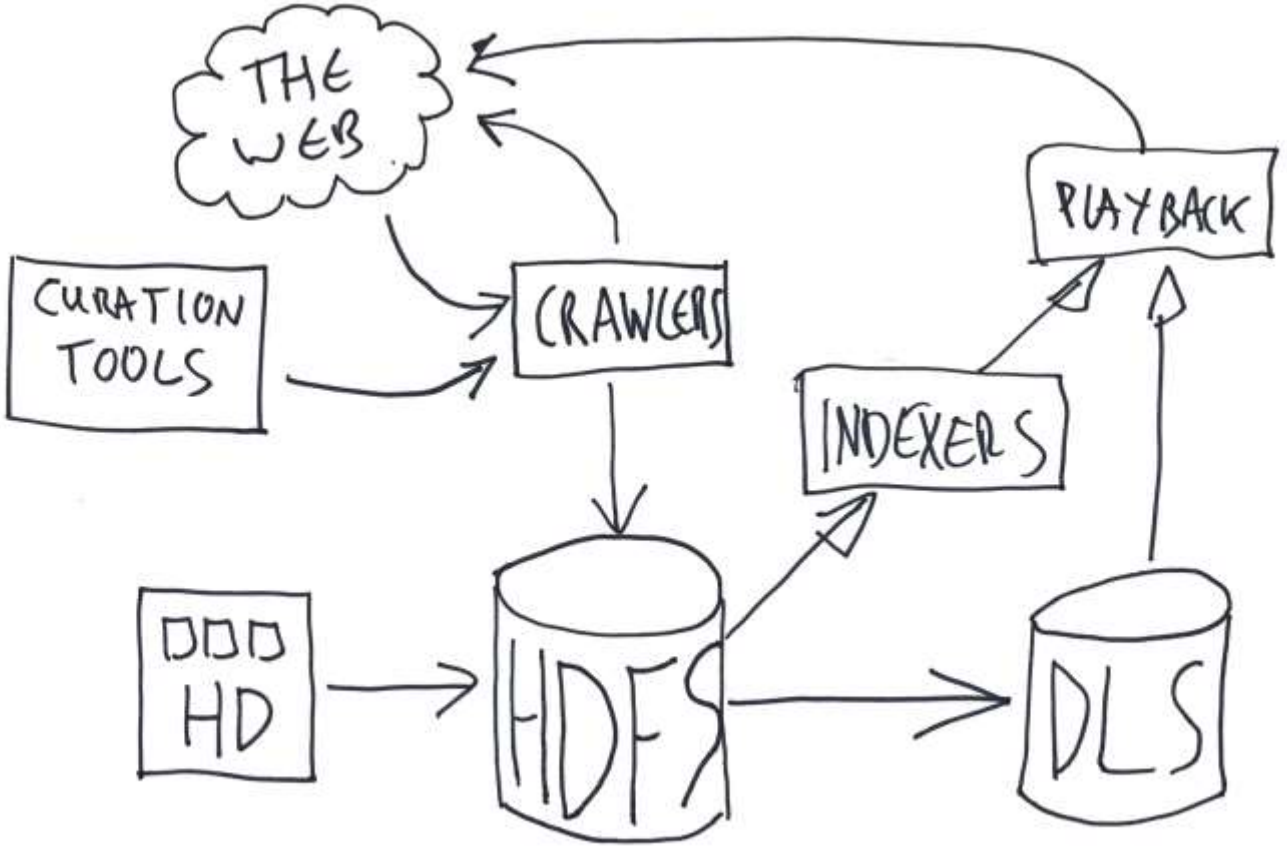
Introduction to Apache Solr

Andrew Jackson

UK Web Archive Technical Lead

Web Archive Overall Architecture



Understanding Your Use Case(s)

- Full text search, right?
  - Yes, but there are many variations and choices to make.
- Work with users to understand their information needs:
  - Are they looking for…
    - Particular (archived) web resources?
    - Resources on a particular issue or subject?
    - Evidence of trends over time?
  - What aspects of the content do they consider important?
  - What kind of outputs do they want?

Working With Historians…

- JISC AADDA Project:
  - Initial index and UI of the 1996-2010 data
  - Great learning experience and feedback
  - http://domaindarkarchive.blogspot.co.uk/
- AHRC ‘Big Data’ Project:
  - Second iteration of index and UI
  - Bursary holders reports coming soon
  - http://buddah.projects.history.ac.uk/
- Interested in trends and reflections of society
  - Who links to who/what, over time?

