CLAIRVOYANT

design engineer deliver

Scalable Search Systems

elasticsearch & Solr

Team
Clairvoyant India Pvt. Ltd.

CLAIRVOYANT

Presentation Agenda

Start

Search Concepts

Lucene
A closer look

Elastic Search
A closer look

Solr
A closer look

Elastic vs Solr
Evaluation criteria

Here is what we think

Your Thoughts

Conclusion

End

Search Concepts - Quick Example

http://www.amazon.in/Books/b?ie=UTF8&node=976389031

- Fielded searching (e.g. title, author, contents)
- Sorting by any field
- Merged results from different types of documents
- Powerful query types: range queries, phrase queries, proximity queries
- Relevant and ranked searching
- Flexible faceting, highlighting, joins and result grouping
- Fast, memory-efficient and typo-tolerant suggesters

Search Concepts - Aspects

- Store the Data
  - Text Acquisition
  - Text Transformation (Analyzers)
  - Organise the data (Index)
  - Deal with incremental data
  - Housekeeping

- Query the Data:
  - Parse the Query
  - Apply the transformations (Analyzers)
  - Search for the matches
  - Return the grouped results (aggregate)

Lucene - a closer look

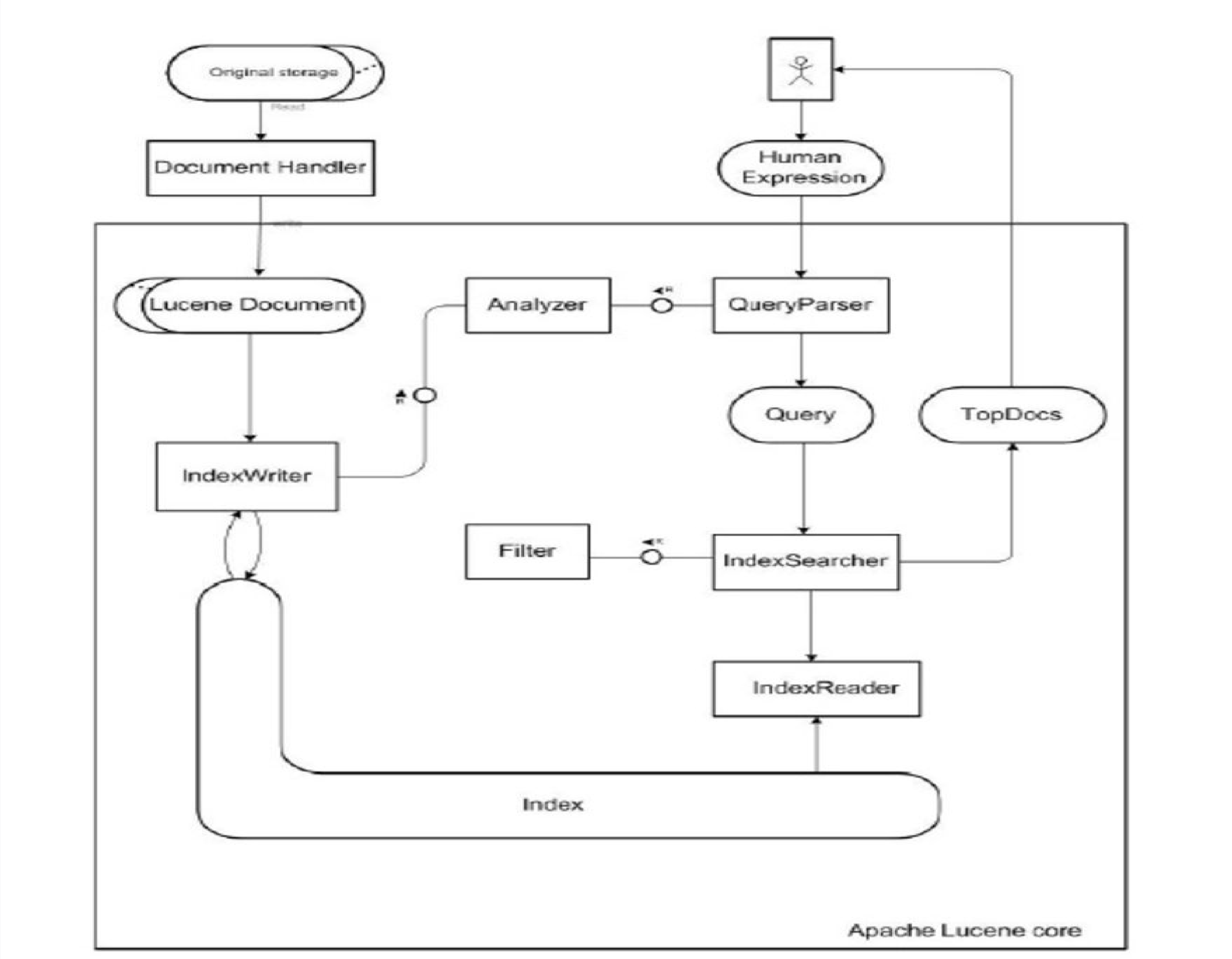
LUCENE

- Lucene Index Space
- Index
- Collection (of Documents)
- Lucene Document
- Document Field
- SQL Query

