MapReduce

February 21, 2019
Data Science CSCI 1951A
Brown University
Instructor: Ellie Pavlick
HTAs: Wennie Zhang, Maulik Dang, Gurnaaz Kaur

Announcements

Announcements

- Next assignment released today—due March 7

Announcements

- Next assignment released today—due March 7
    - First project checkin will be assigned next week, so this will be happening…in parallel (<- that was a hilarious pun, FYI)

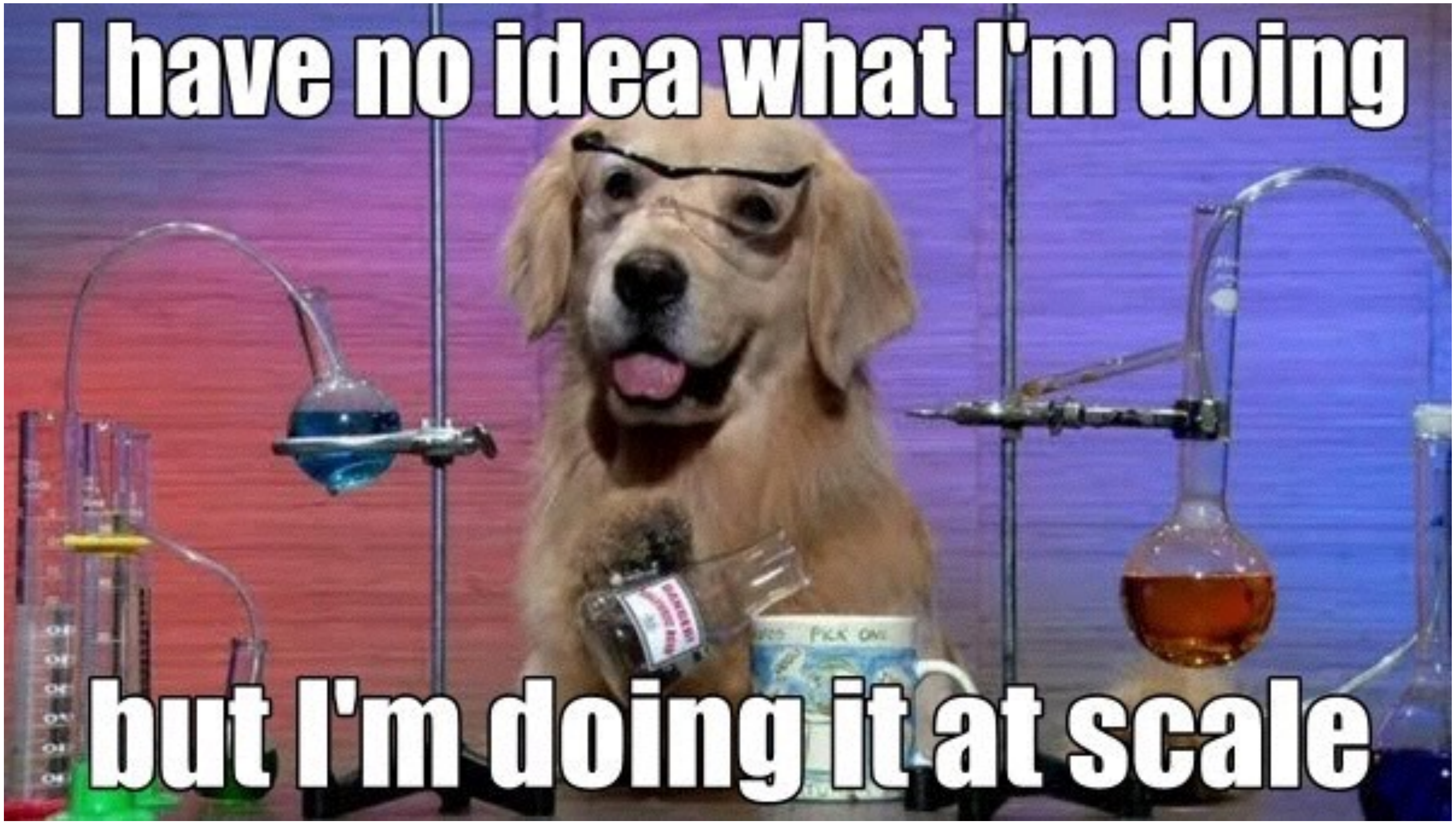
Announcements

- Next assignment released today—due March 7
    - First project checkin will be assigned next week, so this will be happening…in parallel (<- that was a hilarious pun, FYI)
- The HTAs are complaintless 👍

Announcements

- Next assignment released today—due March 7
    - First project checkin will be assigned next week, so this will be happening…in parallel (<- that was a hilarious pun, FYI)
- The HTAs are complaintless 👍
- Questions? Concerns? Anything?

Today



MapReduce

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

- Functional-programming paradigm (inspired by LISP and friends)

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

- Functional-programming paradigm (inspired by LISP and friends)
- Two functions:

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

- Functional-programming paradigm (inspired by LISP and friends)
- Two functions:
    - Map: (in_key, in_value) -> list_of(out_key, intermediate_value)

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

- Functional-programming paradigm (inspired by LISP and friends)
- Two functions:
    - Map: (in_key, in_value) -> list_of(out_key, intermediate_value)
    - Reduce: (out_key, list_of(intermediate_value)) -> list_of(out_value)

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

- Functional-programming paradigm (inspired by LISP and friends)

Extremely ~~Vague~~
General

- Two functions:
    - Map: (in_key, in_value) -> list_of(out_key, intermediate_value)
    - Reduce: (out_key, list_of(intermediate_value)) -> list_of(out_value)

https://research.google.com/archive/mapreduce-osdi04-slides

MapReduce

distributed grep
distributed sort
web link-graph reversal
web access log stats
inverted index construction
document clustering
machine learning
statistical machine translation
...

https://research.google.com/archive/mapreduce-osdi04-slides

Map Reduce

- One “master” scheduler which assigns tasks (mapping or reducing) to machines

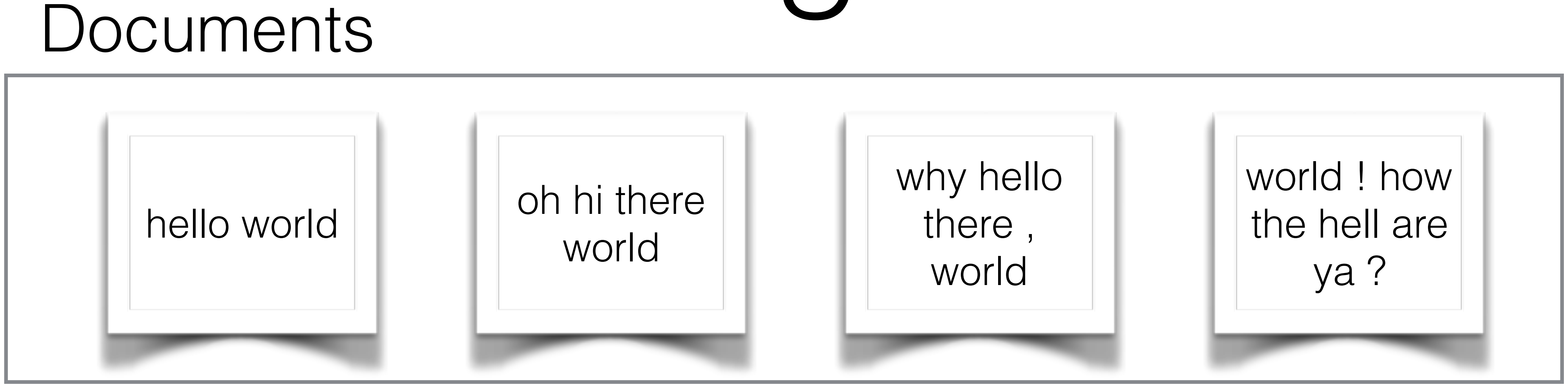
Map Reduce

- One “master” scheduler which assigns tasks (mapping or reducing) to machines
- No shared state between machines—massively parallelizable

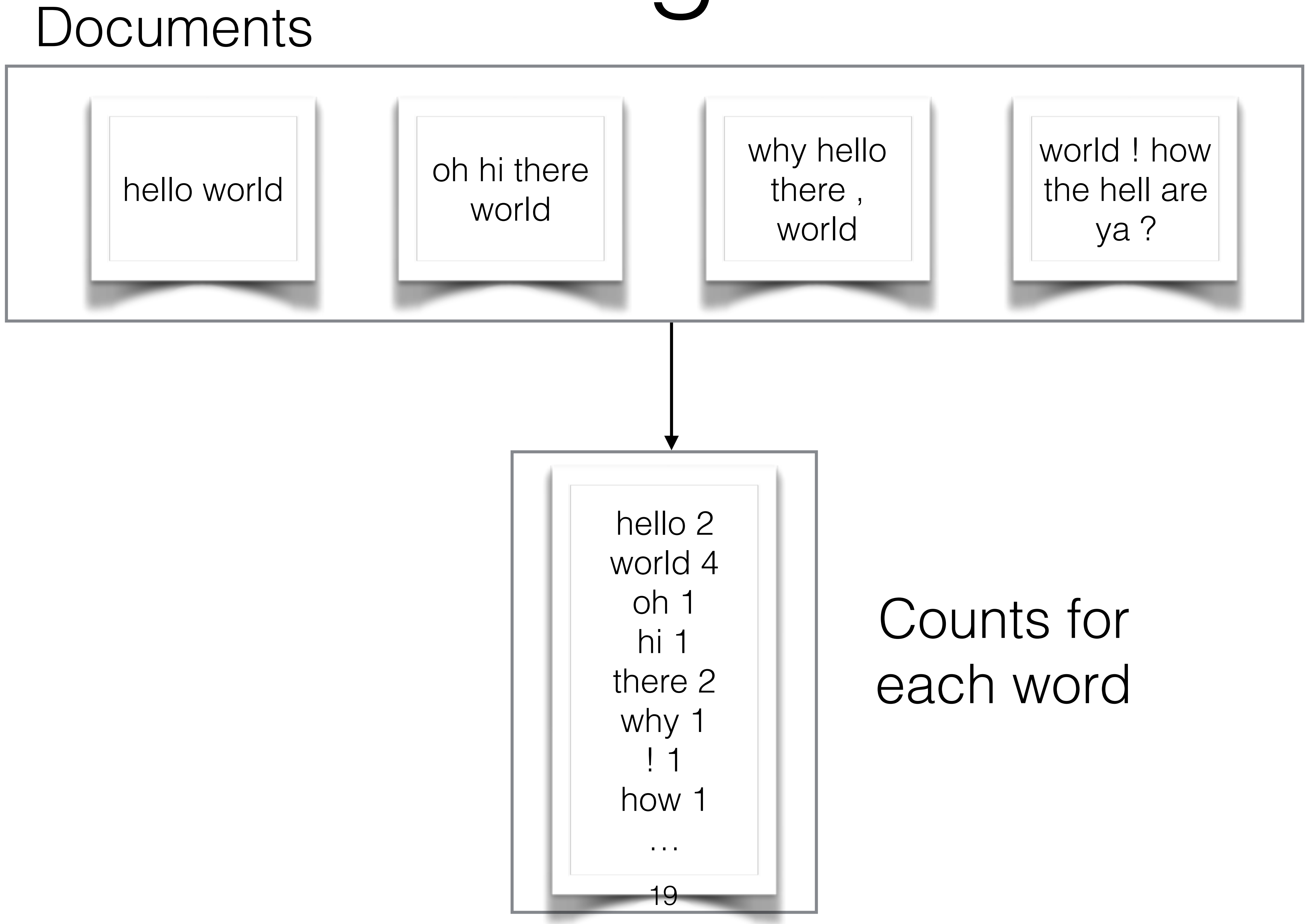
Map Reduce

- One “master” scheduler which assigns tasks (mapping or reducing) to machines
- No shared state between machines—massively parallelizable
- Assume very high failure rates on workers

Counting Words



Counting Words



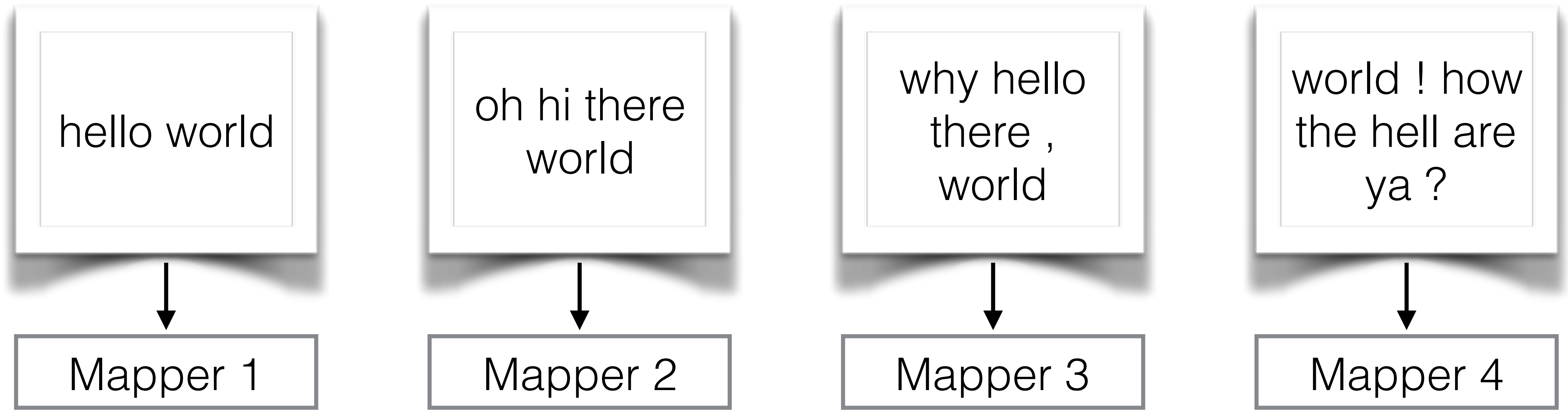
Counts for each word

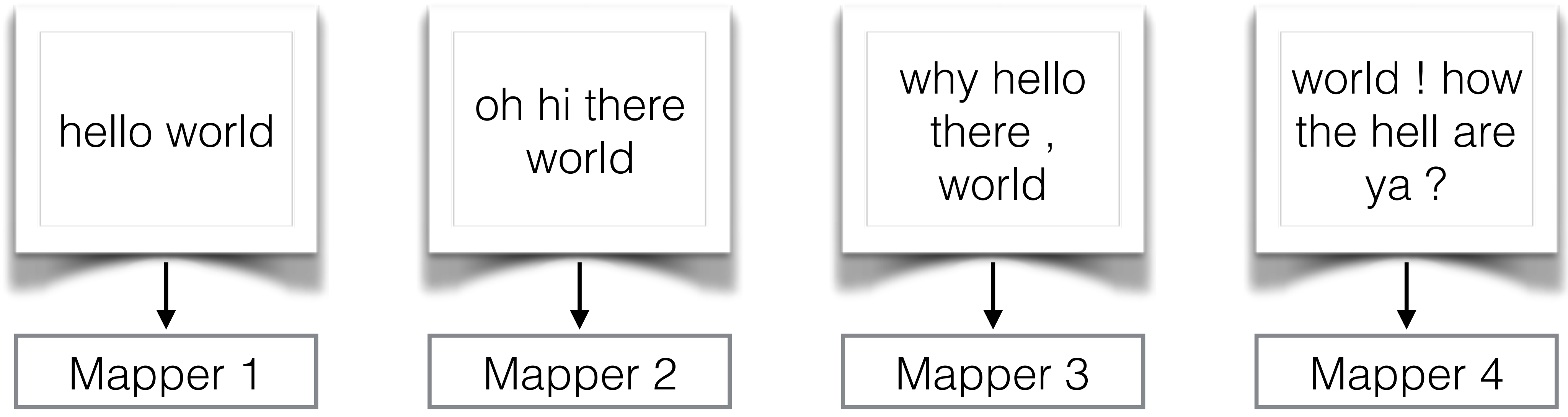
hello world

oh hi there world

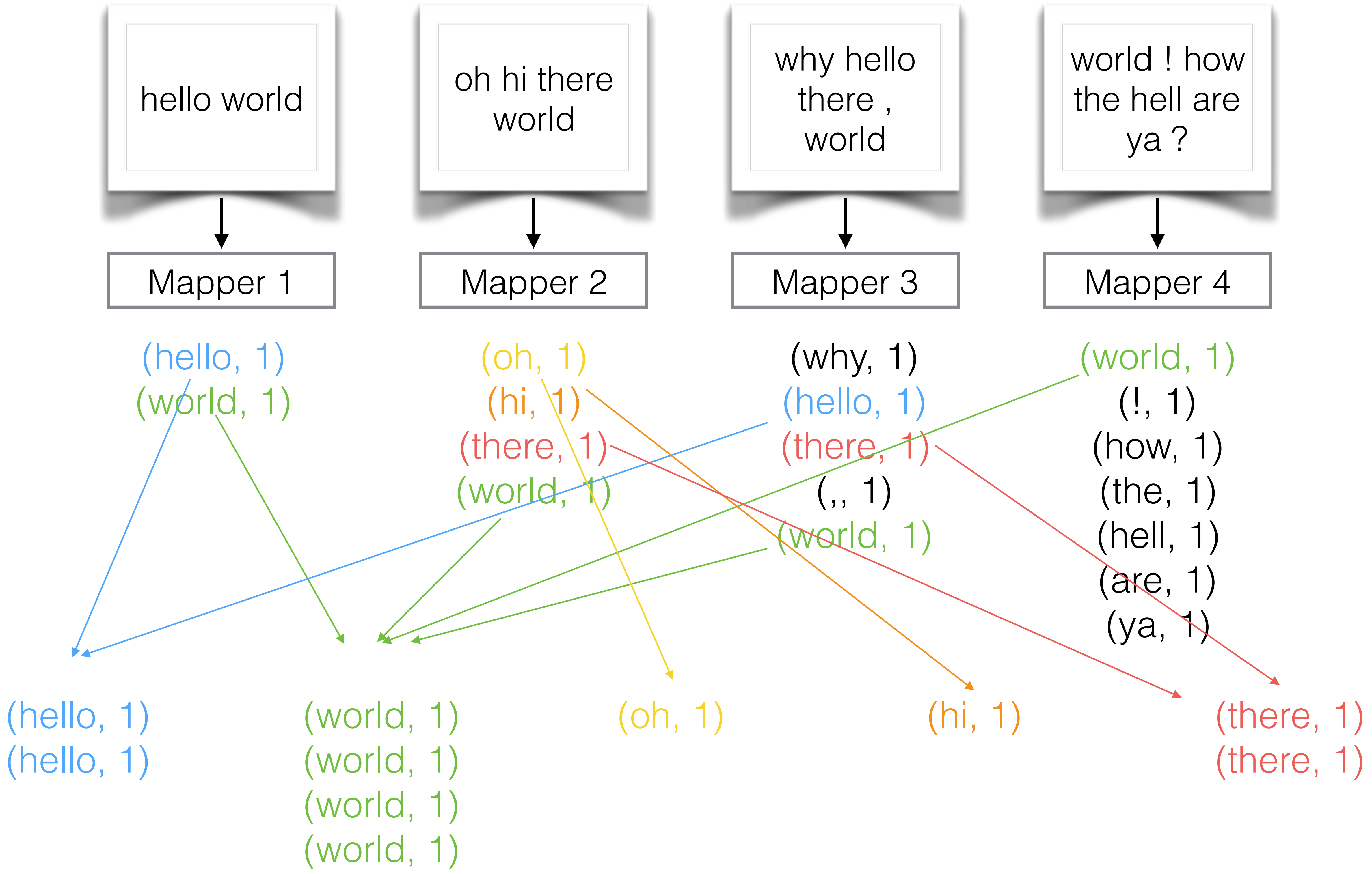
why hello there , world

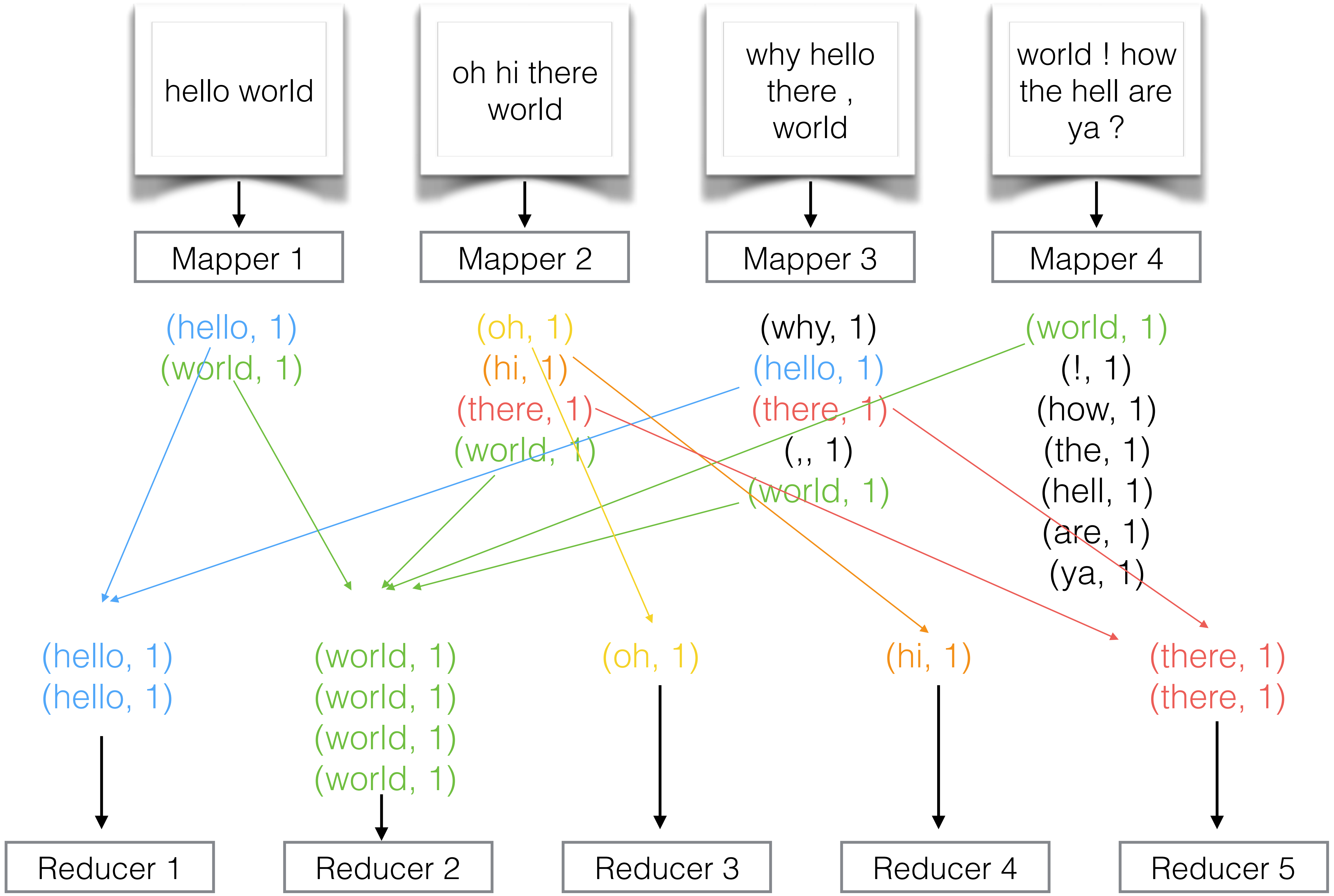
world ! how the hell are ya ?