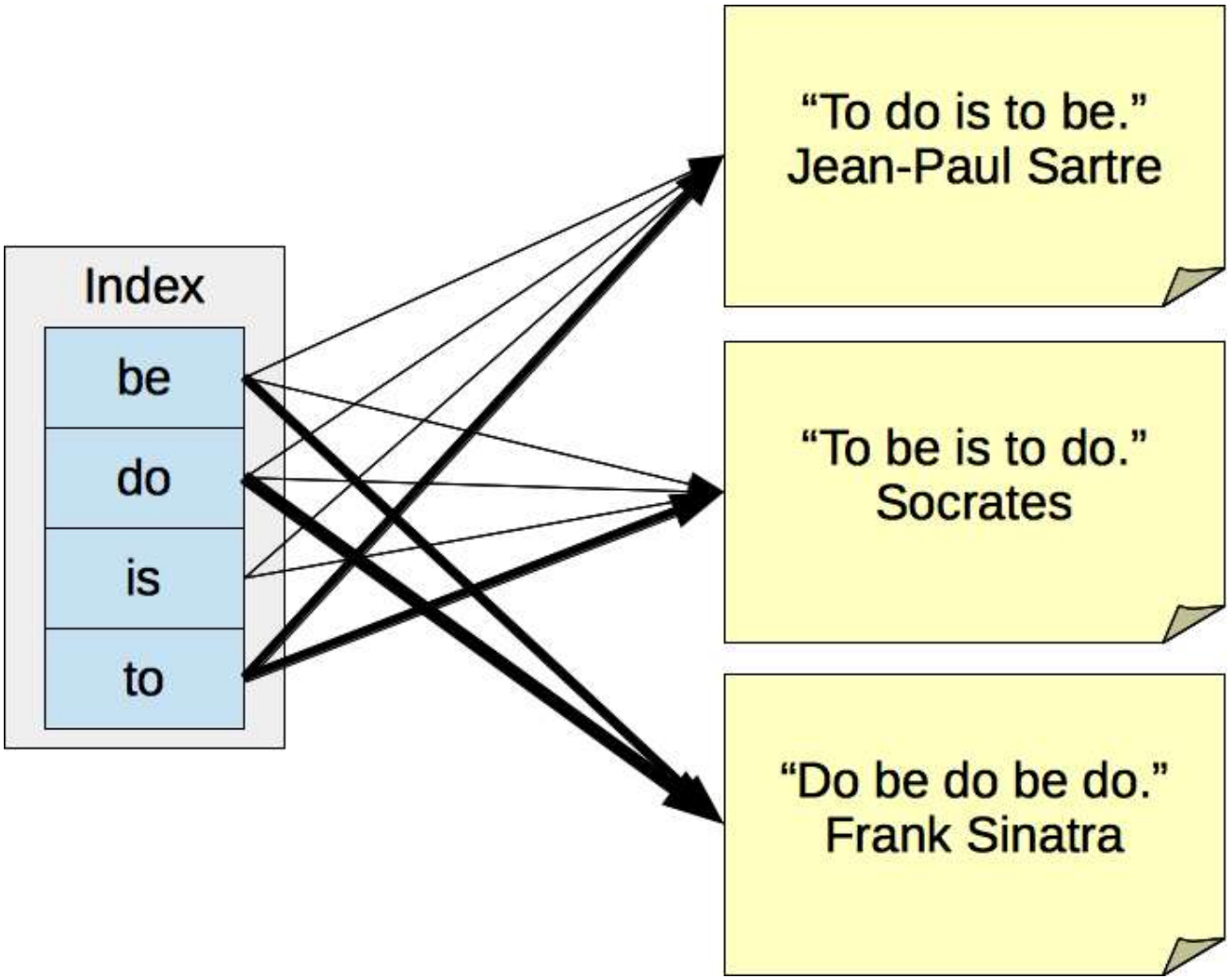
Apache Solr & Lucene

- Apache Lucene:
  - A Java library for full text indexes
- Apache Solr:
  - A web service and API that exposes Lucene functionality in a as a document database
  - Supports SolrCloud mode for distributed searches
- See also:
  - Elasticsearch (also built around Lucene)
  - We ‘chose’ Solr before Elasticsearch existed
  - http://solr-vs-elasticsearch.com/

Example: Indexing Quotes

- Quotes to be indexed:
  - “To do is to be.” - Jean-Paul Sartre
  - “To be is to do.” - Socrates
  - “Do be do be do.” - Frank Sinatra
- Goals:
  - Index the quotation for full-text search.
    - e.g. Show me all quotes that contain “to be”.
  - Index the author for faceted search.
    - e.g. Show me all quotes by “Frank Sinatra”.

