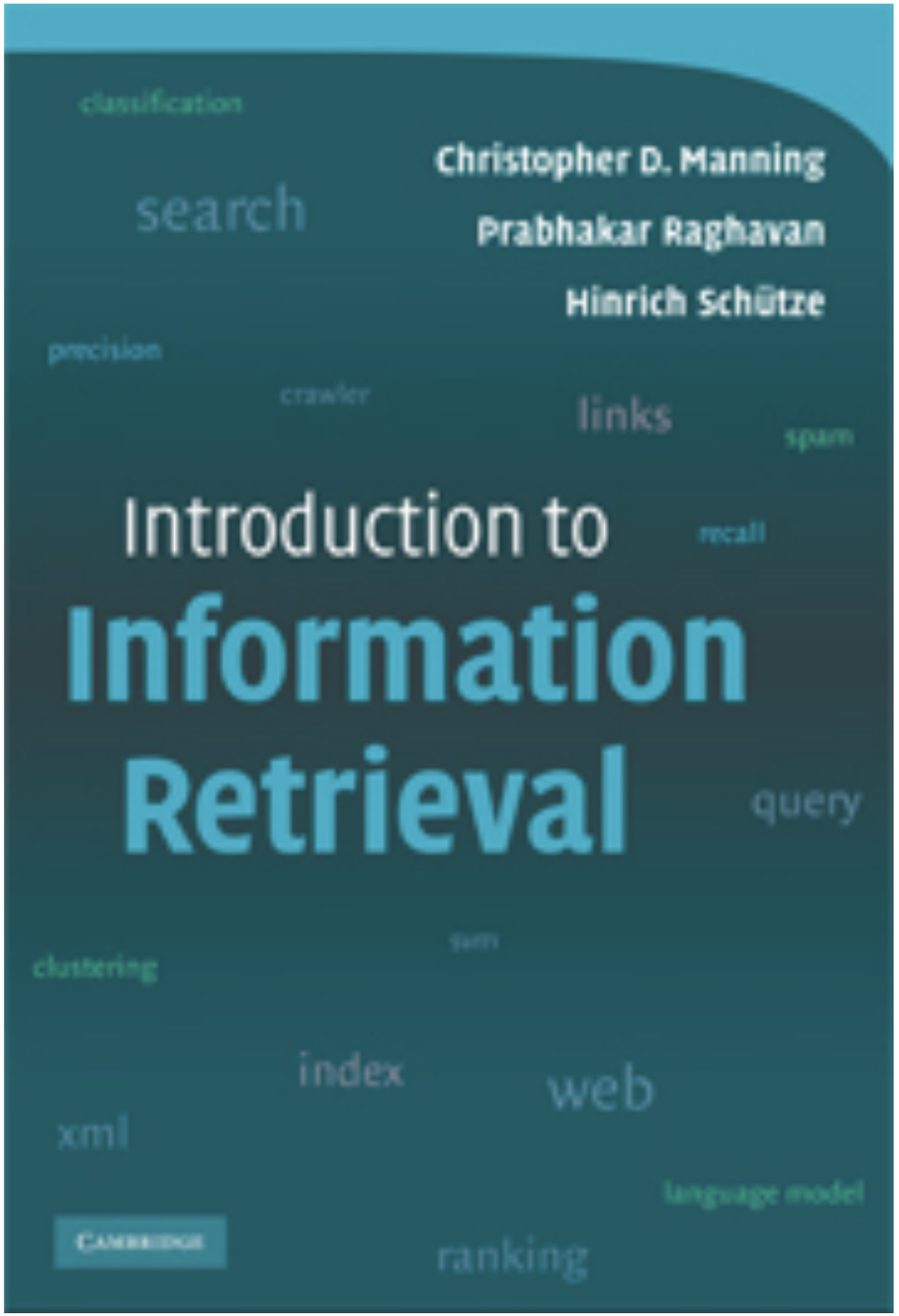
# Scoring for human beings

Britta Weber

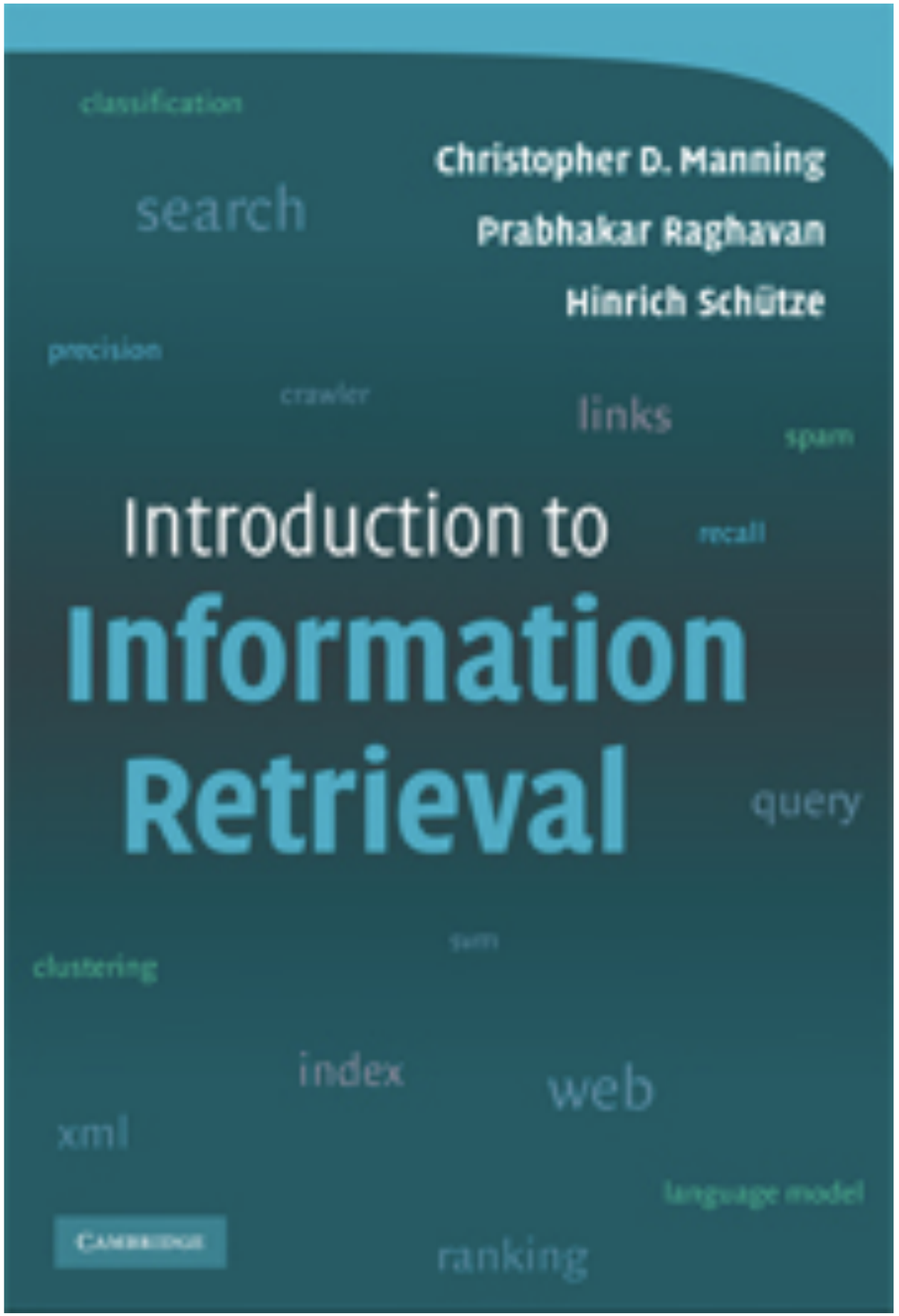
elasticsearch


classification
Christopher D. Manning
search
Prabhakar Raghavan
Hinrich Schütze
precision
crawler
links
spam
Introduction to
recall
Information
Retrieval
query
clustering
index
web
xml
language model
ranking

# What is scoring?

Determine the relevance of a document given a search request

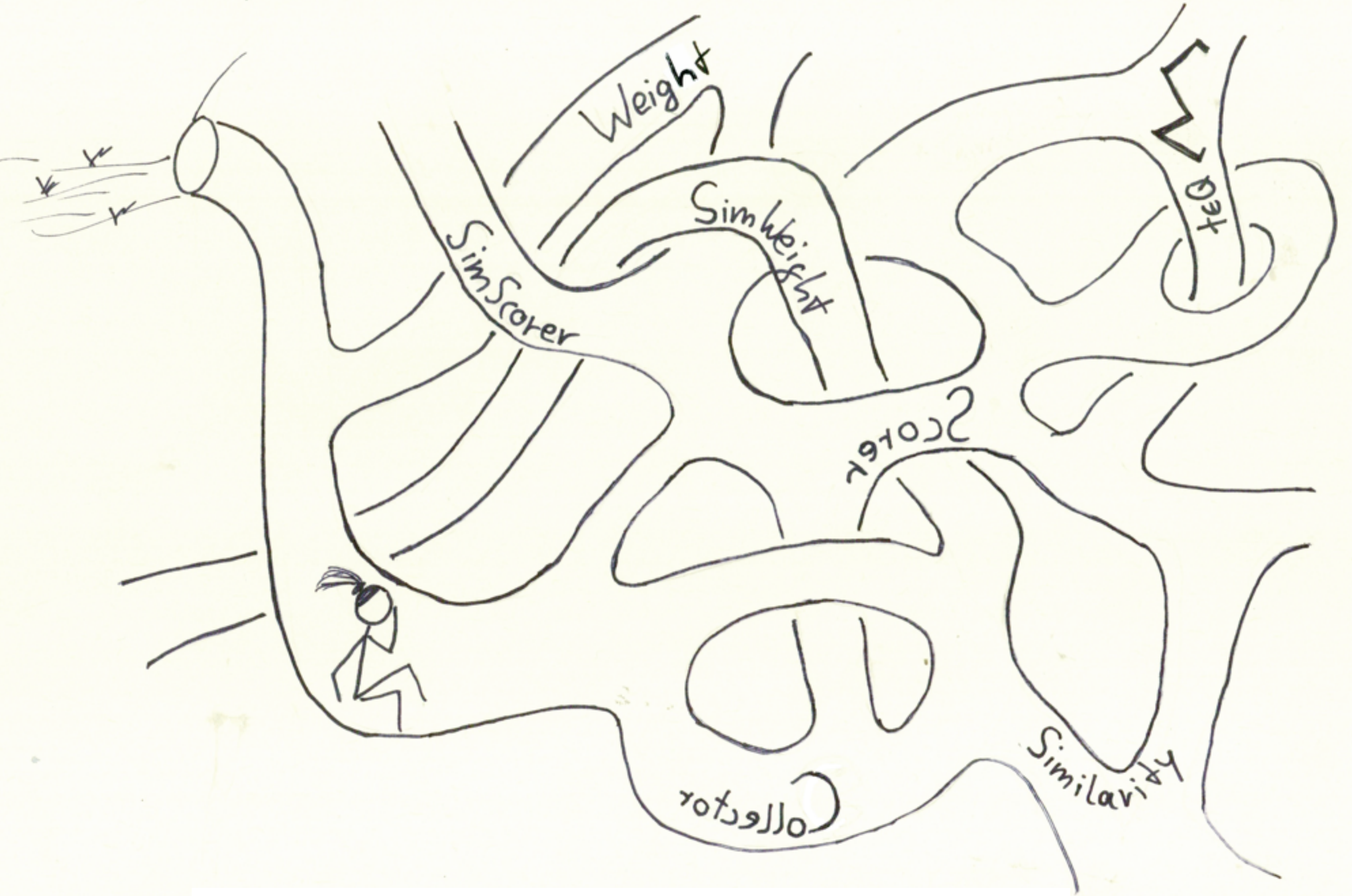
- Given keywords `["football", "world cup"]`, what is the most relevant news article the user might want to read?
- Given the criteria `["java", "expected income", "work location"]`, which candidate in the data set is most likely to be a good employee?


classification
Christopher D. Manning
search
Prabhakar Raghavan
Hinrich Schütze
precision
crawler
links
spam
Introduction to
recall
Information
Retrieval
query
clustering
index
web
xml
language model
ranking

# Hm. So how is this actually implemented?


*Style shamelessly adapted from xkcd.org*

# THE BOTTOMLESS PIT OF INDIRECTION




*Style shamelessly adapted from xkcd.org*

# The purpose of this talk

Relieve you of the burden to find the point where to get started!

