Big Data Analytics in Healthcare- Lesson 12: Spark

출처 : https://www.udacity.com/

1. Introduction to Spark

- Hadoop : 대표적인 빅데이터 시스템

1.분산파일 시스템 (HDFS)
2.분산병렬 처리 framework (MapReduce)
  ①장애처리
  ②머신러닝 알고리즘은 반복적인 작업을 하고 이는 mapreduce 적합하지 않음.
  ③이를 잘 처리할 수 있는 spark에 대해서 알아보자.
3.분산노드 리소스 관리 (YARN)

홈

오늘, 8:05 PM KST

상태 모든 상태 문제 1 구성 4 모든 최근 명령 클러스터 추가

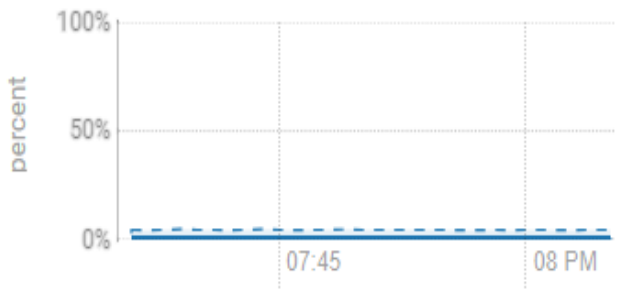
POSTECH (CDH 5.11.1, Parcel)

- 호스트 1
- Flume
- HDFS 1
- Hive
- Hue
- Impala 1
- Oozie
- Sentry
- Solr
- Spark
- YARN (MR... 1
- ZooKeeper

차트

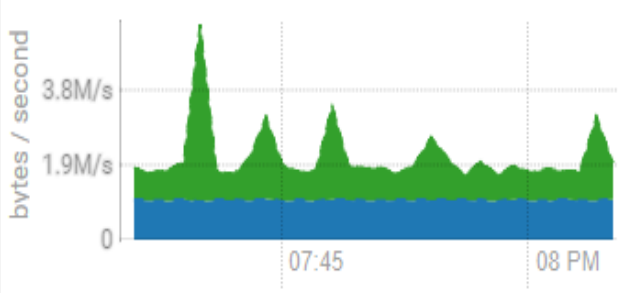
30분 1시간 2시간 6시간 12시간 1일 7d 30d

클러스터 CPU



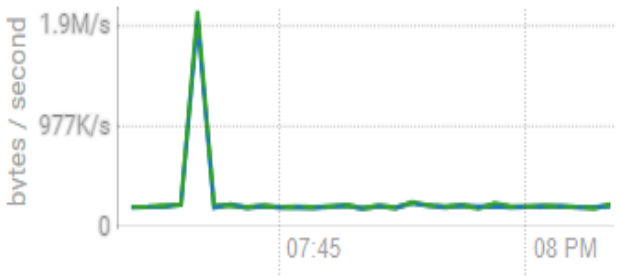
POSTECH, 전체 호스트의 호스트 CPU 사용량 0.85%

클러스터 디스크 IO



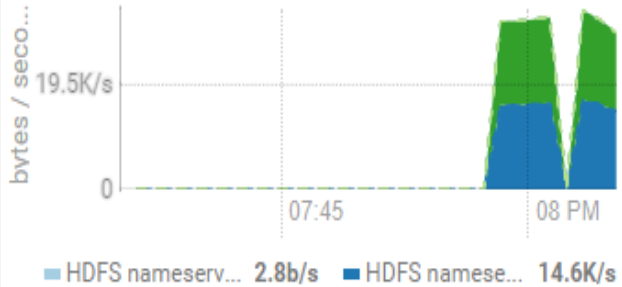
모든 디스크의 ... 959K/s 모든 디스크의 총... 1M/s

클러스터 네트워크 IO



모든 네트워크 ... 203K/s 모든 네트워크 ... 219K/s

HDFS IO



HDFS nameserv... 2.8b/s HDFS namese... 14.6K/s
HDFS, 모든 Dat... 2.8b/s HDFS, 모든 Da... 14.6K/s

파일 브라우저

파일 이름 검색 | 작업 | 휴지통으로 이동 | 업로드 | 새로 만들기

홈 / user / admin / **daliwork** | 기록 | 휴지통

이름	크기	사용자	그룹	권한	날짜
..		admin	admin	drwxrwxrwx+	May 18, 2018 10:49 PM
.		admin	admin	drwxr-xr-x	September 05, 2018 01:13 AM
BE01A_ACC-PNL_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE01A_ACC-PNL_RE-HS2_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE01G_MCC-1A_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE01G_MCC-1A_RE-HS2_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE01G_MCC-1B_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE01G_MCC-1B_RE-HS2_20180904.csv	40.3 KB	admin	admin	-rw-r--r--	September 05, 2018 01:05 AM
BE03G_MCC-3A_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM
BE03G_MCC-3A_RE-HS2_20180904.csv	40.4 KB	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM
BE03G_MCC-3B_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM
BE03G_MCC-3B_RE-HS2_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM
BE05G_MCC-5_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM
BEB1M_INV-PL1_RE-HS1_20180904.csv	0 바이트	admin	admin	-rw-r--r--	September 05, 2018 01:06 AM

표시 45 / 60 항목 페이지 1 of 2

파일 브라우저

바이너리로 ...

파일 편집

다운로드

파일 위치 보기

새로 고침

최종 수정 날짜
2018.09.04 오후 4시 5분

사용자
admin

그룹
admin

크기
40.33 KB

모드
100644

홈

페이지 1 -> 11 / 11

/ user / admin / daliwork / **BE01G_MCC-1B_RE-HS2_20180904.csv**

```
2018-09-04 00:07:00.000000,R,0.418653224767,60.6060606061,219.194985823,60.6060606061,56.1531300073,91.7666876675,0.61191
1919614,53.4773251865,
2018-09-04 00:22:00.000000,R,0.434067233166,60.6060606061,221.130230061,60.6060606061,61.9090991681,95.9853871318,0.64498
4627536,56.4476063388,
2018-09-04 00:37:00.000000,R,0.418906702683,60.6060606061,219.767173183,60.6060606061,55.5734530226,92.0619418762,0.60365
2843835,49.2224091353,
2018-09-04 00:52:00.000000,R,0.423232618124,60.6060606061,220.392163569,60.6060606061,59.2465719466,93.2771524012,0.63516
7030955,48.1775334201,
2018-09-04 01:07:00.000000,R,0.455039925218,60.6060606061,221.196740871,60.6060606061,64.4483436547,100.653348424,0.64030
0046284,56.8304307269,
2018-09-04 01:22:00.000000,R,0.449376666186,60.6060606061,220.077464338,60.6060606061,62.0882431445,98.8976772269,0.62780
2845178,51.0847844805,
2018-09-04 01:37:00.000000,R,0.408848321004,60.6060606061,217.597248185,60.6060606061,62.1937385578,88.9642695757,0.69908
6710366,47.2449405381,
2018-09-04 01:52:00.000000,R,0.429768974341,61.5384615385,216.921068876,60.6060606061,56.5551901003,93.2259452837,0.60664
6464439,50.1547367338,
2018-09-04 02:07:00.000000,R,0.428351983632,60.6060606061,223.074847347,60.6060606061,64.4233331346,95.5545533593,0.67420
4743466,51.003815343,
2018-09-04 02:22:00.000000,R,0.426006092207,60.6060606061,219.849422797,60.6060606061,61.7611050097,93.6571934796,0.65943
7921586,47.9300349142,
2018-09-04 02:37:00.000000,R,0.441782002873,61.5384615385,222.4955565,60.6060606061,63.6722887638,98.2945325807,0.6477704
00775,58.7363665911,
2018-09-04 02:52:00.000000,R,0.418673378794,60.6060606061,221.12893774,60.6060606061,61.8288130201,92.5807995127,0.667836
239756,53.5076623518,
```

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

http://spark.apache.org/examples.html

Word Count

In this example, we use a few transformations to build a dataset of (String, Int) pairs c

Python | Scala | Java

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

2. Environment

ENVIRONMENT

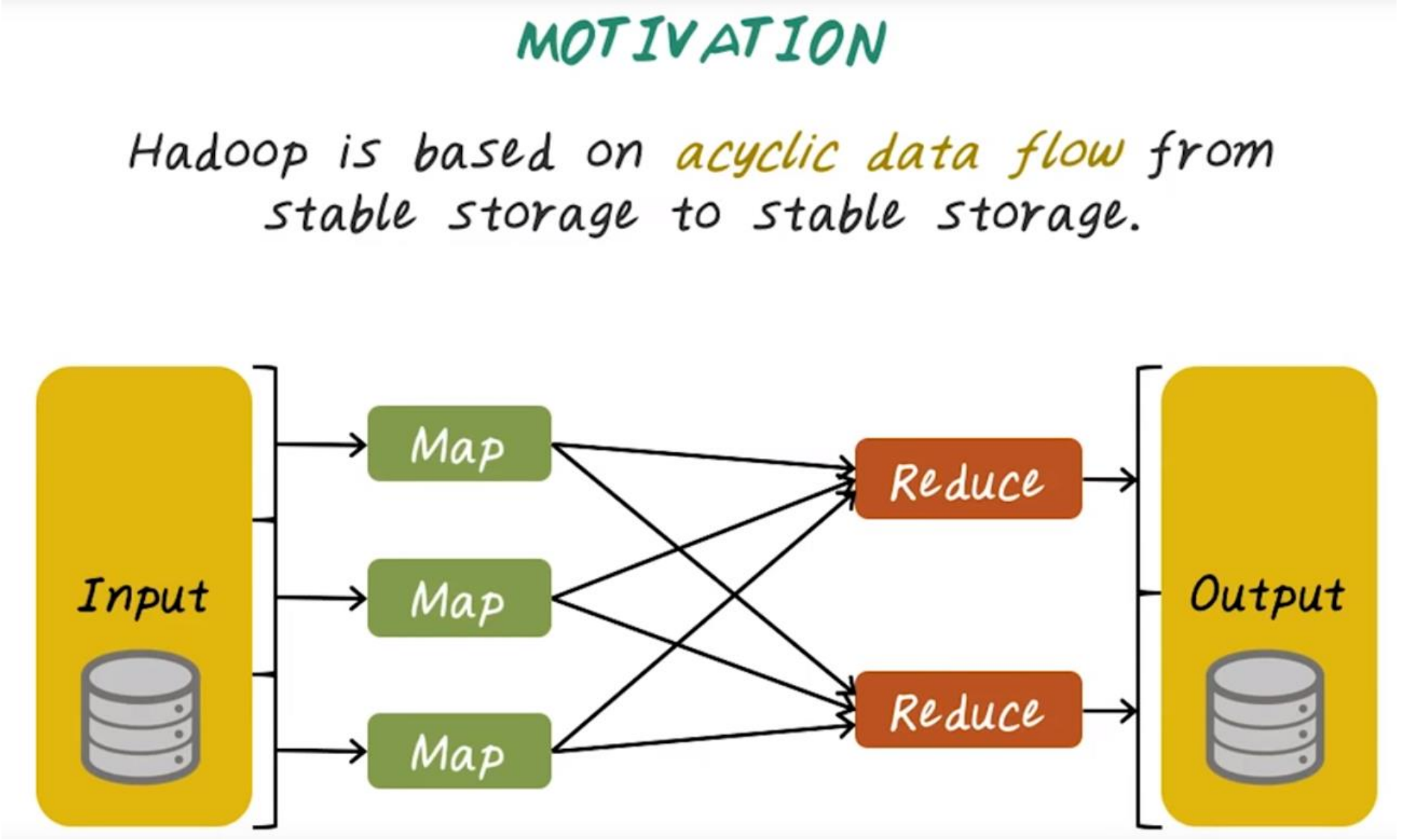
amazon web services™

Google Cloud Platform

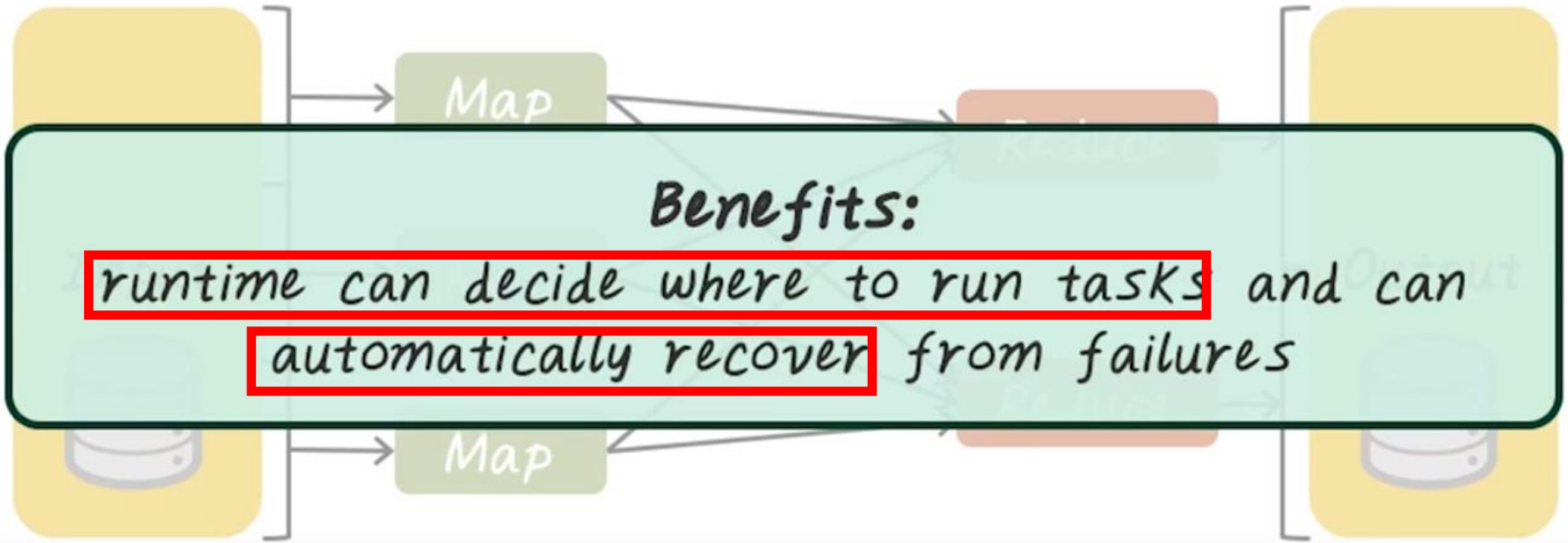
Windows Azure

자막

3. Motivation



Hadoop is based on acyclic data flow from stable storage to stable storage.



MOTIVATION

Hadoop is inefficient for applications that repeatedly reuse a working set of data:

