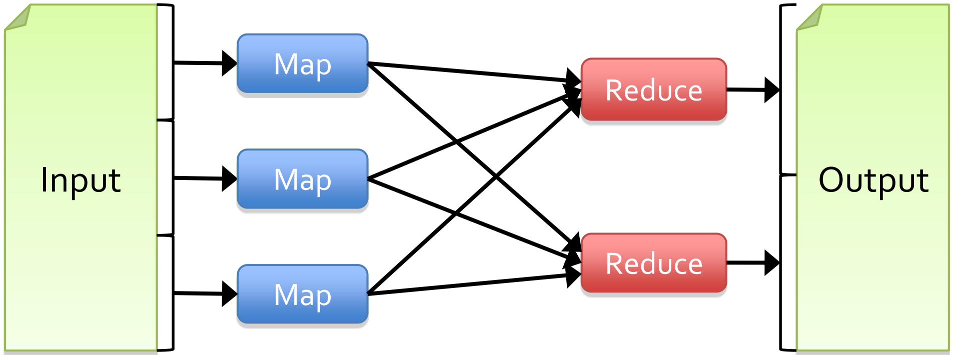
Spark

Bakgrunn og motivasjon

- Observasjon: MapReduce gjorde analyse av Big Data svært mykje enklare ved å skjule aspekt rundt skalering og feilhandsaming for brukarar/applikasjonsutviklarar
- Ønske om tilsvarande abstraksjonar for breiare klasser av applikasjonar
- Motivasjon: Eksisterande programmeringsmodellar som MapReduce basert på asyklisk datastraum *frå* stabilt lager *til* stabilt lager

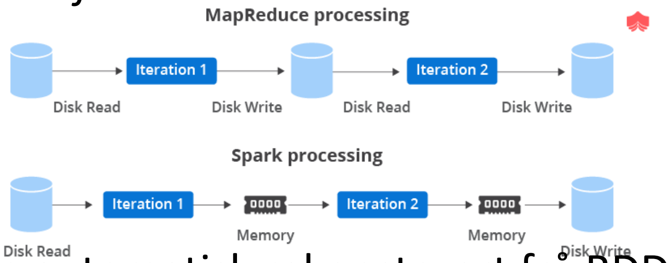


Mål

- Støtte applikasjonar der datasett vert gjenbrukt av (parallelle) operasjonar
  - Iterative jobbar (vanleg i maskinlæring)
  - Interaktiv dataanalyse og datagruvedrift
- Behalde (implisitt) feiltoleranse og skalerbarheit som i MapReduce
- Eksperimentere med programmerbarheit
  - Integrere i programmeringsspråket Scala
  - Støtte interaktiv bruk i Scala-interpreter

Programmingmodell: RDD

- Resilient distributed datasets (RDDs)
  - Uforanderlige (immutable), partitisjonerte samlingar med objekt, konstruert frå:
    - -Eksternlager (t.d. HDFS-filer eller database)
    - -“Parallelliserte” arrays
    - -Resultat av *transformasjon* av eksisterande RDDs med map, filter, groupBy,…
  - Kan verte buffra for effektiv gjenbruk



  - *Resilient*: Tapt partisjon automatisk rekonstruert frå RDDs den er basert på

- Handlingar (actions) på RDDs
  - Count, reduce, collect, save, …

    ➔ resultat til “drivar”-program (applikasjon) eller t.d. fil

- Lat (”lazy”) evaluering: Utføring startar først ved ”action”

- Programmering:
  - spark-shell (Scala interpreter)
  - pyspark (Python interpreter)
  - spark-submit (jar-fil, dvs. Scala eller Java-applikasjon, eller Python-applikasjon)
  - Spark-sql
  - sparkR

Scala
(Akkurat nok til å forstå eksempla ☺)

- Fokusert på funksjonell programmering (sjølv om ein i praksis kan skrive ”Java i Scala”)

- Funksjon som verdi:
  ```
  scala> val inc = (x : Int) => x + 1
  inc: Int => Int = <function1>
  scala> inc(1)
  res0: Int = 2
  ```

-
  ```
  scala> List(1, 2, 3).map((x: Int) => x + 1)
  res1: List[Int] = List(2, 3, 4)
  ```

-
  ```
  scala> List(1, 2, 3).map(_ + 1)
  res2: List[Int] = List(2, 3, 4)
  ```

-
  ```
  scala> List(1, 2, 3).map((x: Int) => inc(x))
  res3: List[Int] = List(2, 3, 4)
  ```

-
  ```
  scala> List(1, 2, 3)(2)
  res4: Int = 3
  ```

- På ”minieksempelnivå” ganske likt Python

Eksempel: Logg-gruvedrift

Laste feilmeldingar frå logg inn i hovudlager, interaktivt søk for forskjellige mønster

```
// Base RDD
lines = sc.textFile(“filename”)
// Transformert RDD
errors = lines.filter(_.startsWith(“ERROR”))
errors.persist()
// Actions
errors.count()
errors.filter(_.contains("MySQL")).count()
. . .
```

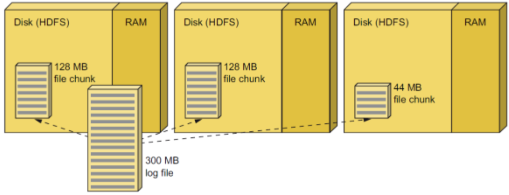


Figure 1.3 Storing a 300 MB log file in a three-node Hadoop cluster

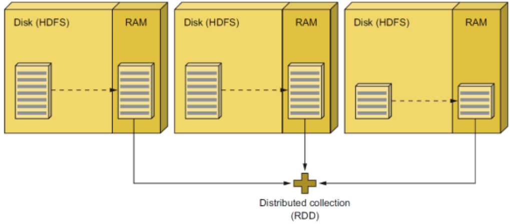


Figure 1.4 Loading a text file from HDFS

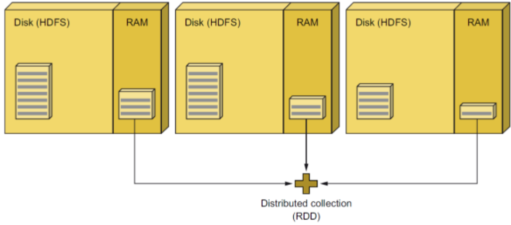
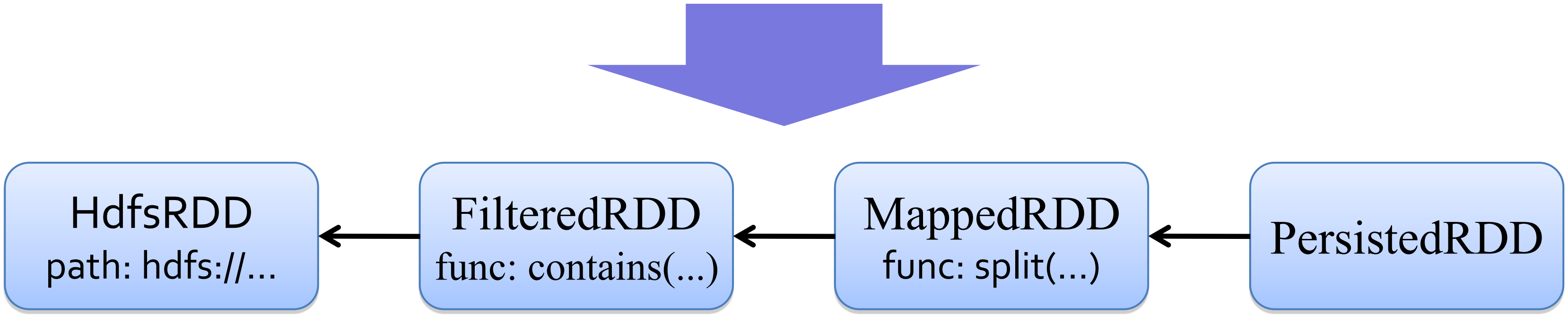


Figure 1.5 Filtering the collection to contain only lines containing the OutOfMemoryError string

RDD-feiltoleranse

- RDDs har avstammings-informasjon (*lineage)* som kan brukast til å rekonstruere tapte partisjonar
- Eksempel:

```
cachedMsgs = sc.textFile(...).filter(_.contains(“error”))
                             .map(_.split(‘\t’)(2))
                             .persist()
```



HdfsRDD
path: hdfs://...

FilteredRDD
func: contains(...)