1. Give an introduction in the theory
   - Bag-of-word model
   - Hint on where to look things up
2. How can you tweak scores with elasticsearch
   - How to use what is there
   - How to implement new things

# How does scoring of text work?

TF - IDF

# Relevancy - the vector space model

| Step | Query | Doc 1 | Doc 2 |
| --- | --- | --- | --- |
| The text | brown fox | The quick brown fox likes brown mice | The red fox |
| The terms | (brown, fox) | (brown, brown, fox, likes, mice, quick, the) | (fox, red, the) |
| A frequency vector | (1, 1) | (2, 1) | (0, 1) |
| Relevancy | - | 3? | 1? |

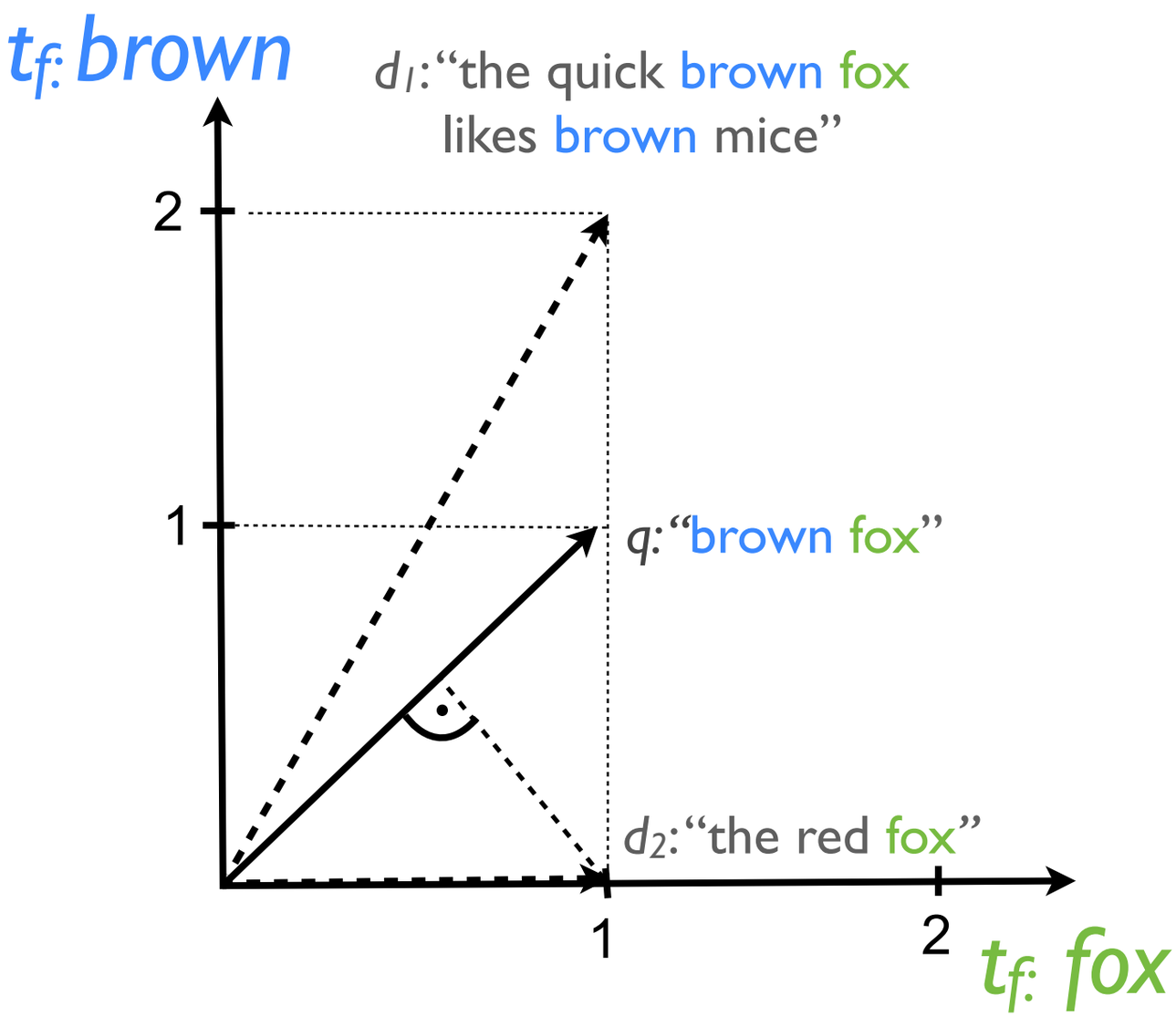
# Relevancy - the vector space model

| Step | Query | Doc 1 | Doc 2 |
|---|---|---|---|
| The text | brown fox | The quick brown fox likes brown mice | The red fox |
| The terms | (brown, fox) | (brown, brown, fox, likes, mice, quick) | (fox, red) |
| A frequency **vector** | (1, 1) | (2, 1) | (0, 1) |
| Relevancy | - | 3? | 1? |

# Relevancy - the vector space model

Queries and documents are *vectors*.

What is the *distance* between query and document vector?

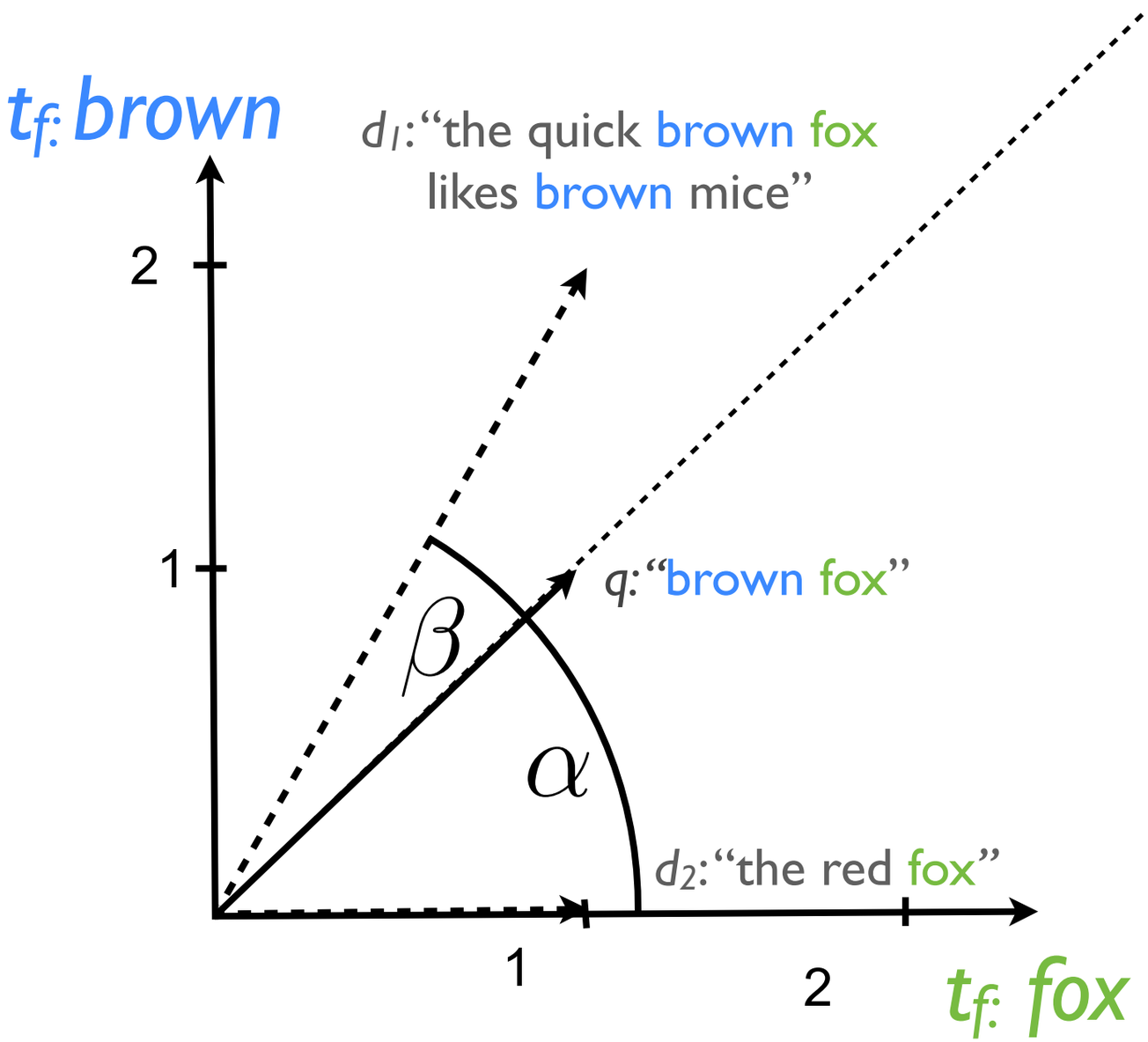


# Relevancy - Cosine Similarity

*Distance* of docs and query:

*Cosine of angle* between document vector on query *axis*.

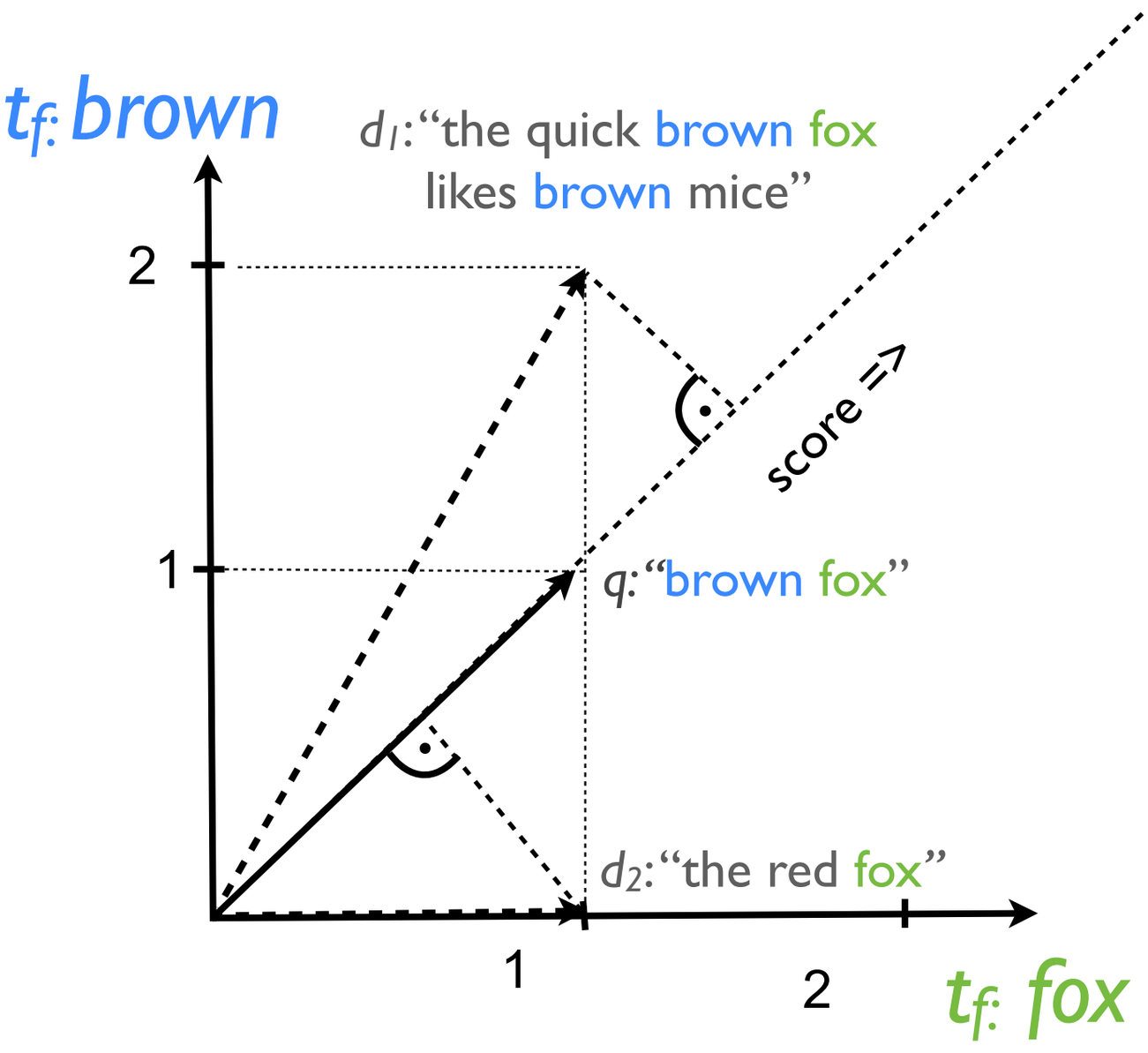
$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$



