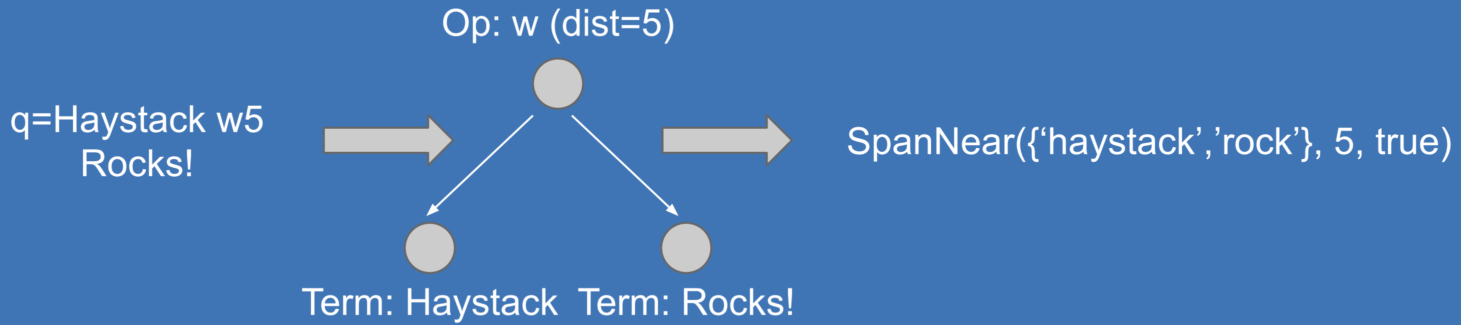
Custom Solr Query Parser Design Options, Pros & Cons

Haystack Training & Conference
April 22nd – 25th, 2019 • Charlottesville, VA, USA