MappedRDD
func: split(...)

PersistedRDD

Viktige transformasjonar: map og filter

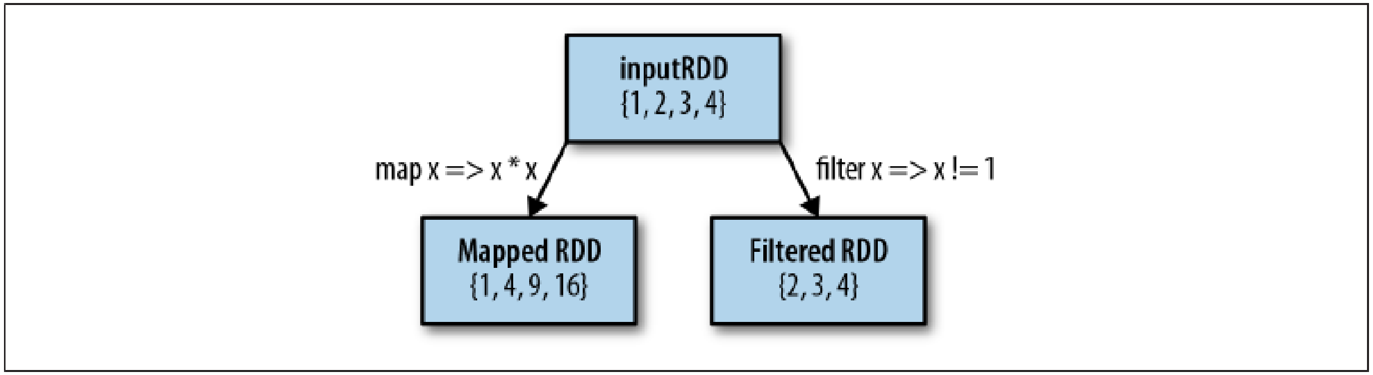
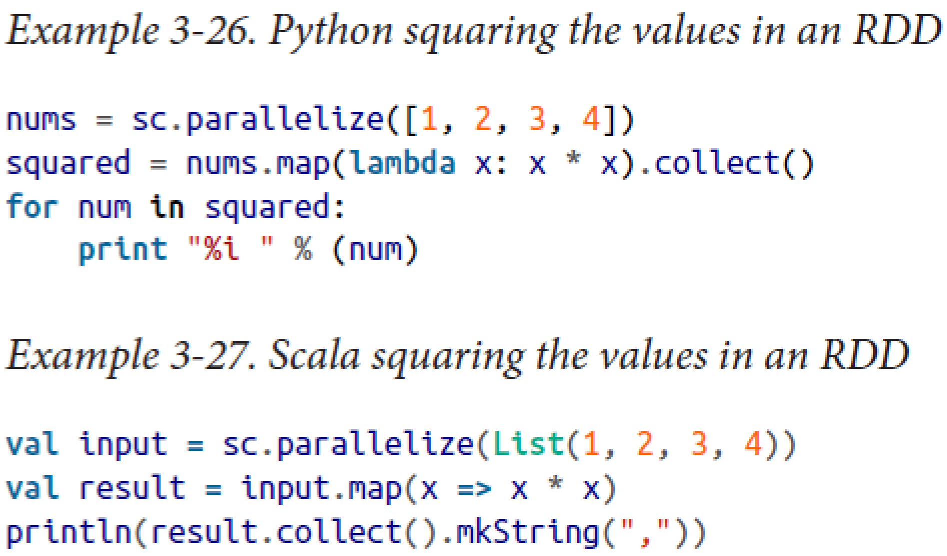


Figure 3-2. Mapped and filtered RDD from an input RDD



Example 3-26. Python squaring the values in an RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
    print "%i " % (num)
```

Example 3-27. Scala squaring the values in an RDD

```
val input = sc.parallelize(List(1, 2, 3, 4))
val result = input.map(x => x * x)
println(result.collect().mkString(","))
```

map vs. flatmap

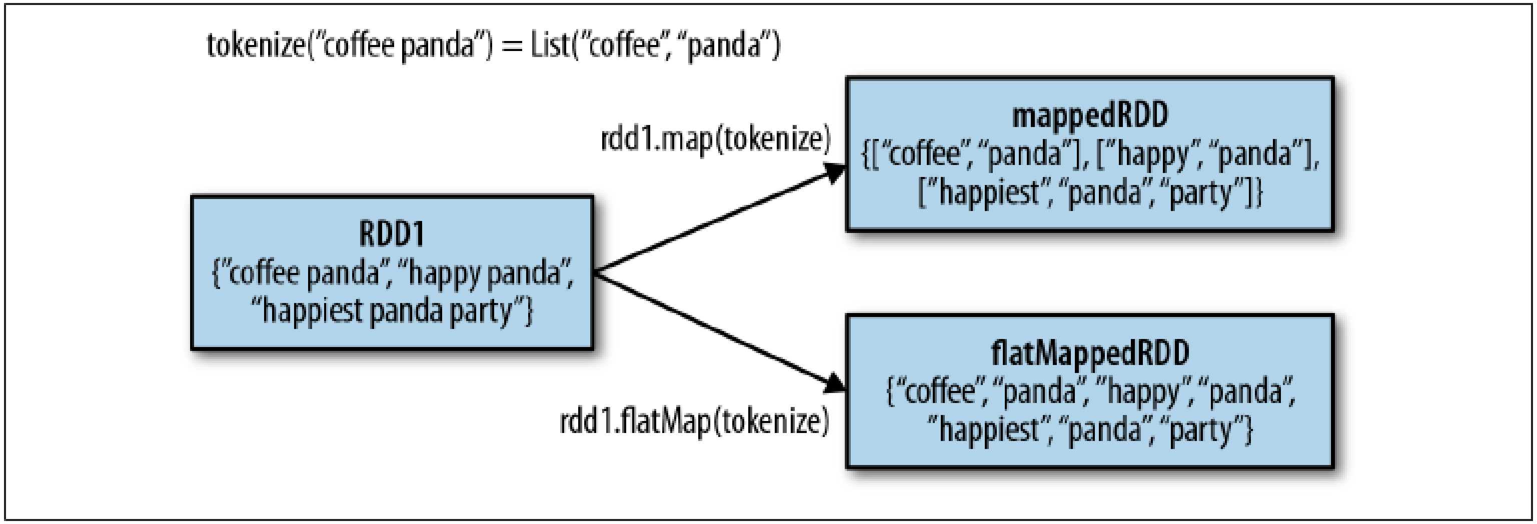
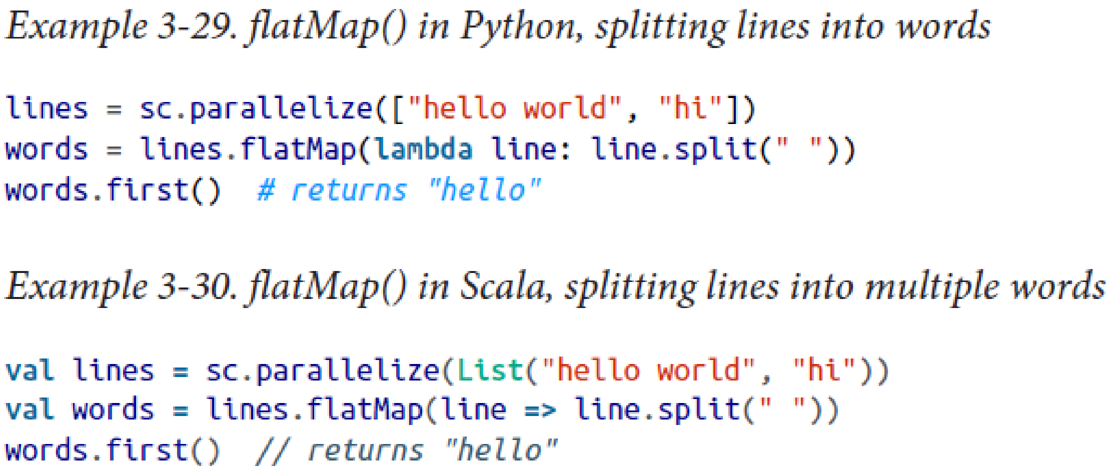

tokenize("coffee panda") = List("coffee", "panda")
RDD1
{"coffee panda", "happy panda", "happiest panda party"}
rdd1.map(tokenize)
mappedRDD
{["coffee", "panda"], ["happy", "panda"], ["happiest", "panda", "party"]}
rdd1.flatMap(tokenize)
flatMappedRDD
{"coffee", "panda", "happy", "panda", "happiest", "panda", "party"}

Figure 3-3. Difference between flatMap() and map() on an RDD



Example 3-29. flatMap() in Python, splitting lines into words

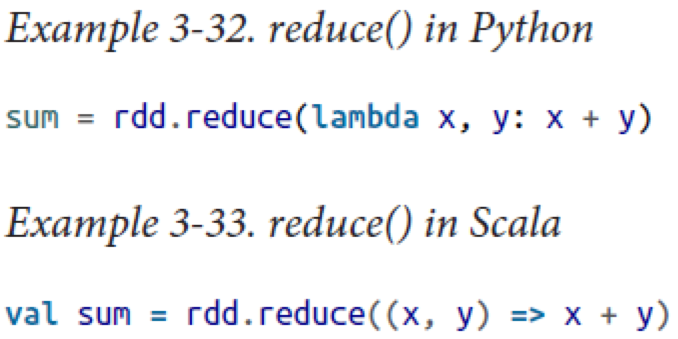
```python
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first()  # returns "hello"
```

Example 3-30. flatMap() in Scala, splitting lines into multiple words