Lucene's Inverted Indexes



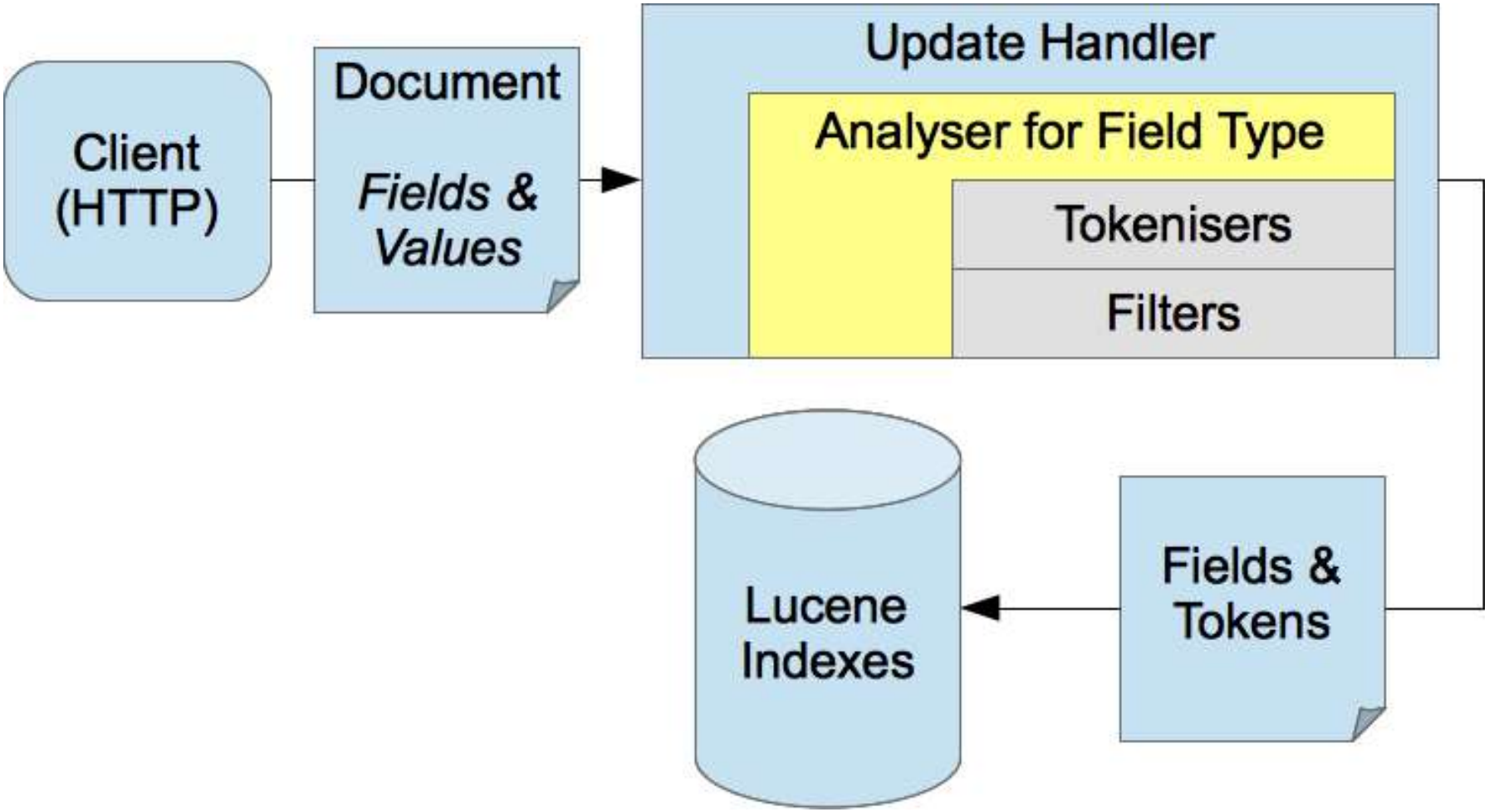
Solr as a Document Database

- Solr Indexes/Stores & Retrieves:
  - **Documents**
    composed of:
    - **Multiple Fields**
      each of which has a defined:
      - **Field Type**
        such as ‘text’, ‘string’, ‘int’, etc.
- The queries you can support depend on on many parameters, but the fields and their types are the most critical factors.
  - See Overview of Documents, Fields, and Schema Design

The Quotes As Solr Documents

- Our Documents contain three fields:
  - ‘id’ field of type ‘string’
  - ‘text’ field of type ‘text_general’
  - ‘author’ field, of type ‘string’
- Example Documents:
  - id: “1”, text: “To do is to be.”, author: “Jean-Paul Sartre”
  - id: “2”, text: “To be is to do.”, author: “Socrates”
  - id: “3”, text: “Do be do be do.”, author: “Frank Sinatra”

