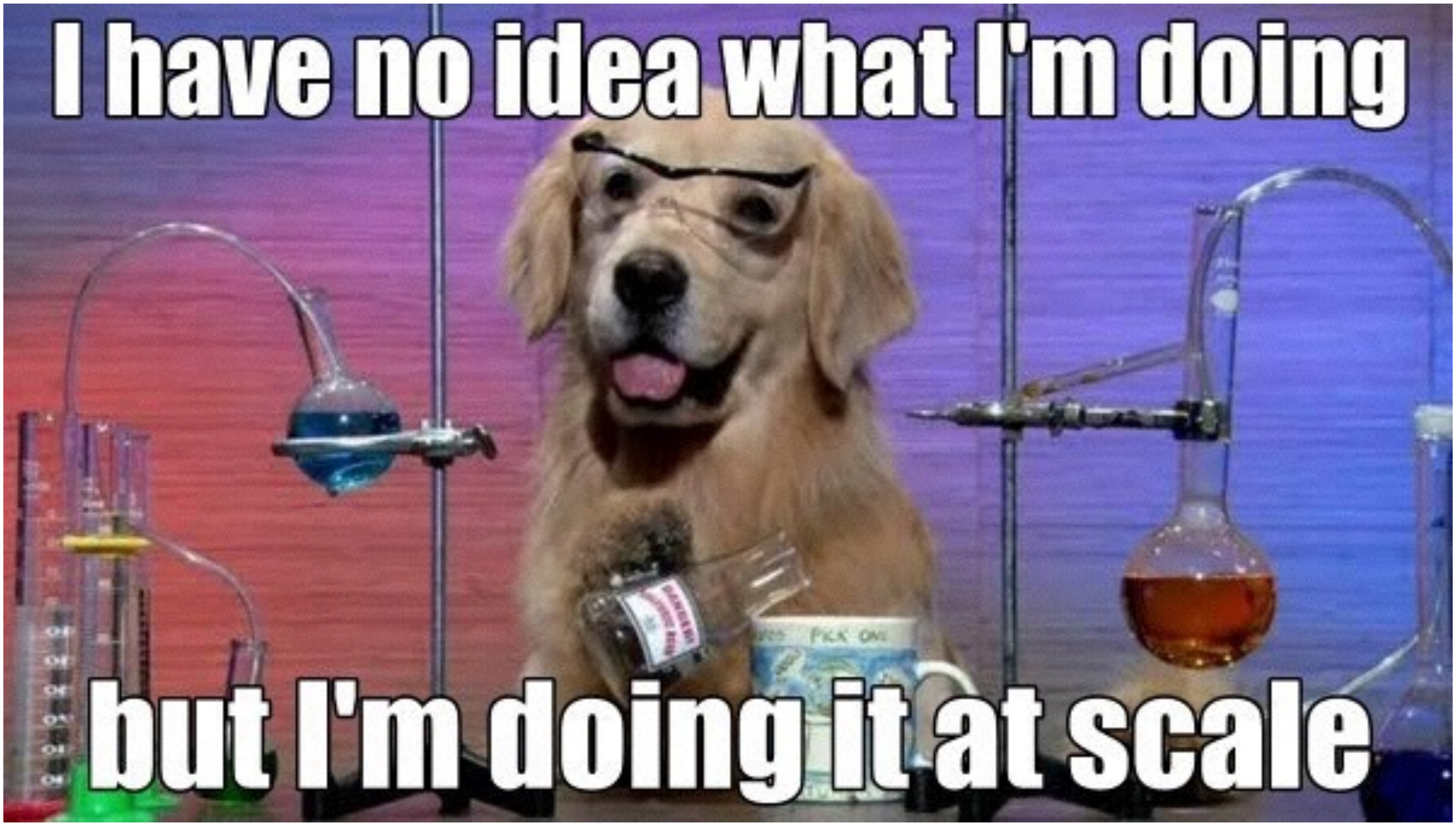
MapReduce

February 13, 2020
Data Science CSCI 1951A
Brown University
Instructor: Ellie Pavlick
HTAs: Josh Levin, Diane Mutako, Sol Zitter

Announcements

- Project Pitch Presentations
- SQL Grades, late handins
- 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(intermediate_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(intermediate_key, intermediate_value)
  - Reduce: (intermediate_key, list_of(intermediate_value)) -> (out_key, out_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(intermediate_key, intermediate_value)

"group by"

    - Reduce: (intermediate_key, list_of(intermediate_value)) -> (out_key, out_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(intermediate_key, intermediate_value)
  - Reduce: (intermediate_key, list_of(intermediate_value)) -> (out_key, out_value)

Extremely
~~Vague~~ General

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

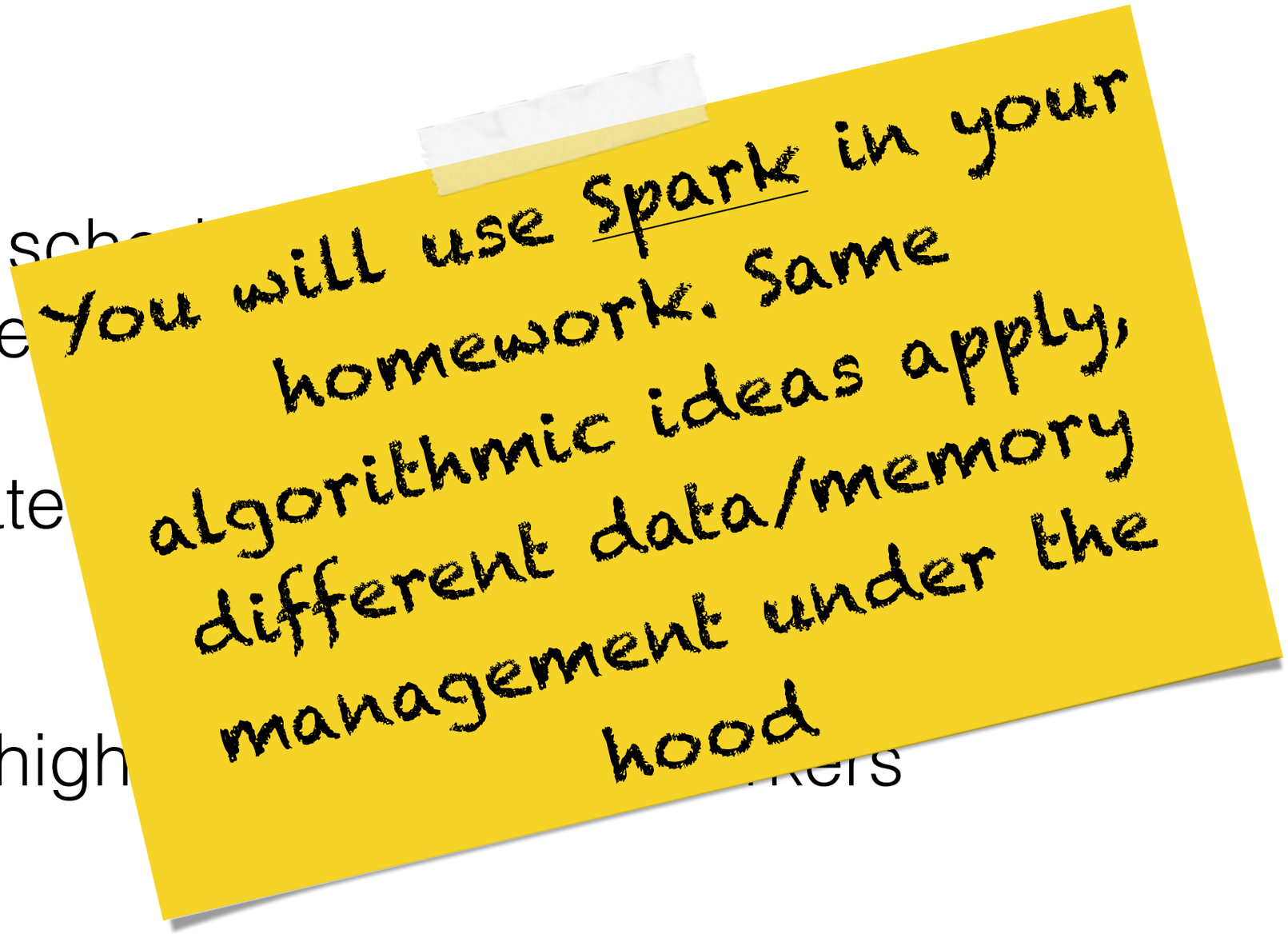
- Assume very high failure rates on workers