# Relevancy - Projection distance

*Distance* of docs and query:

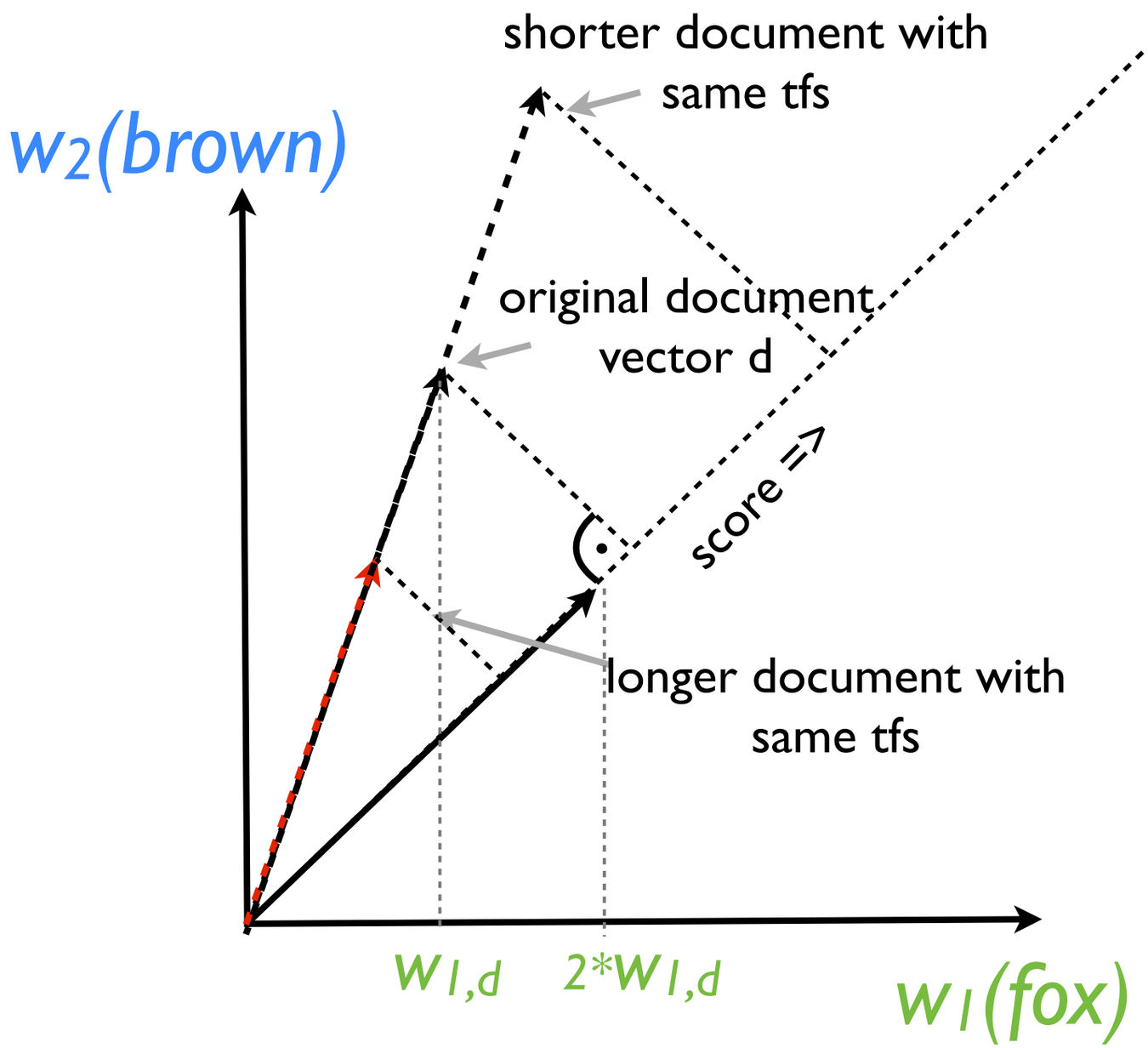
*Project* document vector on query *axis*.



# Relevancy - Field length

Shorter text is more relevant than longer text.

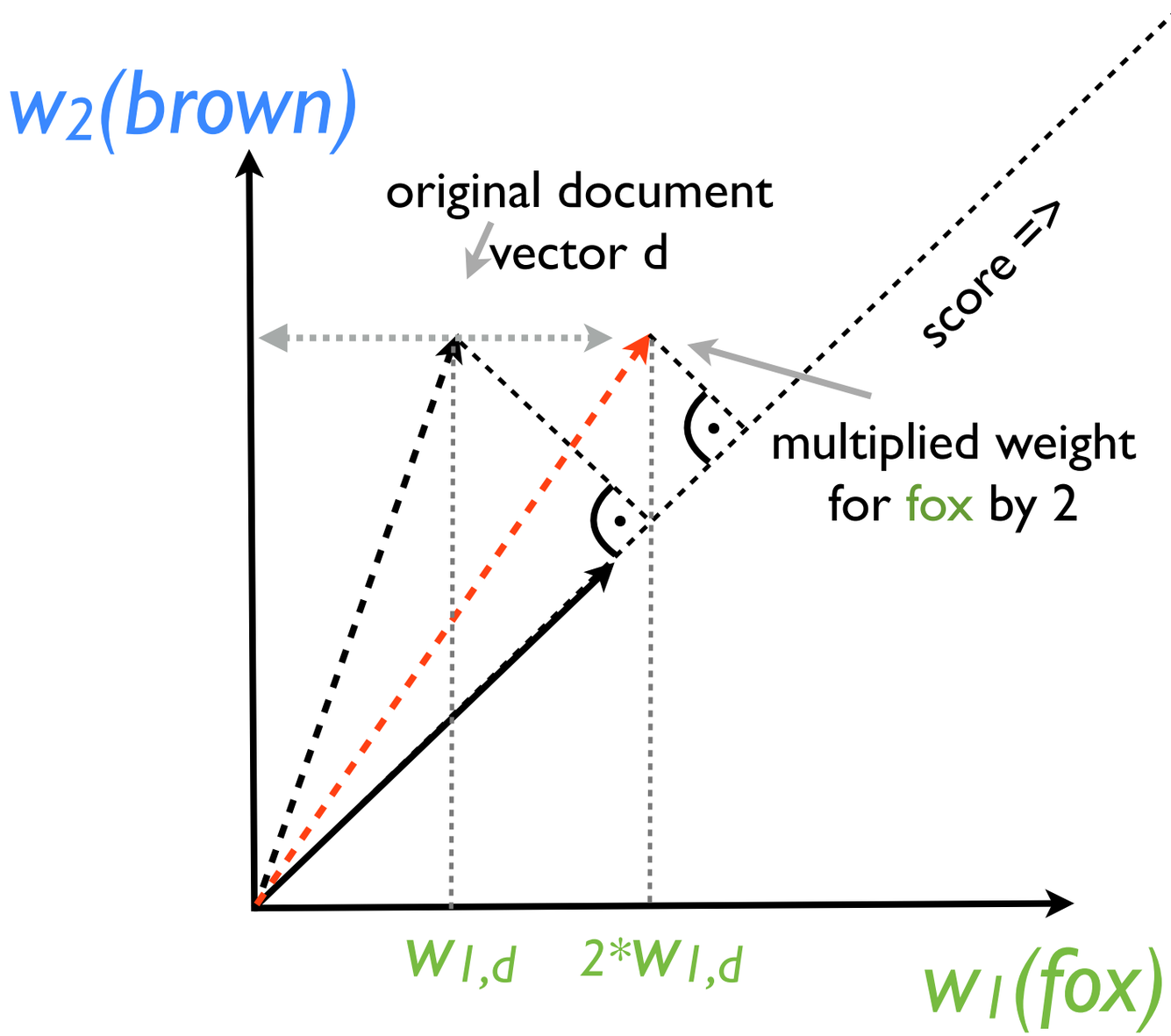
# Relevancy - Field length



# Relevancy - document frequency

Words that appear more often in documents are less important that words that appear less often.

# Relevancy - term weight



How many of these factors are there?

# Lucene Similarity - TF-IDF

inverted document frequency for term *t*

$$\mathrm{idf}_f = 1 + \frac{|D|}{1 + |\{d' \in D | t \in d'\}|}$$

field length, some function turning the number of tokens into a float, roughly:

$$1/\sqrt{\text{num terms in field}}$$

score of a document *d* for a given query *q*

$$\mathrm{score}_{q,d} = \mathrm{norm}(q) \times \sum_{t \text{ in } q} \sqrt{\mathrm{tf}_{t,d}} \times \mathrm{idf}_t^2 \times \mathrm{norm}(d, field) \times \mathrm{boost}(t)$$

query norm

core TF/IDF weight

boost of query term *t*

http://lucene.apache.org/core/4_8_1/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

# That was TF-IFD

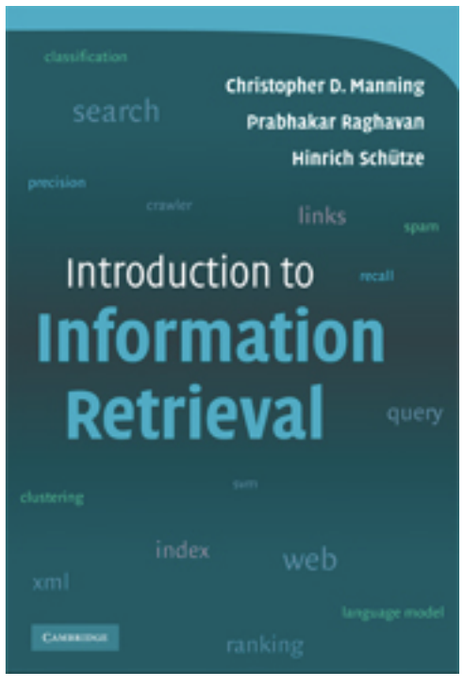
But..there are other fancy equations with lots of greek letters, right?

Yes! Elasticsearch is built on top of Lucene and there we have:

- Language model scoring
- BM25
- DFRSimilarity
- …

*https://lucene.apache.org/core/4_8_1/core/org/apache/lucene/search/similarities/Similarity.html*

And how do I learn about these?