```scala
val lines = sc.parallelize(List("hello world", "hi"))
val words = lines.flatMap(line => line.split(" "))
words.first()  // returns "hello"
```

Ofte brukt *action*: **reduce**(*func*)

- Aggregerer elementa i datasettet ved å bruke funksjon *funk* (som tek to argument og returnerer eitt)



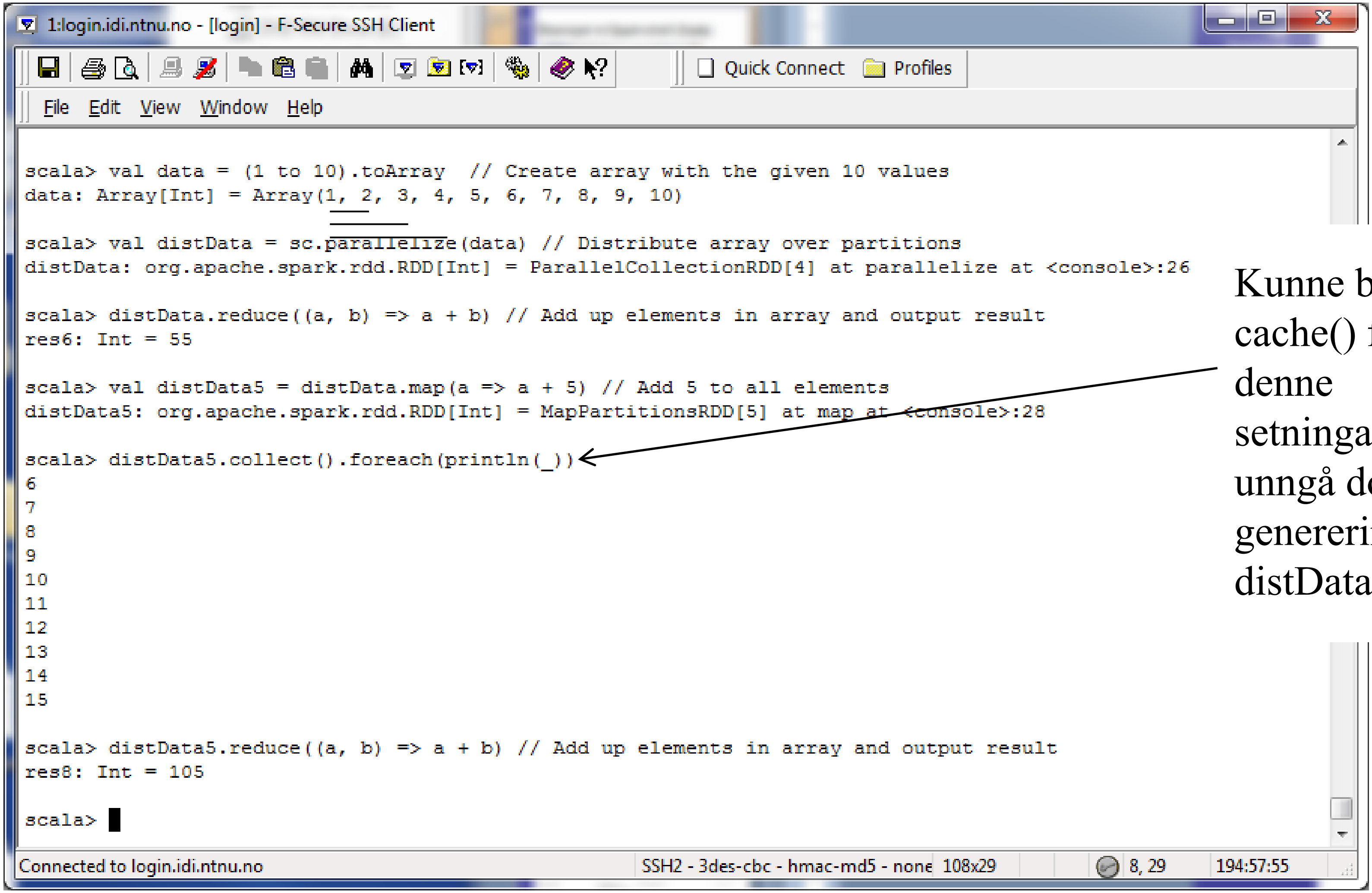
Example 3-32. reduce() in Python

```python
sum = rdd.reduce(lambda x, y: x + y)
```

Example 3-33. reduce() in Scala

```scala
val sum = rdd.reduce((x, y) => x + y)
```

Eksempel m/Spark-shell (Scala)


1:login.idi.ntnu.no - [login] - F-Secure SSH Client
Quick Connect
Profiles
File
Edit
View
Window
Help
Connected to login.idi.ntnu.no
SSH2 - 3des-cbc - hmac-md5 - none
108x29
8, 29
194:57:55

```
scala> val data = (1 to 10).toArray  // Create array with the given 10 values
data: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> val distData = sc.parallelize(data) // Distribute array over partitions
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:26

scala> distData.reduce((a, b) => a + b) // Add up elements in array and output result
res6: Int = 55

scala> val distData5 = distData.map(a => a + 5) // Add 5 to all elements
distData5: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at map at <console>:28

scala> distData5.collect().foreach(println(_))
6
7
8
9
10
11
12
13
14
15

scala> distData5.reduce((a, b) => a + b) // Add up elements in array and output result
res8: Int = 105