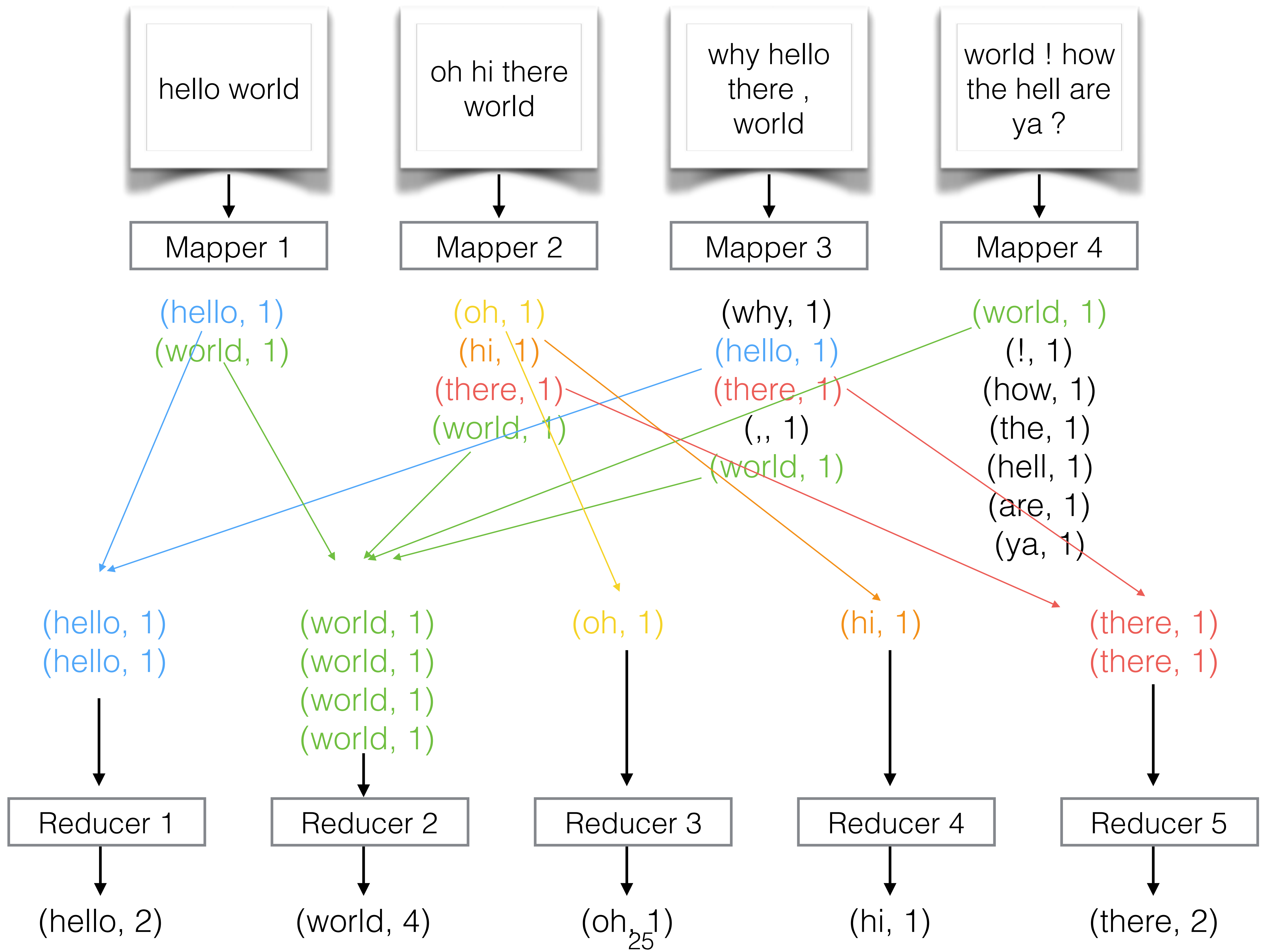

hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4

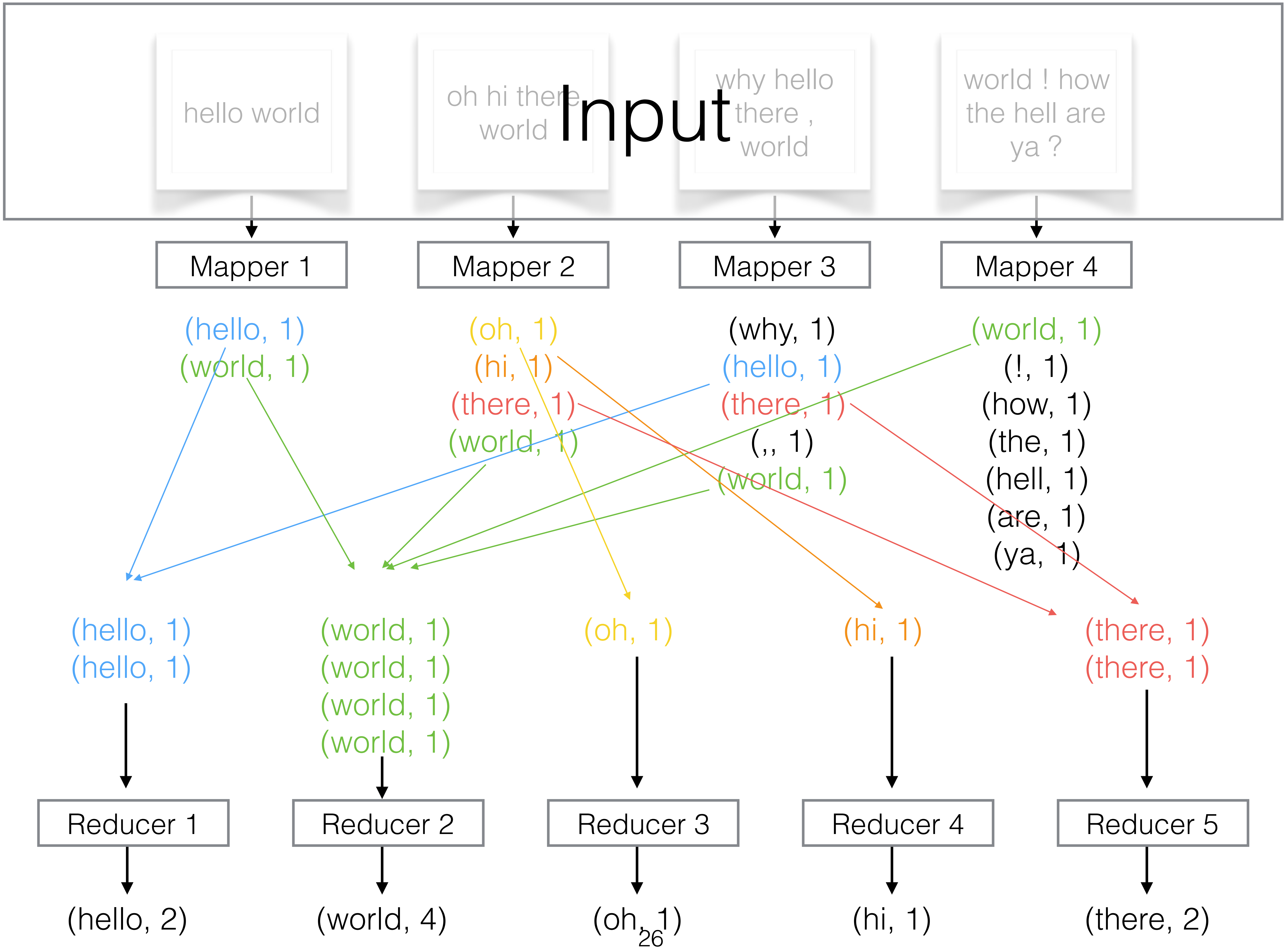


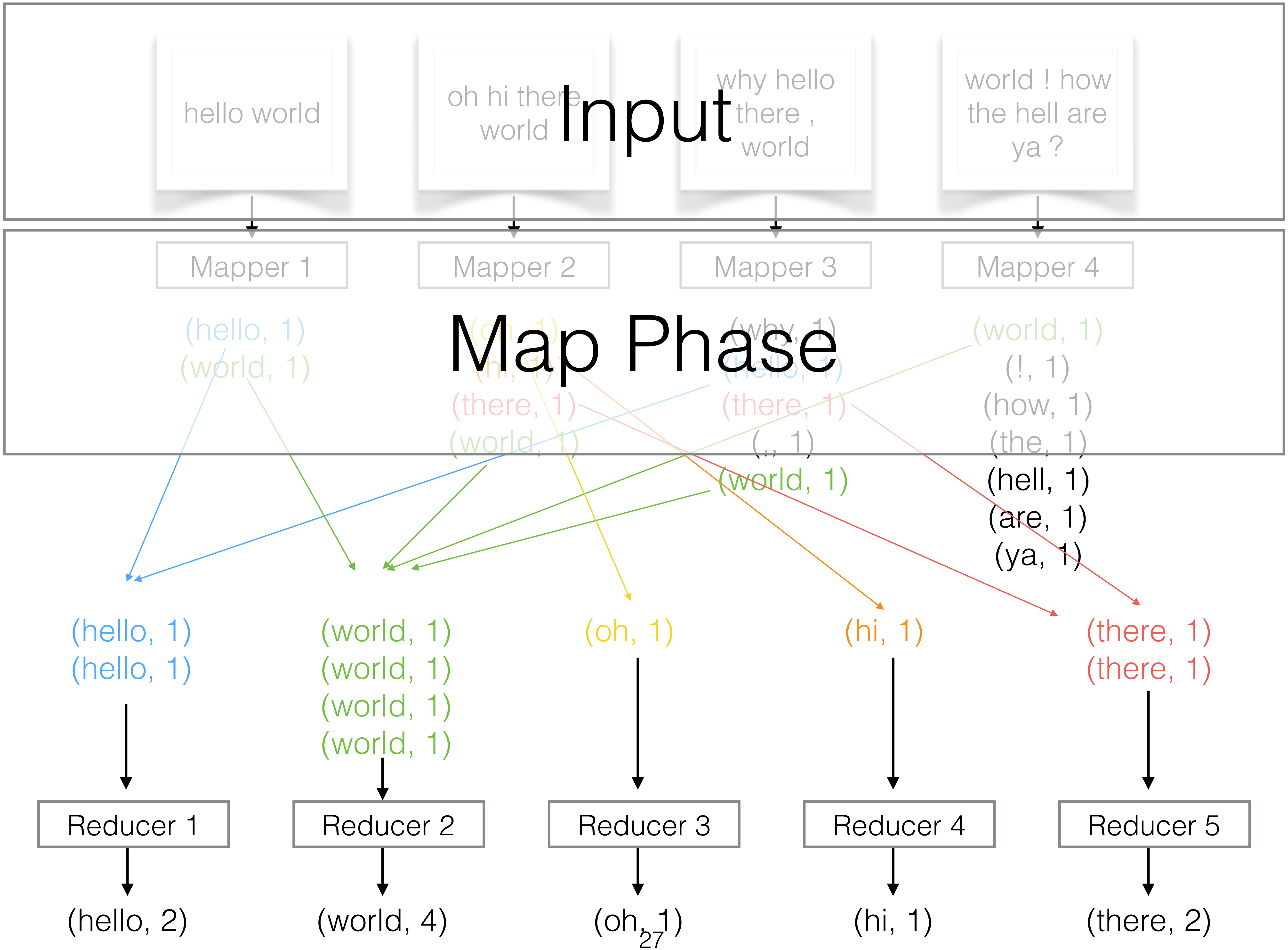
(hello, 1)	(oh, 1)	(why, 1)	(world, 1)
(world, 1)	(hi, 1)	(hello, 1)	(!, 1)
	(there, 1)	(there, 1)	(how, 1)
	(world, 1)	(,, 1)	(the, 1)
		(world, 1)	(hell, 1)
			(are, 1)
			(ya, 1)

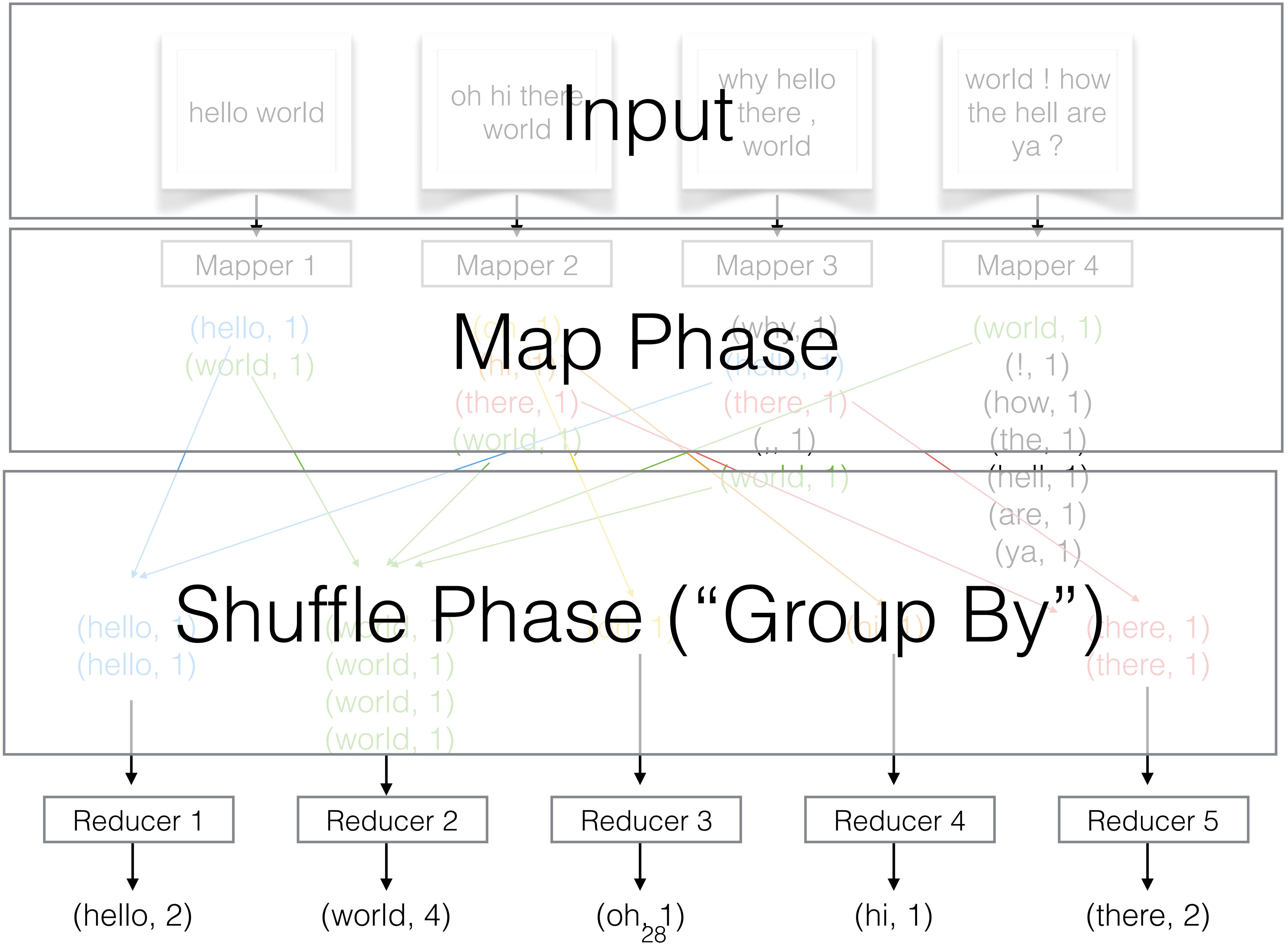

hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)

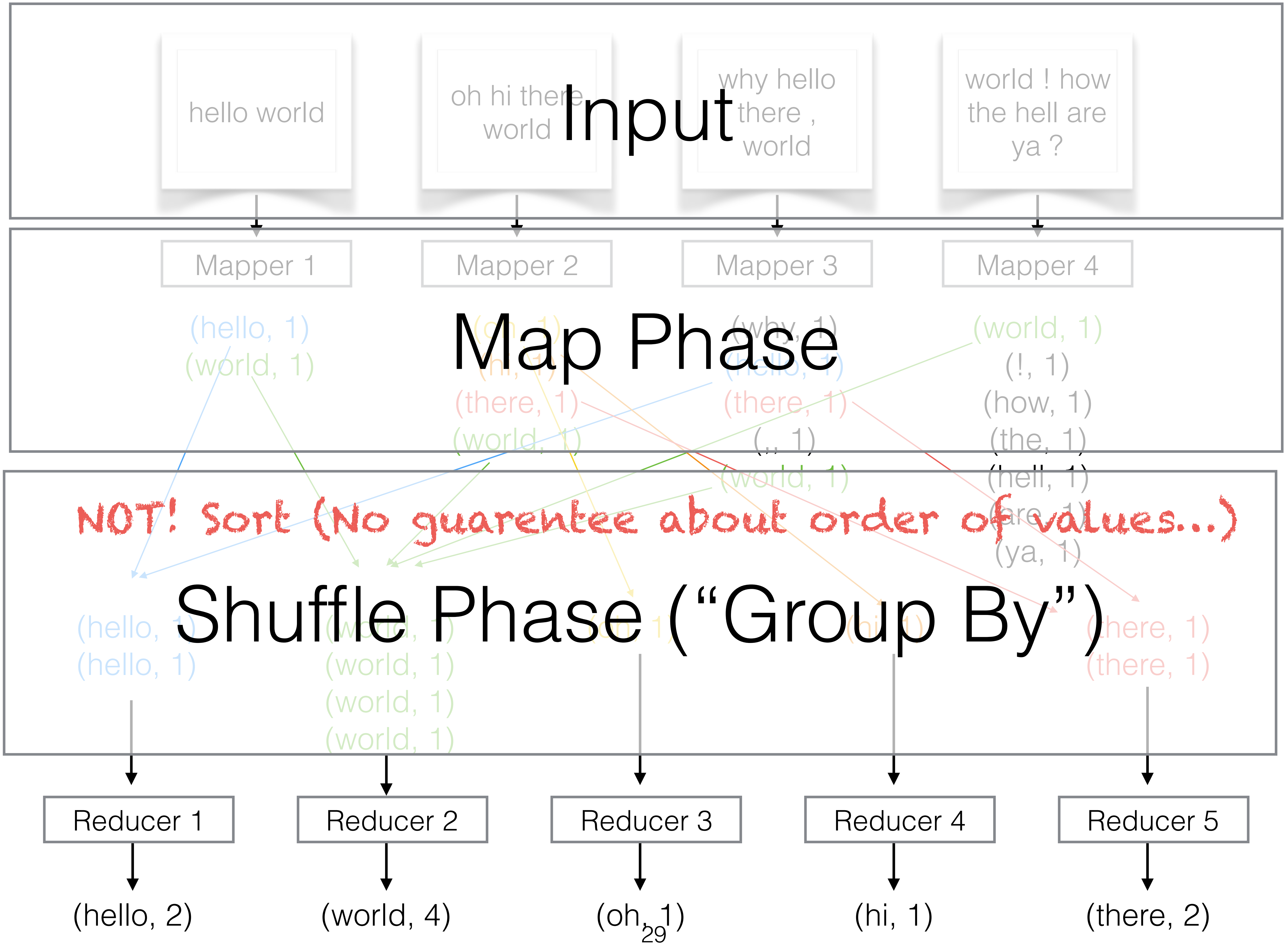

hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5

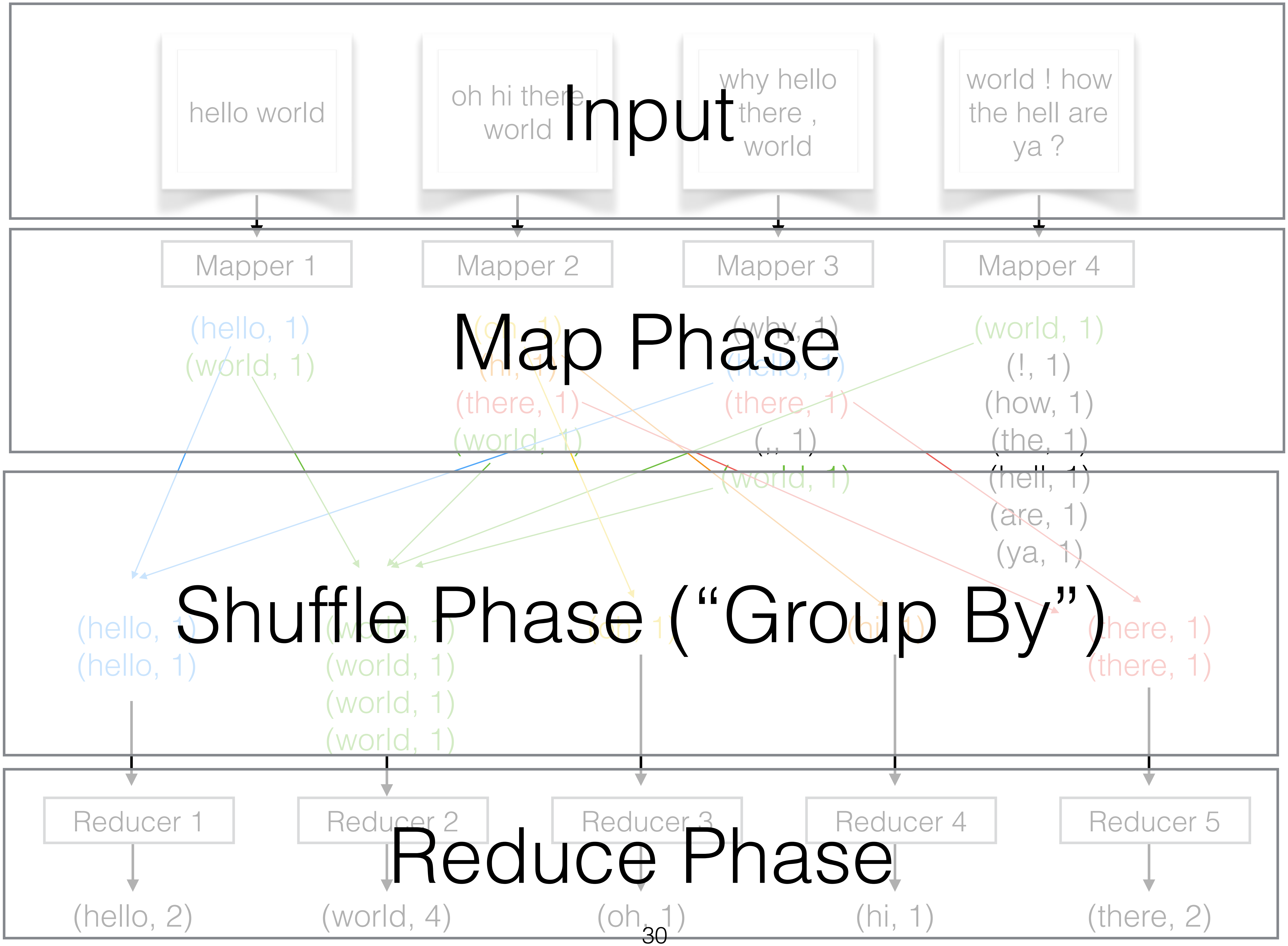

hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)