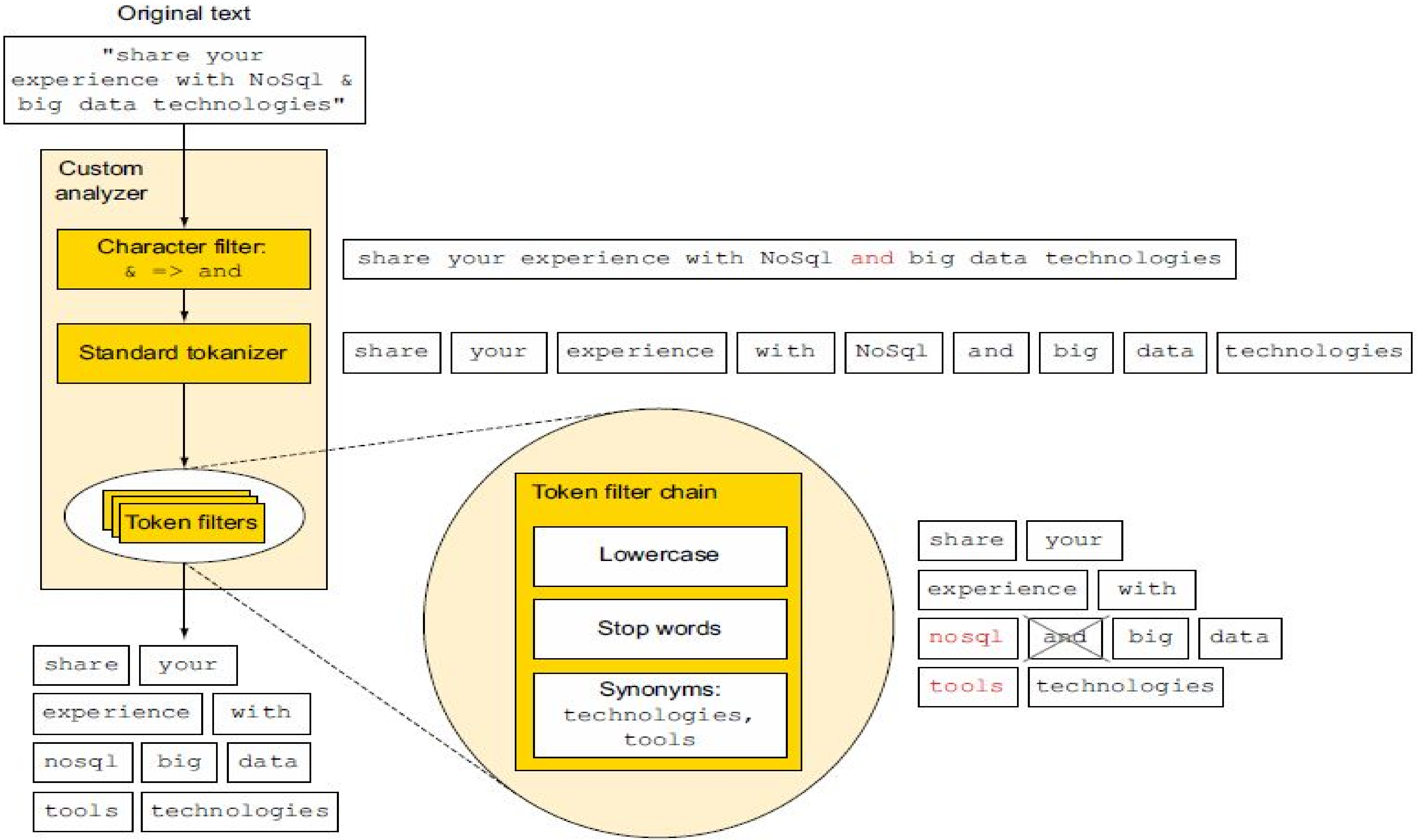
RDBMS

- Database
- Table Index
- Table (of tuples)
- Tuple
- Column
- Search Query

Lucene - a closer look



A N A L Y Z E R



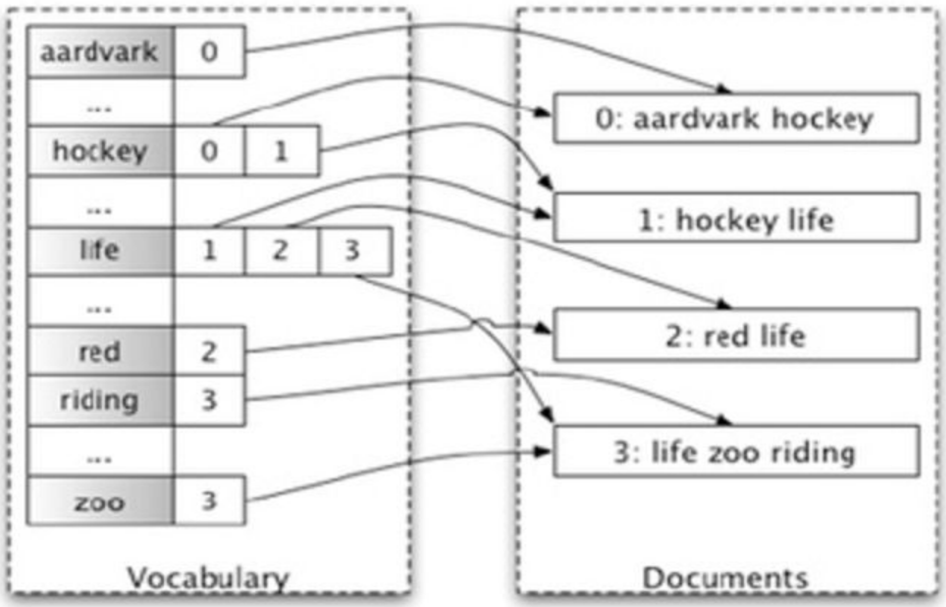
Original text
"share your experience with NoSql & big data technologies"
Custom analyzer
Character filter:
& => and
Standard tokanizer
Token filters
share your experience with NoSql and big data technologies
share
your
experience
with
NoSql
and
big
data
technologies
Token filter chain
Lowercase
Stop words
Synonyms:
technologies, tools
share
your
experience
with
nosql
and
big
data
tools
technologies
share
your
experience
with
nosql
big
data
tools
technologies

Lucene

Inverted Index

- Commonly used search engine data structure
- Efficient lookup of terms across large number of documents
- Usually stores positional information to enable phrase/proximity queries