Solr Update Flow

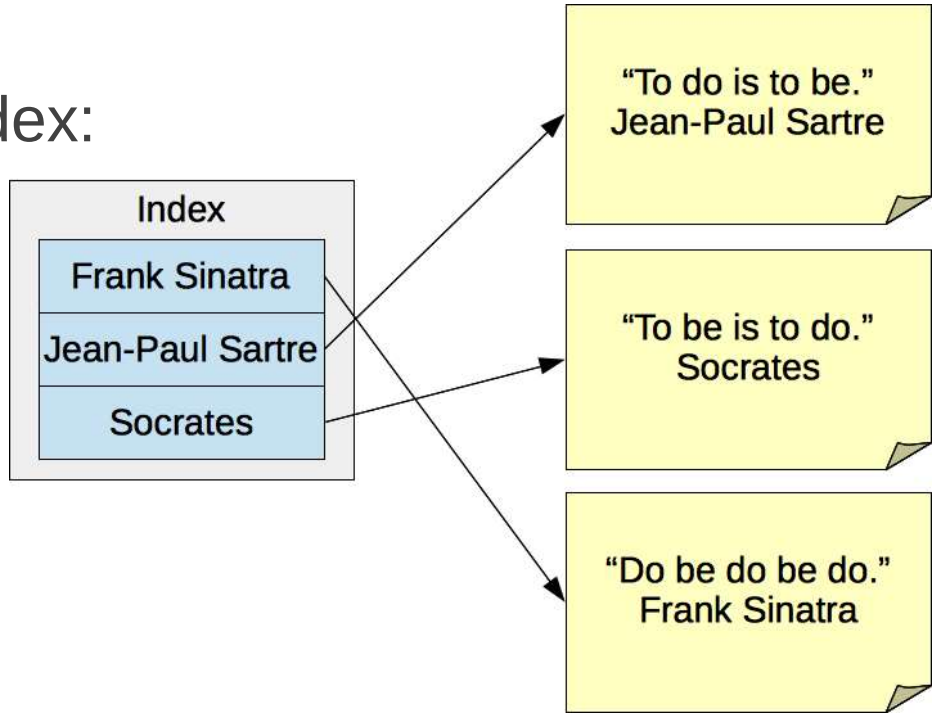


Analyzing The Text Field

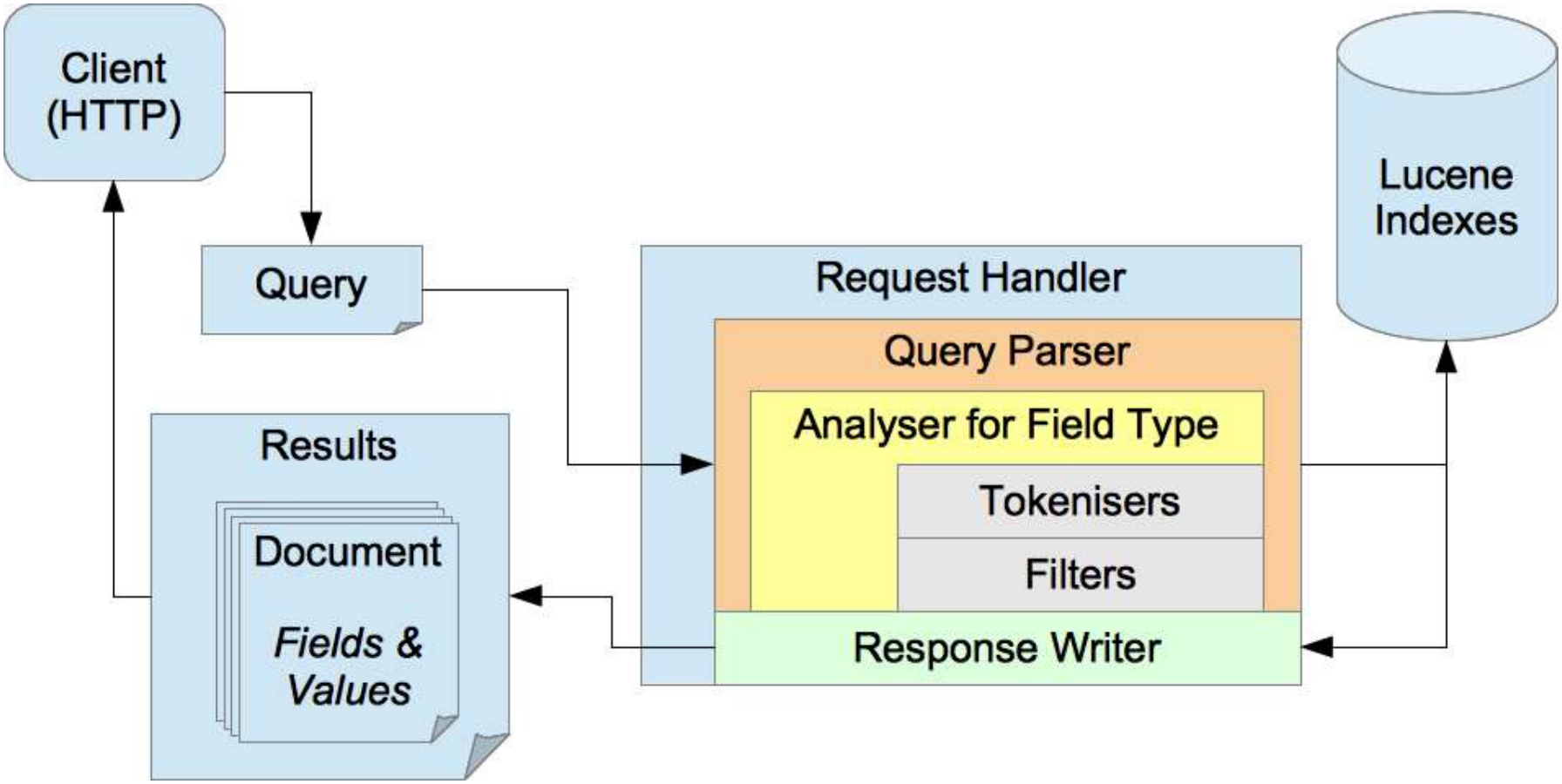
- Analyzing the text on document 1:
  - Input: “To do is to be.”, type = ‘text_general’
  - Standard Tokeniser:
    - ‘To’ ‘be’ ‘is’ ‘to’ ‘do’
  - Lower Case Filter:
    - ‘to’ ‘be’ ‘is’ ‘to’ ‘do’
- Adding the tokens to the index:
  - ‘be’ => id:1
  - ‘do’ => id:1
  - …

Analyzing The Author Field

- Analyzing the author on document 1:
  - Input: “Jean-Paul Sartre”, type = ‘string’
  - Strings are stored as is.
- Adding the tokens to the index:
  - ‘Jean-Paul Sartre’ => id:1