Map Reduce

- One “master” sch
  (mapping or re

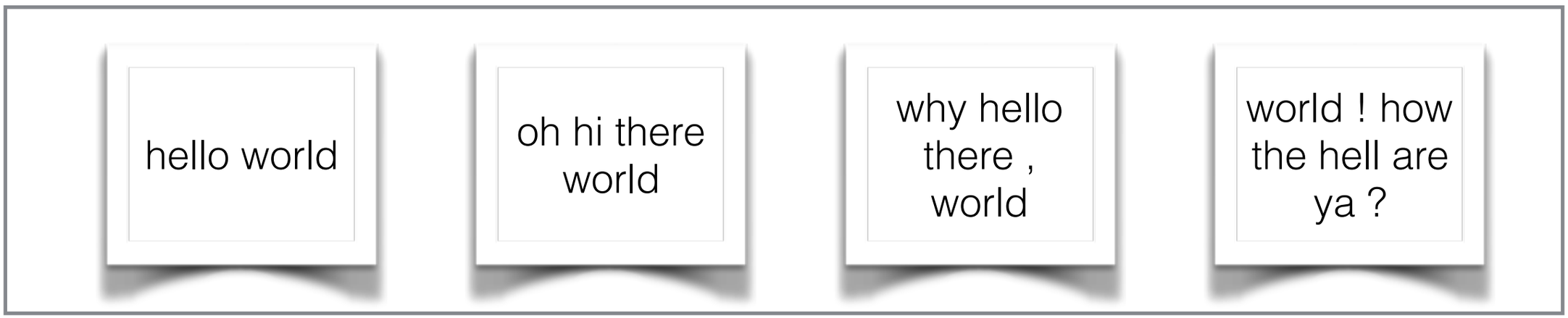
- No shared state
  parallelizable

- Assume very high



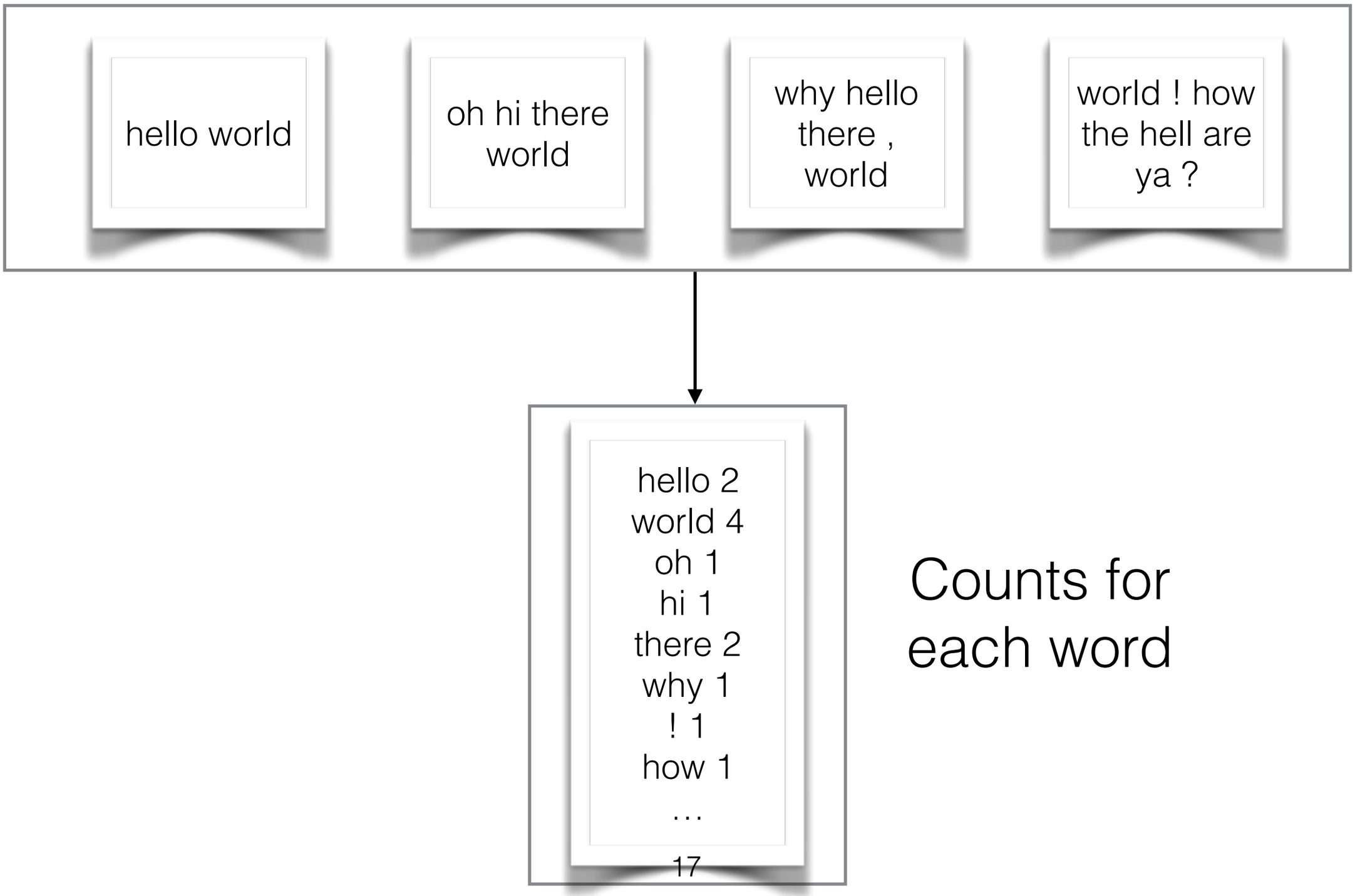
Counting Words

Documents



Counting Words

Documents



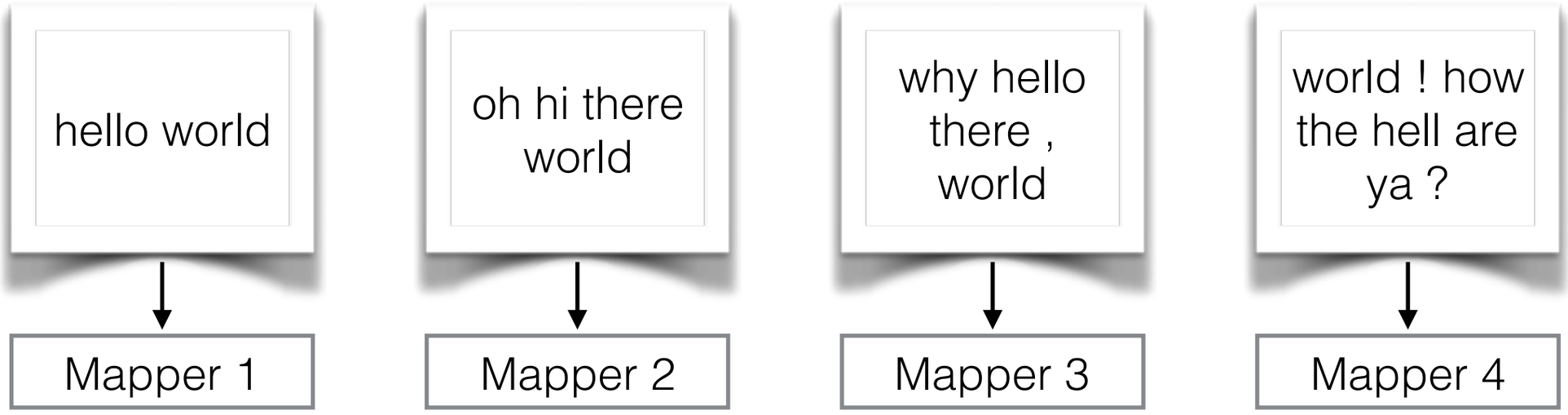
Counts for each word

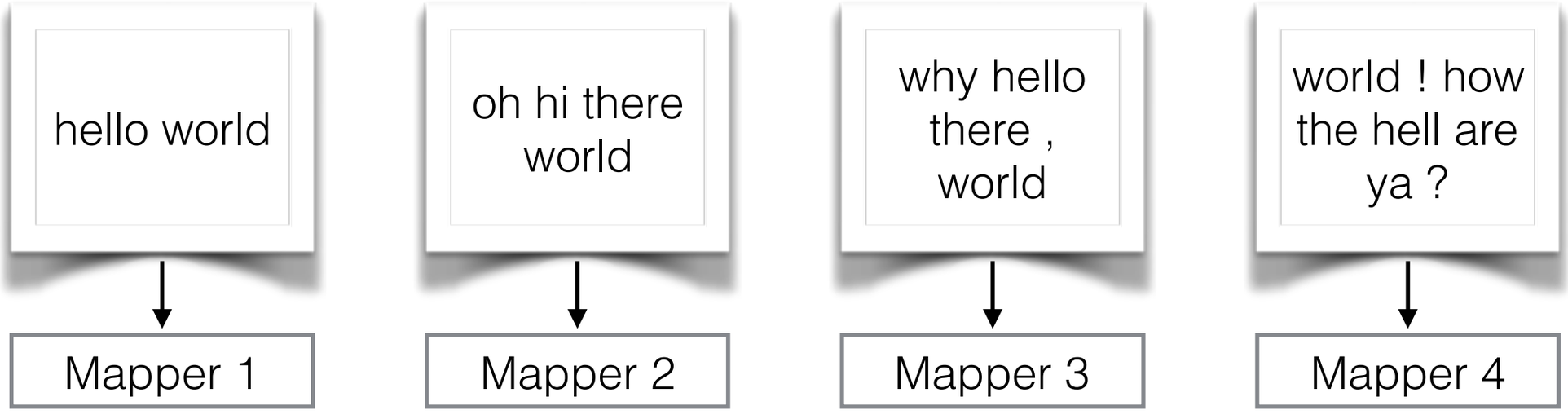
hello world

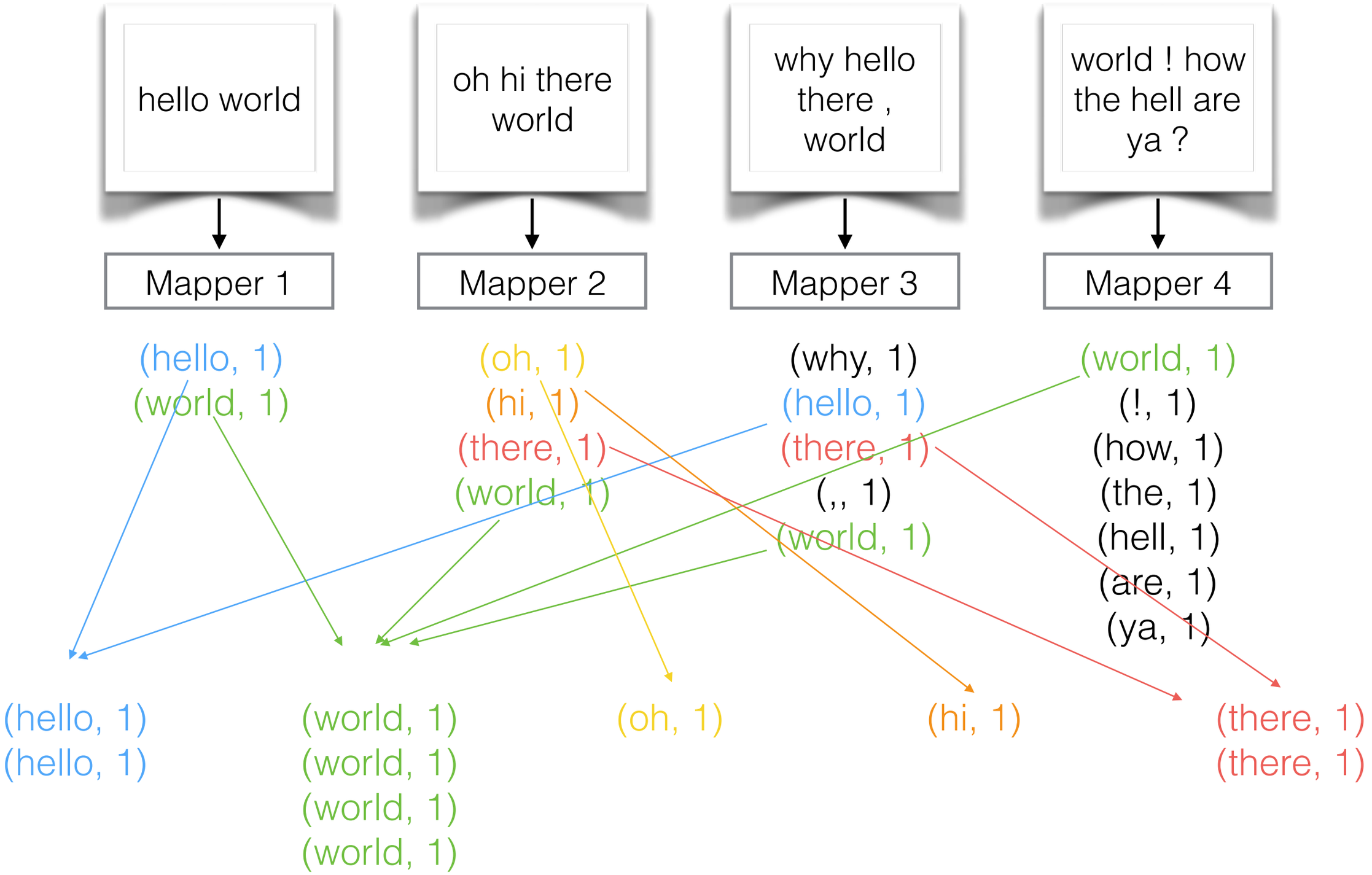
oh hi there world

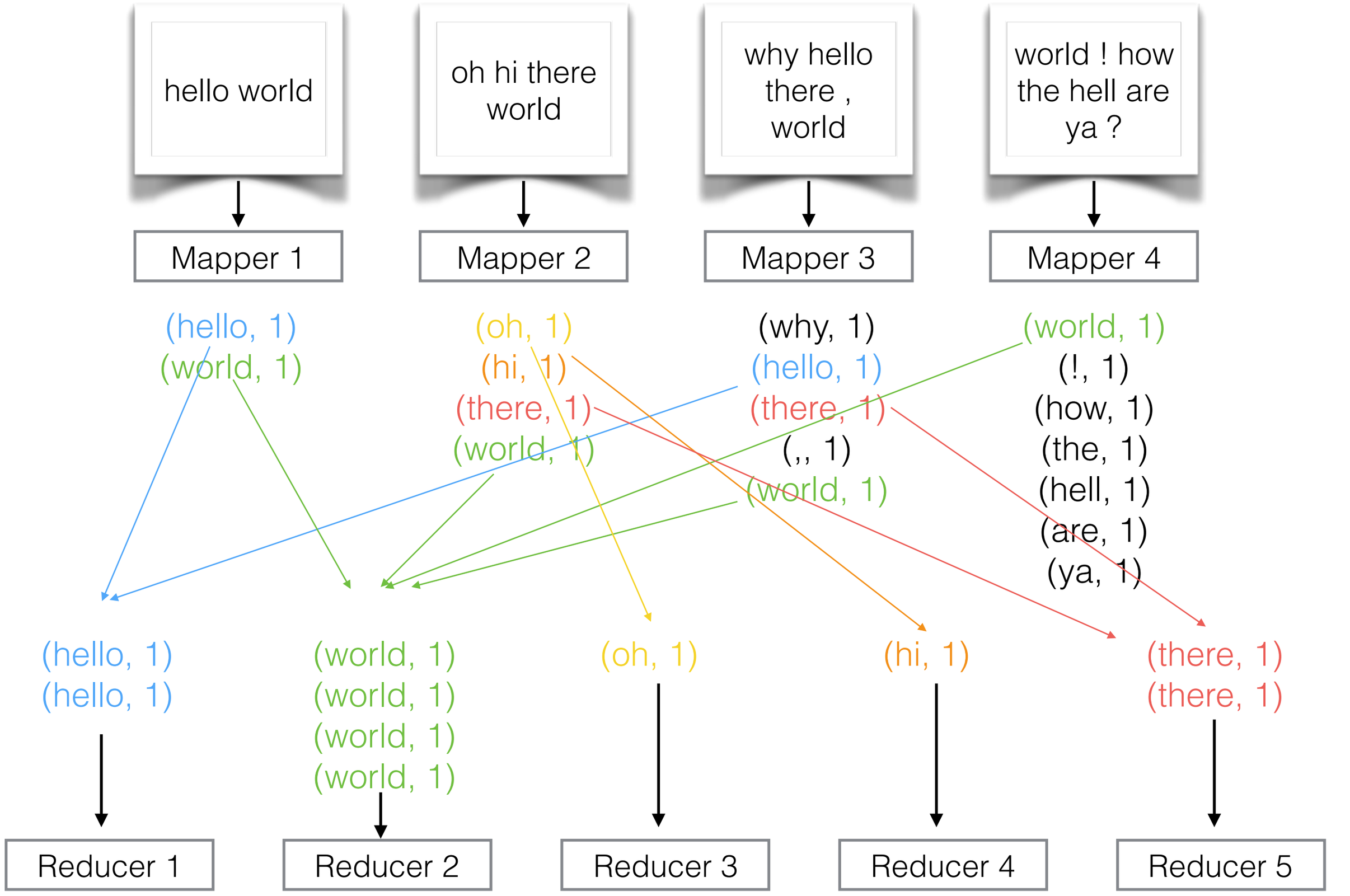
why hello there , world

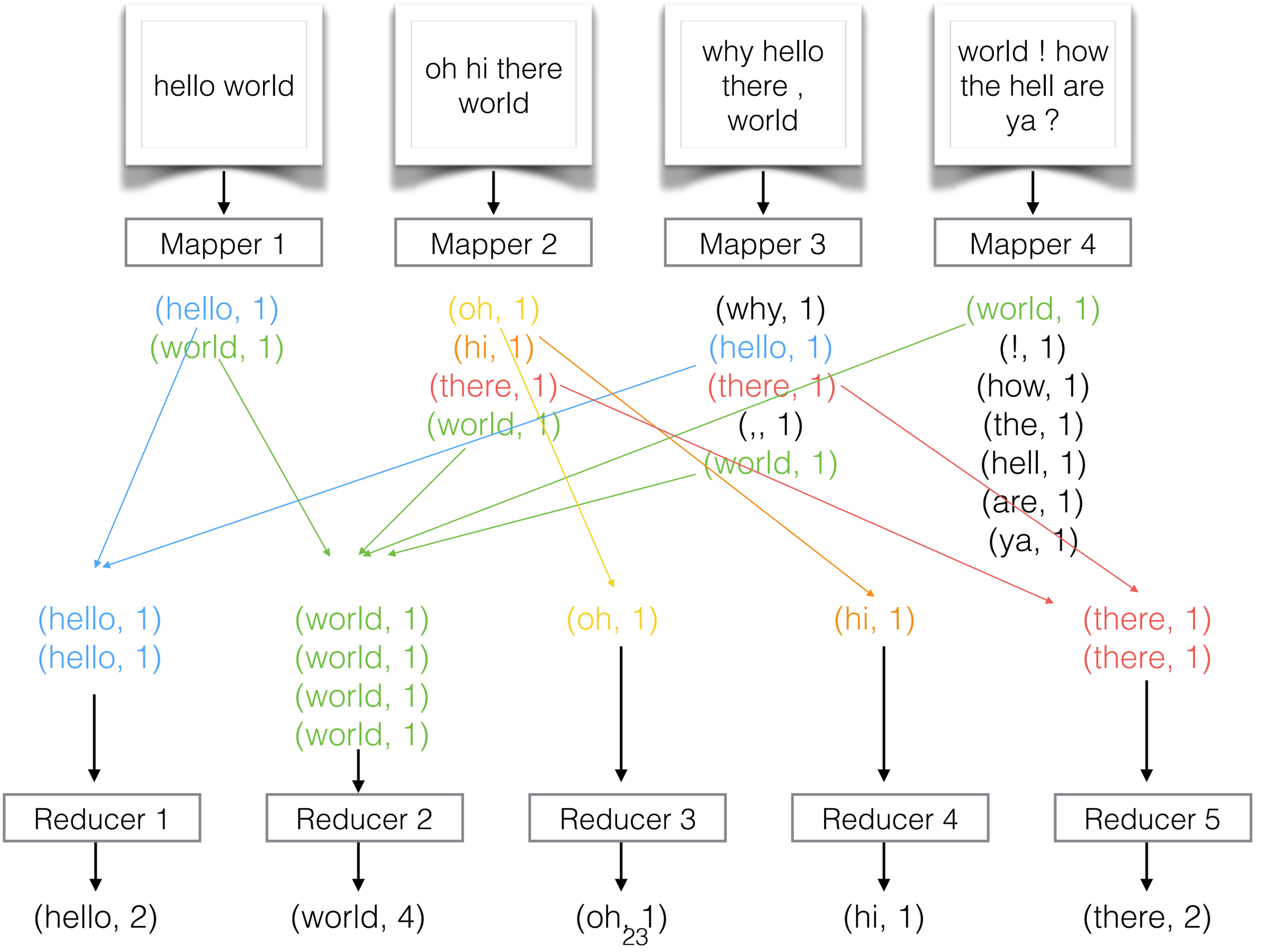
world ! how the hell are ya ?


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 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 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 ?

Map Phase

Mapper 1

(hello, 1)
(world, 1)

Mapper 2

(oh, 1)
(hi, 1)
(there, 1)
(world, 1)

Mapper 3

(why, 1)
(hello, 1)
(there, 1)
(., 1)
(world, 1)

Mapper 4

(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)

Reducer 1

(hello, 1)
(hello, 1)

(hello, 2)

Reducer 2

(world, 1)
(world, 1)
(world, 1)
(world, 1)

(world, 4)

Reducer 3

(oh, 1)

(oh, 1)

Reducer 4

(hi, 1)

(hi, 1)

Reducer 5

(there, 1)
(there, 1)

(there, 2)

Input

hello world

oh hi there world

why hello there , world

world ! how the hell are ya ?

Map Phase

Mapper 1

(hello, 1)
(world, 1)

Mapper 2

(oh, 1)
(hi, 1)
(there, 1)
(world, 1)

Mapper 3

(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)

Mapper 4

(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

(hello, 2)

Reducer 2

(world, 4)

Reducer 3

(oh, 1)

Reducer 4

(hi, 1)

Reducer 5

(there, 2)

Input

hello world

oh hi there world

why hello there , world

world ! how the hell are ya ?

Map Phase

Mapper 1

(hello, 1)
(world, 1)

Mapper 2

(oh, 1)
(hi, 1)
(there, 1)
(world, 1)

Mapper 3

(why, 1)
(hello, 1)
(there, 1)
(, 1)
(world, 1)

Mapper 4

(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)

NOT Sort! (No guarantee 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

(hello, 2)

Reducer 2

(world, 4)

Reducer 3

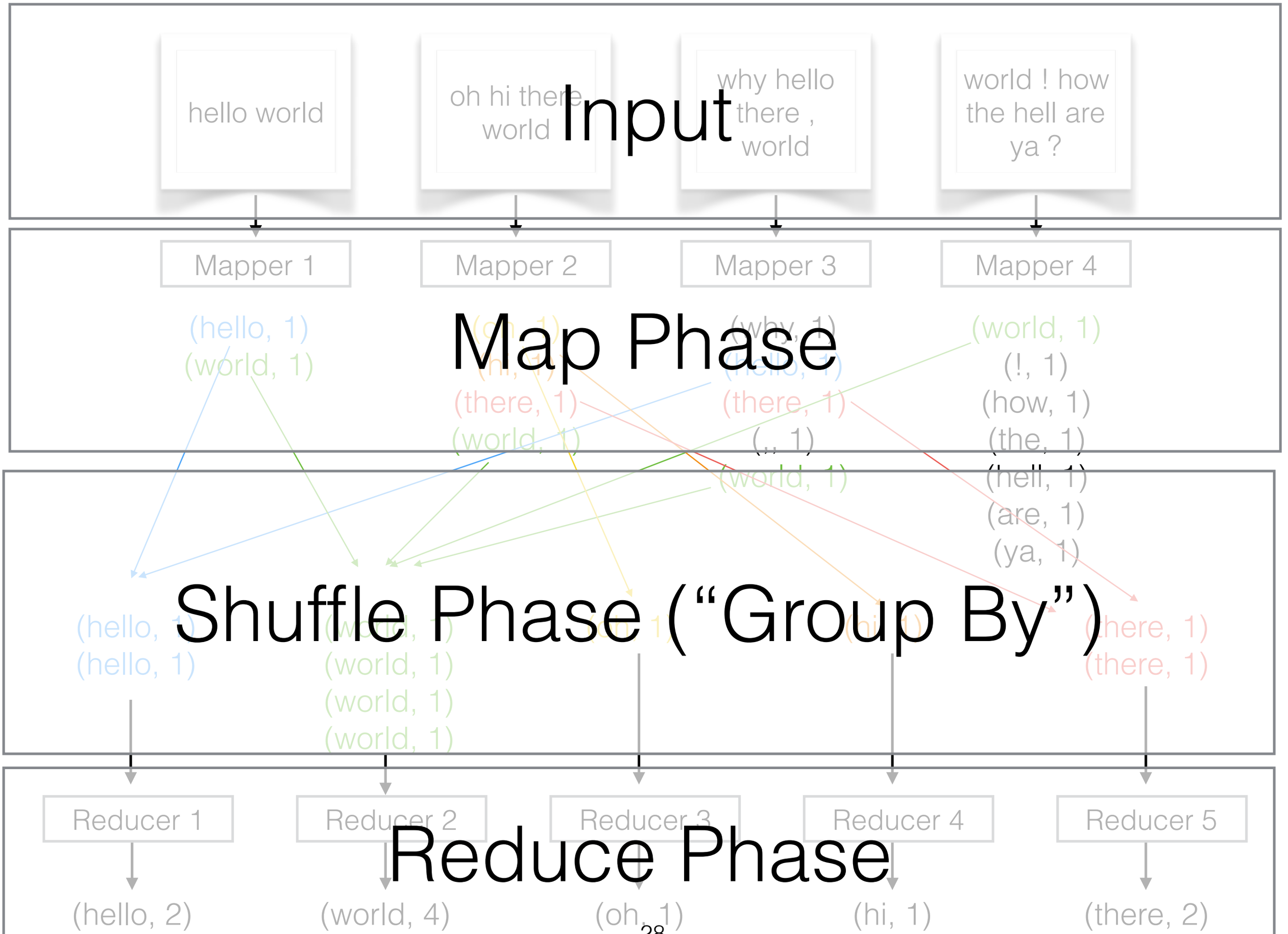
(oh, 1)