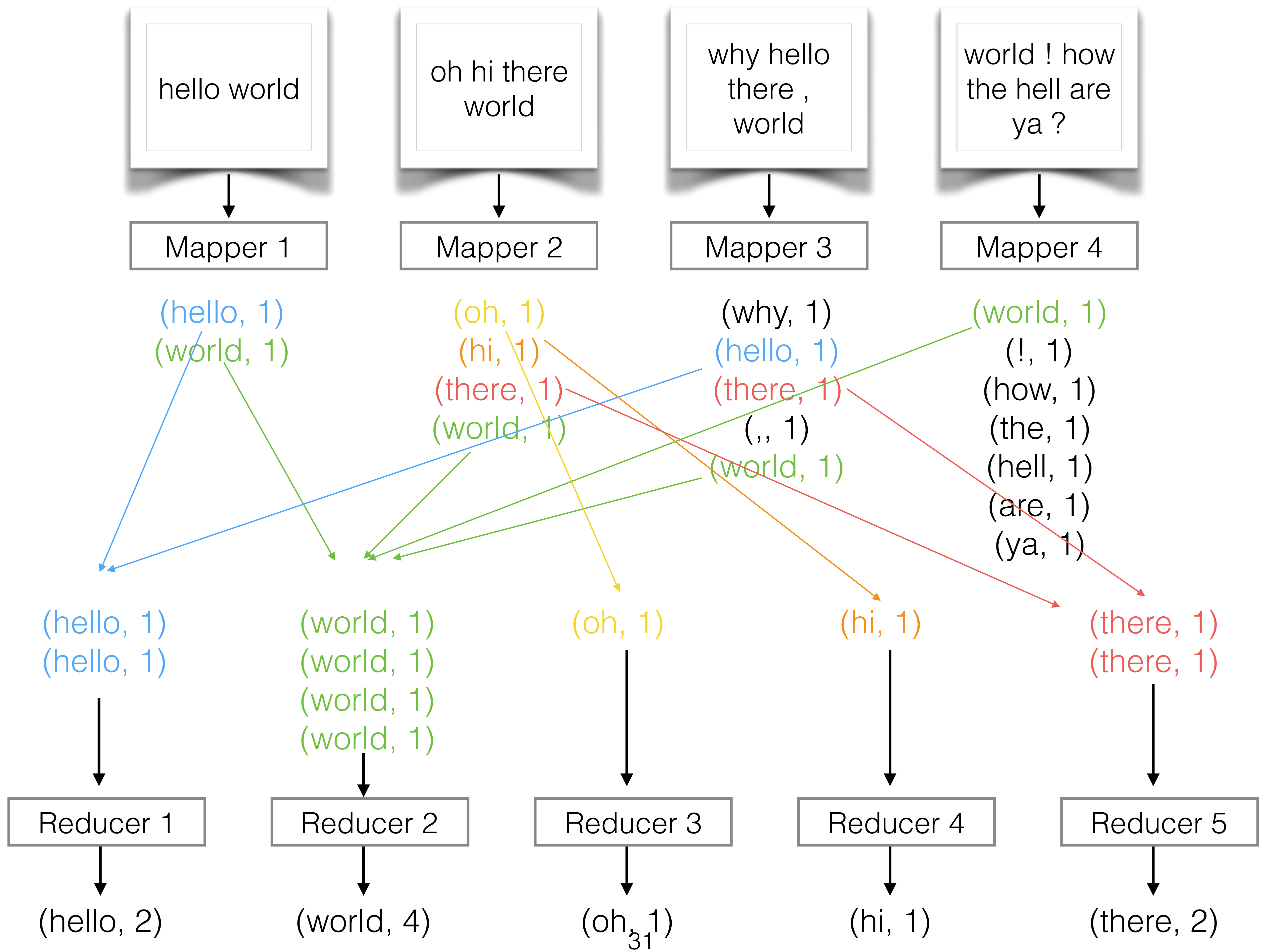

Input
hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)

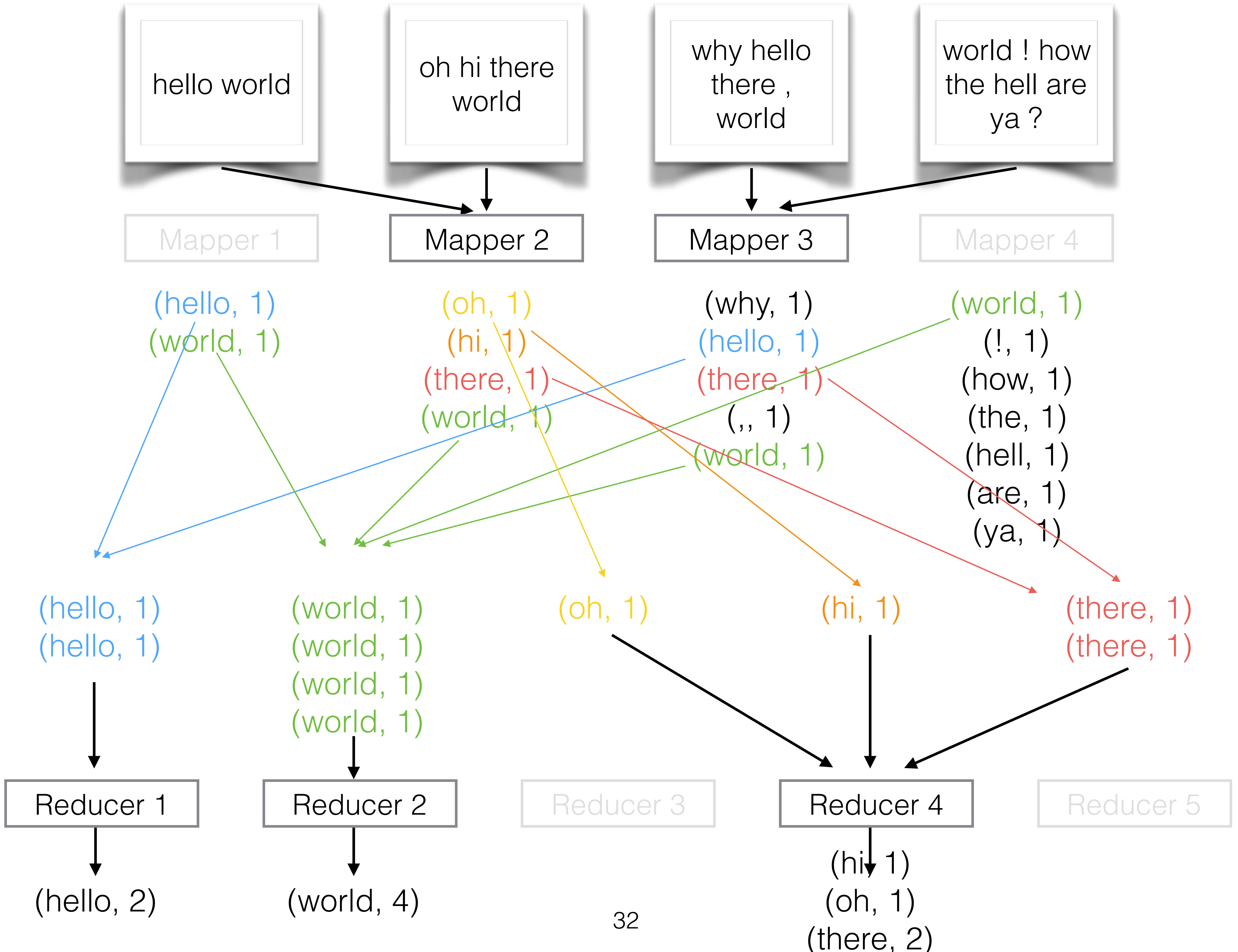

Input
hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
Map Phase
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)

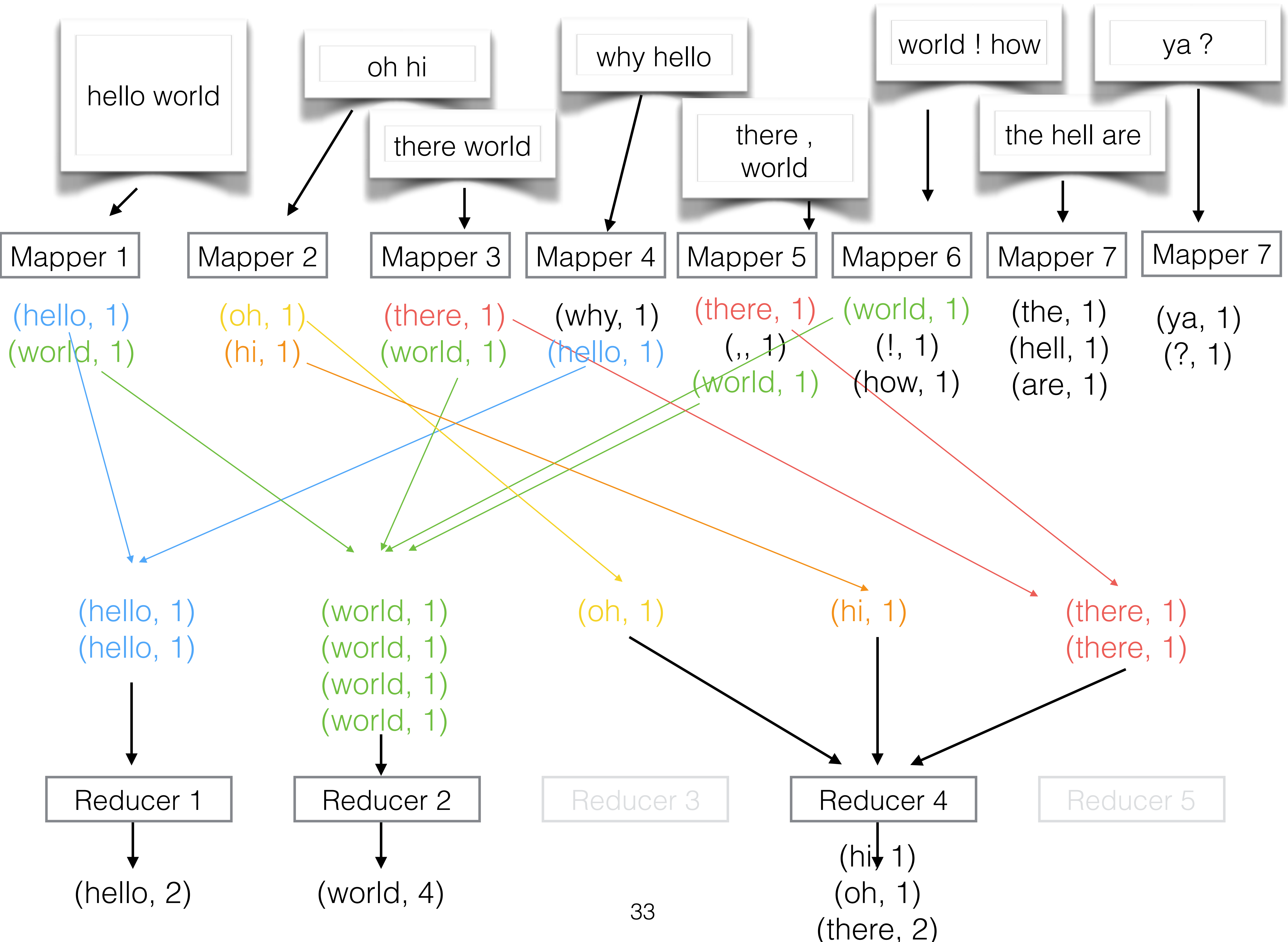

Input
hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Map Phase
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
Shuffle Phase (“Group By”)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)


Input
hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Map Phase
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
NOT! Sort (No guarentee about order of values...)
Shuffle Phase ("Group By")
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)


Input
hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Map Phase
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
Shuffle Phase (“Group By”)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reduce Phase
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)


hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 2)


hello world
oh hi there world
why hello there , world
world ! how the hell are ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(hi, 1)
(oh, 1)
(there, 2)


hello world
oh hi
there world
why hello
there , world
world ! how
the hell are
ya ?
Mapper 1
Mapper 2
Mapper 3
Mapper 4
Mapper 5
Mapper 6
Mapper 7
Mapper 7
(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(?, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
(hello, 2)
(world, 4)
(hi, 1)
(oh, 1)
(there, 2)

Map Reduce

```
//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) {
TODO;
}

//define your reduce function(s)
def ReduceFn:(String, Int) -> (String, Int){
TODO;
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
```

Map Reduce

```
//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) {
TODO;
}

//define your reduce functio
def ReduceFn:(String, Int) -                         {
TODO;
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
```

table

DocID	Text
1	hello world
2	oh hi there world
3	why hello there , world
4	world ! how the hell are ya ?

Map Reduce

```
//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) {
TODO;
}

//c                 ce functio
def                 ng, Int) -                {
TOI
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
```

output

Word	Count
hello	2
world	4
oh	1
hi	1
there	2

table

DocID	Text
1	hello world
2	oh hi there world
3	why hello there , world
4	world ! how the hell are ya ?

Map Reduce

```
//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) {
TODO;
}

//define your reduce function(s)
def ReduceFn:(String, Int) -> (String, Int){
TODO;
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
```

Lots of data types:
String, Int, Float, Tuples thereof

Map Reduce

```
// enumerate occurrences of each word, with
// count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.value().split(){
        emit(w, 1);
    }
}
```

Map Reduce

```
// enumerate occurrences of each word, with
// count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.value().split(){
        emit(w, 1);
    }
}
```

String

Map Reduce

```
// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int){
    sum = 0;
    for c in input.value(){
        sum += c;
    }
    emit(input.key(), sum);
}
```

Map Reduce

```
// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int){
    sum = 0;
    for c in input.value() ← list of ints (counts)
        sum += c;
    }
    emit(input.key(), sum);
}
```

Map Reduce

```
// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int){
    sum = 0;
    for c in input.value() ← list of ints (counts)
        sum += c;
    }
    emit(input.key(), sum); ← the word
}
```

Find the number of occurrences of each word?

```
// enumerate occurrences of each word
// with count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.split(){
        emit(w, 1);
    }
}

// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int){
    sum = 0;
    for (w, c) in input{ sum += c; }
    emit(w, sum);
}

// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
}
```

$$\begin{tikzcd}
\text{Input: String} \arrow[d] \\
\text{Map: output (word, 1) for every word.} \arrow[d] \\
\text{Reduce: Sum counts for each word}
\end{tikzcd}$$

(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().ReduceFn1().ReduceFn2();
write(output)
}
```

(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().ReduceFn1().ReduceFn2();
write(output)
}
```

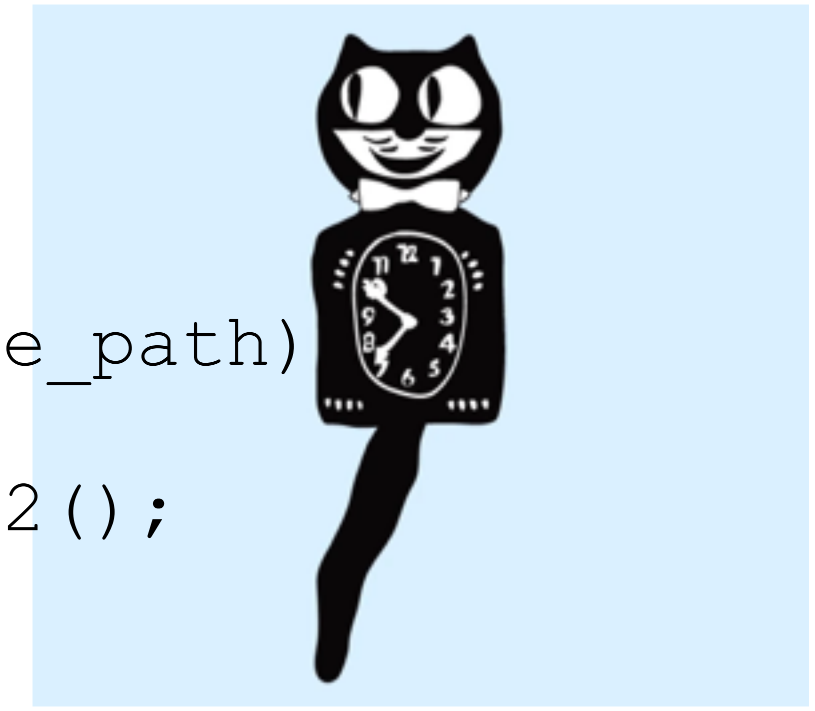
No using sets!

(non)Clicker Question!