+

# “similarity module” doc

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index-modules-similarity.html#configuration*

+

# “Elasticsearch - The definite Guide”

http://www.elasticsearch.org/guide/en/elasticsearch/guide/current/index.html

# Coming soon!

# II: DIY scoring

# Why would you want to tweak the score?

1. If you need numerical values: popularity of an item
2. You want a distance of a numerical value to influence the score
3. You want to score tags
4. You want to write your own text scoring function
5. ….
6. You want to combine these

# http://colors.qbox.io/

*http://blog.qbox.io/boston-elasticsearch-meetup-scoring-images-by-color*

# function_score - basic structure

query or filter

```
"function_score": {
      "(query|filter)": {},
      "functions": [
          {
              "filter": {},
              "FUNCTION": {}
          },
          ...
      ]
}
```

Apply score computation only to docs matching a specific filter (default “match_all”)

Apply this function to matching docs

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-function-score-query.html*

# Options

- field value factor
- distance function
- random scoring
- boost factor
- script scoring

```
"field_value_factor": {
  "field": "popularity",
  "factor": 1.2,
  "modifier": "sqrt"
}
```

# Options

- **field value factor**
- distance function
- random scoring
- boost factor
- script scoring

```
"field_value_factor": {
  "field": "popularity",
  "factor": 1.2,
  "modifier": "sqrt"
}
```

# Options

- field value factor
- **distance function**
- random scoring
- boost factor
- script scoring

```
"DECAY_FUNCTION": {
    "price": {
        "origin": "0",
        "scale": "20"
    }
}
```

# Options

- field value factor
- distance function
- **random scoring**
- boost factor
- script scoring

```
"random_score": {
    "seed" : number
}
```

# Options

- field value factor
- distance function
- random scoring
- **boost factor**
- script scoring

```
"boost_factor": "3"
```

# Scoring odysseys

http://www.elasticsearch.org/videos/introducing-custom-scoring-functions/

https://gist.github.com/brwe/7049473

# Why would you want to tweak the score?

1. If you need numerical values: popularity of an item
2. You want a distance of a numerical value to influence the score
3. You want to score tags
4. You want to write your own text scoring function
5. ....
6. You want to combine these

# function_score - script scoring

```
"function_score": {
      "(query|filter)": {},
      "functions": [
          {
              “script_score": {
                  “params”: {…},
                  “lang”: “…”,
                  “script”: “…”
              }
          },
          ...
      ]
}
```

any constant input that you can pre-compute

python, groovy, mvel, native,…

?

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-function-score-query.html*

# document values

_doc variable allows access to document values:

```
“_doc[‘popularity’].value”
```

```
“Math.pow(_doc[‘popularity’].value,2)”
```

_index variable allows access to term statistics

# What is in an index?

# Term frequency

```
_index['text']['term'].tf()
```

Number of times term is in a document

The quick brown fox likes brown mice:

tf of brown :  2
tf of fox :  2

```
"query": {
      "function_score": {
         "script_score": {
            "script": "_index['text']['berlin'].tf()"
         }
    }
 }
```

document that contains “berlin” most often will score highest

```
"query": {
   "function_score": {
      "filter": {
         "terms": {
            "text": ["john","smith"],
            "execution": "and"
         }
      },
      "script_score": {
         "params": {
            "field": "text",
            "terms": [“john”, "smith"]
         },
         "script": "float score = 0;
                   for (term : terms) {
                         score += _index[field][term].tf();
                    }
                   return score;",
         "lang": "mvel"
      }
   }
}
```

This will speed up things

Search terms and field

Sum term frequency over all terms

# Document frequency

```
_index['text']['token'].df()
```

number of times token appears in a doc, regardless of how often

```
doc1: {"text":  "I am Sam, Sam I am."}

doc2: {"text":  "I know that I don't know."}

_index['text']['i'].df() = 2
```

# And so on…

```
_index['text']['token'].ttf()
```

total term frequency:

sum of term frequency over all documents

```
_index['text']['token'].sumttf()
```

sum total term frequency:

number of tokens in all docs in index

# detour token count

- Lucene does not store number of tokens in a field
- must be enabled in mapping and accessed as regular field:
- access as field value

```
"doc['text.word_count'].value"
```

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/mapping-core-types.html*

# Positions

```
iterator pos_iter =
_index['text'].get('token', _POSITIONS)
```

```
"text":  "I am Sam, Sam I am."

          0 1  2    3    4 5

positions:

"i":  [0, 4]

"am": [1, 5]

…
```

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": “
float score = 0;
indexField = _index[field];
word_count = _doc["text.word_count"].value;
for (term : terms) {
     indexFieldTerm = indexField[term];
     int df = (int) indexFieldTerm.df();
     int tf = indexFieldTerm.tf();
     if (df != 0 && tf != 0) {
         score += Math.sqrt(tf) *
                  Math.pow(
                       1+Math.log((float) indexField.docCount() /
                                  ((float) df + 1.0)),
                       2) /
                  Math.log(word_count);
     }
}
return score;
```

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": “
float score = 0;
indexField = _index[field];
word_count = _doc["text.word_count"].value;
for (term : terms) {
     indexFieldTerm = indexField[term];
     int df = (int) indexFieldTerm.df();
     int tf = indexFieldTerm.tf();
     if (df != 0 && tf != 0) {
         score += Math.sqrt(tf) *
                  Math.pow(
                       1+Math.log((float) indexField.docCount() /
                                  ((float) df + 1.0)),
                       2) /
                  Math.log(word_count);
     }
}
return score;
```

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": “
float score = 0;
indexField = _index[field];
word_count = _doc["text.word_count"].value;
for (term : terms) {
     indexFieldTerm = indexField[term];
     int df = (int) indexFieldTerm.df();
     int tf = indexFieldTerm.tf();
     if (df != 0 && tf != 0) {
         score += Math.sqrt(tf) *
                  Math.pow(
                       1+Math.log((float) indexField.docCount() /
                                  ((float) df + 1.0)),
                       2) /
                  Math.log(word_count);
     }
}
return score;
```

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": “
float score = 0;
indexField = _index[field];
word_count = _doc["text.word_count"].value;
for (term : terms) {
     indexFieldTerm = indexField[term];
     int df = (int) indexFieldTerm.df();
     int tf = indexFieldTerm.tf();
     if (df != 0 && tf != 0) {
         score += Math.sqrt(tf) *
                  Math.pow(
                       1+Math.log((float) indexField.docCount() /
                                  ((float) df + 1.0)),
                       2) /
                  Math.log(word_count);
     }
}
return score;
```

$$\text{idf}_f = 1 + \frac{|D|}{1 + |\{d' \in D | t \in d'\}|}$$

# TF-IDF in 17 lines

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{\text{t in q}} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": "
float score = 0;
indexField = _index[field];
word_count = _doc["text.word_count"].value;
for (term : terms) {
     indexFieldTerm = indexField[term];
     int df = (int) indexFieldTerm.df();
     int tf = indexFieldTerm.tf();
     if (df != 0 && tf != 0) {
         score += Math.sqrt(tf) *
                  Math.pow(
                       1+Math.log((float) indexField.docCount() /
                                  ((float) df + 1.0)),
                       2) /
                  Math.log(word_count);
     }
}
return score;
```

Not exactly the same

# Phrase scorer in 13 lines

```
"params": {"field": "text",
           "words": ["john","smith"]
          },
"script": “

