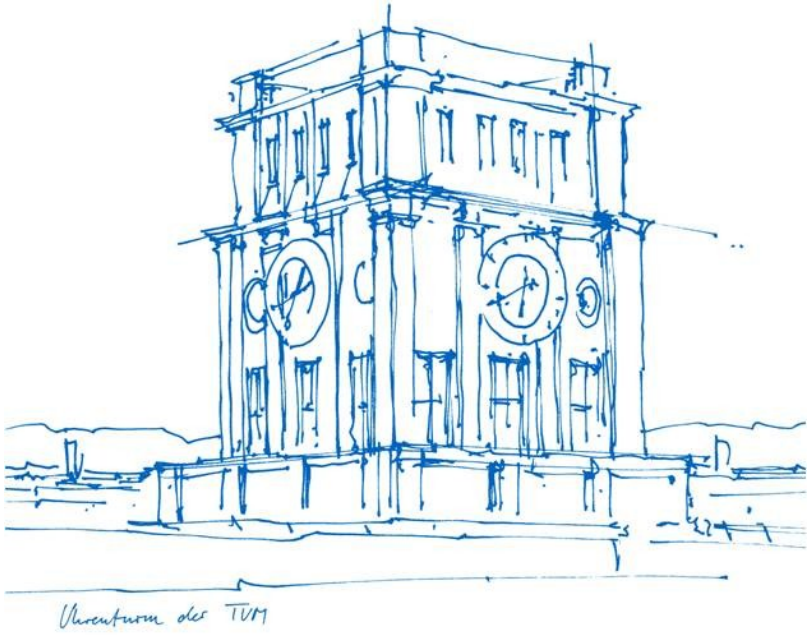
# Imlab: Dremel

Andreas Zimmerer

Technische Universität München

Garching, 27. Januar 2020



# Overview

- Why Dremel?
- How does Dremel Work?
- Example: Record Assembly
- Challenges
- Benchmarks
- Outlook

# Why Dremel?

- It’s a good idea to support non-relational data:
  Alternative would be normalizing data and joining the parts
  -> might be very costly
- Dremel utilizes the benefits of column stores for nested data
- Dremel is highly parallelizable

# How does Dremel Work?

Shredding documents into columns:

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

| DocId0 | r | d |
|---|---|---|
| 10 | 0 | 0 |

| L.Backward | r | d |
|---|---|---|
| NULL | 0 | 1 |

| L.Forward | r | d |
|---|---|---|
| 20 | 0 | 2 |
| 40 | 1 | 2 |
| 60 | 1 | 2 |

| N.L.Code | r | d |
|---|---|---|
| en-us | 0 | 2 |
| en | 2 | 2 |
| NULL | 1 | 1 |
| en-gb | 1 | 2 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |

| N.Url | r | d |
|---|---|---|
| http://A | 0 | 2 |
| http://B | 1 | 2 |
| NULL | 1 | 1 |

How do we compute repetition and definition levels?

# How does Dremel Work?

Computing definition and repetition levels:

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

The *definition level* is always fixed for a field, and is used for specifying which field in the message was NULL.

The definition level of a field is the number of potentially undefined (optional and repeated) fields in the path.

The *repetition level* tells us at which level a field is repeated.

The maximum repetition level of a field is the number of repeated fields in the path.

=> with these two values we can losslessly reconstruct records

# How does Dremel Work?

Retrieving a record: (1) Create a FSM with the fields

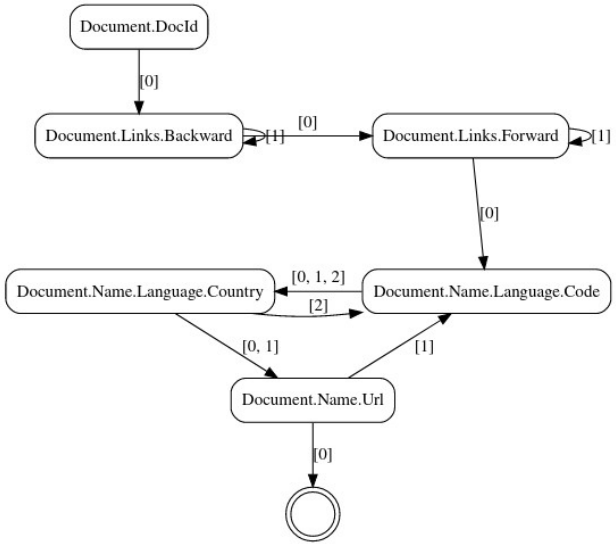
Question: How can we reconstruct a (partial) record?

=> We construct a FSM that defines how we jump between fields.
e.g. given a previous field and a repetition level, what‘s the next field?