scala> 
```

Kunne brukt cache() før denne setninga for å unngå dobbel generering av distData5

RDD transformasjonar & Actions

Oversikt over desse på foilane er berre med for å vise at det er pensum, forventa at de lærer meir om desse (og tilsvarande for pair-RDDS) ved å gå gjennom ”quick tour” lagt ut på Blackboard, og også gjennom prosjektet

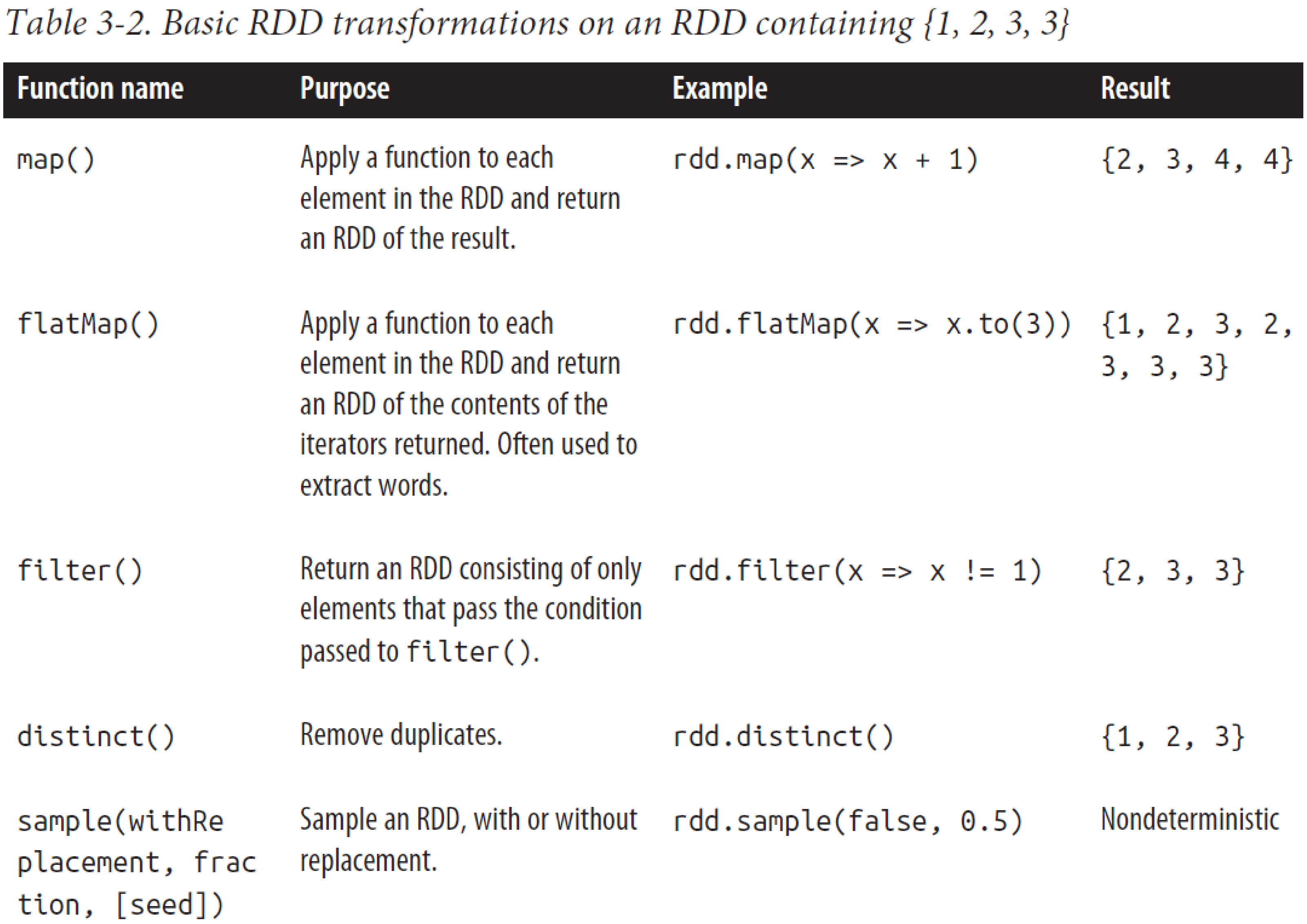


Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

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

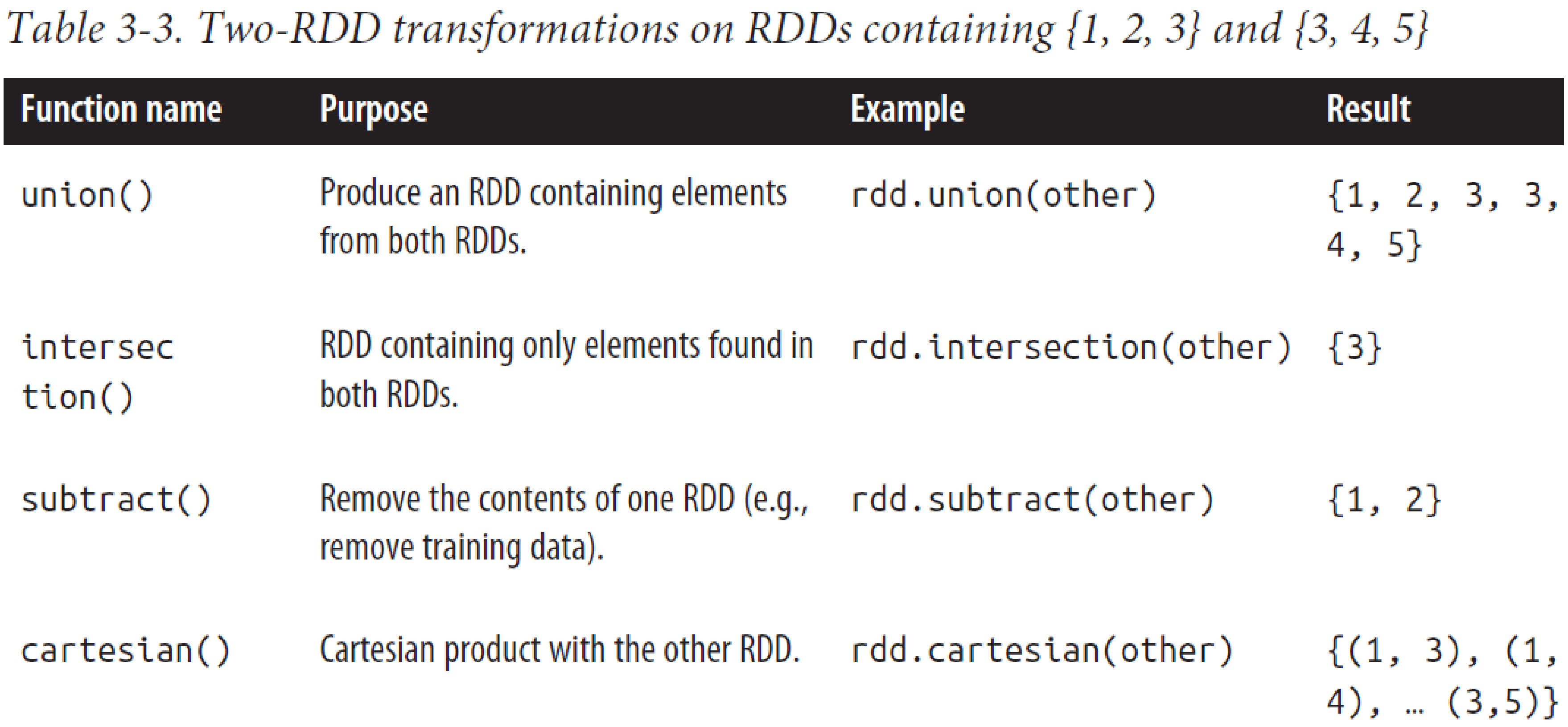


Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

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)}`

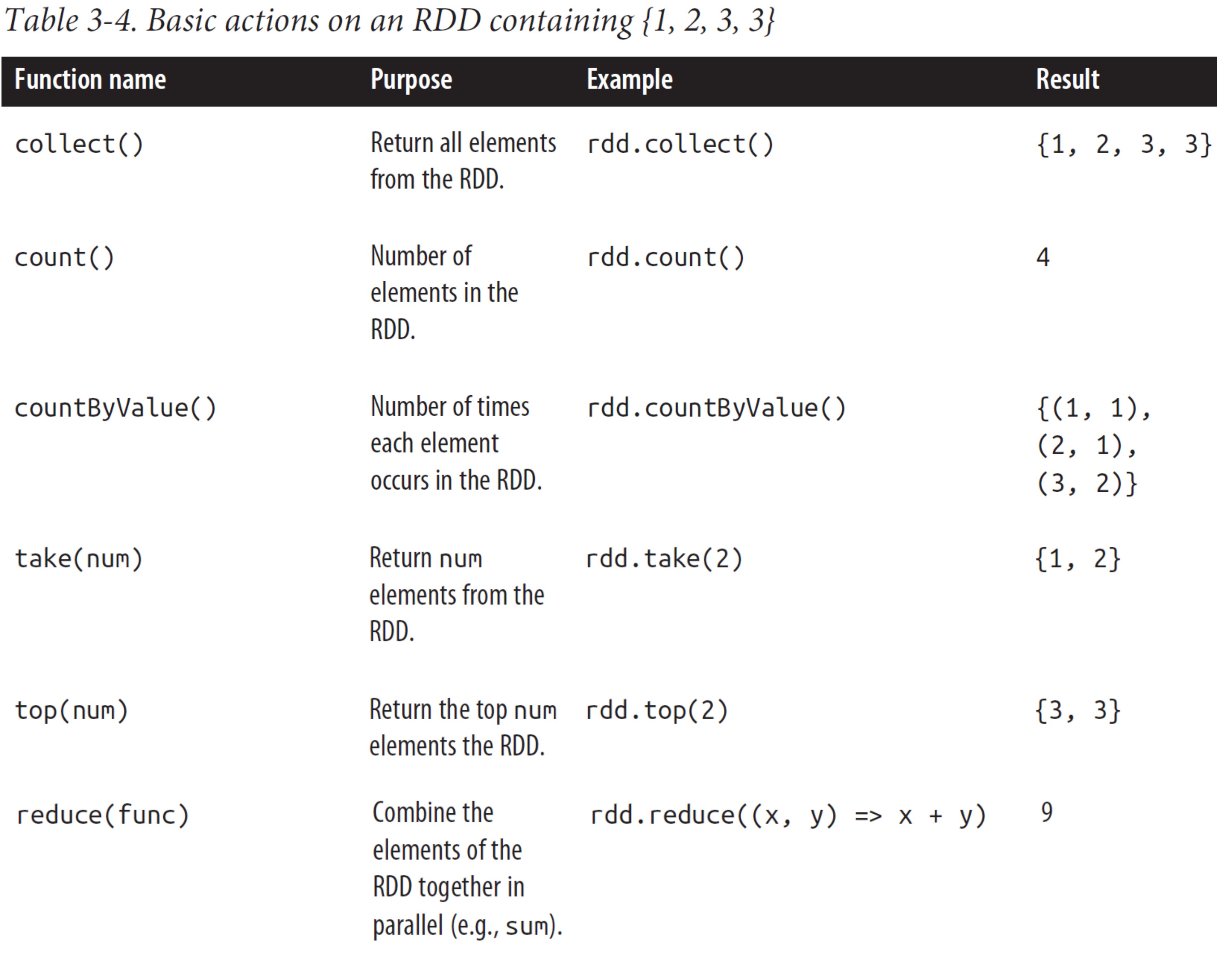


Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	4

`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1),`
			`(2, 1),`
			`(3, 2)}`

`take(num)`	Return `num` elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top `num` elements the RDD.	`rdd.top(2)`	`{3, 3}`
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9

Key/Value-par

- Lage par, eks:
  - Spesielle inputformat frå fil
    - T.d. komma- eller tab-separerte filer
  - Vha. map:

    ```
    // Lage par-RDD ved å bruke første ord som nøkkel
    // Dvs. linje => (først ord på linje, linje)
    val pairs = lines.map(x => (x.split(" ")(0), x))
    ```

Eksempel

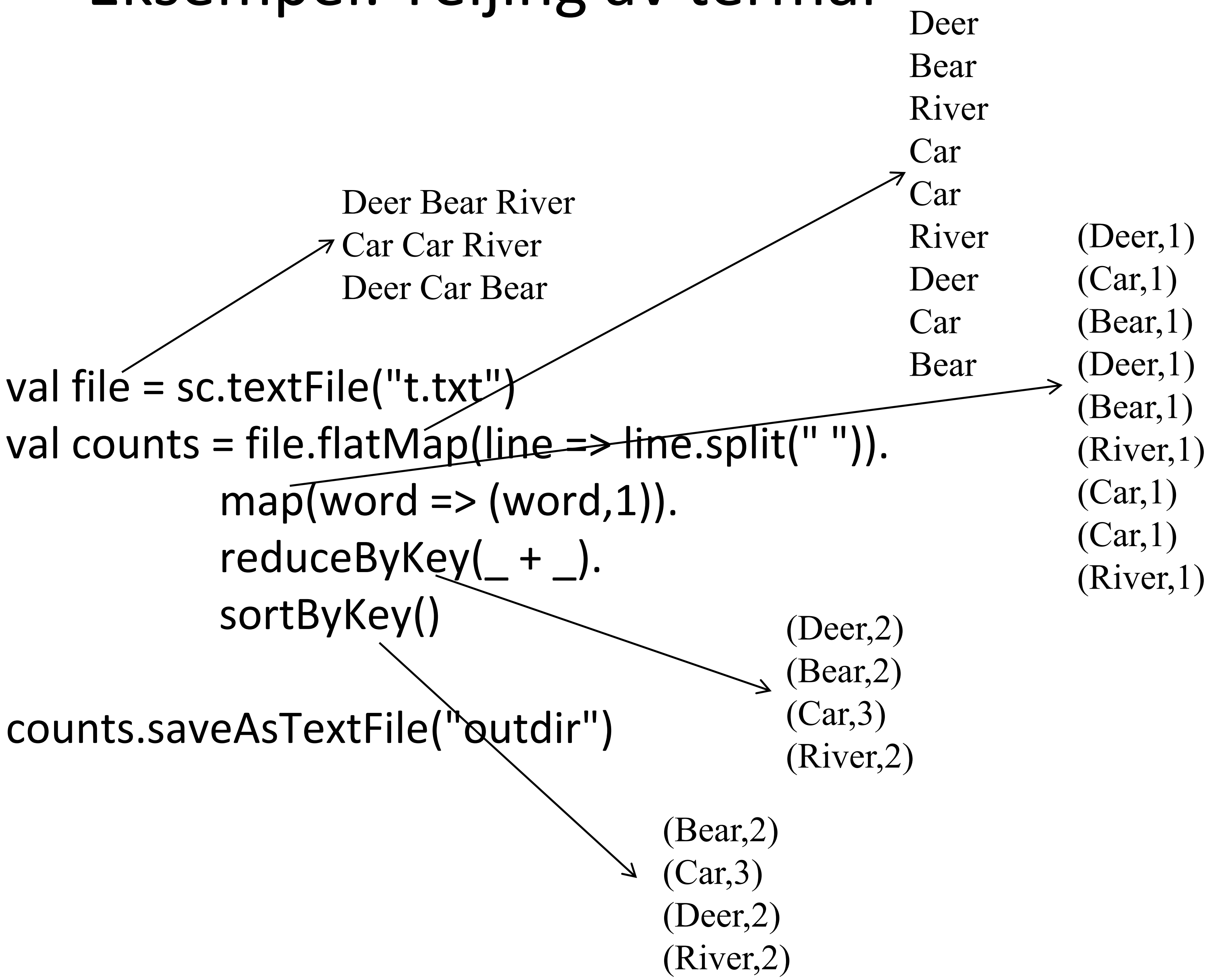
```
1950	0	1
1950	22	1
1950	-11	1
1949	111	1
1949	78	1
```

- Oppgåve: Finne max-temperatur
- Data i tab-separert fil (TSV)