firstNamePositions = _index[field].get(words[0], _POSITIONS);
lastNamePositions = _index[field].get(words[1],_POSITIONS).iterator();
lastNamePosition = -1;
float wordDistance = 1000000;
for (firstNamePosition : firstNamePositions) {
  while (lastNamePositions.hasNext() &&
         (lastNamePosition <= firstNamePosition.position)) {
      lastNamePosition = lastNamePositions.next().position;
  }
  wordDistance= Math.min(wordDistance,
                         lastNamePosition - firstNamePosition.position);
}
return (float)1.0/wordDistance;"
```

# Very rough estimate of runntime

Run 5 times and measure time for phrase scorer and TFIDF

Compare

- Lucene phrase query/Terms query
- MVEL
- native script

Very basic script implementation

| | Lucene | MVEL | native |
|---|---|---|---|
| tf-idf | 317.8 | 52191.25 | 1710.6 |
| phrase | 1185.6 | 39163.2 | 1230.15 |

Check if it is already there

Check if it is already there

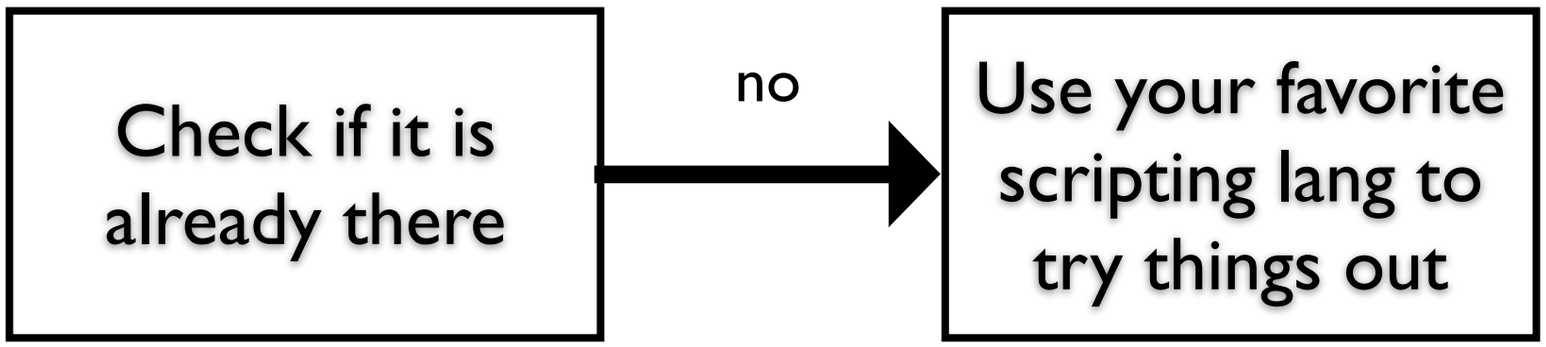
function_score has already built it

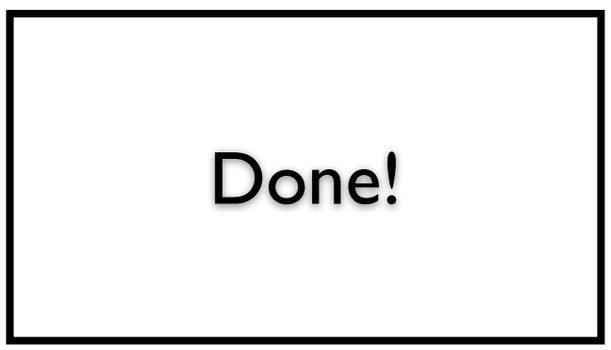
- field value factor
- distance function
- random scoring
- boost factor

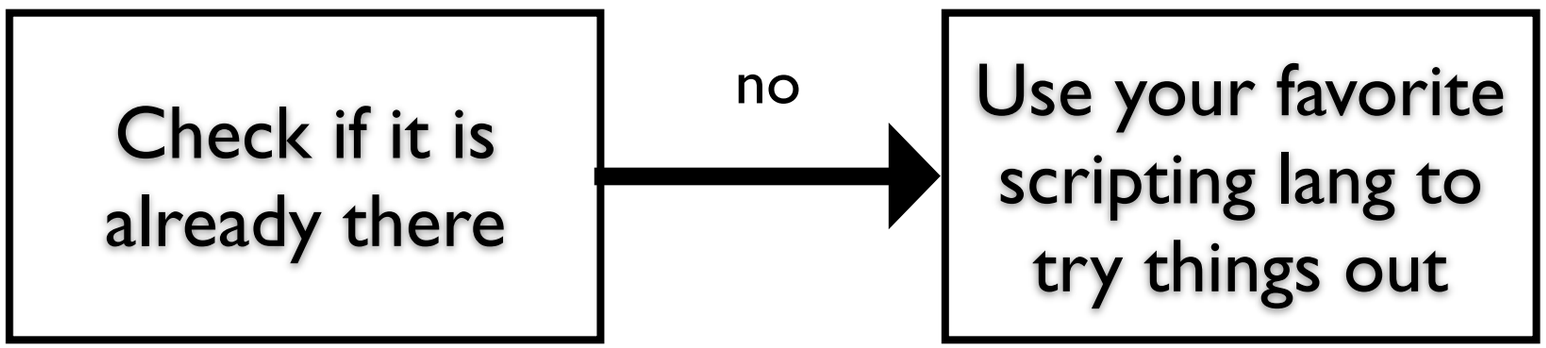
Check if it is already there

yes

Done!

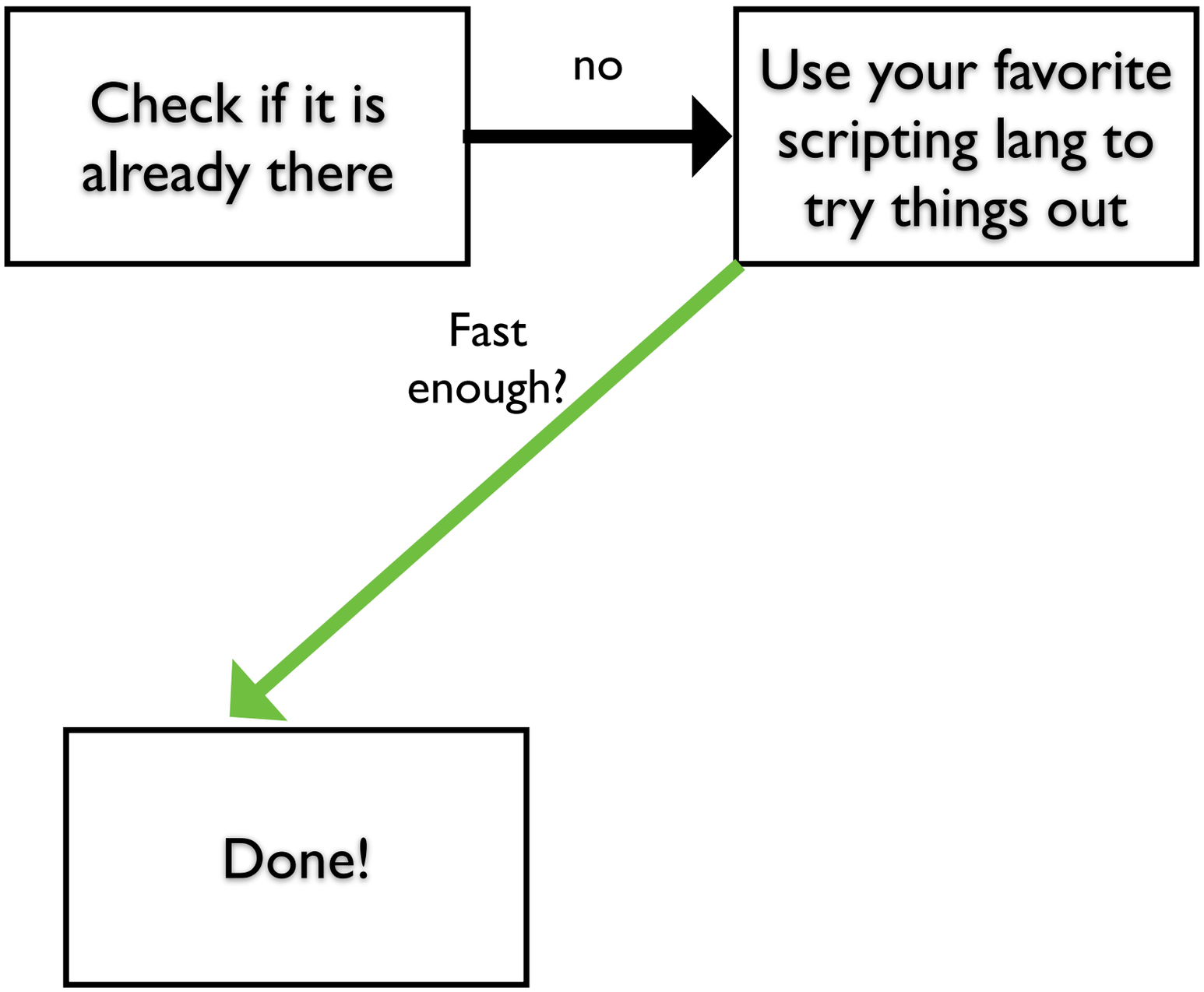