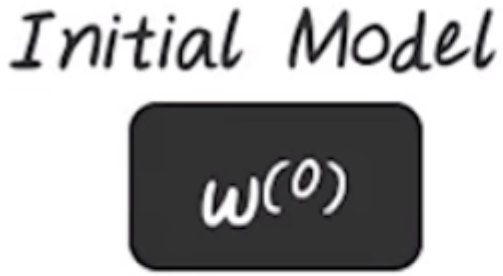
Iterative Algorithms

- Machine learning
- Graph analysis

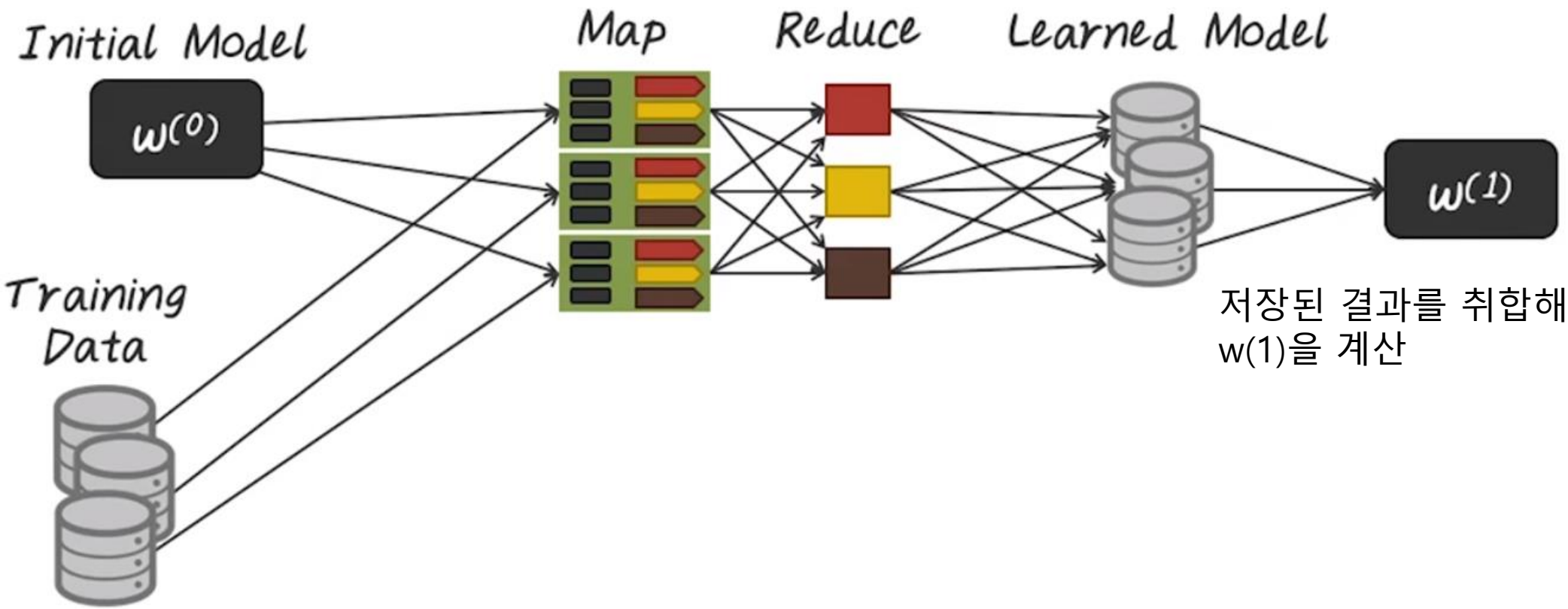
Interactive Data Mining Tools

- R
- Python

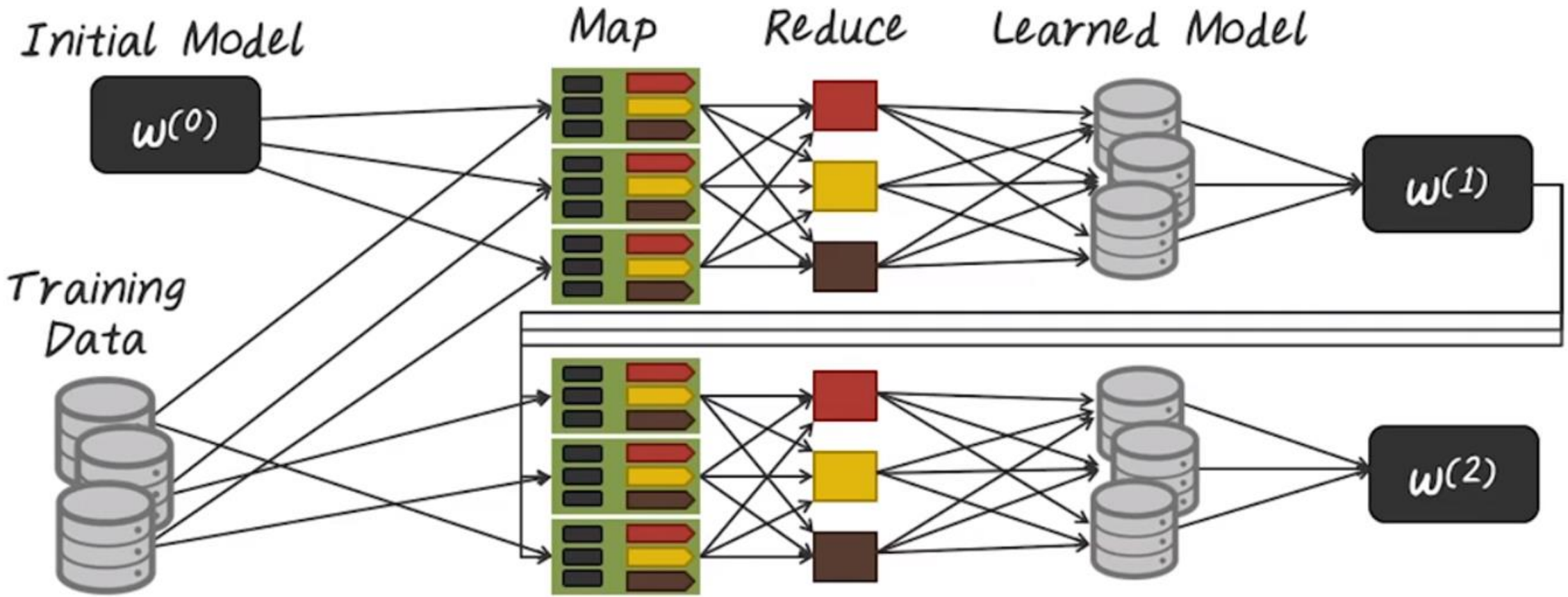
4. Iteration in Map Reduce

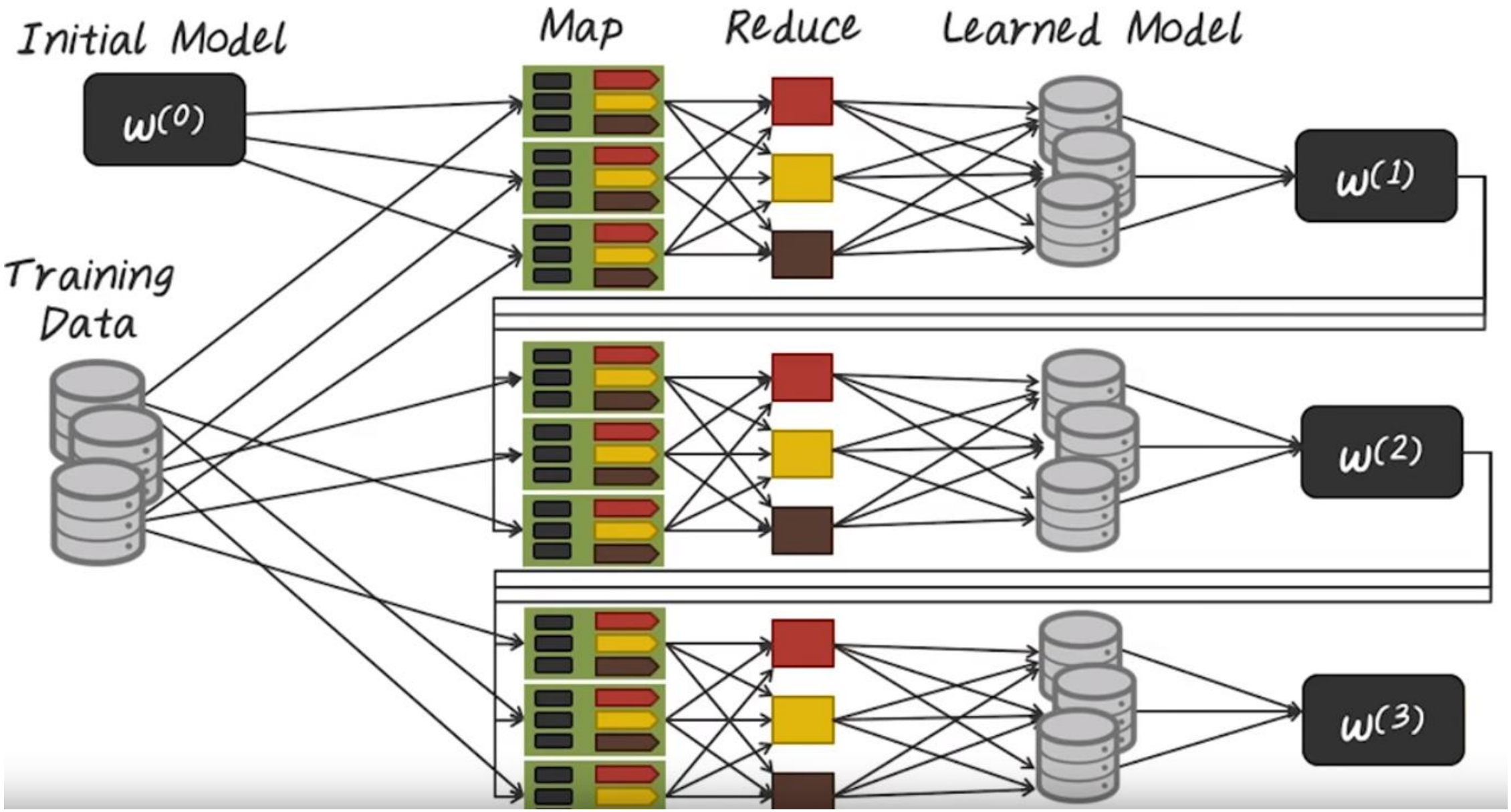


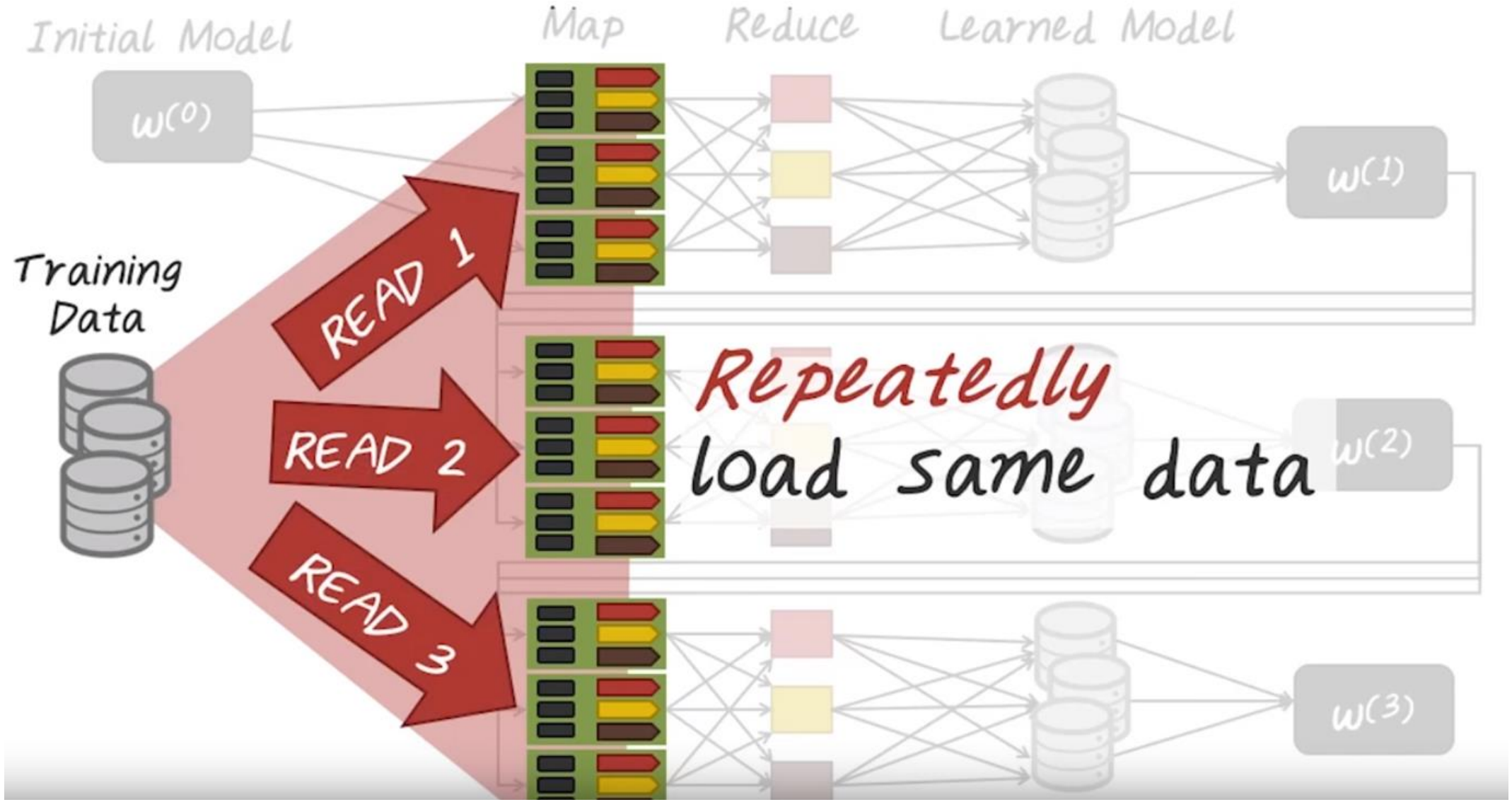
반복적인 처리가 필요한 머신러닝 알고리즘을 MapReduce로 구현하는 방법을 알아보자.

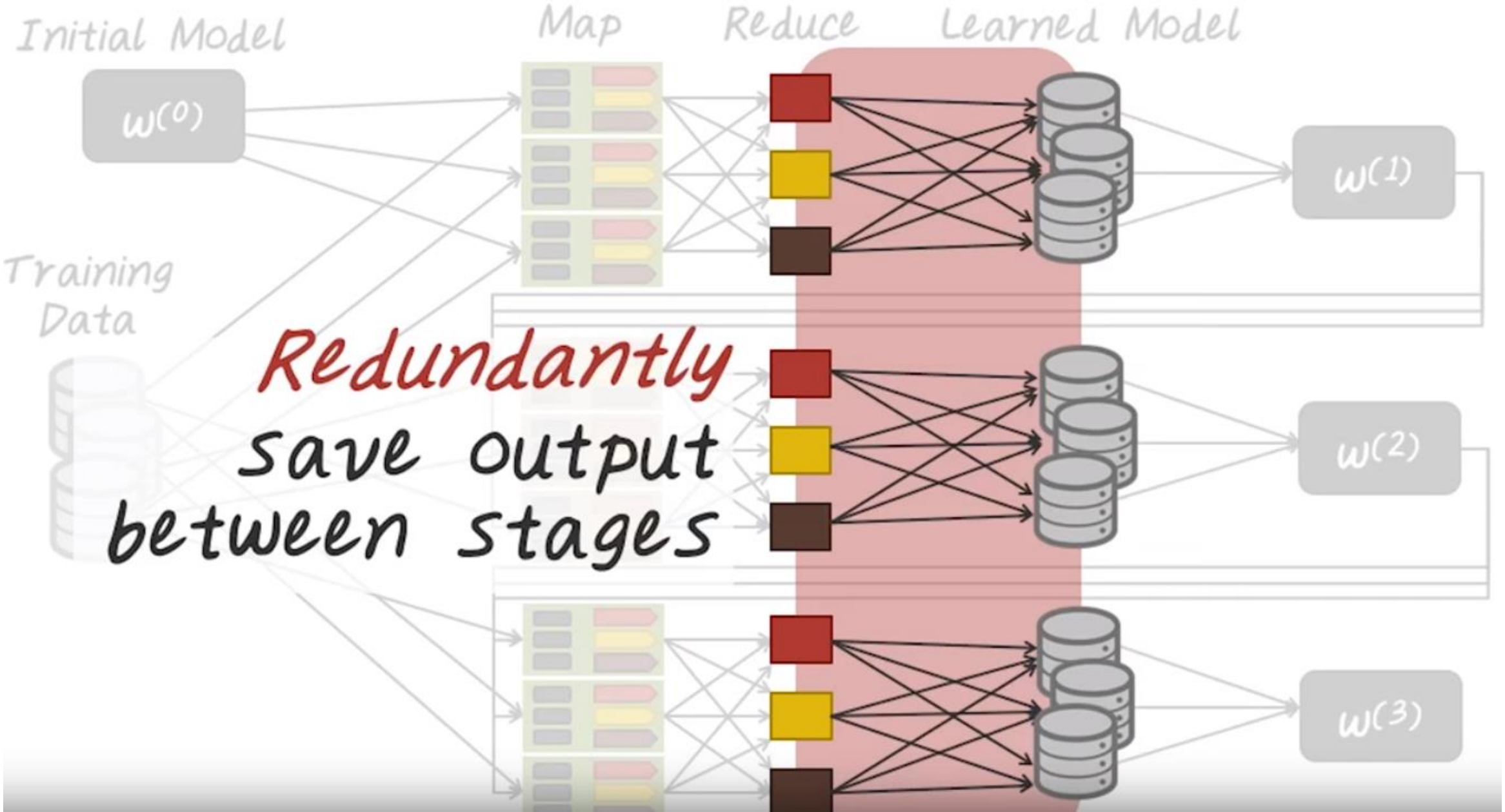


저장된 결과를 취합해서
w(1)을 계산


Initial Model
w(0)
Map
Reduce
Learned Model
w(1)
Training Data
w(2)


Initial Model
w(0)
Map
Reduce
Learned Model
w(1)
Training Data
w(2)
w(3)


Initial Model
Map
Reduce
Learned Model
w(0)
w(1)
w(2)
w(3)
Training Data
READ 1
READ 2
READ 3
Repeatedly
load same data

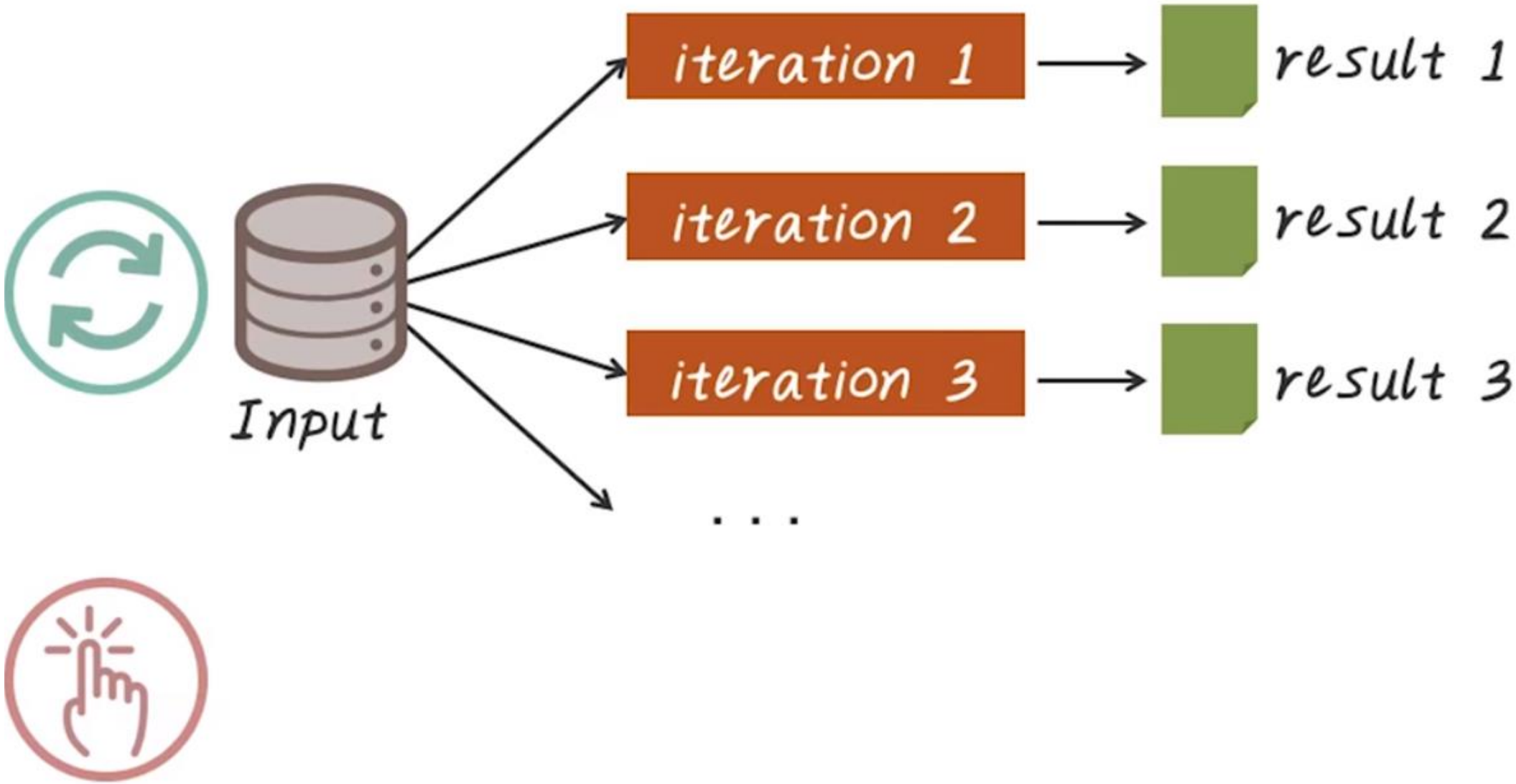

Initial Model
Map
Reduce
Learned Model
w(0)
w(1)
Training Data
Redundantly
save output
between stages
w(2)
w(3)

5. Workload Illustration

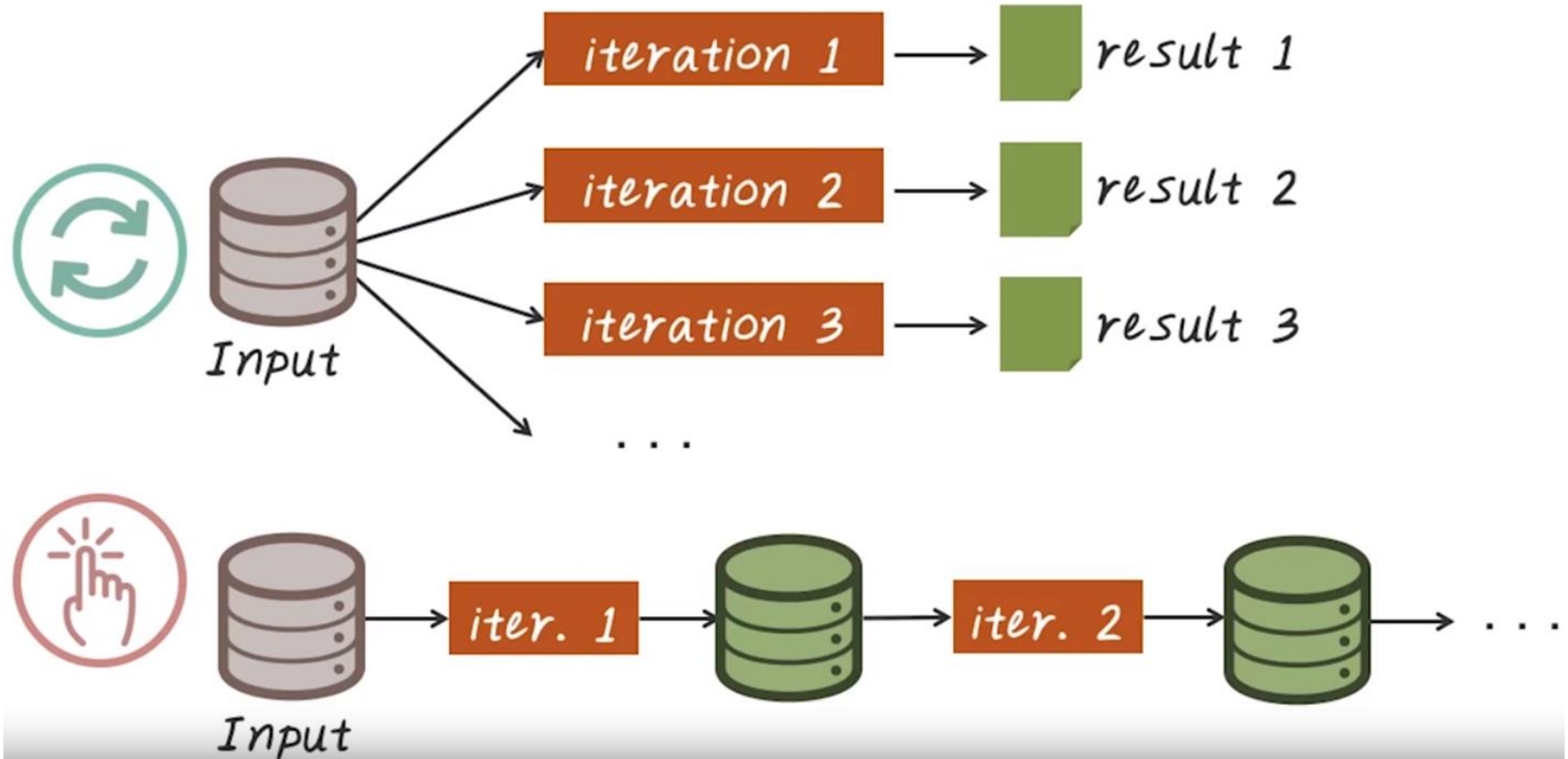
반복적인 처리 관점

인터엑티브한 처리 관점

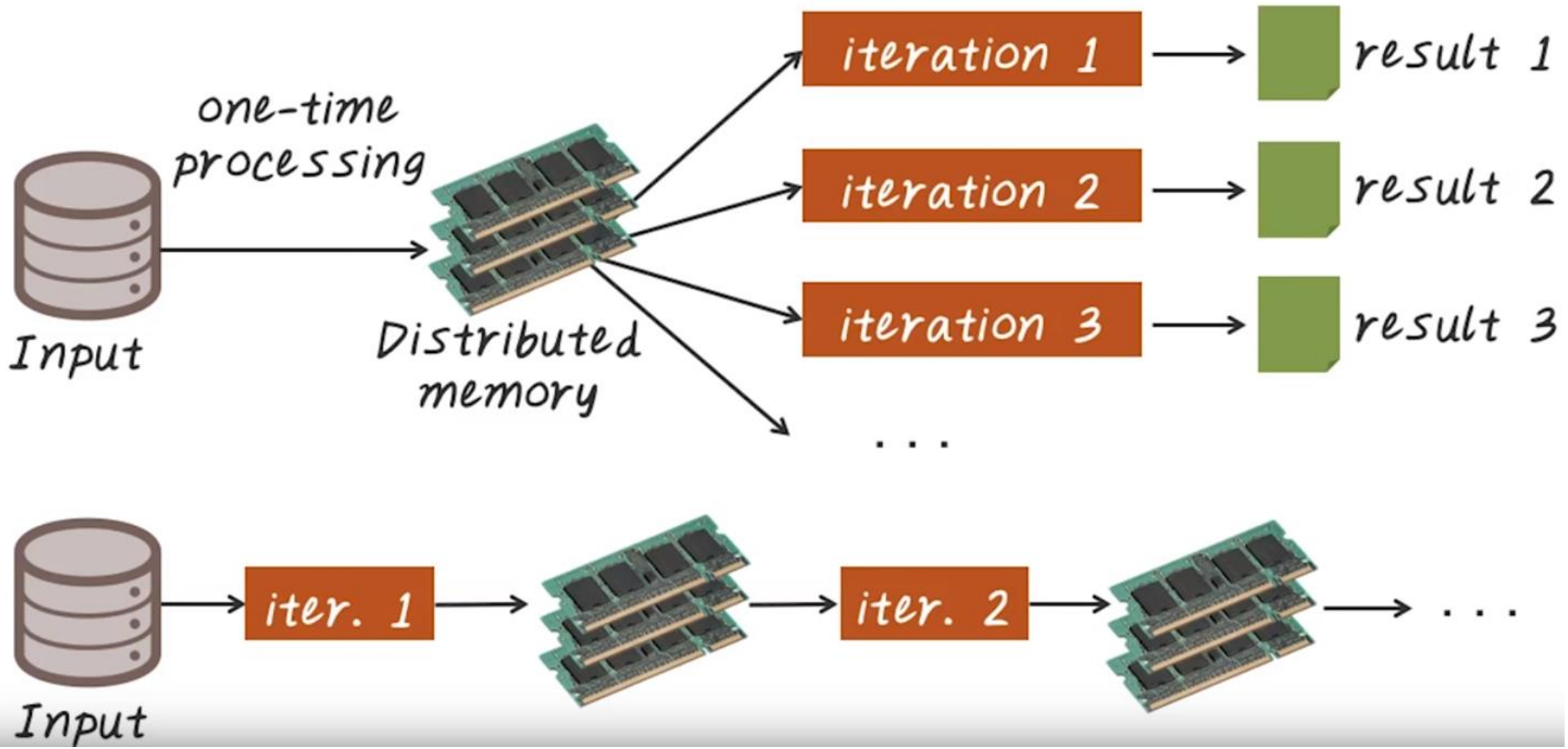
WORKLOAD ILLUSTRATION



WORKLOAD ILLUSTRATION

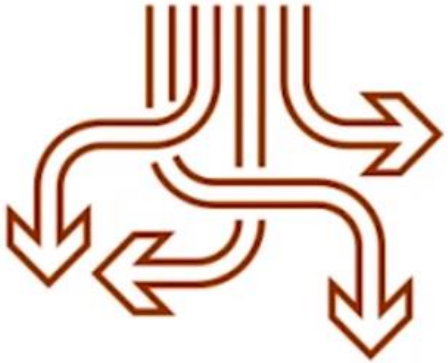


GOAL: KEEP WORKING SET IN MEMORY



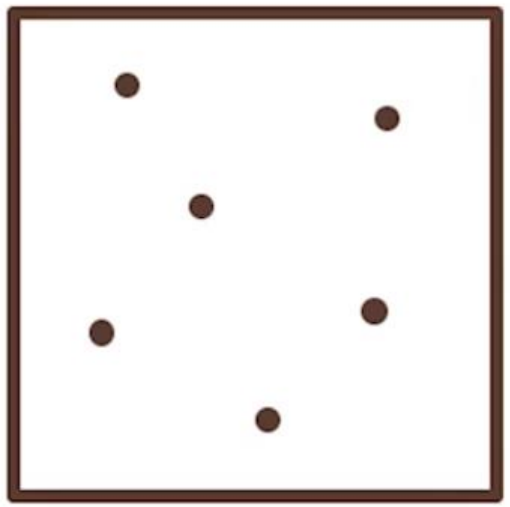
6. Challenge

CHALLENGE



How to design a distributed memory abstraction that is both fault-tolerant and efficient?