```
val lines = sc.textFile("input/ncdc/micro-tab/sample.txt")
// Frå String til Array of Strings
val records = lines.map(_.split("\t"))
// Filtrer ut postar med feil
val filtered = records.filter(rec => (rec(1) != "9999" && rec(2).matches("[01459]")))
// Til Key/Value-par
val tuples = filtered.map(rec => (rec(0).toInt, rec(1).toInt)) //
// For kvar nøkkel (år), finn max
val maxTemps = tuples.reduceByKey((a, b) => Math.max(a, b))
maxTemps.collect().foreach(println(_))
```

```
(1950,22)
(1949,111)
```

Eksempel: Teljing av termar



```
val file = sc.textFile("t.txt")
val counts = file.flatMap(line => line.split(" ")).
               map(word => (word,1)).
               reduceByKey(_ + _).
               sortByKey()

counts.saveAsTextFile("outdir")
```

Deer Bear River
Car Car River
Deer Car Bear

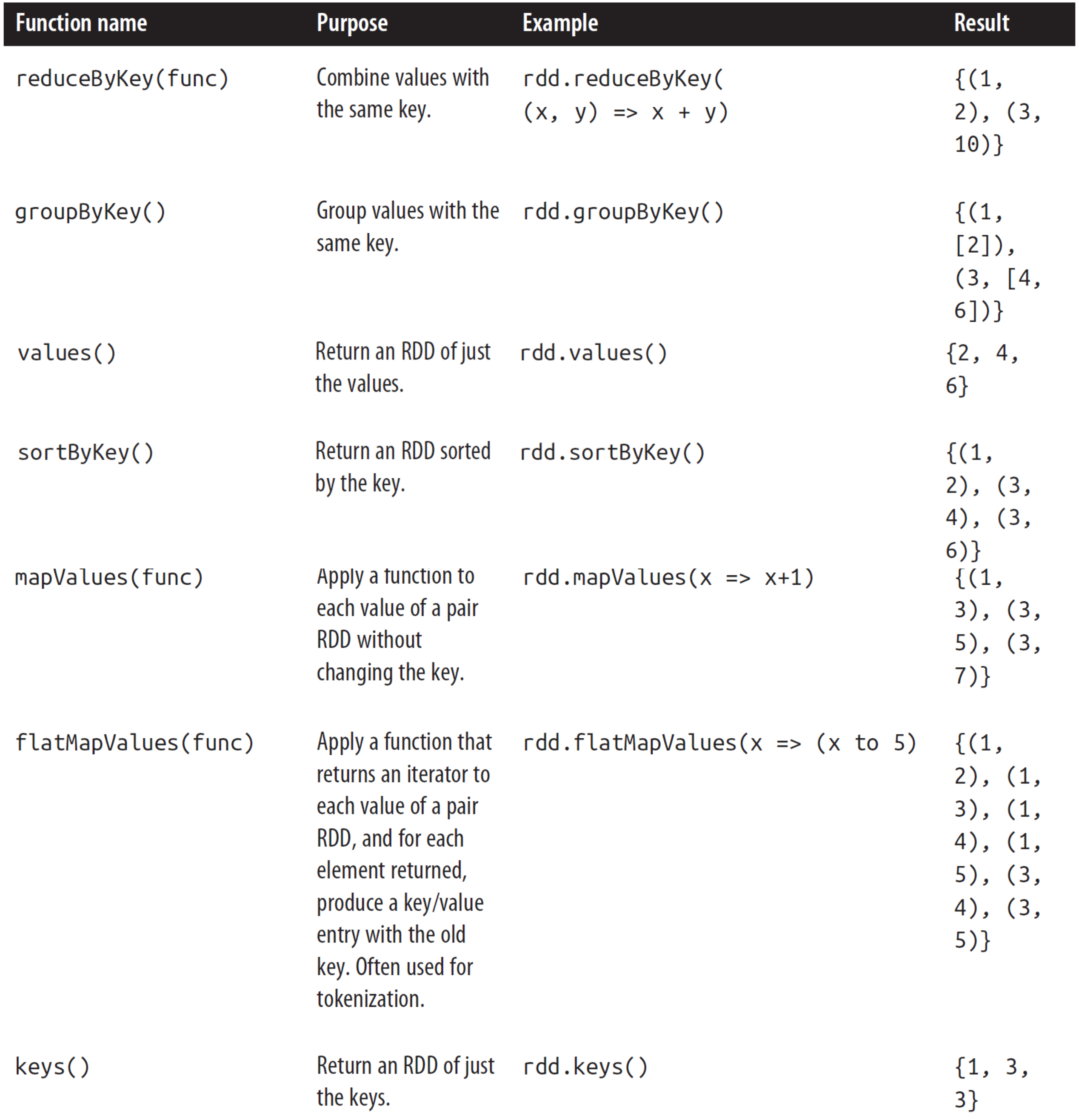
Deer
Bear
River
Car
Car
River
Deer
Car
Bear

(Deer,1)
(Car,1)
(Bear,1)
(Deer,1)
(Bear,1)
(River,1)
(Car,1)
(Car,1)
(River,1)

(Deer,2)
(Bear,2)
(Car,3)
(River,2)

(Bear,2)
(Car,3)
(Deer,2)
(River,2)

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})



Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey( (x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`
`sortByKey()`	Return an RDD sorted by the key.	`rdd.sortByKey()`	`{(1, 2), (3, 4), (3, 6)}`
`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`

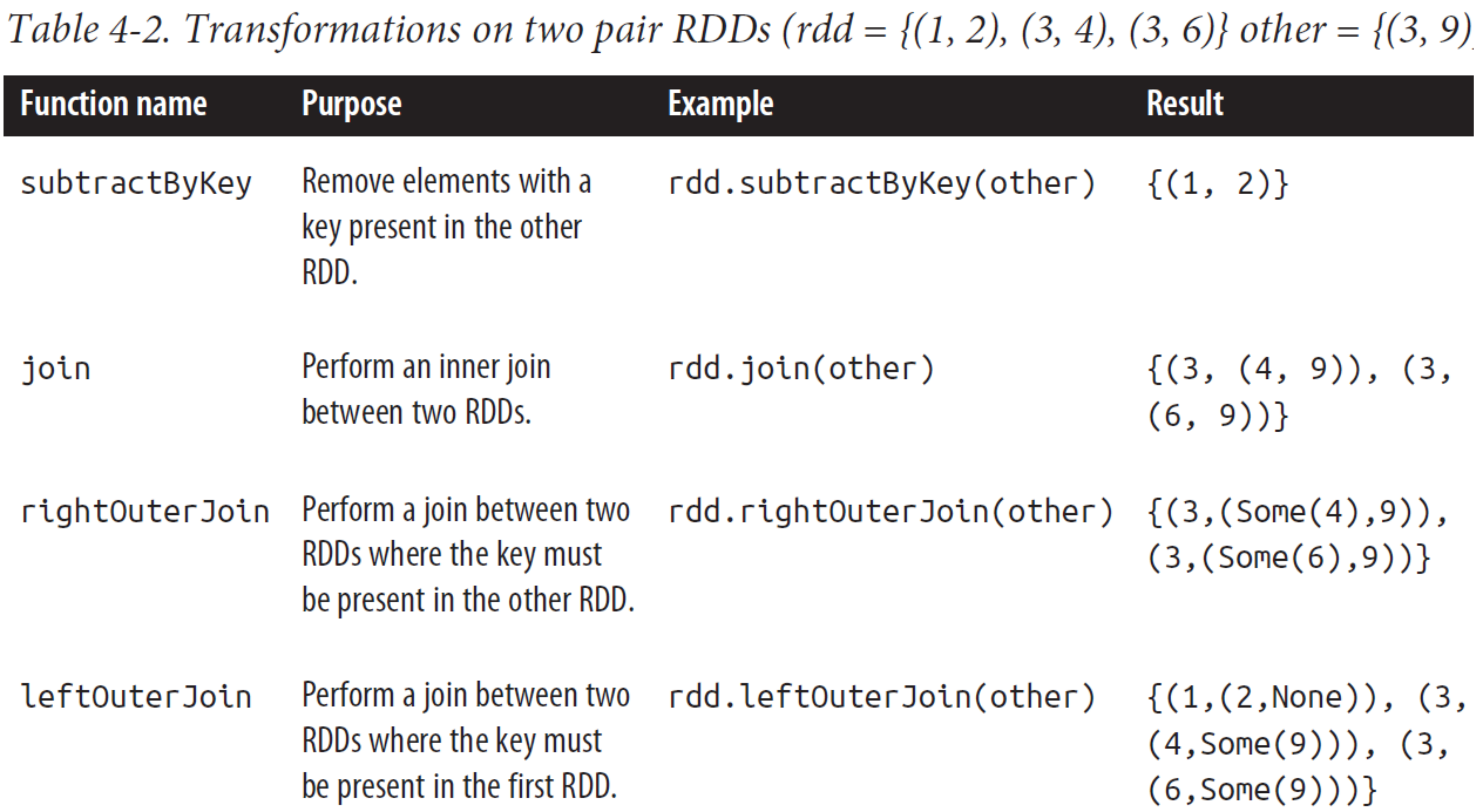


Table 4-2. Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)

Function name	Purpose	Example	Result
`subtractByKey`	Remove elements with a key present in the other RDD.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`

Par-RDD-”actions”

- I tillegg til dei nemnt tidlegare:

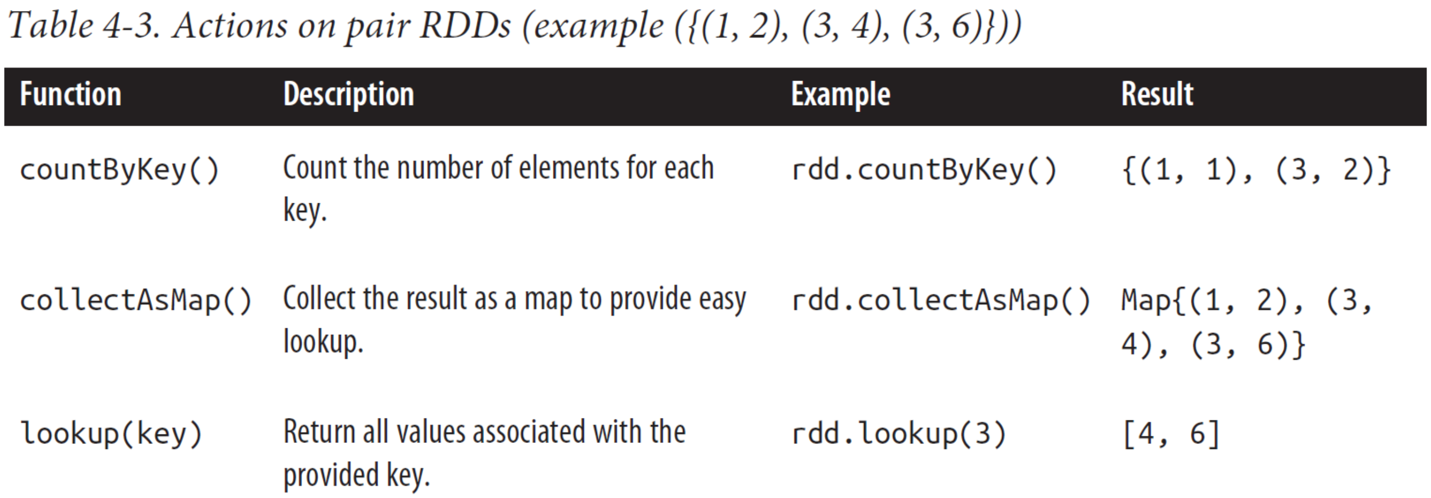


Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

Persistens-nivå: RDD.persist(level)

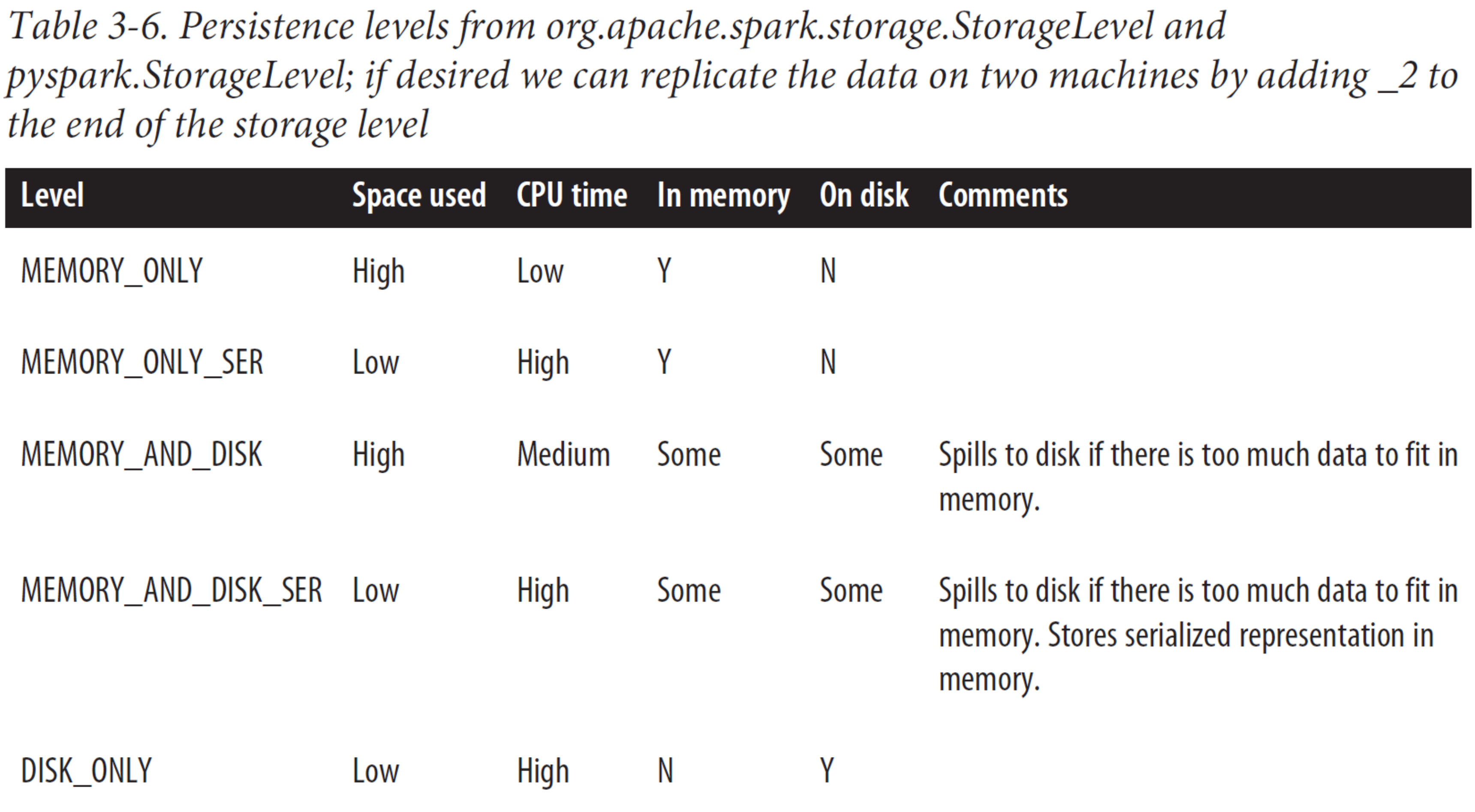


Table 3-6. Persistence levels from org.apache.spark.storage.StorageLevel and pyspark.StorageLevel; if desired we can replicate the data on two machines by adding _2 to the end of the storage level

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	High	Low	Y	N	
