Data Processing in Scale

INST447 Data Source and Manipulation

Wei Ai
aiwei@umd.edu

Think about the following questions:

- Midjourney: What is the most frequent word in prompt in the last hour?

UserID	Timestamp	Prompt
A001		draw a cat licking paws on a sofa
A002		a dog resting in front of a sofa
A001		...
A003		...

- ChatGPT (OpenAI): How many turns on average does the conversations have, do they differ by users' age?

ChatI	UserID	# turns
B001	C001	3
B002		6
B003		1
B004		10

UserID	Age (inferred)
C001	25
C002	50

How would you do this using Pandas?

Word Count with Pandas

UserID	Timestamp	Prompt
A001		draw a cat licking paws on sofa
A002		a dog resting in front of a sofa
A001		…
A003		…

- Split sentences to words

```
words = df[“Prompt"].str.lower().str.split()
```

Prompt
draw
a
cat
licking
…

- Aggregate and count

```
word_count = words.groupby(lambda x:x).count()
```

- Chaining together

```
df.Prompt.str.split().explode().reset_index().groupby("Prompt").count()\
  .sort_values("index", ascending=False).head(10)
```

a	10
cat	8
draw	2
…	

What if the DataFrame Exceeds Memory?

- Option 1: Ditch Pandas, streaming Python

```
ctr = Counter()
for line in sys.stdin:    # or open("prompts.txt")
    ctr.update(line.lower().split())

print(ctr.most_common(20))
```

- Option 2: Ditch Python, use shell commands and external storage
  - This works surprisingly well, actually.

```
cat prompts.txt | tr -s ' ' '\n' | sort | uniq -c | sort -nr | head
```

- Limited by single thread on single machine

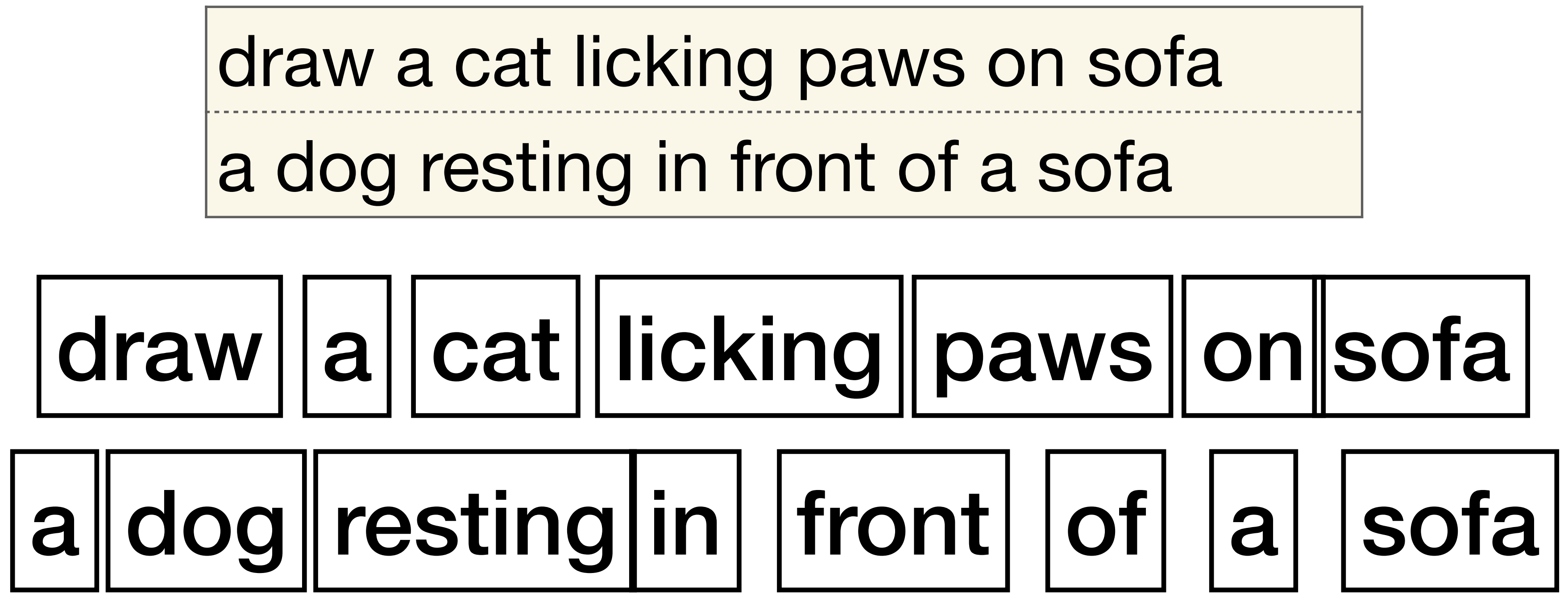
Word Count with Pandas

- Split sentences to words

```
words = df[“Prompt"].str.lower().str.split()
```

- Aggregate and count

```
word_count = words.groupby(lambda x:x).count()
```



Word Count with Pandas

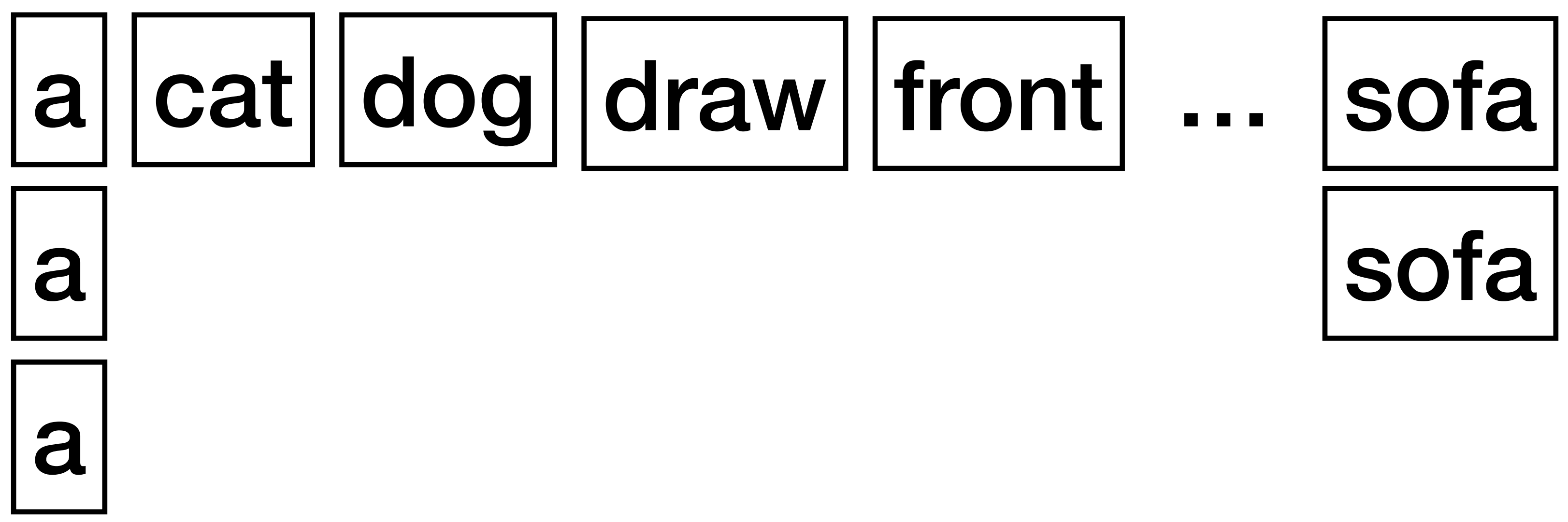
draw a cat licking paws on sofa
a dog resting in front of a sofa