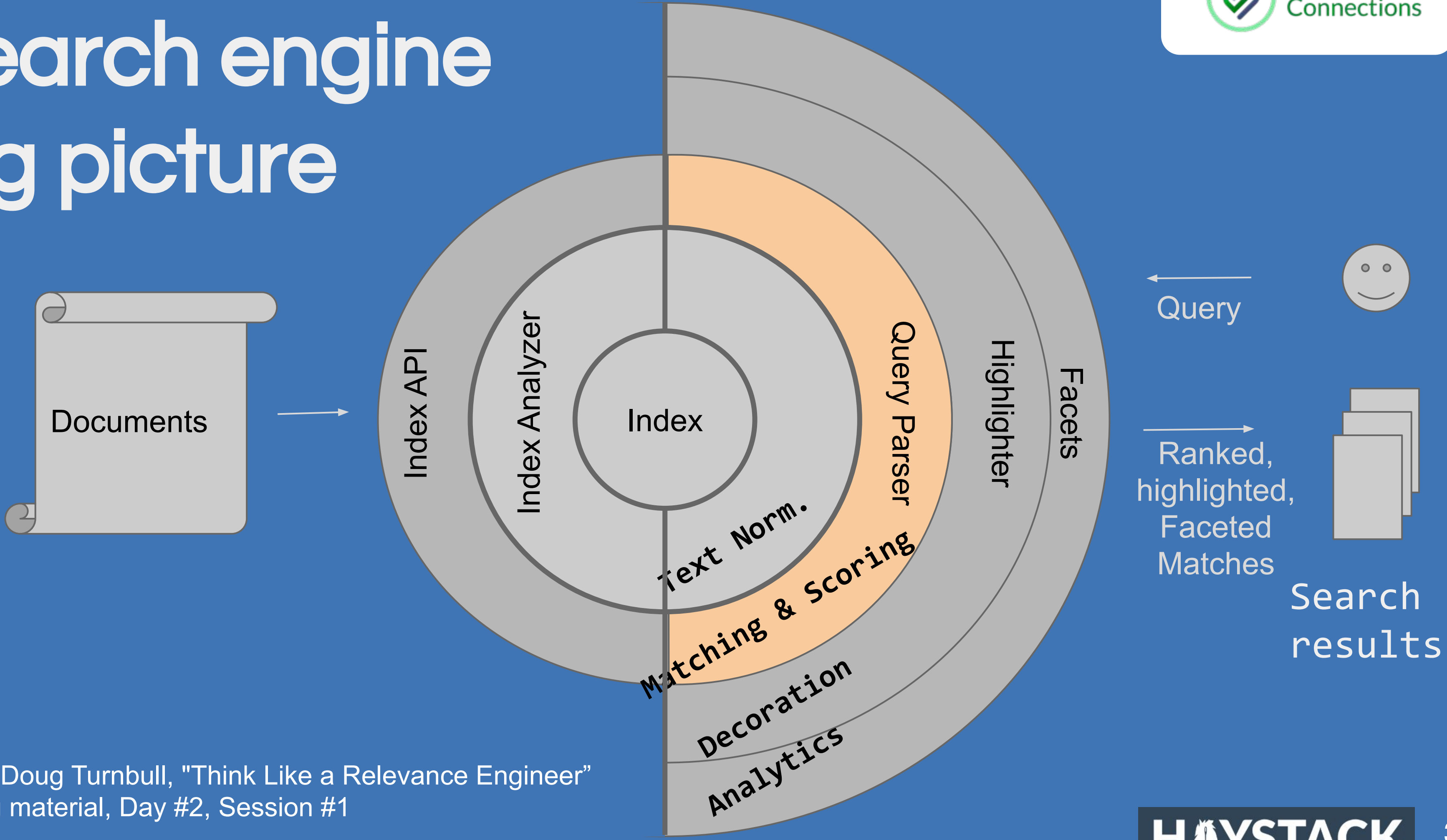


Bertrand Rigaldies
Search Consultant
OpenSource Connections
brigaldies@o19s.com
Linkedin: bertrandrigaldies

Agenda

- Query parsers' purpose
- Query parser composition in Solr
- When do you need a custom query parser?
- How to build a custom query parser?
- Pros and cons of various design approaches
- Beyond query parsers

Search engine big picture



Credit: Doug Turnbull, "Think Like a Relevance Engineer" training material, Day #2, Session #1

What’s The Problem Here?

1. [Expression → Search Executable] compilation
2. Query Understanding
3. How do your users search?
   - “Natural” language, as we increasingly do everyday
   - Or, a more formal search language:
     - With operators like boolean and proximity
     - Advanced custom query syntax
   - Or, some kind of hybrid of the above

End-Users Spectrum

Casual, Occasional ⟷ Seasoned ⟷ Professional Librarian

What’s The Problem? (again)

Is it the **FIRST relevancy issue** in a search application project: How do we translate the end-user’s high-level search expression into an executable that will most effectively approximate what the end-user is looking for?

What Can We Do Out-of-the-box?

- A lot! Solr (ES too) offers powerful query parsers out of the box:
  - “Classic” Lucene:
    - df=title, q=I love search
      ```
      → title:i title:love title:search
      ```
  - “Swiss Army Knife” edismax:
    - qf=title body, q=**I love search**
      ```
      → +( (title:i      | body:i)
           (title:love   | body:love)
           (title:search | body:search)
         )
      ```

How far can I go?

Search for the capitalized term “*Green*”, but not the adjective “green”, that is 5 positions or less <u>before</u> the noun “*deal*”.

```
{!lucene} “green deal”~5

{!surround} green 5w deal

{!surround} 5w(2w(green,deal), congress OR
legislation)

_query_:”{!cap}firstcap(green)” AND
_query_:”{!proximity}green 5w deal”
```

Query Parsers Composition

- Solr provides a large variety of QPs (28 and counting, JSON Query DSL), that are composable:

```
_query_:"{!lucene}\"green deal\""
AND
_query_:"{!surround} 5n(congress,
democrat)"
```

Query QPs Composition (Cont’d)

