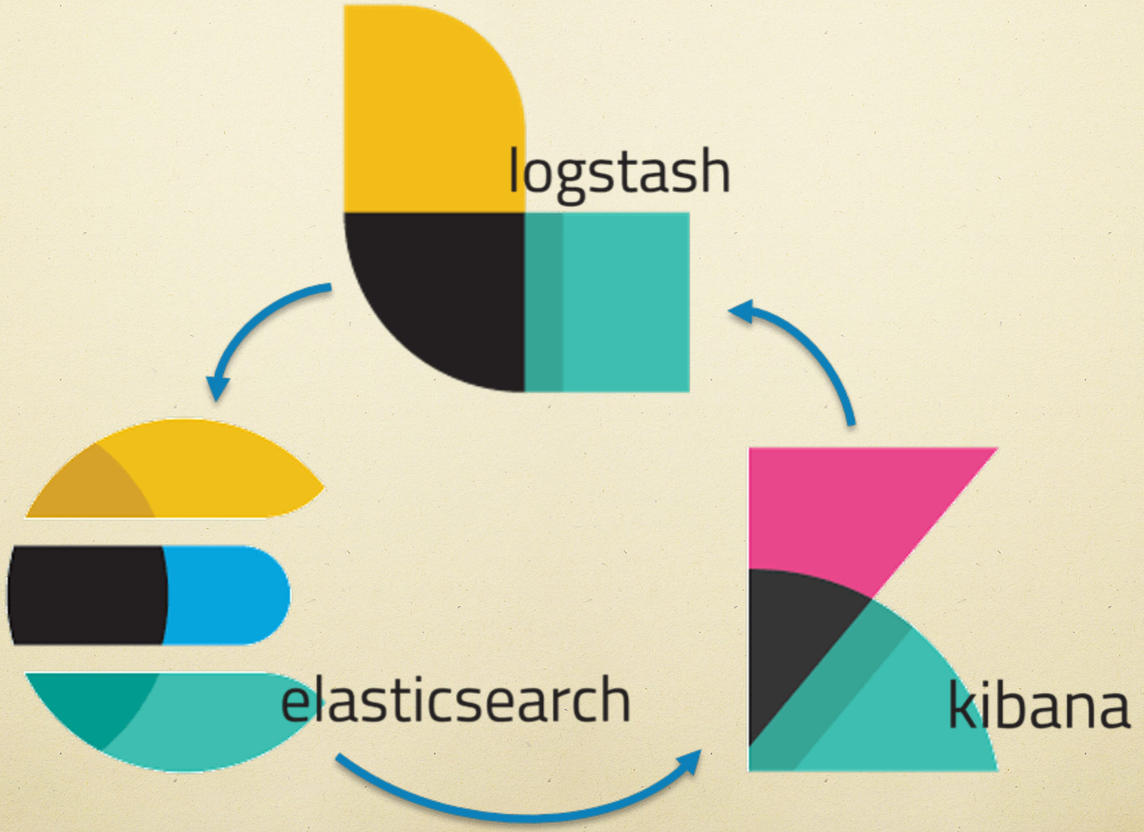