- Split sentences to words

```
words = df["Prompt"].str.lower().str.split()
```

- Aggregate and count

```
word_count = words.groupby(lambda x:x).count()
```

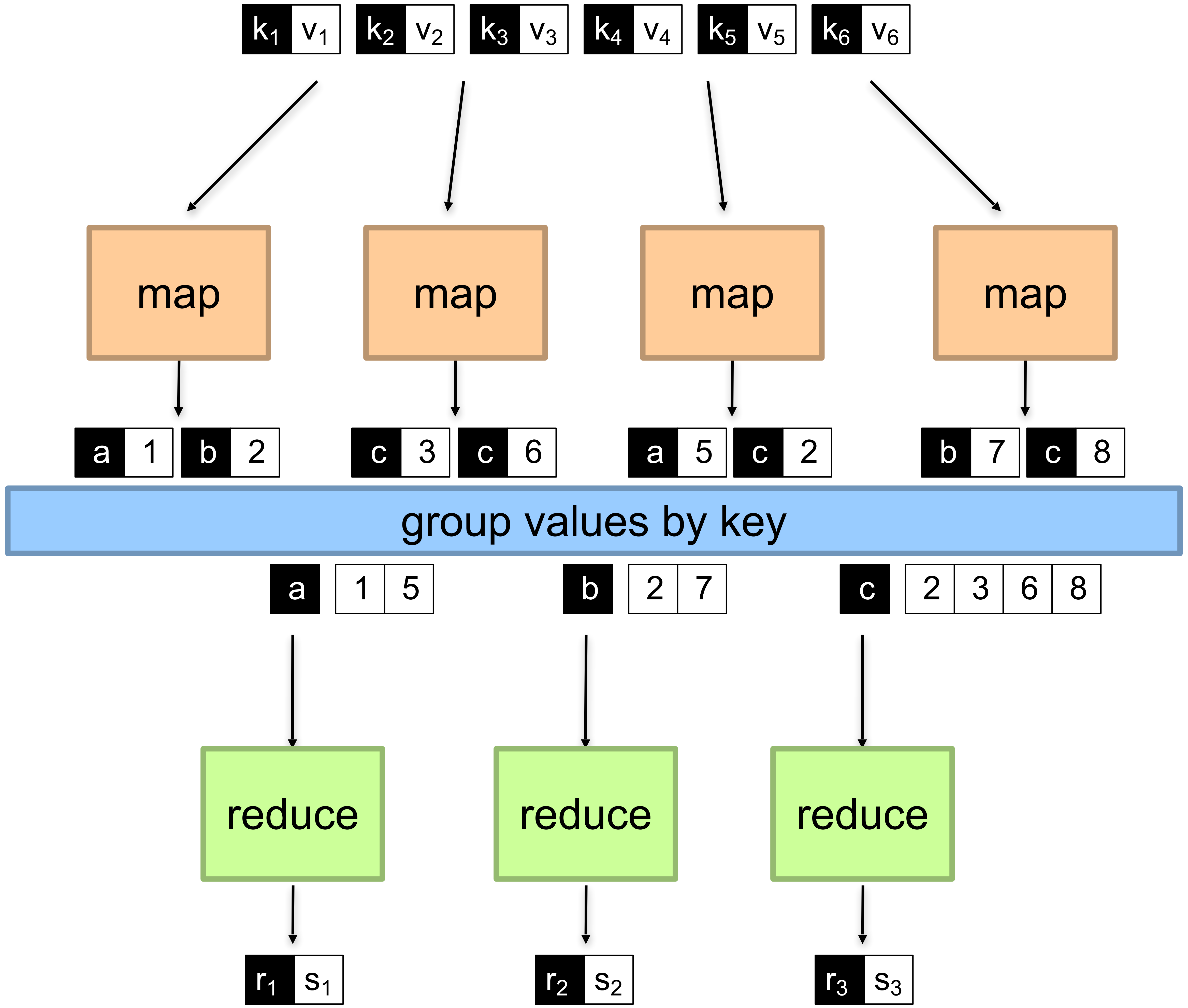


a	10
cat	1
sofa	2
…	

MapReduce - A Data Parallel Abstraction

- Process a large number of records
- “Do something” to each **Map**
- Group intermediate results
- “Aggregate” intermediate results **Reduce**
- Write final results

MapReduce



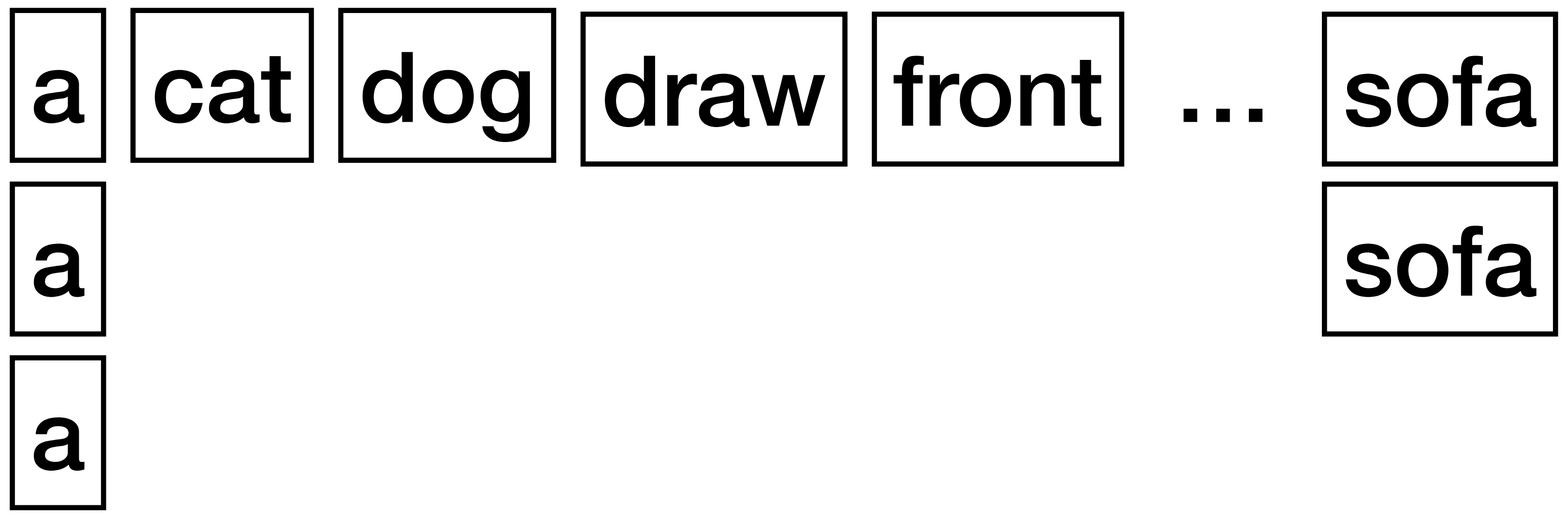
- Programmer specifies two functions:
  - map $(k1, v1) \rightarrow$ List[(k2, v2)]
  - reduce (k2, List[v2]) $\rightarrow$ List[(k3, v3)]
- All values with the same key are sent to the same reducer
- The execution framework handles everything else…

Word Count with MapReduce

- **Map**: for each record → emit (word, 1)
- **Combiner** (local reduce): sum counts per word within a mapper (cuts shuffle size)
- **Shuffle**: route same words to same reducer
- **Reduce**: sum all counts; output (word, total_count)

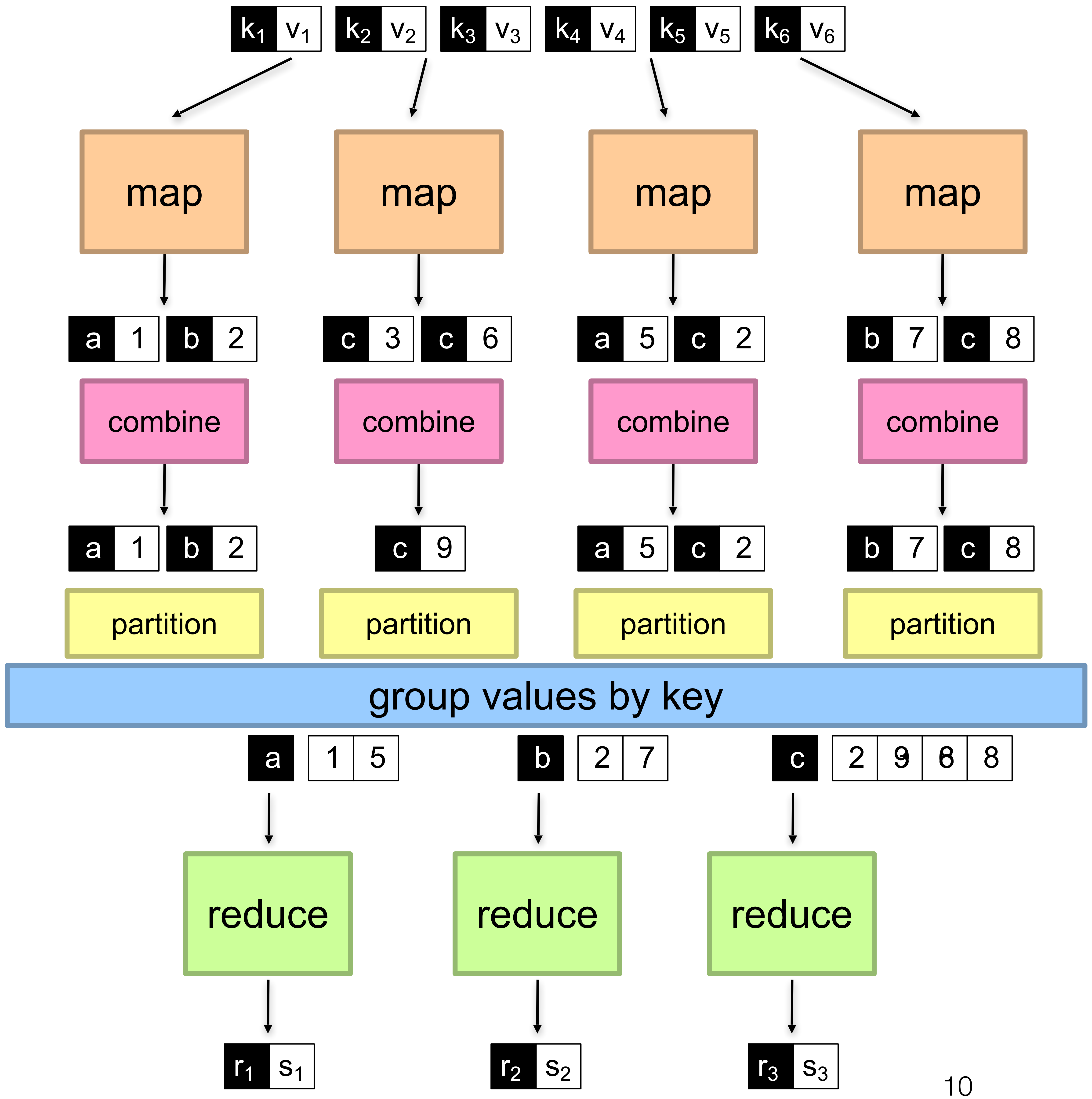
```
map(record):
    for word in tokenize(df.prompt):
        emit(word, 1)

combine(word, counts):
    emit(word, sum(counts))

reduce(word, counts):
    emit(word, sum(counts))
```