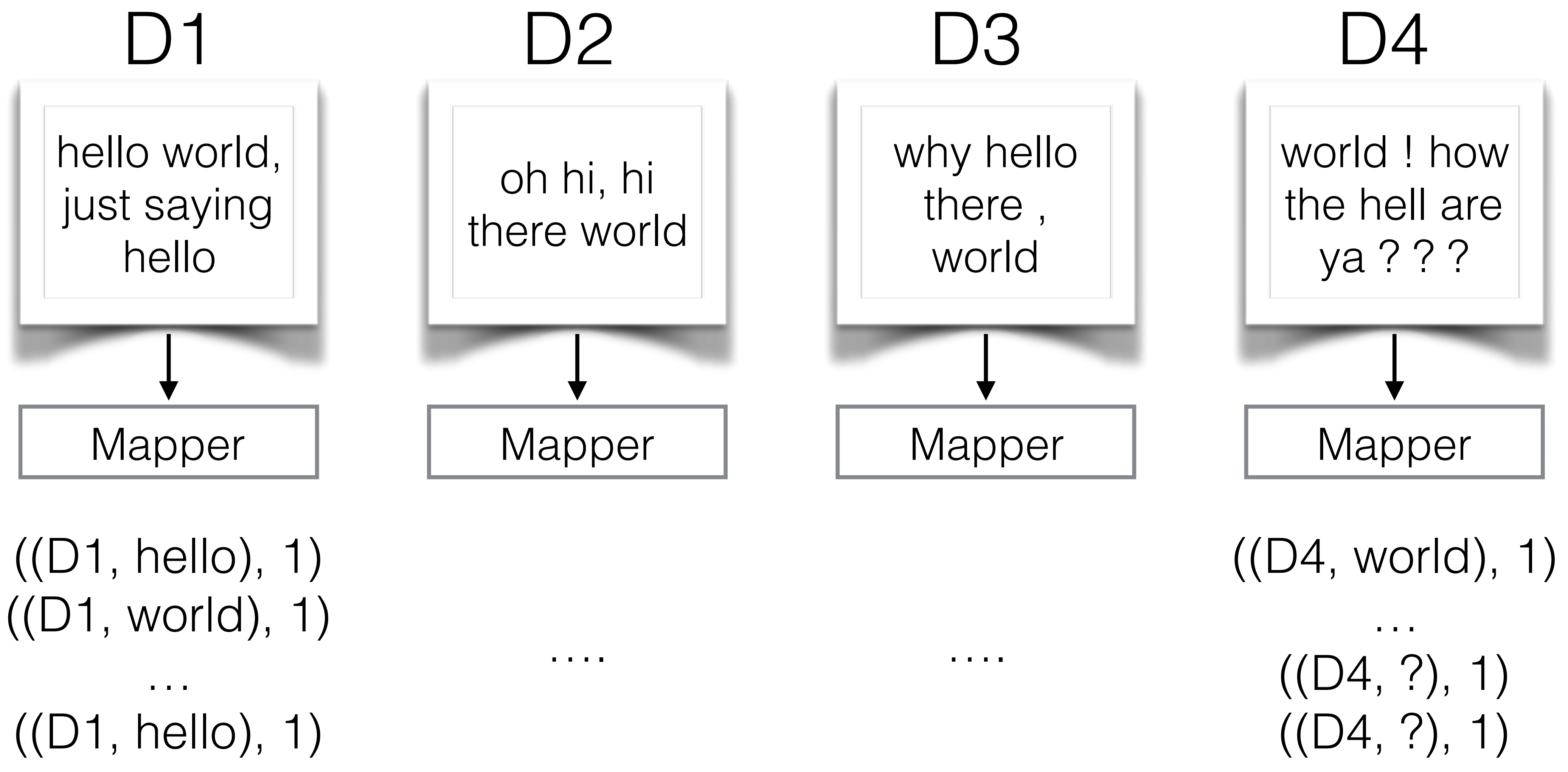
Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().ReduceFn1().ReduceFn2();
write(output)
}
```

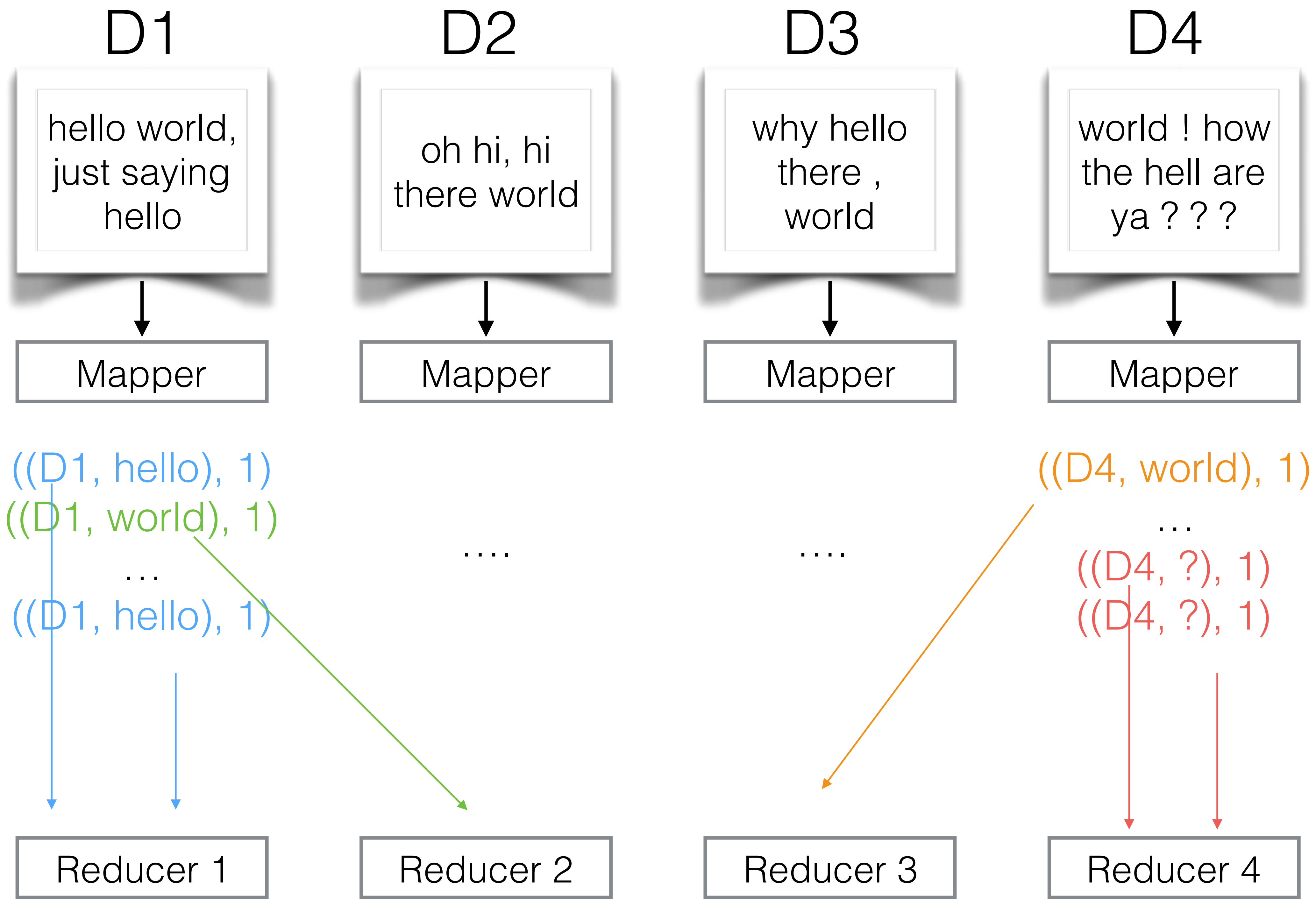
No using sets!

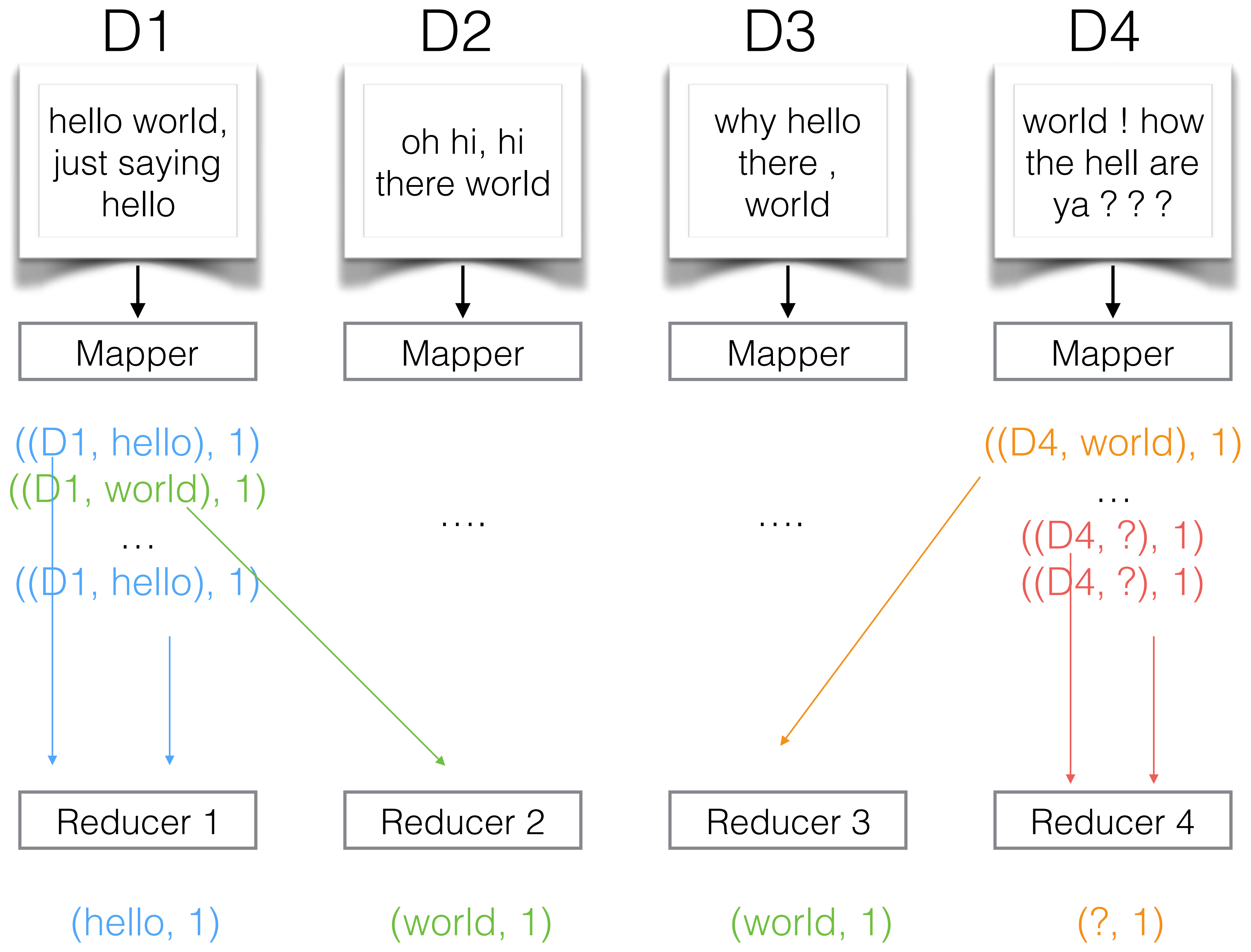


D1	D2	D3	D4
hello world, just saying hello	oh hi, hi there world	why hello there , world	world ! how the hell are ya ? ? ?

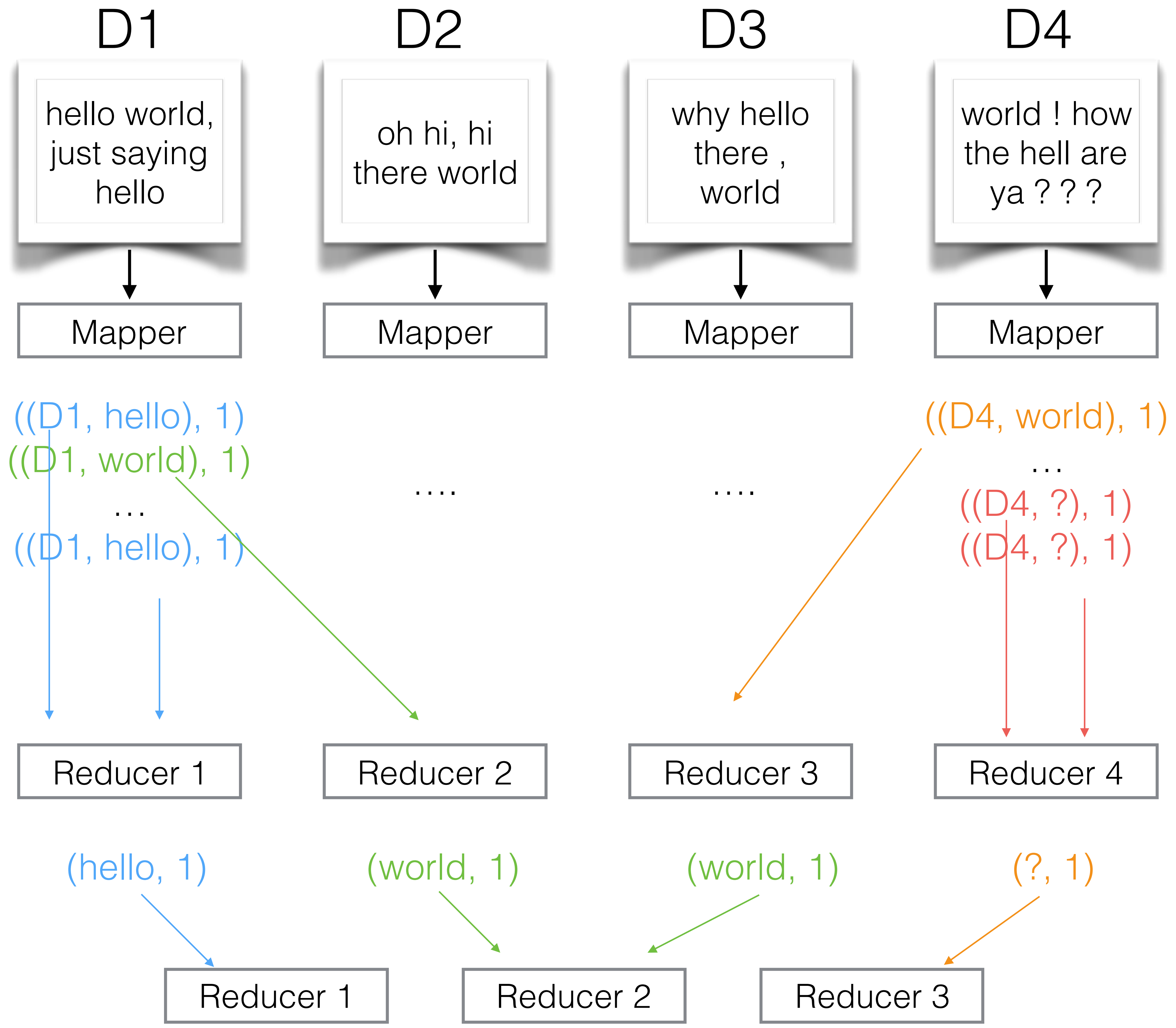


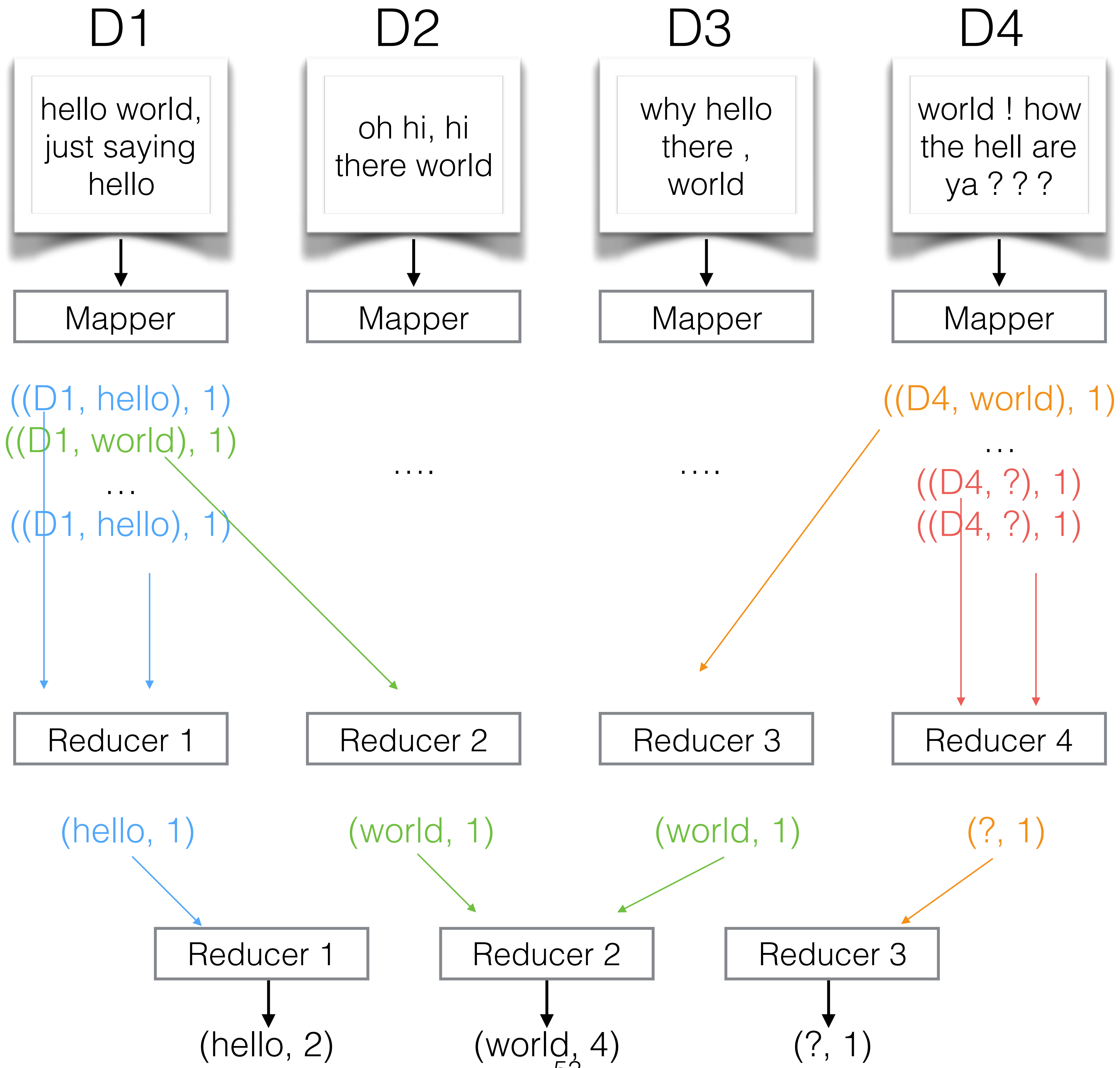
D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)
....
....
((D4, world), 1)
...
((D4, ?), 1)
((D4, ?), 1)


D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)
....
....
((D4, world), 1)
...
((D4, ?), 1)
((D4, ?), 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4



D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
. . .
((D1, hello), 1)
. . . .
. . . .
((D4, world), 1)
. . .
((D4, ?), 1)
((D4, ?), 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
(hello, 1)
(world, 1)
(world, 1)
(?, 1)


D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)
....
....
((D4, world), 1)
...
((D4, ?), 1)
((D4, ?), 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
(hello, 1)
(world, 1)
(world, 1)
(?, 1)
Reducer 1
Reducer 2
Reducer 3


D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)
....
....
((D4, world), 1)
...
((D4, ?), 1)
((D4, ?), 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4
(hello, 1)
(world, 1)
(world, 1)
(?, 1)
Reducer 1
Reducer 2
Reducer 3
(hello, 2)
(world, 4)
(?, 1)

(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().ReduceFn1().ReduceFn2();
write(output)
}
```

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: (String, String) -> ((String, String), Int) {
    for w in input.value().split(){
        emit((input.key(), w), 1)
    }
}
def ReduceFn1: ((String, String), Int) -> (String, Int) {
    emit(input.key()[1], 1)
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    sum = 0;
    for (w, c) in input{ sum += c; }
    emit(w, sum);
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().MapFn2().ReduceFn();
write(output)
}
```

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: (String, String) -> ((String, String), Int) {
    for w in input.value().split(){
        emit((input.key(), w), 1)
    }
}           ignore the value list! ("unique")
def ReduceFn1: ((String, String), Int) -> (String, Int) {
    emit(input.key()[1], 1)
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    sum = 0;
    for (w, c) in input{ sum += c; }
    emit(w, sum);
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().MapFn2().ReduceFn();
write(output)
}
```

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
 for w in input.value().split(){
  emit((input.key(), w), 1)
 }
}
def ReduceFn1: {
 emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2:{
 sum = 0;
 for (w, c) in input{ sum += c; }
 emit(w, sum);
}
```

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
 for w in input.value().split(){
  emit((input.key(), w), 1)
 }
}
def ReduceFn1: {
 emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2:{
 sum = 0;
 for (w, c) in input{ sum += c; }
 emit(w, sum);
}
```

```
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
 for w in input.value().split(){
  emit(input.key(), w)
 }
}
def ReduceFn1: {
 for w in input.value(){emit(w, 1)}
}
// sum the total counts
// of each word
def ReduceFn2:(S, I) -> (S, I){
 sum = 0;
 for (w, c) in input{ sum += c; }
 emit(w, sum);
}
```

(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
 for w in input.value().split(){
  emit((input.key(), w), 1)
 }
}
def ReduceFn1: {
 emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2:{
 sum = 0;
 for (w, c) in input{ sum += c; }
 emit(w, sum);
}
```

```
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
 for w in input.value().split(){
  emit(input.key(), w)
 }
}
def ReduceFn1: {
 for w in input.value(){emit(w, 1)}
}
// sum the total counts
// of each word
def ReduceFn2:(S, I) -> (S, I){
 sum = 0;
 for (w, c) in input{ sum += c; }
 emit(w, sum);
}
```

Do these produce the same output?

(a)Yes **(b) No**

Clicker Question!

```
Input K: V
Doc1 : here are some words
Doc2: words words words
Doc3: here are words
```

```
def MapFn1: (S, S) -> (S, S) {
    for w in input.value().split(){
        emit(input.key(), w)
    }
}
```

```
def ReduceFn1: (S, S) -> (S, I) {
    for w in input.value(){
        emit(w, 1)
    }
}

def ReduceFn2:(S, I) -> (S, I){
    sum = 0;
    for (w, c) in input{
       sum += c;
    }
    emit(w, sum);
}
```

What will this produce?

(a) here:2, are:2, some:1, words:3
(b) here:2, are:2, some:1, words:5
(c) here:1, are:1, some:1, words:1

Clicker Question!

```
Input K: V
Doc1 : here are some words
Doc2: words words words
Doc3: here are words
```

```
def MapFn1: (S, S) -> (S, S) {
    for w in input.value().split(){
        emit(input.key(), w)
    }
}
```

```
def ReduceFn1: (S, S) -> (S, I) {
    for w in input.value(){
        emit(w, 1)
    }
}

def ReduceFn2:(S, I) -> (S, I){
    sum = 0;
    for (w, c) in input{
        sum += c;
    }
    emit(w, sum);
}
```

What will this produce?

(a) here:2, are:2, some:1, words:3

[✓ (b)] here:2, are:1, some:1, words:5

(c) here:1, are:1, some:1, words:1

Clicker Question!

```
Input K: V
Doc1 : here are some words
Doc2: words words words
Doc3: here are words
```

```
def ReduceFn1: (S, S) -> (S, I) {
     for w in input.value(){
          emit(w, 1)
     }
}

def ReduceFn2:(S, I) -> (S, I){
     sum = 0;
     for (w, c) in input{
         sum += c;
     }
     emit(w, sum);
}
```