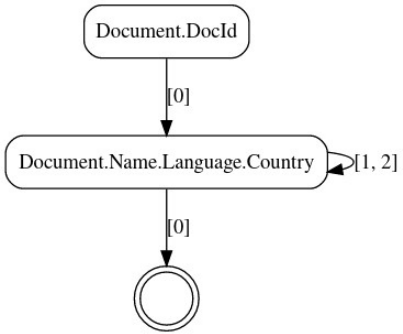
We can also construct a FSM that constructs only a partial record, but still preserves the record structure.

# How does Dremel Work?

Full FSM with all fields




Partial FSM for DocId and Country



# How does Dremel Work?

Retrieving a record: (2) Assemble record by jumping from field to field with FSM

Example: Record Assembly

# Example: Record Assembly

Three Ingredients:

(1) Document Structure

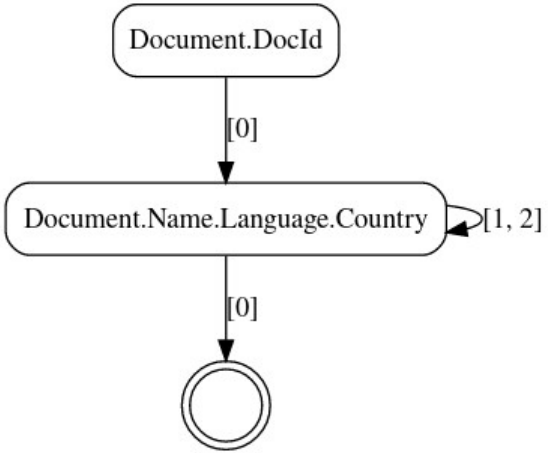
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

(2) Record Data

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |

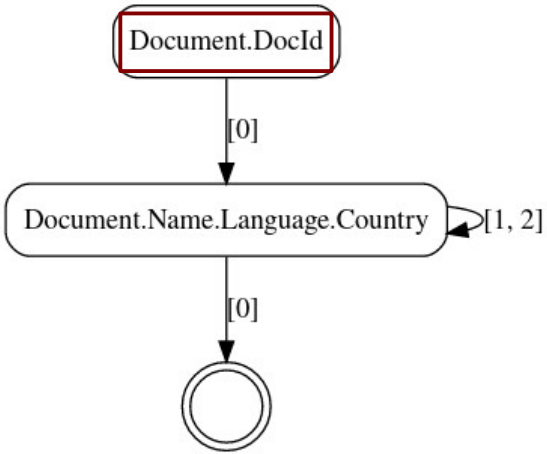
(3) FSM



# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |

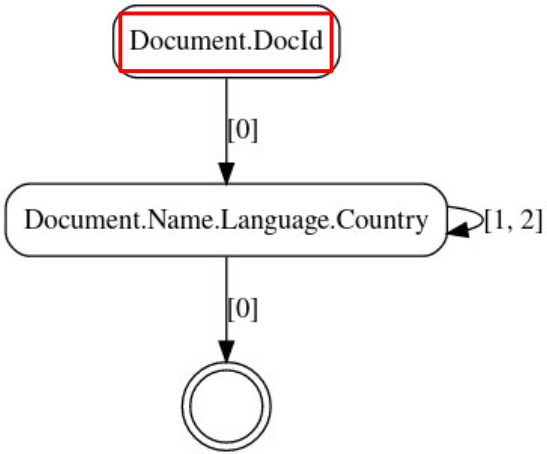



Assembled Record:

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



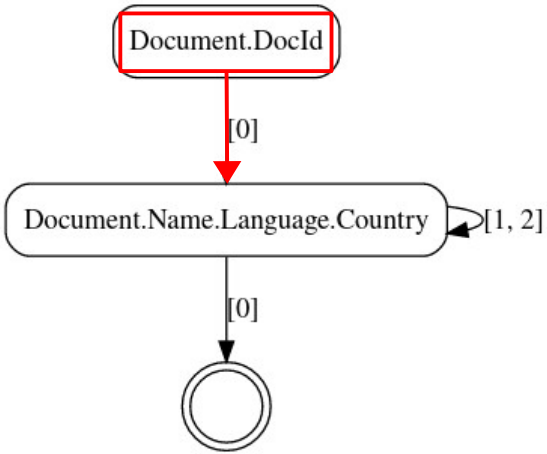

Assembled Record:

DocId: 10

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



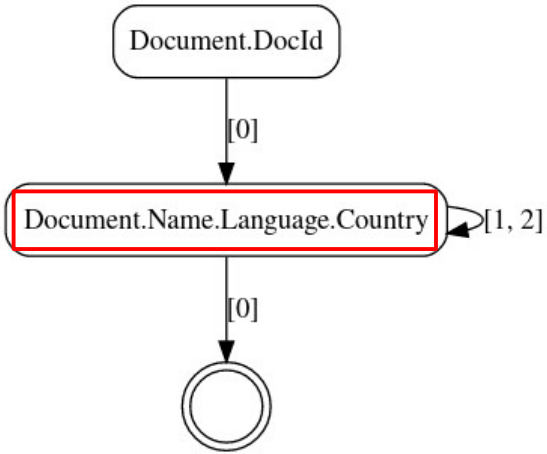

Assembled Record:

DocId: 10

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



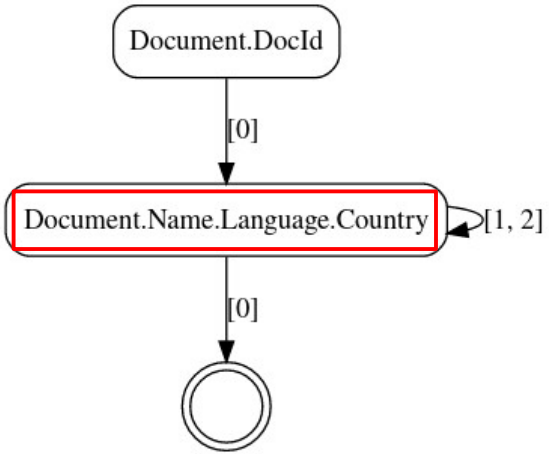

Assembled Record:

DocId: 10

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



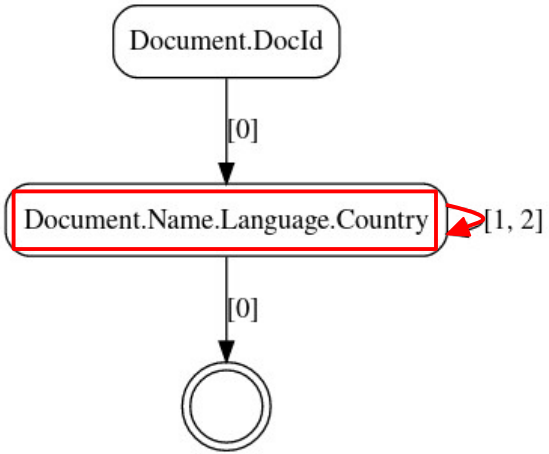

Assembled Record:

DocId: 10
Name: {
  Lang: {
    Country: 'us'

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



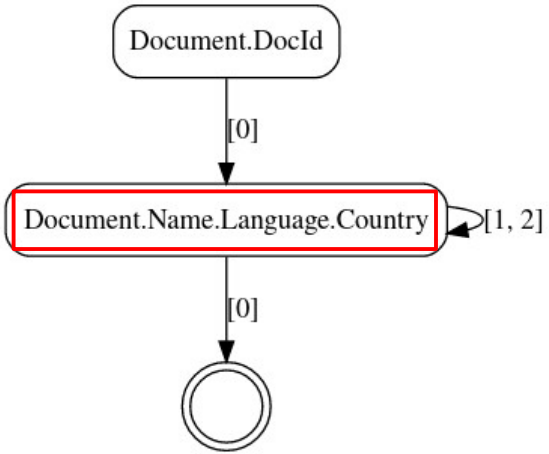

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



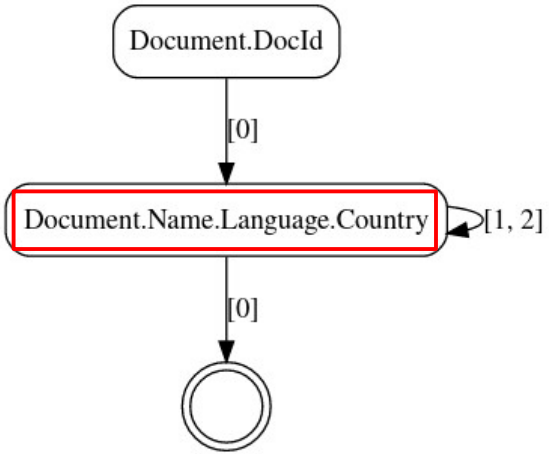

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



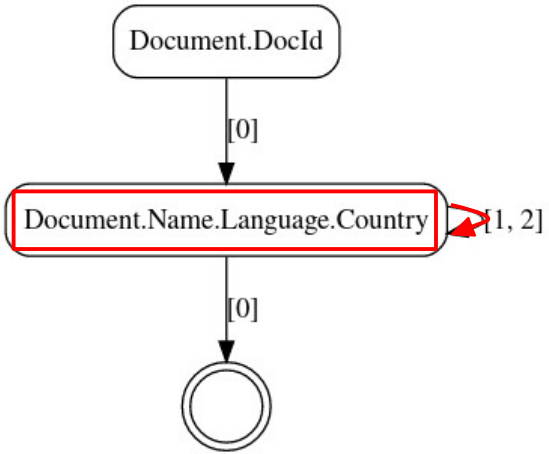

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: {
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