draw a cat licking paws on sofa
a dog resting in front of a sofa



a	10
cat	1
sofa	2
…	

MapReduce — Beyond Map and Reduce



MapReduce — Caveats with Combiner

- ChatGPT (OpenAI): How many turns on average does the conversations have, do they differ by users' age?

```
map(record):
    emit(uid, n_turns)

reduce(uid, [n_turns]):
    S = sum([n_turns])
    C = count([n_turns])
    avg = S/C
    emit(uid, avg)
```

```
map(record):
    emit(uid, n_turns)

combiner(uid, [n_turns]):
    S = sum([n_turns])
    C = count([n_turns])
    avg = S/C
    emit(uid, avg)

reduce(uid, [avgs]):
    S = sum([avgs])
    C = count([avgs])
    avg = S/C
    emit(uid, avg)
```

Will this work?

```
map(record):
    emit(uid, (n_turns, 1))

combiner/reduce(mid, partials):
    S = sum([S for S in partials])
    C = sum([C for C in partials])
    emit(uid, (S, C))

#final (after reduce):
    forall:
        emit(uid, S/C)
```

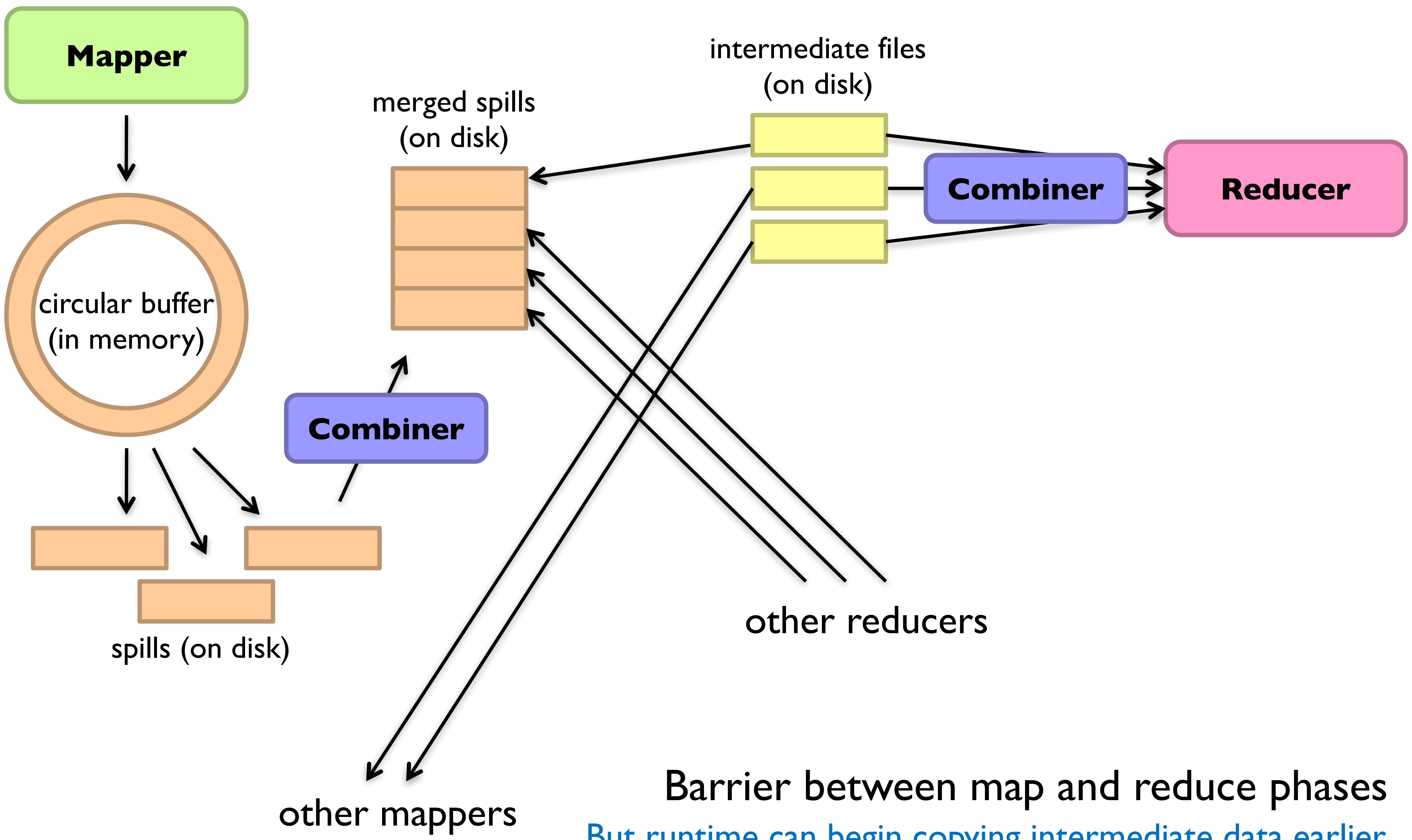
Will this work?

The Execution Framework (“Runtime”) Handles “Everything Else”…

- Scheduling
  - Assigns workers to map and reduce tasks
- “Data distribution”
  - Moves processes to data
- Synchronization
  - Groups intermediate data
- Errors and faults
  - Detects worker failures and restarts

Everything happens on top of a distributed FS (later)

Distributed Group By in MapReduce



Barrier between map and reduce phases

But runtime can begin copying intermediate data earlier

Don't move data to workers… move workers to the data!

Key idea: co-locate storage and compute

Start up worker on nodes that hold the data

We need a distributed file system for managing this

GFS (Google File System) for Google's MapReduce

HDFS (Hadoop Distributed File System) for Hadoop

GFS: Assumptions

Commodity hardware over "exotic" hardware

Scale "out", not "up"

High component failure rates

Inexpensive commodity components fail all the time

"Modest" number of huge files

Multi-gigabyte files are common, if not encouraged

Files are write-once, mostly appended to

Logs are a common case

Large streaming reads over random access

Design for high sustained throughput over low latency

GFS: Design Decisions

Files stored as chunks

Fixed size (64MB)

Reliability through replication

Each chunk replicated across 3+ chunkservers

Single master to coordinate access and hold metadata

Simple centralized management

No data caching

Little benefit for streaming reads over large datasets

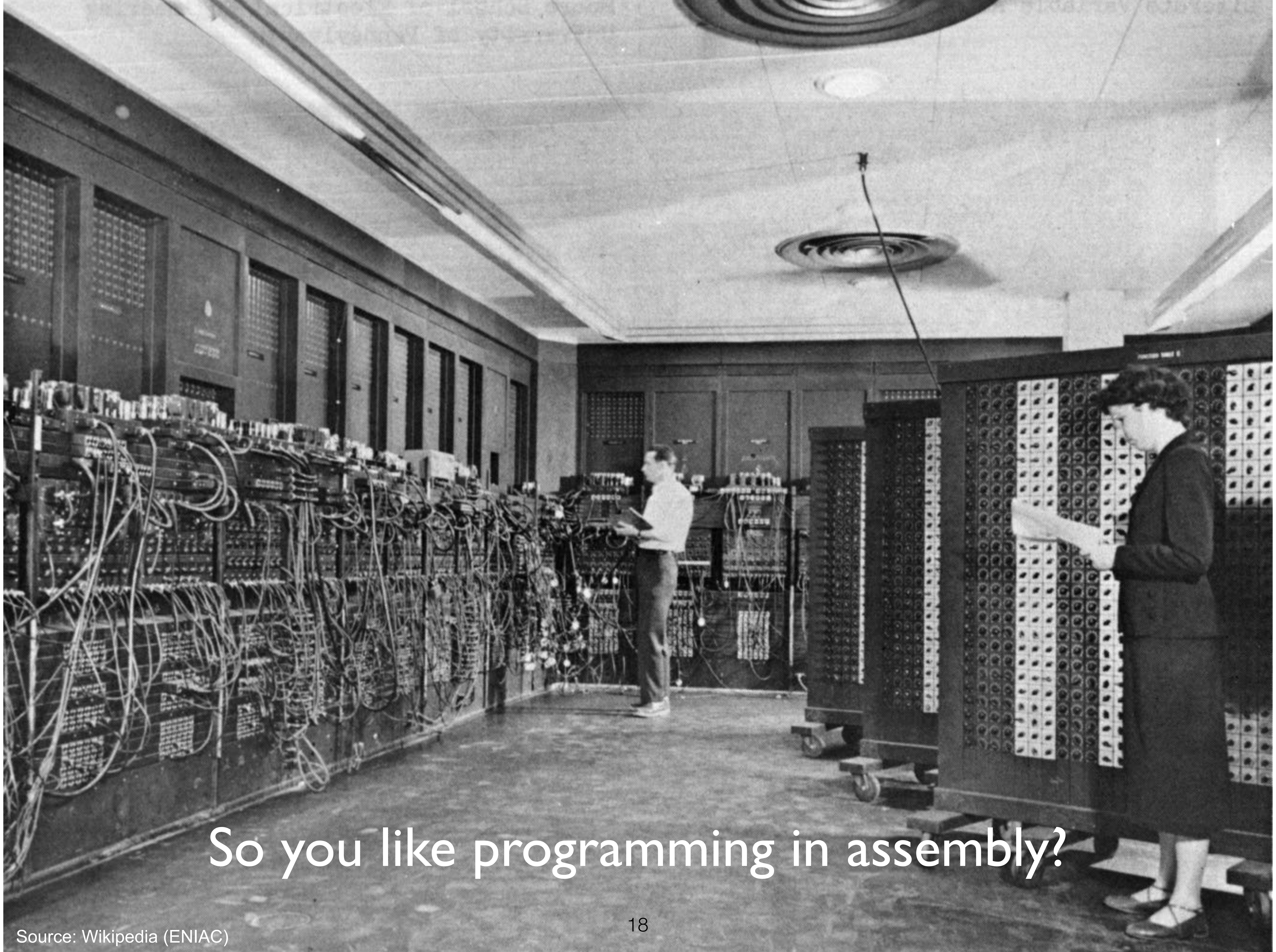
Simplify the API: not POSIX!