```
def MapFn1: (S, S) -> (S, S) {
     for w in input.value().split(){
          emit(input.key(), w)
     }
}
```

Reducer is by DocId only

What will this produce?

(a) here:2, are:2, some:1, words:3

[✓ (b)] here:2, are:1, some:1, words:5

(c) here:1, are:1, some:1, words:1

Other MapReduce Functions

- Sort
- Unique
- Sample
- First
- Filter
- Join

Other MapReduce Functions

- Sort
- Unique
- Sample
- First
- Filter
- Join

Joins

Joins

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)

Joins

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)
- But you can imagine having to do them yourself…

💃🕺 Hacky Joins 💃🕺

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)
- But you can imagine having to do them yourself…
- …or, if you aren’t that imaginative type, you can just look at the homework

💃🕺 Hacky Joins 💃🕺

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)
- But you can imagine having to do them yourself…
- …or, if you aren’t that imaginative type, you can just look at the homework
- (sry)

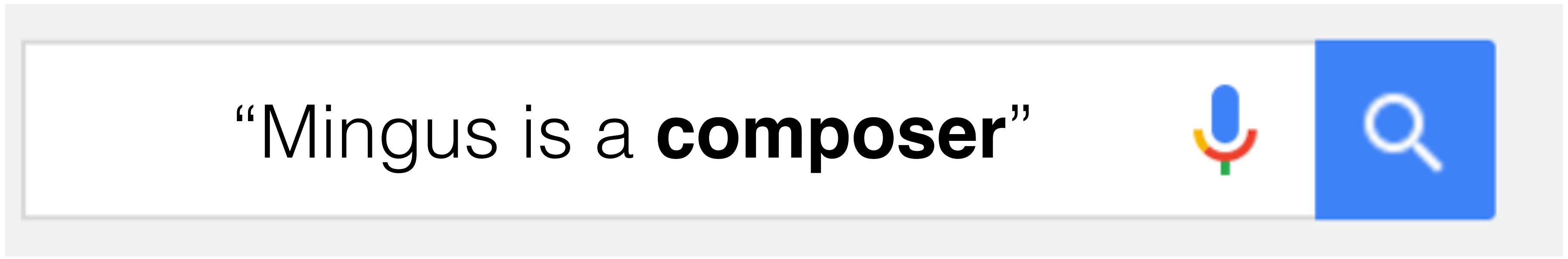
Real Life Application

Real Life Application

Is Charles Mingus a **composer**?

Real Life Application

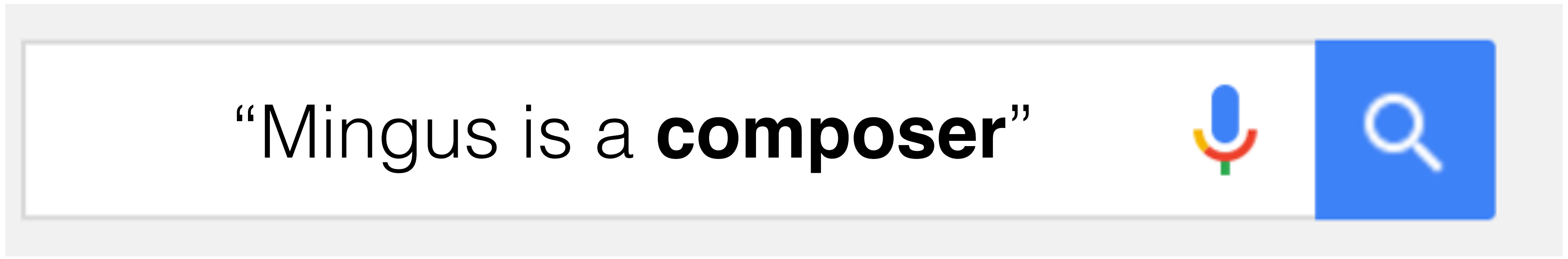
Is Charles Mingus a **composer**?



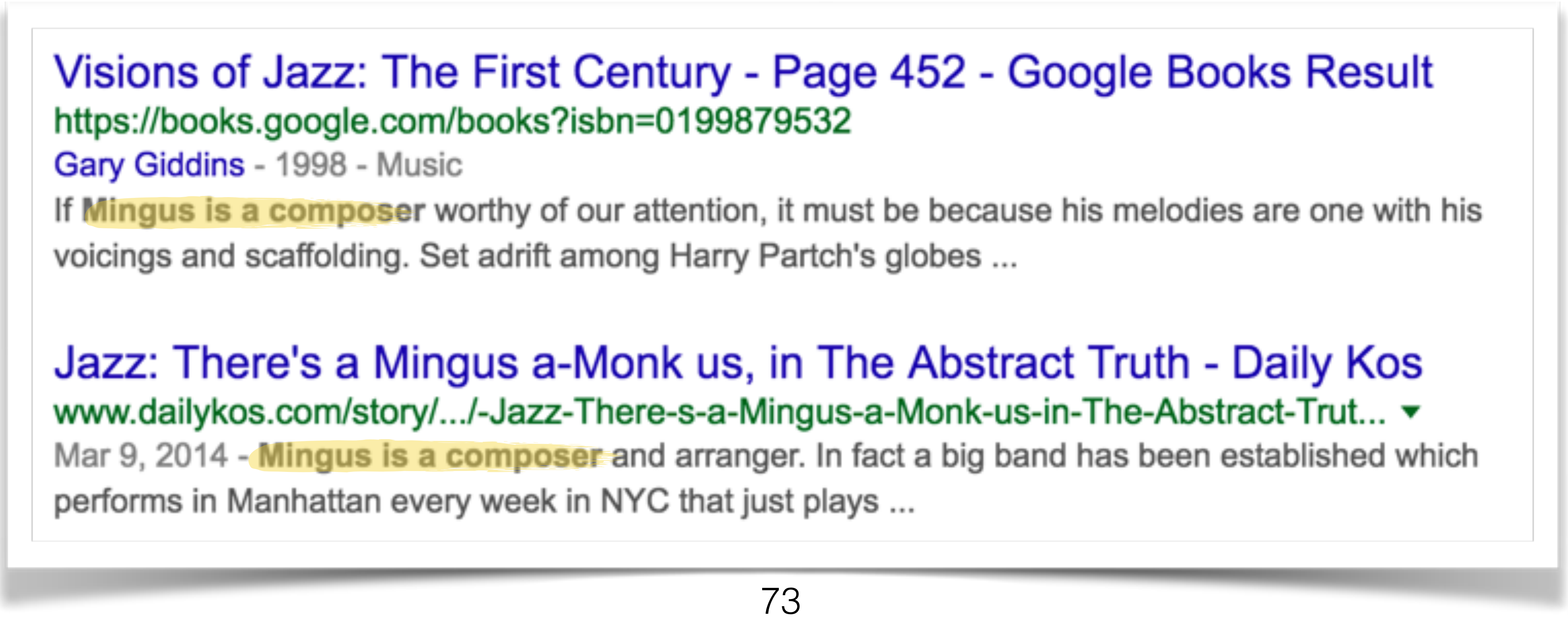
“Mingus is a **composer**”

Real Life Application

Is Charles Mingus a **composer**?



“Mingus is a **composer**”

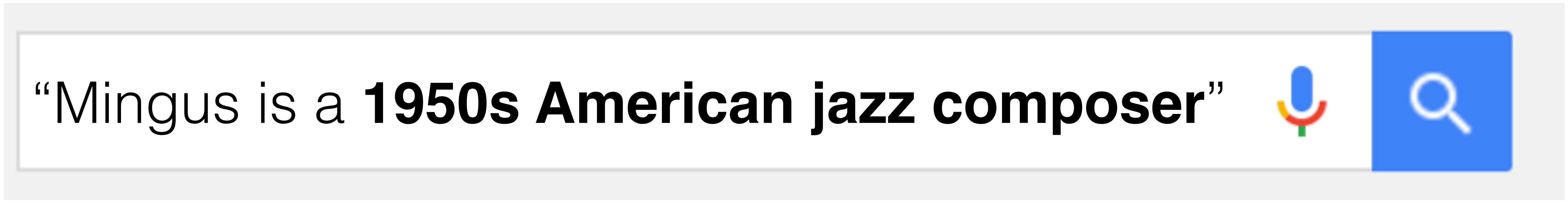


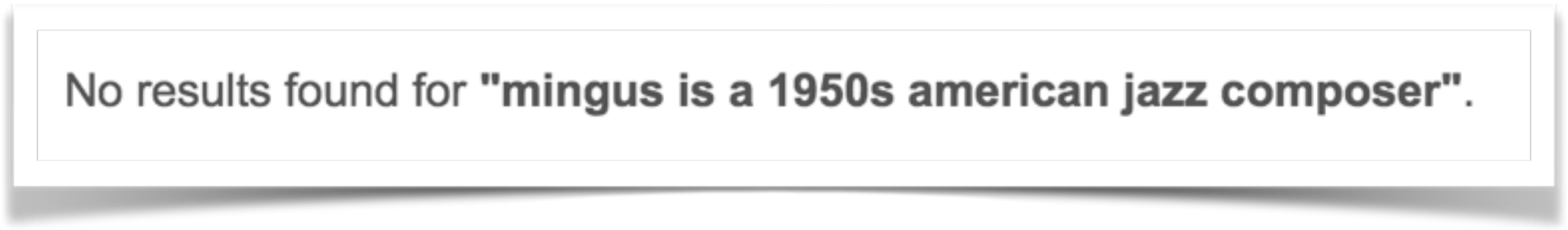
Visions of Jazz: The First Century - Page 452 - Google Books Result
https://books.google.com/books?isbn=0199879532
Gary Giddins - 1998 - Music
If Mingus is a composer worthy of our attention, it must be because his melodies are one with his voicings and scaffolding. Set adrift among Harry Partch's globes ...

Jazz: There's a Mingus a-Monk us, in The Abstract Truth - Daily Kos
www.dailykos.com/story/.../-Jazz-There-s-a-Mingus-a-Monk-us-in-The-Abstract-Trut...
Mar 9, 2014 - Mingus is a composer and arranger. In fact a big band has been established which performs in Manhattan every week in NYC that just plays ...

Real Life Application

Is Charles Mingus a **1950s American jazz composer**?


“Mingus is a **1950s American jazz composer**”


No results found for **"mingus is a 1950s american jazz composer"**.

Real Life Application

Is Charles Mingus a **1950s American jazz composer**?

Real Life Application

Is Charles Mingus a **1950s American jazz composer**?

… if **Mingus is a composer** worthy of our attention, it must be because…

Mingus dominated the scene back in the 1950s and 1960s.

Mingus was truly a product of America in all its historic complexities…

A virtuoso bassist and composer, **Mingus** irrevocably **changed the face of jazz**…

Real Life Application

ComposerX dominated the scene back in the 1950s and 1960s.

$\downarrow$

ComposerX is a **1950s composer.**

Real Life Application

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

Joins

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

Desired output:

Subject	Predicate	Object	Categories
Barack Obama	won	the electoral vote	Person, US_Presidents, Huffington_Post_Columnists
Kamala Lopez	wrote	an op-ed for HuffPo	Person, Huffington_Post_Columnists,

Joins

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

Desired output:

Subject	Predicate	Object	Categories
Barack Obama	won	the electoral vote	Person, US_Presidents, Huffington Post Columnists
Kamala Lopez	wrote	an op-ed for HuffPo	Person, Huffington_Post_Columnists,

Joins

Facts

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Categories

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

```
Select * from Facts, Categories
Where Subject == Entity
```

Joins

Facts

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Categories

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

```
Select * from Facts, Categories
Where Subject == Entity
GroupBy Subject
```

Joins

Facts

Subject	Predicate	Object
Barack Obama	won	the electoral vote
Kamala Lopez	wrote	an op-ed for HuffPo
Charles Mingus	wrote	jazz
Barack Obama	opposed	the appropriations bill
Barack Obama	listens to	jazz

Categories

Category	Entity
Person	Barack Obama
Person	Kamala Lopez
Person	Charles Mingus
Huffington Post Columnists	Barack Obama
Huffington Post Columnists	Kamala Lopez
US Presidents	Barack Obama
Jazz Composers	Charles Mingus

```
Select * from Facts, Categories
Where Subject == Entity
GroupBy Subject
```

$\downarrow$

```
                  Key: String
Value: (list_of((String, String, String), list_of((String, String))
```

DIY Joins

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Obj) -> (String, Obj){






}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){








}
```

DIY Joins

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
         emit(v.Subject, v)
    } else {
         emit(v.Entity, v)
    }
}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){









}
```

DIY Joins

Facts

Subject	Predicate	Object

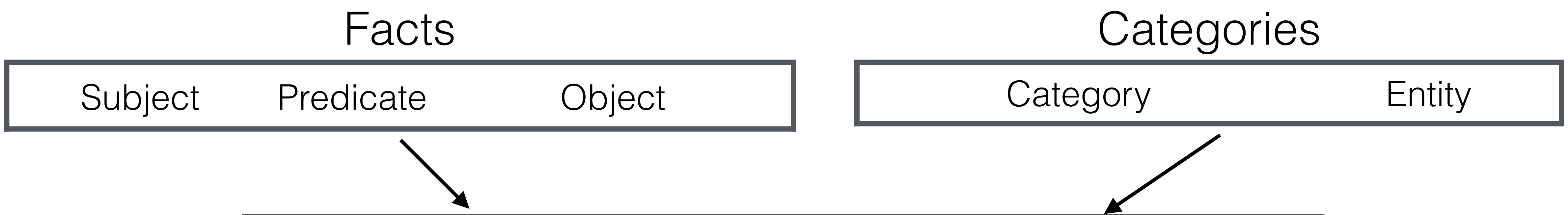
Categories

Category	Entity

```
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
         emit(v.Subject, v)
    } else {
         emit(v.Entity, v)
    }
}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){






    }

}
```

DIY Joins



```
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
         emit(v.Subject, v)
    } else {
         emit(v.Entity, v)
    }
}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (typeof(v) == Fact) {
             all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    }

}
```

DIY Joins

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
         emit(v.Subject, v)
    } else {
         emit(v.Entity, v)
    }
}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (typeof(v) == Fact) {
             all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
```

DIY Joins

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
         emit(v.Subject, v)
    } else {
         emit(v.Entity, v)
    }
}
```

```
def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (typeof(v) == Fact) {
             all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
```

💃🕺 Hacky Joins 💃🕺

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Tuple) -> (String, Tuple){
    v = input.value()
    if (len(v) == 3) {
         emit(v[0], v)
    } else {
         emit(v[1], v)
    }
}
```

```
def ReduceFn: (String, Tuple) -> (Tuple, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (len(v) == 3) {
             all_facts.append(v)
        } else {
            all_cats.append(v[0])
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
```

Hacky Joins

Facts

Subject	Predicate	Object

Categories

Category	Entity

```
def MapFn: (String, Tuple) -> (String, Tuple){
    v = input.value()
    if (len(v) == 3) {
        emit(v[0], v)
    } else {
        emit(v[1], v)
    }
}
```

<blink>

This is not a thing you ever actually do! Please do not do this!

</blink>

```
def ReduceFn: (String, Tuple) -> (Tuple, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (len(v) == 3) {
            all_facts.append(v)
        } else {
            all_cats.append(v[0])
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
```

💃🕺 Hacky Joins 💃🕺

Facts

Subject	Predicate	Object

Categories

Category	Entity

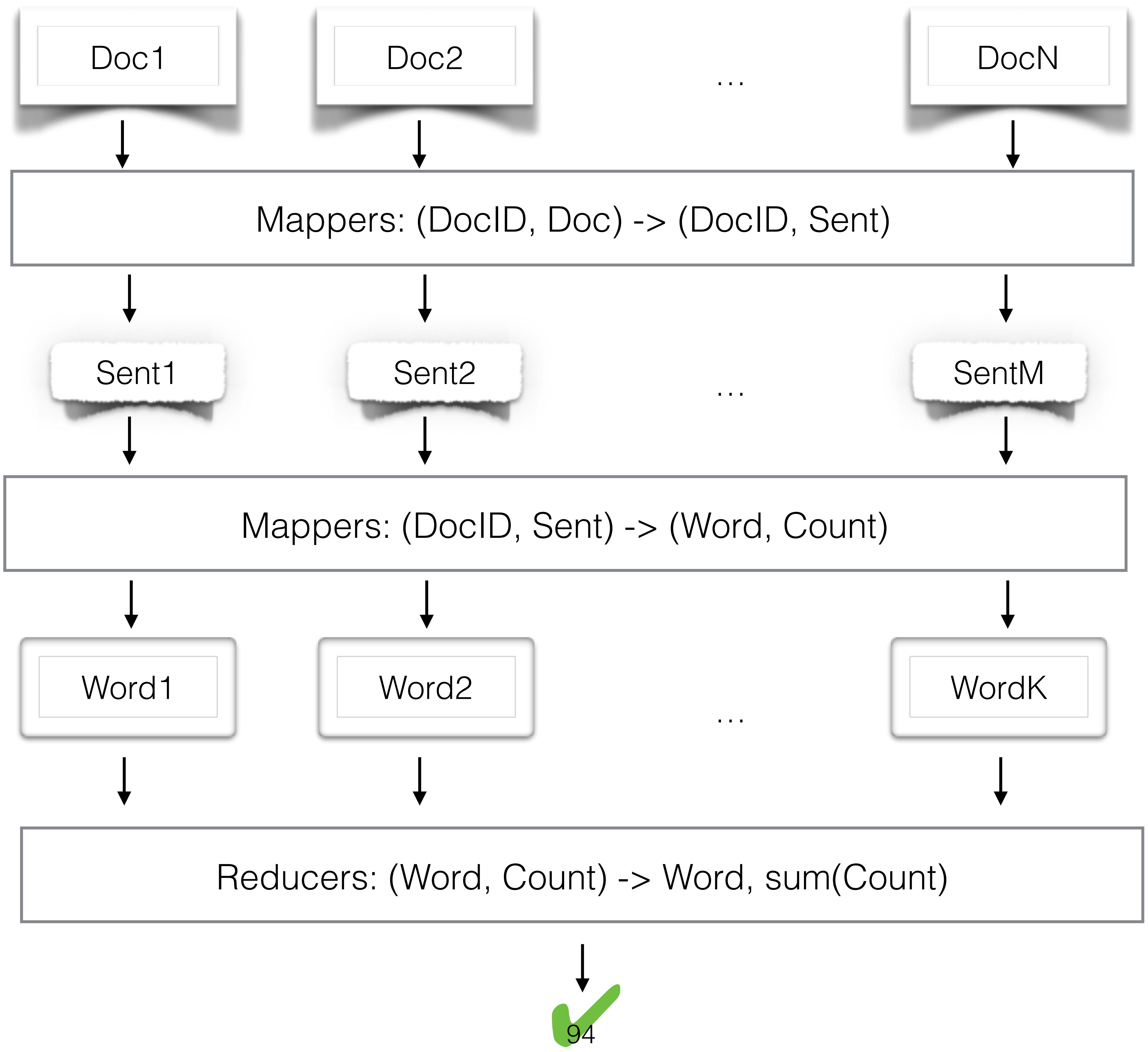
```
def MapFn: (String, Tuple) -> (String, Tuple){
    v = input.value()
    if (len(v) == 3) {
        emit(v[0], v)
    } else {
        emit(v[1], v)
    }
}
```

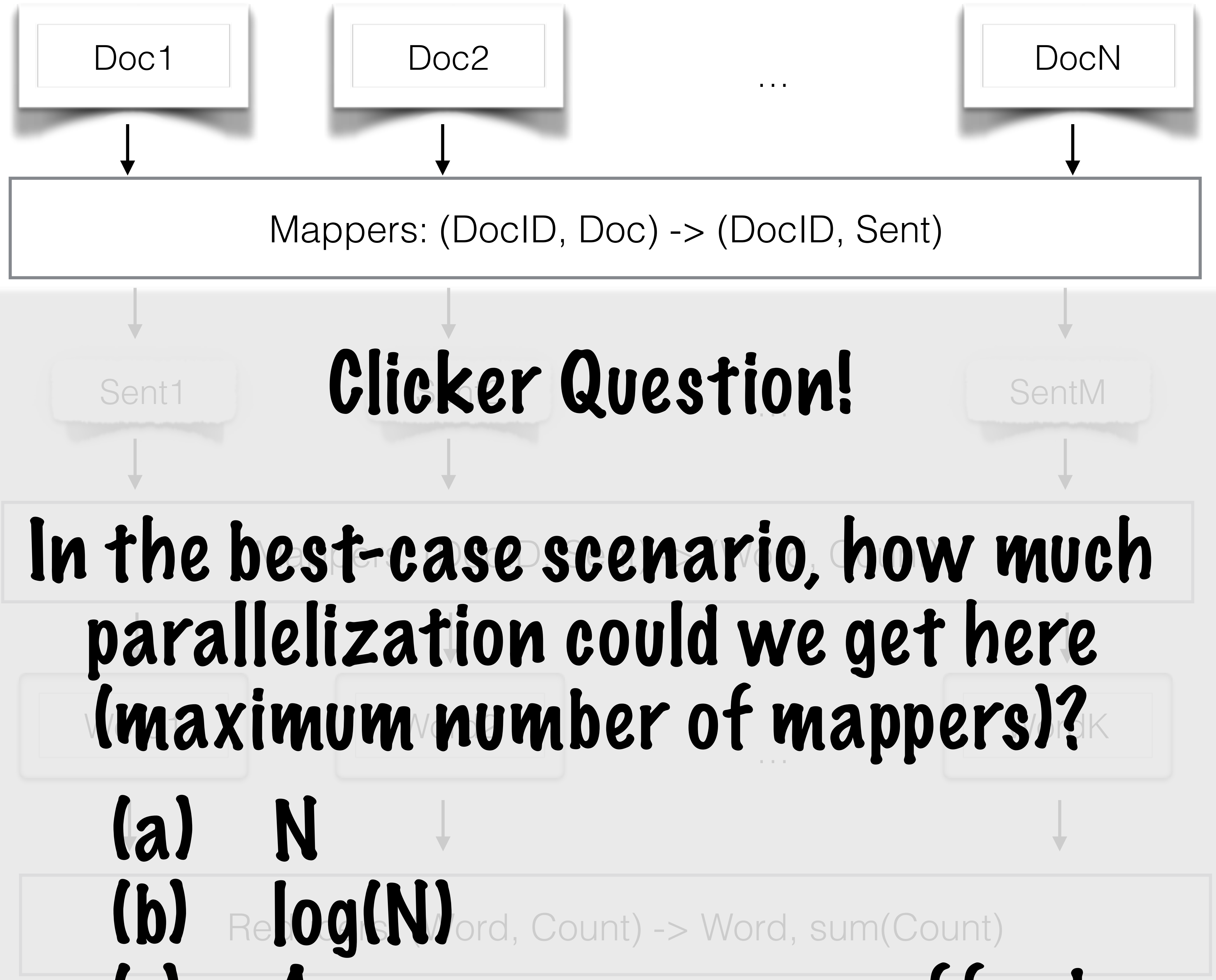
<blink>
This is not a thing you ever actually do! Please do not do this!
</blink>

(But do it for the homework, and I will never tell, and we will never speak of it, and if someone asks you in an interview if you are the kind of person who would do this in a map reduce, you will deny deny deny. Agreed?)

```
def ReduceFn: (String, Tuple) -> (Tuple, List[String]){
    all_cats = []; all_facts = []
    for v in input.value(){
        if (len(v) == 3) {
            all_facts.append(v)
        } else {
            all_cats.append(v[0])
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
```

Bottlenecks!

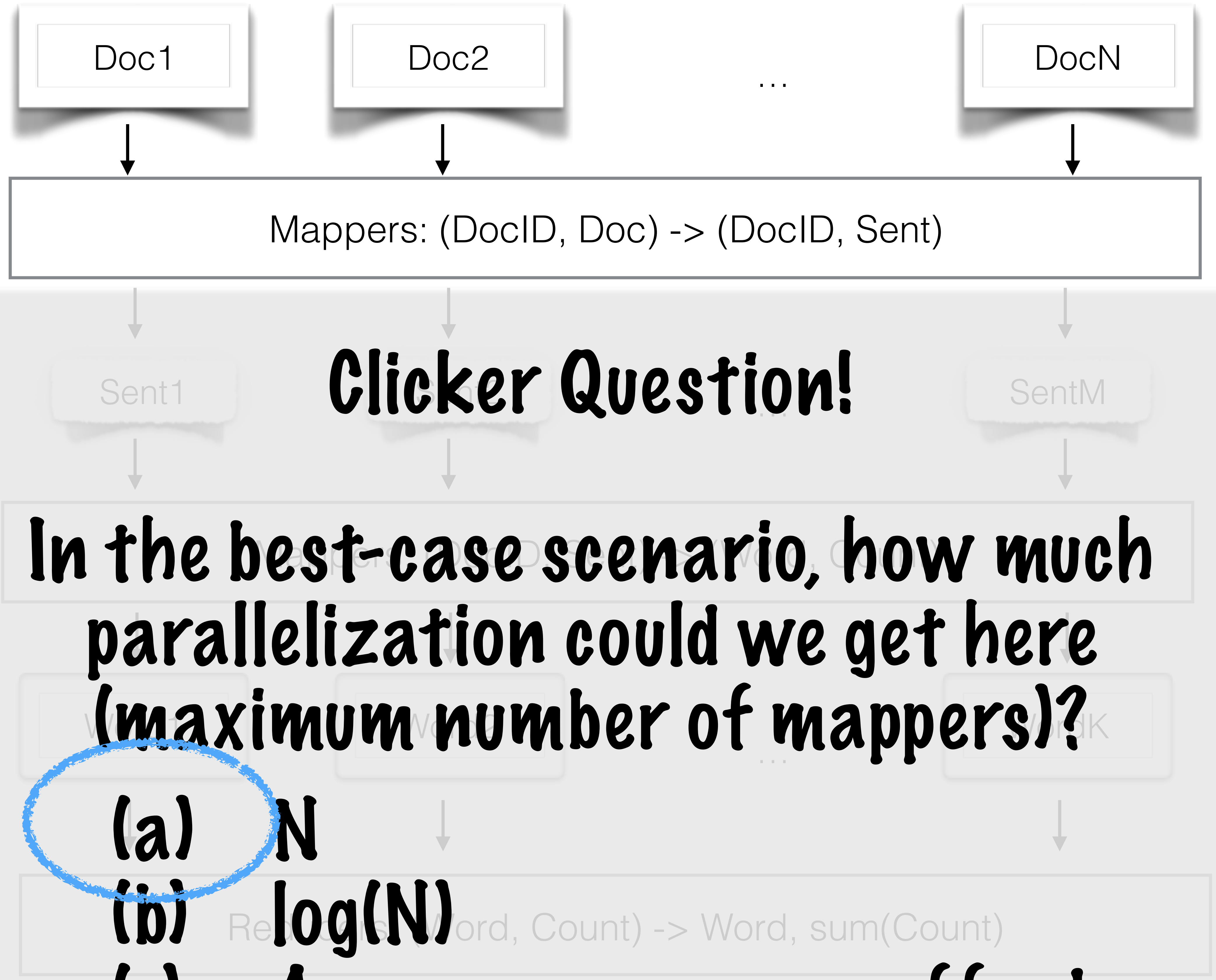

Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
Sent2
...
SentM
Mappers: (DocID, Sent) -> (Word, Count)
Word1
Word2
...
WordK
Reducers: (Word, Count) -> Word, sum(Count)



Clicker Question!