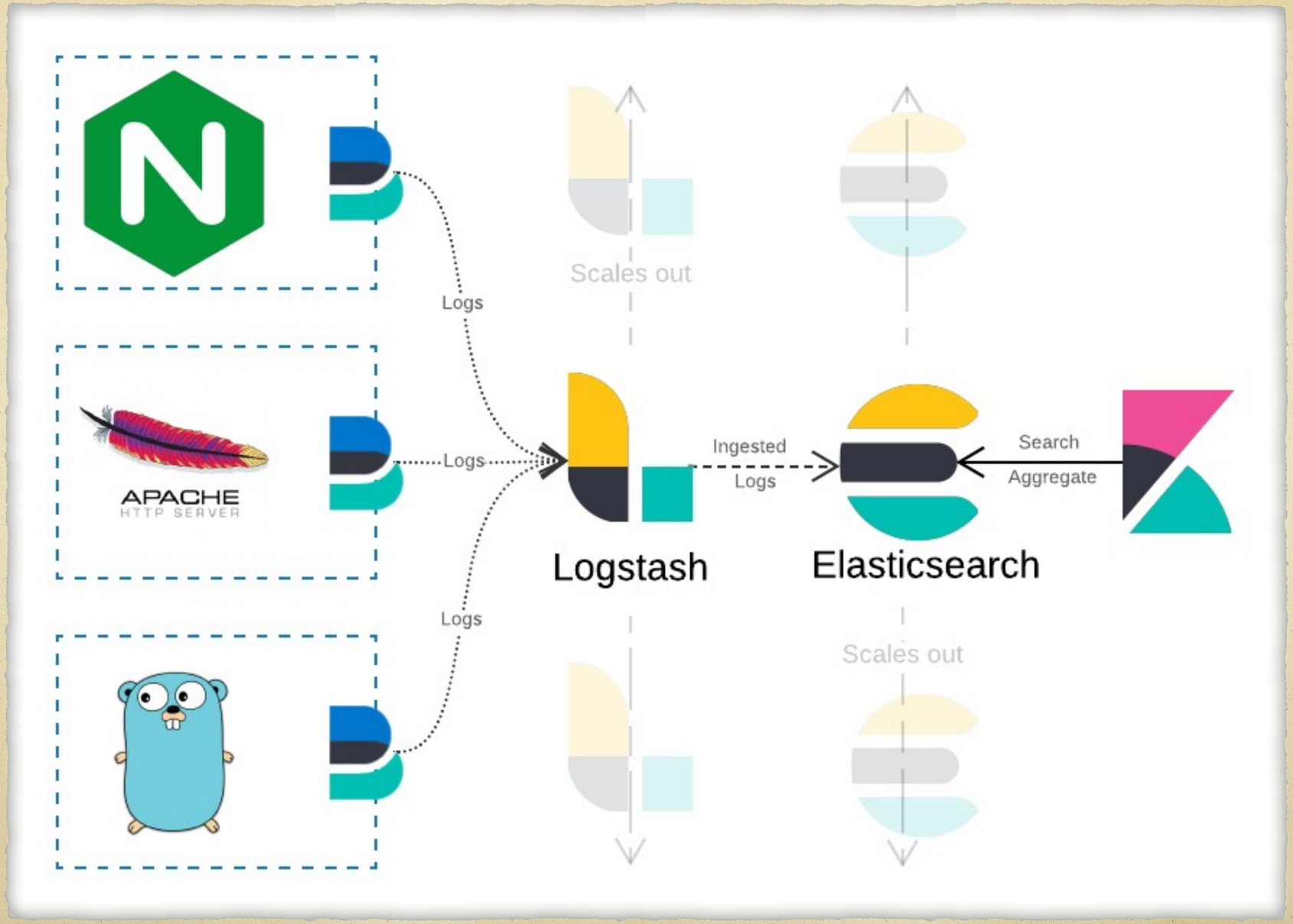
elasticsearch