Check if it is already there
no
Use your favorite scripting lang to try things out


Done!

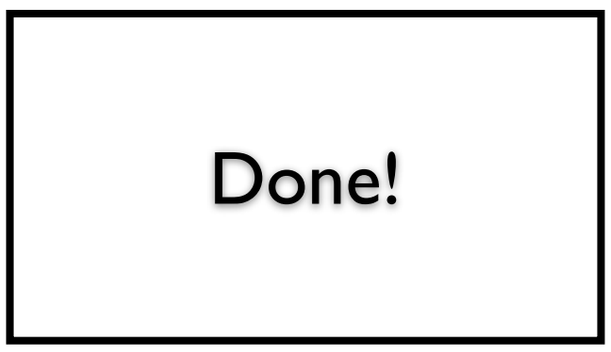



Done!

- python
- groovy
- mvel
- javascript


Check if it is already there
no
Use your favorite scripting lang to try things out
Fast enough?
Done!


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script

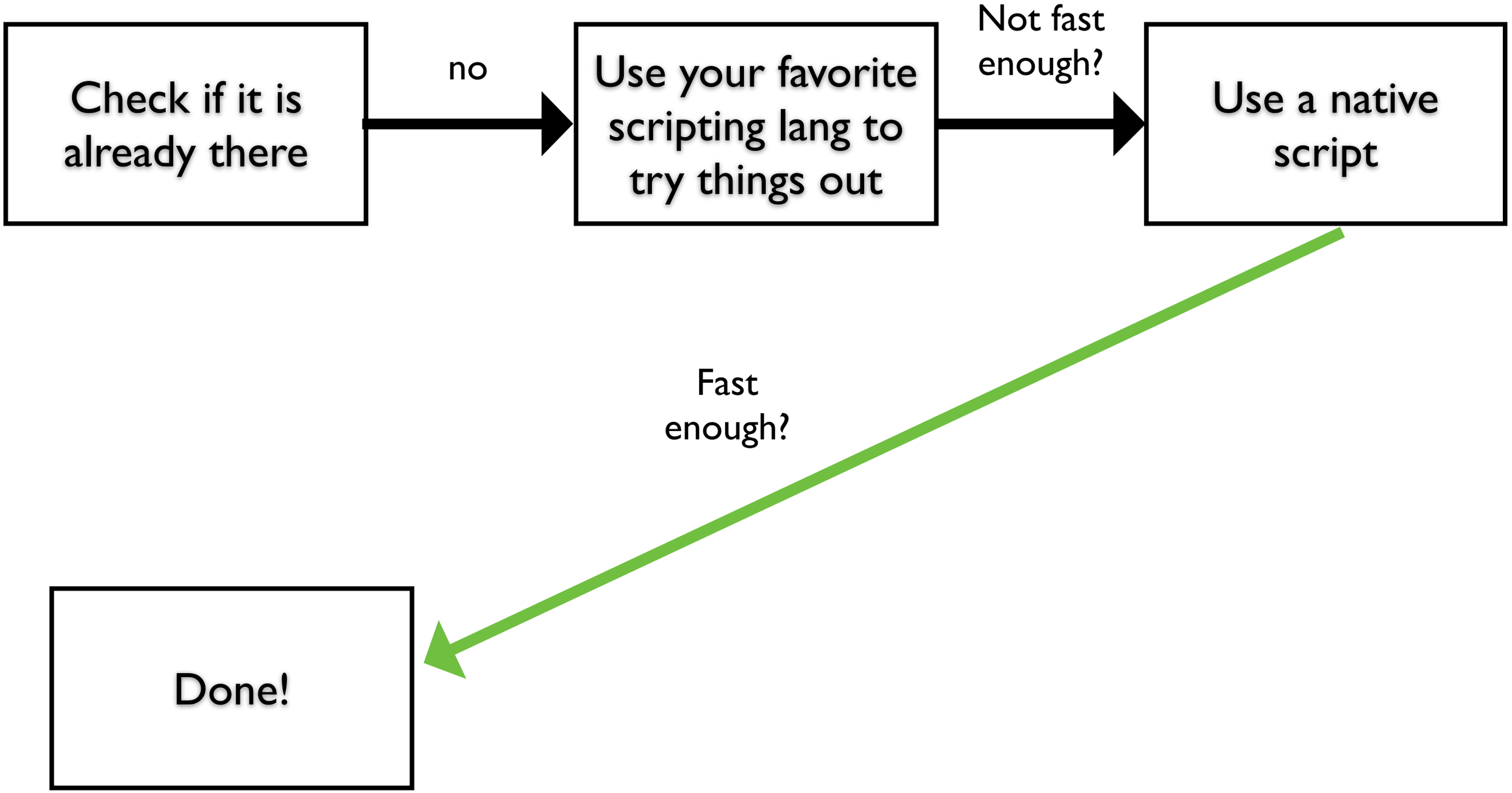

Done!

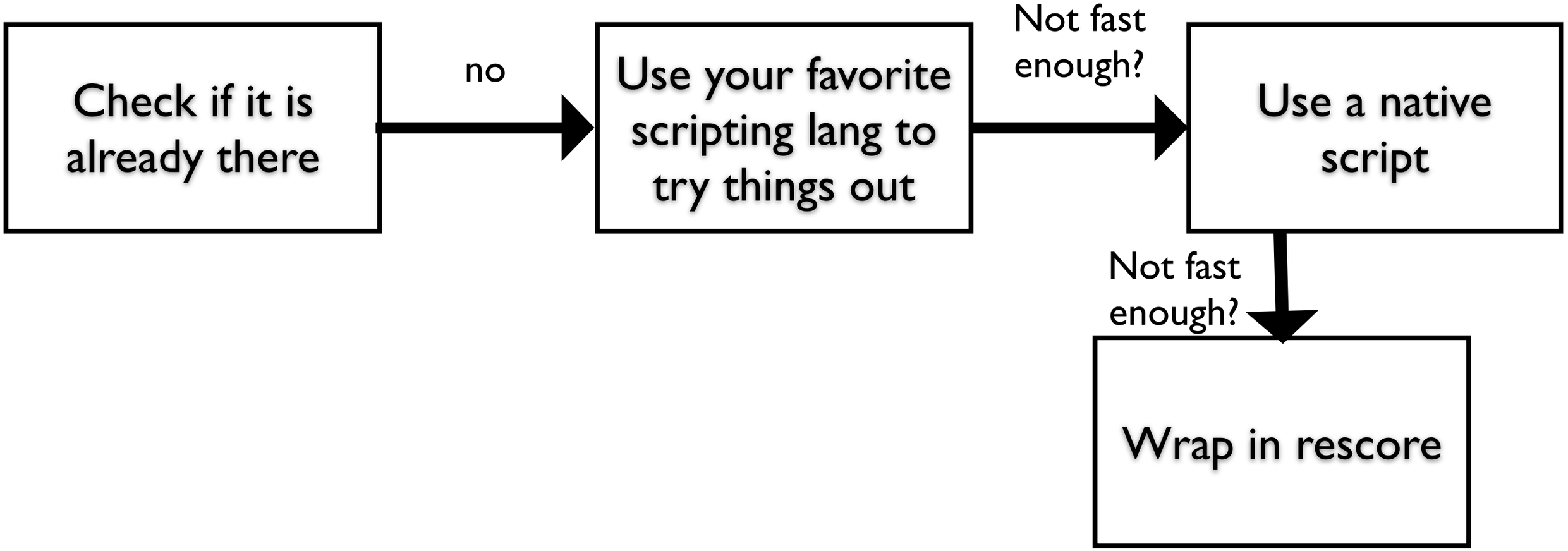

Done!

Plugin
Pro:
faster than scripting, because in java
Con:

- Needs to be maintained
- Need to restart node when changed
- More code

https://github.com/imotov/elasticsearch-native-script-example


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Fast enough?
Done!




Done!

- Apply scoring function only to top N documents.
- good for reordering
- need to know the best results are within the top N

http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/search-request-rescore.html

Check if it is already there

no

Use your favorite scripting lang to try things out

Not fast enough?

Use a native script

Not fast enough?

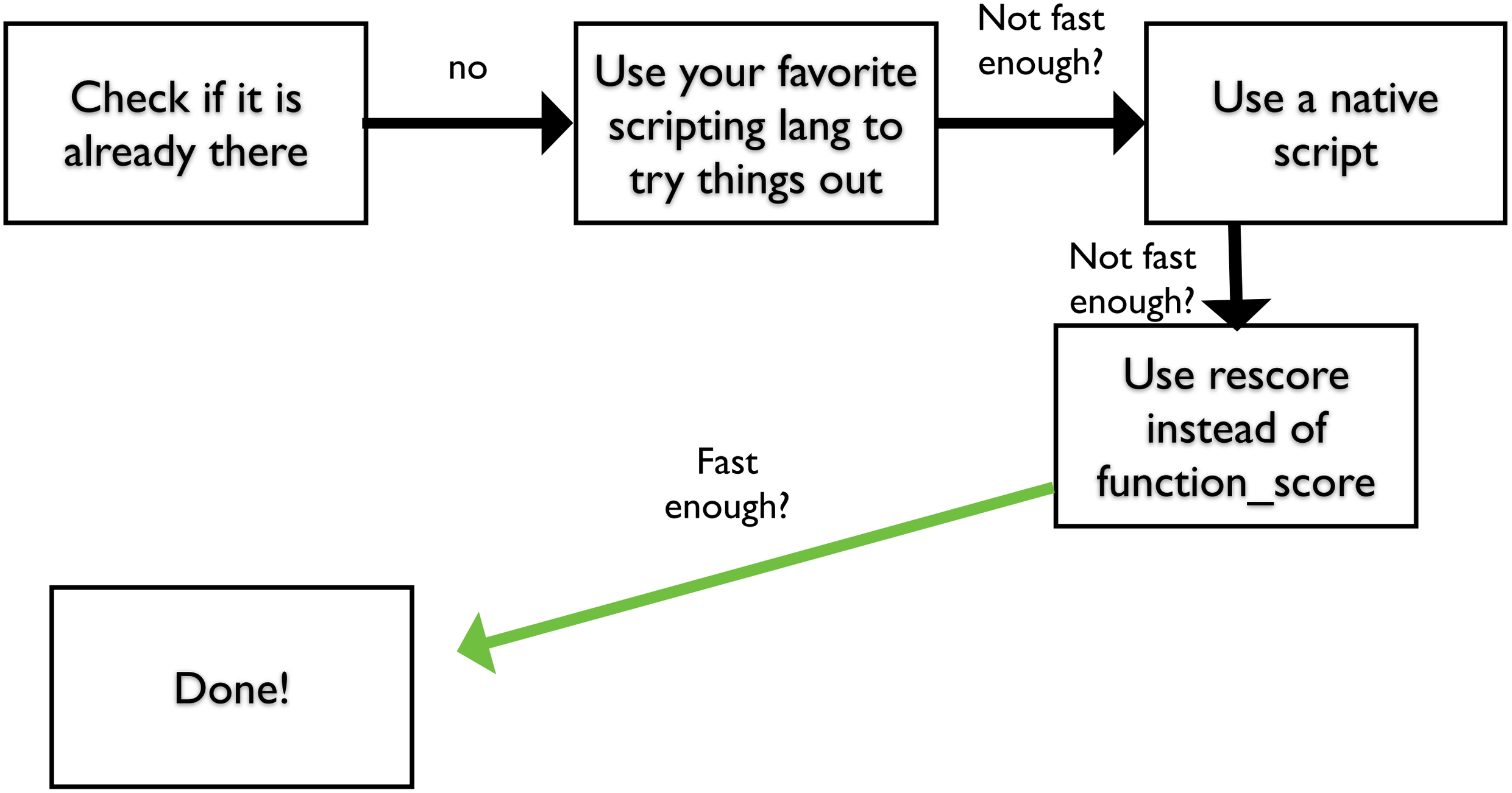
Use rescore instead of function_score

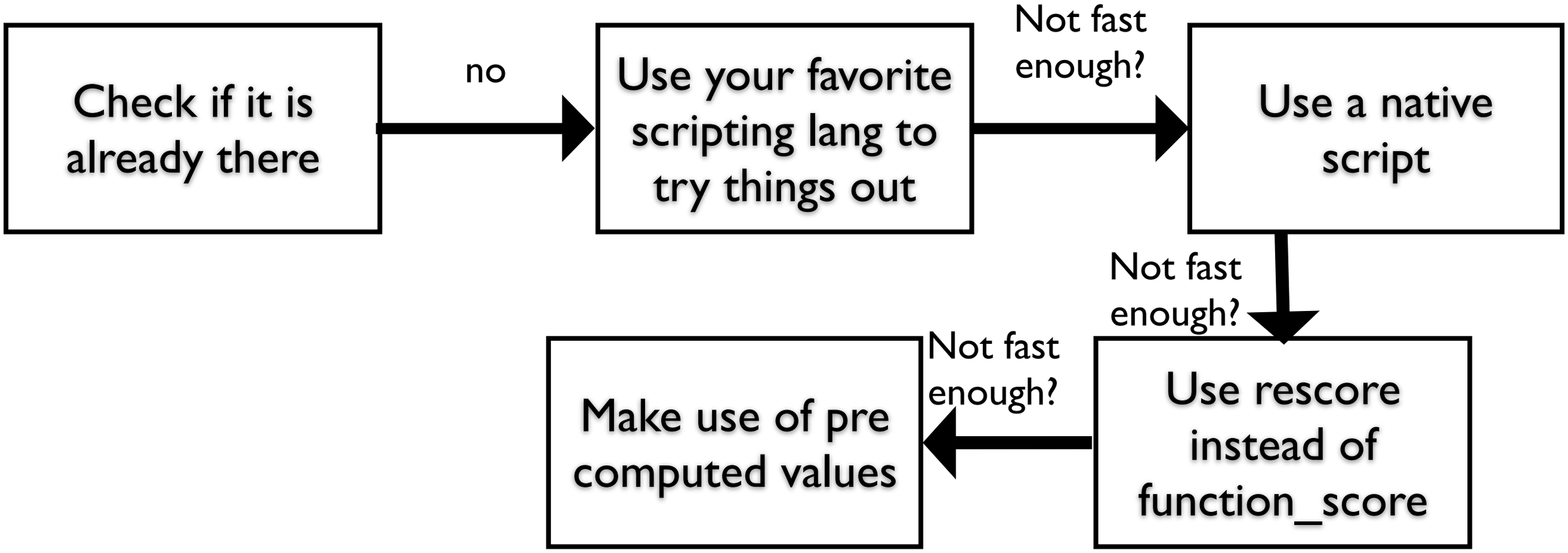
Done!