In the best-case scenario, how much parallelization could we get here (maximum number of mappers)?

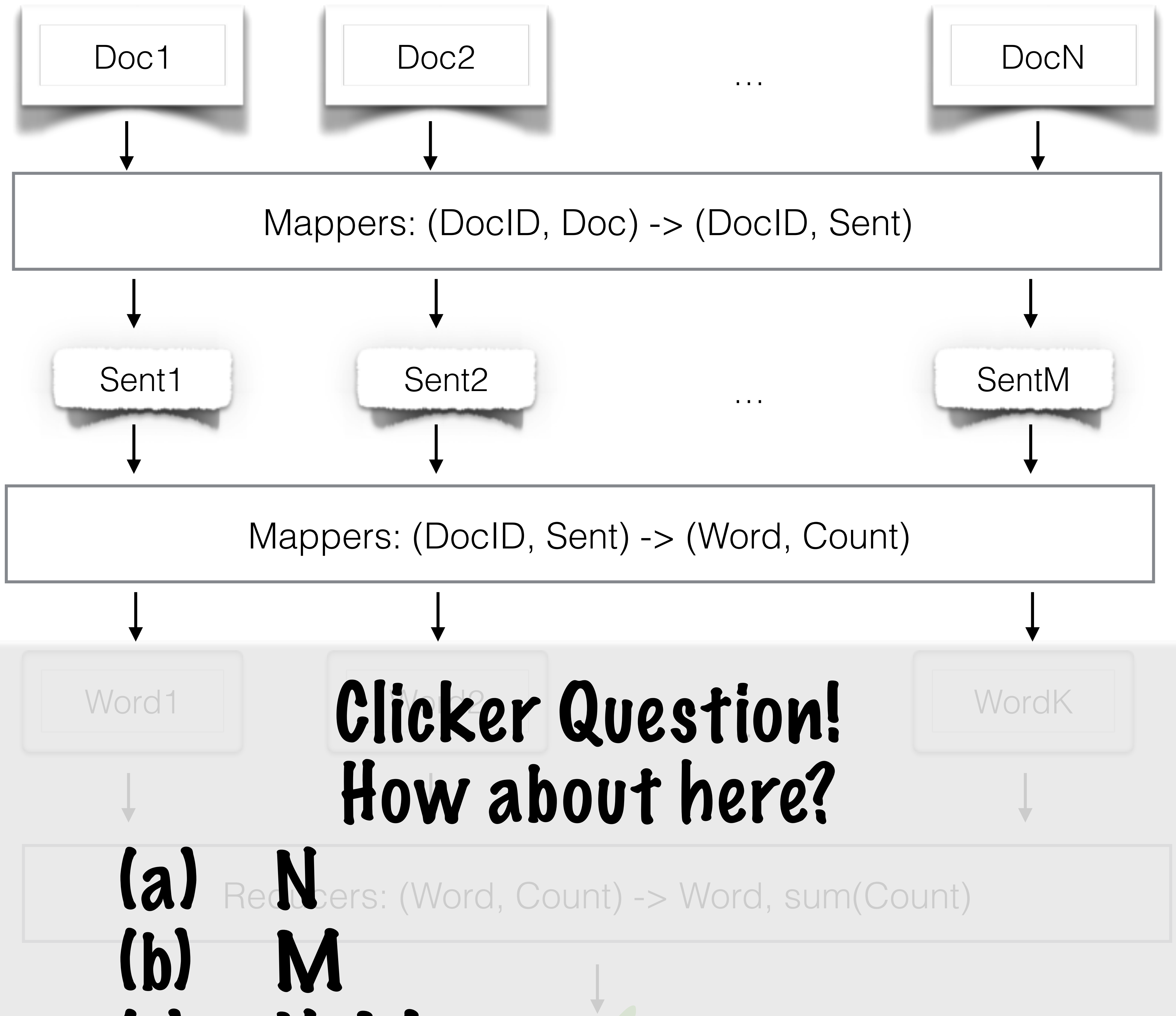
(a) N
(b) log(N)
[✓ (c) As many as we can afford.]


Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
SentM
WordK
Reducers: (Word, Count) -> Word, sum(Count)

Clicker Question!

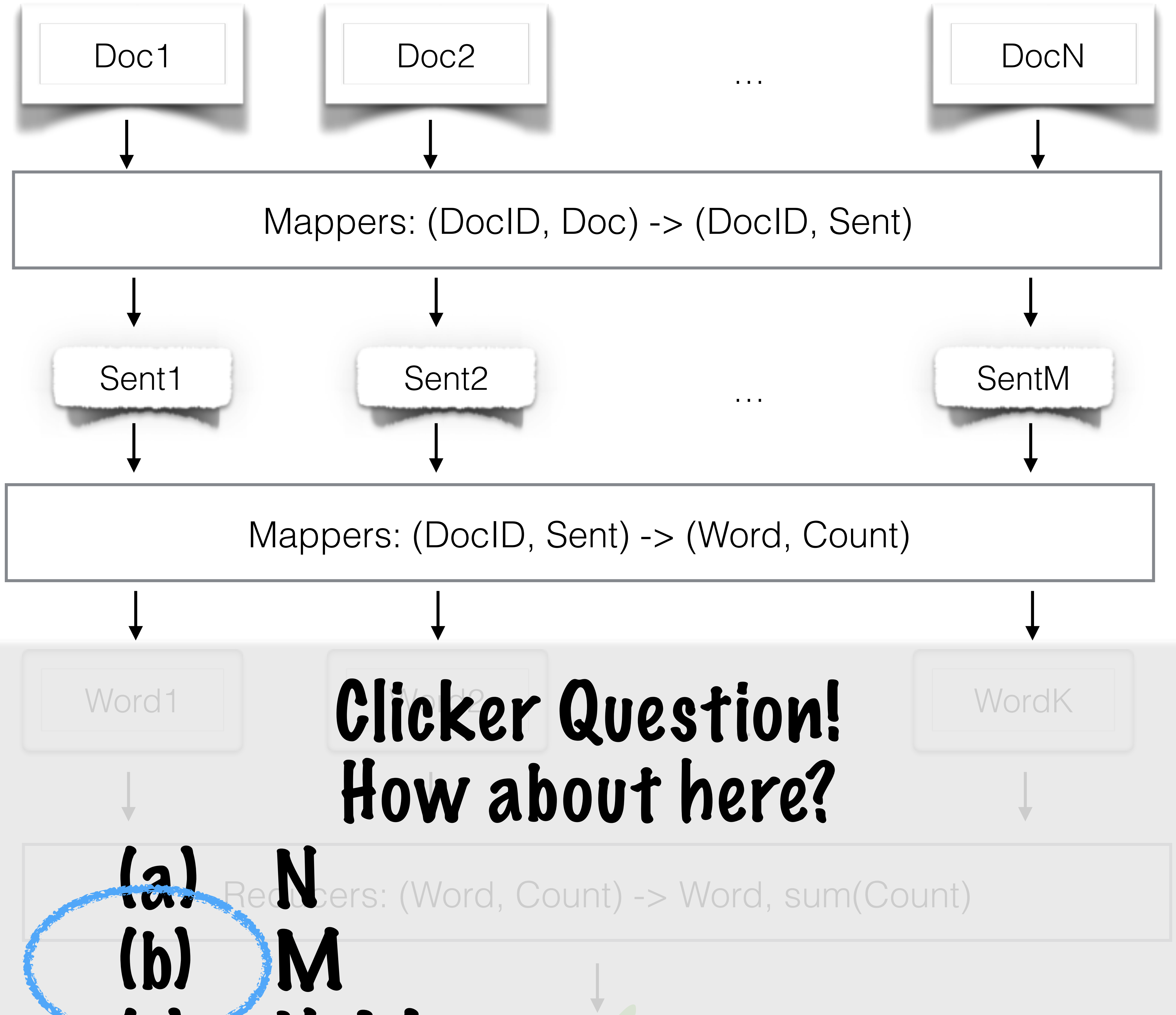
In the best-case scenario, how much parallelization could we get here (maximum number of mappers)?

[✓ (a)] N
(b) log(N)
(c) As many as we can afford.


Doc1
Doc2
…
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
Sent2
…
SentM
Mappers: (DocID, Sent) -> (Word, Count)
Word1
Word2
WordK
Reducers: (Word, Count) -> Word, sum(Count)

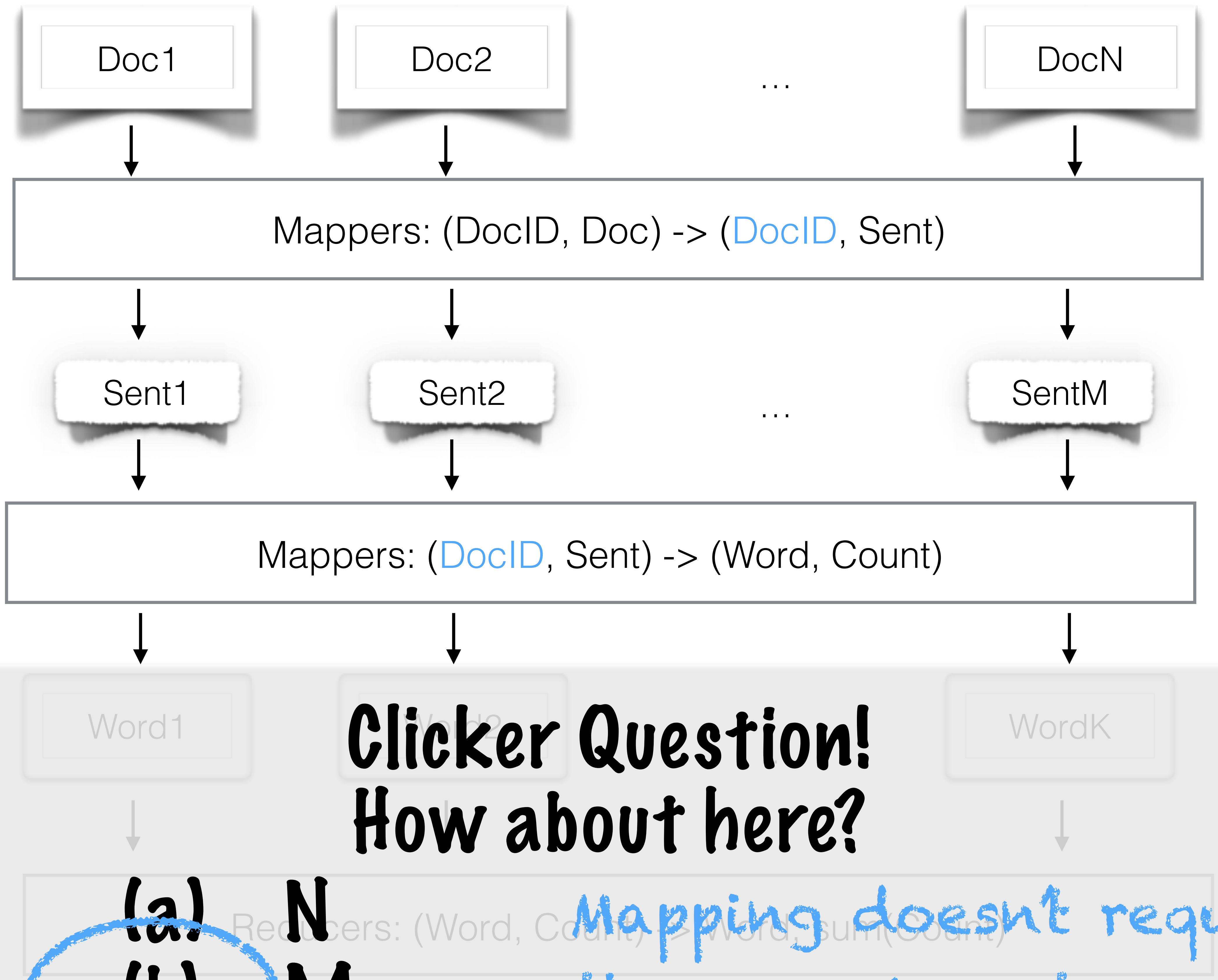
Clicker Question!
How about here?

(a) N
(b) M
(c) N*M


Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
Sent2
...
SentM
Mappers: (DocID, Sent) -> (Word, Count)
Word1
Word2
WordK
Reducers: (Word, Count) -> Word, sum(Count)

Clicker Question!
How about here?

(a) N
[✓ (b)] M
(c) N*M


Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
Sent2
...
SentM
Mappers: (DocID, Sent) -> (Word, Count)
Word1
Word2
WordK
Reducers: (Word, Count) -> (Word, sum(Count))

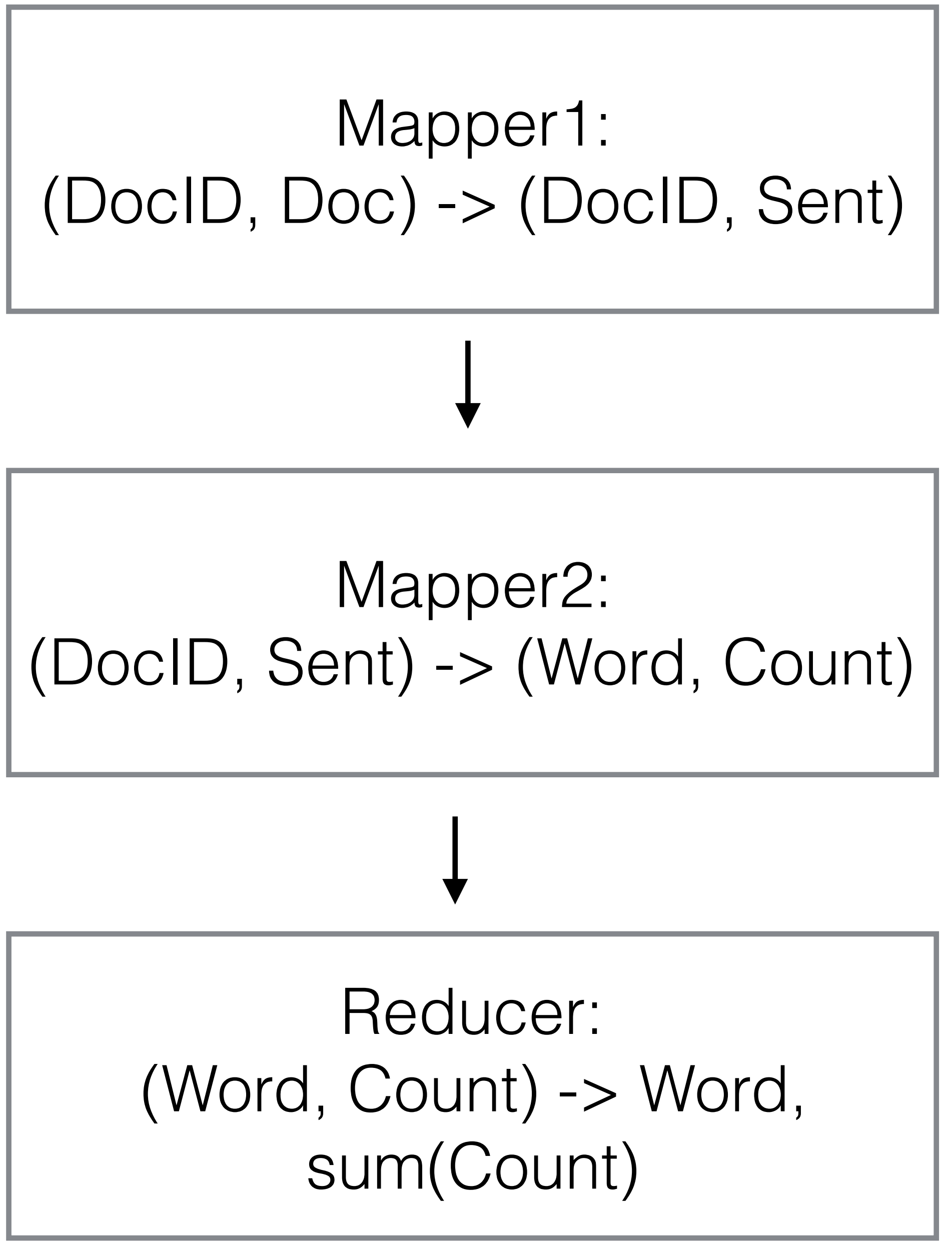
Clicker Question!
How about here?

(a) N
[✓ (b) M]
(c) N*M

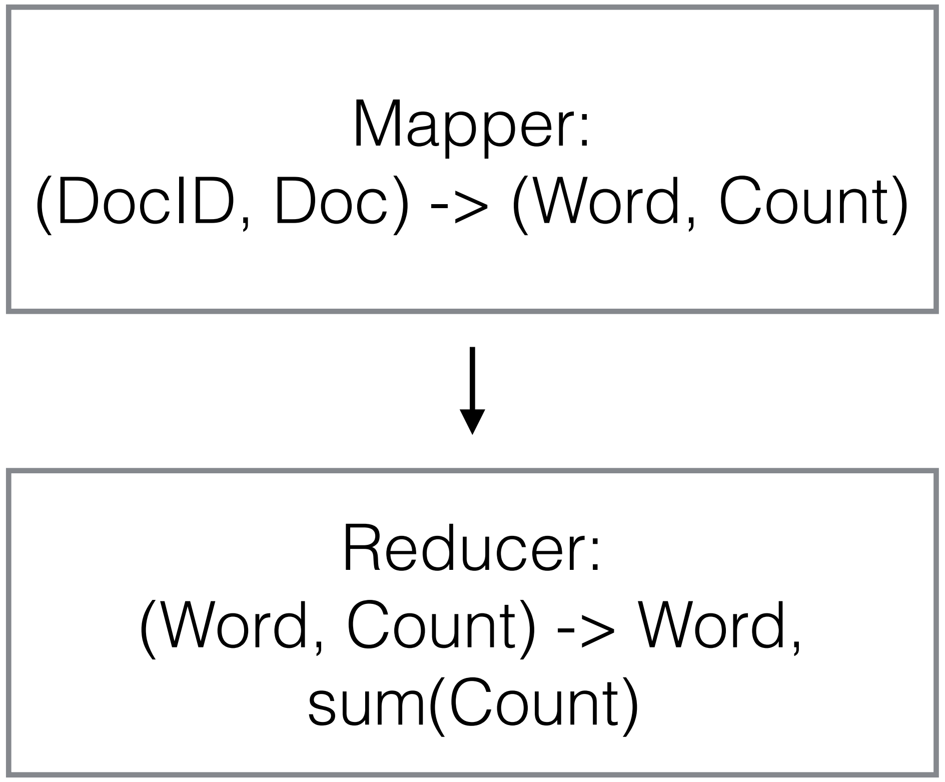
Mapping doesn't require the same keys to route to the same machine.

Clicker Question!
Which is (likely to be) faster?

(a)


Mapper1:
(DocID, Doc) -> (DocID, Sent)
Mapper2:
(DocID, Sent) -> (Word, Count)
Reducer:
(Word, Count) -> Word, sum(Count)

(b)


Mapper:
(DocID, Doc) -> (Word, Count)
Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same

```
Doc = list_of(Sentence)
Sentence = list_of(Word)
```

er Question!
hich is (likely to be) faster?