search, analyze, and visualize that data in real time.

Surasuk Oakkharaamonphong
Technical Coach
Infinitas by Krungthai

Elastic Stack



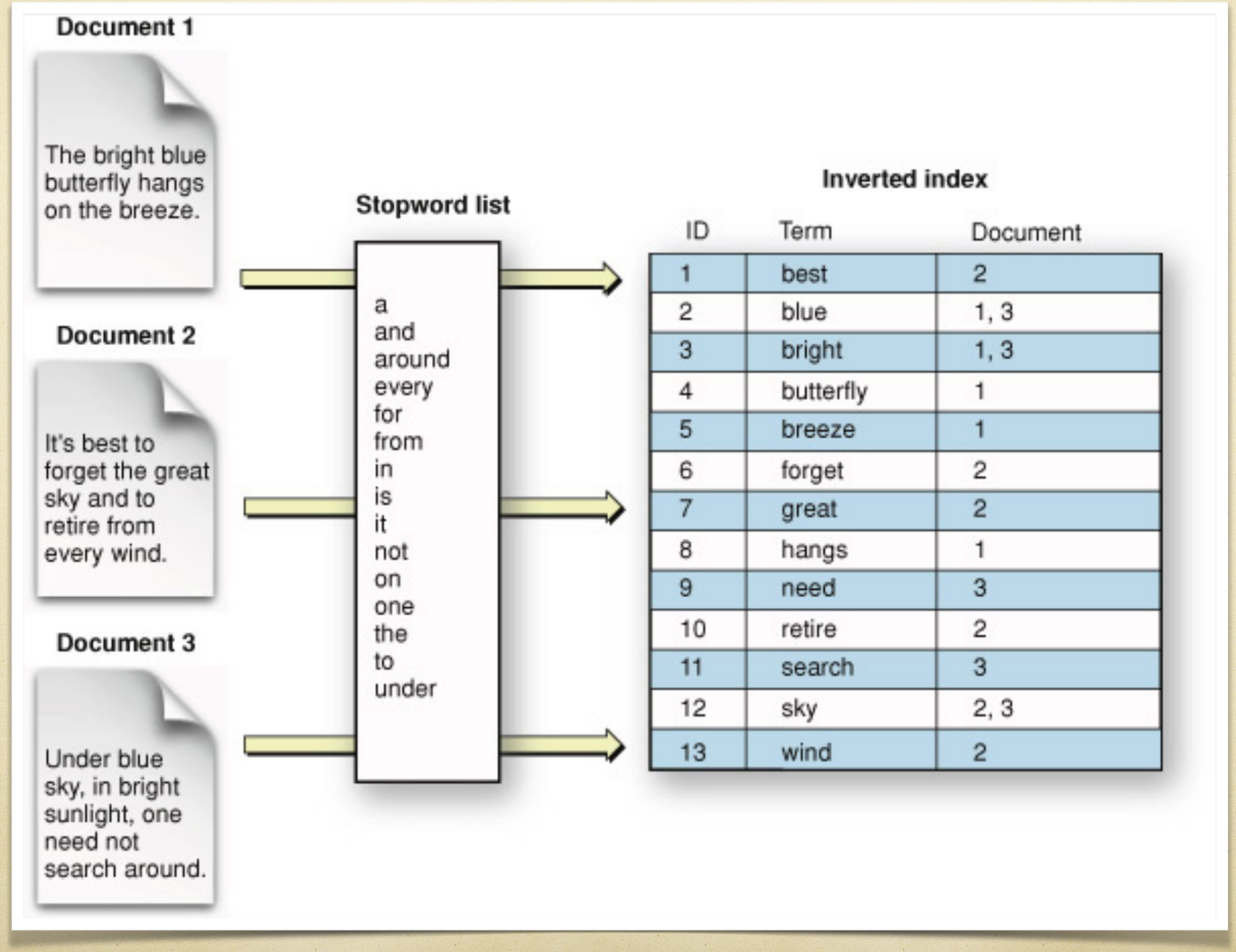

Logs
Logs
Logs
Scales out
Ingested
Logs
Logstash
Elasticsearch
Search
Aggregate
Scales out
APACHE
HTTP SERVER


Collect
&
Transform
Search
&
Analyze
Visualize
&
Manage
LOG
logstash
elasticsearch
kibana

Relational Database vs Elasticsearch

Relational Database	Elasticsearch
(Structured Query Language) SQL	DSL (Domain Specific Language)
Table	Index
Rows	Documents
Column	Field