MEMORY_ONLY_SER	Low	High	Y	N	
MEMORY_AND_DISK	High	Medium	Some	Some	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.
DISK_ONLY	Low	High	N	Y	

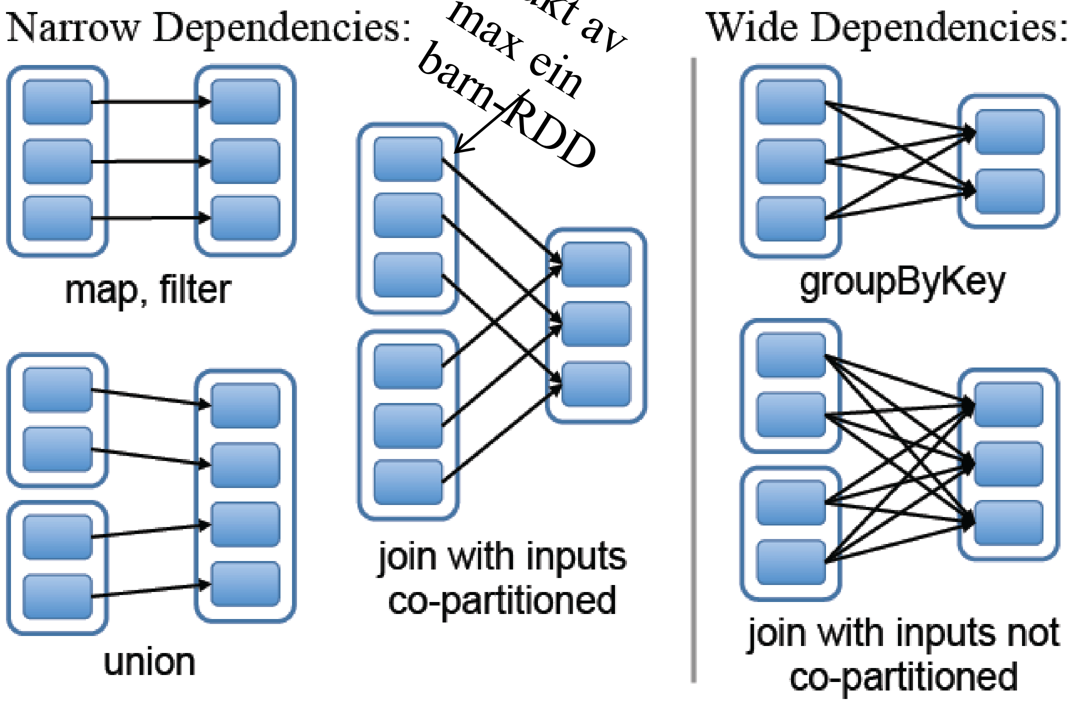
(SER: Serialiserte Java-objekt, dvs. eit byte-array i staden for mange objekt)

RDD.cache() ekvivalent med RDD.persist(MEMORY_ONLY)

Representasjon av RDDs

- Viktige metadata:
  - Partisjonar (atomiske delar av datasettet) og lokasjonar for desse
    - T.d. RDD som representerer HDFS-fil har ein partisjon for kvar blokk og veit lokasjon for desse
  - Partisjonering: Om (og korleis) RDD er hash/range-partisjonert
  - Avhengigheiter (foreldre-RDDs): Kva RDD(s) ein RDD er basert på
    - Narrow dependency: Kvar partisjon i forelder-RDD brukt av max ein partisjon i barn-RDD
    - Wide dependency (shuffle): Fleire barn-partisjonar kan vere avhengig av ein forelder-RDD

Narrow vs. wide dependency


Narrow Dependencies:
Wide Dependencies:
map, filter
union
join with inputs co-partitioned
groupByKey
join with inputs not co-partitioned

Brukt av max ein barn-RDD

Figure 4: Examples of narrow and wide dependencies. Each box is an RDD, with partitions shown as shaded rectangles.

- Skilnad mellom narrow og wide viktig:
  - Narrow: samlebands-utrekning på ei node, recovery enklare/billegare
  - Wide: shuffle nødvendig

Eksempel på utføring i Spark

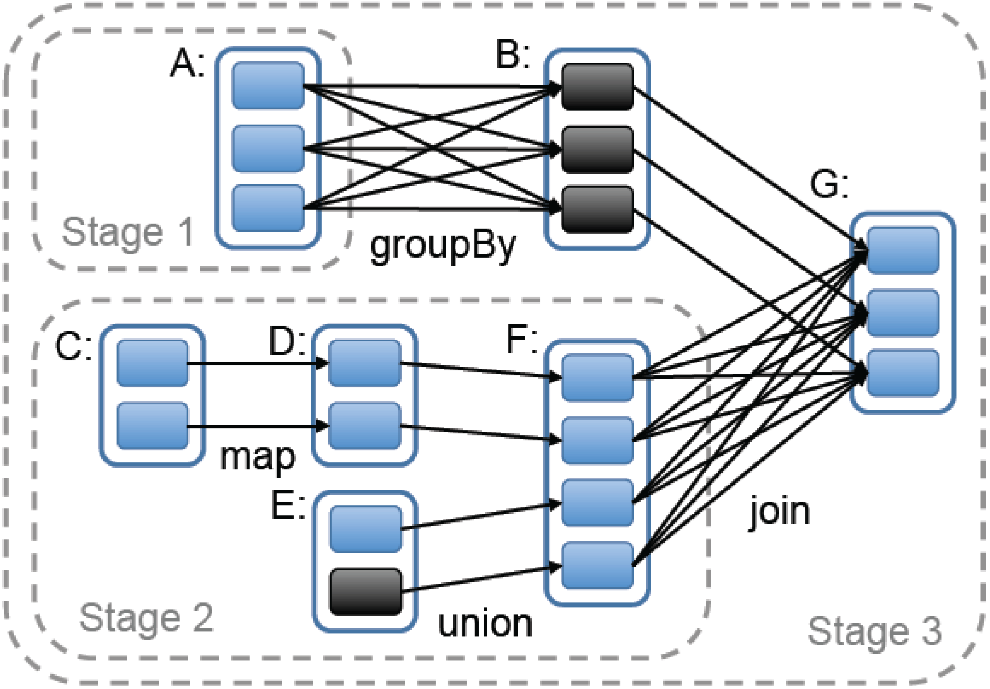


Figure 5: Example of how Spark computes job stages. Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory. To run an action on RDD G, we build build stages at wide dependencies and pipeline narrow transformations inside each stage. In this case, stage 1's output RDD is already in RAM, so we run stage 2 and then 3.

Application
Jobs
Stages
Tasks

Dataframes og Datasets

- DataFrames:
  - Distribuert samling med *rad-objekt*
  - Namngjevne kolonner
  - Tilsvarande *tabell* i relasjonsdatabasesystem
  - Kan spørjast med SQL
  - Logisk plan og optimalisering ved utføring
  - Rask/effektiv intern representasjon
- Datasets:
  - Hovudsakleg som DataFrames, men ”*type-sikker*”
- Eksempel:

// If you want a copy of file: https://folk.idi.ntnu.no/noervaag/TDT4305/