CHALLENGE

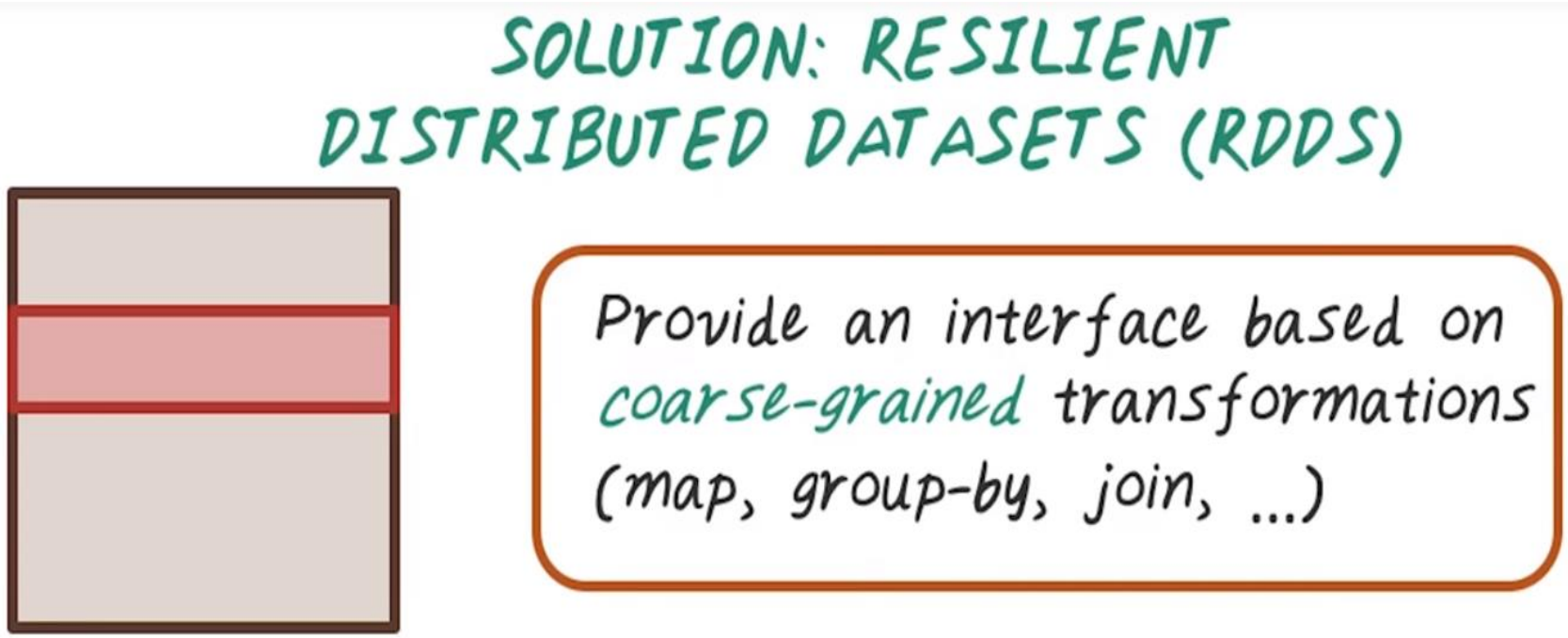


Existing distributed storage abstractions depend on *fine-grained* updates

- Reads and writes to cells in a table
- E.g. databases, key-value stores, distributed memory

Require replicating data or logs across nodes for **fault tolerance** =

7. Solution RDDs

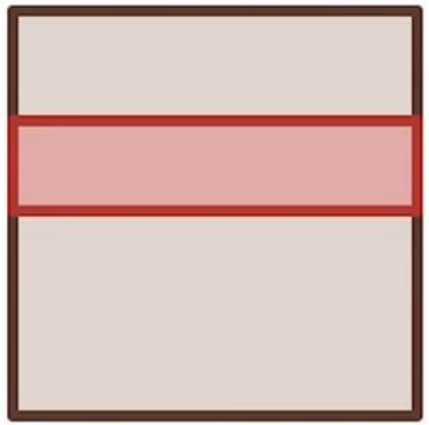


앞장에서 얘기한 2가지 문제를 해결하기 위해서, 스파크는 계산의 세분성과 내결함성을 위한 효율성 간의 균형을 결정합니다.

해법 : RDDS

RDD는 전체 데이터 세트의 거친 변환을 기반으로 인터페이스를 제공합니다. =>
이는 특정 데이터셋에 관심을 두지 않아서 전체데이터셋에 모두 적용
=>
coarse-grained 변환만 추적하면 모두 연산에 대해서 효율적으로 추적이 가능함.

SOLUTION: RESILIENT DISTRIBUTED DATASETS (RDDS)

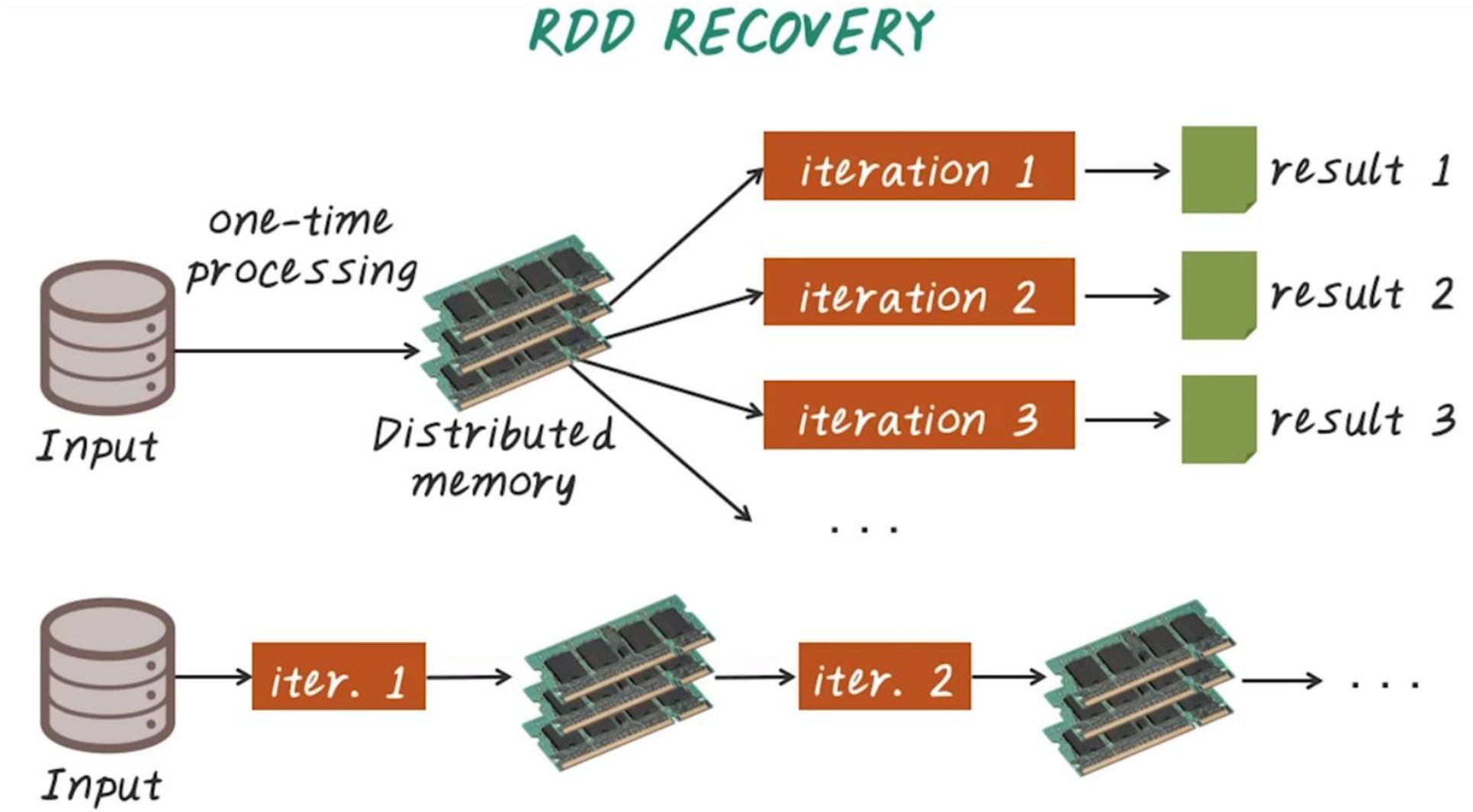


Provide an interface based on *coarse-grained* transformations (map, group-by, join, ...)

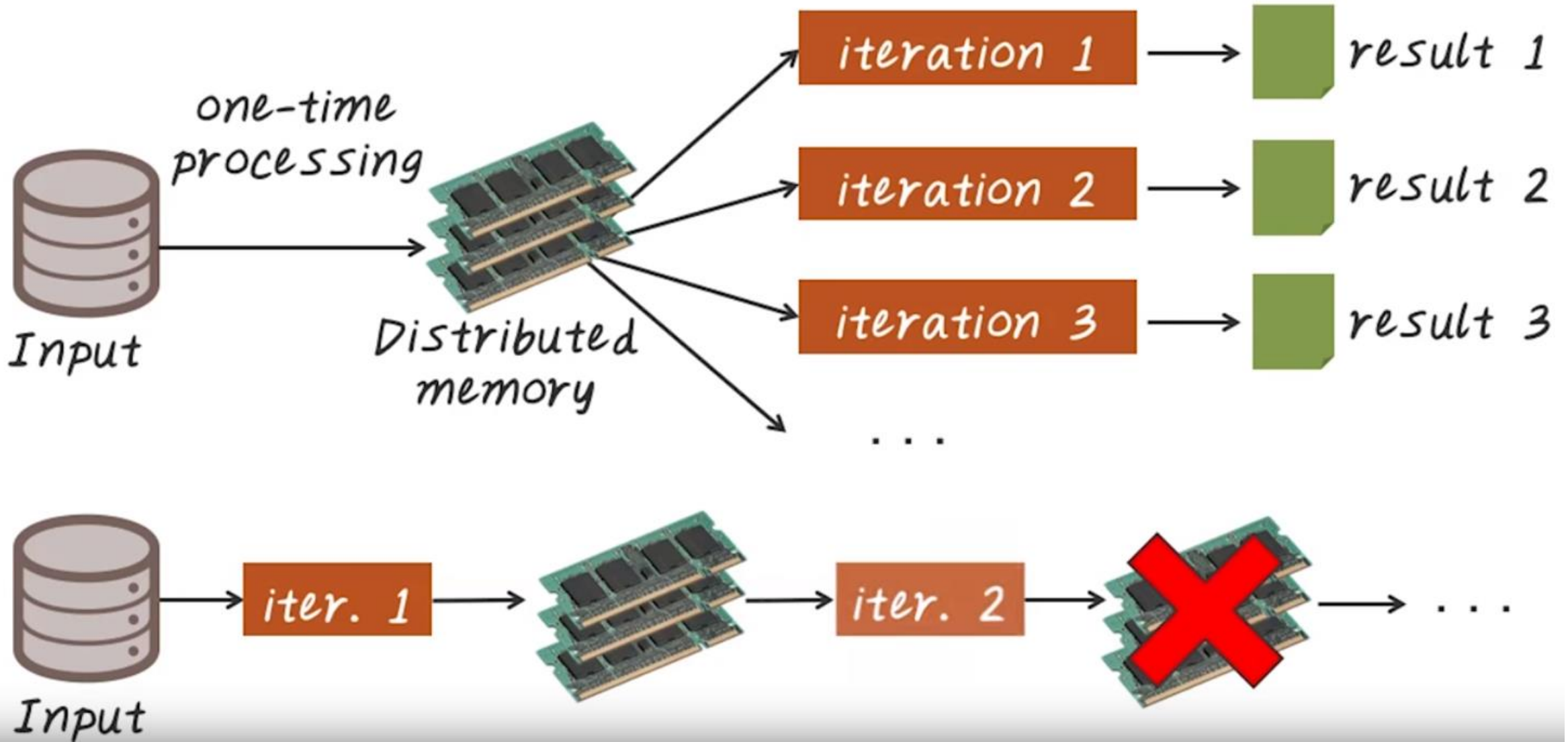
Efficient fault recovery using *lineage*

- Log one operation to apply to many elements
- Recompute lost partitions on failure
- No cost if nothing fails