Push many issues onto the client (e.g., data layout)

HDFS = GFS clone (same basic ideas)

The datacenter *is* the computer!
What's the instruction set?

Source: Google



So you like programming in assembly?

Source: Wikipedia (ENIAC)

Hadoop is great, but it's really waaaaay too low level!

(circa 2007)

Source: Wikipedia (DeLorean time machine)

Data-Parallel Dataflow Languages

We have a collection of records,
want to apply a bunch of operations
to compute some result

What are the dataflow operators?

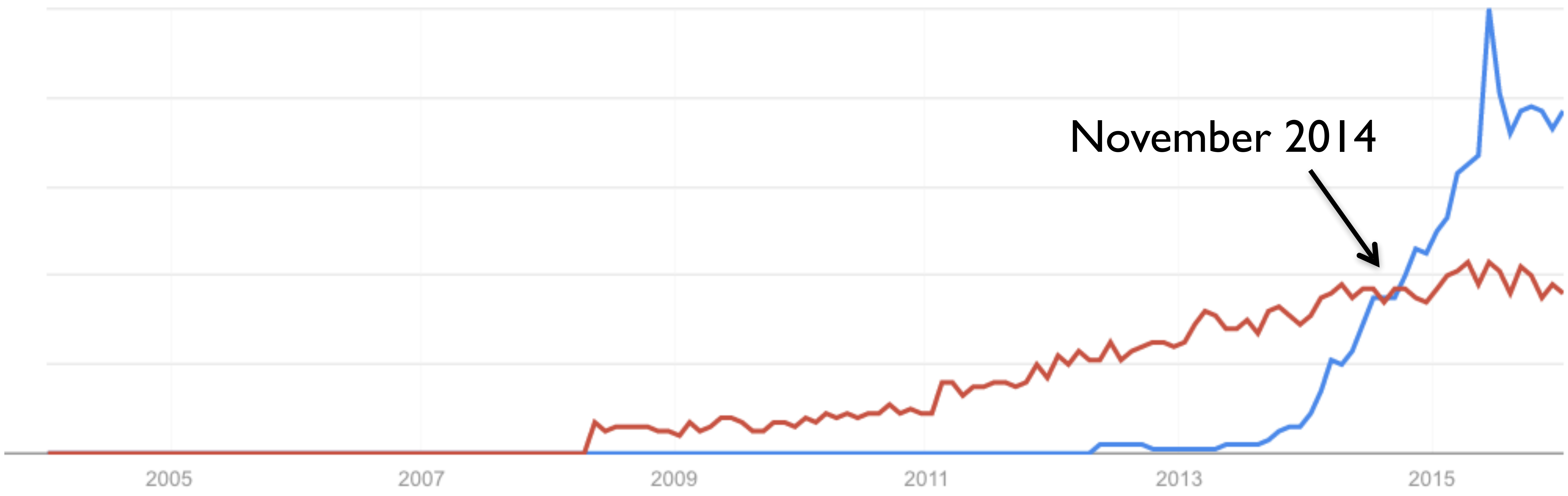
Spark

Answer to “What’s beyond MapReduce?”

Brief history:

Developed at UC Berkeley AMPLab in 2009
Open-sourced in 2010
Became top-level Apache project in February 2014
Commercial support provided by DataBricks

Spark vs. Hadoop



Google Trends

Source: Datanami (2014): http://www.datanami.com/2014/11/21/spark-just-passed-hadoop-popularity-web-heres/

MapReduce

List[(K1,V1)]

map

f: (K1, V1)

⇒ List[(K2, V2)]

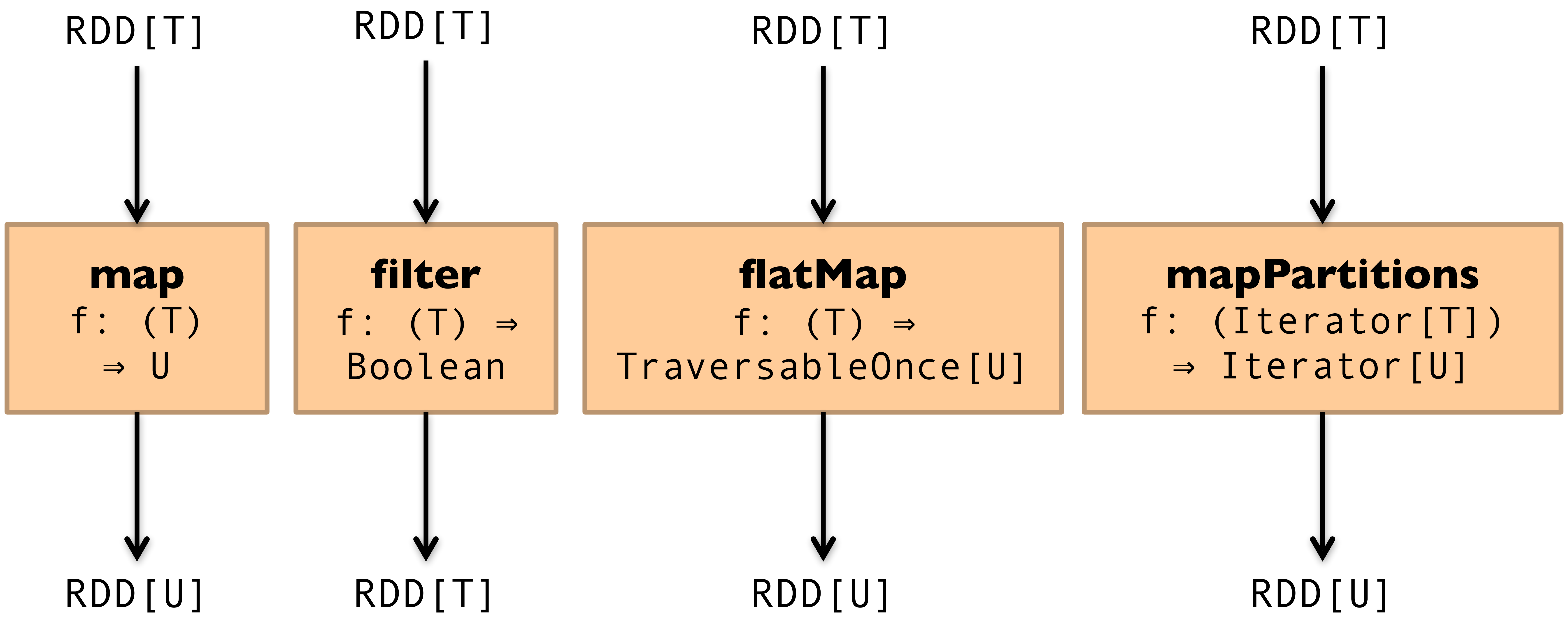
reduce

g: (K2, Iterable[V2])

⇒ List[(K3, V3)]