```
val tweetsDF = spark.read.json("tweets.json.gz")
tweetsDF.printSchema()
tweetsDF.createOrReplaceTempView("tweets")
val numPopularUsers =
    spark.sql("SELECT user.screen_name FROM tweets
    WHERE user.followers_count > 5000").count()
```

Eksempel på tweet i json-format

{"in_reply_to_status_id_str":null,"in_reply_to_status_id":null,"created_at":"Sat Dec 26 14:53:24 +0000 2015","in_reply_to_user_id_str":null,"source":"<a href=\"http://twitter.com\" rel=\"nofollow\">Twitter Web Client<\/a>","retweet_count":0,"retweeted":false,"geo":null,"filter_level":"low","in_reply_to_screen_name":null,"is_quote_status":false,"id_str":"680763346614890496","in_reply_to_user_id":null,"favorite_count":0,"id":680763346614890496,"text":"Our poster \"Top-k Dominating Queries, in Parallel, in Memory\" w/Orestis Gkorgkas and Sean Chester accepted at #EDBT2016!","place":{"country_code":"NO","country":"Norge","full_name":"Norway","bounding_box":{"coordinates":[[[-9.083133,57.956207],[-9.083133,80.824562],[33.63209,80.824562],[33.63209,57.956207]]],"type":"Polygon"},"place_type":"country","name":"Norway","attributes":{},"id":"0ce8b9a7b2742f7e","url":"https://api.twitter.com/1.1/geo/id/0ce8b9a7b2742f7e.json"},"lang":"en","favorited":false,"coordinates":null,"truncated":false,"timestamp_ms":"1451141604461","entities":{"urls":[],"hashtags":[{"indices":[110,119],"text":"EDBT2016"}],"user_mentions":[],"symbols":[]},"contributors":null,"user":{"utc_offset":3600,"friends_count":243,"profile_image_url_https":"https://pbs.twimg.com/profile_images/470768125/noervaag_2000px_normal.jpg","listed_count":9,"profile_background_image_url":"http://abs.twimg.com/images/themes/theme1/bg.png","default_profile_image":false,"favourites_count":84,"description":"Professor at NTNU, Trondheim, Norway.","created_at":"Wed Oct 14 11:30:14 +0000 2009","is_translator":false,"profile_background_image_url_https":"https://abs.twimg.com/images/themes/theme1/bg.png","protected":false,"screen_name":"noervaag","id_str":"82337208","profile_link_color":"0084B4","id":82337208,"geo_enabled":true,"profile_background_color":"C0DEED","lang":"en","profile_sidebar_border_color":"C0DEED","profile_text_color":"333333","verified":false,"profile_image_url":"http://pbs.twimg.com/profile_images/470768125/noervaag_2000px_normal.jpg","time_zone":"Copenhagen","url":"http://www.idi.ntnu.no/~noervaag/","contributors_enabled":false,"profile_background_tile":false,"statuses_count":295,"follow_request_sent":null,"followers_count":240,"profile_use_background_image":true,"default_profile":true,"following":null,"name":"Kjetil NÃ¸rvÃ¥g","location":"Trondheim, Norge","profile_sidebar_fill_color":"DDEEF6","notifications":null}}