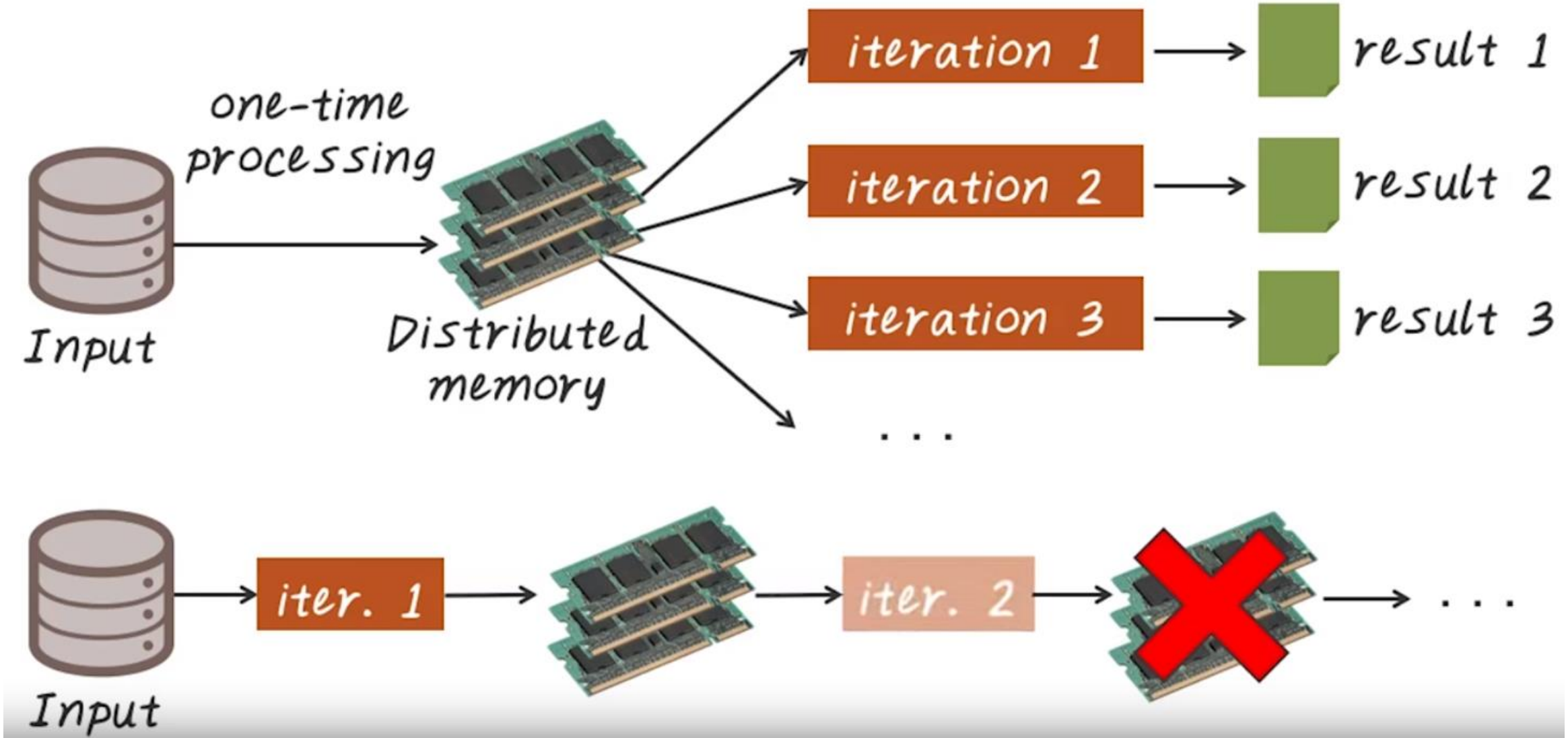
8. RDD Recovery



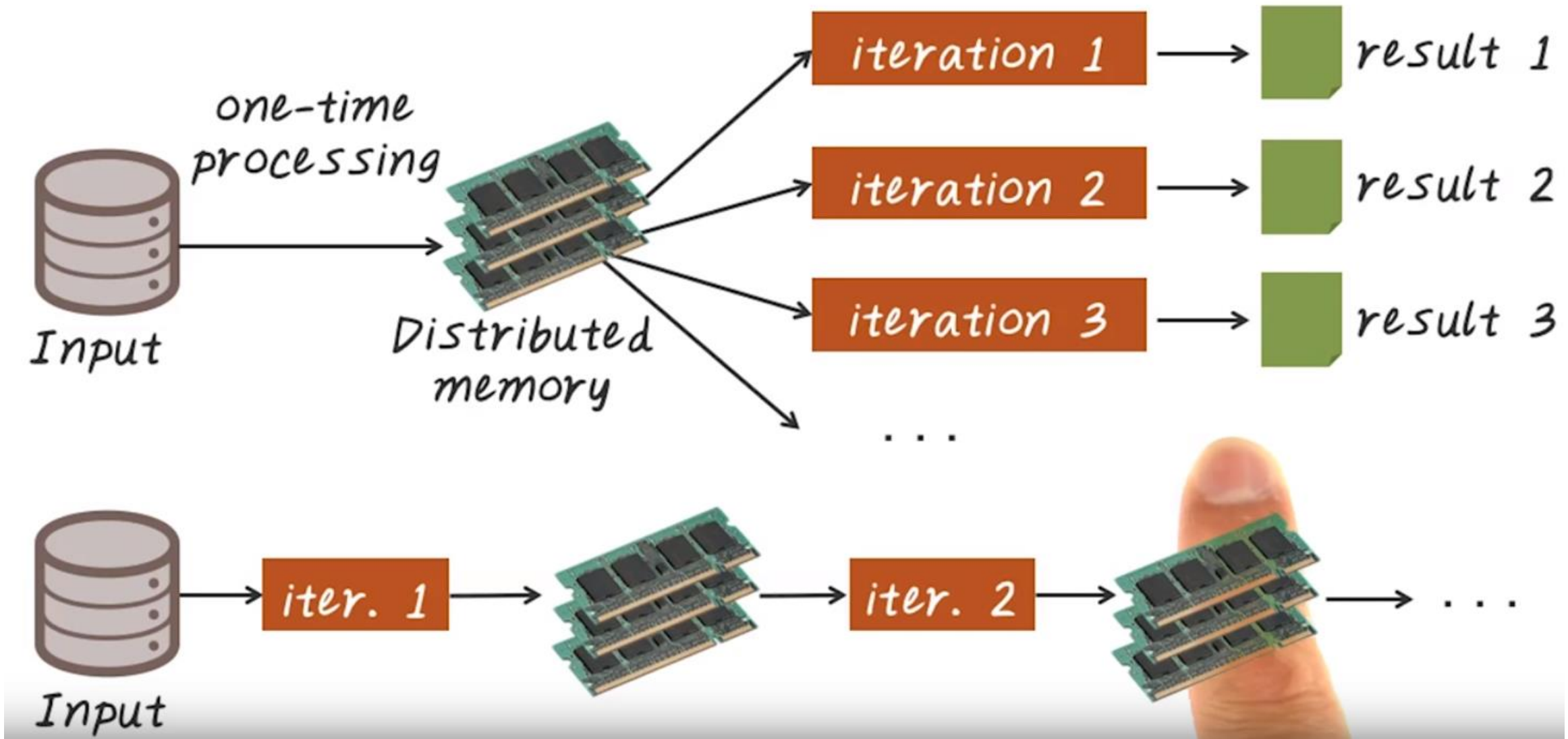
RDD RECOVERY



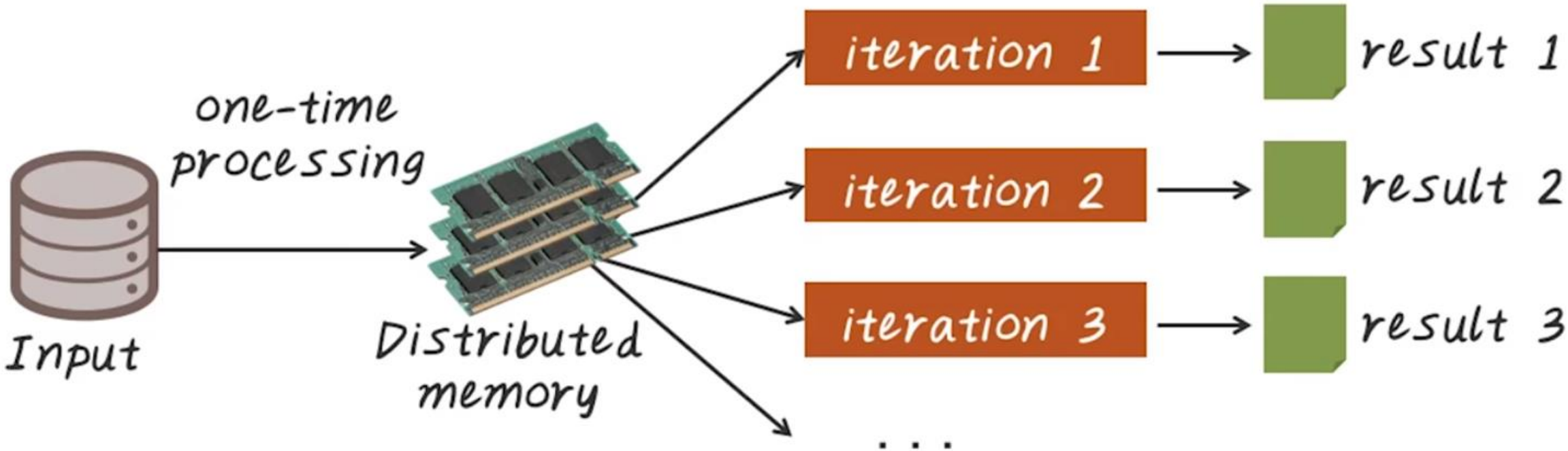
RDD RECOVERY

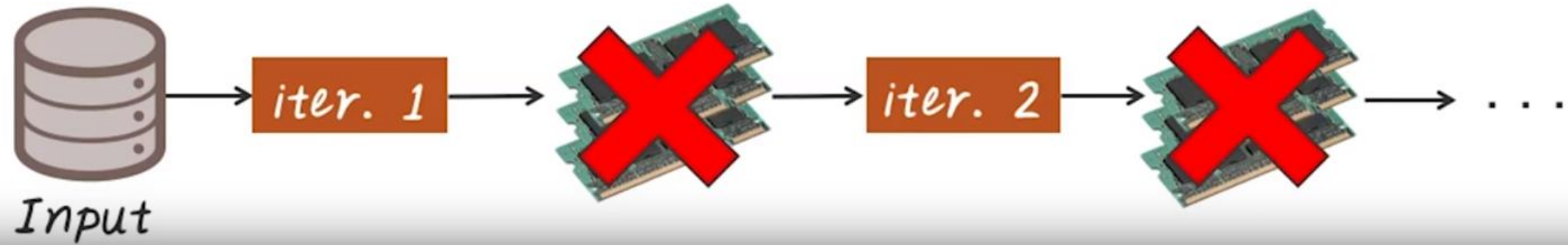


RDD RECOVERY

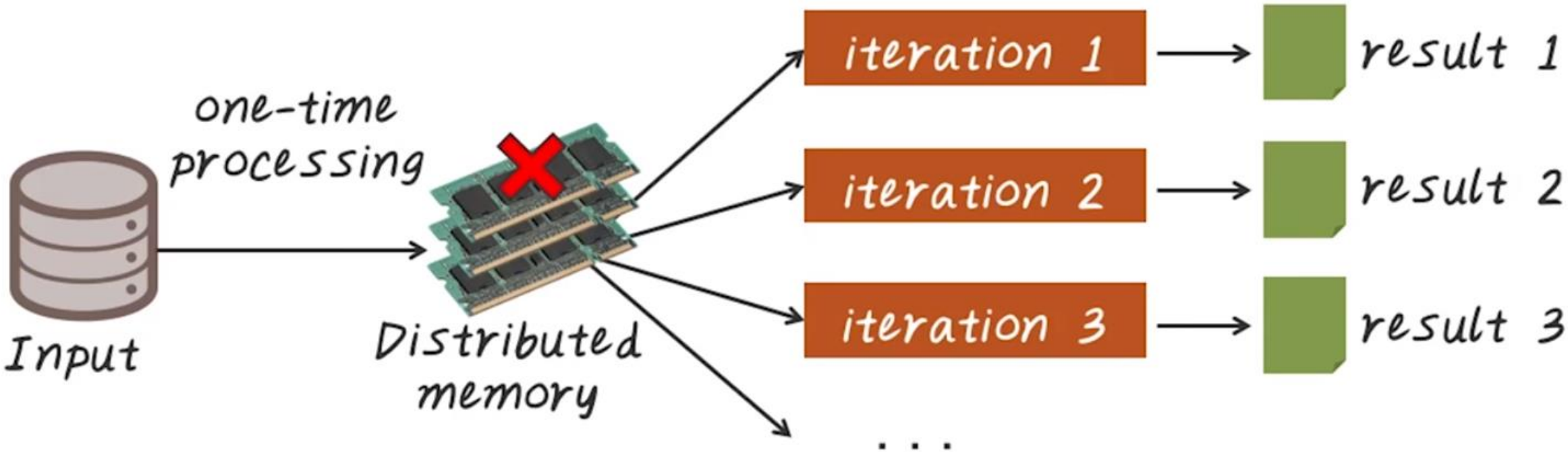


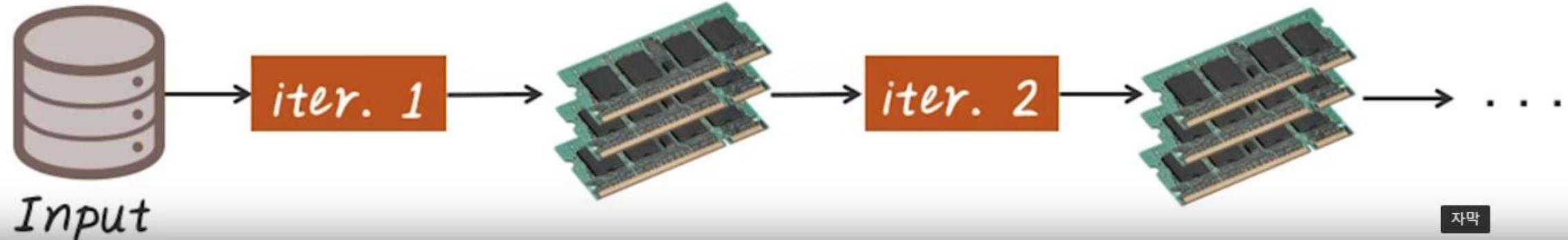
RDD RECOVERY



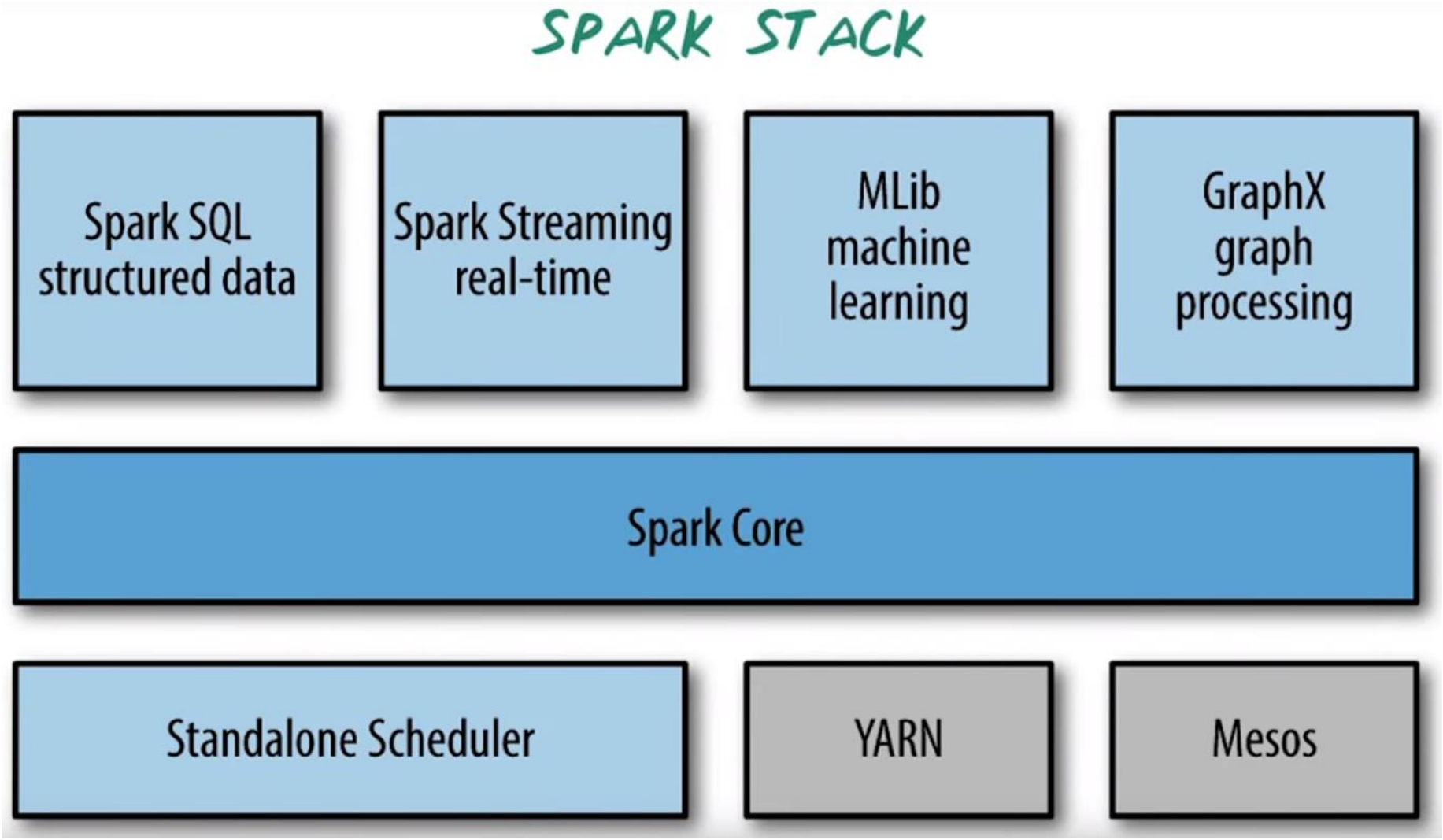


RDD RECOVERY

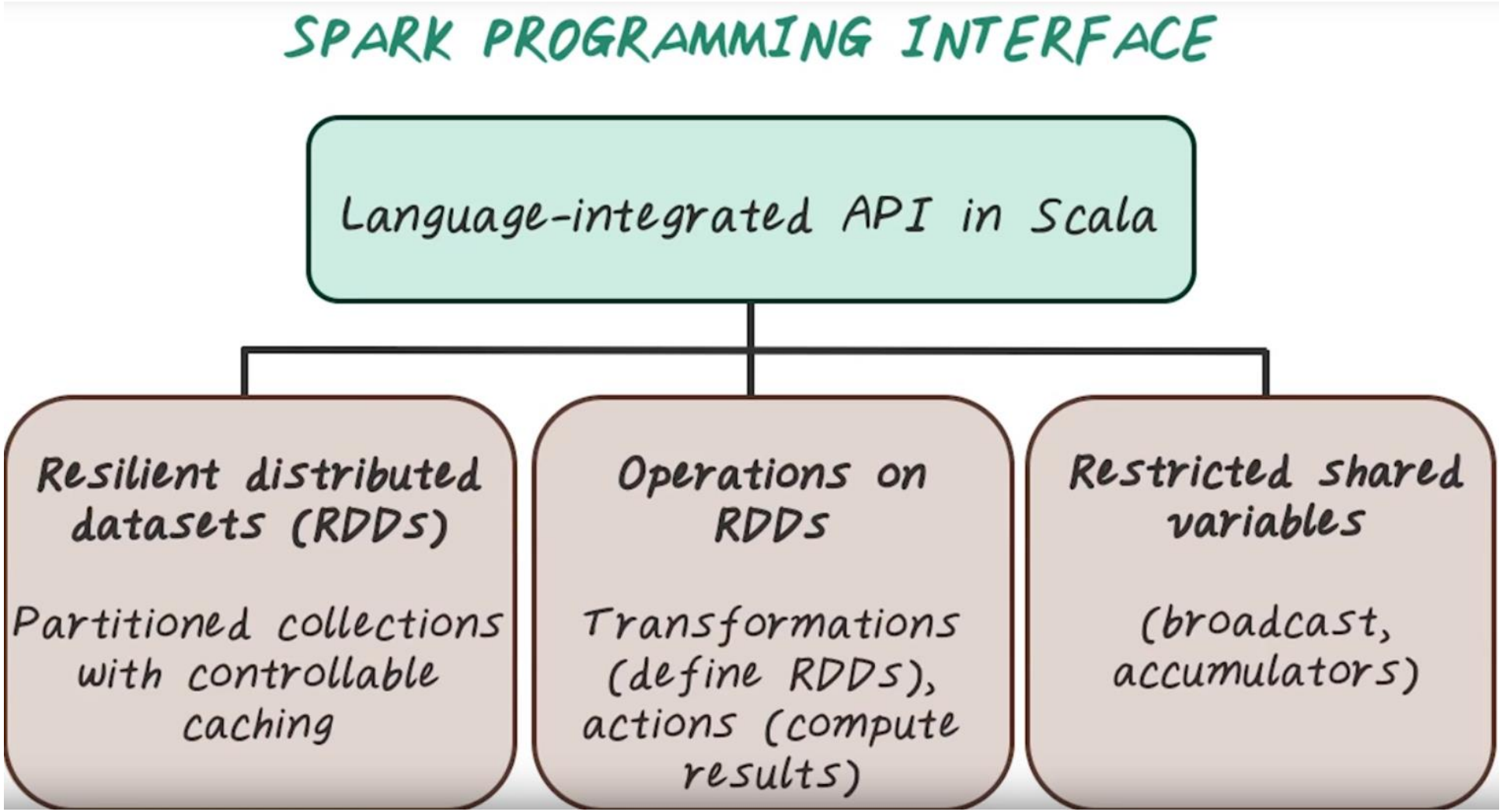




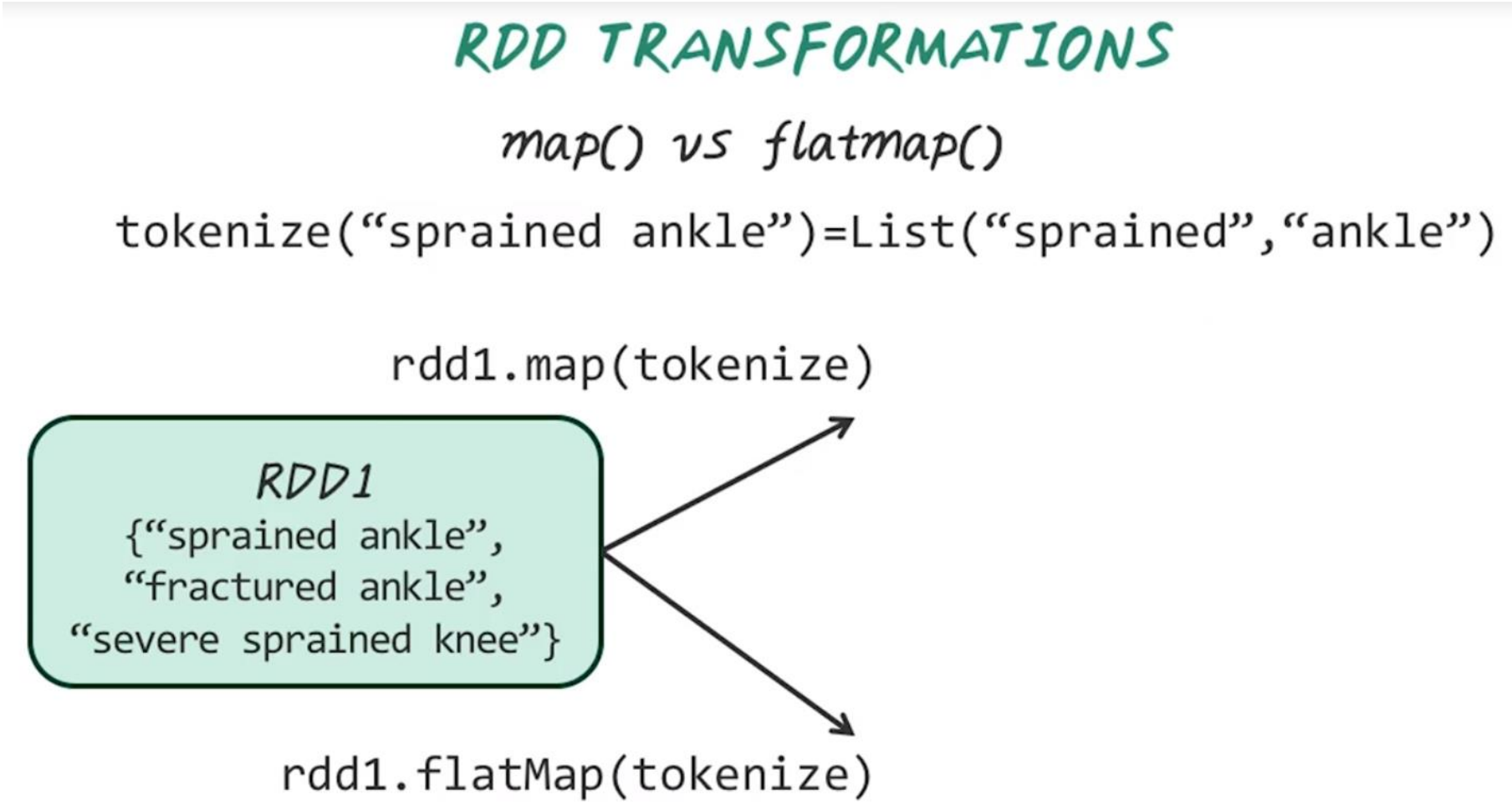
9. Spark Stack



10. Spark Programming Interface



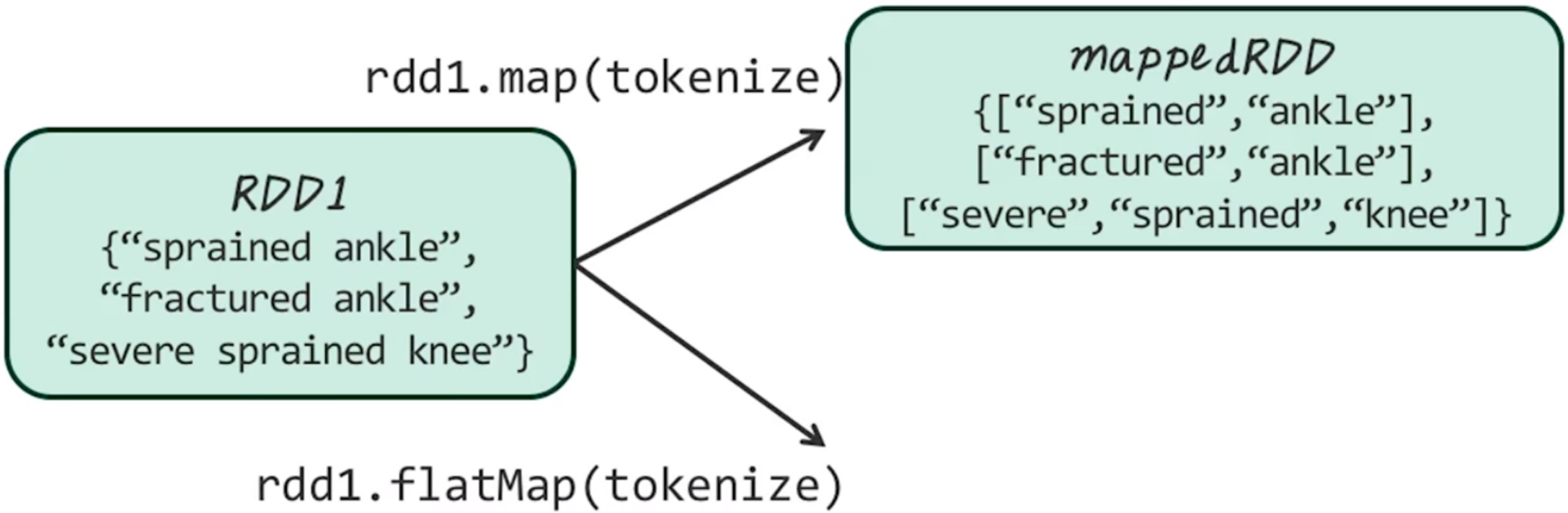
11. RDD Transformations



RDD TRANSFORMATIONS

map() vs flatmap()