```
"query" : {
   …
},
"rescore" : {
   "window_size" : 50,
   "query" : {
      "rescore_query" : {
         “function_score”: {

…
```

http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/search-request-rescore.html

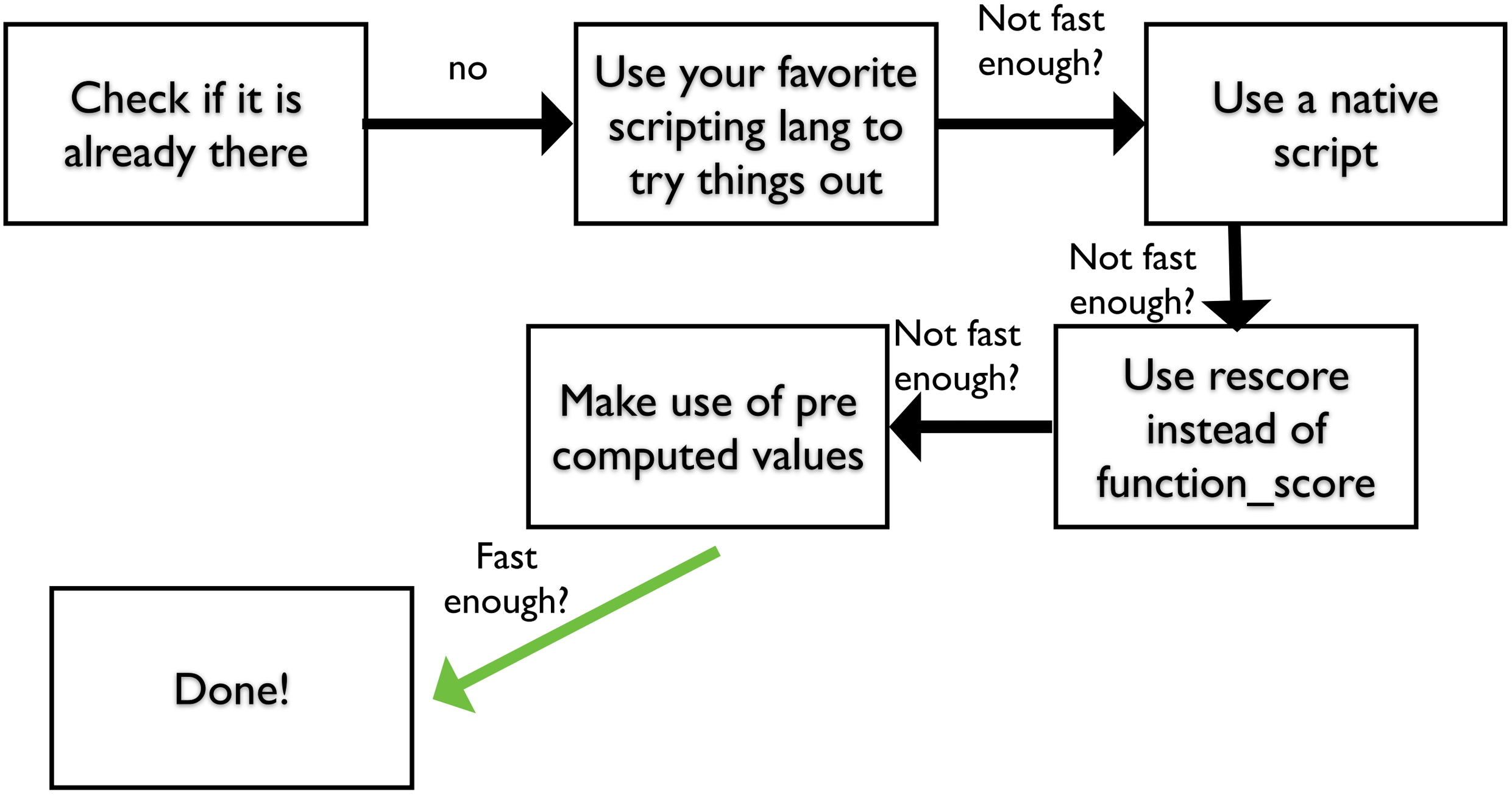

Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Not fast enough?
Use rescore instead of function_score
Fast enough?
Done!

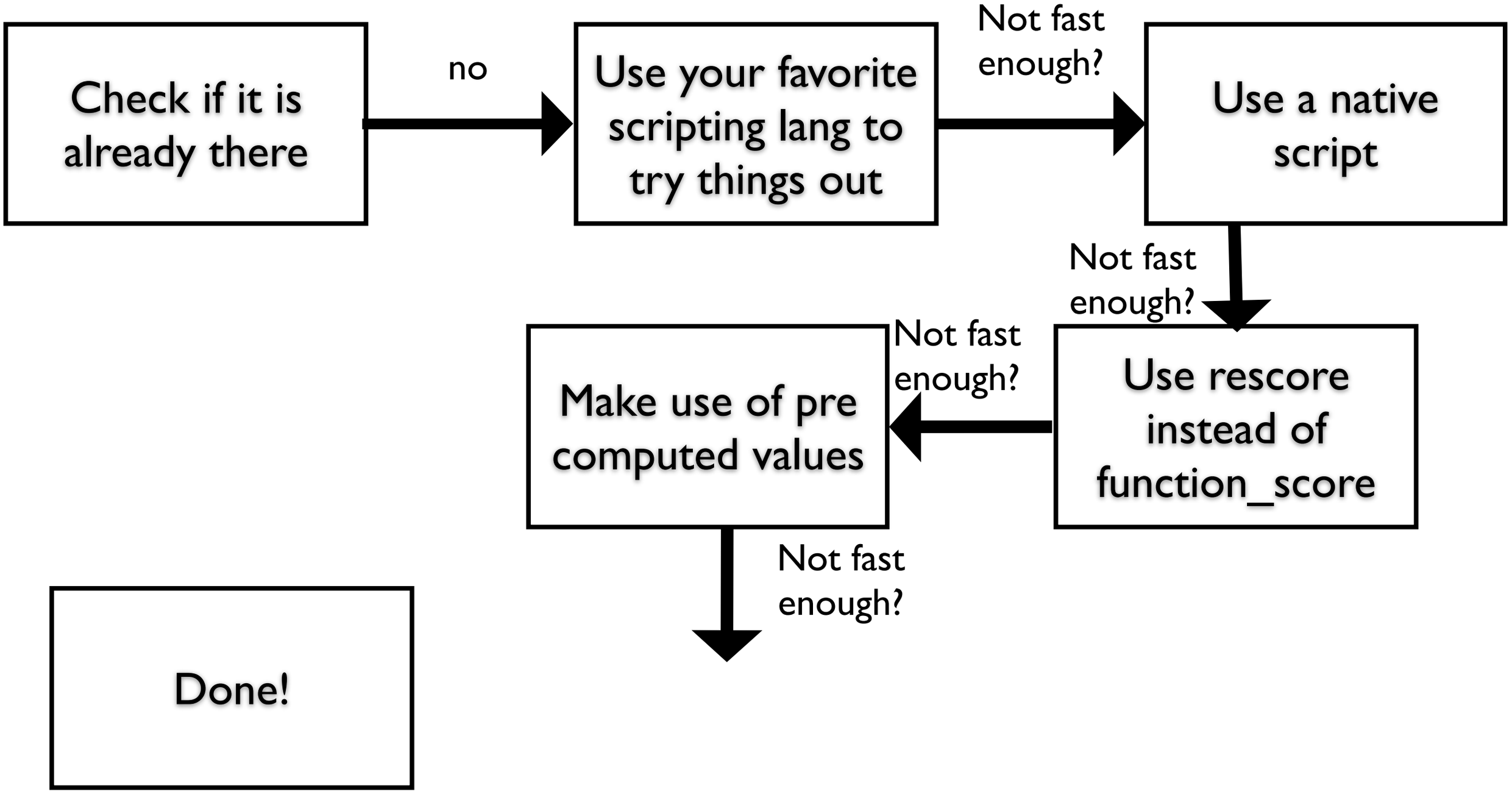


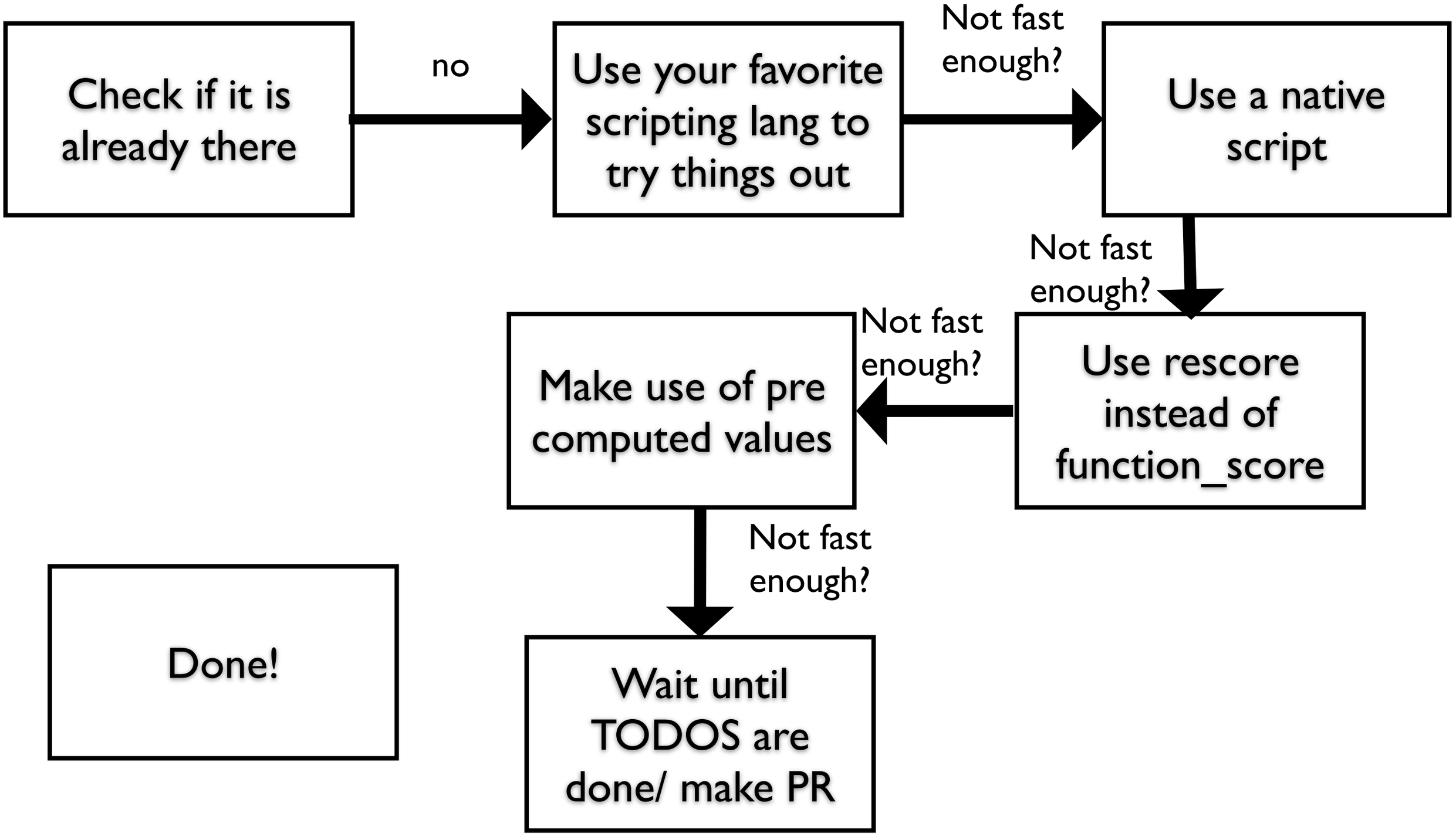

Done!

Lucene is quicker because of pre-computed values

- Pass as parameters
- Store with the document


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Not fast enough?
Use rescore instead of function_score
Not fast enough?
Make use of pre computed values
Fast enough?
Done!


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Not fast enough?
Use rescore instead of function_score
Not fast enough?
Make use of pre computed values
Not fast enough?
Done!


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Not fast enough?
Use rescore instead of function_score
Not fast enough?
Make use of pre computed values
Not fast enough?
Wait until TODOS are done/ make PR
Done!

# TODOS

- Pre compute values when indexing: Script to compute values one the fly after analysis
- Pre compute values before search execution on shard (idf)
- Currently only uses shard statistics, collect for whole index instead before execution
- Analyze query terms, like match query

Hm. This is all interesting, but I really do not need to tweak the score….

# Where else can you use this?

- Script fields: Get a document and compute your favurite value, class,… based on term statistics
- Aggregations: Use a script to aggregate term statistics based on a class, query,…

# Other nice related Buzzwords talks

1. Different queries in elasticsearch

   See “Elasticsearch Query DSL - Not just for wizards…”, Clinton Gormley

2. Evaluation

   See “Search quality in practice”, Alexander Sibiryakov

3. Learning how to rank

   See ”Lean Ranking infrastructure with Solr”, Sergej Khmoneko

4. Implementation details

   See “The ultimate guide for Elasticsearch plugins”, Itamar Syn-Hershko

# But wait…there's more!

# Naive Bayes

Use script to gather term statistics and to learn the model

Use script field to apply the model to new documents!

Plus: Use significant terms aggregation to apply the features.

```
posProbs = hash map, probabilities for each term, P(t|C=positive)
negProbs = hash map, probabilities for each term, P(t|C=negative)
terms = list of all the terms
pPos=0; pNeg =0;
for( term : terms) {
    pPos+=_index[\"text\"][term].tf()*log(posProbs[term]);
    pNeg+=_index[\"text\"][term].tf()*log(negProbs[term]);
}
pPos+=log(posClassProb);
pNeg+=log(negClassProb);
classname = \"\";
if (pPos>pNeg){
    classname = \"pos\"
} else {
    classname = \"neg\"
}
return classname;
```

# Practical advise

- Create evaluation data
- Write native script if you settled on one function (see https://github.com/imotov/elasticsearch-native-script-example)
- Filter out as much as you can before applying scoring function

If you need only simple stuff…

- Distance functions built-in
- boost built in
  - `random function`
  - `field boost`

…but sometimes you need more.

# rescore - basic structure EDIT

```
{
    "query" : {
        …
    },
    "rescore" : {
        "window_size" : 50,
        "query" : {
            "rescore_query" : {
                …
            }     }
    }
}
```

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/search-request-rescore.html*

# KNN

Use script scoring to define the metric

Count class labels in top N results

# Scoring odysseys

http://www.elasticsearch.org/videos/introducing-custom-scoring-functions/

https://gist.github.com/brwe/7049473

# TODOs

- Index wide statistics, similar to DFS_QUERY_THEN_FETCH
- Analysis of parameter string - script execution prior to search
- More optimizing…

Hm…I can just use a match query and filters, right?

```
“query”:
   “match”:
      “proglang”: “java”
…
```

# Agenda

PART 1: Text scoring for human beings

- How does in work in theory?
- How does it work in practice?
- How do I use it with elasticsearch?

PART 2: How do I tweak the score?

- writing your own scoring function in a script

- function_score in general

# How to tweak the score

Change the mapping

use function_score or resorer
write your won similarity class

So...more matching words mean higher score, right?

# Why am I giving this talk?

# When do you need to tweak?

- When you want to use field values of documents
- You are a researcher and wanna try new method
- You are a student and learn new things
- When you are en expert and know that your data cannot be properly scored by tf-idf

`script scoring` - LM Smilarity in 5 lines

`script scoring` - tfidf in 5 lines

# Define a similarity in a mapping

```
{
    "book":{
        "properties":{
            "title":{
                "type":"string", "similarity":"BM25"
            }
```

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/mapping-core-types.html*

# Customize your similarity