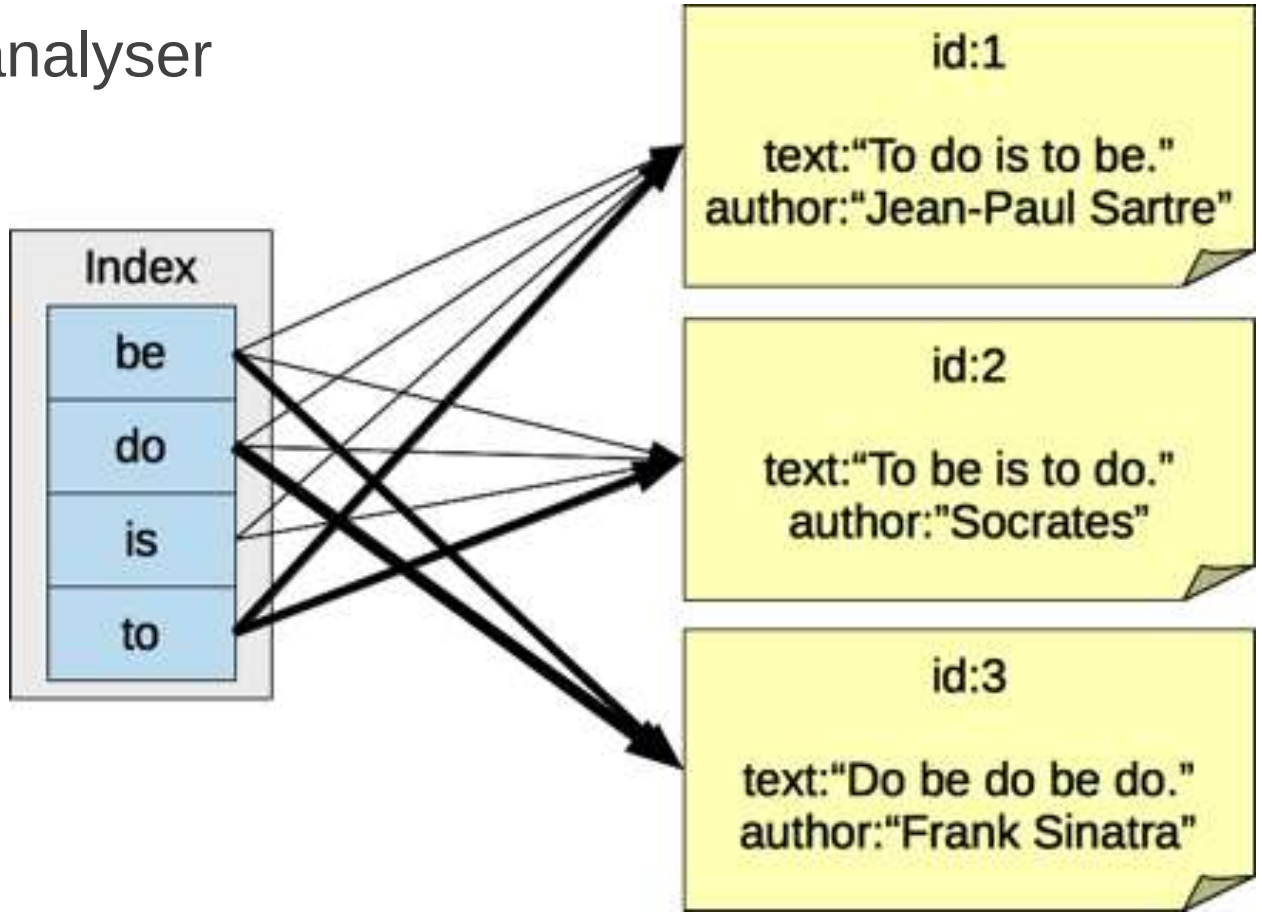


Solr Query Flow

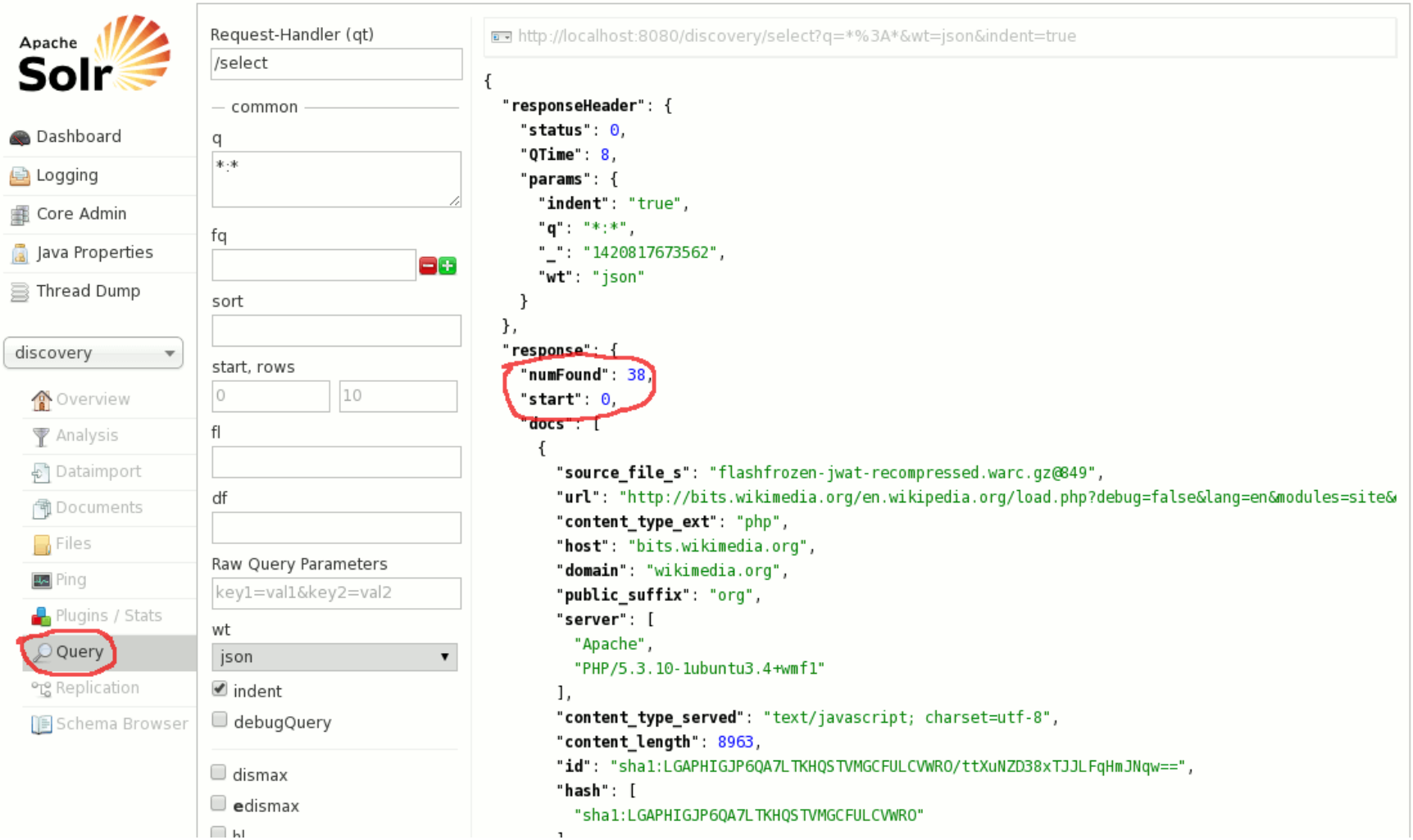


Query for text:“To be”