```
scala> val tweetsDF = spark.read.json("tweets.json.gz")
20/01/27 15:01:06 WARN util.Utils: Truncated the string representation of a plan since
it was too large. This behavior can be adjusted by setting
'spark.debug.maxToStringFields' in SparkEnv.conf.
tweetsDF: org.apache.spark.sql.DataFrame = [_corrupt_record: string, contributors:
string ... 31 more fields]
scala> tweetsDF.printSchema()
root
 |-- coordinates: struct (nullable = true)
 |-- created_at: string (nullable = true)
 |-- id_str: string (nullable = true)
 |-- in_reply_to_screen_name: string (nullable = true)
....
 |-- retweet_count: long (nullable = true)
 |-- text: string (nullable = true)
 |-- timestamp_ms: string (nullable = true)
 |-- user: struct (nullable = true)
 |    |-- followers_count: long (nullable = true)
 |    |-- following: string (nullable = true)
...and many many more
scala> tweetsDF.createOrReplaceTempView("tweets")
scala> val numPopularUsers =     spark.sql("SELECT user.screen_name FROM tweets
WHERE user.followers_count > 5000").count()
numPopularUsers: Long = 654
```

Prøv Spark sjølve! ☺

På Blackboard:

1) Quickstart to installing tools used in TDT4305
2) Quick Tour of Spark
3) Eksempel-datafiler

For dei som vil vite meir...

- http://spark.apache.org/docs/latest/
- http://spark.apache.org/docs/latest/rdd-programming-guide.html