```
"similarity" : {
  "my_similarity" : {
    "type" : "DFR",
    "basic_model" : "g",
    "after_effect" : "l",
    "normalization" : "h2",
```

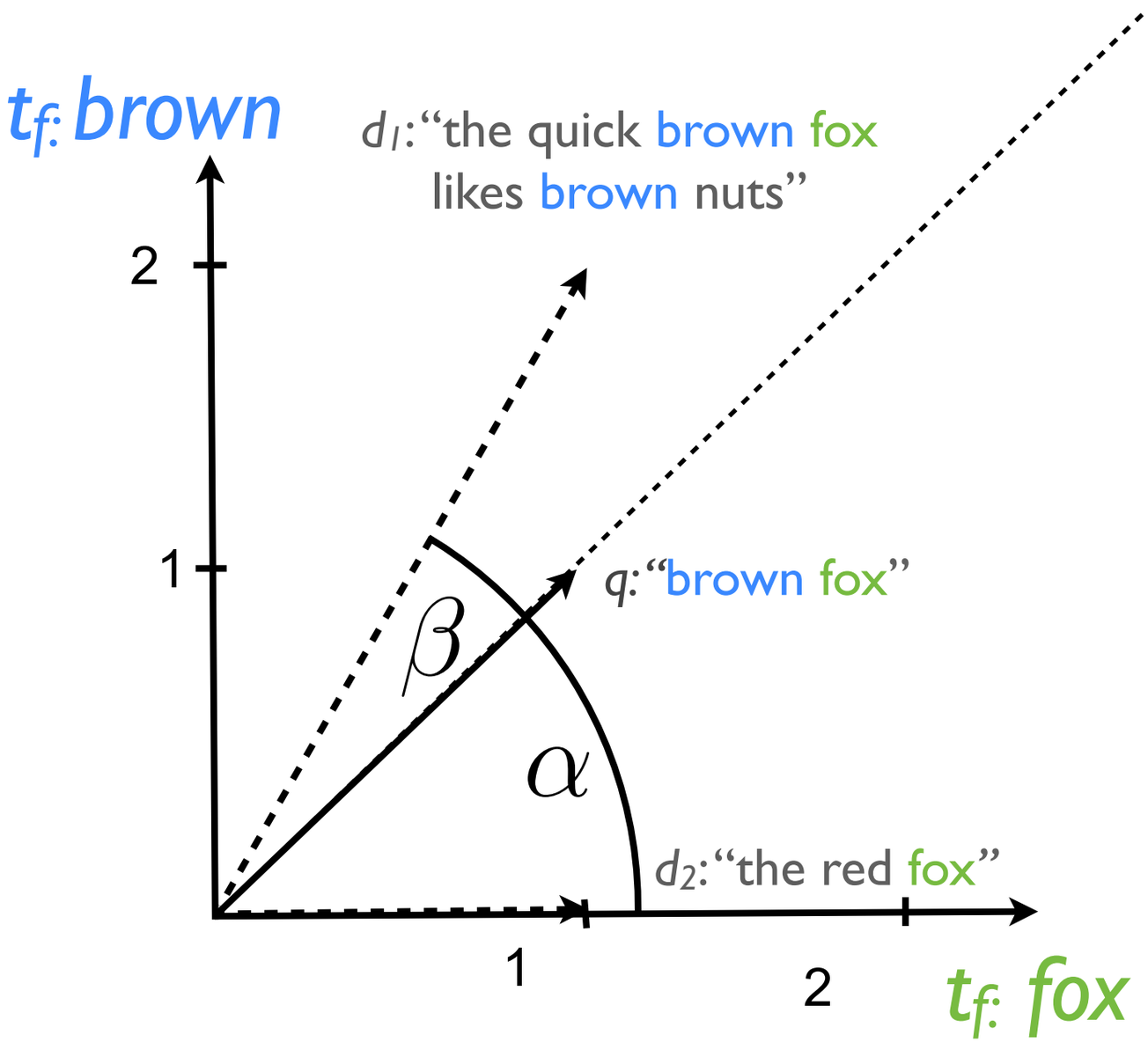
*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index-modules-similarity.html*

# Relevancy - Cosine Similarity

*Distance* of docs and query:

*Cosine of angle* between document vector on query *axis*.

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$



# Cosine similarity as script

```
"params": {
    "field": "fieldname",
    "words": [“word1", …]
},
"script": “

score = 0.0;
queryLength = 0.0;
docLength = 0.0;
for (word : words){
  tf = _index[field][word].tf();
  score = score + tf * 1.0;
  queryLength = queryLength + 1.0;
  docLength = docLength + pow(tf, 2.0);
}
return (float)score /
       (sqrt(docLength) * sqrt(queryLength));

"
```

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

# Cosine similarity as script

```
"params": {
    "field": "fieldname",
    "words": [“word1", …]
},
"script": “

score = 0.0;
queryLength = 0.0;
docLength = 0.0;
for (word : words){
  tf = _index[field][word].tf();
  score = score + tf * 1.0;
  queryLength = queryLength + 1.0;
  docLength = docLength + pow(tf, 2.0);
}
return (float)score /
       (sqrt(docLength) * sqrt(queryLength));

"
```

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

# Cosine similarity as script

```
"params": {
    "field": "fieldname",
    "words": [“word1", …]
},
"script": “

score = 0.0;
queryLength = 0.0;
docLength = 0.0;
for (word : words){
  tf = _index[fieldname][word].tf();
  score = score + tf * 1.0;
  queryLength = queryLength + 1.0;
  docLength = docLength + pow(tf, 2.0);
}
return (float)score /
       (sqrt(docLength) * sqrt(queryLength));

"
```

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

# Cosine similarity as script

```
"params": {
    "field": "fieldname",
    "words": [“word1", …]
},
"script": “

score = 0.0;
queryLength = 0.0;
docLength = 0.0;
for (word : words){
  tf = _index[fieldname][word].tf();
  score = score + tf * 1.0;
  queryLength = queryLength + 1.0;
  docLength = docLength + pow(tf, 2.0);
}
return (float)score /
       (sqrt(docLength) * sqrt(queryLength));

"
```

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

# Cosine similarity as script

```
"params": {
    "field": "fieldname",
    "words": [“word1", …]
},
"script": “

score = 0.0;
queryLength = 0.0;
docLength = 0.0;
for (word : words){
  tf = _index[fieldname][word].tf();
  score = score + tf * 1.0;
  queryLength = queryLength + 1.0;
  docLength = docLength + pow(tf, 2.0);
}
return (float)score /
       (sqrt(docLength) * sqrt(queryLength));

"
```

$$cos(\omega) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

# What is in an index?

- Bag-of-words - do not need the ordering.
- But what if we need the ordering?
- Positions
- Payloads
- Offset -> co care, only used for highlighting

# Token

doc: {“text”: “I am Sam, Sam I am.”}

Tokens: ‘i’, ‘am’, ‘sam’

# Explain api

If you do not understand the score:

```
curl -XPOST "http://localhost:9200/idfidx/test/_search" -d'
{
    "query": {
        "match": {
            "location": "berlin kreuzberg"
        }
    },
    "explain": true
}'
```

# Exciting Quiz!

https://gist.github.com/brwe/7229896

The point is… EDIT

- Text scoring per default is tuned for natural language text.
- Empirical scoring formula works well for articles, mails, reviews, etc.
- This way to score might be undesirable if the text represents *tags*.

# Remember…Lucene Similarity

"Can we not make this idf^1.265?"

"I do not like the field length - how can I get rid of it?"

$$\text{idf}_f = 1 + \frac{|D|}{1 + |\{d' \in D | t \in d'\}|}$$

$$1/\sqrt{\text{num terms in field}}$$

$$\text{score}_{q,d} = \text{norm}(q) \times \sum_{t \text{ in } q} \sqrt{\text{tf}_{t,d}} \times \text{idf}_t^2 \times \text{norm}(d, field) \times \text{boost}(t)$$

"I do not need that!"

"Can I have the tf squared?"

"I want my boost to depend on the ratio of number of characters and average hight of my former gfs divided by the number of Friday 13ths in the last year!"

http://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

# How to tweak the score in elasticsearch

1. function_score

compute new score for all docs that match a certain filter

2. rescore

take top n documents and rescore them

3. Write your own Similarity class and plug it in

# Write your own Similarity

Pros:

- super fast! You cannot beat Lucene.

Cons:

- It's a plugin: need to restart node when changing the scoring function
- It's a plugin: must be maintained,
- lots of code before seeing a result

You will want to test how well your scoring actually works before digging through Lucene code.

Example: https://github.com/tlrx/elasticsearch-custom-similarity-provider

# function_score - basic structure

query or filter

```
"function_score": {
      "(query|filter)": {},
      "functions": [
          {
              "filter": {},
              "FUNCTION": {}
          },
          ...
      ]
}
```

Apply score computation only to docs matching a specific filter (default “match_all”)

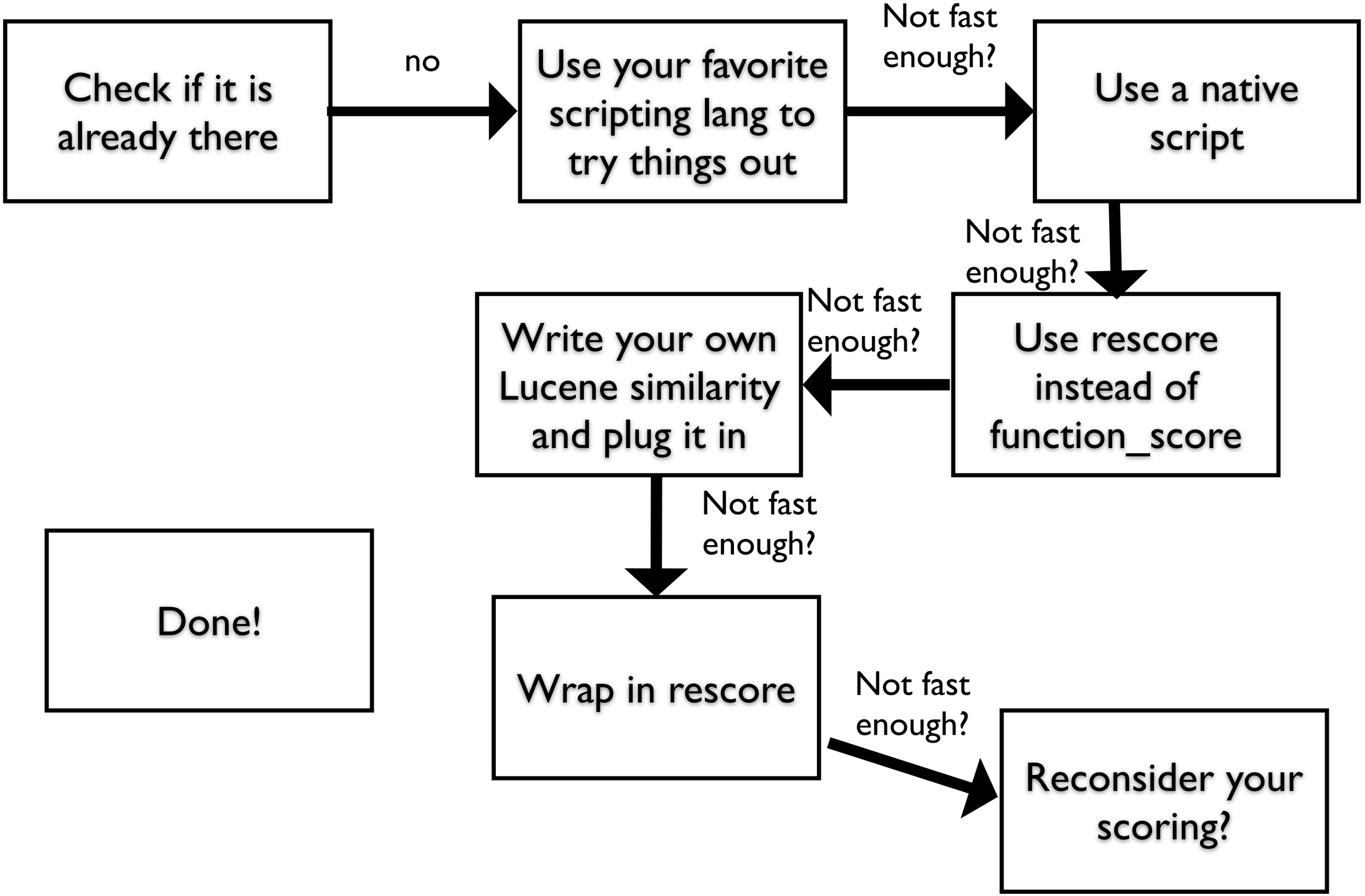
Apply this function to matching docs

*http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-function-score-query.html*

If you want to know more….

http://www.elasticsearch.org/videos/introducing-custom-scoring-functions/

https://gist.github.com/brwe/7049473


Check if it is already there
no
Use your favorite scripting lang to try things out
Not fast enough?
Use a native script
Not fast enough?
Use rescore instead of function_score
Not fast enough?
Write your own Lucene similarity and plug it in
Not fast enough?
Wrap in rescore
Not fast enough?
Reconsider your scoring?
Done!