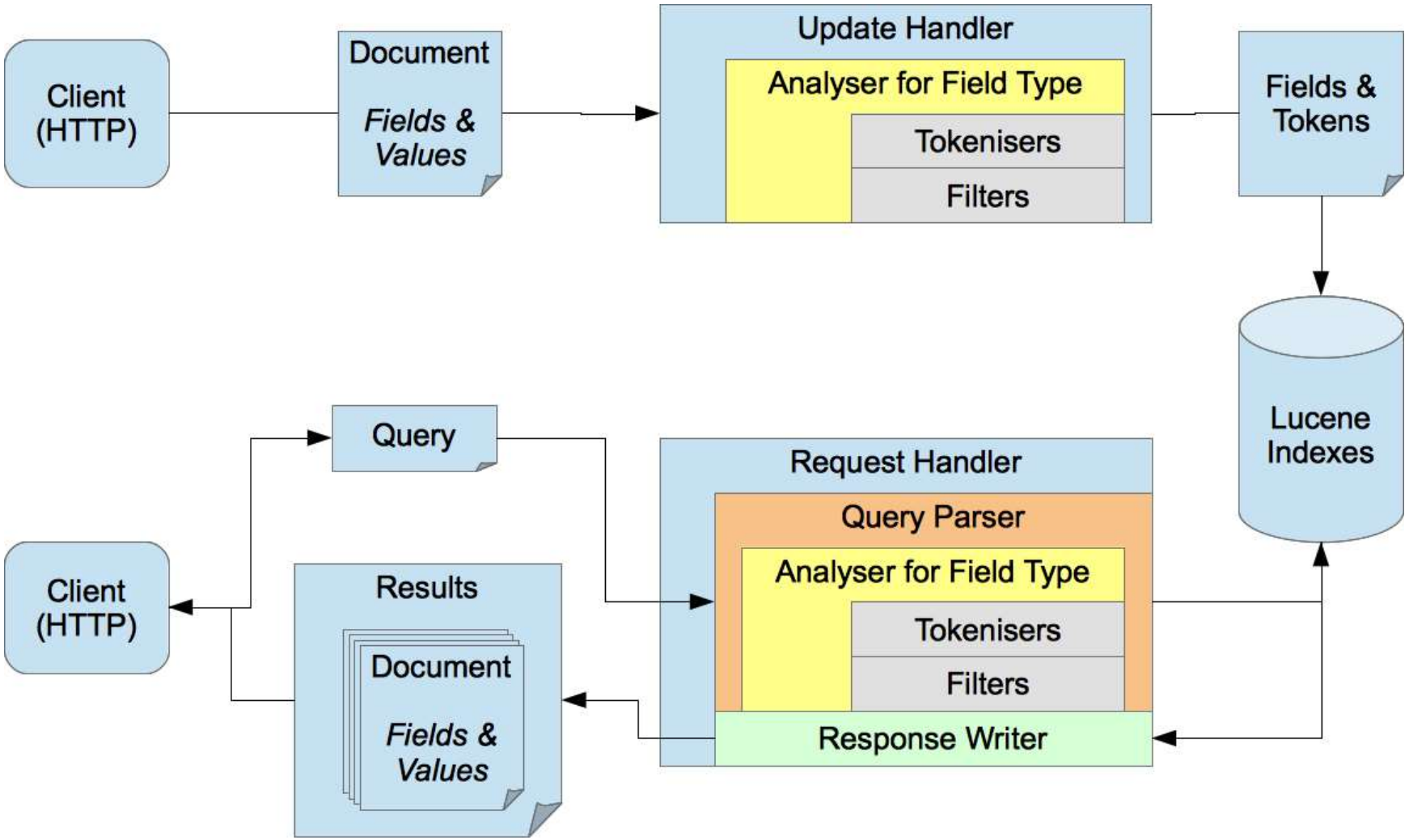
- Uses the same analyser as the indexer:
  - “To be?”
  - ST: “To” “be”
  - LCF: “to” “be”
- Returns documents:
  - 1
  - 2