(a)

Mapper1:
(DocID, Doc) -> (DocID, Sent)

↓

Mapper2:
(DocID, Sent) -> (Word, Count)

↓

Reducer:
(Word, Count) -> Word, sum(Count)

(b)

Mapper:
(DocID, Doc) -> (Word, Count)

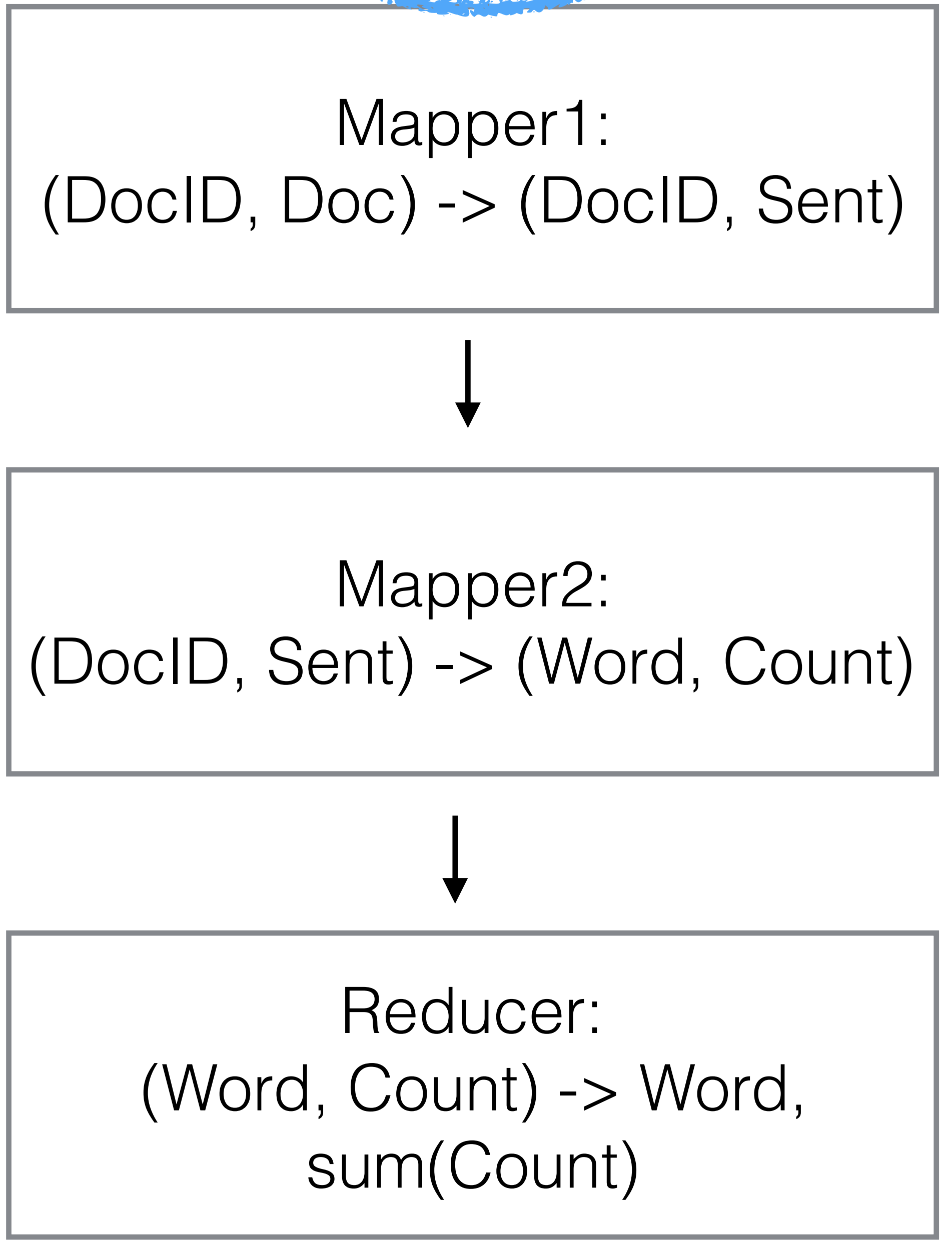
↓

Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same

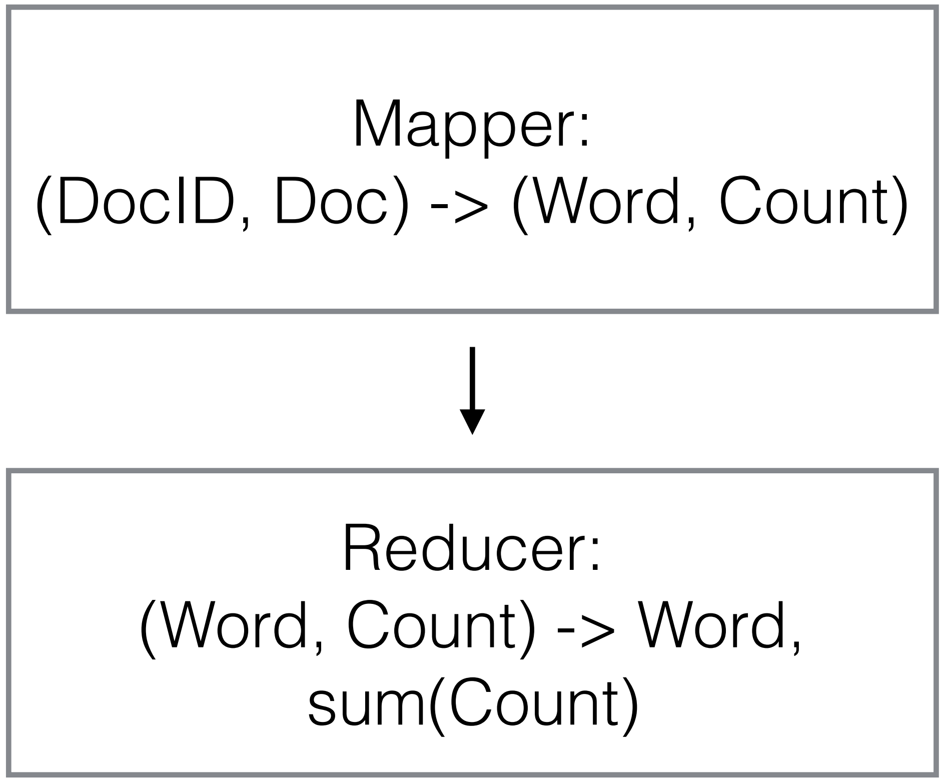
Clicker Question!
Which is (likely to be) faster?

[✓ (a)]



Mapper1:
(DocID, Doc) -> (DocID, Sent)
Mapper2:
(DocID, Sent) -> (Word, Count)
Reducer:
(Word, Count) -> Word, sum(Count)

(b)



Mapper:
(DocID, Doc) -> (Word, Count)
Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same

Clicker Question!
Which is (likely to be) faster?

[✓ (a)]

Mapper1:
(DocID, Doc) -> (DocID, Sent)

Ma
(DocID, Sent)

↓

Reducer:
(Word, Count) -> Word, sum(Count)

(b)

Mapper:
(DocID, Doc) -> (Word, Count)

Word,
t)

Smaller jobs = more dynamic load balancing and faster recovery from failure

(c) They are the same

Clicker Question!
Which is (likely to be) faster?

[✓ (a)]

Mapper1:
(DocID, Doc) -> (DocID, Sent)

↓

In general, nested loops should be refactored into multiple mappers

sum(Count)

(b)

Mapper:
(DocID, Doc) -> (Word, Count)

```
for sentence in doc:
    for word in sentence:
        blah blah
```

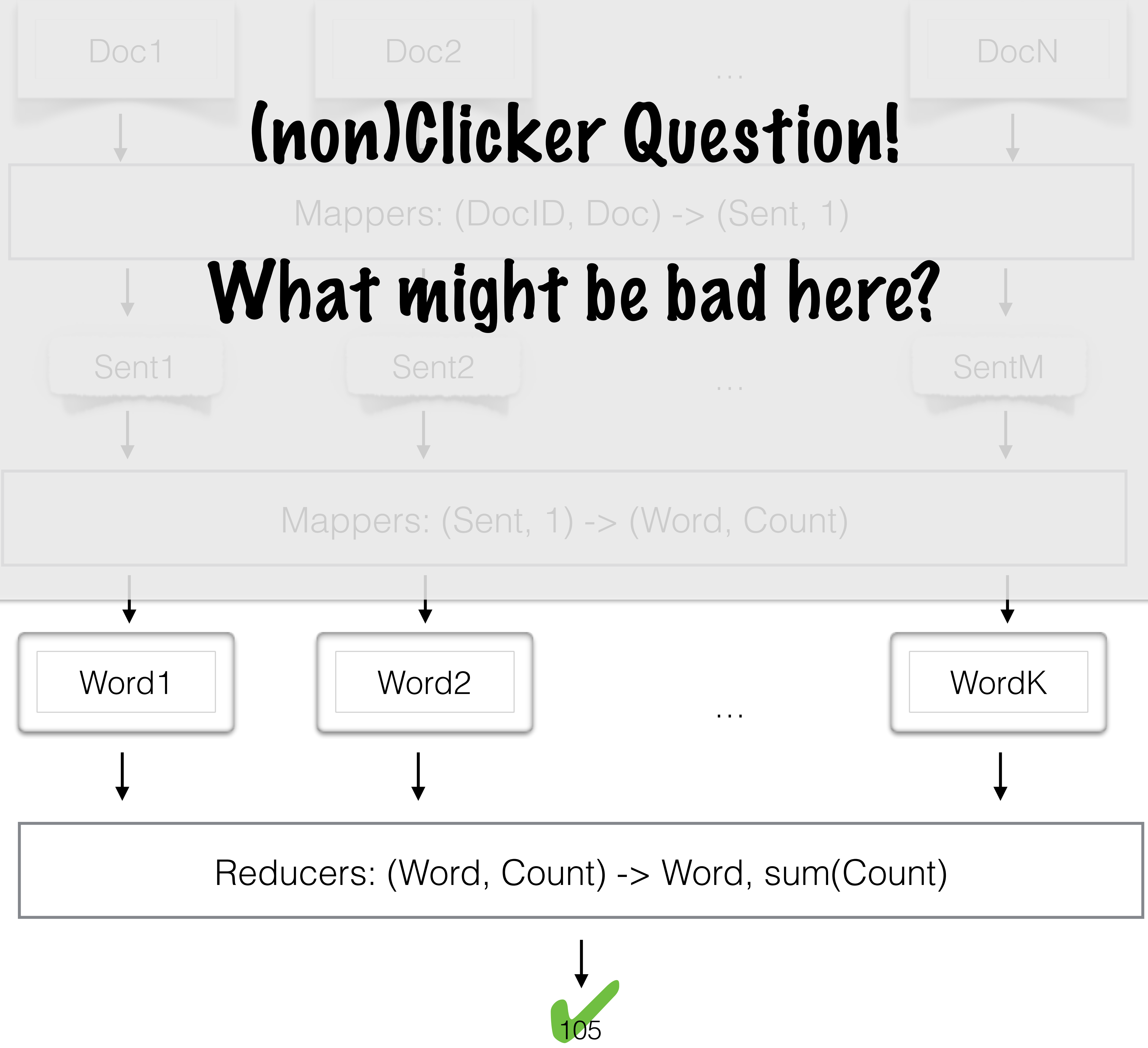
↓

Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same

(non)Clicker Question!

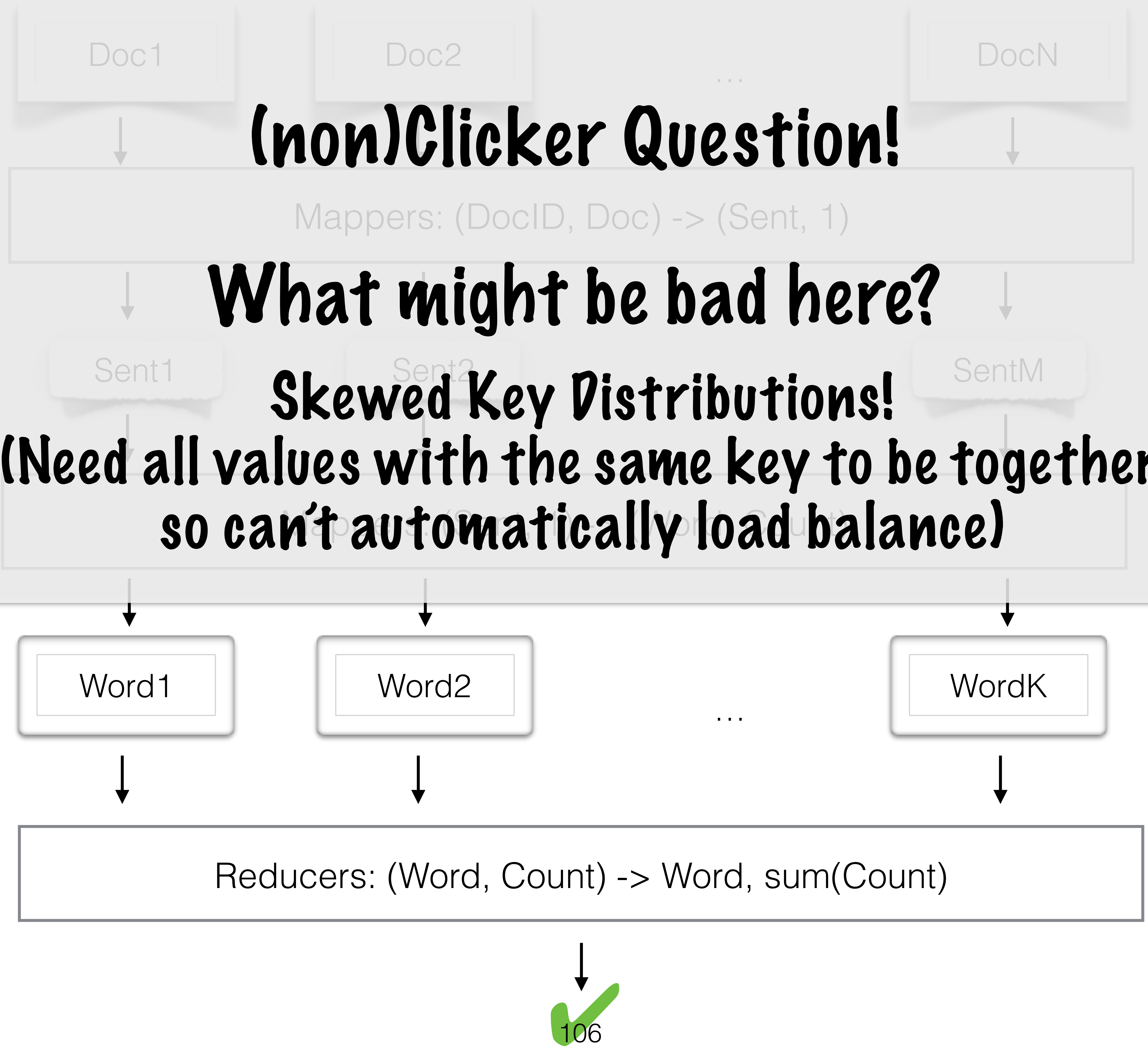
What might be bad here?


Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (Sent, 1)
Sent1
Sent2
...
SentM
Mappers: (Sent, 1) -> (Word, Count)
Word1
Word2
...
WordK
Reducers: (Word, Count) -> Word, sum(Count)

(non)Clicker Question!

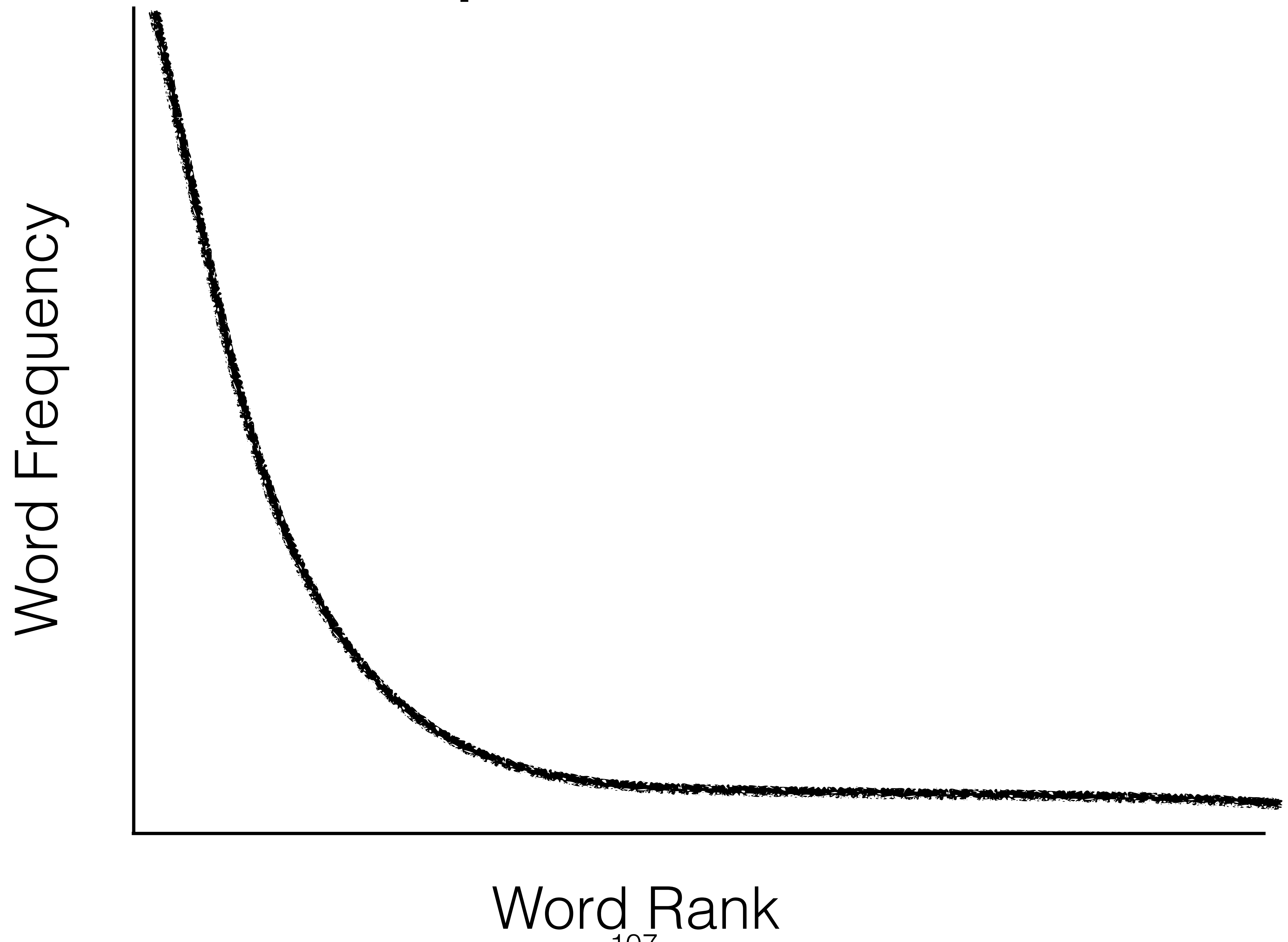
What might be bad here?

Skewed Key Distributions!
(Need all values with the same key to be together, so can't automatically load balance)



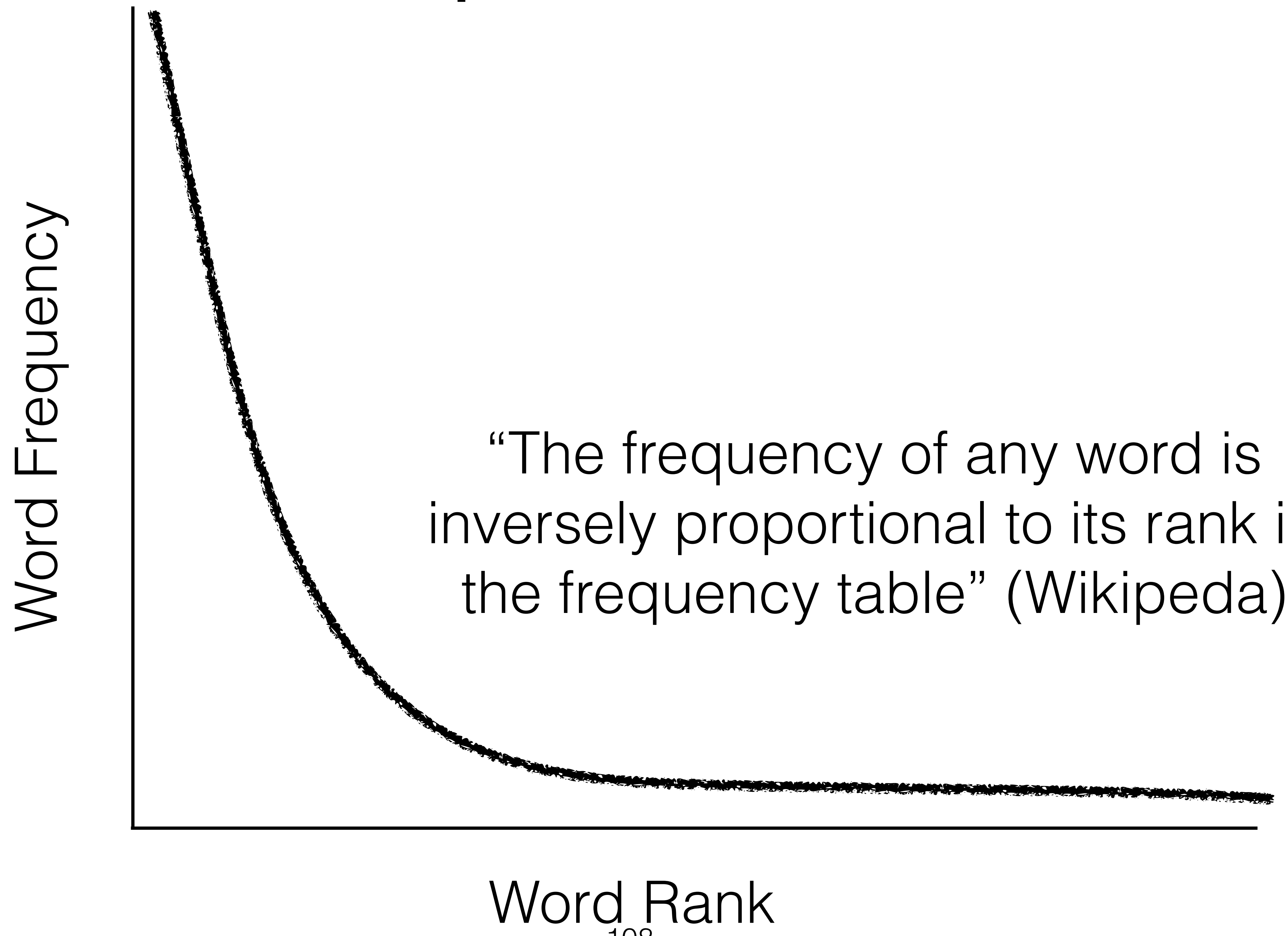
Doc1
Doc2
...
DocN
Mappers: (DocID, Doc) -> (Sent, 1)
Sent1
Sent2
SentM
Mappers: (Sent, 1) -> (Word, Count)
Word1
Word2
...
WordK
Reducers: (Word, Count) -> Word, sum(Count)