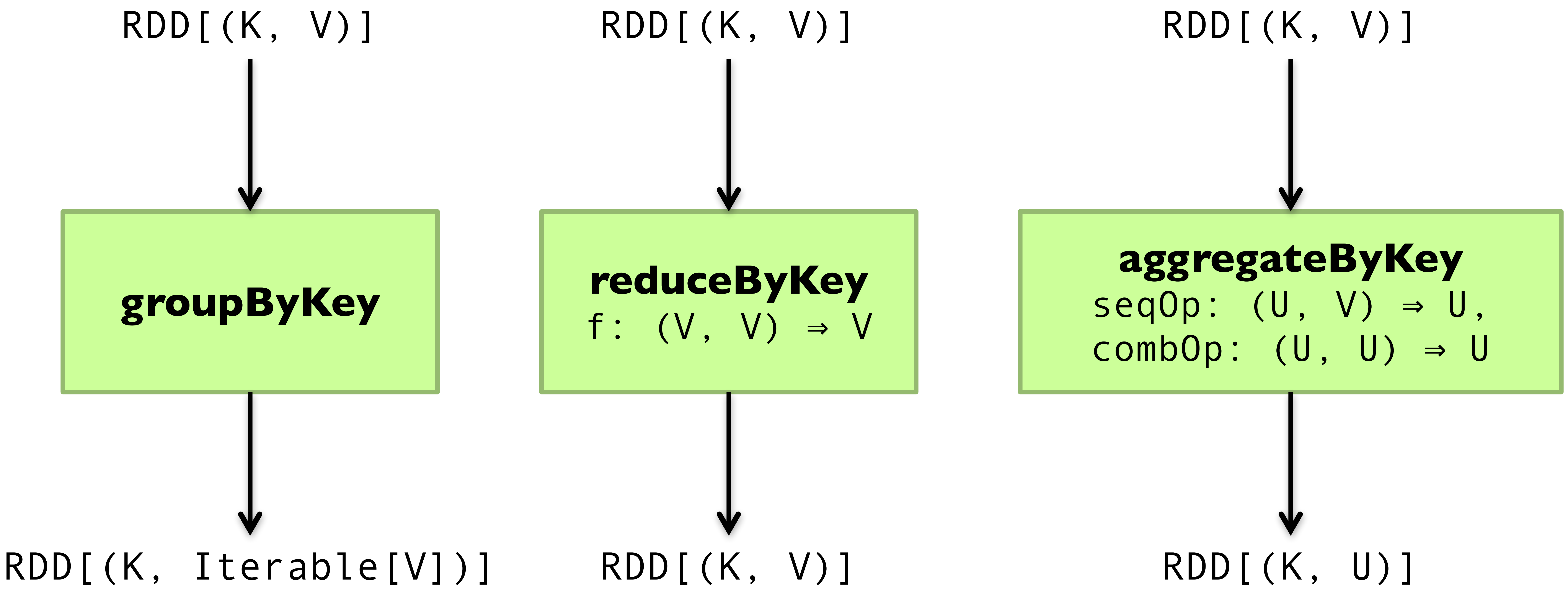
List[K3,V3])

Map-like Operations



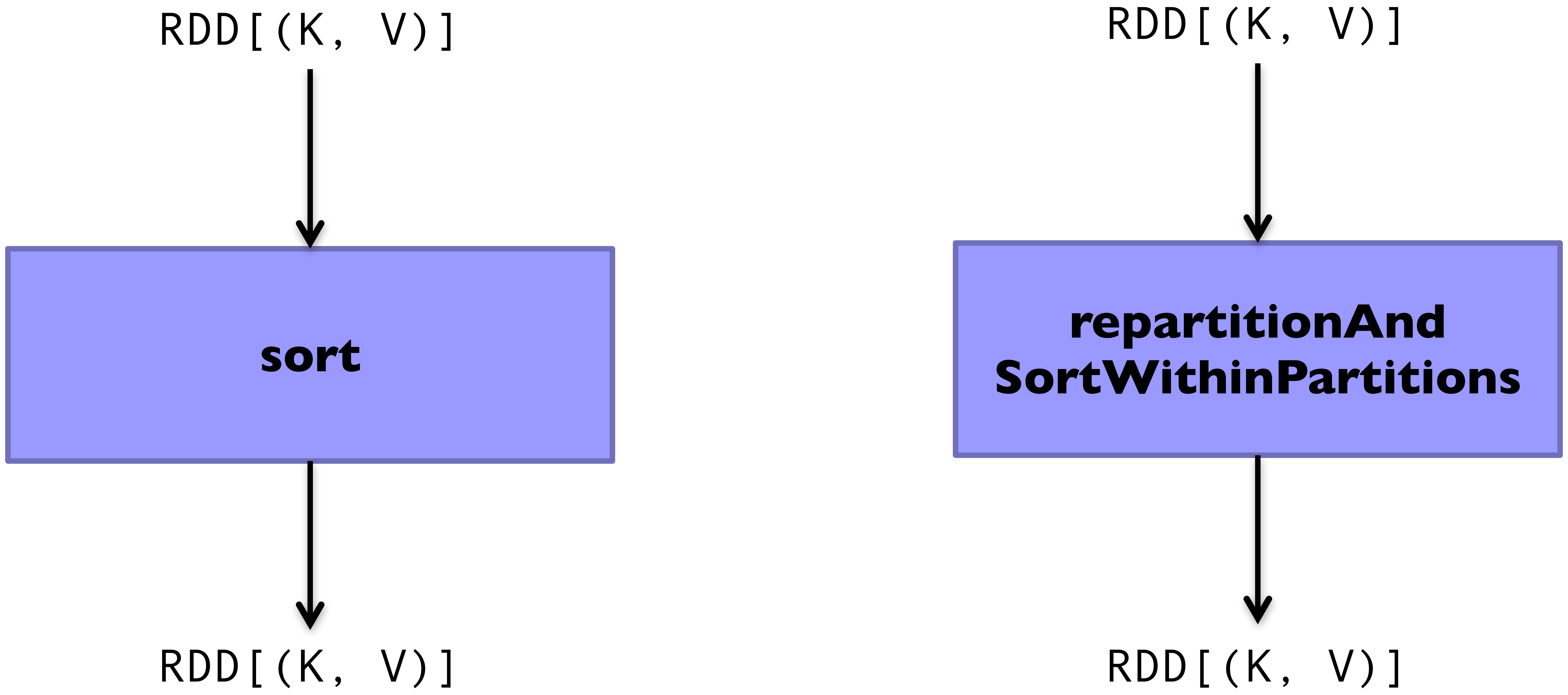
(Not meant to be exhaustive)

Reduce-like Operations



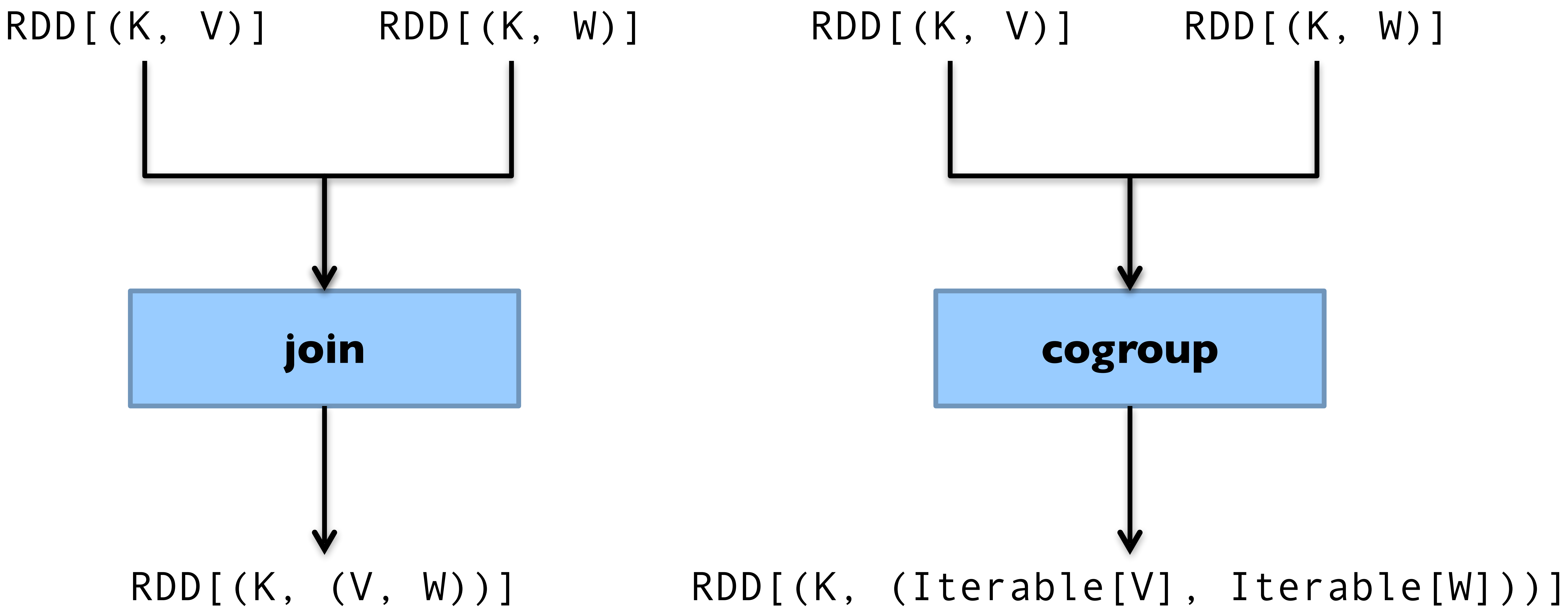
(Not meant to be exhaustive)