Solr's Built-in UI



Solr Overall Flow



Choice: Ignore ‘stop words’?

- Removes common words, unrelated to subject/topic
  - Input: “To do is to be”
  - Standard Tokeniser:
    - ‘To’ ‘be’ ‘is’ ‘to’ ‘do’
  - Stop Words Filter (*stopwords_en.txt*):
    - ‘do’
  - Lower Case Filter:
    - ‘do’
- Cannot support phrase search
  - e.g. searching for “to be”

Choice: Stemming?

- Attempts to group concepts together:
  - "fishing", "fished", "fisher" => "fish"
  - "argue", "argued", "argues", "arguing", "argus" => "argu"
- Sometimes confused:
  - "axes" => "axe", or "axis"?
- Better at grouping related items together
- Makes precise phrase searching difficult

So Many Choices…

- Lots of text indexing options to tune:
  - Punctuation and tokenization:
    - is www.google.com one or three tokens?
  - Stop word filter (“the” => “”)
  - Lower case filter (“This” => “this”)
  - Stemming (choice of algorithms too)
  - Keywords (excepted from stemming)
  - Synonyms (“TV” => “Television”)
  - Possessive Filter (“Blair’s” => “Blair”)
  - …and many more Tokenizers and Filters.

Even More Choices: Query Features

- As well as full-text search variations, we have
  - Query parsers and features:
    - Proximity, wildcards, term frequencies, relevance…
  - Faceted search
  - Numeric or Date values and range queries
  - Geographic data and spatial search
  - Snippets/fragments and highlighting
  - Spell checking i.e. ‘Did you mean …?’
  - MoreLikeThis
  - Clustering

How to get started?

- Experimenting with the UKWA stack:
  - Indexing:
    - webarchive-discovery
  - User Interfaces:
    - Drupal Sarnia
    - Shine (Play Framework, by UKWA)
    - See https://github.com/ukwa/webarchive-discovery/wiki/Front-ends

The webarchive-discovery system

- The webarchive-discovery codebase is an indexing stack that reflects our (UKWA) use cases
  - Contains our choices, reflects our progress so far
  - Turns ARC or WARC records into Solr Documents
  - Highly robust against (W)ARC data quality problems
- Adds custom fields for web archiving
  - Text extracted using Apache Tika
  - Various other analysis features
- Workshop sessions will use our setup
  - but this is only a starting point…

Features: Basic Metadata Fields

- From the file system:
  - The source (W)ARC filename and offset
- From the WARC record:
  - URL, host, domain, public suffix
  - Crawl date(s)
- From the HTTP headers:
  - Content length
  - Content type (as served)
  - Server software IDs

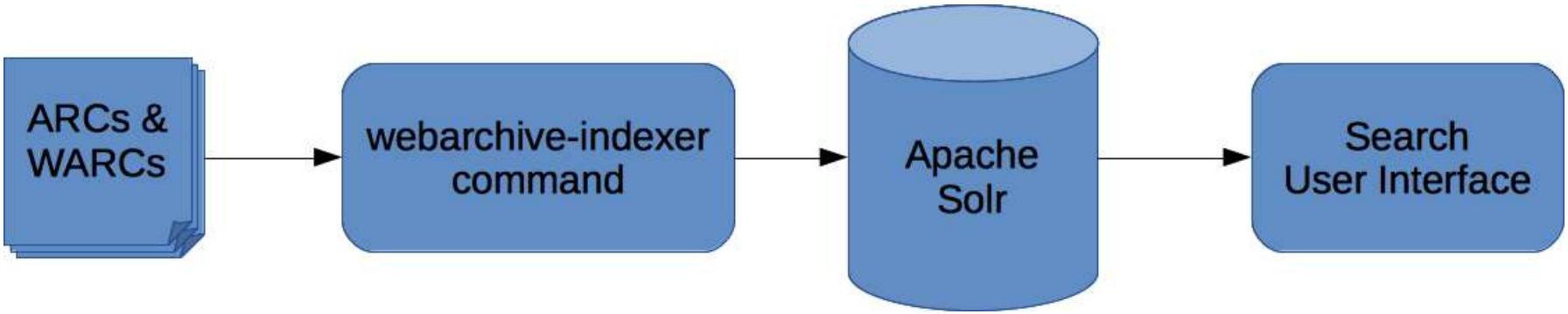
Features: Payload Analysis

- Binary hash, embedded metadata
- Format and preservation risk analysis:
  - Apache Tika & DROID format and encoding ID
  - Notes parse errors to spot access problems
  - Apache Preflight PDF risk analysis
  - XML root namespace
  - Format signature generation tricks
- HTML links, elements used, licence/rights URL
- Image properties, dominant colours, face detection