Solr XML QP:

```
<BooleanQuery fieldName="title_txt">
  <Clause occurs="must">
    <SpanNear slop="0" inOrder="true">
      <SpanTerm>green</SpanTerm>
      <SpanTerm>deal</SpanTerm>
    </SpanNear>
  </Clause>
  <Clause occurs="must">
    <SpanNear slop="5" inOrder="false">
      <SpanTerm>congress</SpanTerm>
      <SpanTerm>democrat</SpanTerm>
    </SpanNear>
  </Clause>
</BooleanQuery>
```

Query QPs Composition (Cont’d)

Solr JSON QP:

```
{
  "query": {
    "bool": {
      "must": [
        {"lucene": {"df": "title_t", "query": "\"green
deal\""}},
        {"surround": {"df": "title_t", "query": "5n(congress,
democrat)"}}
      ]
}}}
```

What If We Need To Go Beyond?

- There are limitations and quirks, e.g., the Solr “Surround” QP:
  - Distance <= 99;
  - Search terms are not analyzed! What?
- What about operators that do not exit?
  - Capitalization: Match `Green`, but not `green`
  - Frequency: Must match N times or less
  - As-is: Search for a term as written.
- What do we do now? Enter the world of custom query parsers!

Demo: Let's build a simple proximity query parser!

… CVille Haystack w5 Rocks 2019 …

- **Analyze terms**
- **Distance >= 0, no upper limit**
- **Operator**: Same as surround (w<dist>, n<dist>)
- https://github.com/o19s/solr-query-parser-demo

Query Parser Plugin Anatomy

ProximityQParserPlugin.java:

QP “Factory” Class

```
public class ProximityQParserPlugin extends QParserPlugin {
    public QParser createParser(String s, SolrParams localParams, SolrParams
globalParams, SolrQueryRequest solrQueryRequest) {
        return new ProximityQParser(s, localParams, globalParams,
solrQueryRequest);
    }
}
```

In solrconfig.xml:

Solr Config

```
<queryParser name="proximity"
class="com.o19s.solr.qparser. ProximityQParserPlugin"/>
<requestHandler name="/proximity" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="defType">proximity</str>
    ...
```

Custom QP & Request Handler

Solr Config (cont'd)

```
<requestHandler name="/proximity" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="defType">proximity</str>
    <str name="qf">title_txt</str>
    <str name="fl">id, title_txt, pub_dt, popularity_i, $luceneScore, $dateBoost, $popularityBoost, $myscore</str>

    <str name="dateBoost">recip(ms(NOW,pub_dt),3.16e-11,1,1)</str>
    <str name="popularityBoost">sum(1,log(sum(1, popularity_i)))</str>
    <str name="mainQuery">{!proximity v=$q}</str>
    <str name="luceneScore">query($mainQuery)</str>
    <str name="myscore">product(product($luceneScore, $dateBoost), $popularityBoost)</str>
    <str name="order">$myscore desc, pub_dt desc, title_s desc</str>

    <str name="hl">true</str>
    <str name="hl.method">unified</str>
    <str name="hl.fl">title_txt</str>

    <str name="facet">true</str>
    <str name="facet.mincount">1</str>
    <str name="facet.field">popularity_i</str>
    <str name="facet.range">pub_dt</str>
    <str name="f.pub_dt.facet.range.start">NOW/DAY-30DAYS</str>
    <str name="f.pub_dt.facet.range.end">NOW/DAY+1DAYS</str>
    <str name="f.pub_dt.facet.range.gap">+1DAY</str>
  </lst>
</requestHandler>
```