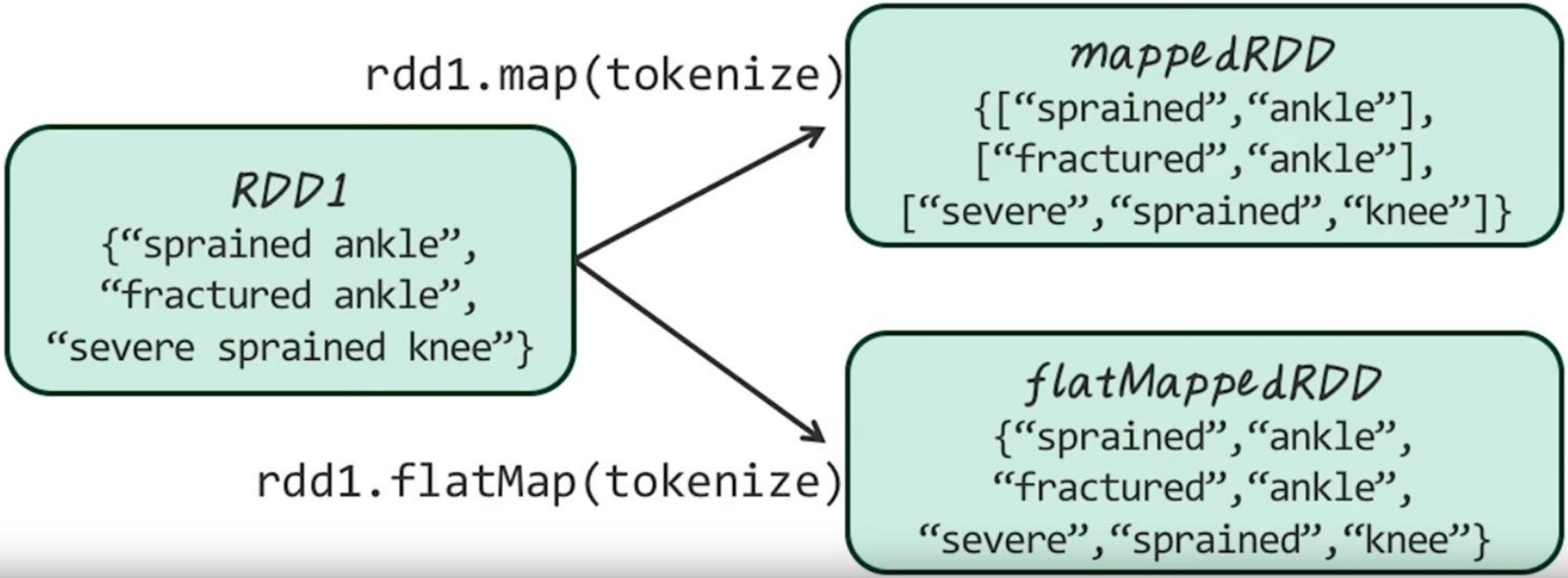
tokenize("sprained ankle")=List("sprained","ankle")



RDD TRANSFORMATIONS

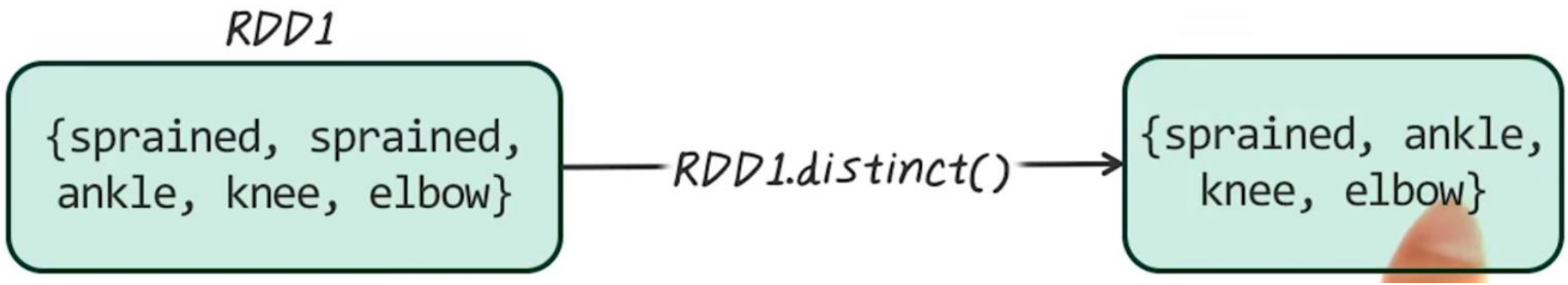
map() vs flatmap()

tokenize("sprained ankle")=List("sprained","ankle")

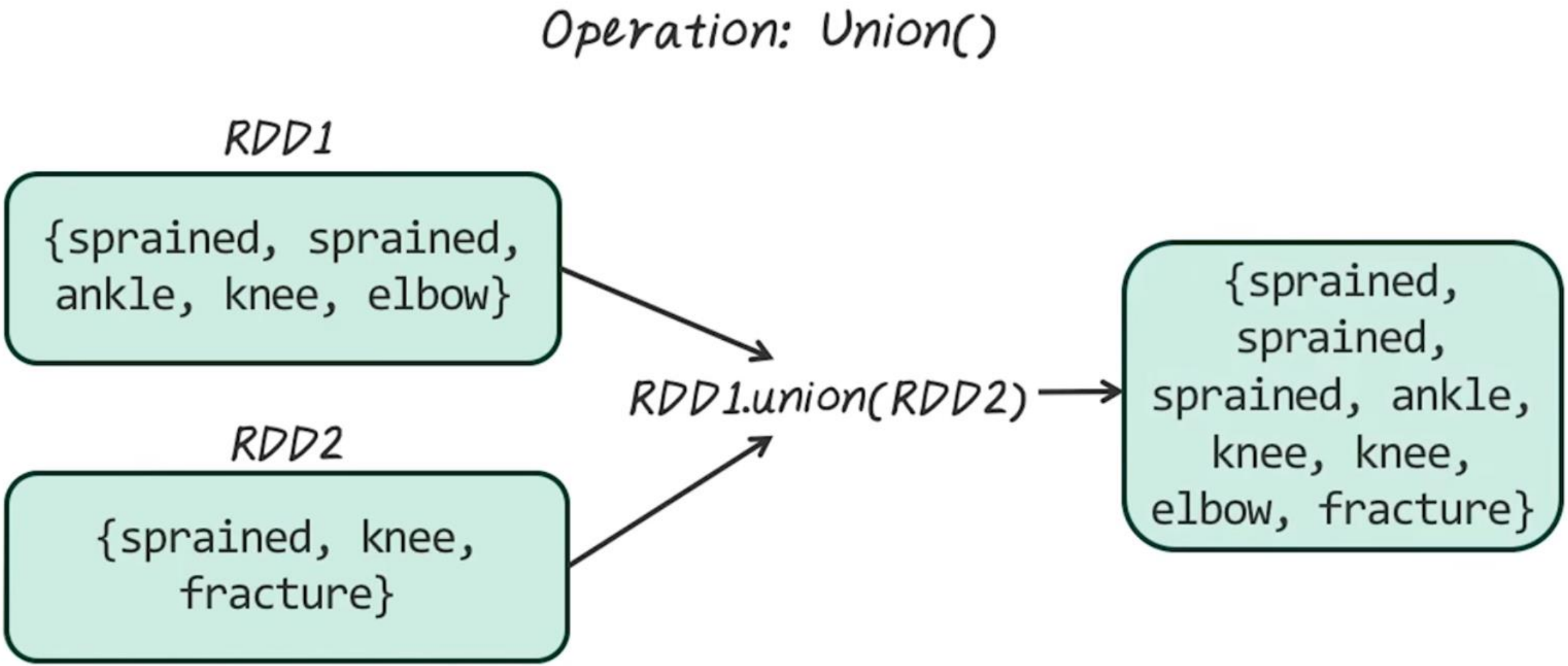


RDD TRANSFORMATIONS

Operation: Distinct()

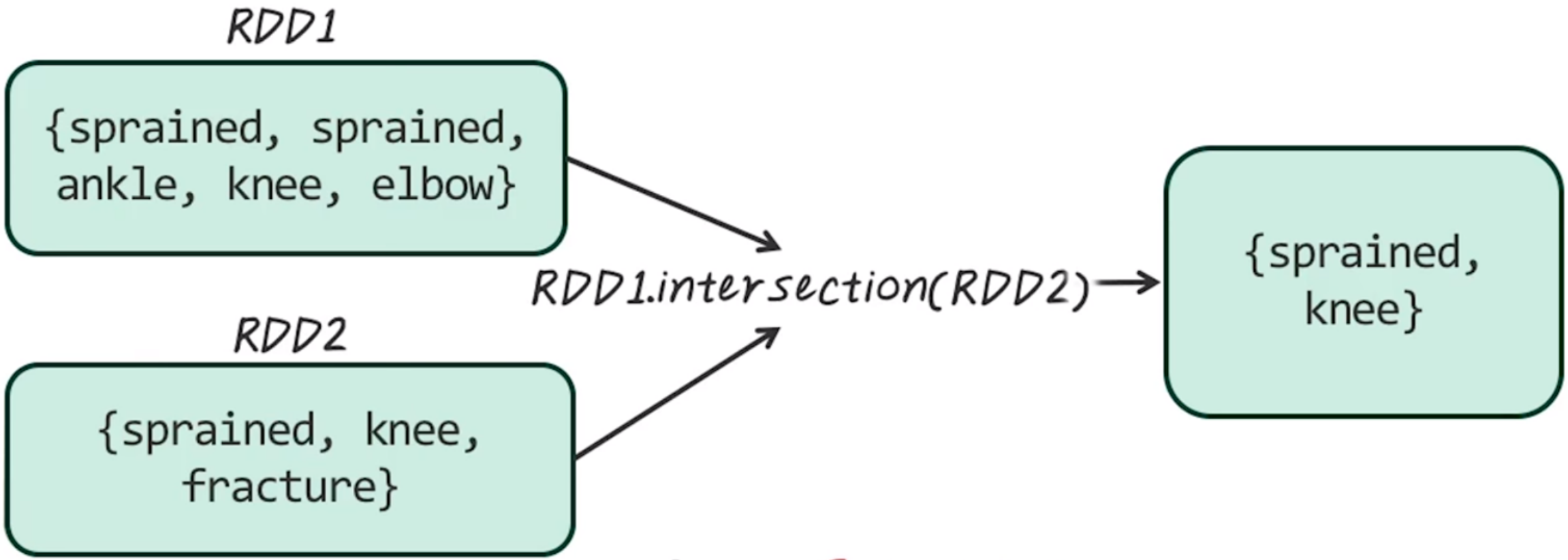


RDD TRANSFORMATIONS



RDD TRANSFORMATIONS

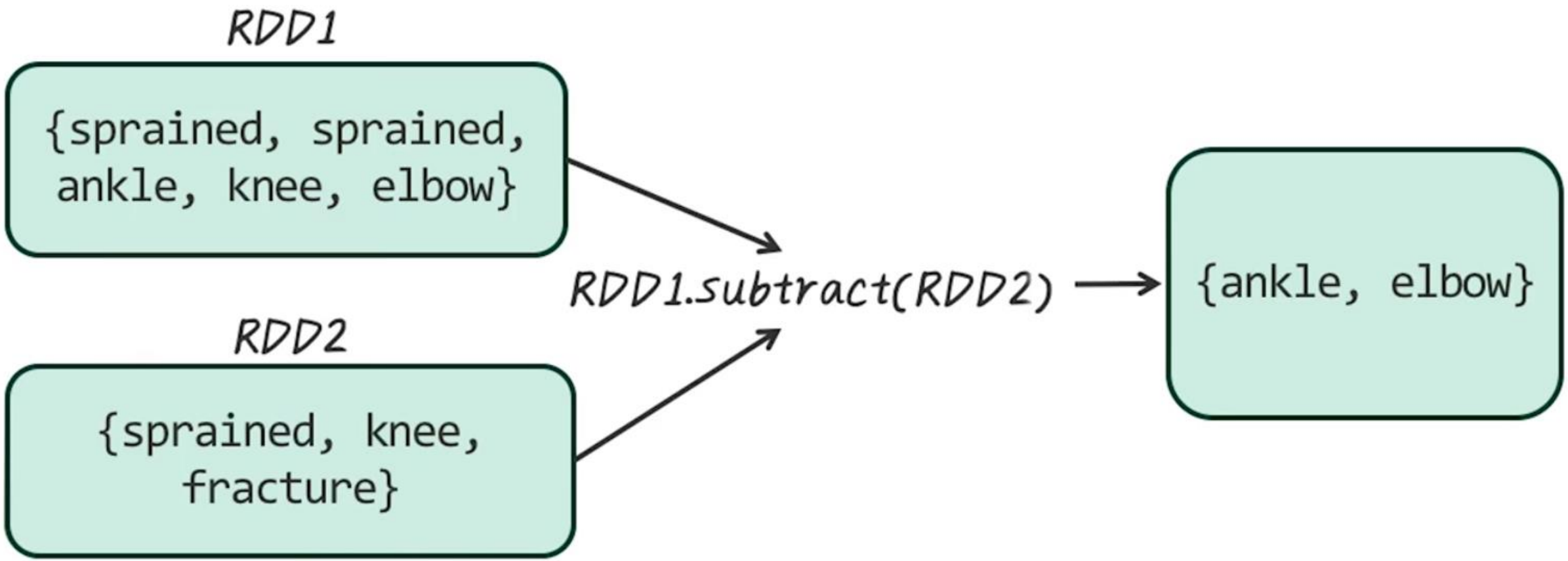
Operation: Intersection()



Cost: Expensive

RDD TRANSFORMATIONS

Operation: Subtract()



12. Basic RDD Transformations

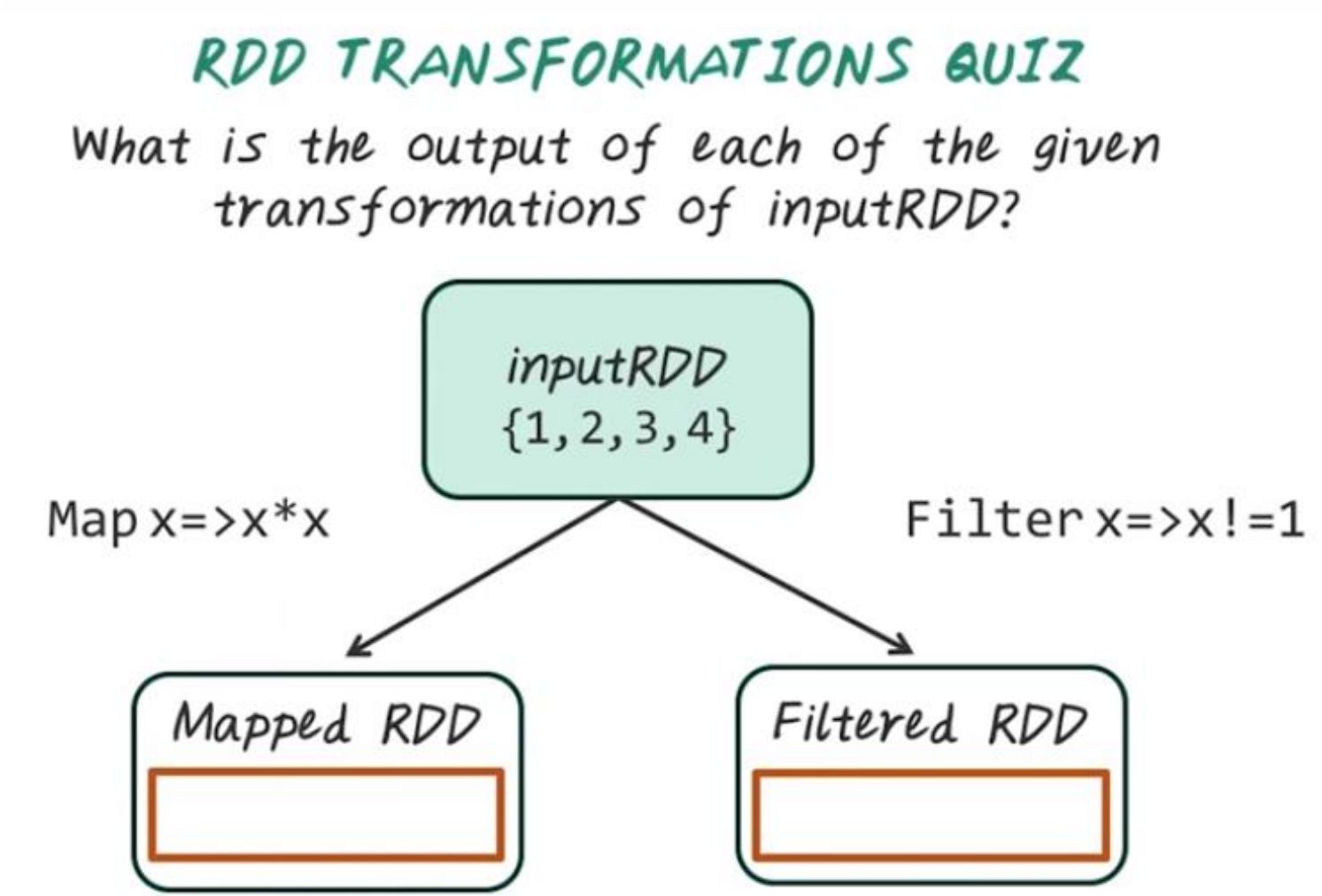
Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic

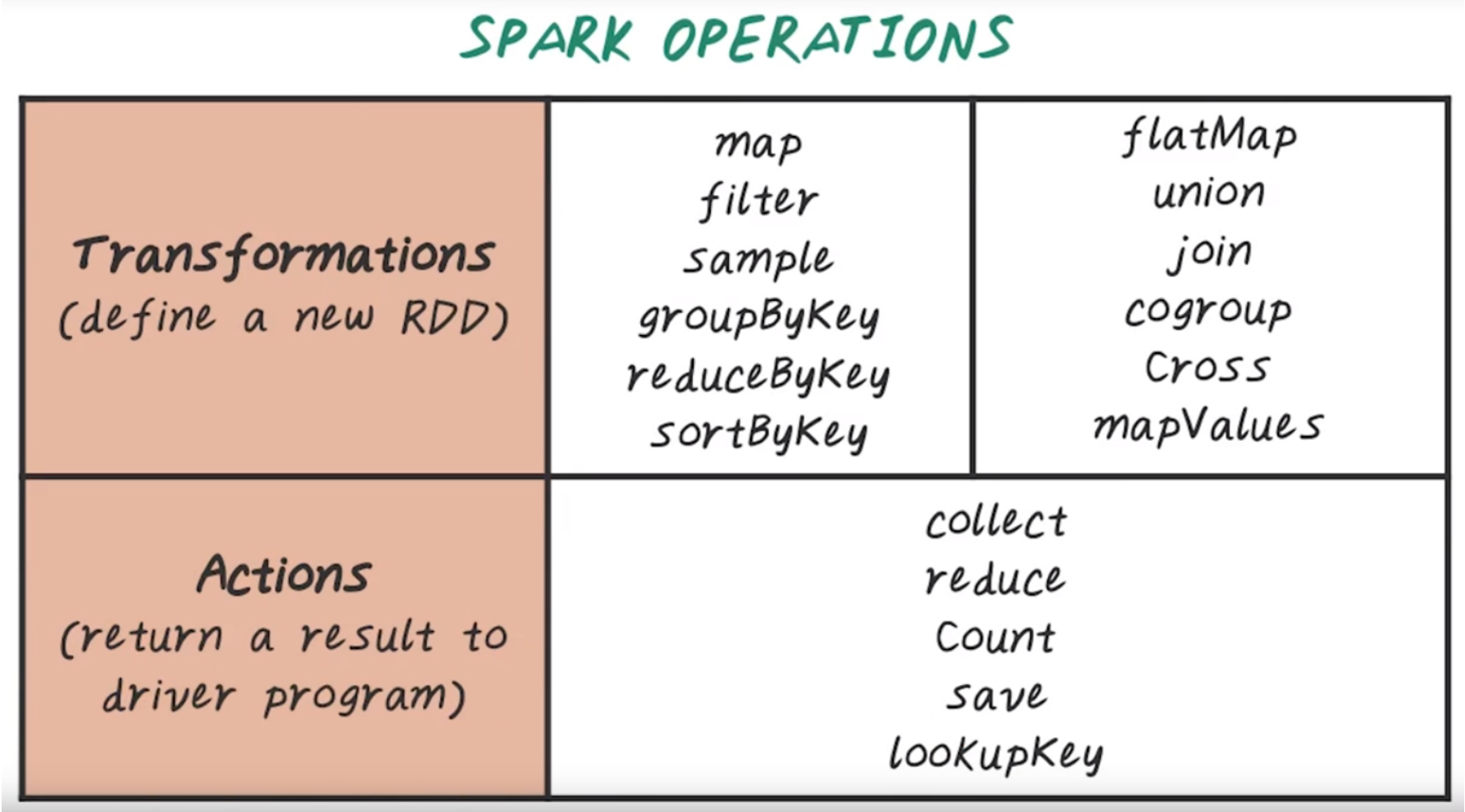
Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