Sort Operations



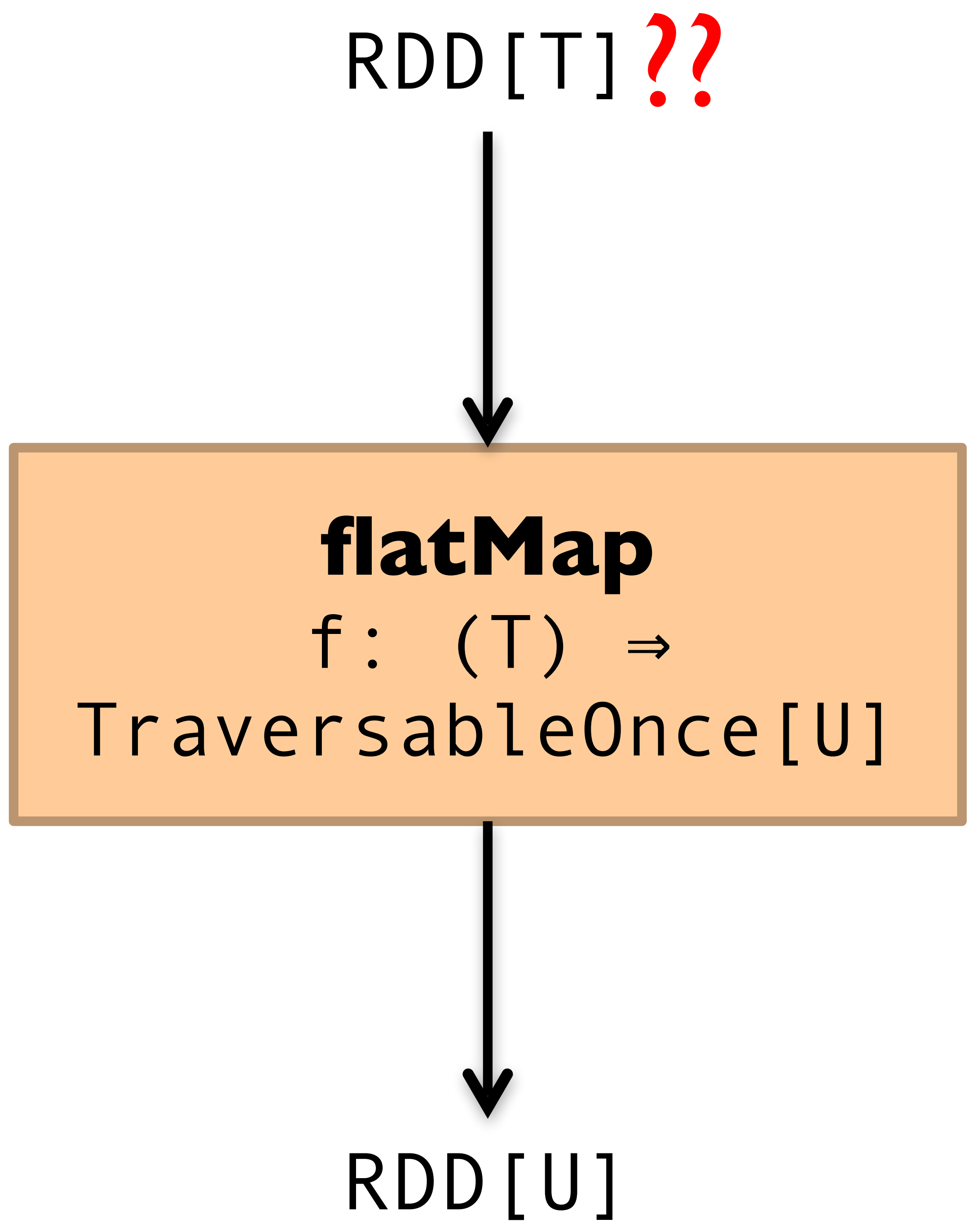
(Not meant to be exhaustive)

Join-like Operations



(Not meant to be exhaustive)

Spark Word Count




```
val textFile = sc.textFile(args.input())

textFile
  .flatMap(line => tokenize(line))
  .map(word => (word, 1))
  .reduceByKey((x, y) => x + y)
  .saveAsTextFile(args.output())
```

What's an RDD?

Resilient Distributed Dataset (RDD)

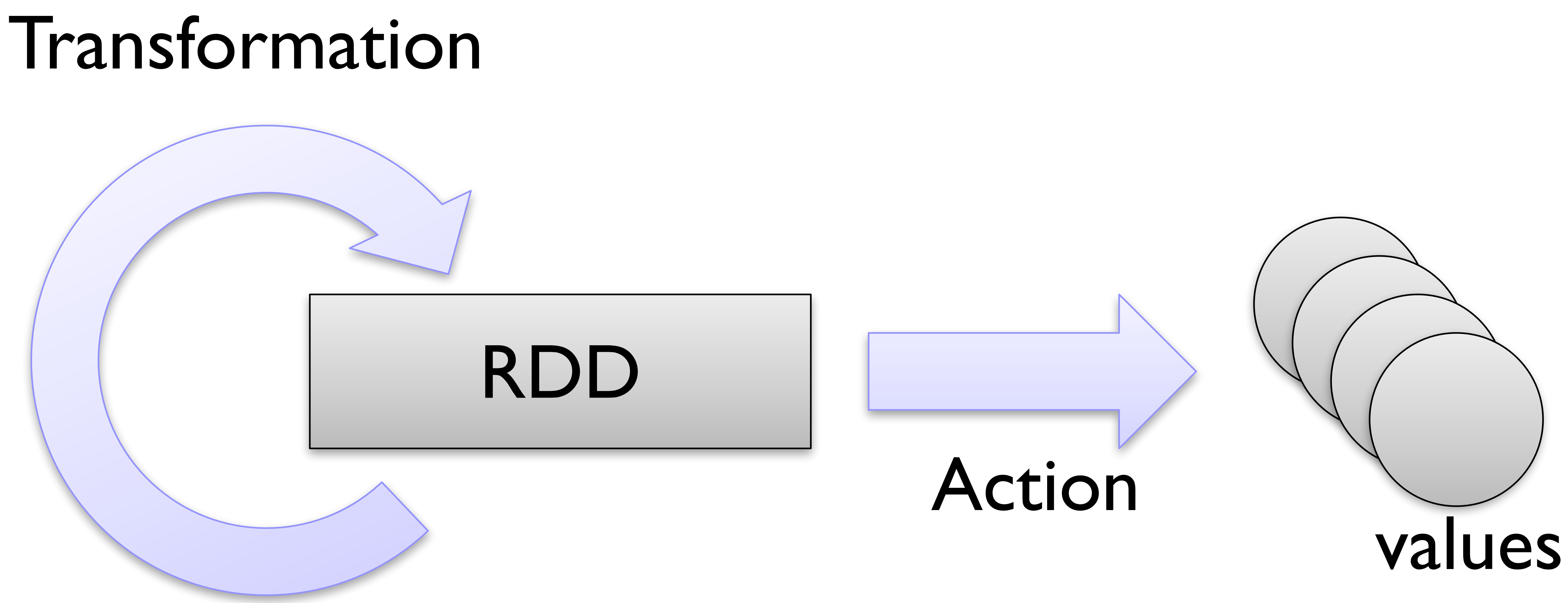
= immutable = partitioned

Wait, so how do you actually do anything?