From "Taming Text" by Grant Ingersoll and Tom Morton

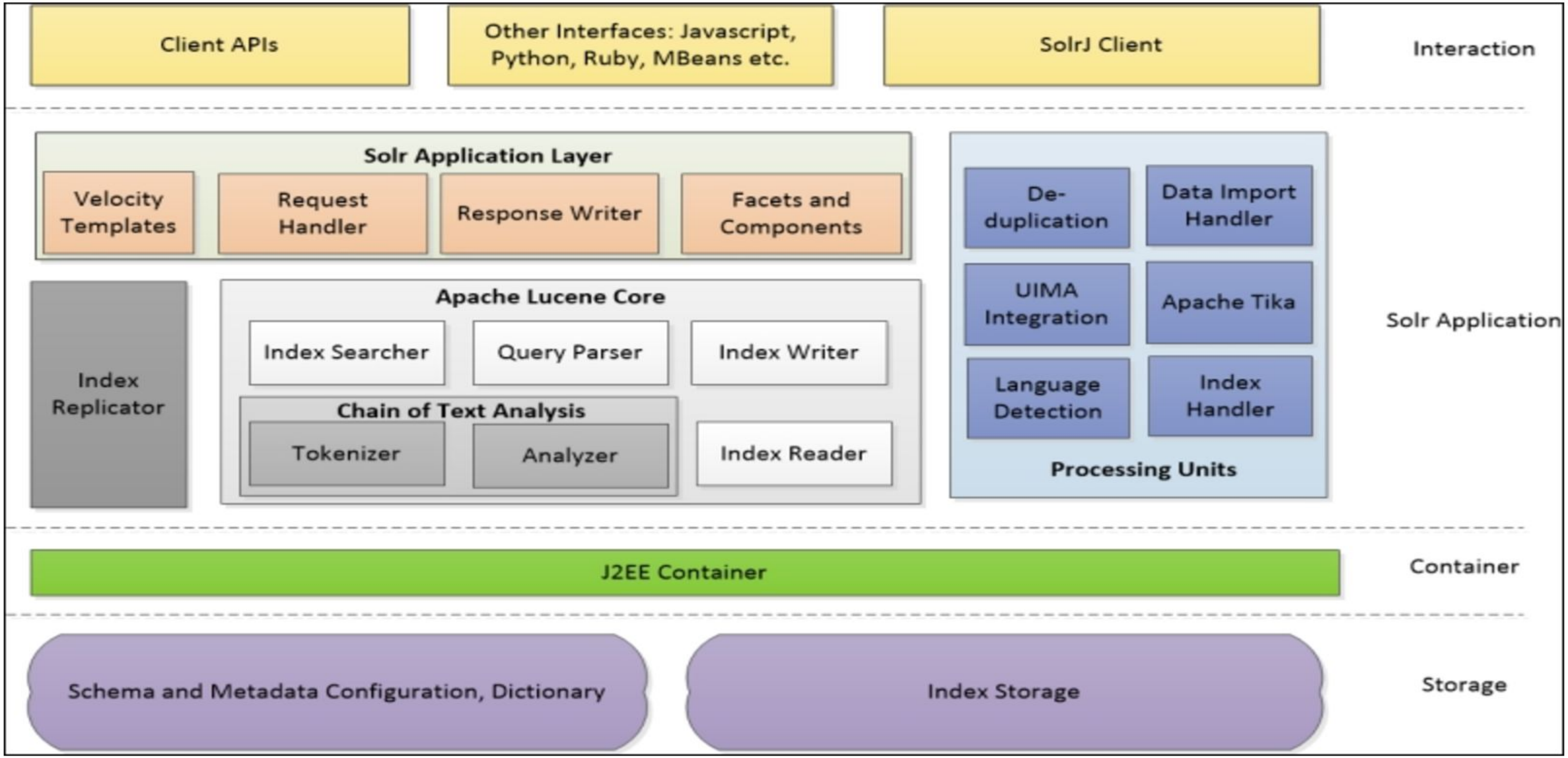


Solr - a closer look

- Created in 2004 by Yonik Seely, to add search capabilities to the company website CNET networks.

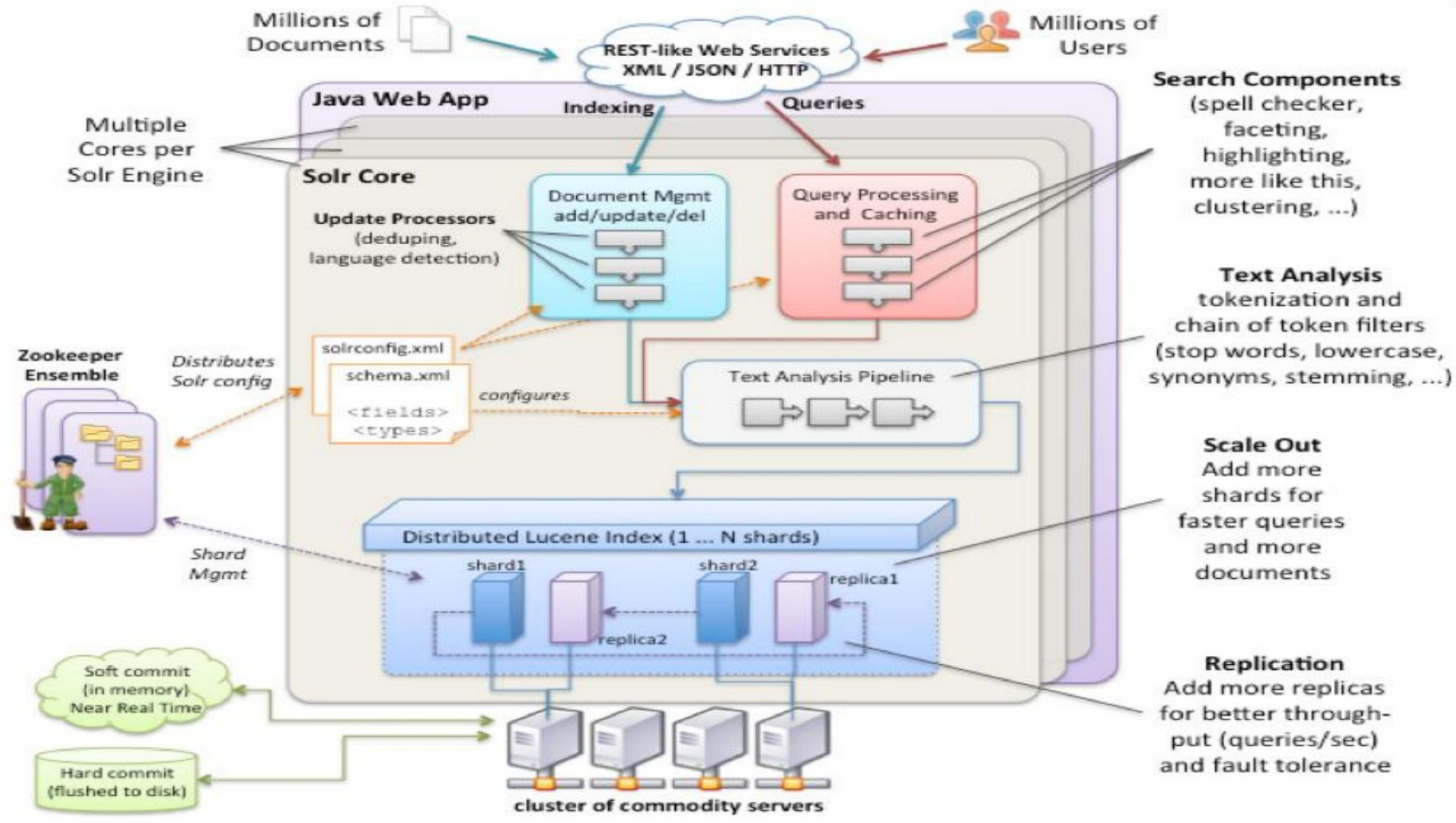
- Open source in 2006 under Apache Software Foundation.
- Latest Release Solr 6.0 in 2016.

Solr Architecture



Solr Architecture Flow

Solr Architecture



Source: **Solr In Action**

Elastic - a closer look

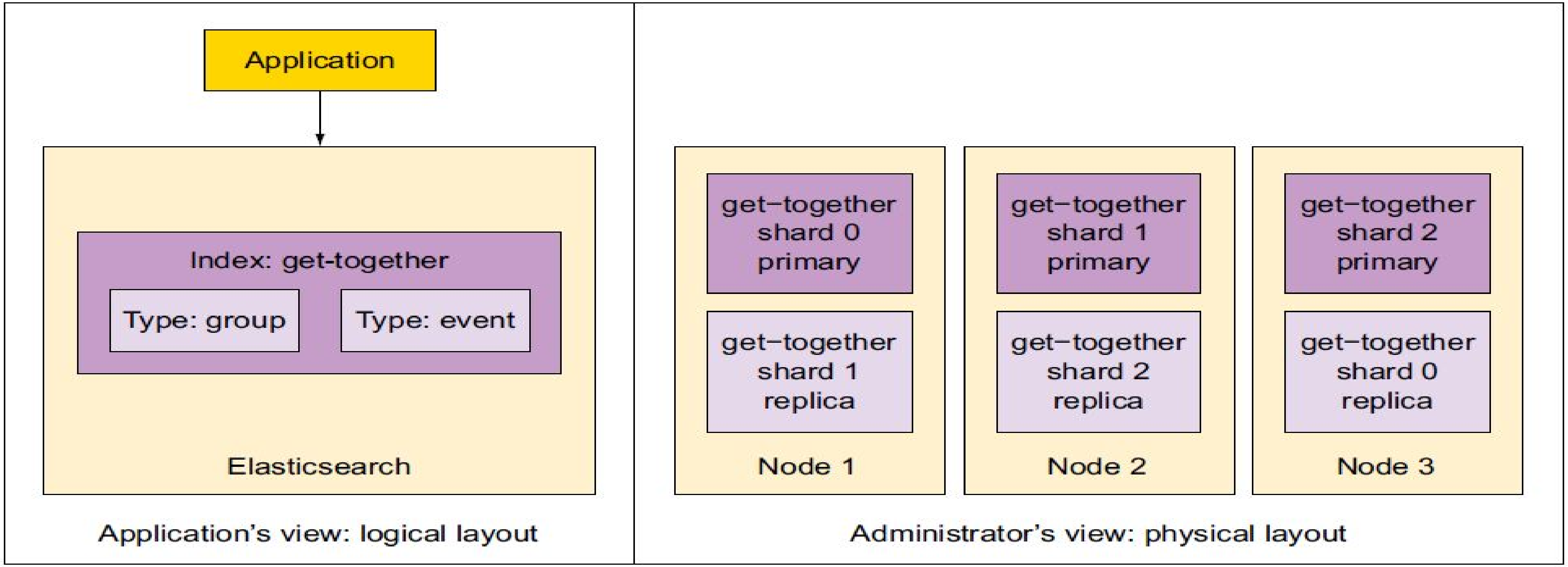
Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface

