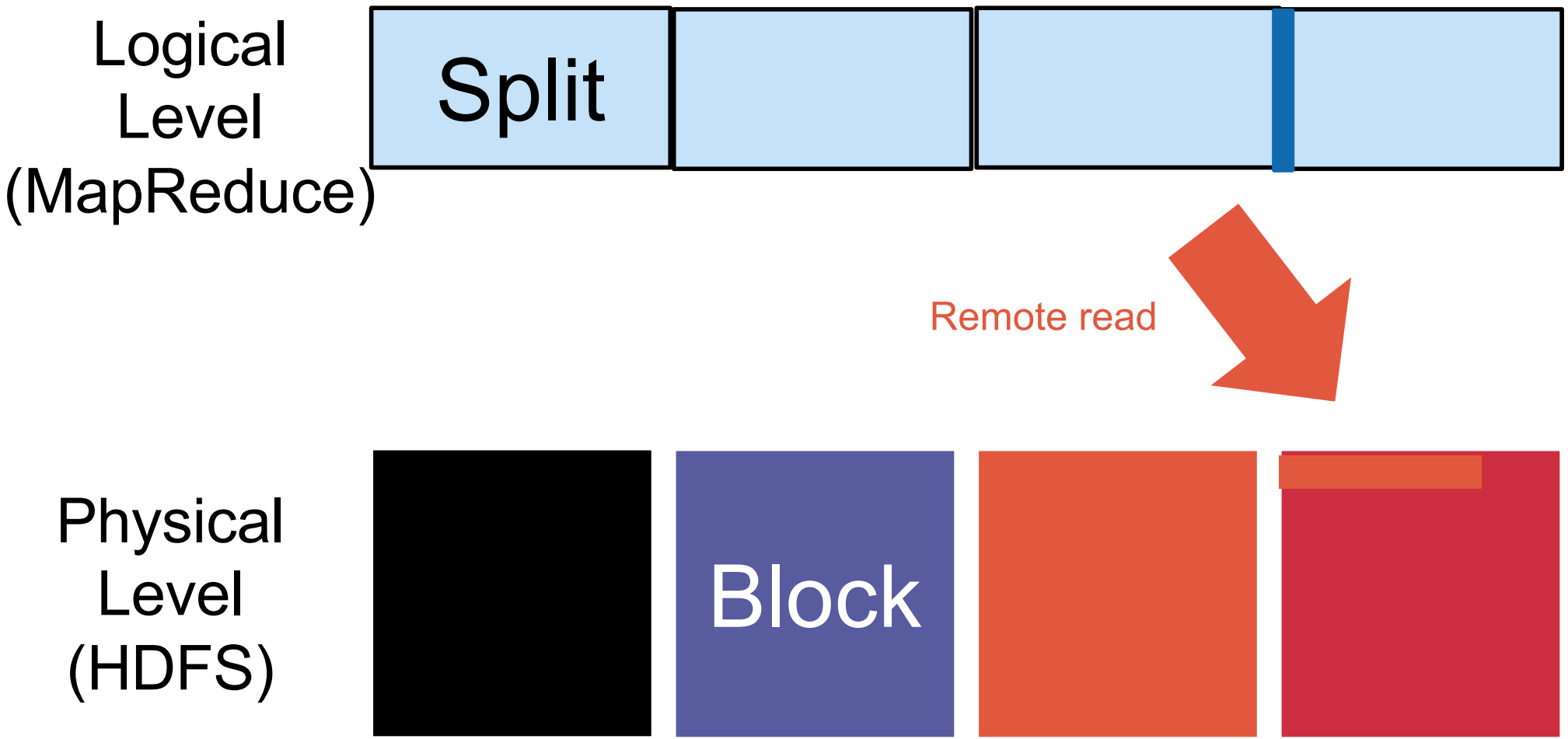
Splits vs. blocks: possible confusion



Records across blocks

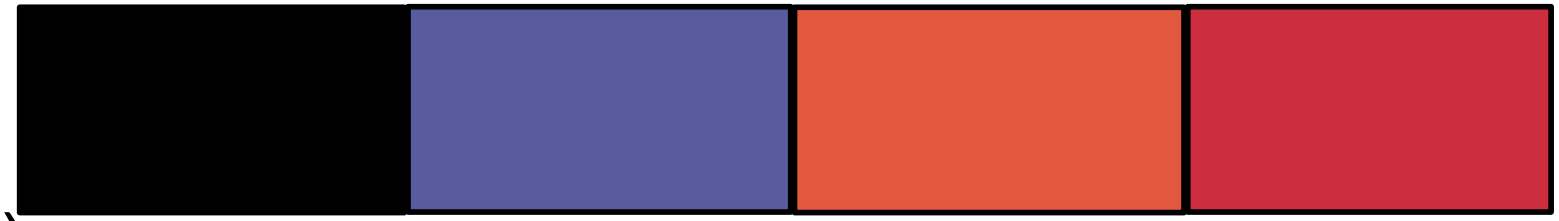


Records across blocks



Fine-tuning to adjust splits to blocks

Logical Level (MapReduce)



Physical Level (HDFS)