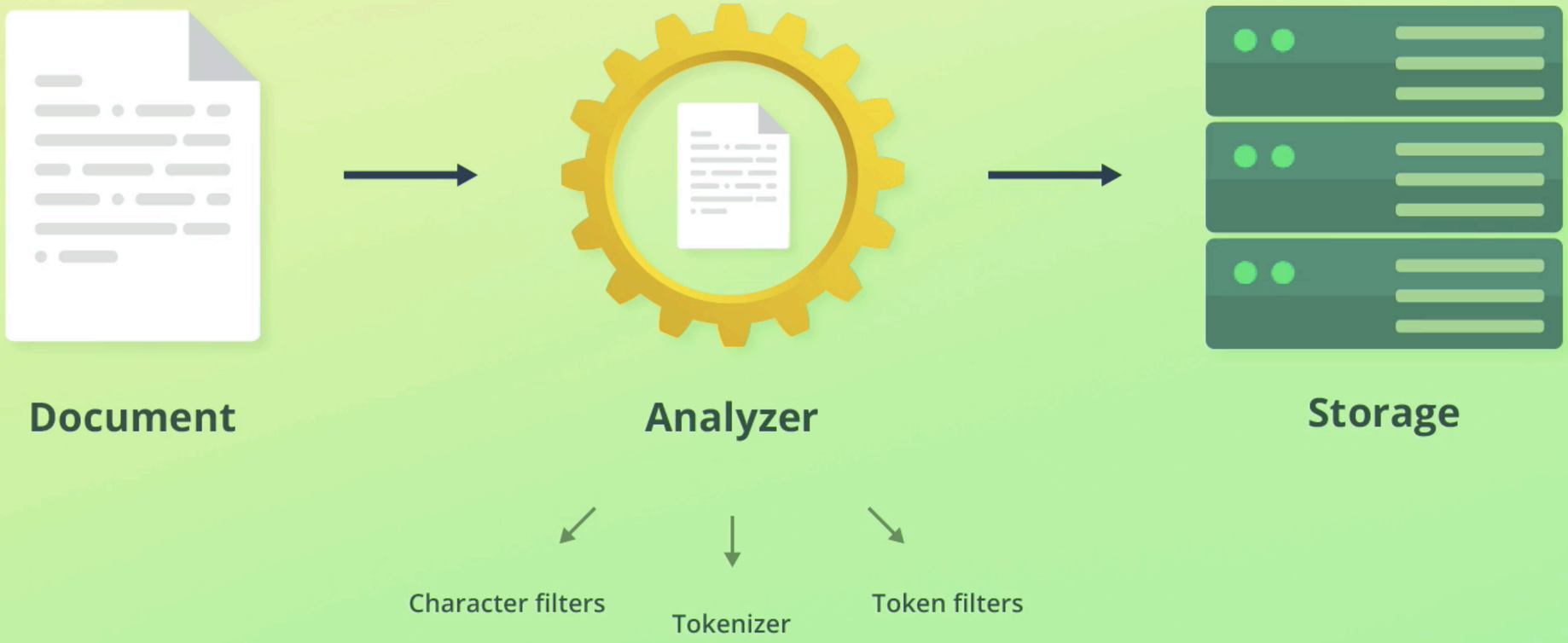
Inverted Index



TF-IDF

- **Term Frequency** ยิ่งมีคำในเอกสารมากเท่าไหร่ คะแนนก็ยิ่งมากเท่านั้น
- **Inverse Document Frequency** ยิ่งคำๆ นั้น ไปปรากฏบนเอกสารจำนวนน้อยเท่าไหร่ คะแนนก็ยิ่งมาก ยกตัวอย่างเช่น “กับ แก่ แต่ ต่อ และ ก็ ซึ่ง อัน” เป็นคำเชื่อมที่ไม่มีความหมายอะไร และมักจะปรากฏอยู่ในเอกสารทุกฉบับอยู่แล้ว ดังนั้น คำพวกนี้ ระบบจะไม่ให้น้ำหนักเลย

https://medium.com/machinereading/elasticsearch-ภาคลุยสนาม-ตอนที่-1-19077ab210b3


Document
Analyzer
Storage
Character filters
Tokenizer
Token filters

Character filters

- Adds, removes, or changes characters
- Analyzers contain zero or more character filters
- Character filters are applied in the order in which they are specified
- Example (`html_strip` filter)
  - **Input:** `"I&apos;m in a <em>good</em> mood - and I <strong>love</strong> açaí!"`
  - **Output:** `"I'm in a good mood - and I love açaí!"`

Tokenizers

- An analyzer contains **one** tokenizer
- Tokenizes a string, i.e. splits it into tokens
- Characters may be stripped as part of the tokenization
- Example
  - **Input:** `"I REALLY like beer!"`
  - **Output:** `["I", "REALLY", "like", "beer"]`

Token filters

- Receive the output of the tokenizer as input (i.e. the tokens)
- A token filter can add, remove, or modify tokens
- An analyzer contains zero or more token filters
- Token filters are applied in the order in which they are specified
- Example (`lowercase` filter)
  - **Input:** `["I", "REALLY", "like", "beer"]`
  - **Output:** `["i", "really", "like", "beer"]`

Character filters

(none)

Input: "I REALLY like beer!"

Output: "I REALLY like beer!"

Tokenizer

standard

Input: "I REALLY like beer!"

Output: ["I", "REALLY", "like", "beer"]

Token filters

["lowercase"]

Input: "I", "REALLY", "like", "beer"

Output: "i", "really", "like", "beer"

STANDARD ANALYZER

Inverted indices

- Mapping between terms and which documents contain them
- Outside the context of analyzers, we use the terminology “terms”

Sentence → Tokens

```
"2 guys walk into   a bar, but the third... DUCKS! :-)"
```

→

```
["2", "guys", "walk", "into", "a", "bar", "but", "the", "third", "ducks"]
```

TERM	DOCUMENT #1
2	X
a	X
bar	X
but	X
ducks	X
guys	X
into	X
the	X
third	X
walk	X

Sentence → Tokens

`"2 guys walk into  a bar, but the third... DUCKS! :-)"`	→	`["2", "guys", "walk", "into", "a", "bar", "but", "the", "third", "ducks"]`
`"2 guys went into a bar"`	→	`["2", "guys", "went", "into", "a", "bar"]`
`"2 ducks walk around the lake"`	→	`["2", "ducks", "walk", "around", "the", "lake"]`