Reducer 4

(hi, 1)

Reducer 5

(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)

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)

(there, 1)
(there, 1)

Guarantees same key processed together

Reduce Phase

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5

(hello, 2)

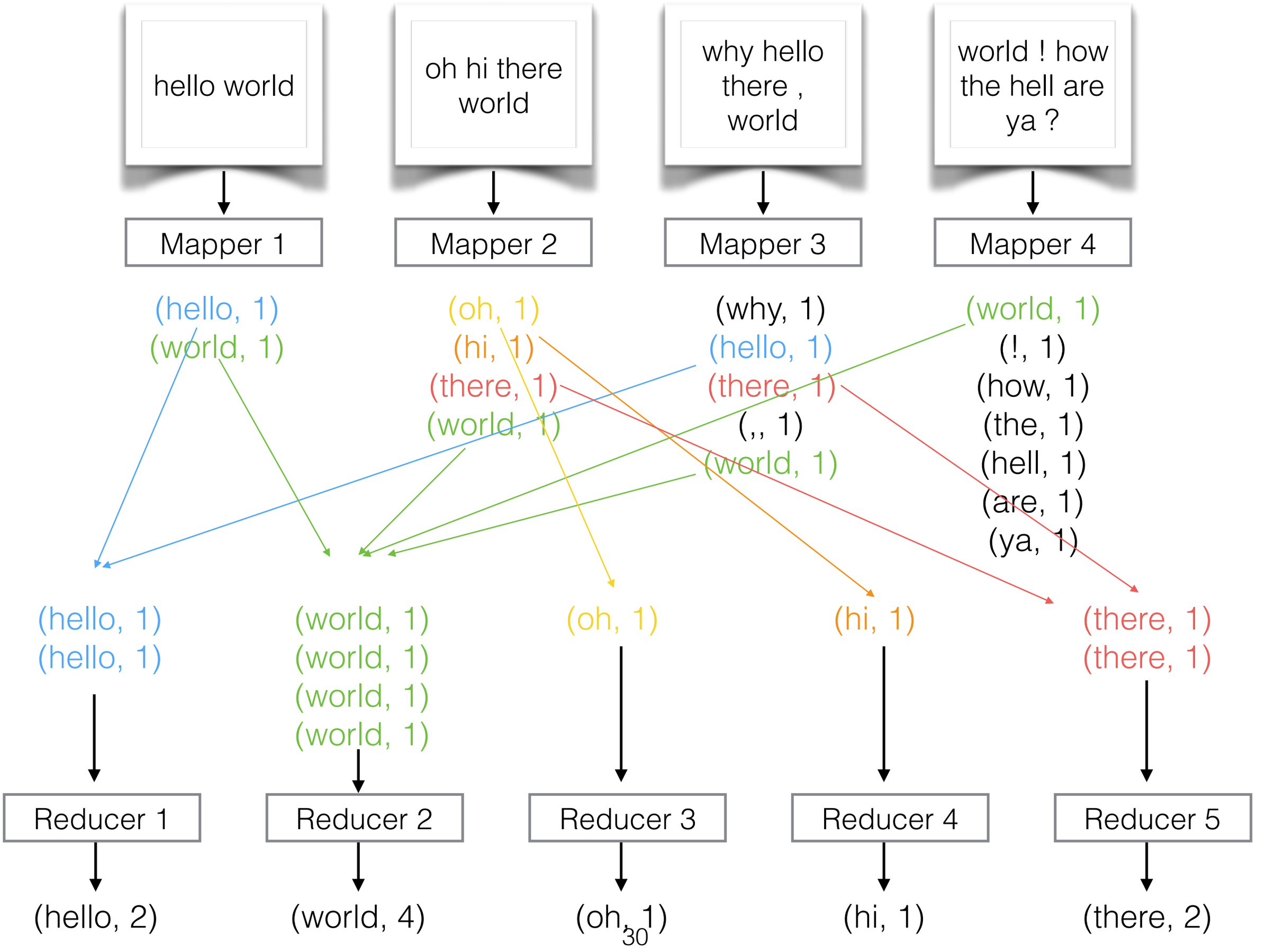
(world, 4)

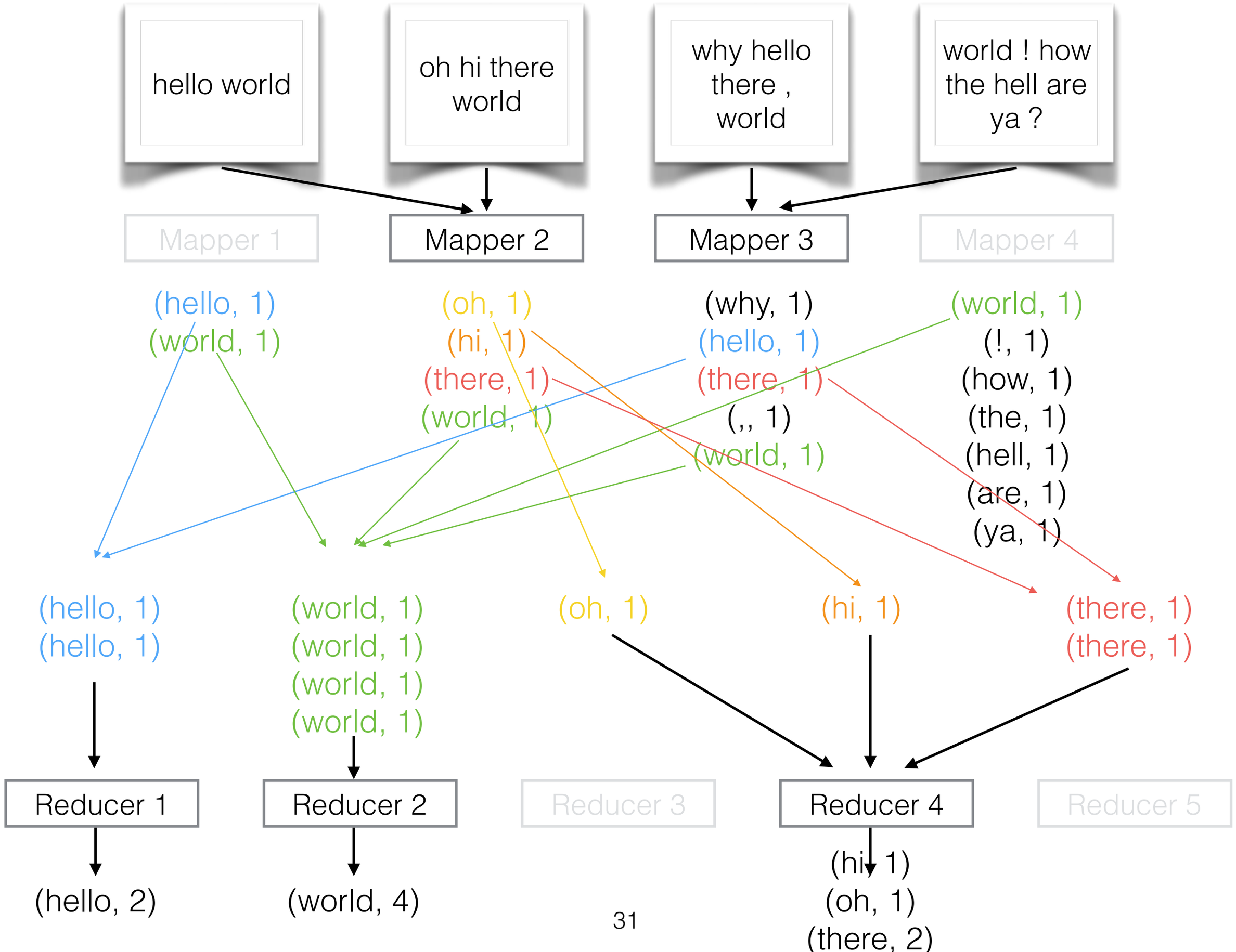
(oh, 1)

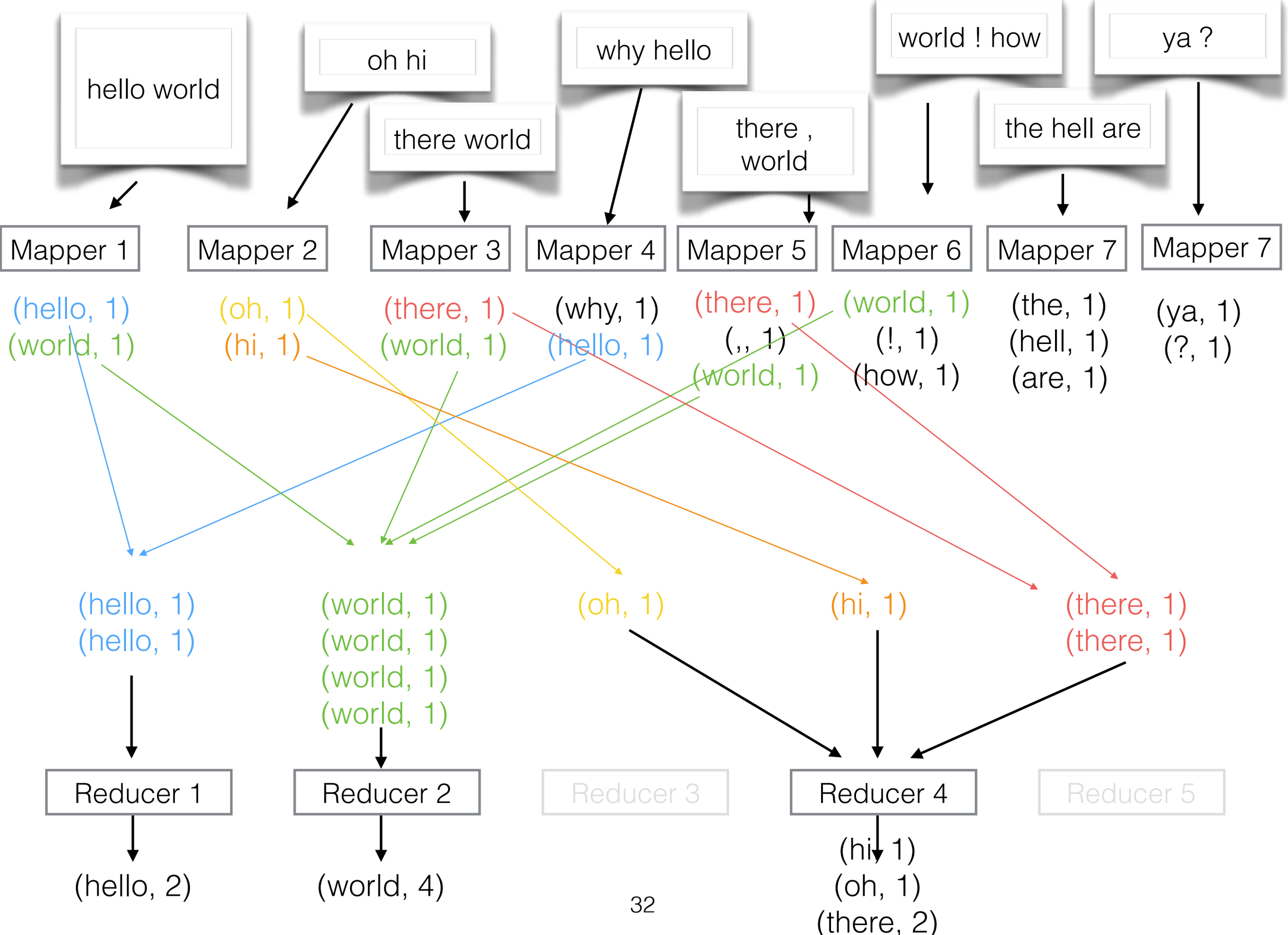
(hi, 1)

(there, 2)

Use for e.g. uniquing, sorting, etc.


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, List(Int)) -> (String, Int){
TODO;
}

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

Reduce

Warning:
Code Snippets/
Pseudocode
(Don't assume this
will look exactly like
this in the HW)

```
/                    ction(s)
d                    ng) -> (String, Int) {
T
}

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

//de                    e function
def                   g, List(Int                    Int){
TODO
}

//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, List(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:

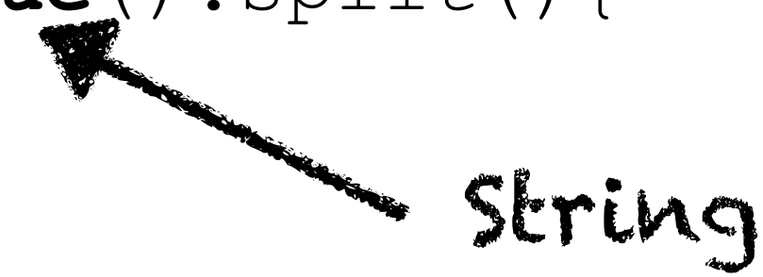
String, Int, Float, Tuples thereof

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);
    }
}
```



Map Reduce

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

List of ints (counts)

Map Reduce

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

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

Input: String

Map: output (word, 1) for every word.

Reduce: Sum counts for each word

(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, List(Int)) -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2: (String, List(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, List(Int)) -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2: (String, List(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! (use reducers instead)

(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, List(Int)) -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2: (String, List(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! (use reducers instead)