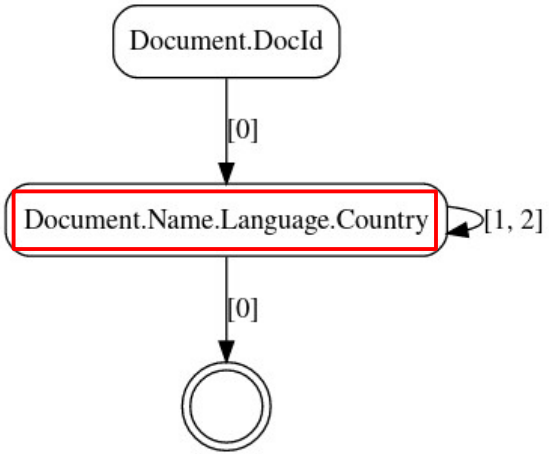

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



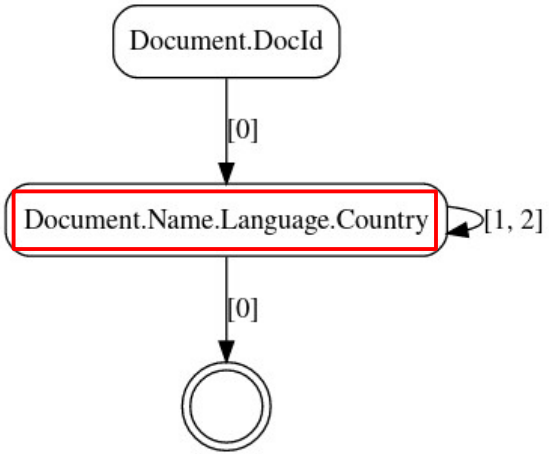

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



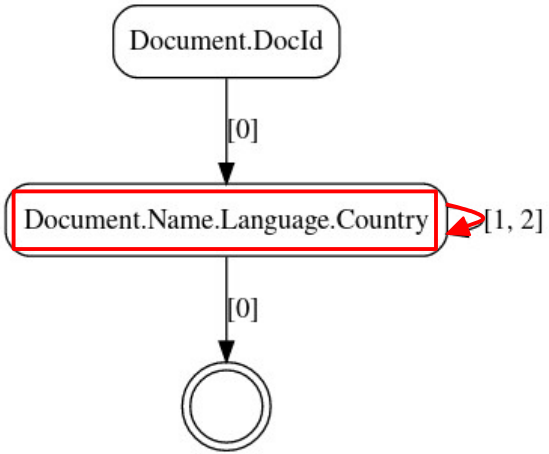

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



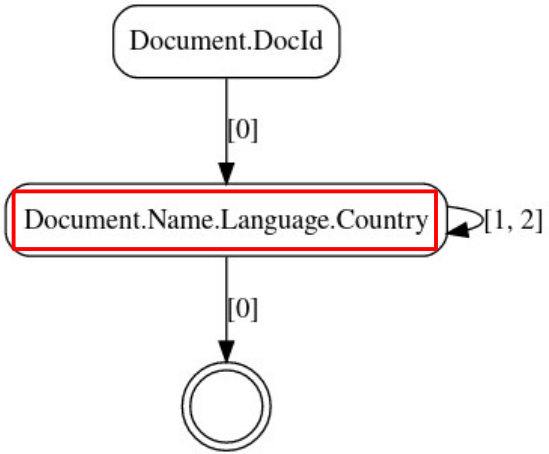

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



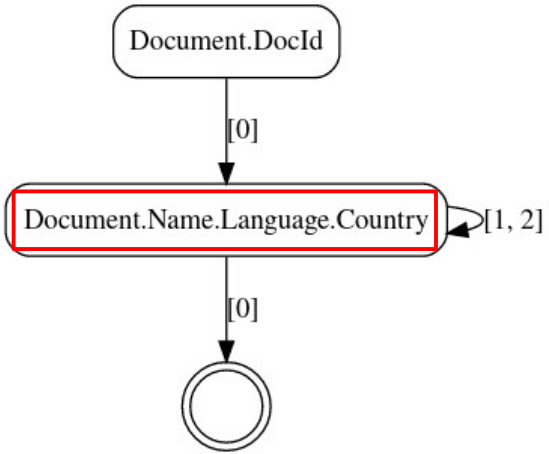

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



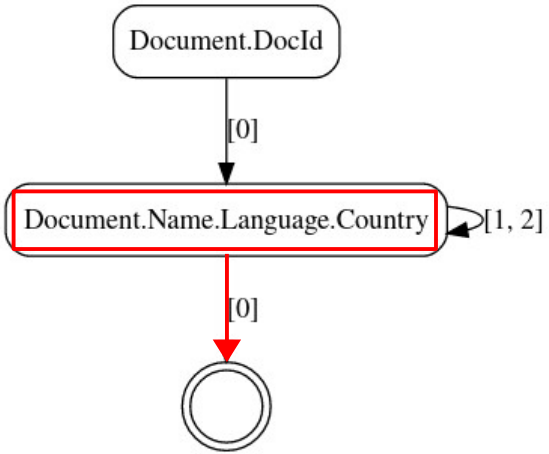

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
Name: {
    Lang: {
        Country: 'gb'
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |



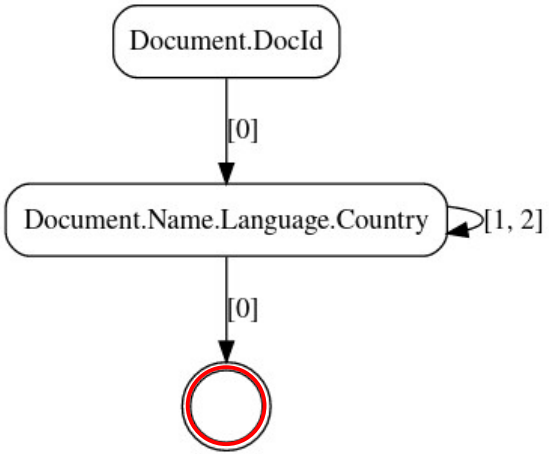

Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
Name: {
    Lang: {
        Country: 'gb'
```

# Example: Record Assembly

| DocId | r | d |
|---|---|---|
| 10 | 0 | 0 |
| ... | 0 | 0 |

| N.L.Country | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| ... | 0 | 3 |




Assembled Record:

```
DocId: 10
Name: {
    Lang: {
        Country: 'us'
    }
    Lang: { }
}
Name: {
    Lang: {
        Country: 'gb'
    }
}
```

# Challenges

# FSM Construction