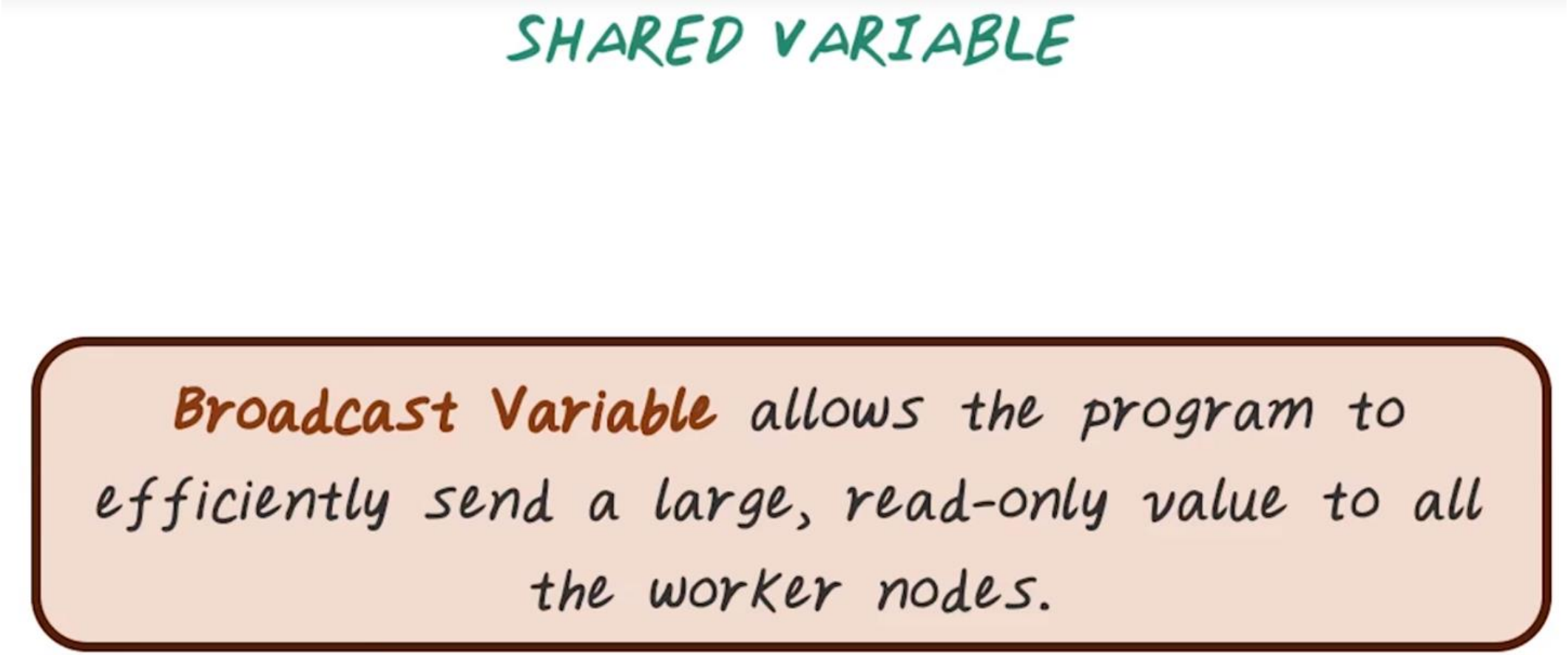
13. RDD Transformations Quiz



14. Spark Operations



15. Shared Variable

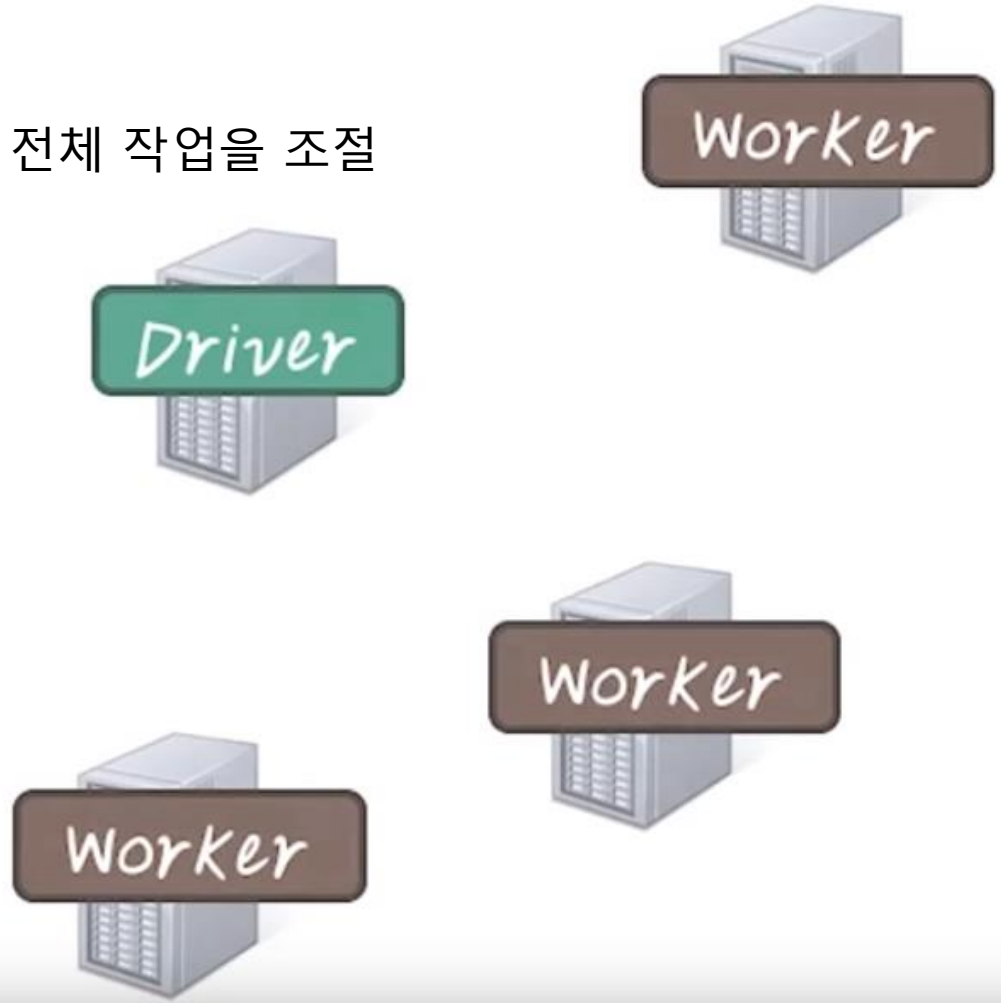


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

실제 작업을 수행

전체 작업을 조절



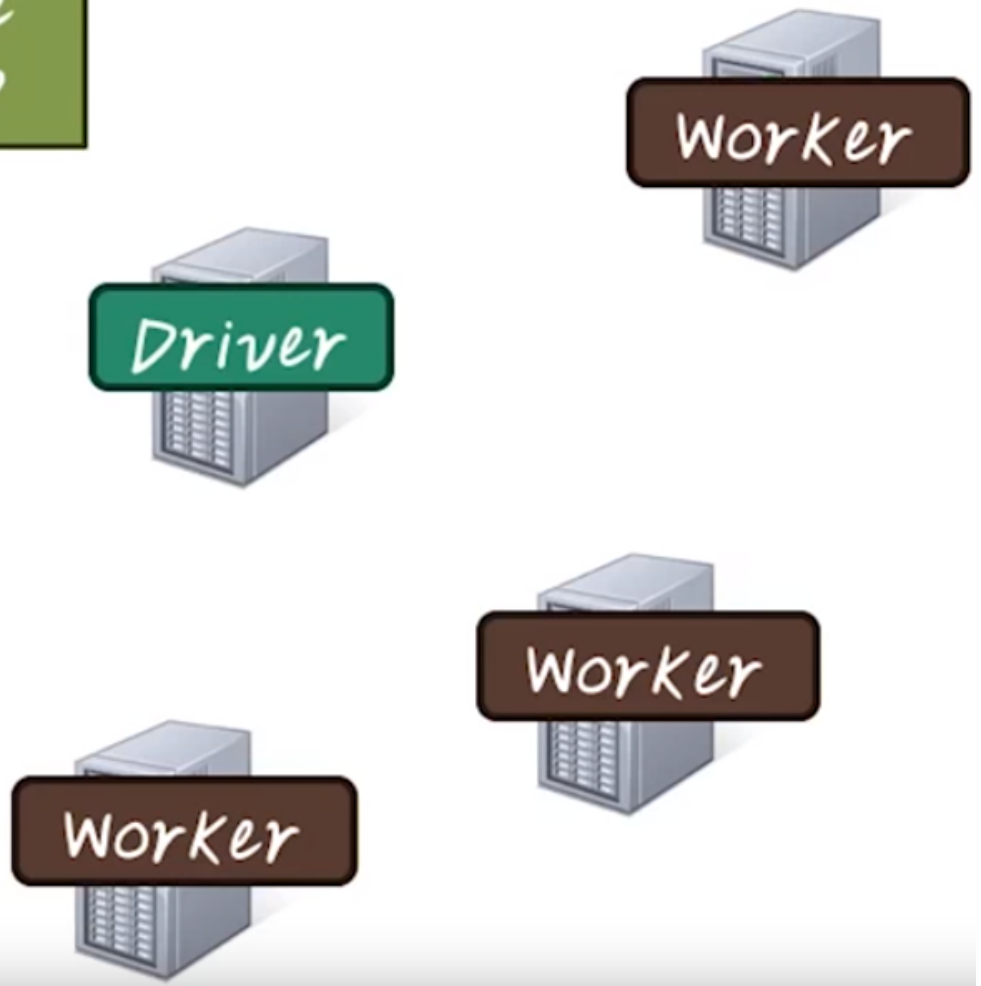
EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
```

Base RDD

하둡 분산 파일시스템으로부터
임상데이터를 읽어옴.

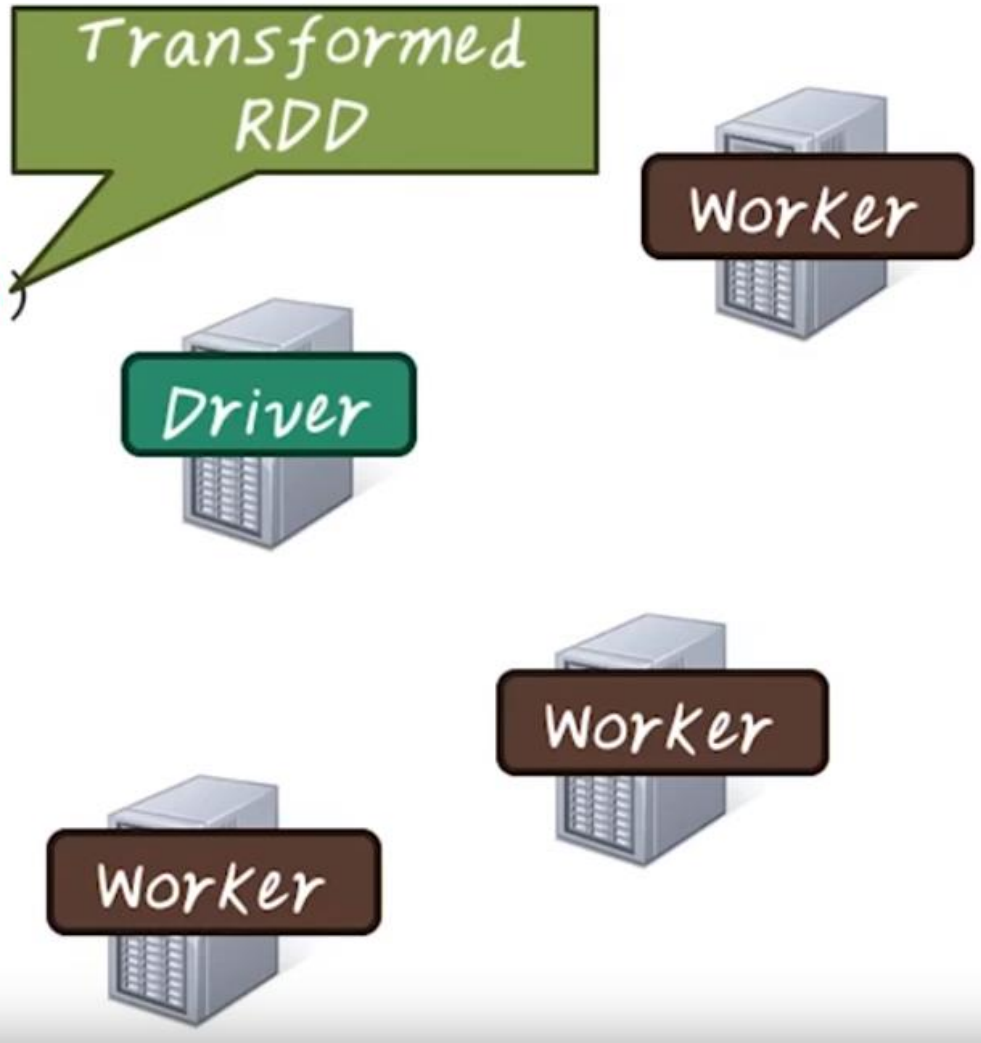


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
symptoms= lines.filter(_.startsWith("SYMPTOM"))
```

SYMPTOM 이라는 단어로 시작하는 라인만을 추출

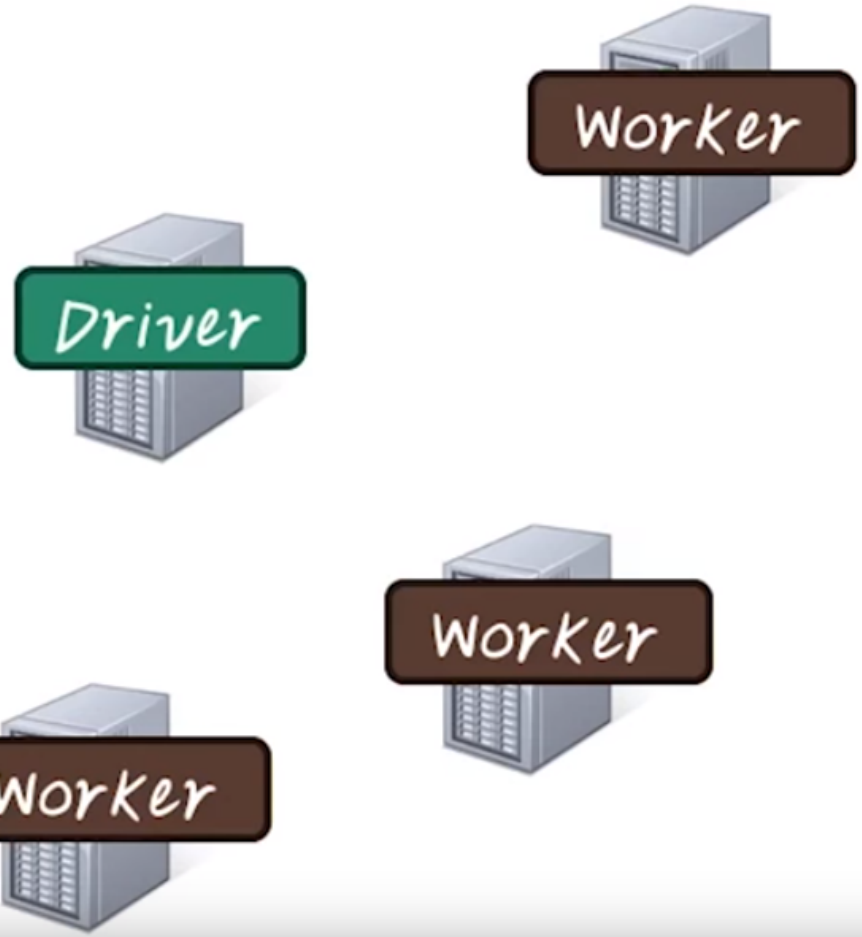


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
symptoms= lines.filter(_.startsWith(“SYMPTOM”))
messages = symptoms.map(_.split(‘\t’)(2))
```

SYMPTOM으로 시작하는 라인을 탭문자을
기준으로 자르고 2번째 단어만을 추출
=> SYMPTOM 이름

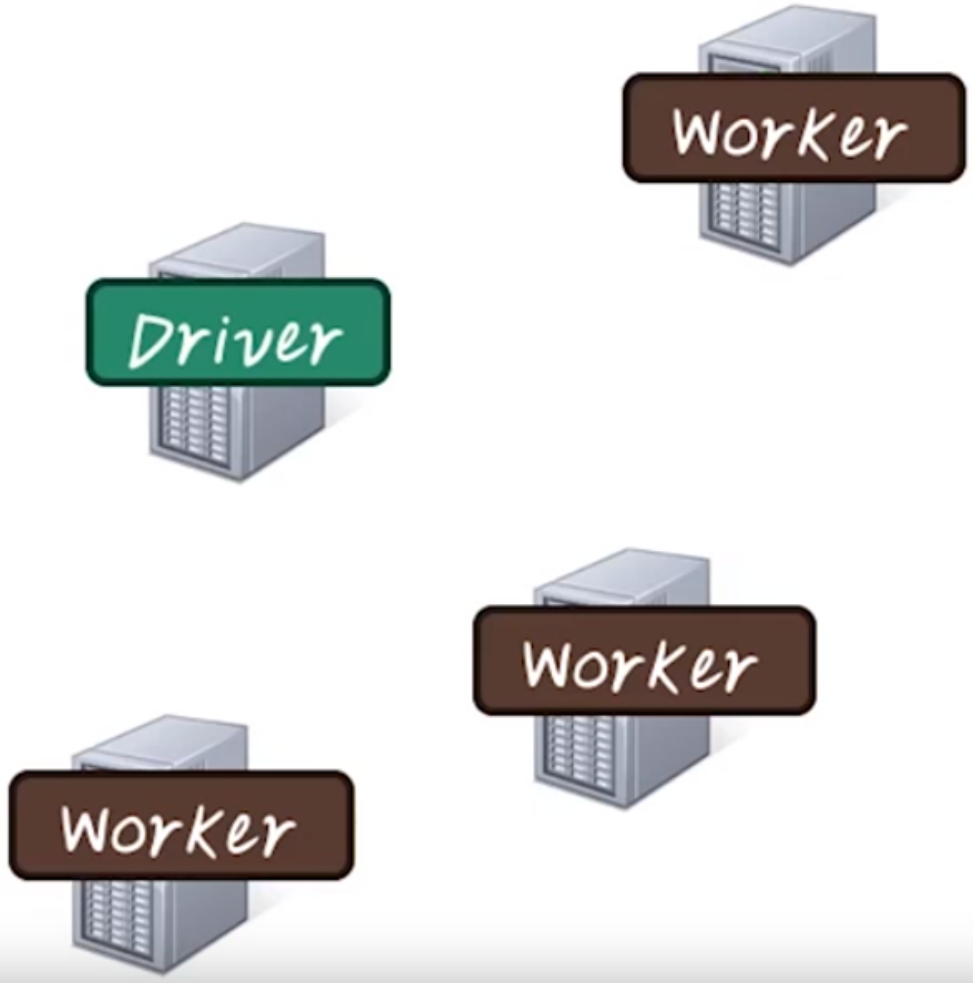


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
symptoms= lines.filter(_.startsWith(“SYMPTOM”))
messages = symptoms.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()
```

SYMPTOM 이름까지 추출한 결과를 메모리에 저장
Action이 일어날때까지는 아무일도 일어나지 않음.