D1

hello world, just saying hello

D2

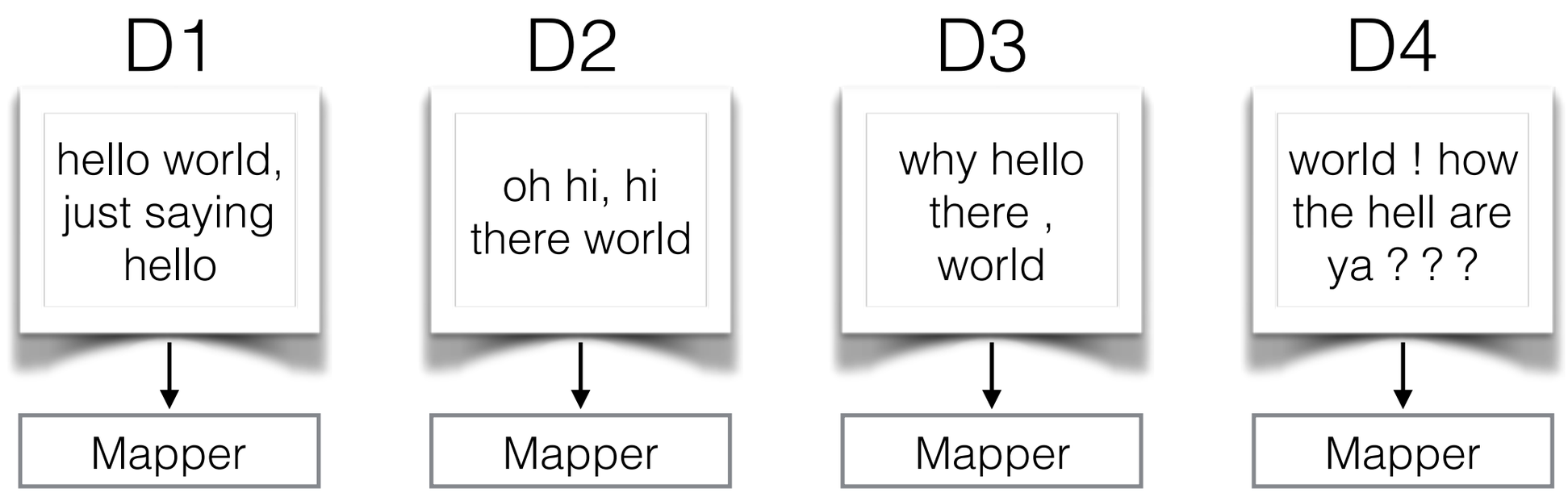
oh hi, hi there world

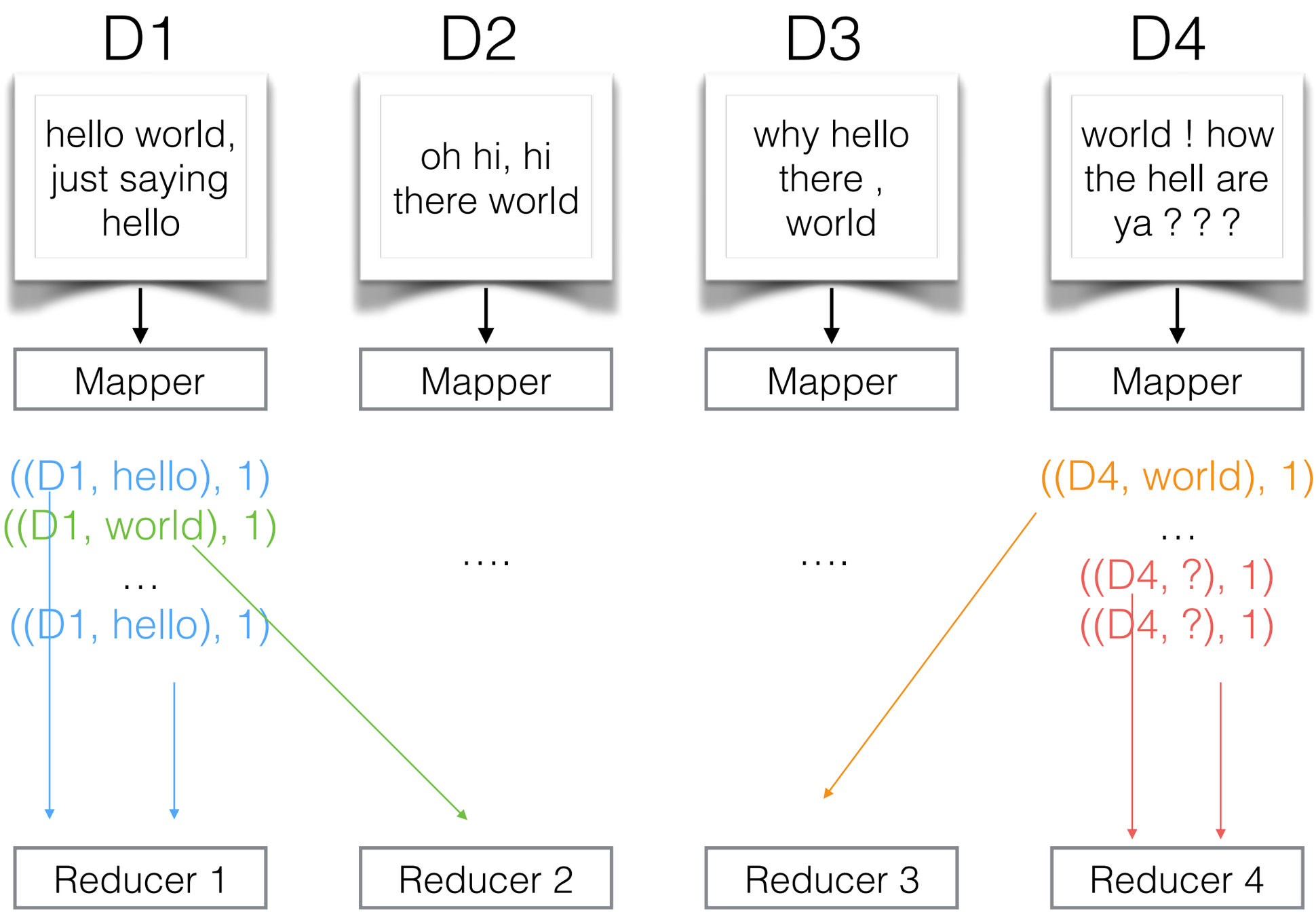
D3

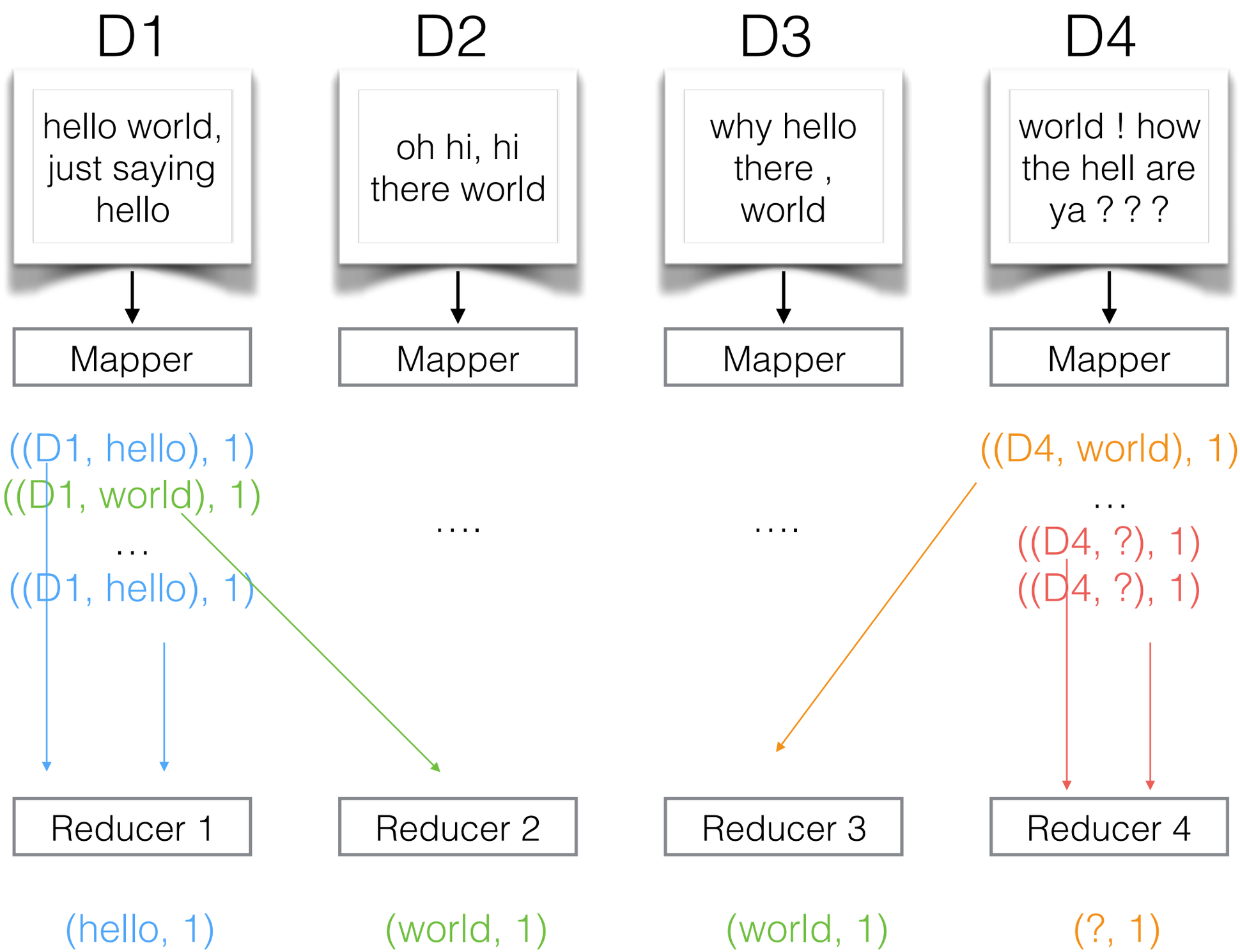
why hello there , world

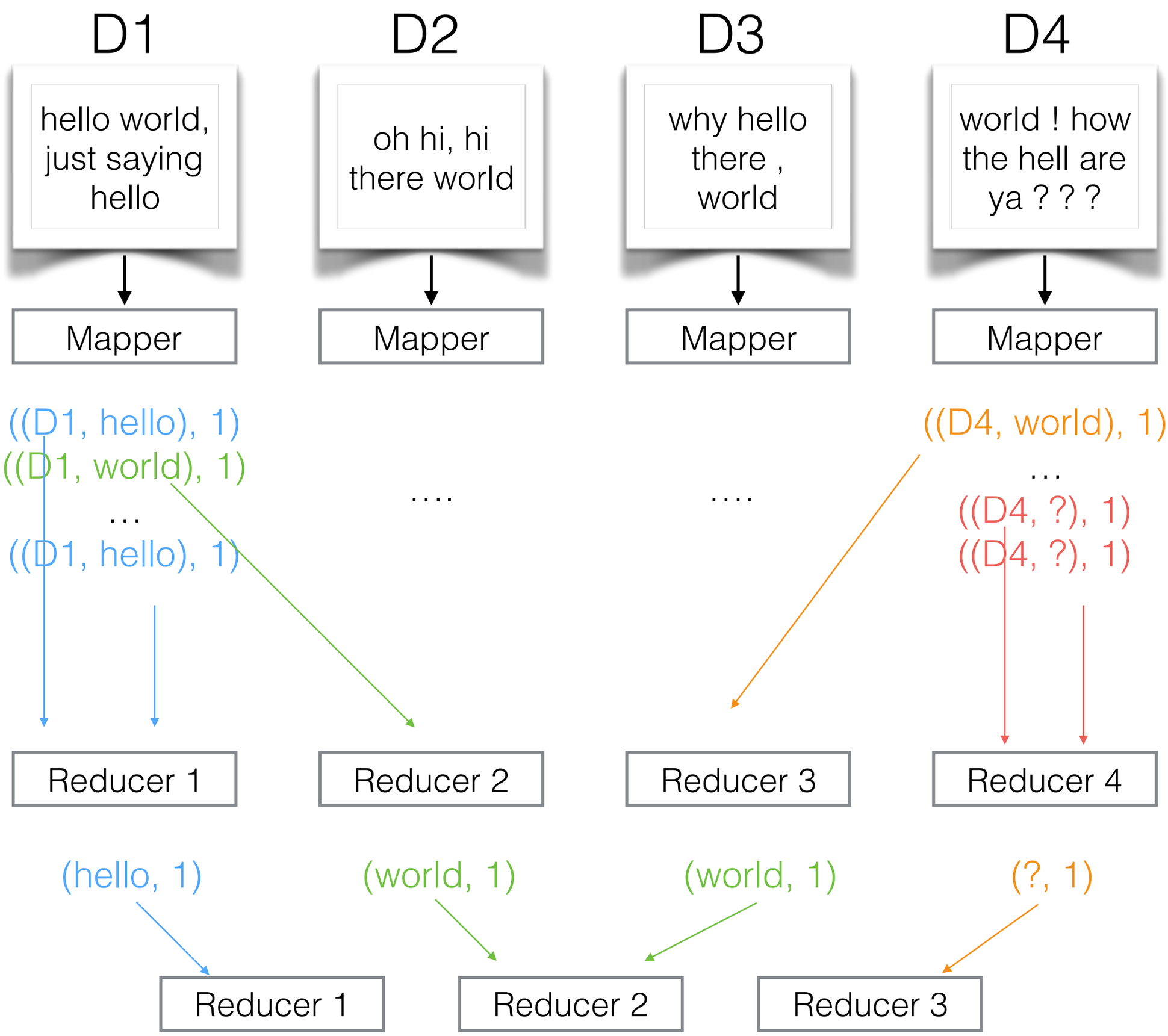
D4

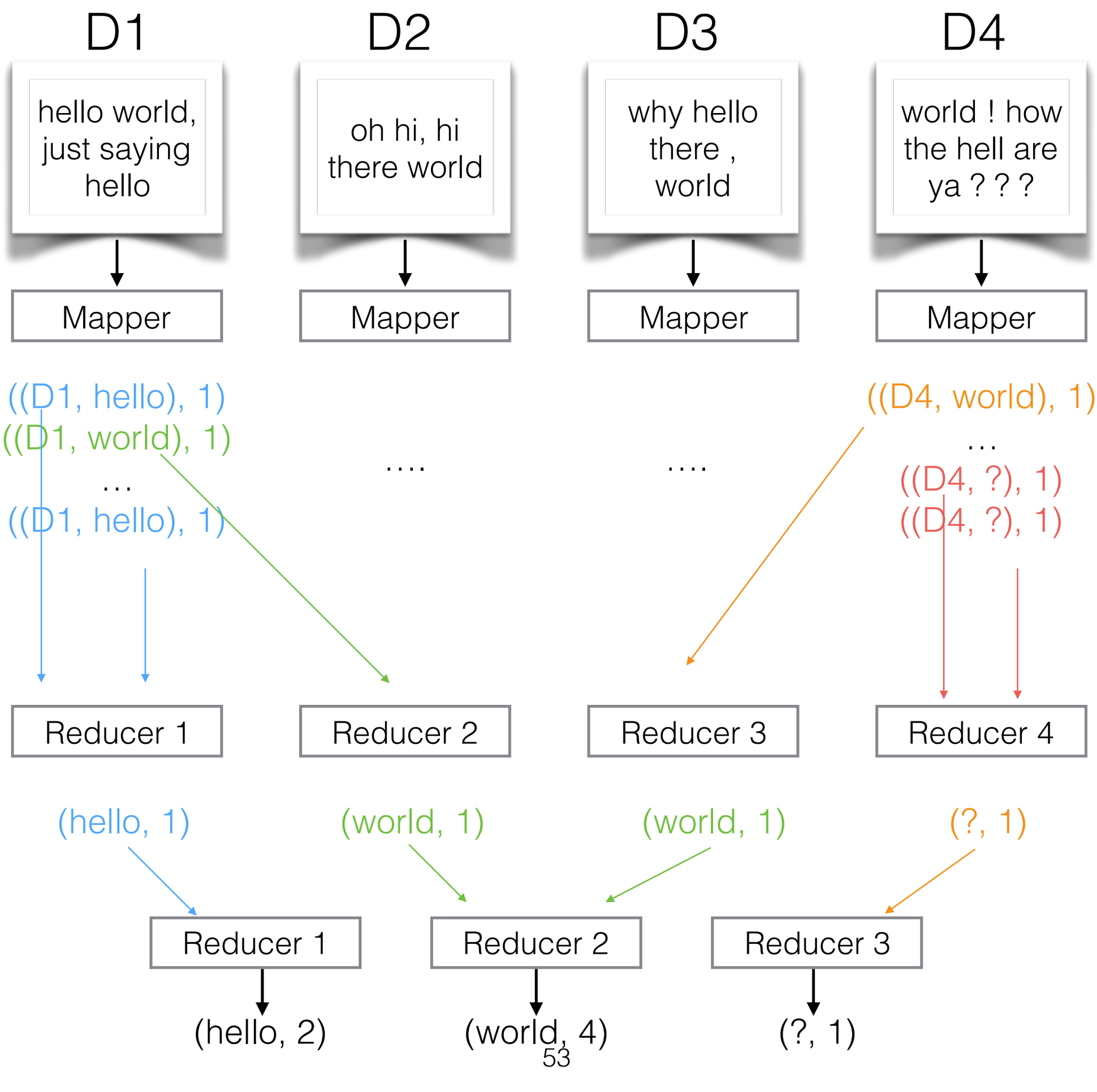
world ! how the hell are ya ? ? ?