```
1  procedure ConstructFSM(Field[] fields):
2  for each field in fields:
3    maxLevel = maximal repetition level of field
4    barrier = next field after field or final FSM state otherwise
5    barrierLevel = common repetition level of field and barrier
6    for each preField before field whose
7           repetition level is larger than barrierLevel:
8      backLevel = common repetition level of preField and field
9      Set transition (field, backLevel) -> preField
10   end for
11   for each level in [barrierLevel+1..maxLevel]
12         that lacks transition from field:
13     Copy transition's destination from that of level-1
14   end for
15   for each level in [0..barrierLevel]:
16     Set transition (field, level) -> barrier
17   end for
18 end for
19 end procedure
```

# FSM Construction

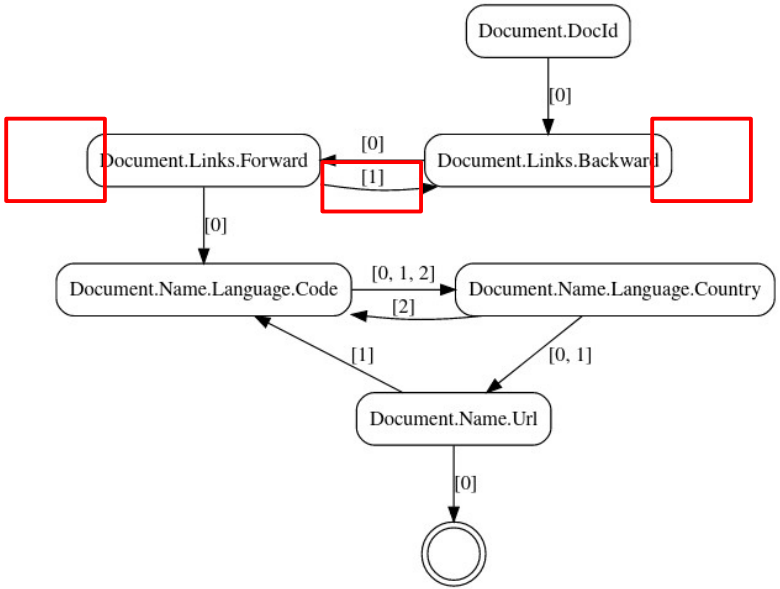
```
1  procedure ConstructFSM(Field[] fields):
2  for each field in fields:
3    maxLevel = maximal repetition level of field
4    barrier = next field after field or final FSM state otherwise
5    barrierLevel = common repetition level of field and barrier
6    for each preField before field whose
7           repetition level is larger than barrierLevel:
8      backLevel = common repetition level of preField and field
9      Set transition (field, backLevel) -> preField
10   end for
11   for each level in [barrierLevel+1..maxLevel]
12         that lacks transition from field:
13     Copy transition's destination from that of level-1
14   end for
15   for each level in [0..barrierLevel]:
16     Set transition (field, level) -> barrier
17   end for
18 end for
19 end procedure
```

No

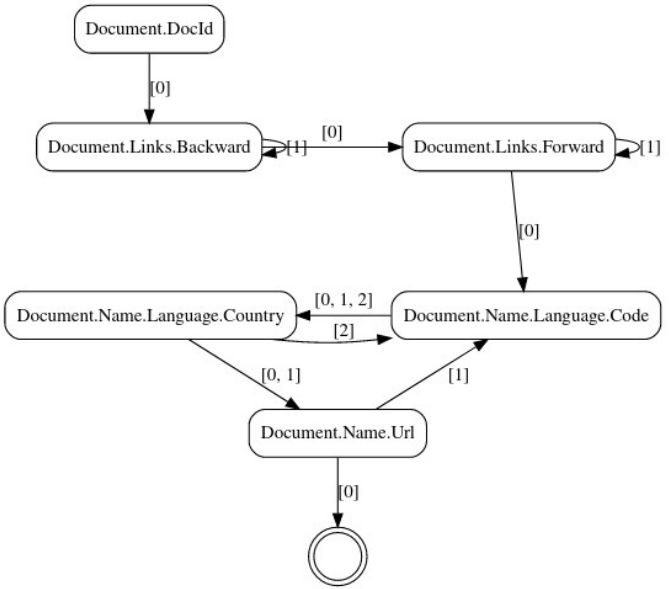
Instead: Insert reflexive edge

# FSM Construction: Algorithms Compared

Version from Paper:




Modified Version:



# Common Ancestor

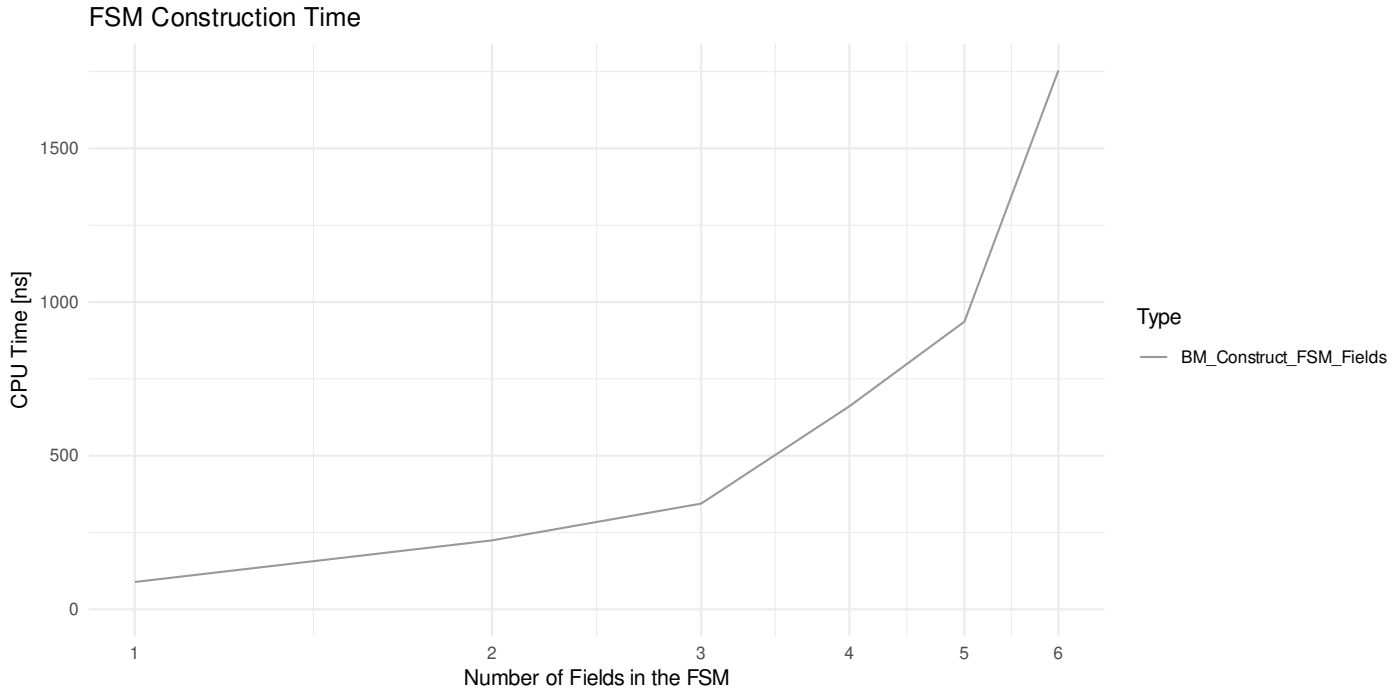