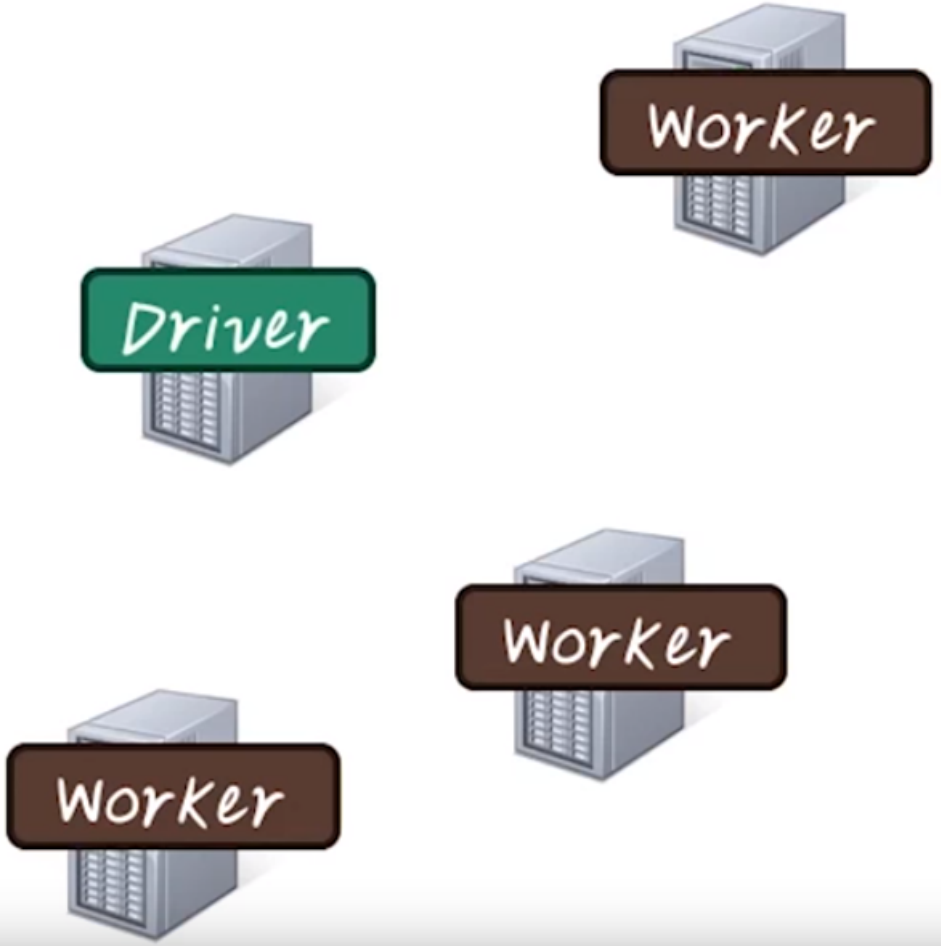


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
symptoms= lines.filter(_.startsWith("SYMPTOM"))
messages = symptoms.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("fever")).count
```

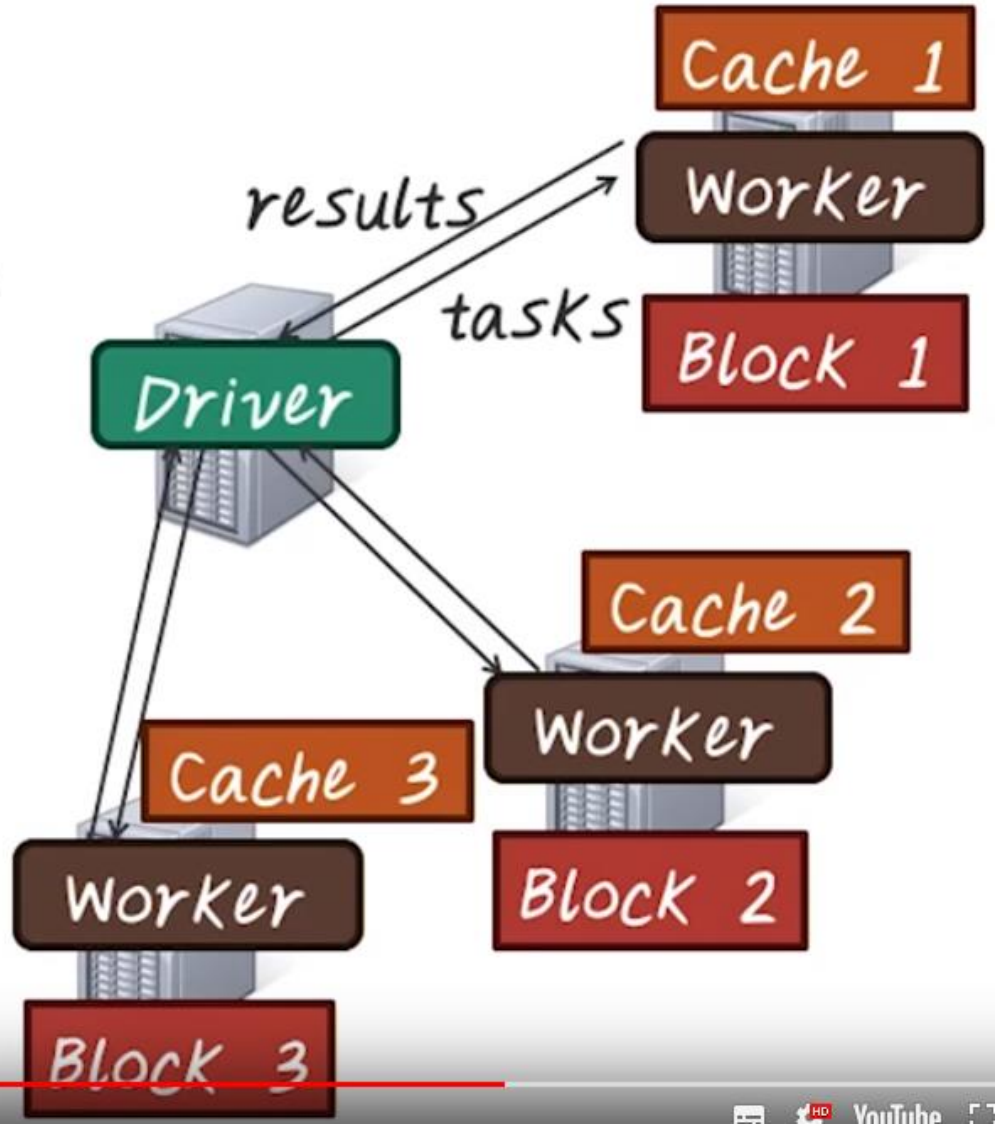
SYMPTOM 이름이 fever인 경우를 카운트
Count는 actio이라서 실제 작업이 일어남.



EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
symptoms= lines.filter(_.startsWith("SYMPTOM"))
messages = symptoms.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("fever")).count
cachedMsgs.filter(_.contains("cough")).count
```