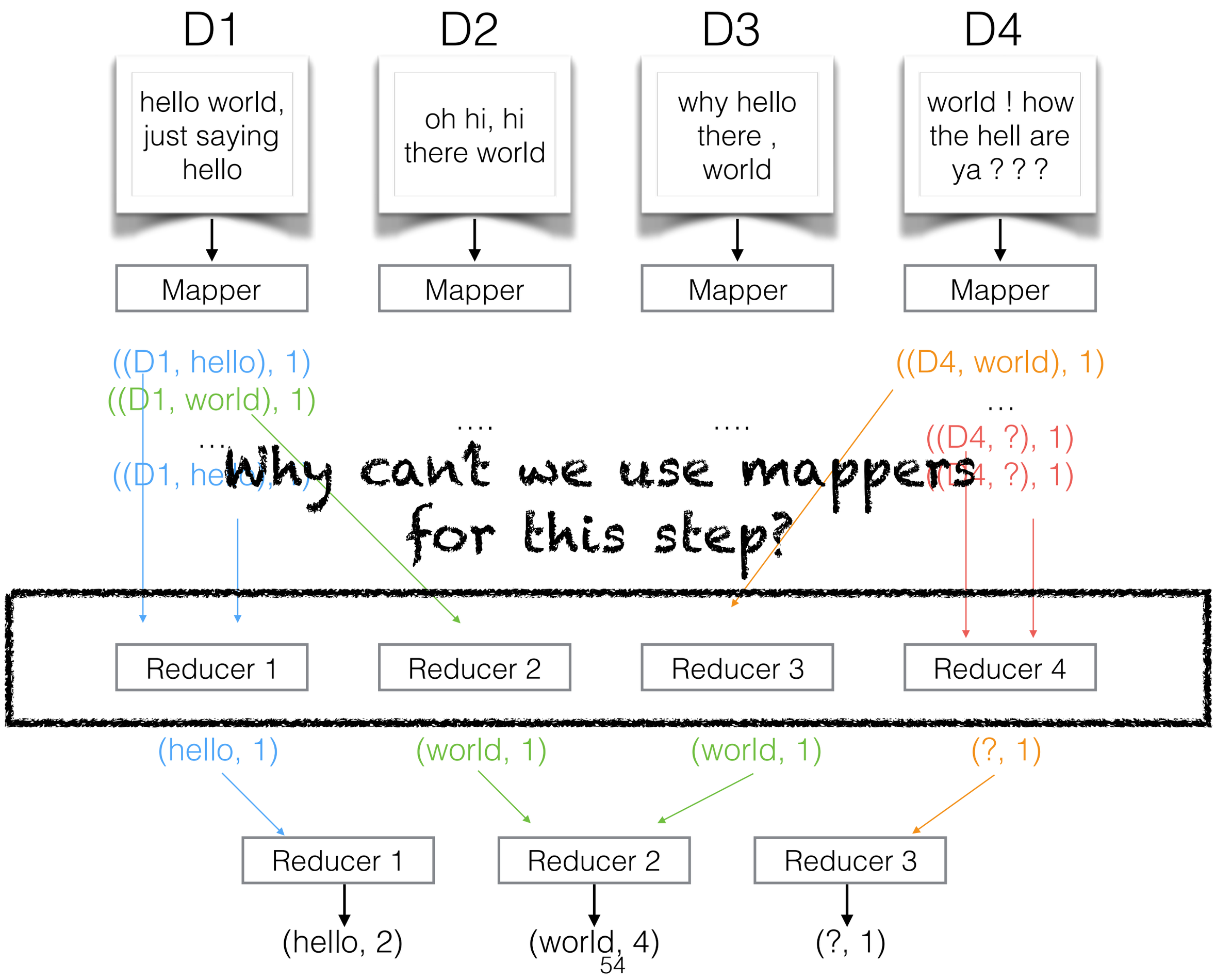

D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
…
((D1, hello), 1)
….
….
((D4, world), 1)
…
((D4, ?), 1)
((D4, ?), 1)

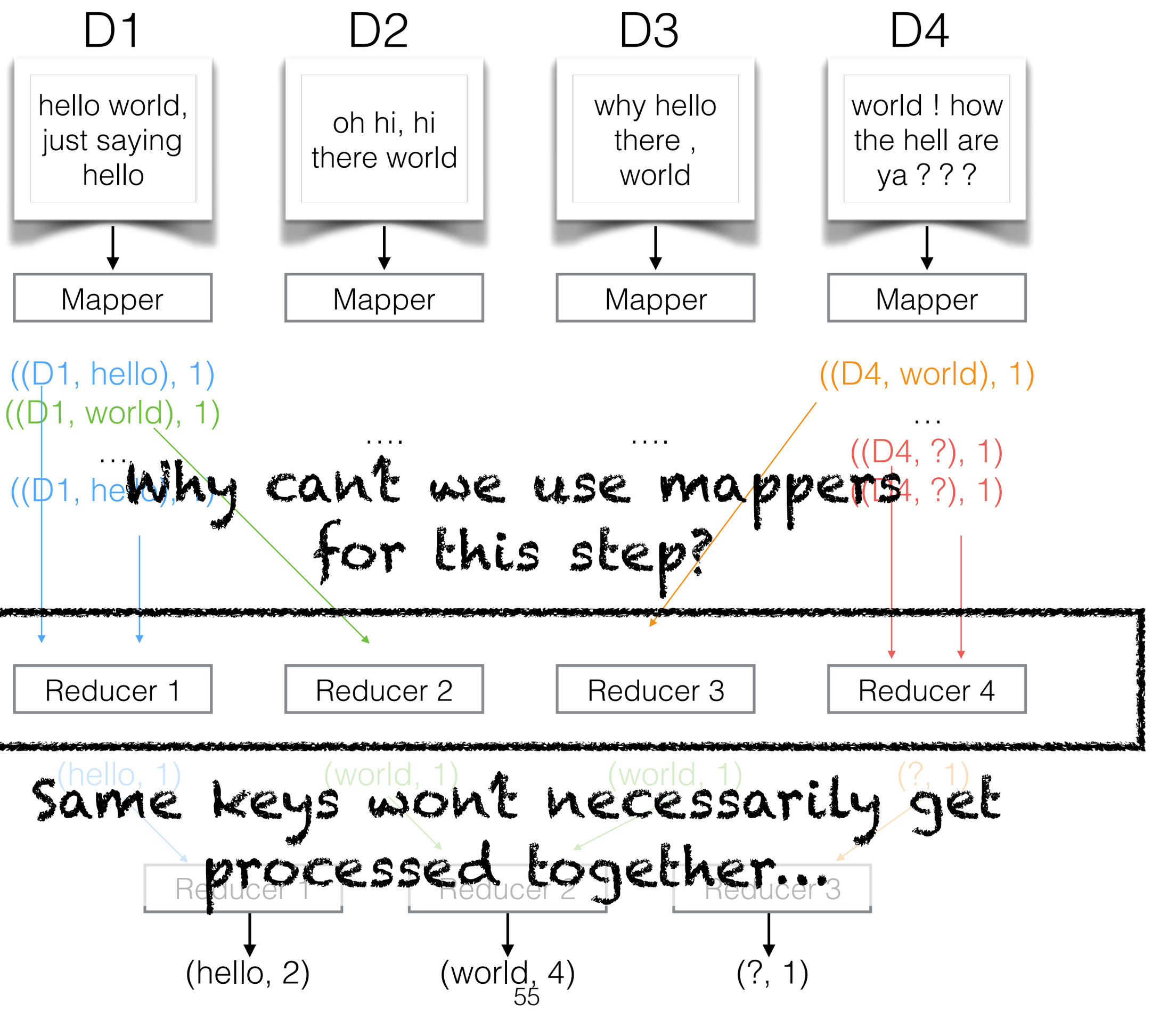

D1
D2
D3
D4
hello world, just saying hello
oh hi, hi there world
why hello there , world
world ! how the hell are ya ? ? ?
Mapper
Mapper
Mapper
Mapper
((D1, hello), 1)
((D1, world), 1)
…
((D1, hello), 1)
….
….
((D4, world), 1)
…
((D4, ?), 1)
((D4, ?), 1)
Reducer 1
Reducer 2
Reducer 3
Reducer 4


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)


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)
Why can't we use mappers for this step?
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)


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)
Why can't we use mappers for this step?
Reducer 1
Reducer 2
Reducer 3
Reducer 4
(hello, 1)
(world, 1)
(world, 1)
(?, 1)
Same keys wont necessarily get processed together...
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, List(Int)) -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2: (String, List(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, List(Int)) -> (String, Int) {
    emit(input.key()[1], 1)
}
// sum the total counts of each word
def ReduceFn2: (String, List(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)
    }
}
def ReduceFn1: (String, List(Int)) -> (String, Int) {
    emit(input.key()[1], 1)
}
// sum the total counts of each word
def ReduceFn2: (String, List(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)
}
```

ignore the value list! ("unique")

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);
}
```

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);
}
```

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!

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!

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)
}
// su
// of
def F
 sum
 for                                  :; }
 emit
}
```

unique documents a word occurs in

```
// 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!

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)
}
// su
// of
def F
 sum
 for                                  ; }
 emit
}
```

unique documents a word occurs in

```
// 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: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);
```

Reducer is by DocId only, so just counts total occurrences

...duce?

(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

Other MapReduce Functions

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

Other MapReduce Functions

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

Other MapReduce Functions

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

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

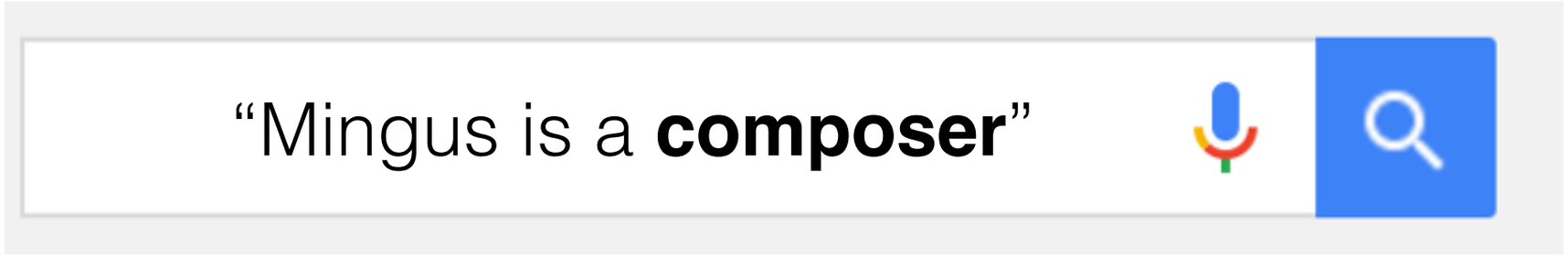
Real Life Application

Real Life Application

Is Charles Mingus a **composer**?

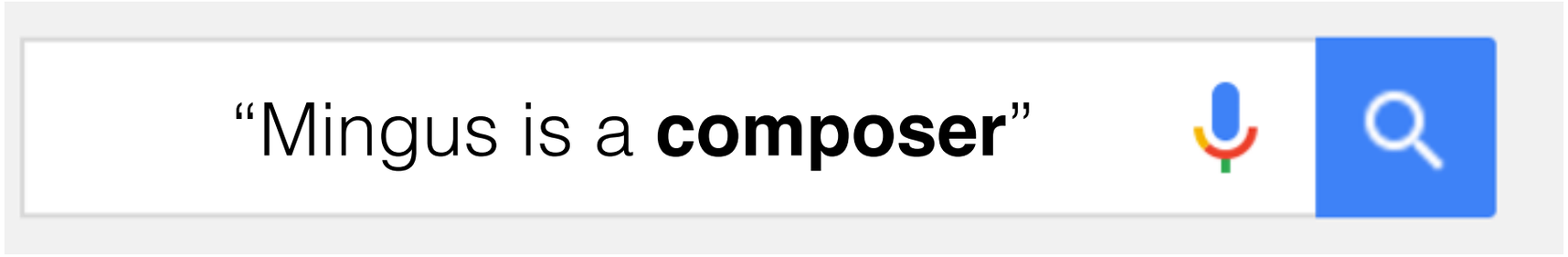
Real Life Application

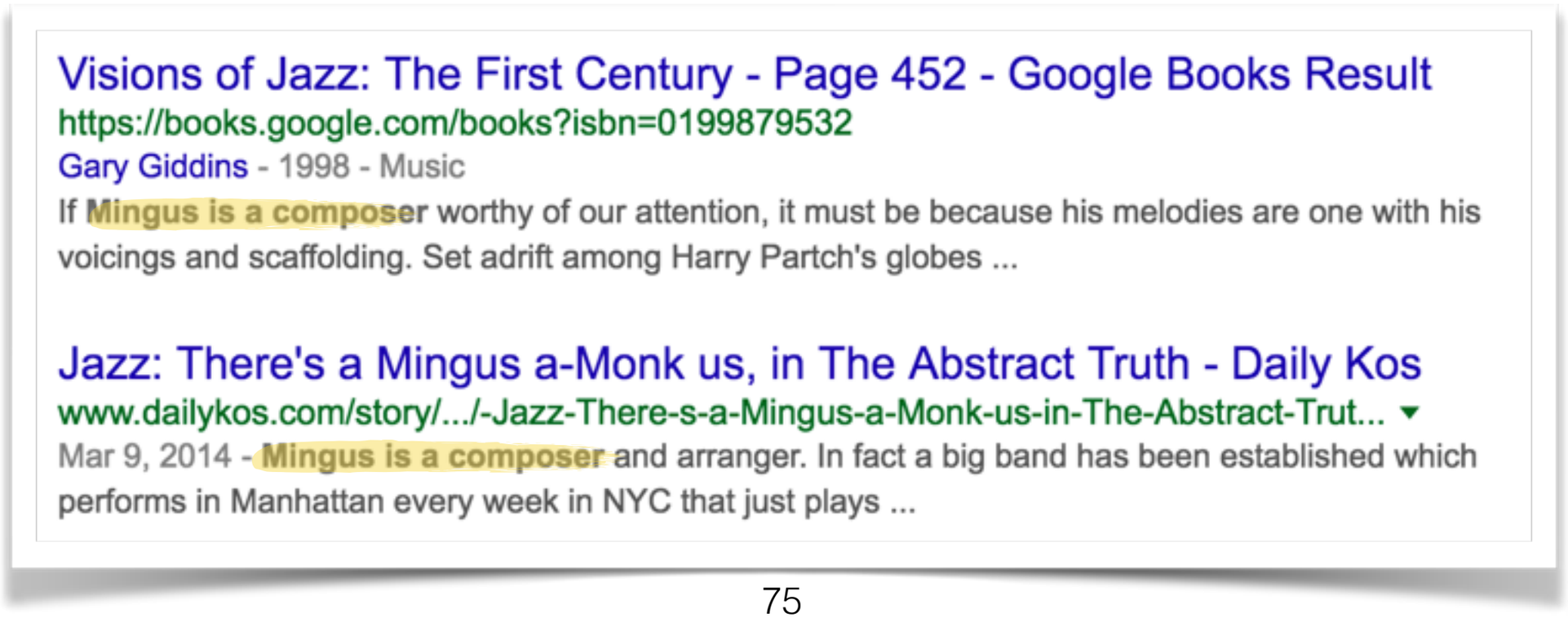
Is Charles Mingus a **composer**?



Real Life Application

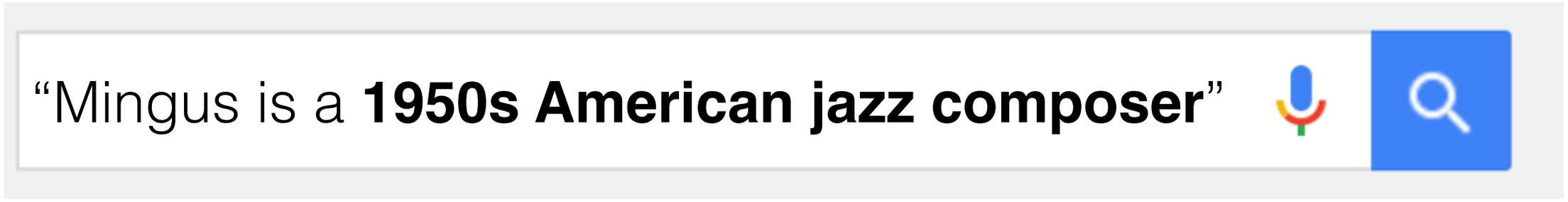
Is Charles Mingus a **composer**?

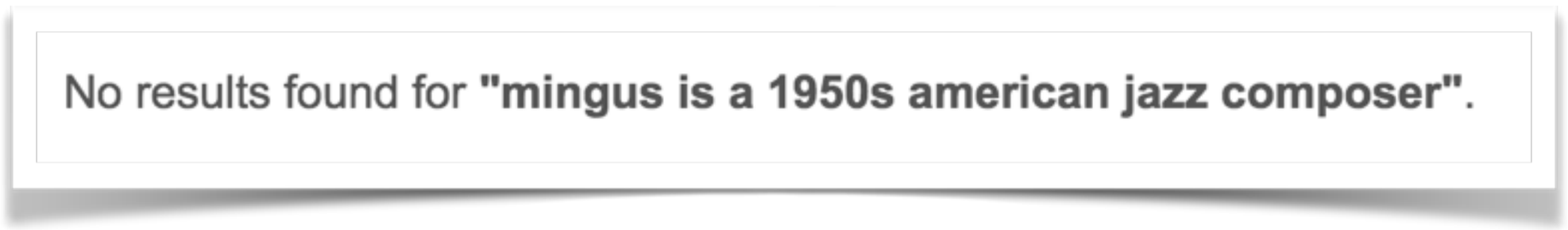




Real Life Application

Is Charles Mingus 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.