Shay Banon created the precursor to Elasticsearch, called Compass, in 2004

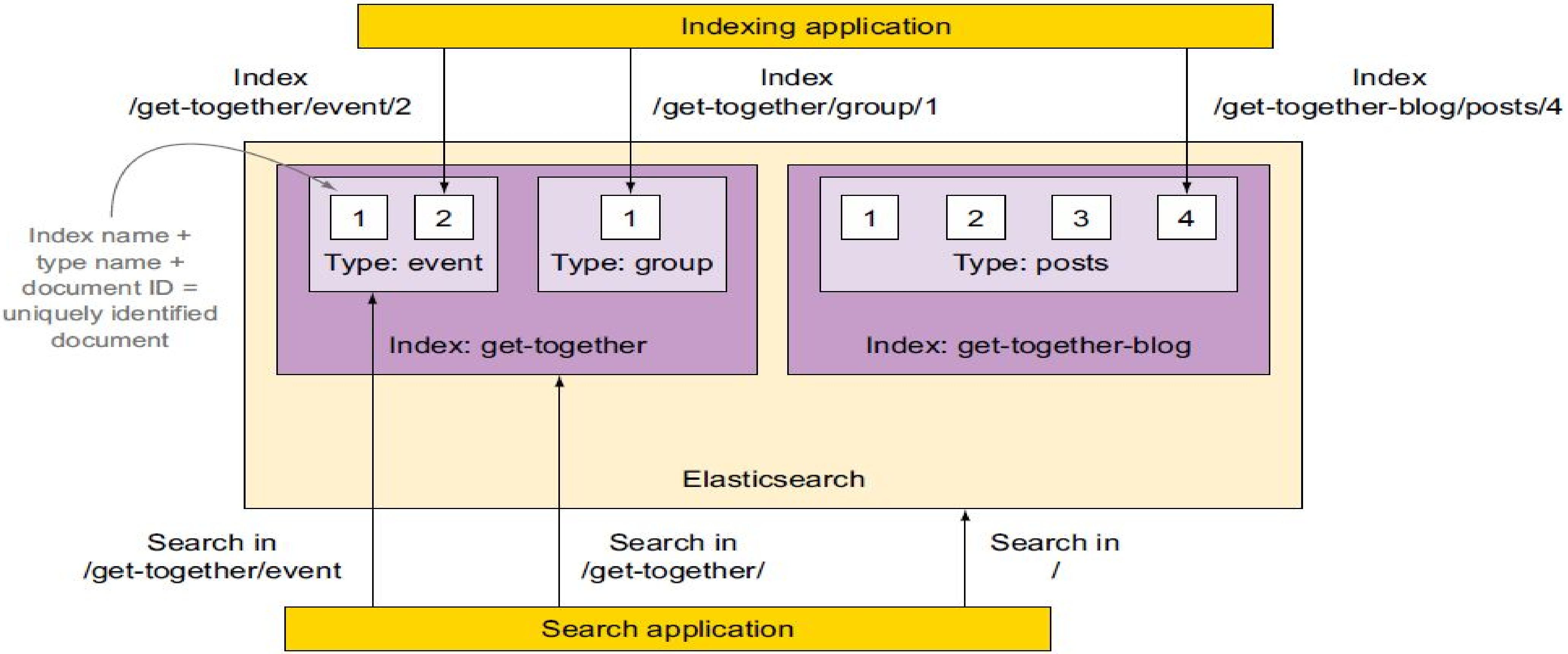
to "create a scalable search solution", Shay Banon released the first version of Elasticsearch in February 2010