TERM	DOCUMENT #1	DOCUMENT #2	DOCUMENT #3
2	X	X	X
a	X	X	
around			X
bar	X	X	
but	X		
ducks	X		X
guys	X	X	
into	X	X	
lake			X
the	X		X
third	X		
walk	X		X
went		X	

```
{
  "name": "Coffee Maker",
  "description": "Makes coffee super fast!",
  "price": 64,
  "in_stock": 10,
  "created_at": "2009-11-08T14:21:51Z"
}
```



```
{
  "name": "Toaster",
  "description": "Makes delicious toasts...",
  "price": 49,
  "in_stock": 4,
  "created_at": "2007-01-29T09:44:15Z"
}
```


name field

TERM	DOCUMENT #1	DOCUMENT #2
coffee	X	
maker	X	
toaster		X

description field

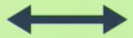
TERM	DOCUMENT #1	DOCUMENT #2
coffee		X
delicious	X	
fast		X
makes	X	X
super		X
toasts	X	

What is mapping?

- Defines the structure of documents (e.g. fields and their data types)
    - Also used to configure how values are indexed
- Similar to a table's schema in a relational database

```sql
CREATE TABLE employees (
  id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  dob DATE,
  description TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

MySQL



```
PUT /employees
{
  "mappings": {
    "properties": {
      "id": { "type": "integer" },
      "first_name": { "type": "text" },
      "last_name": { "type": "text" },
      "dob": { "type": "date" },
      "description": { "type": "text" },
      "created_at": { "type": "date" }
    }
  }
}
```

Elasticsearch

Queries

- **Full Text Queries:** Analysis is conducted on query before execution.
- **Term Queries:** No analysis is conducted on query before execution.

Full Text Queries

Elasticsearch queries where analysis is conducted on query before execution.

Full Text Queries

Elasticsearch queries where analysis is conducted on query before execution.

Built in Analyzers

Standard – default for all *text* fields

Simple – divides terms on any non letter character

Whitespace – divides terms on whitespace or (spaces)

Stop – supports the removal of words as defined by the user

Built in Analyzers

Keyword – outputs text as one term without modification