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
```

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

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
```

Entity

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

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
```

All the facts for that entity

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

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
```

All the categories for that entity

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

Joins

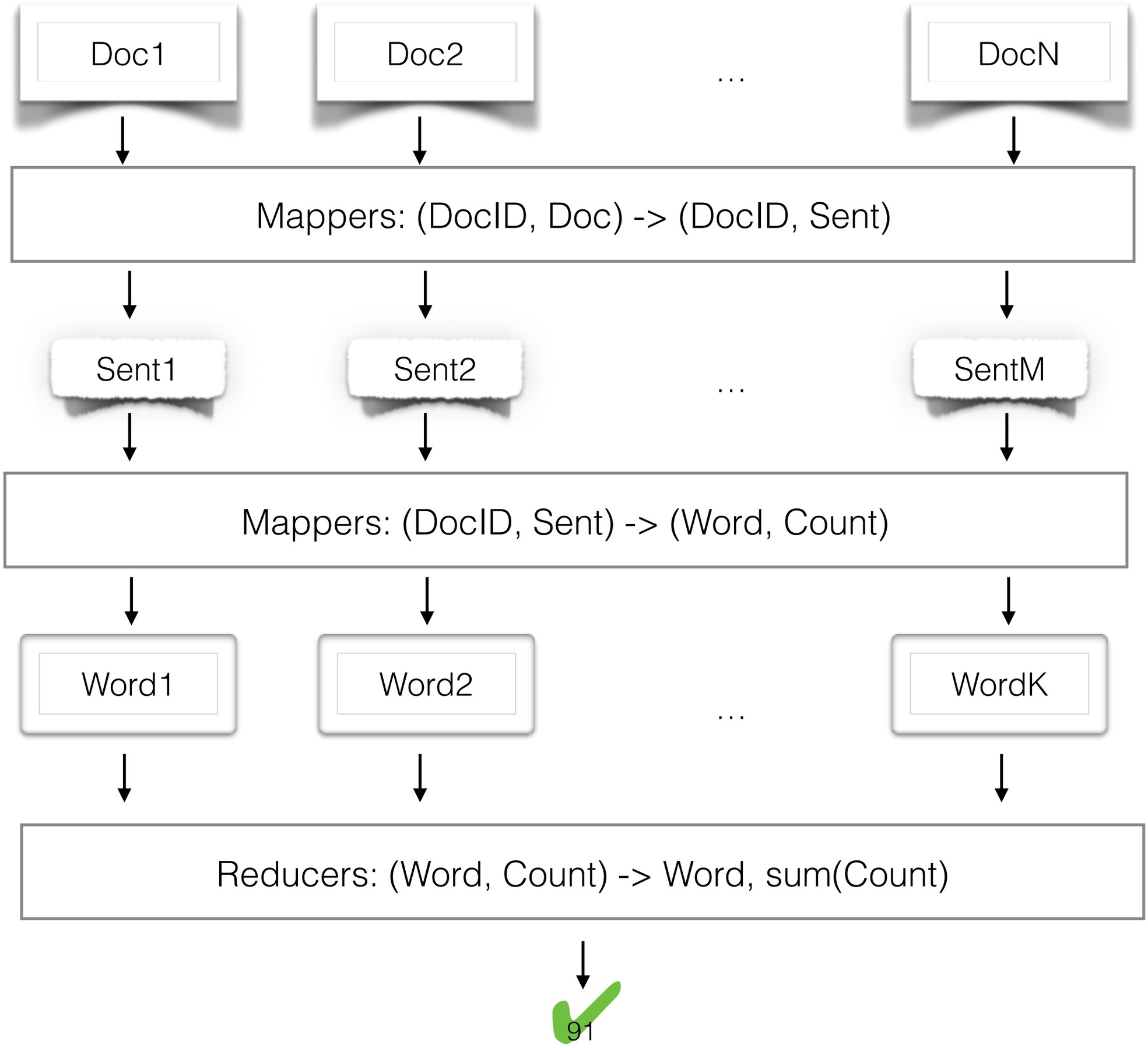
```
// rekey table by entity
def MapFn1: (String, Obj) -> (String, Obj) {
    emit(input.value().entity(), input.value())
}

// rekey table by subject
def MapFn2: (String, Obj) -> (String, Obj) {
    emit(input.value().subject(), input.value())
}

// define your pipeline
def main() {
Table<String, Obj> cats = read(table1_path).MapFn1()
Table<String, Obj> facts = read(table2_path).MapFn2()
output = cats.join(facts).MapFn3(. . .
```

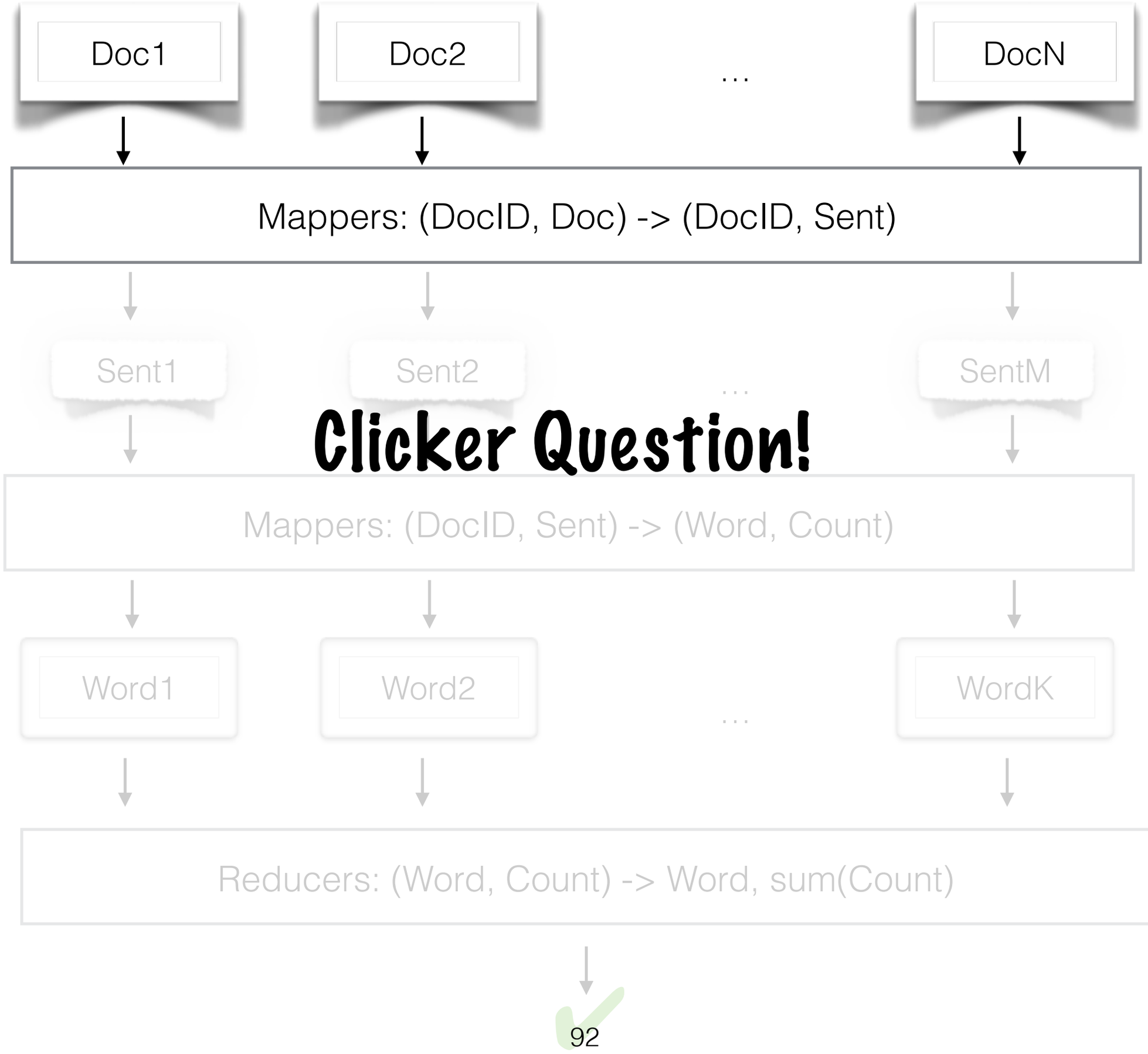
that entity

Key: String

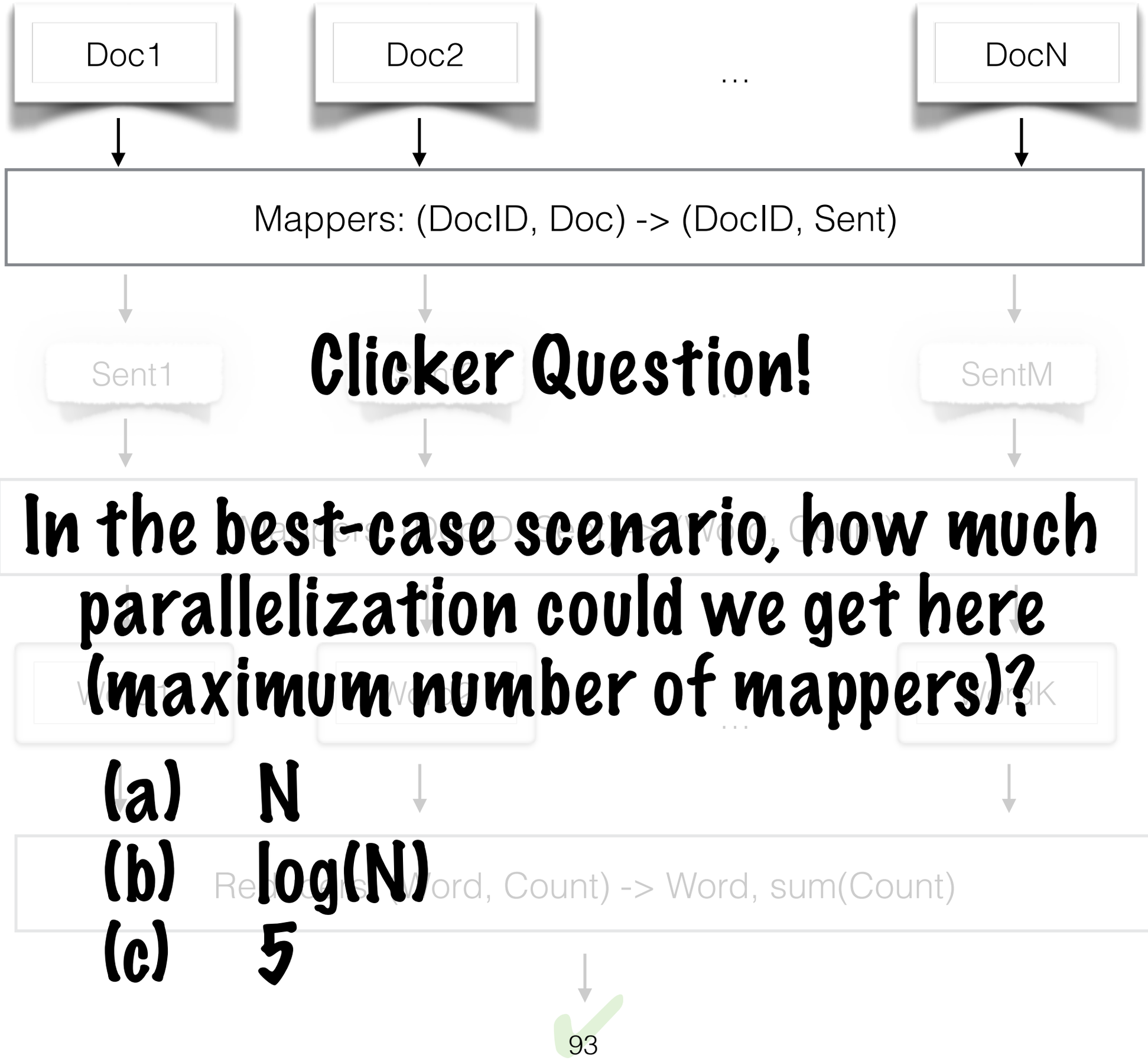
Value: (list_of((String, String, String), **list_of((String, String)**)

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)



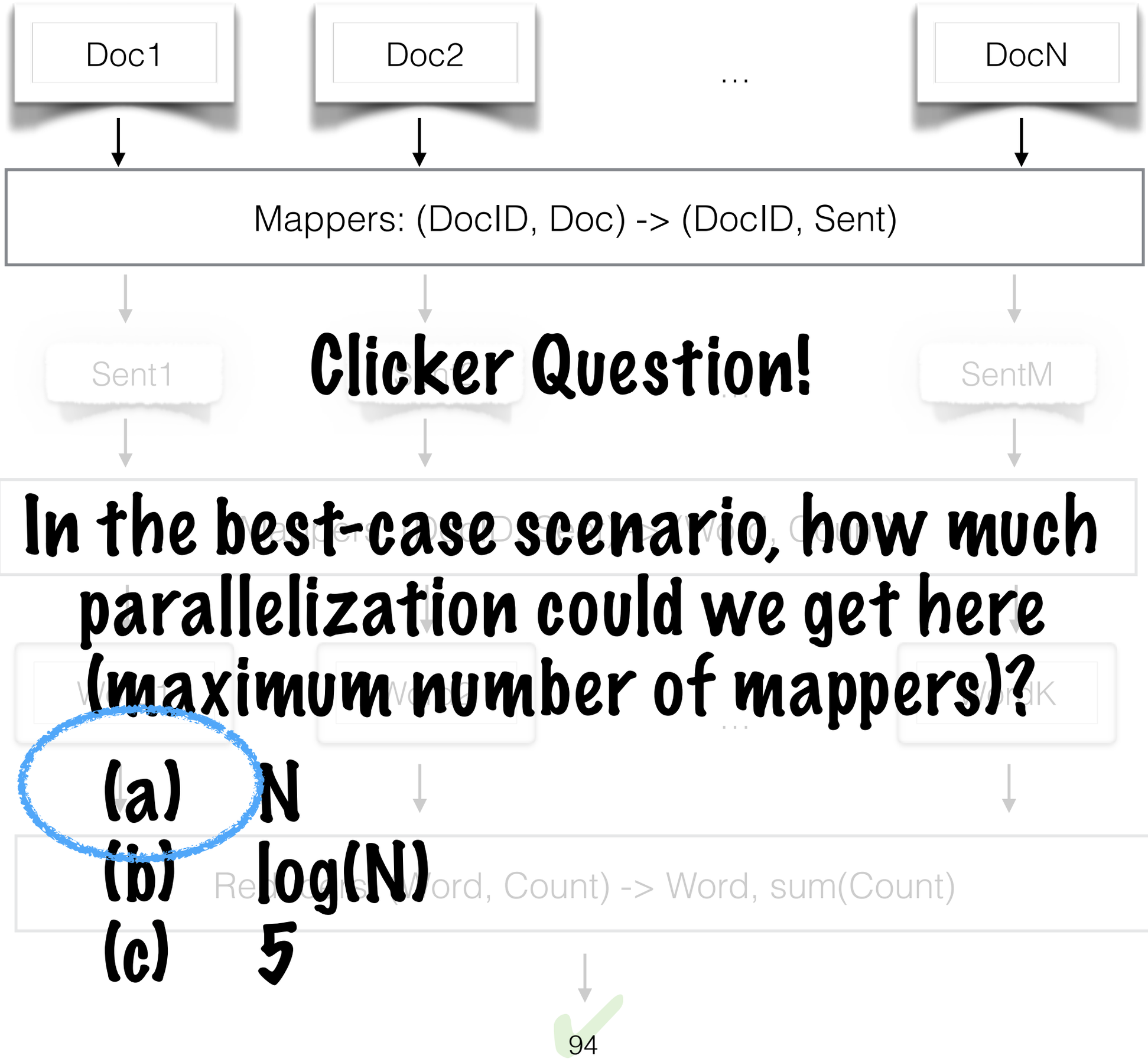
Clicker Question!