In March 2015, the company Elasticsearch changed their name to Elastic

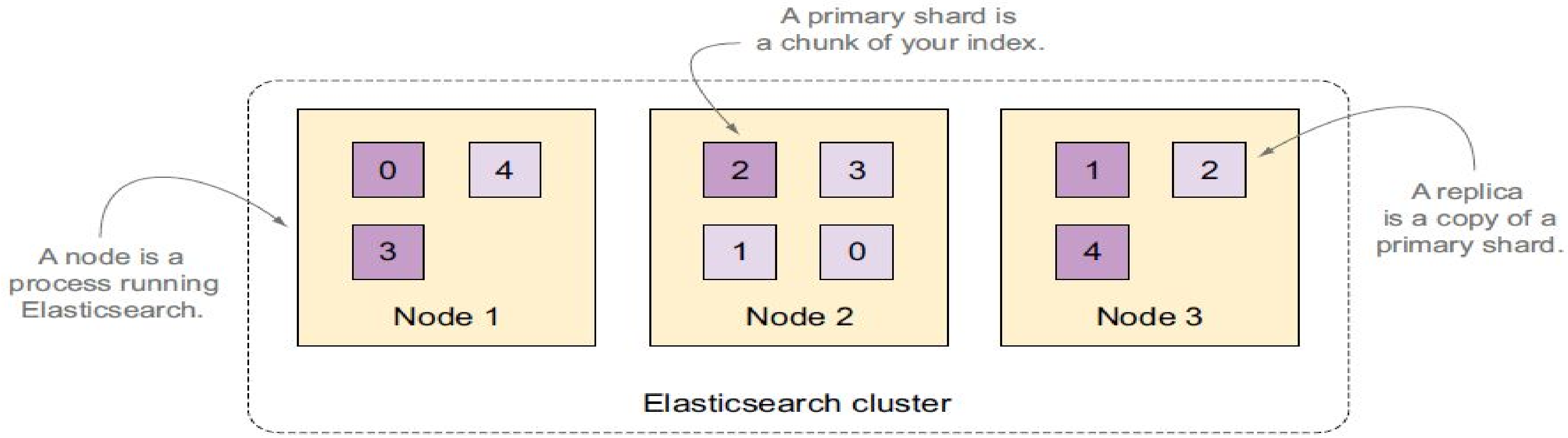
Logical and Physical Layout



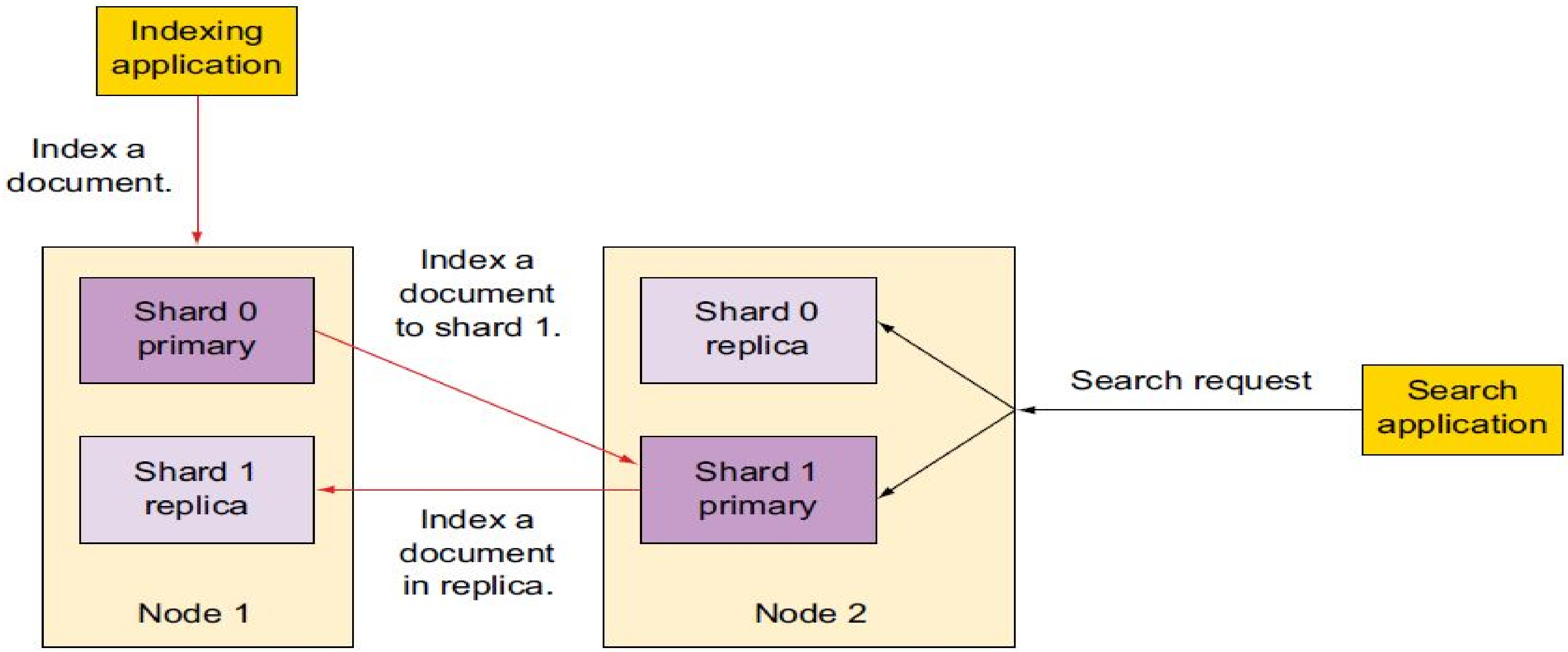
Logical structure



Physical: 5 shard 1 replica



Indexing


Indexing application
Index a document.
Shard 0 primary
Shard 1 replica
Node 1
Index a document to shard 1.
Index a document in replica.
Shard 0 replica
Shard 1 primary
Node 2
Search request
Search application

Term Frequency: Lucene Index

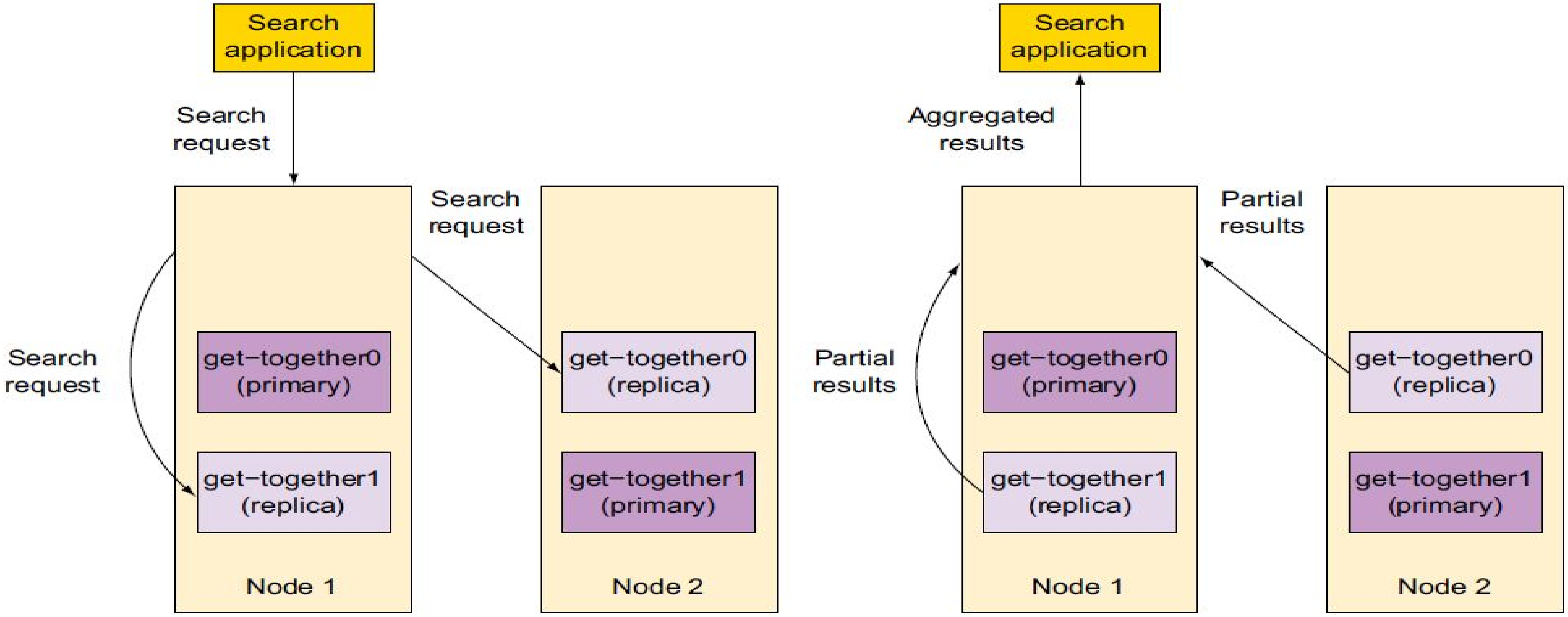
A shard is a Lucene index.