Zipf’s Law



https://en.wikipedia.org/wiki/Zipf%27s_law

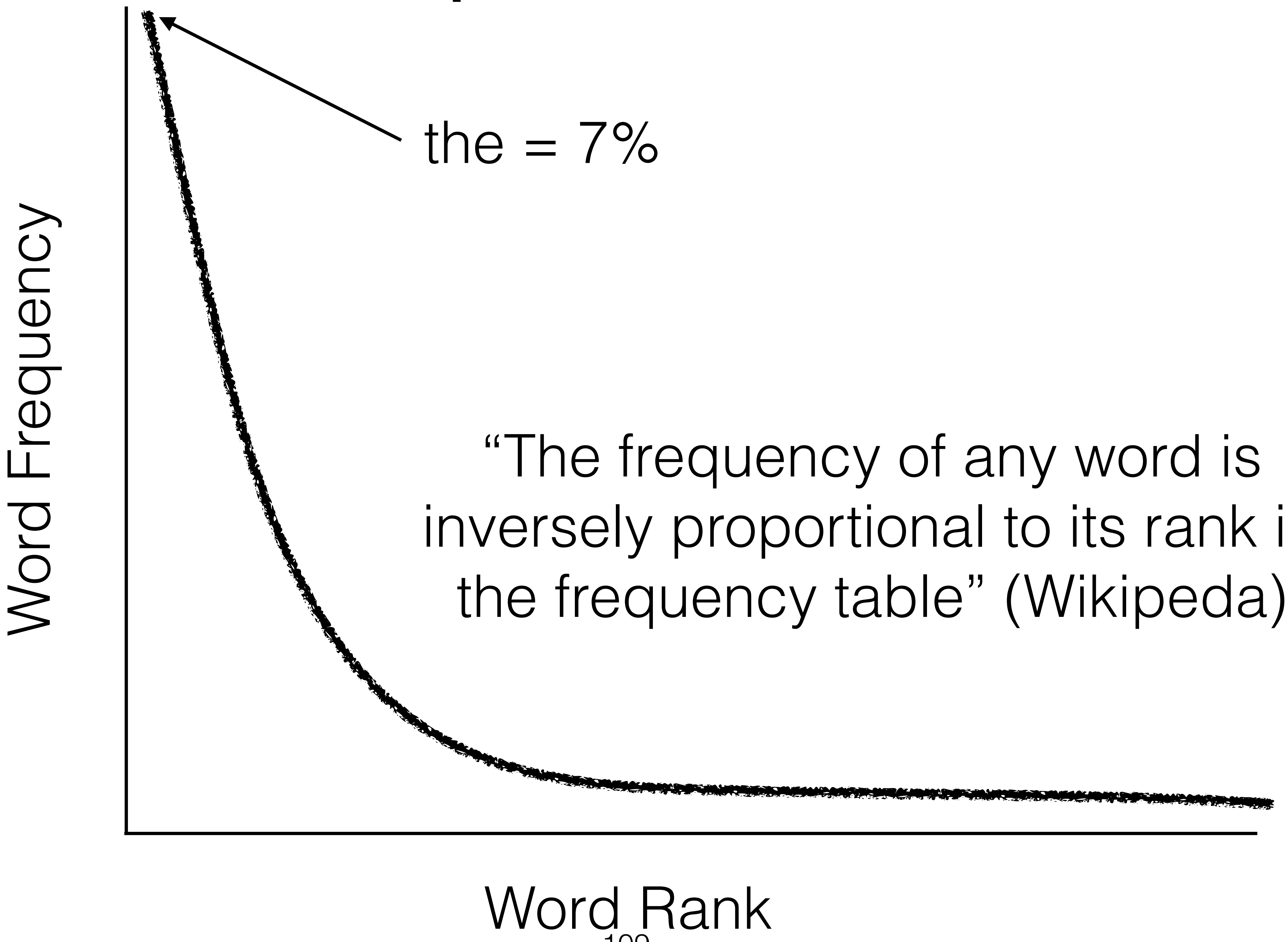
Zipf’s Law



“The frequency of any word is inversely proportional to its rank in the frequency table” (Wikipeda)

https://en.wikipedia.org/wiki/Zipf%27s_law

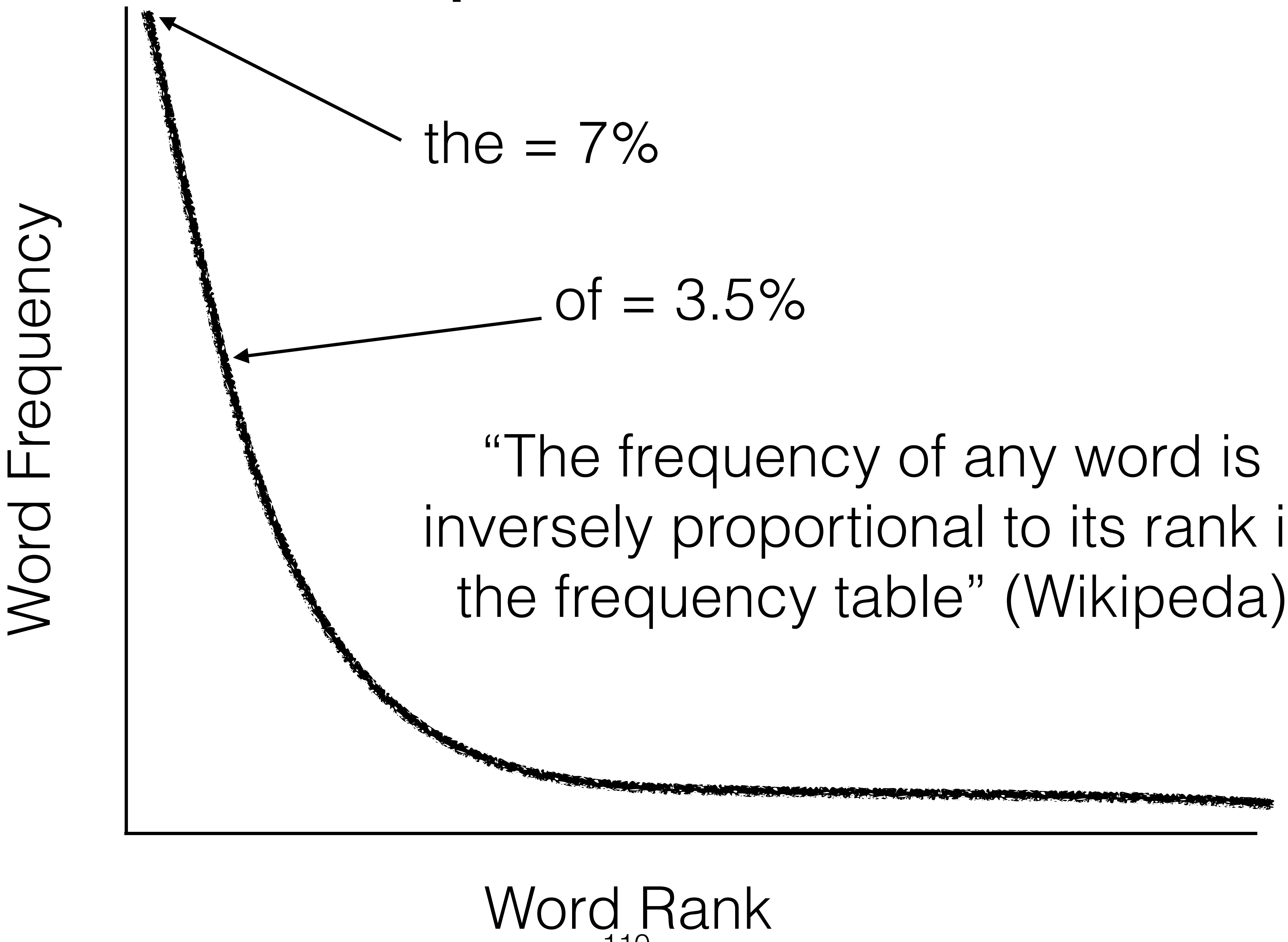
Zipf's Law



"The frequency of any word is inversely proportional to its rank in the frequency table" (Wikipeda)

https://en.wikipedia.org/wiki/Zipf%27s_law

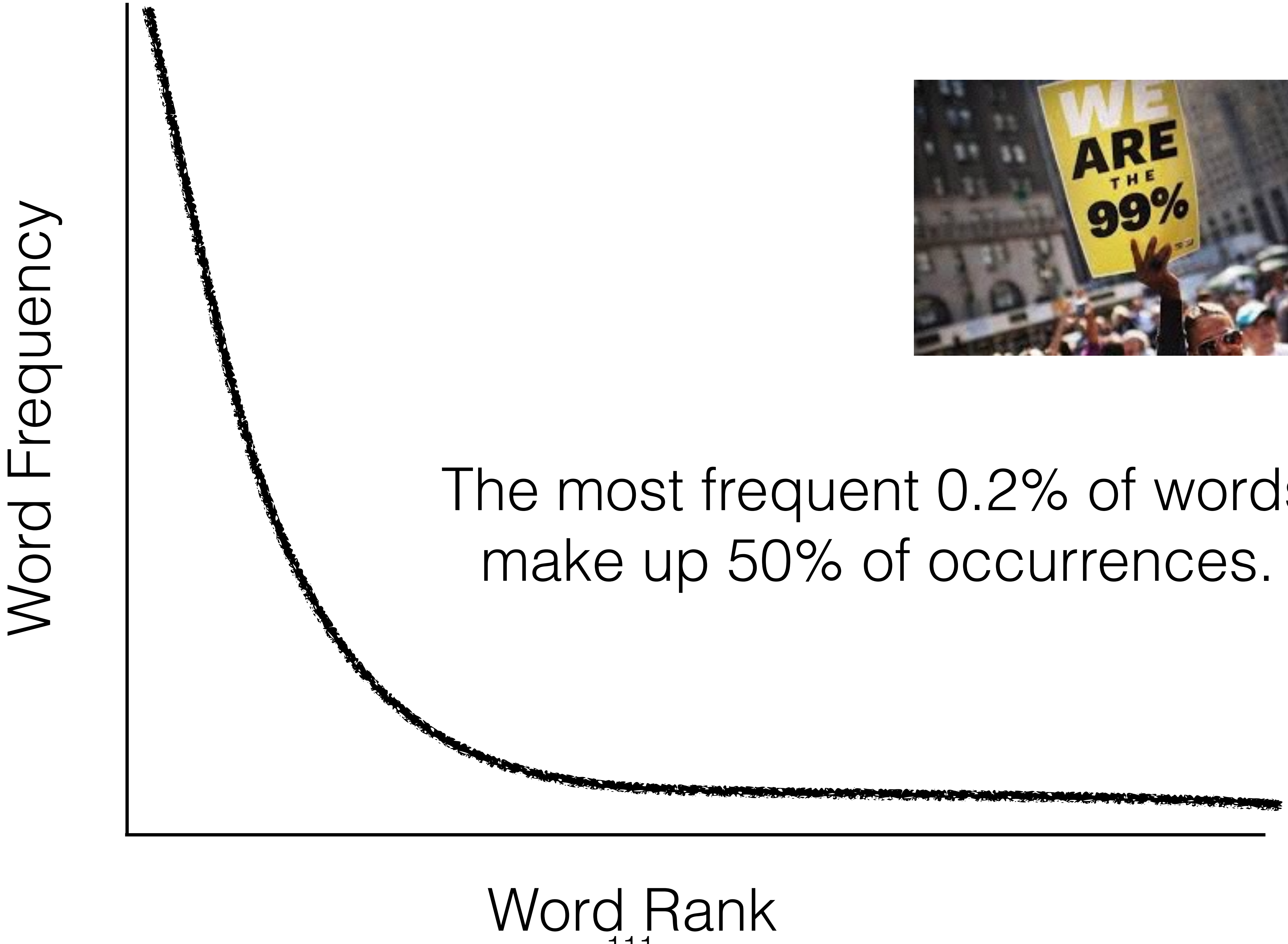
Zipf's Law



"The frequency of any word is inversely proportional to its rank in the frequency table" (Wikipeda)

https://en.wikipedia.org/wiki/Zipf%27s_law

Zipf's Law



The most frequent 0.2% of words make up 50% of occurrences.

Real Life Application

Subject	Predicate	Object	Categories
Barack Obama	won	the electoral vote	Person, US_Presidents, Huffington_Post_Columnists
Kamala Lopez	wrote	an op-ed for HuffPo	Person, Huffington_Post_Columnists, Actor

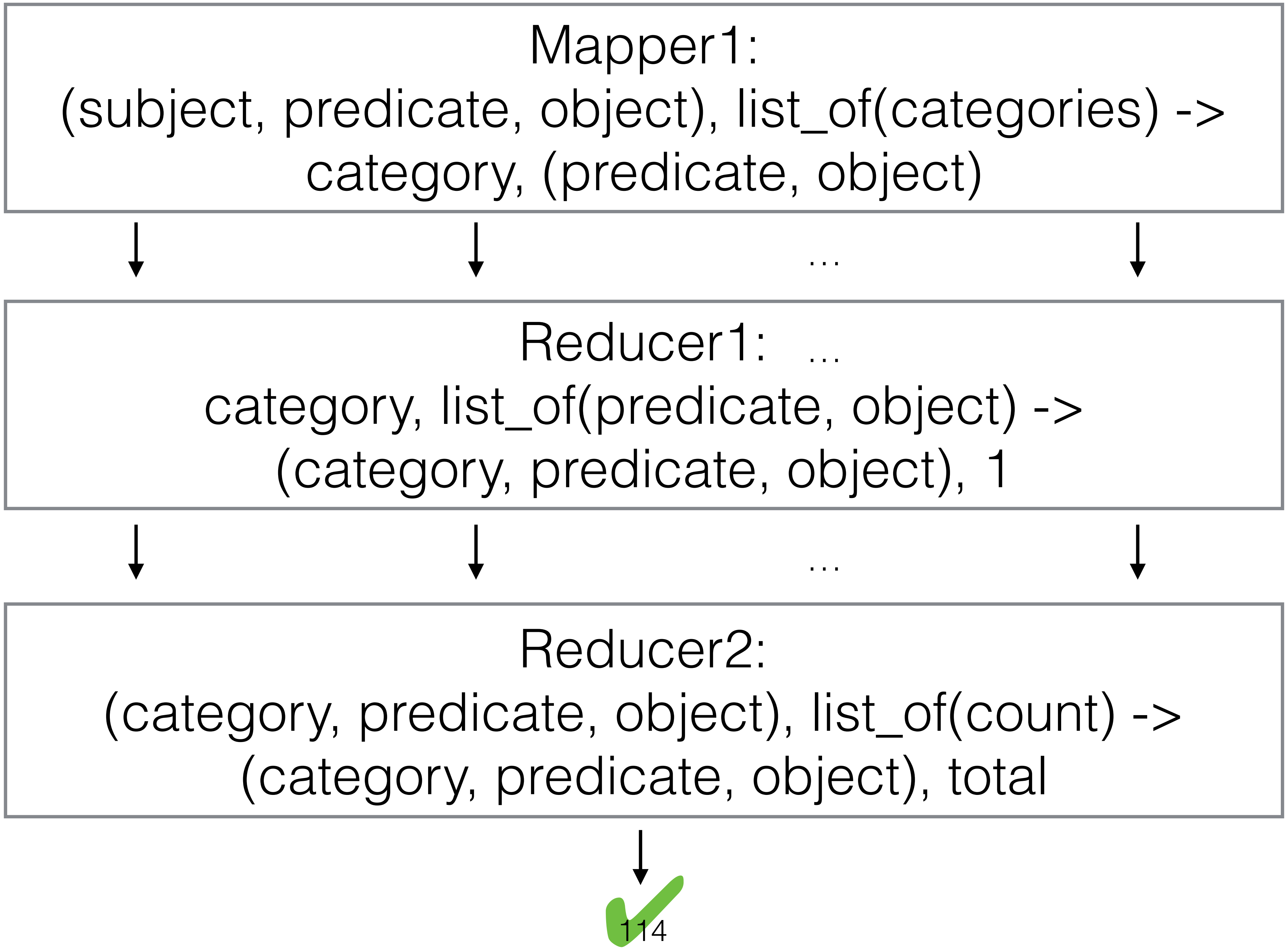
Predicate	Object	Category	Score
won	the electoral vote	US_Presidents	0.92
won	the electoral vote	Person	0.89
won	the electoral vote	Huffington Post Columnists	0.23
wrote	an op-ed for HuffPo	Huffington Post Columnists	0.99
wrote	an op-ed for HuffPo	Person	0.91

Real Life Application

Subject	Predicate	Object	Categories
Barack Obama	won	the electoral vote	Person, US_Presidents, Huffington_Post_Columnists
Kamala Lopez	wrote	an op-ed for HuffPo	Person, Huffington_Post_Columnists, Actor

Predicate	Object	Category	Score
won	the electoral vote	US_Presidents	702,345
won	the electoral vote	Person	812,485
won	the electoral vote	Huffington Post Columnists	24,571
wrote	an op-ed for HuffPo	Huffington Post Columnists	134,213
wrote	an op-ed for HuffPo	Person	136,091

First Attempt



Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

↓ ↓ … ↓

Reducer1: …
category, list_of(predicate, object) ->
(category, predicate, object), 1

↓ ↓ … ↓

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

↓

✓

First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

↓ ↓ … ↓

Reducer1: …
category, list_of(predicate, object) ->
(category, predicate, object), 1

↓ ↓ … ↓

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

↓ ✓

First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

↓ ↓ … ↓

Reducer1: …
category, list_of(predicate, object) ->
(category, predicate, object), 1

↓ ↓ … ↓

(c >

Every tuple involving a signle category (e.g. "Person") has to go through the same reducer...

First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

↓ ↓ ... ↓

Reducer1: ...
category, list_of(predicate, object) ->
(category, predicate, object), 1

↓ ↓ ... ↓

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

↓

✓

First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

↓ ↓ ... ↓

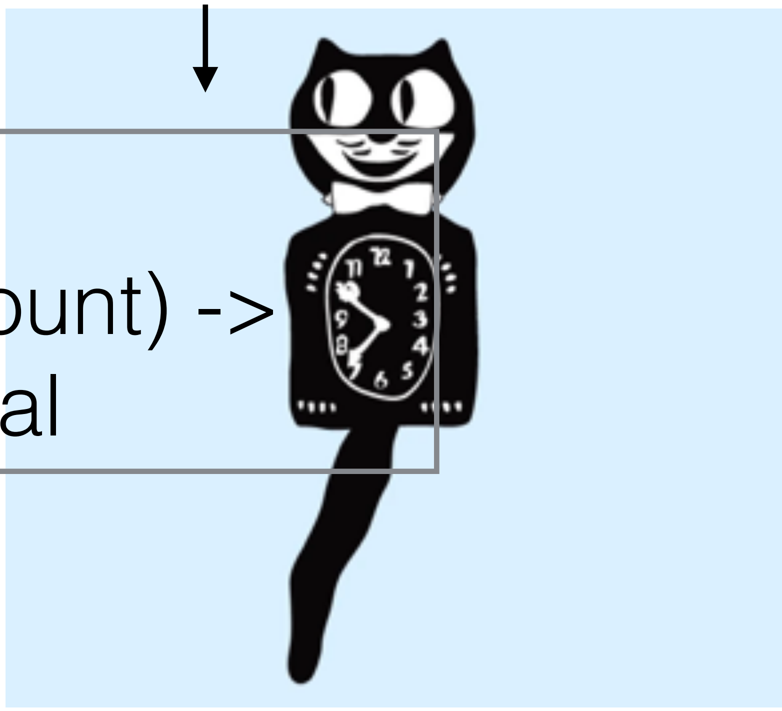
Reducer1: ...
category, list_of(predicate, object) ->
(category, predicate, object), 1

↓ ↓ ... ↓

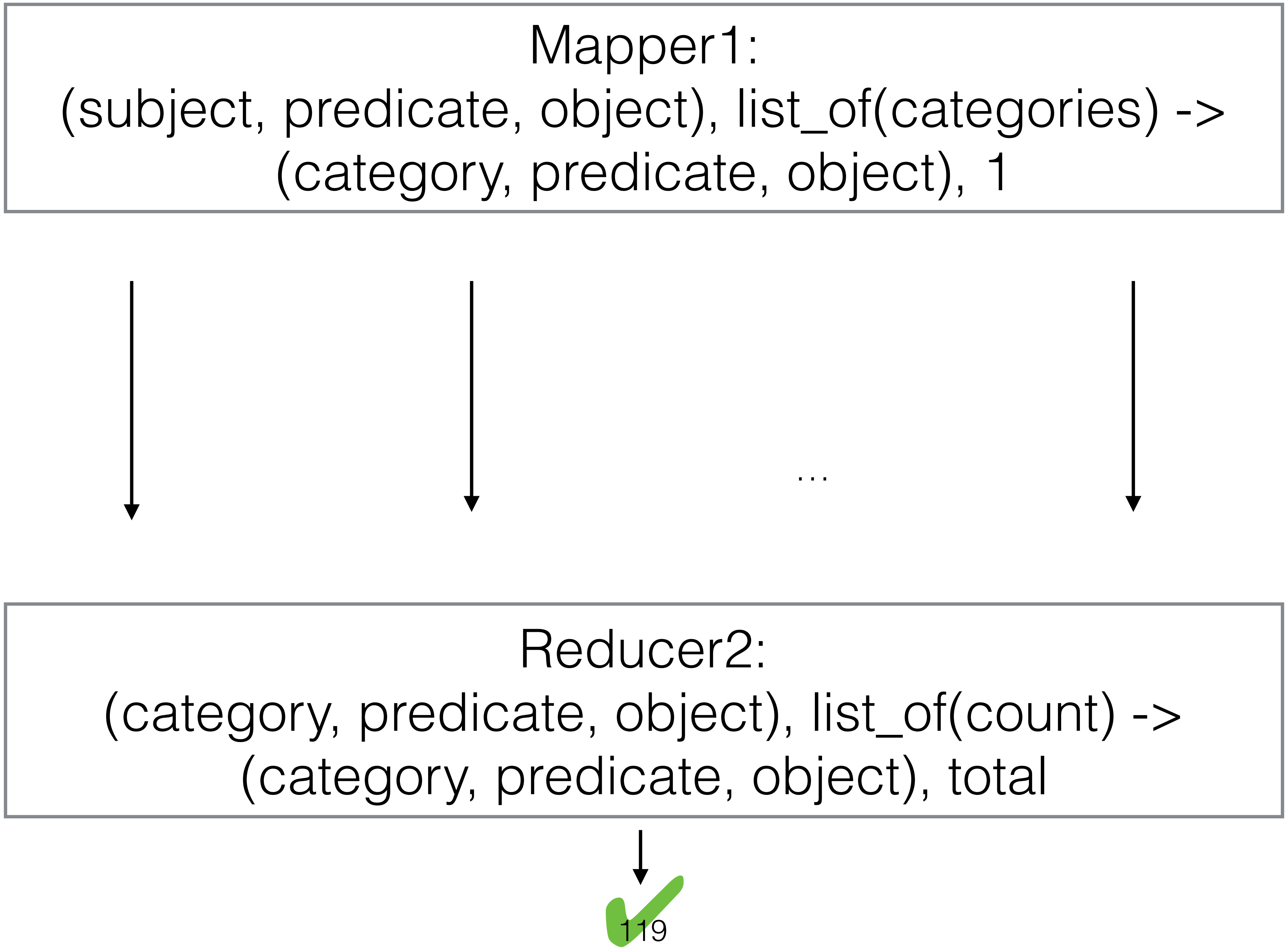
Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

↓

✓



So much better!



Mapper1:
(subject, predicate, object), list_of(categories) ->
(category, predicate, object), 1

…

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

Alright, scram.