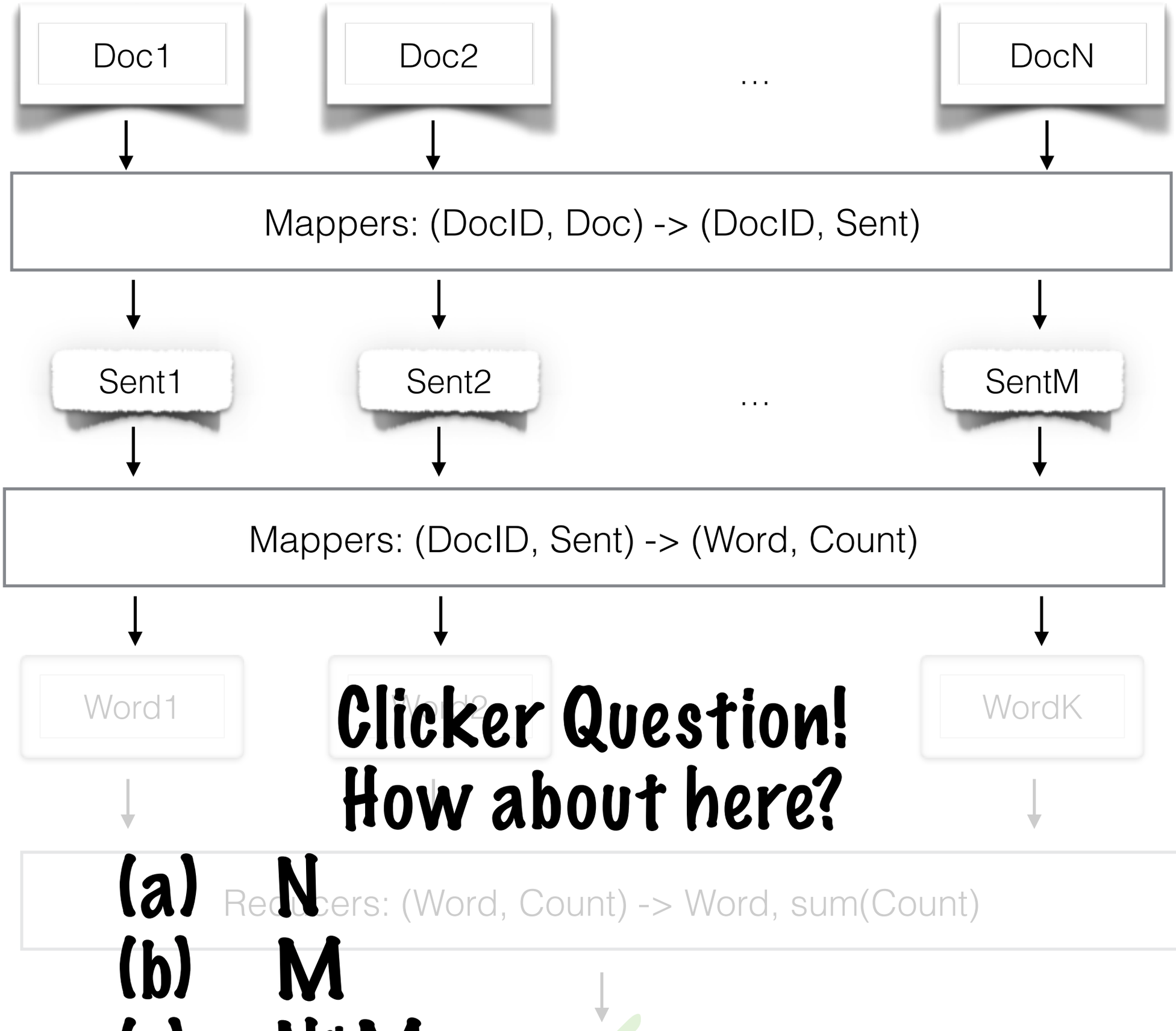


Clicker Question!

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

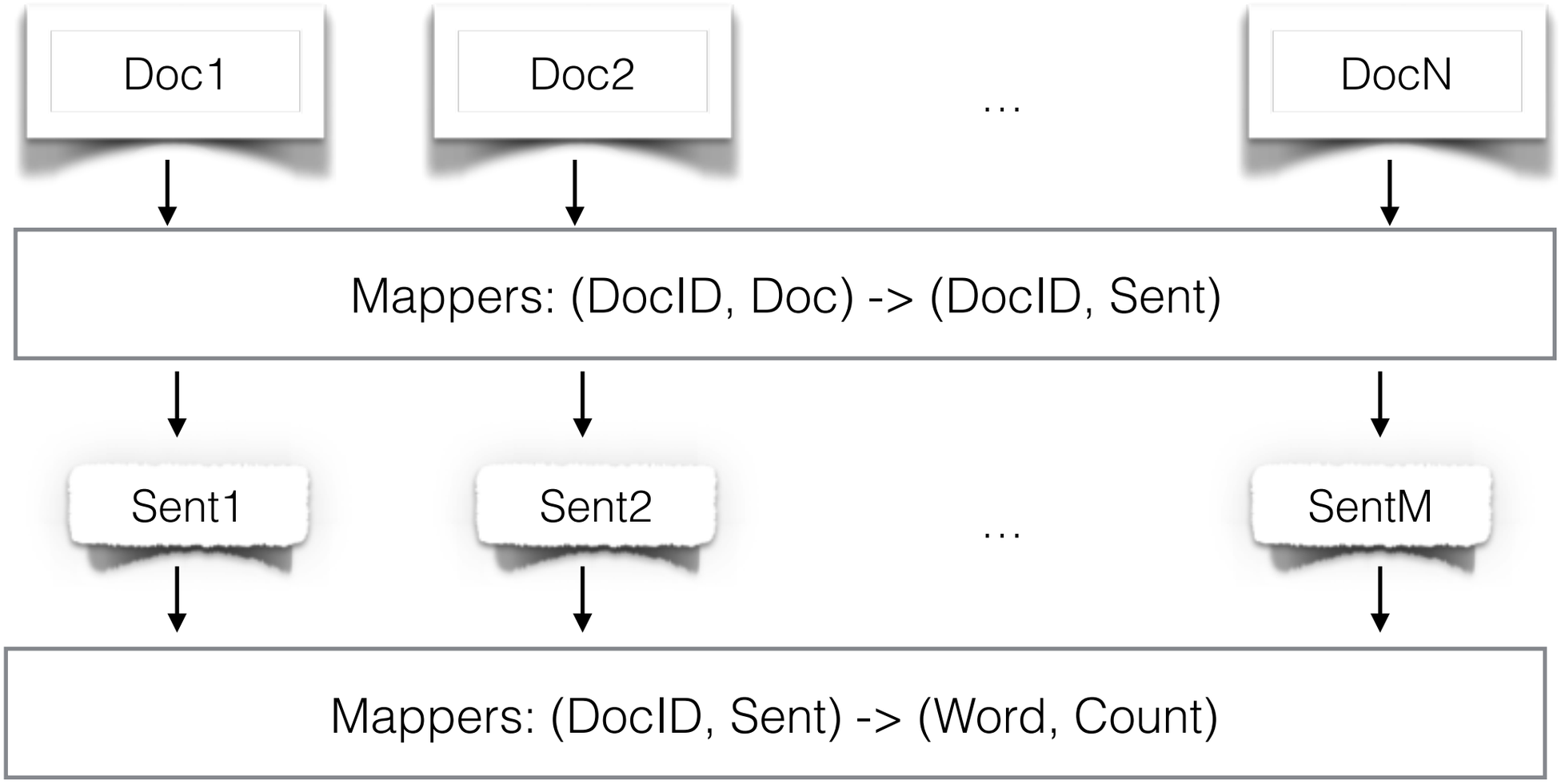
(a) N
(b) log(N)
(c) 5


Doc1
Doc2
…
DocN
Mappers: (DocID, Doc) -> (DocID, Sent)
Sent1
SentM
Clicker Question!
In the best-case scenario, how much parallelization could we get here (maximum number of mappers)?
(a) N
(b) log(N)
(c) 5



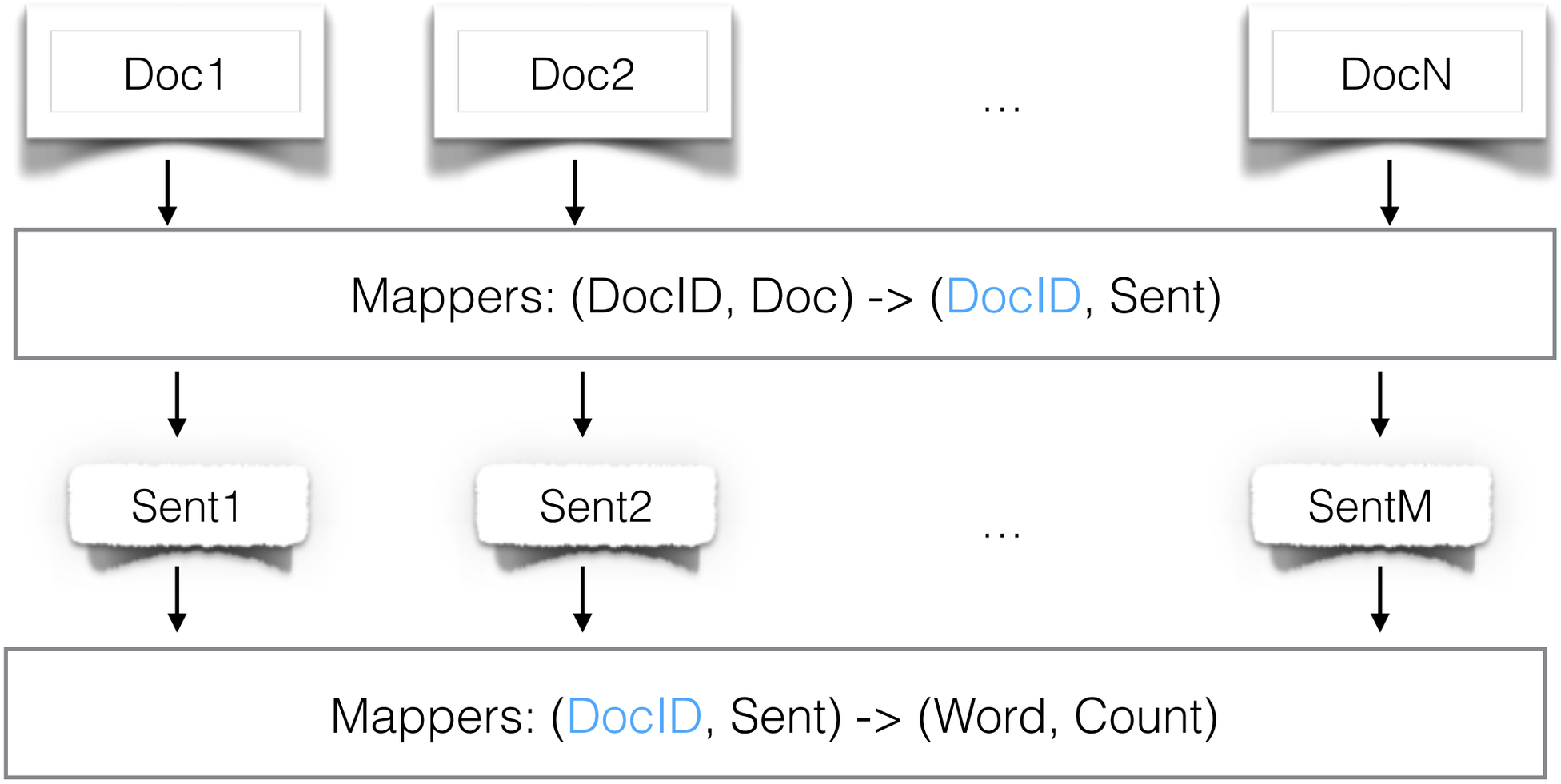
Clicker Question! How about here?

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



Clicker Question!
How about here?

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



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)

(b)

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

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

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

Ma
(DocID, Sent)

· Word,
t)

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)

↓

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

In general, nested loops should be refactored into multiple mappers

(b)

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

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

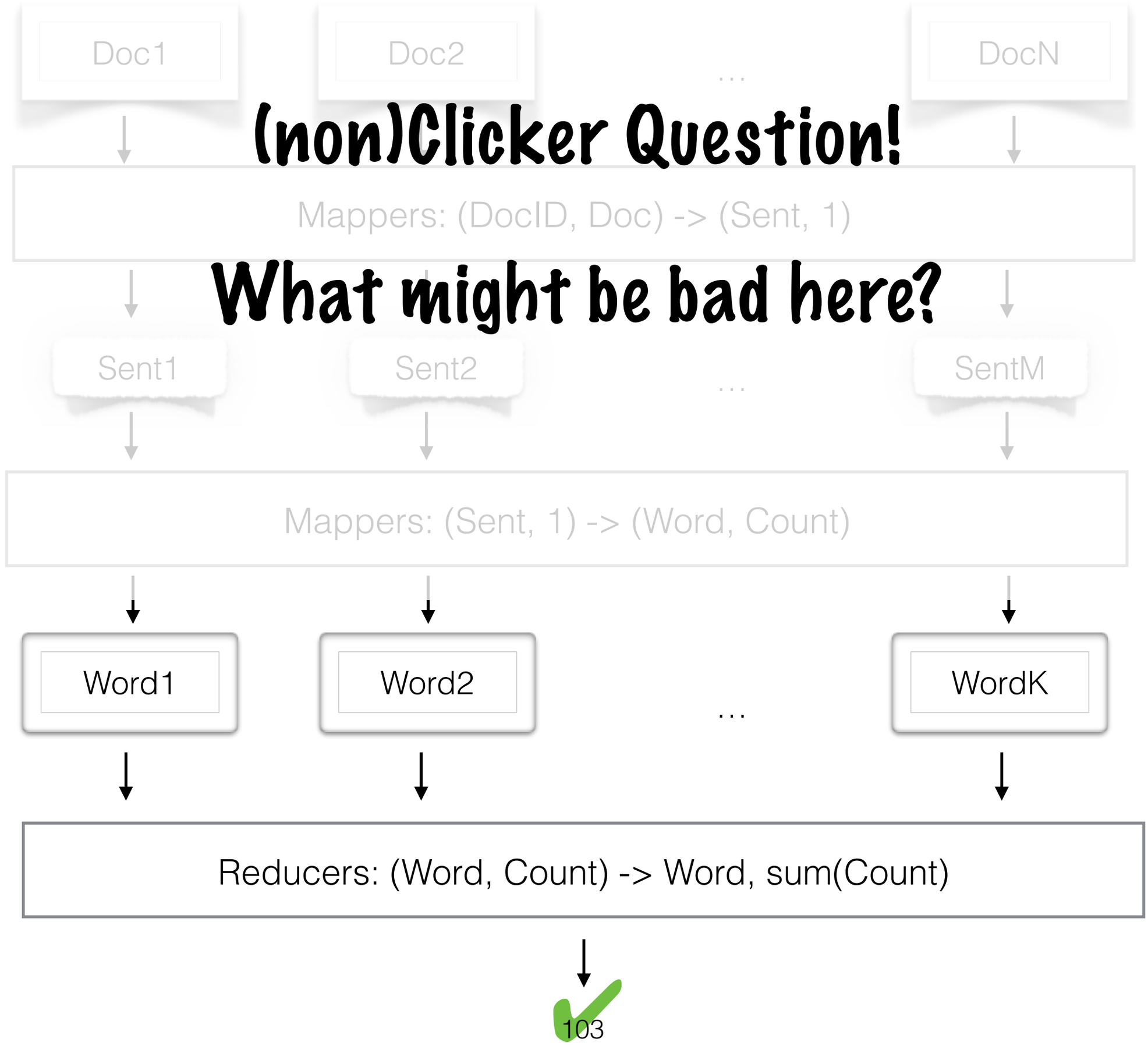
↓

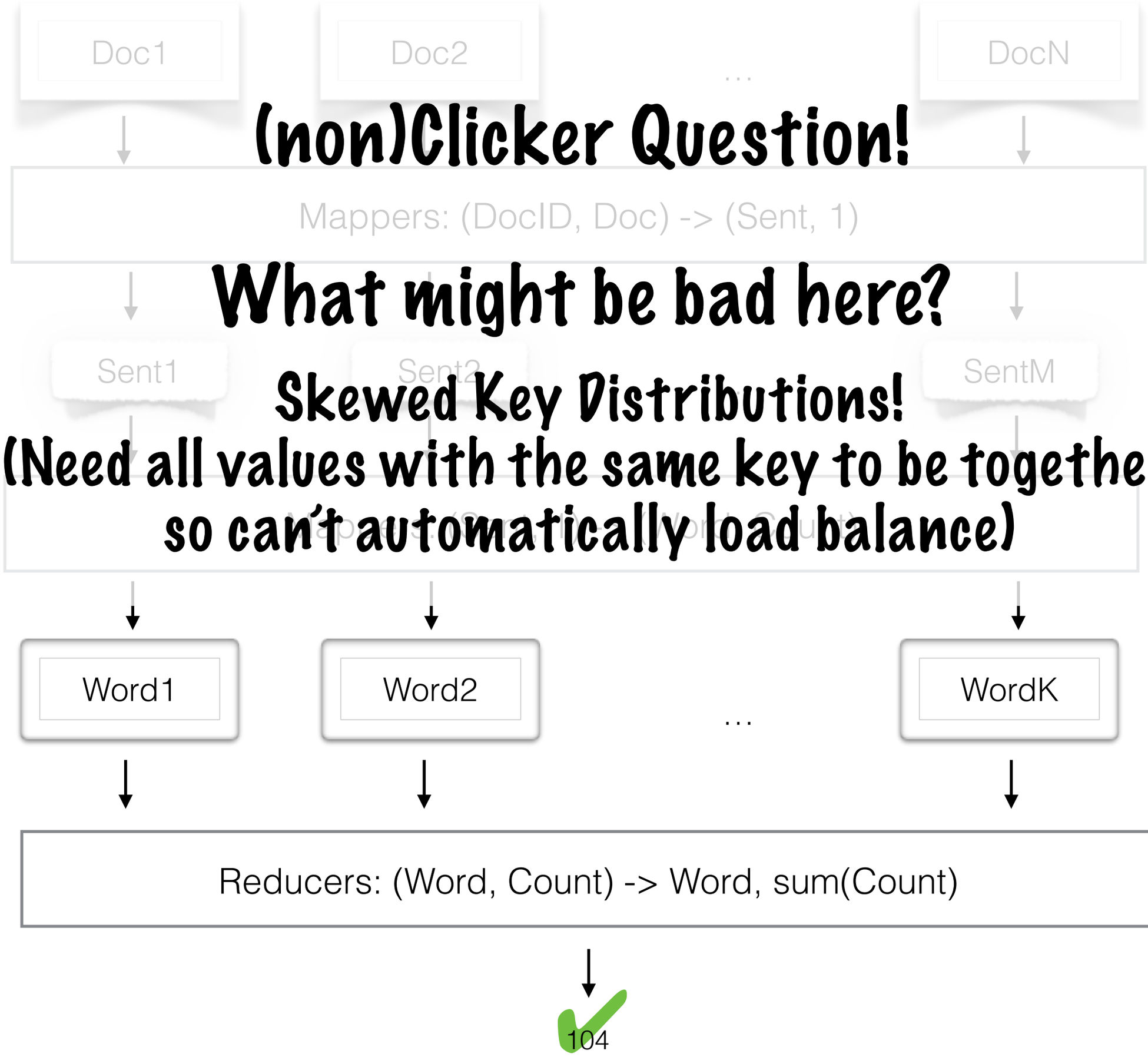
Reducer:
(Word, Count) -> Word,
sum(Count)

(c) They are the same

(non)Clicker Question!