Developers define *transformations* on RDDs

Framework keeps track of lineage

RDD Lifecycle



Transformations are lazy:
Framework keeps track of lineage

Actions trigger actual execution

Spark Word Count

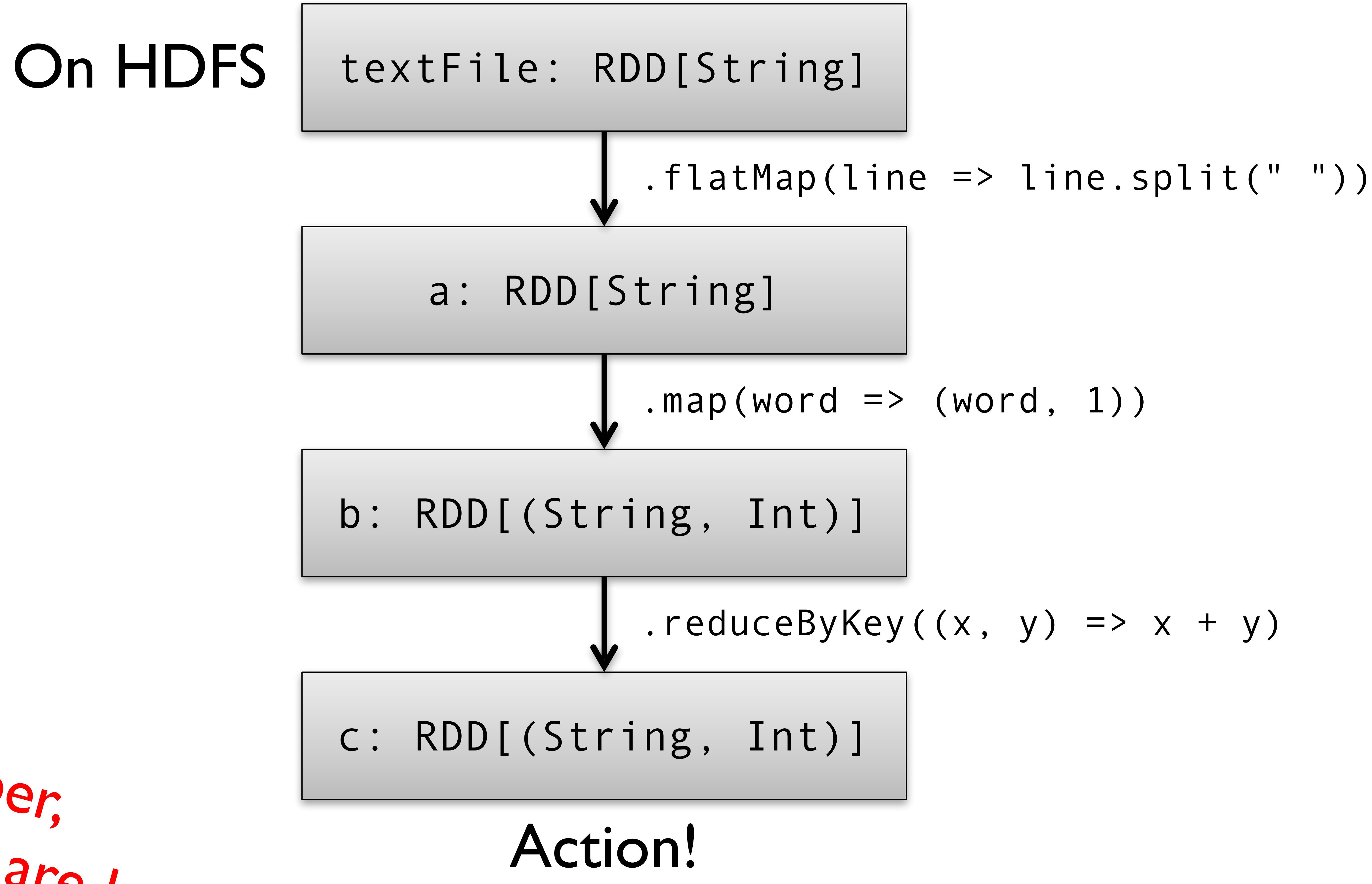
RDDs

```
val textFile = sc.textFile(args.input())

val a = textFile.flatMap(line => line.split(" "))
val b = a.map(word => (word, 1))
val c = b.reduceByKey((x, y) => x + y)

c.saveAsTextFile(args.output())
```

Action

Transformations

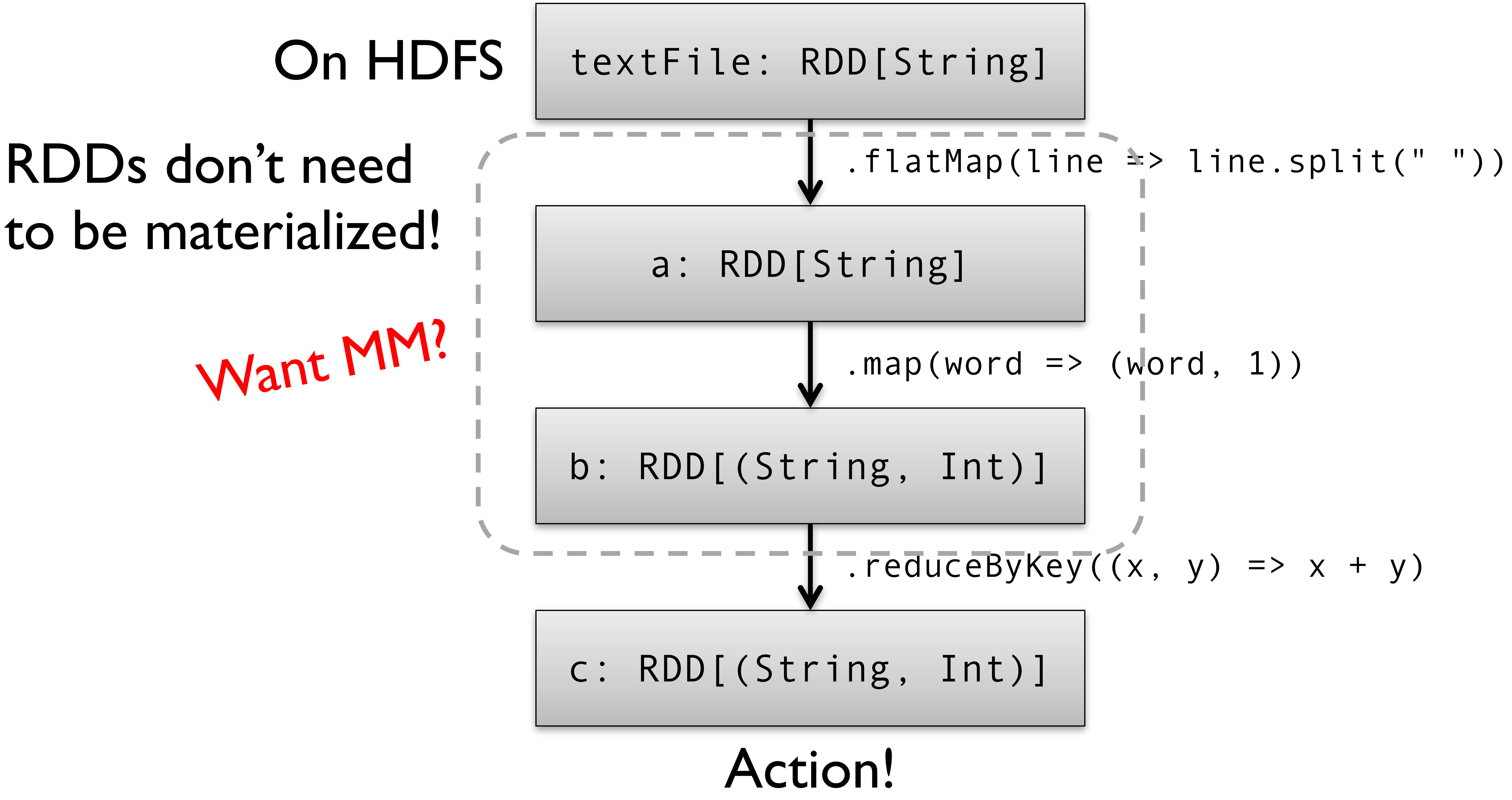
RDDs and Lineage



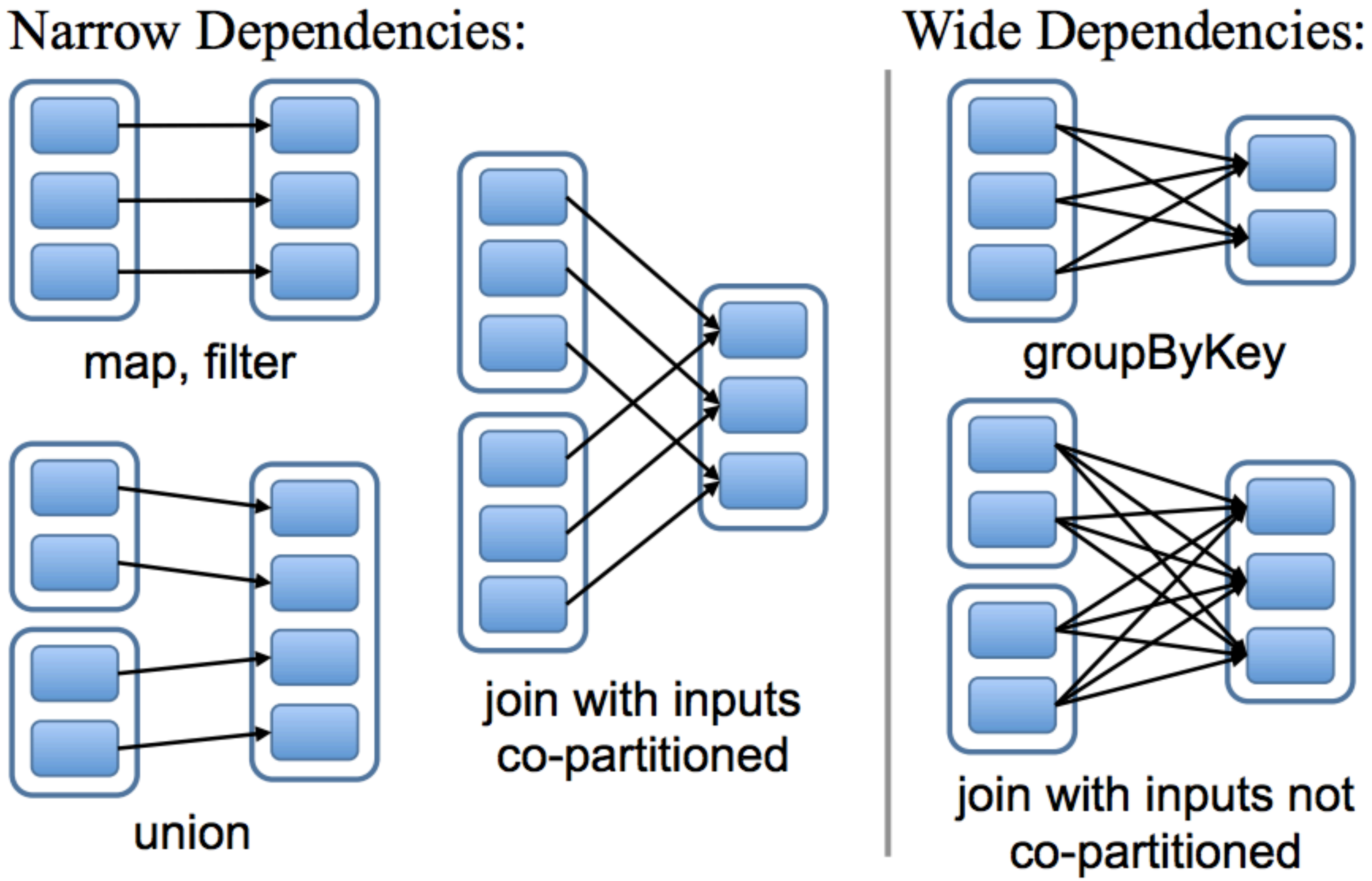
Remember, transformations are lazy!