Pattern – filters based on regex pattern

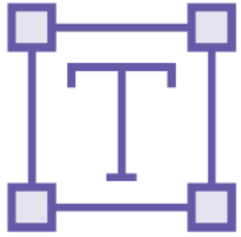
Language – specific to a written language

Fingerprint – text cluster algorithm from the OpenRefine project

Custom Analyzers

tokenizer

character filter



token filter

Pre-defined combinations of tokenizers, character filters and token filters are what make up the built-in analyzer types

Search and Query

Surasuk Oakkharaamonphong
Technical Coach
Infinitas by Krungthai

Understanding Relevance scores

Okapi BM25

The relevance scoring algorithm currently used by Elasticsearch.

Term Frequency (TF)

How many times does the term appear in the field for a given document?

Inverse Document Frequency (IDF)

How often does the term appear within the index (i.e. across all documents)?

Field-length norm

How long is the field?

Don’t use Elasticsearch as a primary data store

Elasticsearch only supports simple joins

Joins are expensive!

Documents must be stored within the same index

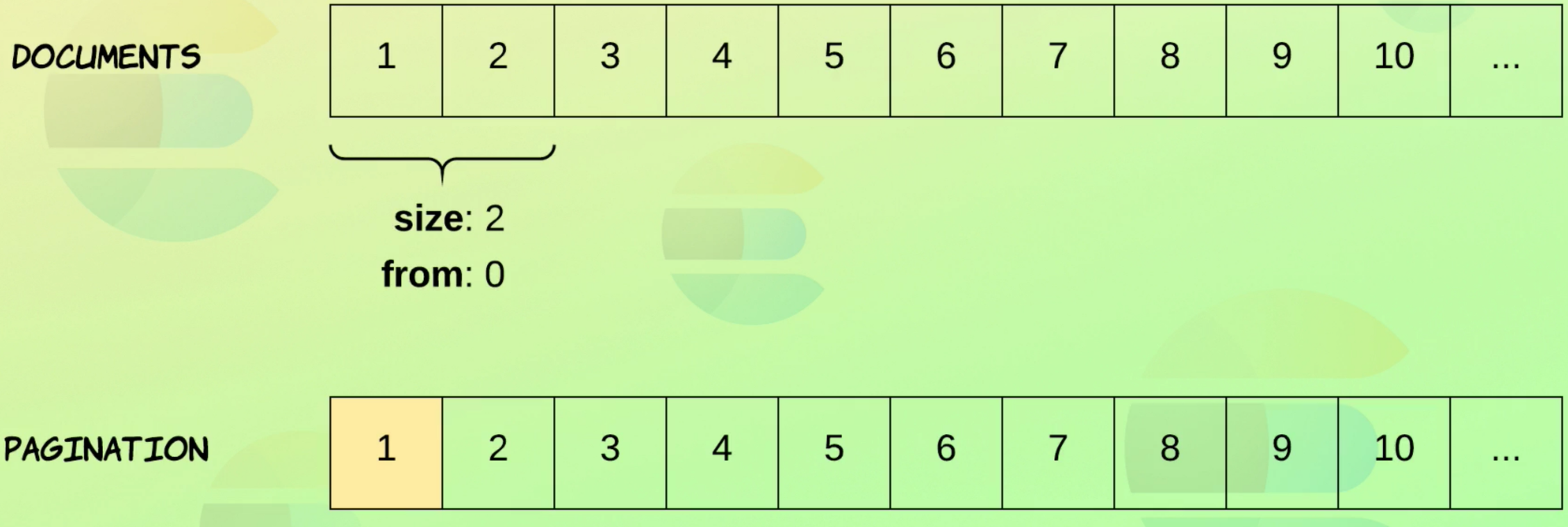
(for performance reasons)

Only one join field per index

Using join fields is slow!

total_pages = ceil(total_hits / page_size)

from = page_size * (page_number - 1)

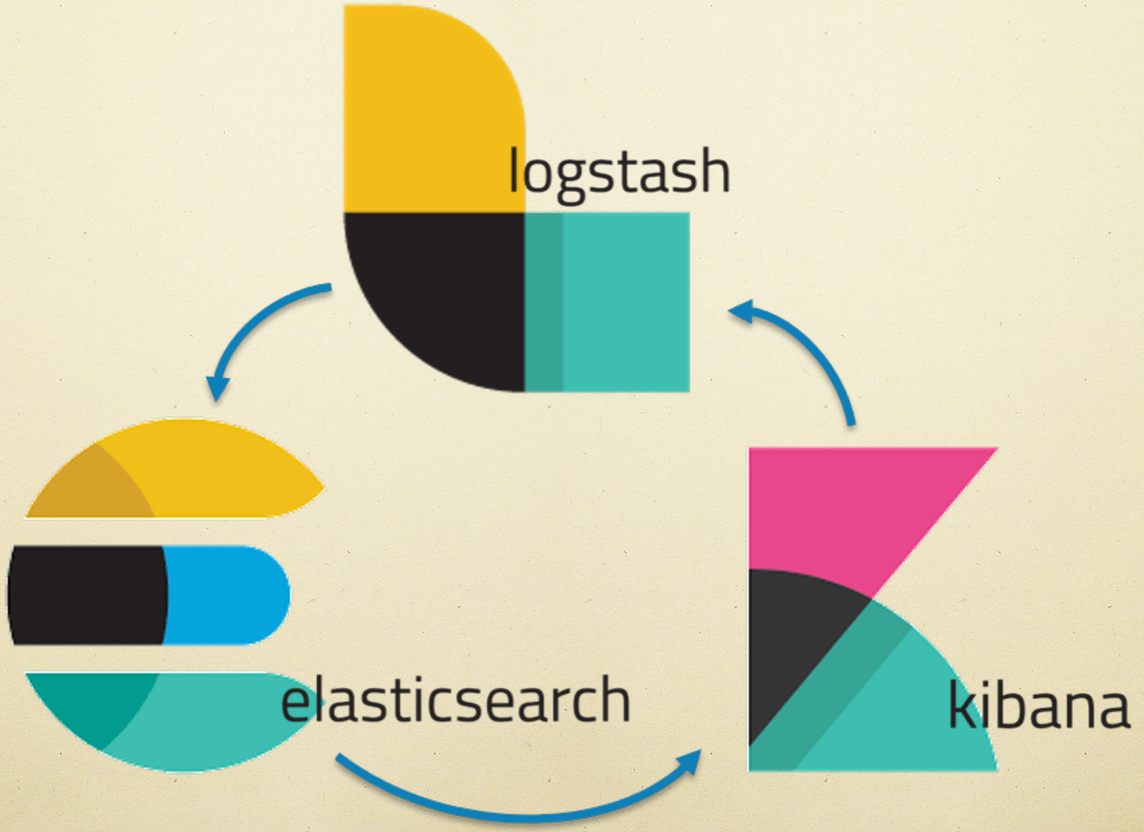

DOCUMENTS
1
2
3
4
5
6
7
8
9
10
...
size: 2
from: 0
PAGINATION
1
2
3
4
5
6
7
8
9
10
...

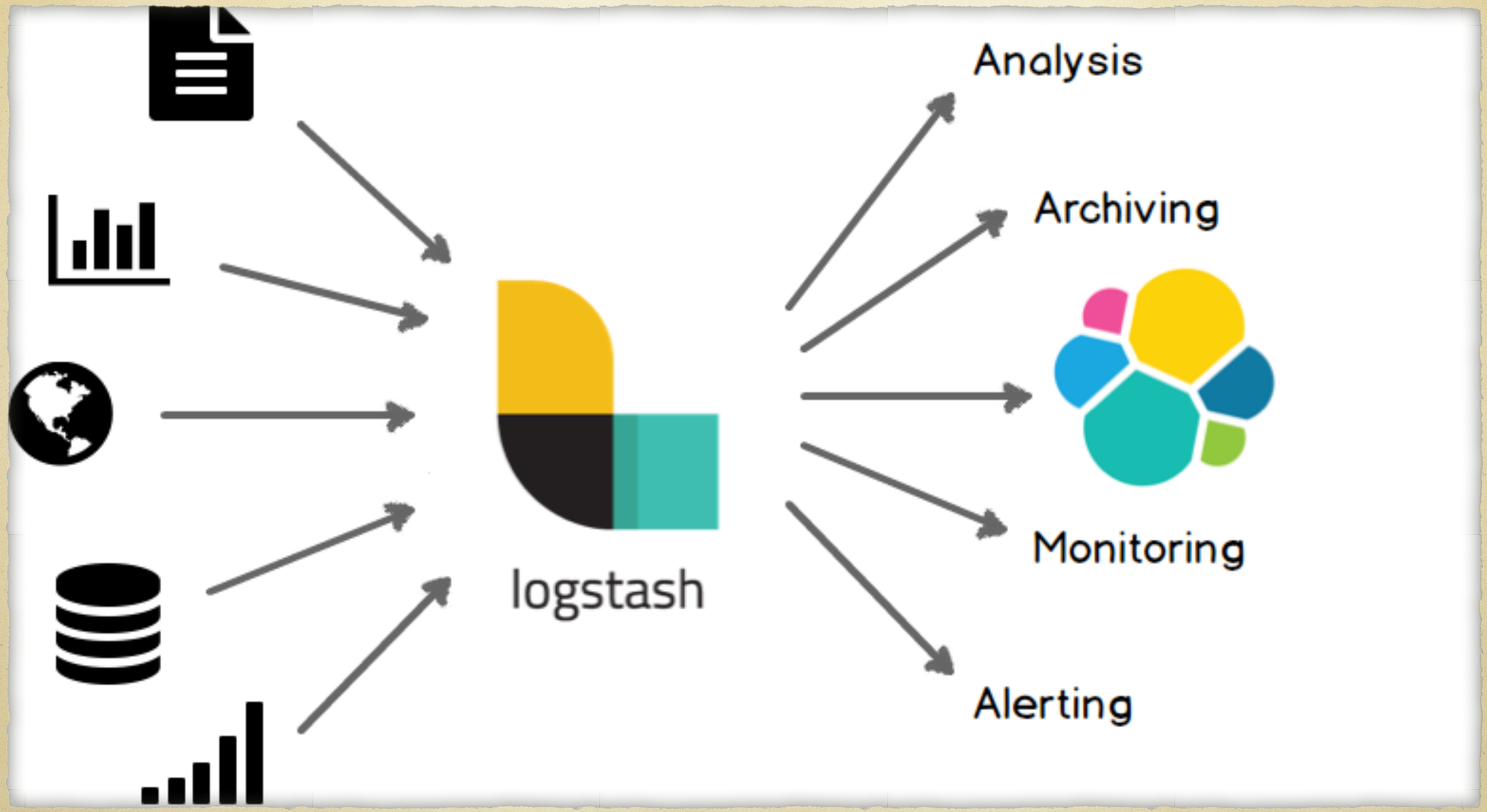
logstash

server-side data processing pipeline

Surasuk Oakkharaamonphong
Technical Coach
Infinitas by Krungthai

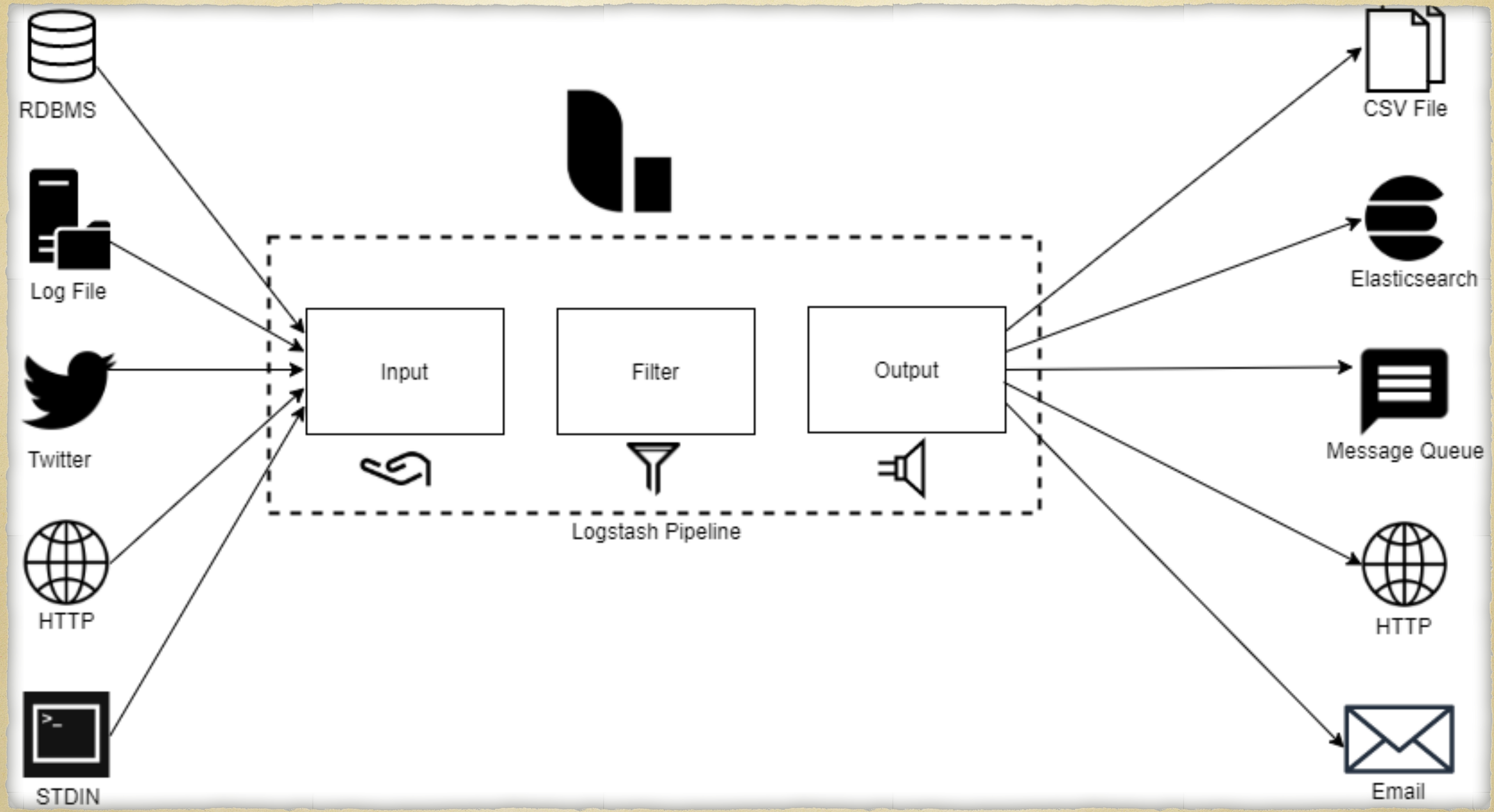
Elastic Stack




Analysis
Archiving
Monitoring
Alerting
logstash


Collect
&
Transform
Search
&
Analyze
Visualize
&
Manage
LOG
logstash
elasticsearch
kibana

Pipeline = input + (filter) + output


RDBMS
Log File
Twitter
HTTP
STDIN
Input
Filter
Output
Logstash Pipeline
CSV File
Elasticsearch
Message Queue
HTTP
Email

%{SYNTAX:SEMANTIC}

%{PATTERN:FIELD:TYPE}

Comparision Operator	==, !=, <, >, <=, >=
Regular Expression	=~, !~
Inclusion	in, not in
Boolean	and, or
Unary	!

%{+DATE_FORMAT}