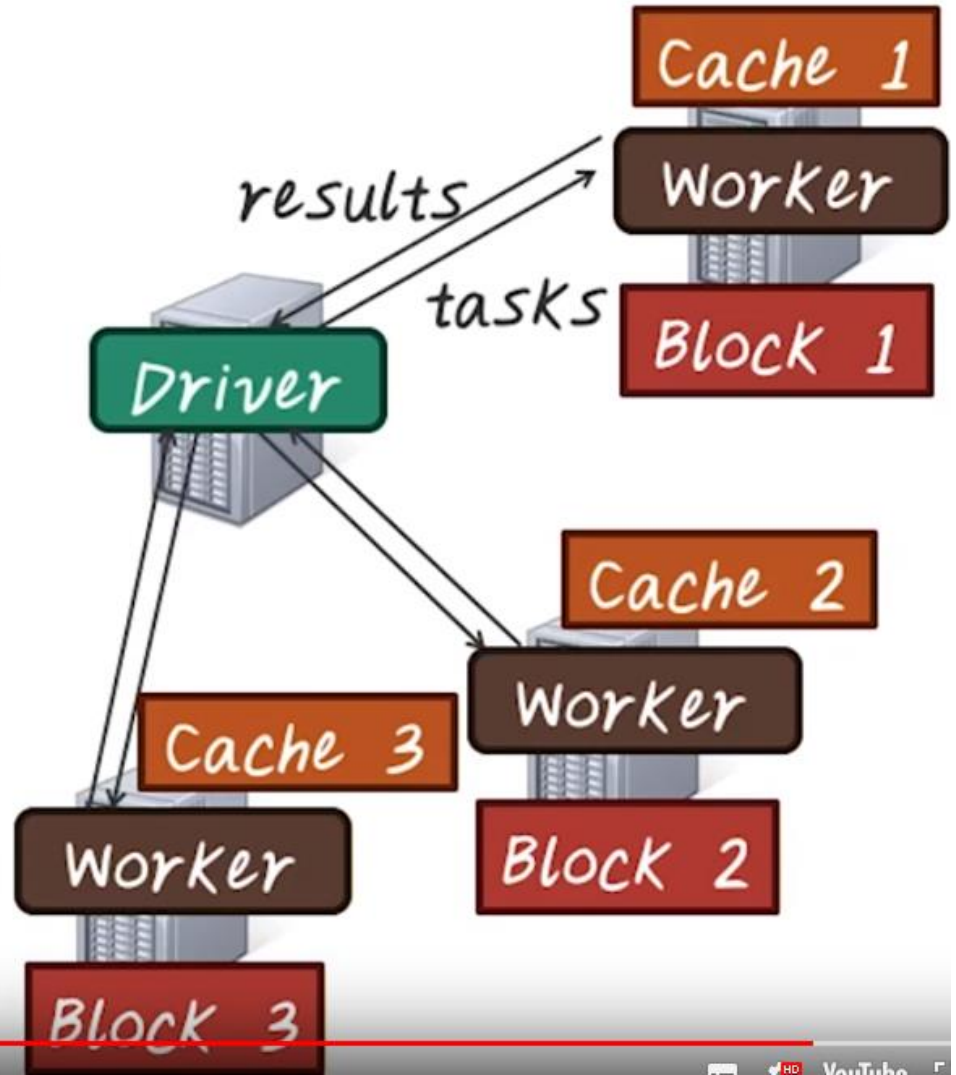


EXAMPLE: MINING CLINICAL NOTES

Load clinical notes into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
symptoms= lines.filter(_.startsWith(“SYMPTOM”))
messages = symptoms.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“fever”)).count
cachedMsgs.filter(_.contains(“cough”)).count
    . . .
```

Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)



16. Fault Tolerance

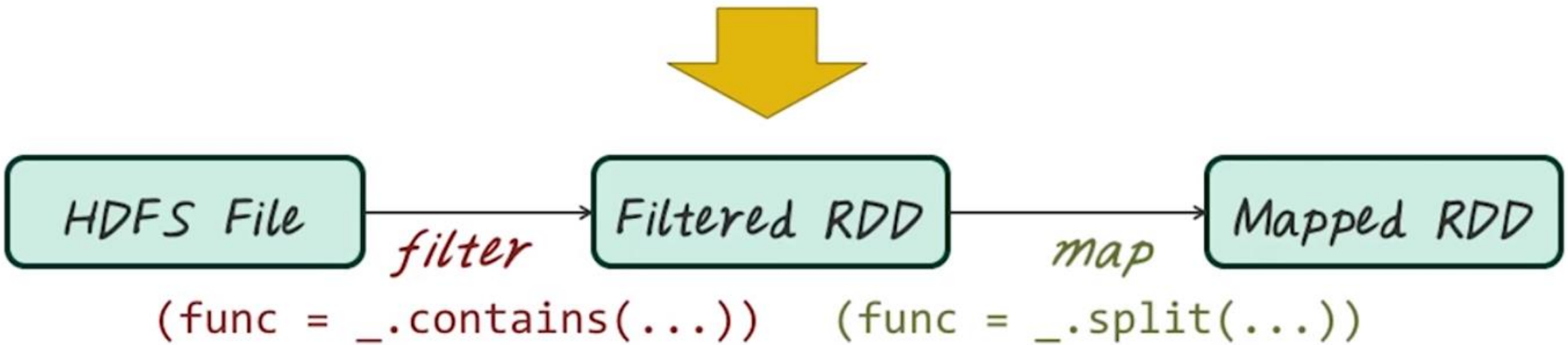
FAULT TOLERANCE

RDDs track *lineage* information that can be used to efficiently reconstruct lost partitions.

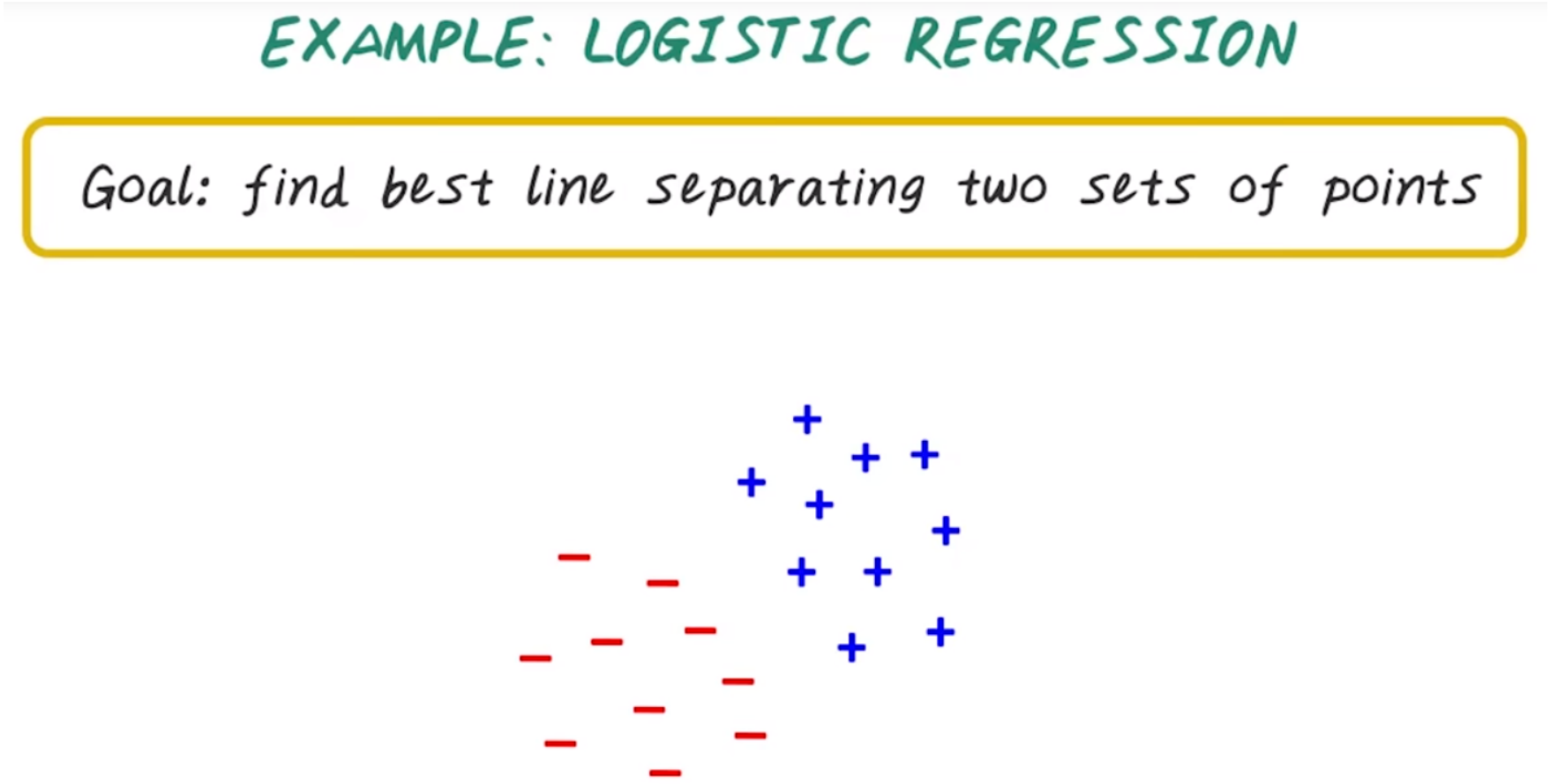
FAULT TOLERANCE

RDDs track *lineage* information that can be used to efficiently reconstruct lost partitions.

Ex:
```
messages = textFile(...).filter(_.startsWith("SYMPTOM"))
                        .map(_.split('\t')(2))
```

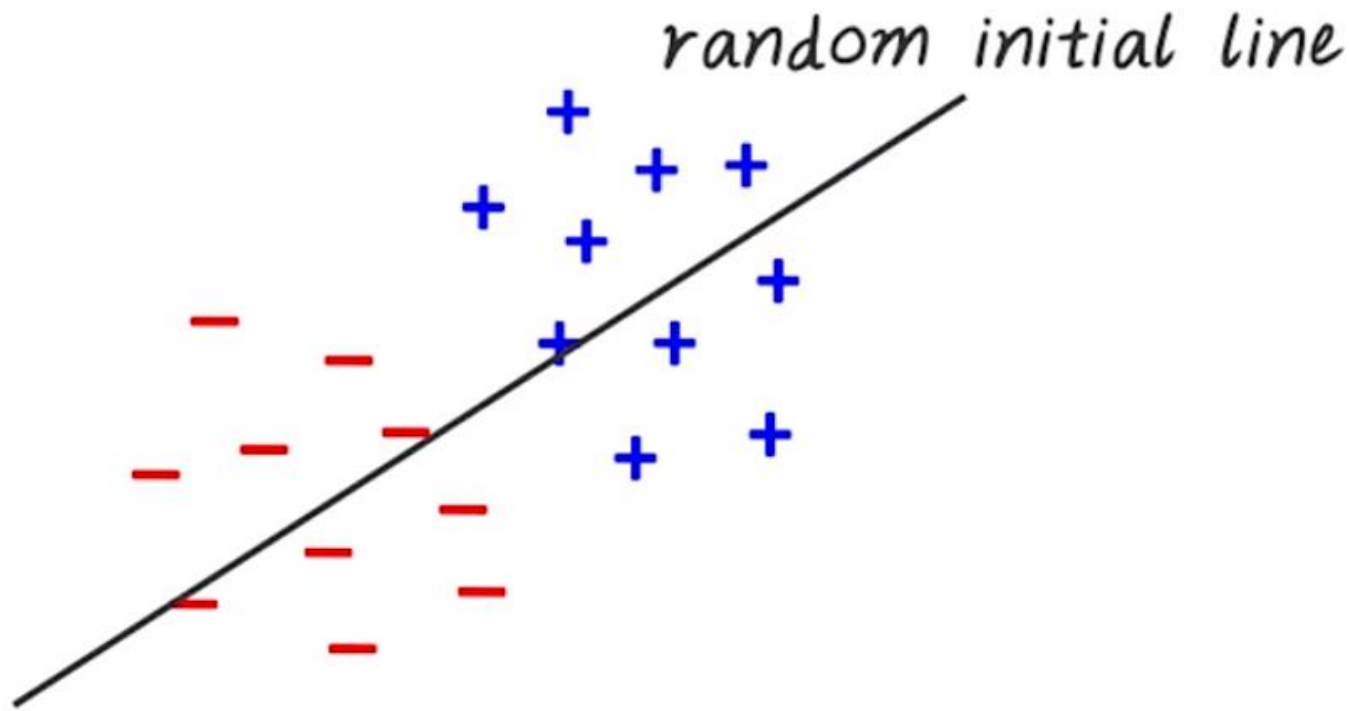


17. Example Logistic Regression



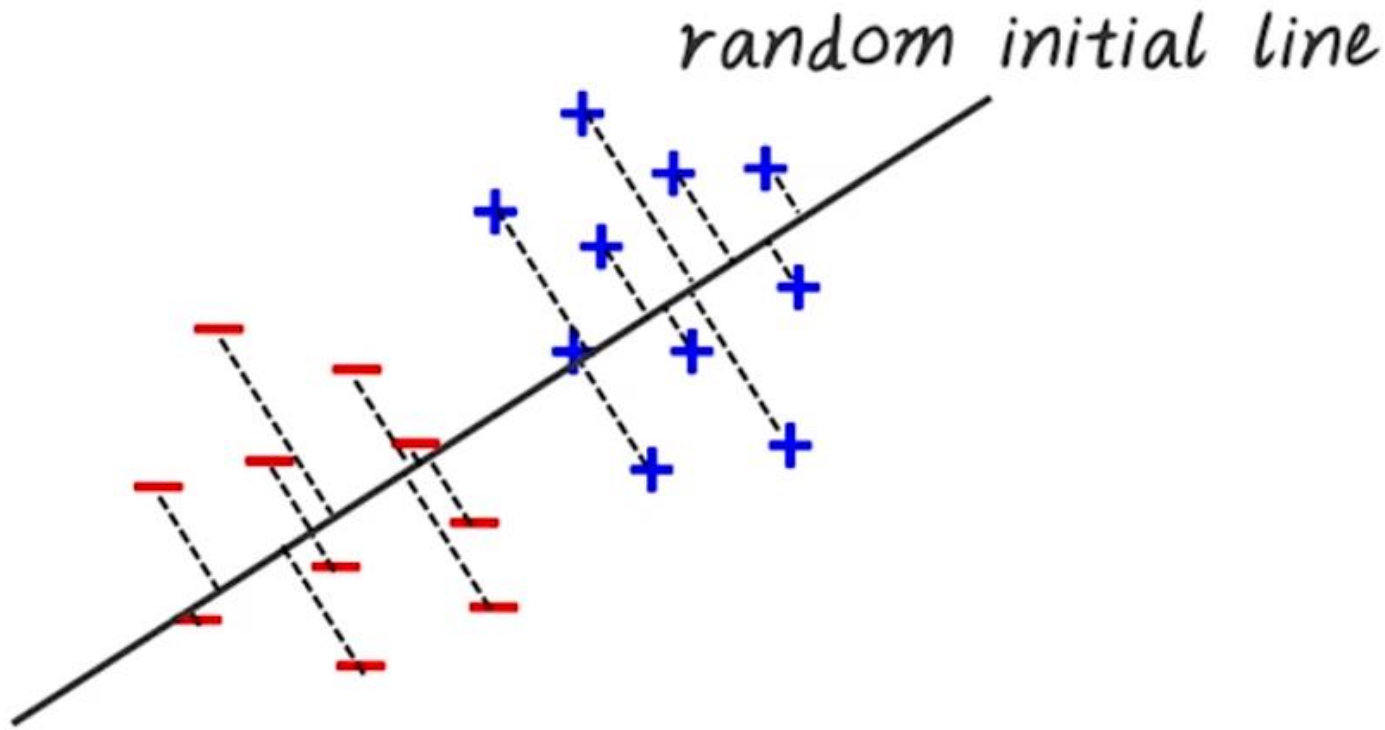
EXAMPLE: LOGISTIC REGRESSION

Goal: find best line separating two sets of points



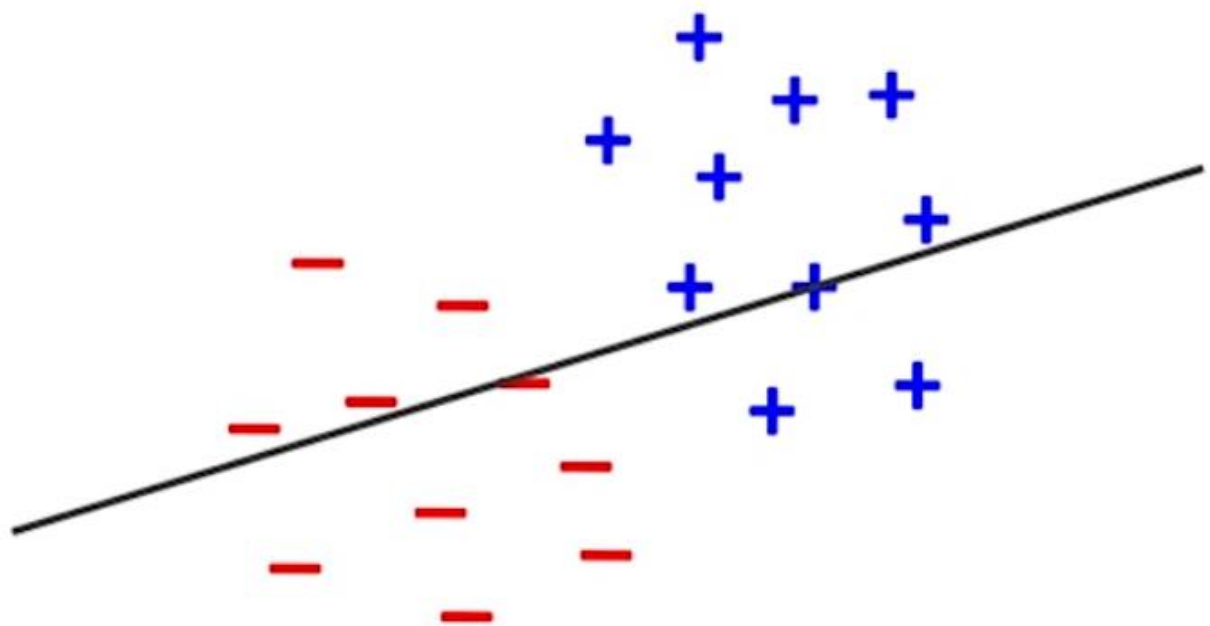
EXAMPLE: LOGISTIC REGRESSION

Goal: find best line separating two sets of points



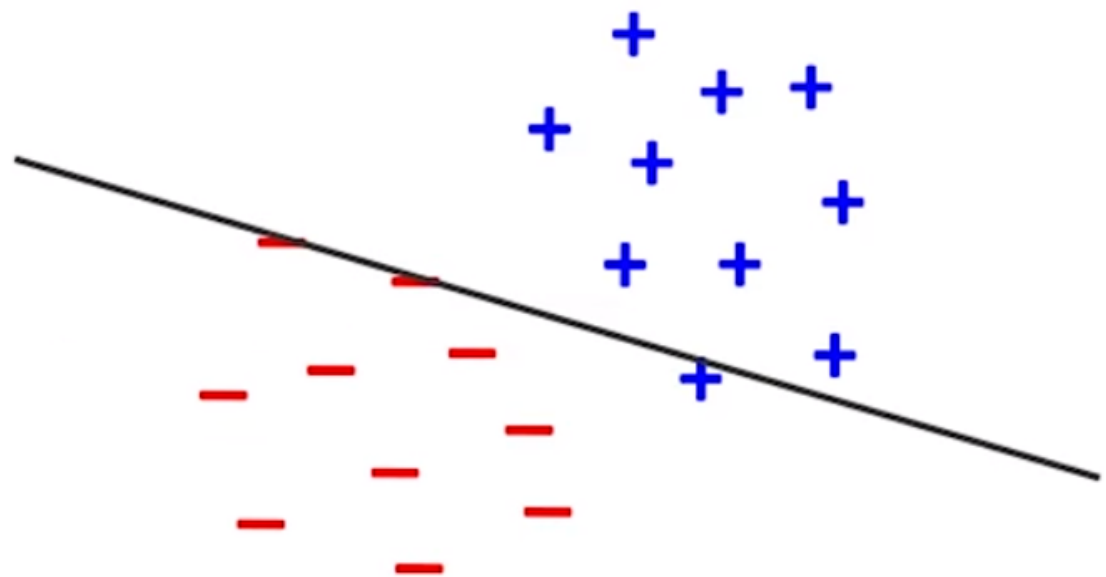
EXAMPLE: LOGISTIC REGRESSION

Goal: find best line separating two sets of points



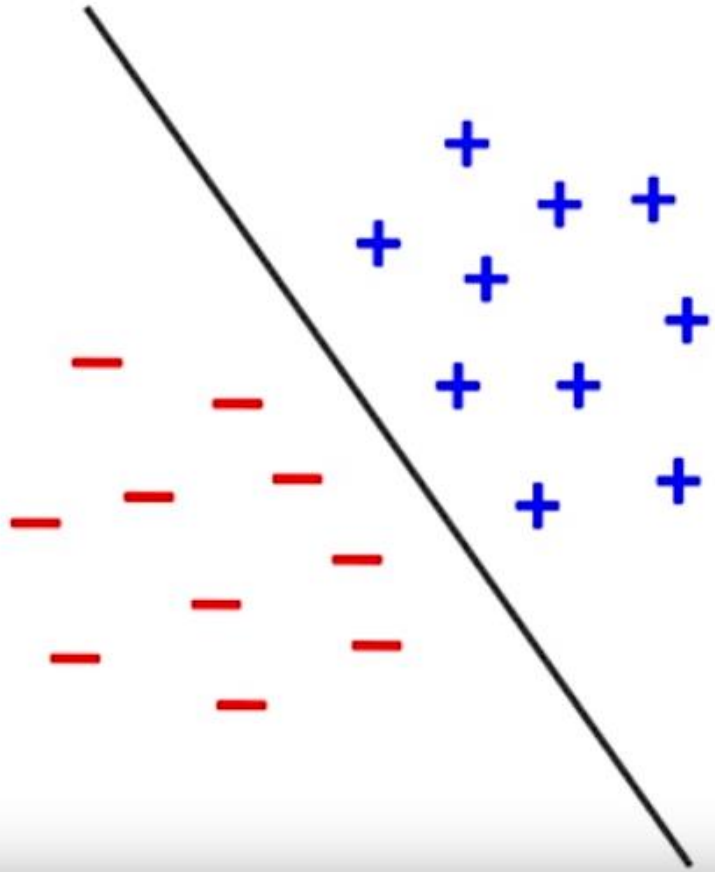
EXAMPLE: LOGISTIC REGRESSION

Goal: find best line separating two sets of points



EXAMPLE: LOGISTIC REGRESSION

Goal: find best line separating two sets of points



EXAMPLE: LOGISTIC REGRESSION

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

EXAMPLE: LOGISTIC REGRESSION

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Load data in memory once

EXAMPLE: LOGISTIC REGRESSION

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

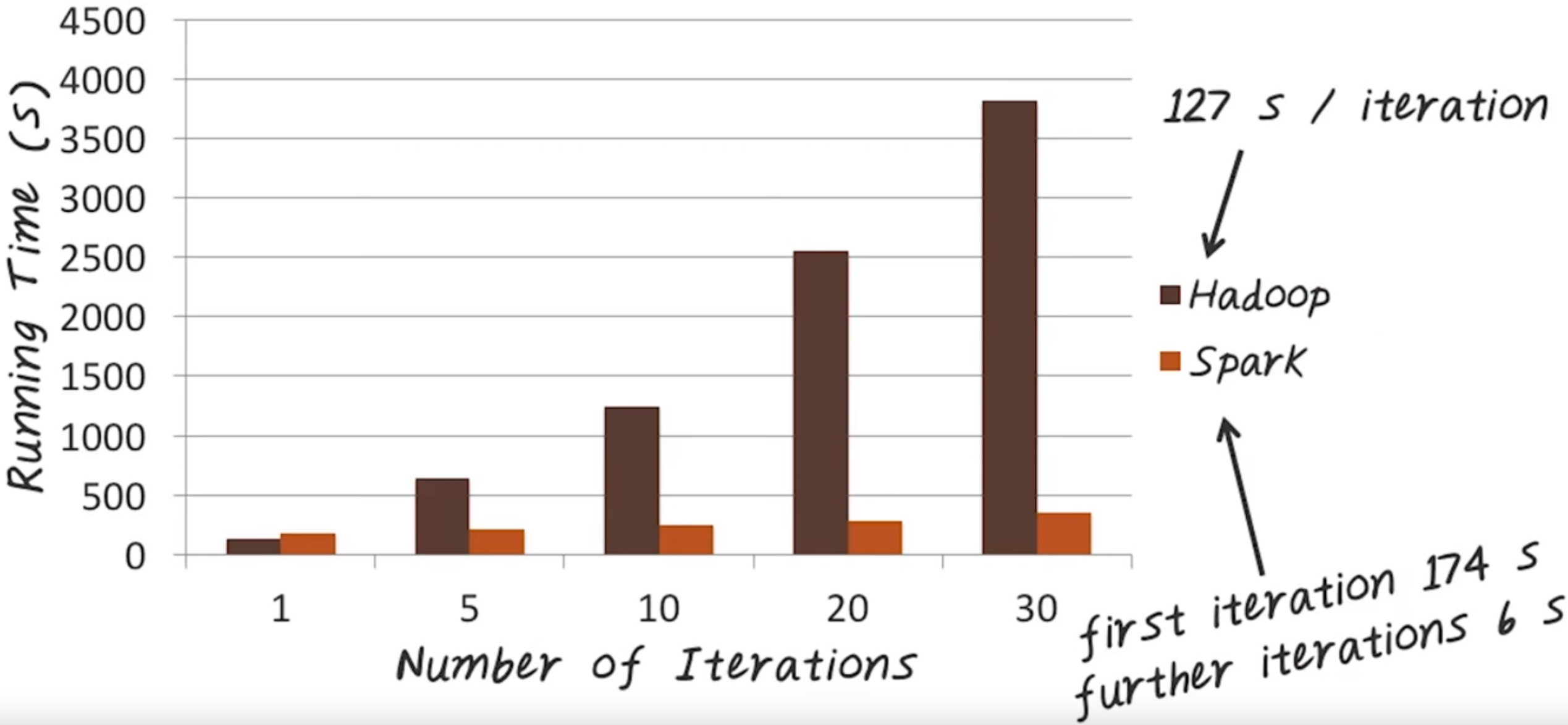
Initial parameter vector

EXAMPLE: LOGISTIC REGRESSION

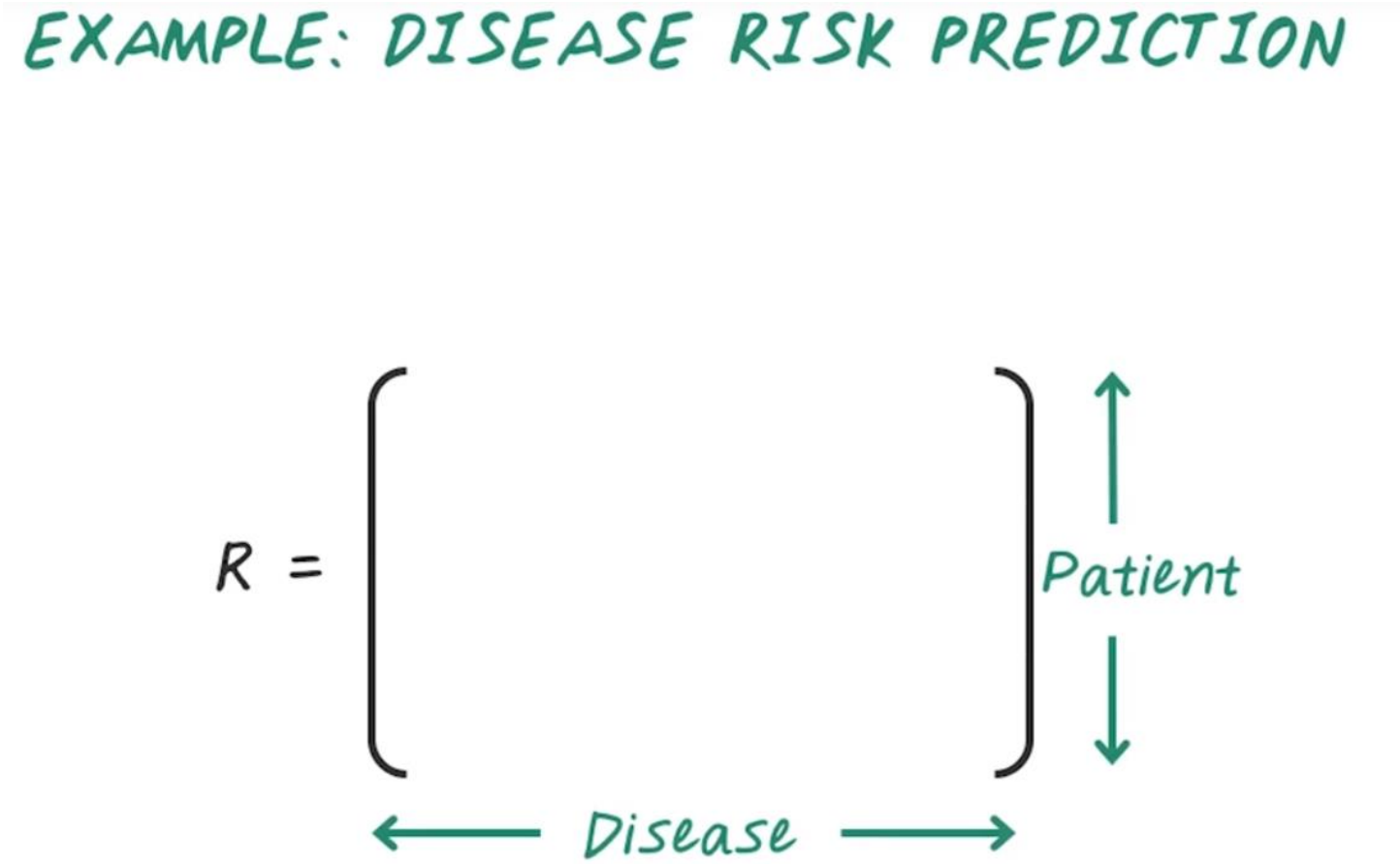
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Repeated MapReduce steps to do gradient descent

LOGISTIC REGRESSION PERFORMANCE



18. Example Disease Risk Prediction



EXAMPLE: DISEASE RISK PREDICTION

Goal: predict disease risk based on existing disease diagnosis

EXAMPLE: DISEASE RISK PREDICTION

Goal: predict disease risk based on existing disease diagnosis

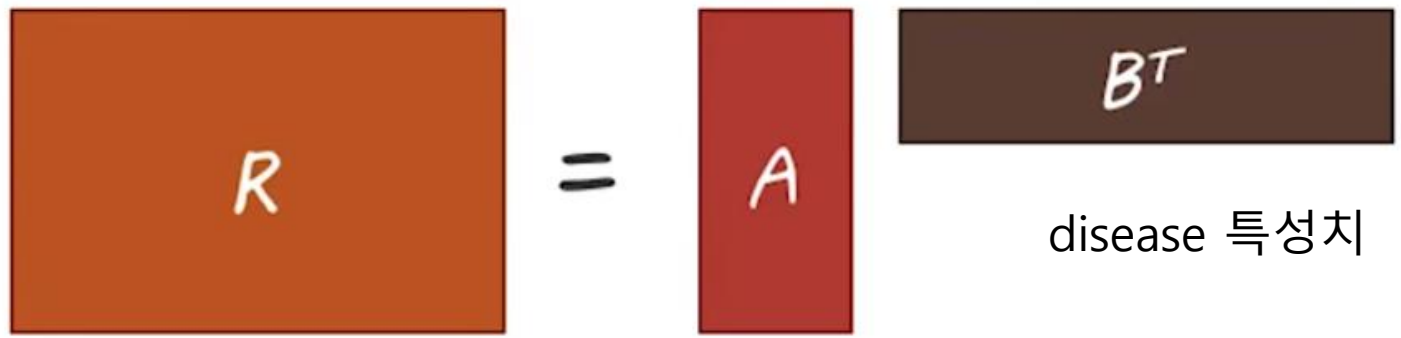
$$R = \begin{pmatrix} 1 & ? & ? & 1 & 1 & ? & 1 \\ ? & ? & 1 & 1 & ? & ? & 1 \\ 1 & ? & 1 & ? & ? & ? & 1 \\ 1 & ? & ? & ? & ? & 1 & ? \end{pmatrix}$$

Patient ↑ ↓

← Disease →

MODEL AND ALGORITHM

Model R as product of patient and disease feature matrices A and B of size $U \times K$ and $M \times K$

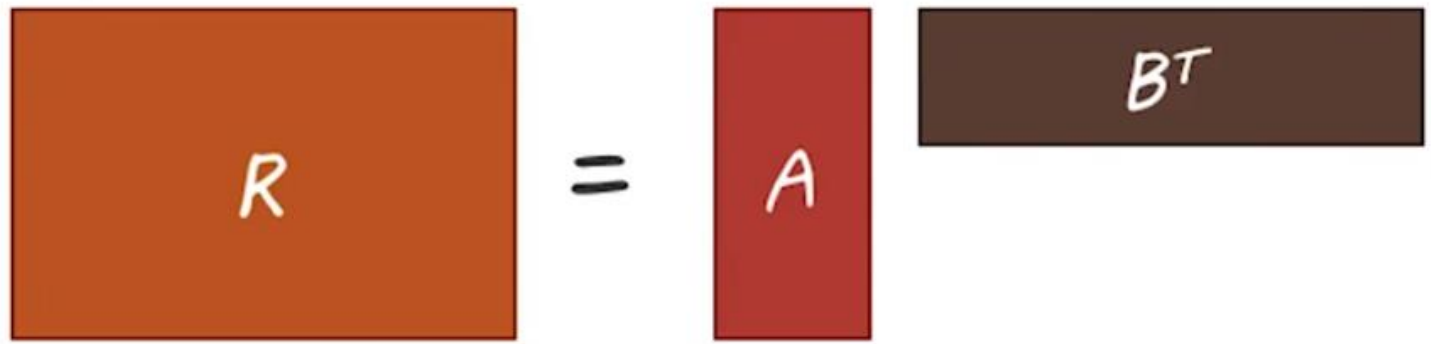


disease 특성치

Patient 특성치

MODEL AND ALGORITHM

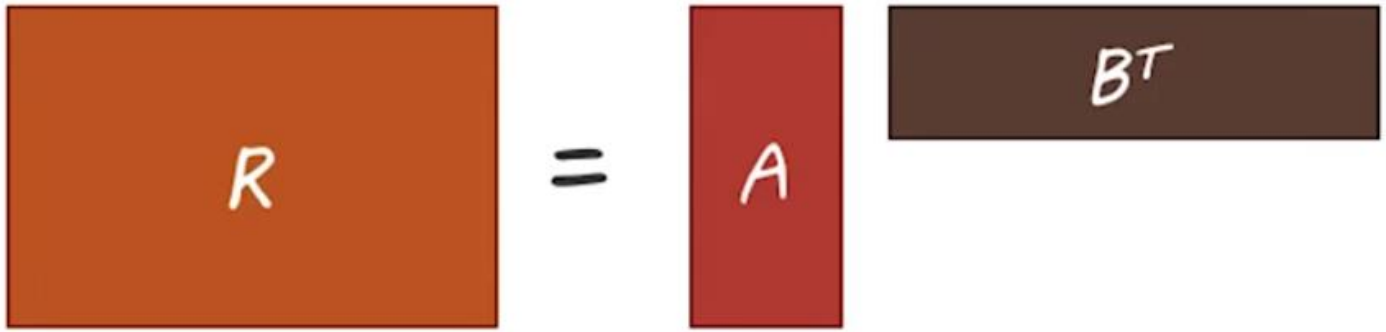
Model R as product of patient and disease feature matrices A and B of size $U \times K$ and $M \times K$



Alternating Least Squares (ALS)

MODEL AND ALGORITHM

Model R as product of patient and disease feature matrices A and B of size $U \times K$ and $M \times K$



Alternating Least Squares (ALS)

1. Start with random A & B
2. Optimize patient vectors (A) based on diseases
3. Optimize disease vectors (B) based on patients
4. Repeat until converged

19. Serial ALS

SERIAL ALS

```
var R = readDataMatrix(...)

var A = // array of U random vectors
var B = // array of M random vectors

for (i <- 1 to ITERATIONS) {
  A = (0 until U).map(i => updatePatient(i, B, R))
  B = (0 until M).map(i => updateDisease(i, A, R))
}
```

20. Naive Spark ALS

NAIVE SPARK ALS

```
var R = readDataMatrix(...)

var A = // array of U random vectors
var B = // array of M random vectors

for (i <- 1 to ITERATIONS) {
  A = spark.parallelize(0 until U, numSlices)
           .map(i => updatePatient(i, B, R))
           .collect()
  B = spark.parallelize(0 until M, numSlices)
           .map(i => updateDisease(i, A, R))
           .collect()
}
```

NAIVE SPARK ALS

```
var R = readDataMatrix(...)

var A = // array of U random vectors
var B = // array of M random vectors

for (i <- 1 to ITERATIONS) {
  A = spark.parallelize(0 until U, numSlices)
           .map(i => updatePatient(i, B, R))
           .collect()
  B = spark.parallelize(0 until M, numSlices)
           .map(i => updateDisease(i, A, R))
           .collect()
}
```

Problem: R re-sent to all nodes in each iteration

EFFICIENT SPARK ALS

```
var R = spark.broadcast(readDataMatrix(...))

var A = // array of U random vectors
var B = // array of M random vectors

for (i <- 1 to ITERATIONS) {
  A = spark.parallelize(0 until U, numSlices)
           .map(i => updatePatient(i, B, R.value))
           .collect()
  B = spark.parallelize(0 until M, numSlices)
           .map(i => updateDisease(i, A, R.value))
           .collect()
}
```

EFFICIENT SPARK ALS

```
var R = spark.broadcast(readDataMatrix(...))

var A = // array of U random vectors
var B = // array of M random vectors

for (i <- 1 to ITERATIONS) {
  A = spark.parallelize(0 until U, numSlices)
           .map(i => updatePatient(i, B, R.value))
           .collect()
  B = spark.parallelize(0 until M, numSlices)
           .map(i => updateDisease(i, A, R.value))
           .collect()
}
```

Solution: mark R as broadcast variable

Result: 3× performance improvement