RDDs and Optimizations

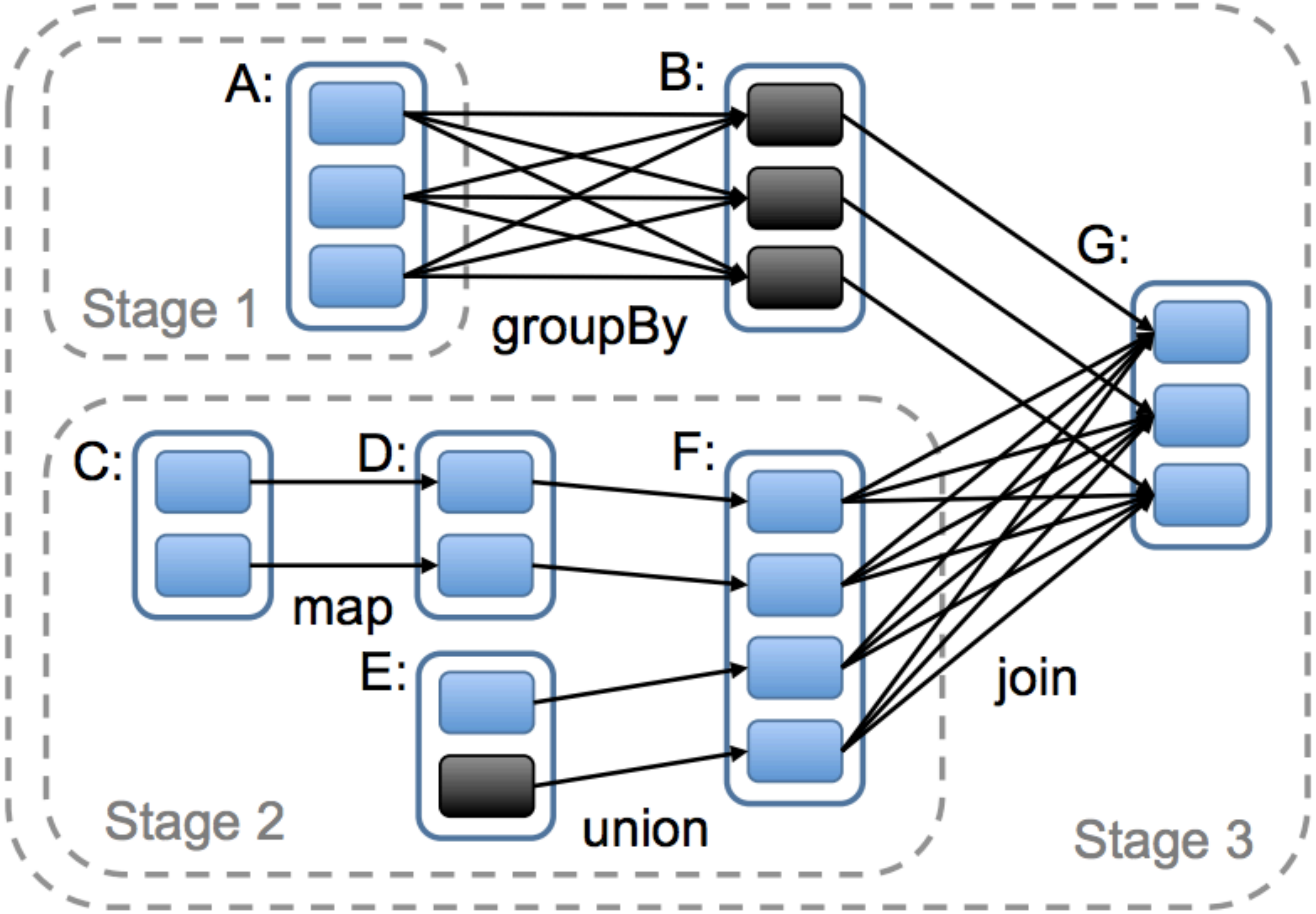
Lazy evaluation creates optimization opportunities



Physical Operators



Execution Plan



Kinda like a sequence of MapRedue jobs

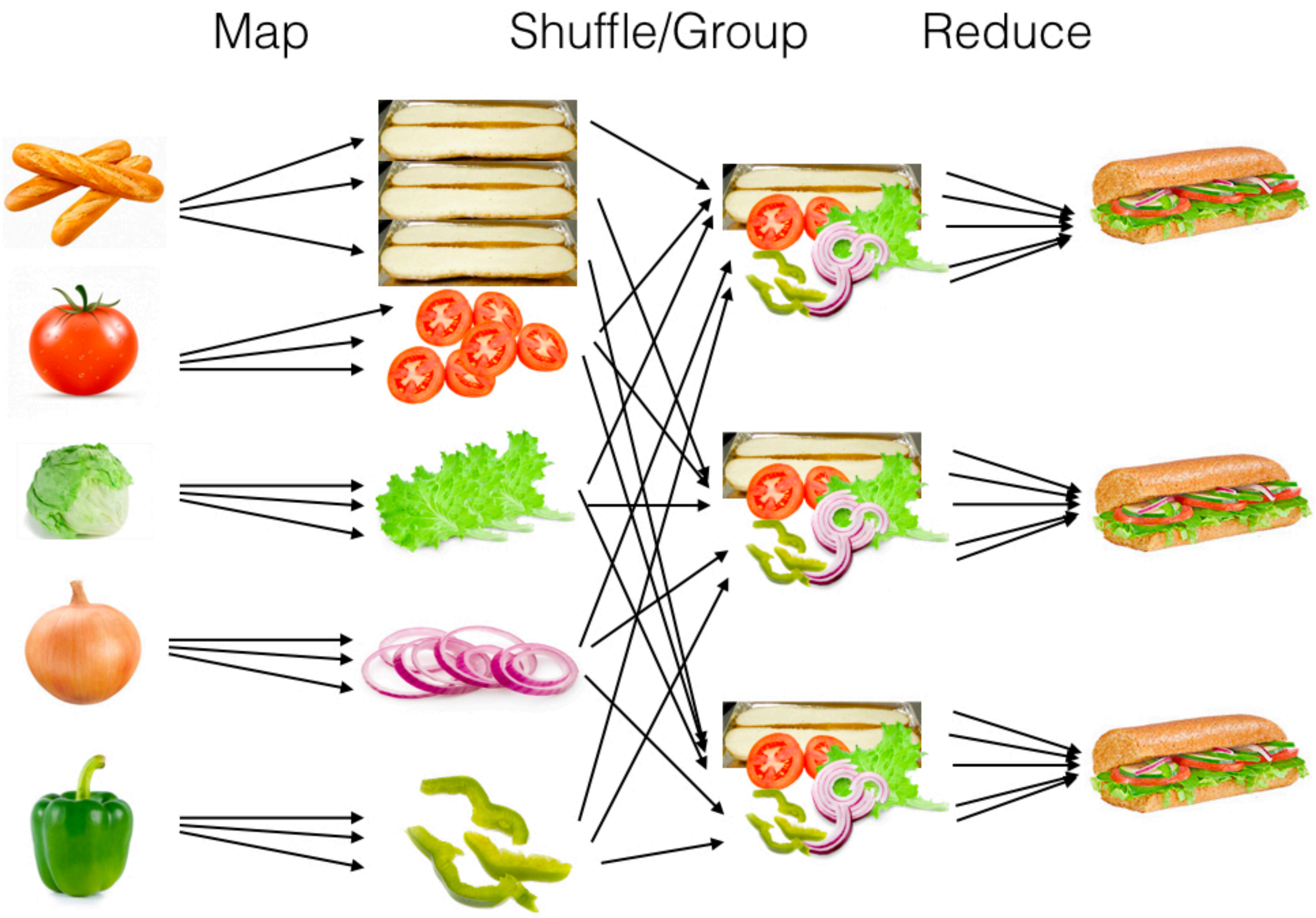
Can I abstract away RDD? I Like DataFrame

```
wc = (prompts_df.select("Prompt").rdd
        .flatMap(lambda r: (r["Prompt"] or "").split())
        .map(lambda w: (w, 1))
        .reduceByKey(lambda a, b: a + b))
```

```
from pyspark.sql.functions import col, split, explode

wc = (prompts_df
      .withColumn("word", explode(split(col("Prompt"), " ")))
      .groupBy("word").count())
```

```
df.Prompt.str.split().explode().reset_index().groupby("Prompt").count()
```

How to *Maproduce* Sandwiches



https://reberhardt.com/cs110/summer-2018/lecture-notes/lecture-14/