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

Order is important!

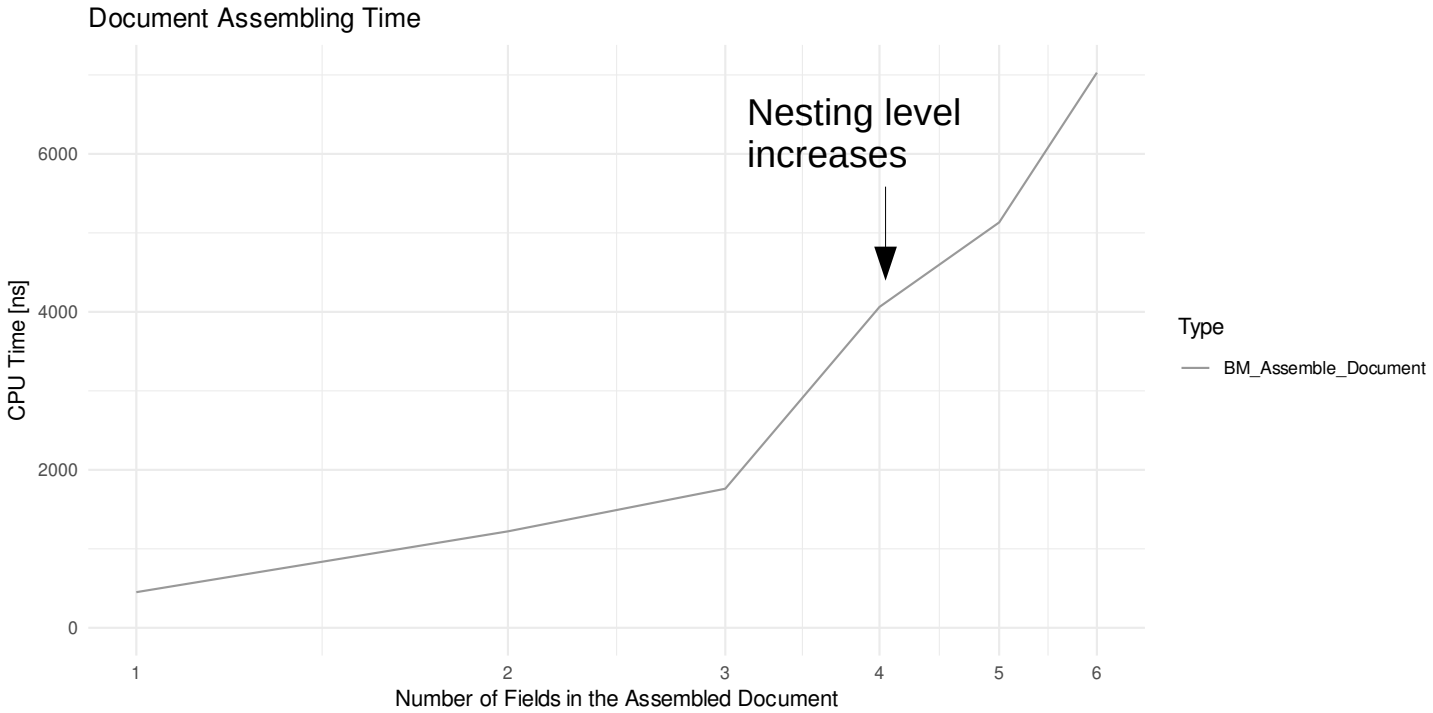
CoAn of Code and Country -> Language
CoAn of Country and Code -> Name

# Benchmarks

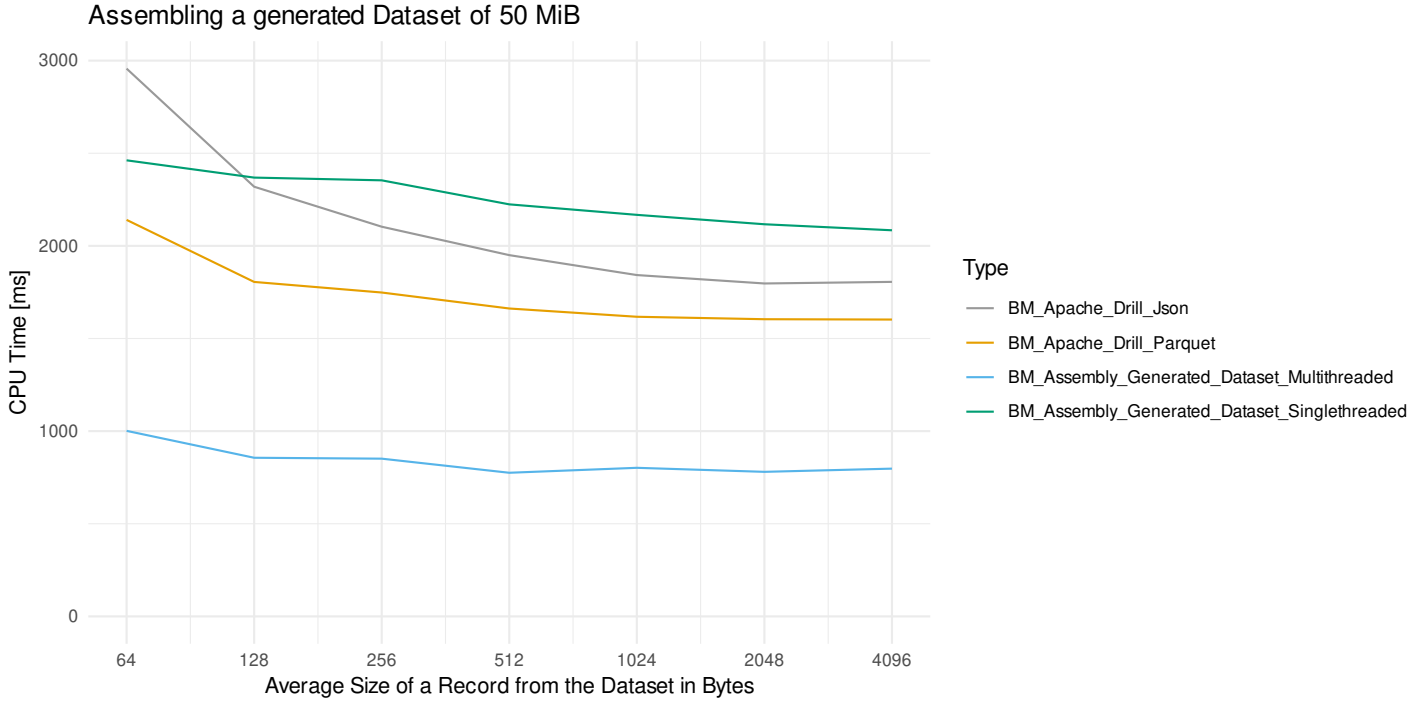
# Benchmarks


FSM Construction Time
CPU Time [ns]
1500
1000
500
0
1
2
3
4
5
6
Number of Fields in the FSM
Type
BM_Construct_FSM_Fields

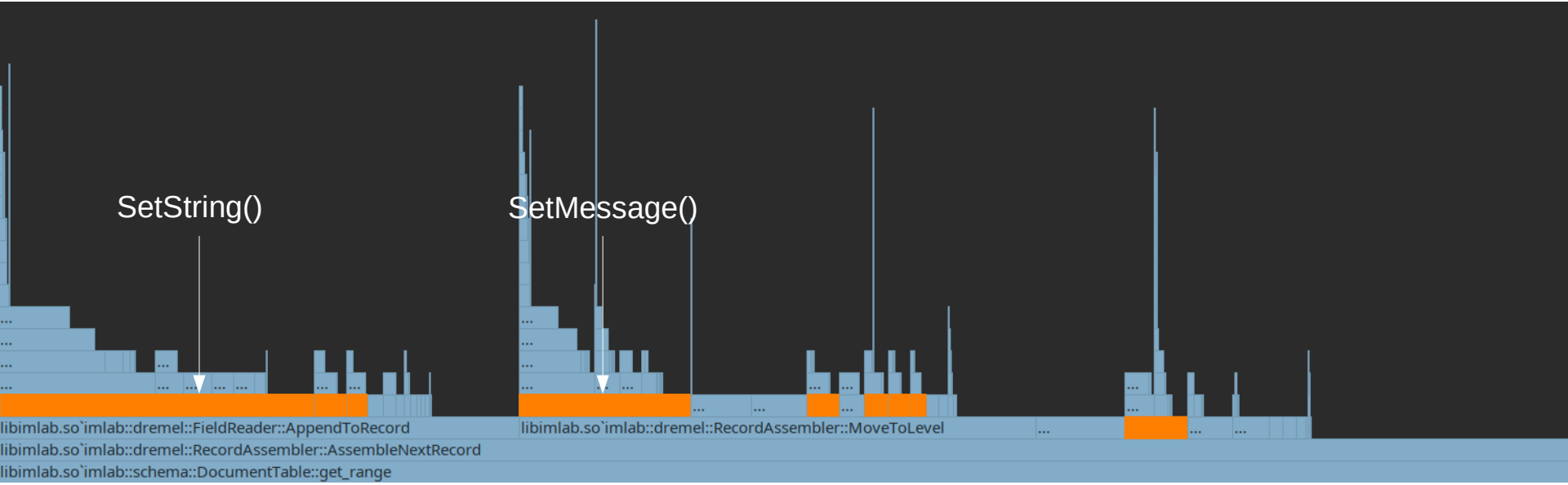
# Benchmarks


Document Assembling Time
Nesting level increases
CPU Time [ns]
0
2000
4000
6000
1
2
3
4
5
6
Number of Fields in the Assembled Document
Type
BM_Assemble_Document

# Performance against Apache Drill



# Profiling




Time spent with Protobuf

# Summary & Outlook

- FSM construction is fast and only needed once (=> neglectable overhead)
- Record assembly takes a lot of time (copies & heap allocs with Proto)

- Compilation instead of interpretation
- Not using Protobuf