get-together0 shard

Inverted index

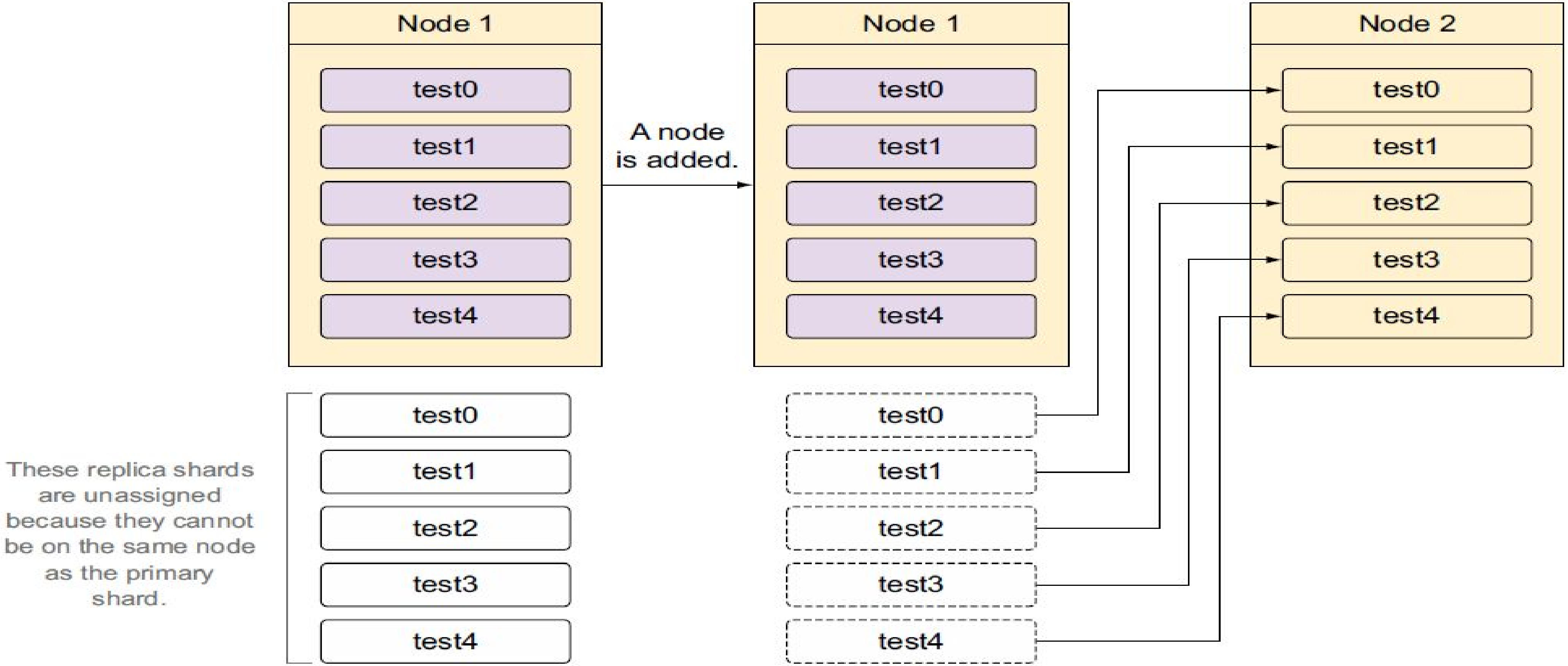
Term	Document	Frequency
elasticsearch	id1	1 occurrence: id1->1 time
denver	id1,id3	3 occurrences: id1->1 time, id3->2 times
clojure	id2,id3	5 occurrences: id2->2 times, id3->3 times
data	id2	2 occurrences: id2->2 times

Searching



Search application
Search request
Search request
Search request
get−together0 (primary)
get−together1 (replica)
Node 1
get−together0 (replica)
get−together1 (primary)
Node 2
Search application
Aggregated results
Partial results
Partial results
get−together0 (primary)
get−together1 (replica)
Node 1
get−together0 (replica)
get−together1 (primary)
Node 2

Scaling : Adding Node



Scaling

- Over sharding
- Load balancing
- Alias
- Routing

Routing

```
% curl -XPOST 'localhost:9200/get-together/group/10?routing=denver' -d'
{
  "name": "Denver Ruby",
  "description": "The Denver Ruby Meetup"
}'
```

Indexing a document with a routing value of denver

```
% curl -XPOST 'localhost:9200/get-together/group/11?routing=boulder' -d'
{
  "name": "Boulder Ruby",
  "description": "Boulderites that use Ruby"
}'
```

Indexing a document with the routing value boulder

```
% curl -XPOST 'localhost:9200/get-together/group/12?routing=amsterdam' -d'
{
  "name": "Amsterdam Devs that use Ruby",
  "description": "Mensen die genieten van het gebruik van Ruby"
}'
```

```
% curl -XPOST 'localhost:9200/get-together/group/
     _search?routing=denver,amsterdam' -d'
{
  "query": {
    "match": {
      "name": "ruby"
    }
  }
}'
```

Executing a query with a routing value of denver and amsterdam

Differences between Elastic and Solr

Category	Solr	Elastic
Installation/Configuration/maintenance	Heavy (Jar size 150 MB)	Light Wiegth (Jar size 26 MB)
	Comments allowed in Config files	Comments not allowed. It becomes difficult to understand the config as it grows (or to new people)
		If the overall application is using JSON heavily, go with Elastic
	Everything is configuration driven	Magic (but it comes at a cost)
	AdminUI	Marvel (but costs for production use)

Differences between Elastic and Solr

Category	Solr	Elastic
Collections	Simple Collections	Collections/Types (sometimes having same names type fields may cause issues)
	Sold does not index new collection automatically	Elastic can index the collections automatically based on the index_templates
Indexing/Searching	Strong at Text Search	In addition to Text search, analytical querying, filtering, and grouping are supported well.
	Control on the order of Filters in Analyzers is limited	More Control on the order of Filters in Analyzers
	Faceting	Aggregations (a super set of Facets, has bucketing, metric, matrix and pipelines)
	Relevance, Ranking, Caching (BitSet, FieldValue), Filtering	Same. This is because those are features from Lucene lib

Differences between Elastic and Solr

Category	Solr	Elastic
Related documents querying	The join index has to be a single-shard and replicated across all nodes to search inter-document relationships	Inner Documents, Nested Documents, Parent_Child documents retrieve related documents using has_children and top_children queries that make it more efficient.
	Nested Documents queries are faster in Solr	Slower here
Monitoring	Percolation Queries not supported	Percolation Queries supported
	JMX Support	Stats API

Differences between Elastic and Solr

Category	Solr	Elastic
Distributed Deployment	Sold Cloud. Depends on Apache ZooKeeper Zookeeper is a mature application	ZooKeeper-like component called Zen need to deal with Split Brain problem carefully
Scaling	Shard splitting is allowed	Shard splitting is not allowed but encourages to go with automatic shard rebalancing
	Shard rebalancing is not supported in Solr	Shard rebalancing is supported
	Adding new shards and replicas is supported	Adding new shards is not supported. However, adding new replicas is supported
Backup and Restore	Solr has different APIs for standalone and cluster modes. This could be a problem from a maintanence perspective	A standard API for backup and restore. This is due to the fact that the system has been build ground up with clustering in mind