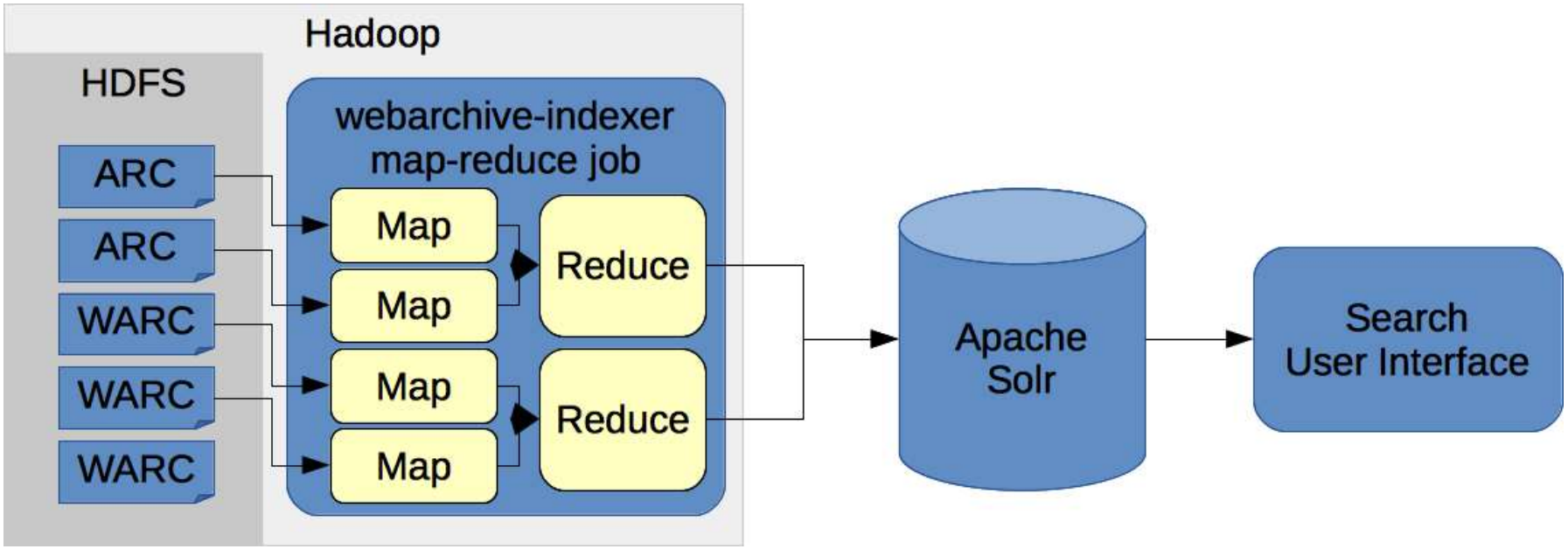
Features: Text Analysis

- Text extraction from binary formats
- ‘Fuzzy’ hash (ssdeep) of text
  - for similarity analysis
- Natural language detection
- UK postcode extraction and geo-indexing
- Experimental language analysis:
  - Simplistic sentiment analysis
  - Stanford NLP named entity extraction
  - Initial GATE NLP analyser

Command-line Indexing Architecture



Hadoop Indexing Architecture



Scaling Solr

- We are operating outside Solr's sweet spot:
  - General recommendation is RAM = Index Size
  - We have a 15TB index. That's a lot of RAM.
- e.g. from this email
  - "100 million documents [and 16-32GB] per node"
  - "it's quite the fool's errand for average developers to try to replicate the "heroic efforts" of the few."
- So how to scale up?

Basic Index Performance Scaling

- One Query:
  - Single-threaded binary search
  - Seek-and-read speed is critical, not CPU
- Add RAID/SAN?
  - More IOPS can support more concurrent queries
  - BUT each query is no faster
- Want faster queries?
  - Use SSD, and/or
  - More RAM to cache more disk, and/or
  - Split the data into more shards (on independent media)

Sharding & SolrCloud

- For > ~100 million documents, use shards
  - More, smaller independent shards == faster search
- Shard generation:
  - SolrCloud ‘Live’ shards
    - We use Solr’s standard sharding
    - Randomly distributes records
    - Supports updates to records
  - Manual sharding
    - e.g. ‘static’ shards generated from files
    - As used by the Danish web archive (see later today)

Next Steps

- Prototype, Prototype, Prototype
  - Expect to re-index
  - Expect to iterate your front and back end systems
  - Seek real user feedback
- Benchmark, Benchmark, Benchmark
  - More on scaling issues and benchmarking this afternoon
- Work Together
  - Share use cases, indexing tactics
  - Share system specs, benchmarks
  - Share code where appropriate