What might be bad here?





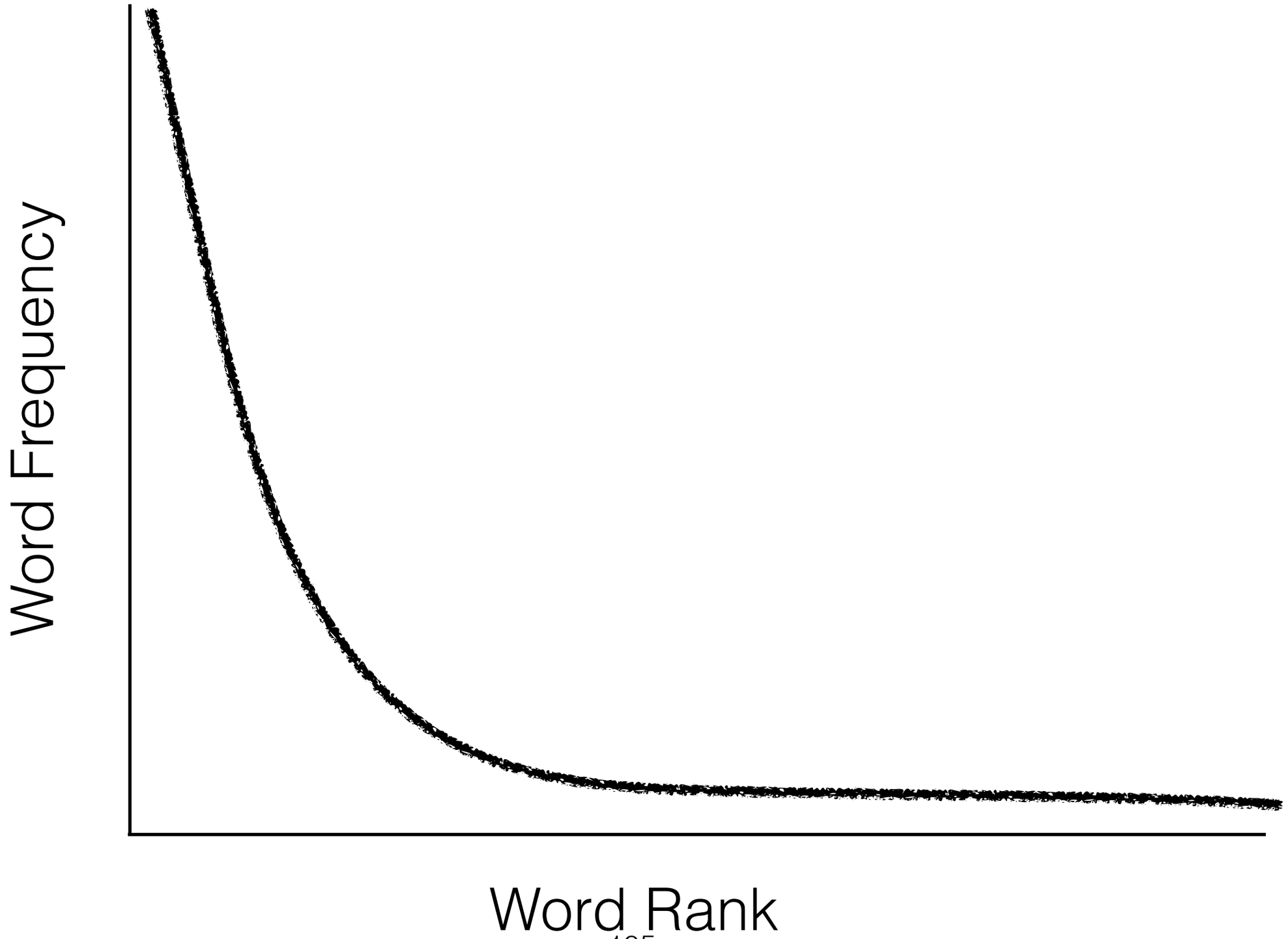
(non)Clicker Question!

What might be bad here?

Skewed Key Distributions!

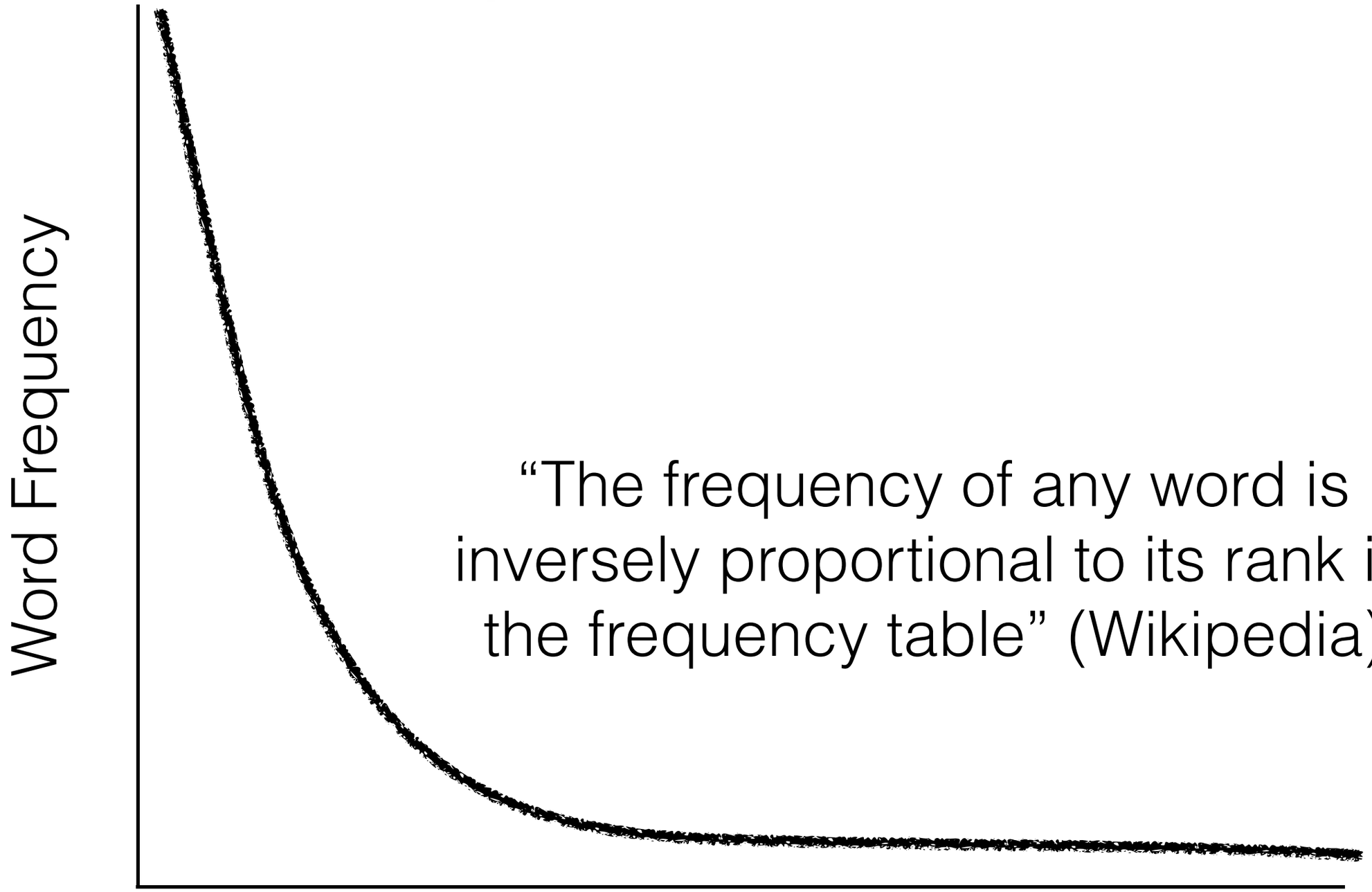
(Need all values with the same key to be together, so can't automatically load balance)

Zipf's Law



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

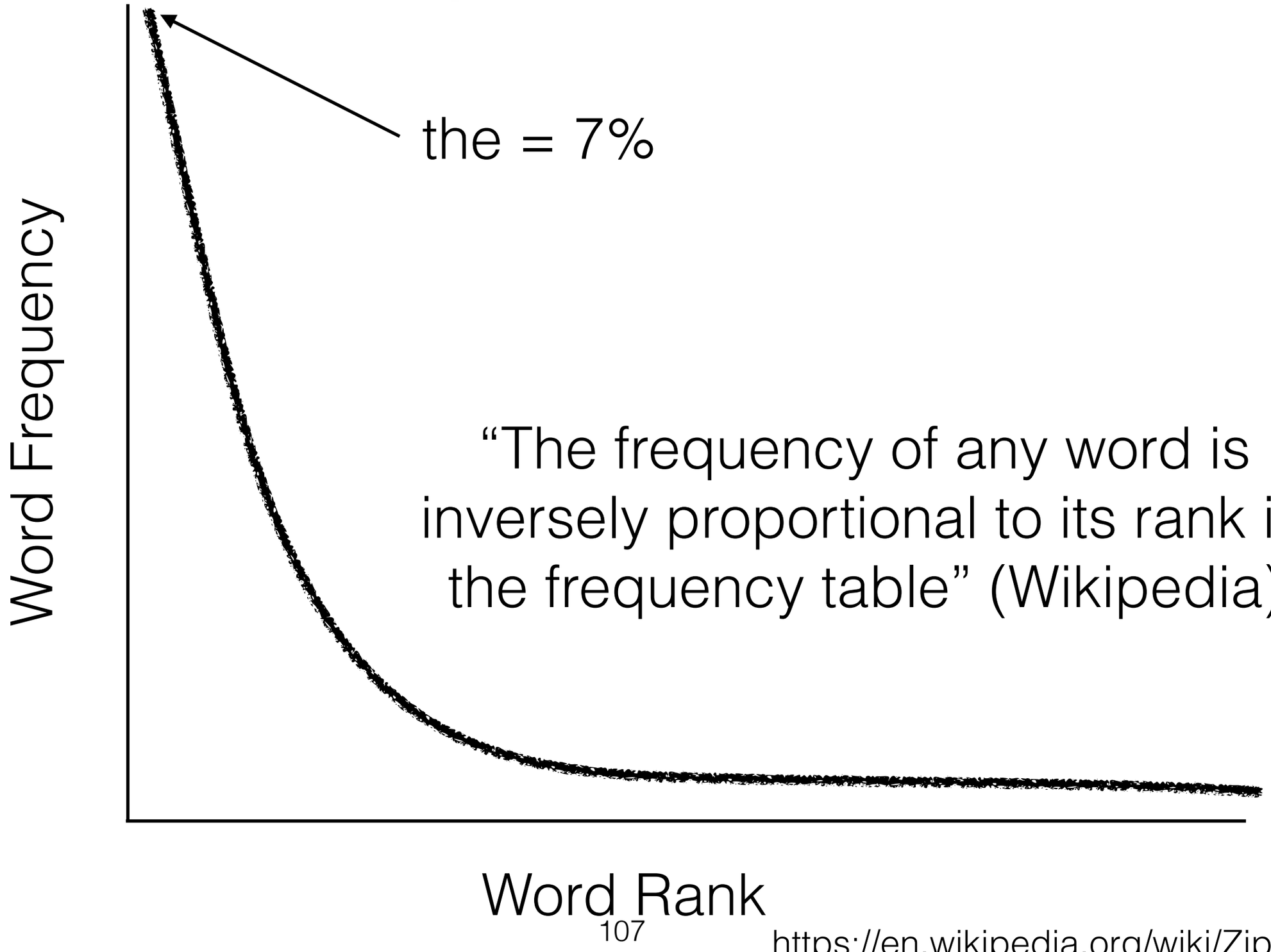
Zipf's Law



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

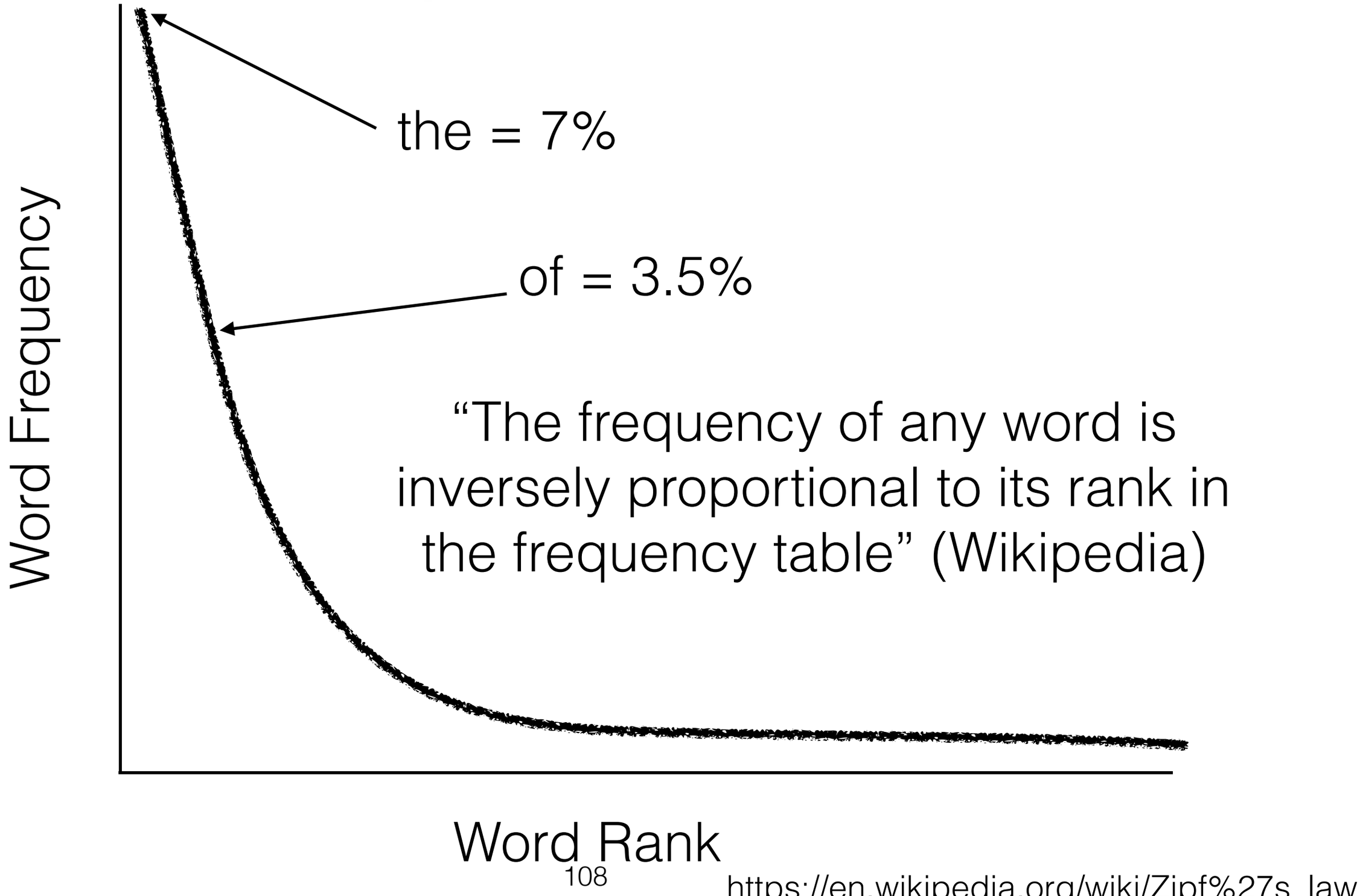
https://en.wikipedia.org/wiki/Zipf%27s_law

Zipf's Law



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

Zipf's Law



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 single 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

ok ok ok go go go.

enjoy the long weekend!