Differences between Elastic and Solr

Category	Solr	Elastic
Open Source - contribution	Open for the community and the committee members are from various companies. Less monopoly	Open source code and contributions are welcomed by anyone. But final decisions are made by Elastic. More of Company driven decisions.
Documentation	Sold is very well documented. More consistent	Not so well documented. Inconsistent in the examples vs whats mentioned in the documentation

What to choose when

Category	Solr	Elastic
Our Thoughts	Solr primarily concentrates on Search	Elastic is more into Search, Analytics
	Solr's focus is on building the core search features and making the same work at scale	Elastic's focus ~~has~~ is on building an echosystem of log/text files analysis most of which is commercialised
	Solr is to explorers (to create new platforms ground up)	Elastic is to Application developers
	Solr is to D3/HTML	Elastic is to Tableau

THANK YOU

References

https://www.elastic.co/guide/en/elasticsearch/guide/current/nested-aggregation.html

https://www.youtube.com/watch?v=yqAKfwGZpE0

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html#_structuring_aggregations

http://logz.io/blog/solr-vs-elasticsearch/

http://www.searchtechnologies.com/blog/solr-vs-elasticsearch-top-open-source-search

https://www.hcltech.com/blogs/elasticsearch-vs-solr

https://www.datanami.com/2015/01/22/solr-elasticsearch-question/

http://solr-vs-elasticsearch.com/

http://www.solrtutorial.com/solr-vs-elasticsearch.html

http://blog.florian-hopf.de/2014/02/elasticsearch-is-distributed-by-default.html

https://www.thinkbiganalytics.com/2013/07/10/solr-vs-elastic-search/

BACKUP SLIDES

Caching

- This cache is allocated at the node level, like the index buffer size you saw earlier. It defaults to 10%,but you can change it from elasticsearch.yml according to your needs
- Elasticsearch caches queries automatically based on usage frequency. If a non-scoring query has been used a few times (dependent on the query type) in the last 256 queries, the query is a candidate for caching. However, not all segments are guaranteed to cache . Only segments that hold more than 10,000 documents (or 3% of the total documents, whichever is larger) will get cached.
- Eviction is done on an LRU basis

Caching :bitset [1,0,0,0]

```
POST /my_store/products/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3" }
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5" }
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7" }
{ "index": { "_id": 4 }}
{ "price" : 30, "productID" : "QQPX-R-3956-#aD8" }
```

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "productID" : "XHDK-A-1293-#fJ3"
                }
            }
        }
    }
}
```

Caching:bitsets

- They're compact and easy to create, so the overhead of creating the cache when the filter is first run is insignificant.
- They're stored per individual filter; for example, if you use a term filter in two different queries or within two different bool filters, the bitset of that term can be reused.
- They're easy to combine with other bitsets. If you have two queries that use bitsets,it's easy for Elasticsearch to do a bitwise AND or OR in order to figure out which documents match the combination.
- Term, terms, exists/missing, prefix provide bitset feature

Taking advantage of bitsets

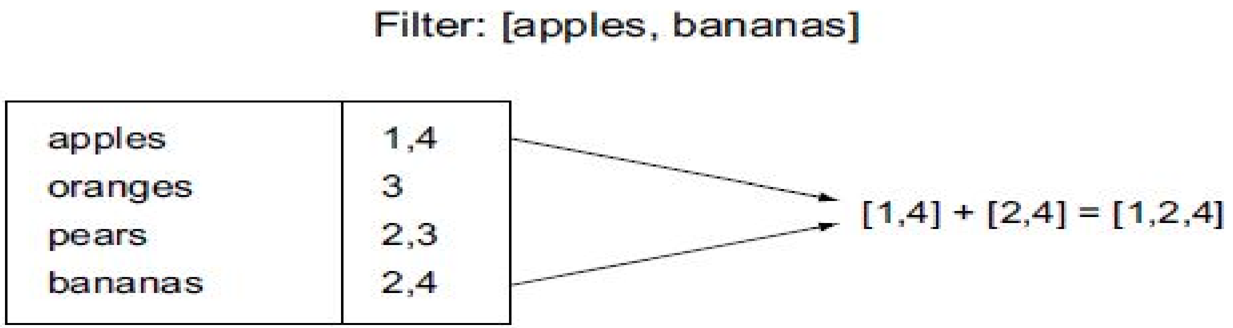
```
"filter": {
  "bool": {
    "should": [
      {
        "term": {
          "tags.verbatim": "elasticsearch"
        }
      },
      {
        "term": {
          "members": "lee"
        }
      }
    ]
  }
}
```

No bitsets: Caching overall results

```
"filter": {
  "and": [
        {
          "has_child": {
            "type": "event",
            "filter": {
              "range": {
                "date": {
                  "from": "2013-07-01T00:00",
                  "to": "2013-08-01T00:00"
                }
              }
            }
          }
        },
        {
          "script": {
            "script": "doc['members'].values.length > minMembers",
            "params": {
              "minMembers": 2
            }
          }
        }
    ]
}
```

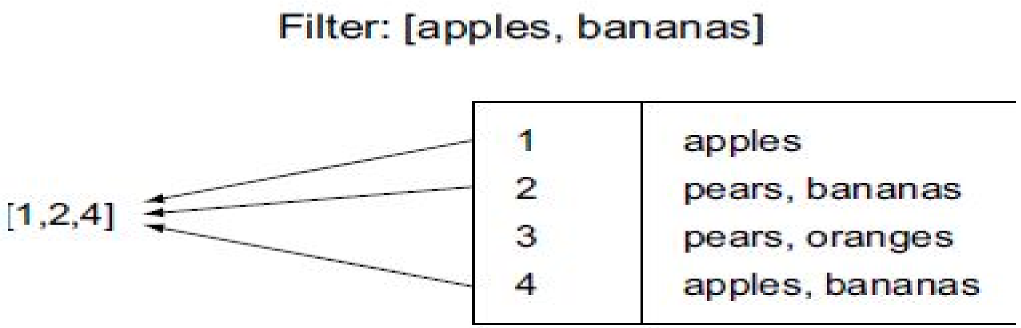
Caching :Field data

- Using inverted index



Caching :Field data

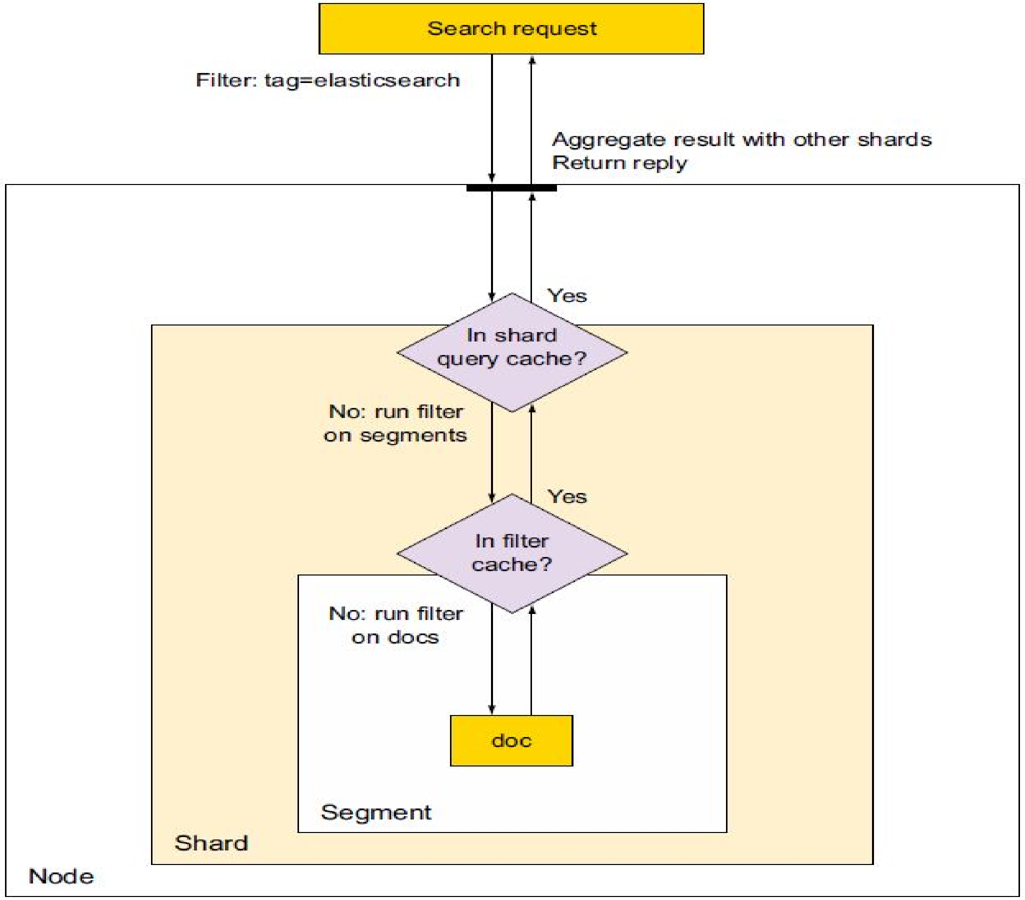
- Mapping documents to terms when we have lots of terms, and a range filter with a wide range, sort operation or an aggregation




```
"filter": {
  "range": {
    "date": {

      "gte": "2013-01-01T00:00",
      "lt": "2014-01-01T00:00"
    },
    "execution": "fielddata"
  }
}
```

Shard query cache



Shard query cache

- The shard query cache entries differ from one request to another, so they apply only to a narrow set of requests
- This narrowness of cache entries makes the shard query cache valuable only when shards rarely change and you have many identical requests.
- For example, if you're indexing logs and have time-based indice, older indices are ideal candidates for a shard query cache.s

Cache warmers/hot cache

```
curl -XPUT 'localhost:9200/hot_index' -d '{
"warmers": {
  "date sorting": {

      "types": [],
      "source": {
        "sort": [{
          "date": {
            "order": "desc"
          }
        }]
      }
    }
  }}'
```

Name of this warmer. You can register multiple warmers, too.

Which types this warmer should run on. Empty means all types.

This warmer sorts by date.

Under this key define the warmer itself.

Cache warmers/hot cache

- A *warmer : it can contain queries, filters,* sort criteria, and aggregations.
- Makes Elasticsearch run the query with every refresh operation. This will slow down the refresh, but the user queries will always run on “warm” caches.
- Useful when first-time queries are too slow and it’s preferable for the refresh operation to take that hit rather than the user(Slower refreshes shouldn’t concern you too much, searched for more often than they’re modified.)

Relationships

- Nested Types
- Parent child
- Update the parent document without reindexing the children.
- Get child documents in search request results.
- Add, change, or delete child documents without affecting the parent—or other children. This is especially useful for large collections of child documents that require frequent changes.

Nested Type

```
curl -XPUT localhost:9200/get-together/_mapping/group-nested -d '{
  "group-nested": {
    "properties": {
      "name": { "type": "string" },

       "members": {
         "type": "nested",
         "properties": {
           "first_name": { "type": "string" },
           "last_name": { "type": "string" }
         }
       }
     }
   }
}'
curl -XPUT localhost:9200/get-together/group-nested/1 -d '{
  "name": "Elasticsearch News",
  "members": [
    {
      "first_name": "Lee",
      "last_name": "Hinman"
    },
    {
      "first_name": "Radu",
      "last_name": "Gheorghe"
    }
  ]
}'
```

This signals Elasticsearch to index members objects in separate documents of the same block.

This property goes in the main document.

These objects go into their own documents, part of the same block as the root document.

Nested Type

- Internally in same Lucene block

first_name: Lee last_name: Hinman	first_name: Radu last_name: Gheorghe	name: Elasticsearch news Previous 2 documents are members

Parent child

```
# from mapping.json
    "event" : {
      "_source" : {
        "enabled" : true
      },
      "_all" : {
        "enabled" : false
      },
      "_parent" : {
        "type" : "group"
      },
      "properties" : {
```

Mapping for the event type starts here.

parent points to the group type.

Properties (fields) of the event type start here.

```
% curl -XPOST 'localhost:9200/get-together/event/1103?parent=2' -d '{
  "host": "Radu,
  "title": "Yet another Elasticsearch intro in Denver"
}'

% curl 'localhost:9200/get-together/event/1103?parent=2&pretty'
{
  "_index" : "get-together",
  "_type" : "event",
  "_id" : "1103",
  "_version" : 1,
  "found" : true, "_source" : {
  "host": "Radu",
  "title": "Yet another Elasticsearch intro in Denver"
  }
}
```

Nested type: Pro Cons

Using nested type to define document relationships: pros and cons

Before moving on, here's a quick recap of why you should (or shouldn't) use nested documents. The plus points:

- Nested types are aware of object boundaries: no more matches for "Radu Hinman"!
- You can index the whole document at once, as you would with objects, after you define your nested mapping.
- Nested queries and aggregations join the parent and child parts, and you can run any query across the union. No other option described in this chapter provides this feature.
- Query-time joins are fast because all Lucene documents making the Elasticsearch document are together in the same block in the same segment.
- You can include child documents in parents to get all the functionality from objects if you need it. This functionality is transparent for your application.

The downsides:

- Queries will be slower than their object equivalents. If objects provide you all the needed functionality, they're the better option because they're faster.
- Updating a child will re-index the whole document.

Parent child : Pro cons

Using parent-child designation to define document relationships: pros and cons

Before moving on, here's a quick recap of why you should or shouldn't use parent-child relationships. The plus points:

- Children and parents can be updated separately.
- Query-time join performance is better than if you did joins in your application because all related documents are routed to the same shard and joins are done at the shard level without adding network hops.

The downsides:

- Queries are more expensive than the nested equivalent and need more memory than field data.
- Aggregations can only join child documents to their parents and not the other way around, at least up to version 1.4.

Search Concepts - Index Creation

- Factors for indexing
  - Merge factors: How to merge new data into the index?
  - Storage techniques: Is the index compressed or not?
  - Index Size: How much size the index takes on disk?
  - Lookup speed : For information retrieval
  - Fault tolerance :   index corruption, dealing with bad hardward
    - partitioning, replication
  - Maintenance: Cost of maintaining an index over time