QP

Boosting and custom scoring

Highlighting

Faceting

Query Parser Plugin Anatomy

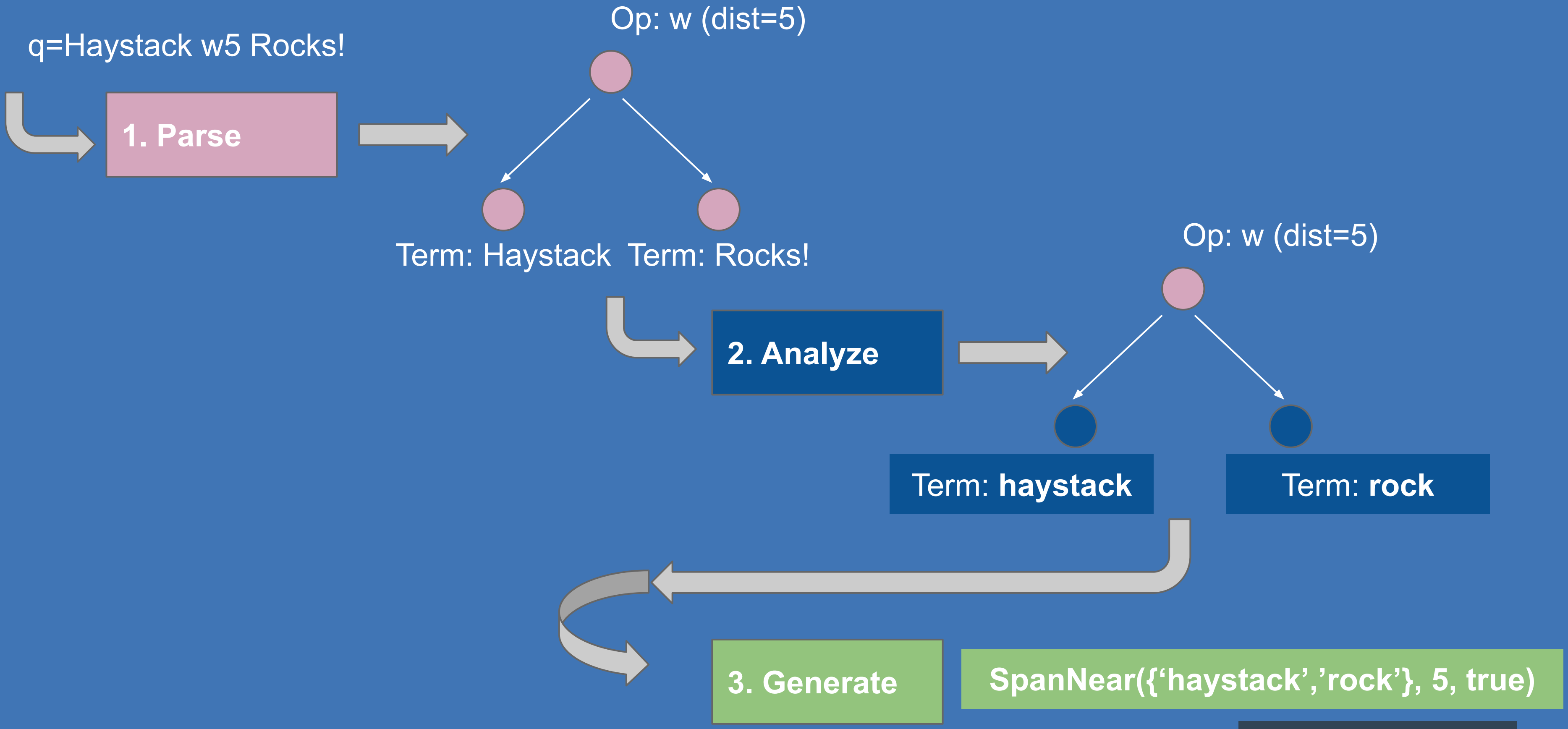
ProximityQParser.java:

```
public class ProximityQParser extends QParser {
    public ProximityQParser(String qstr, SolrParams localParams,
SolrParams params, SolrQueryRequest req) {
        super(qstr, localParams, params, req);
    }

    public Query parse() throws SyntaxError {
        // Parse and build the Lucene query
        Query query = parseAndComposeQuery(qstr);
        return query;
    }
}
```

Parse end-user's search string;
Generate the Lucene query,
and return it to Solr.

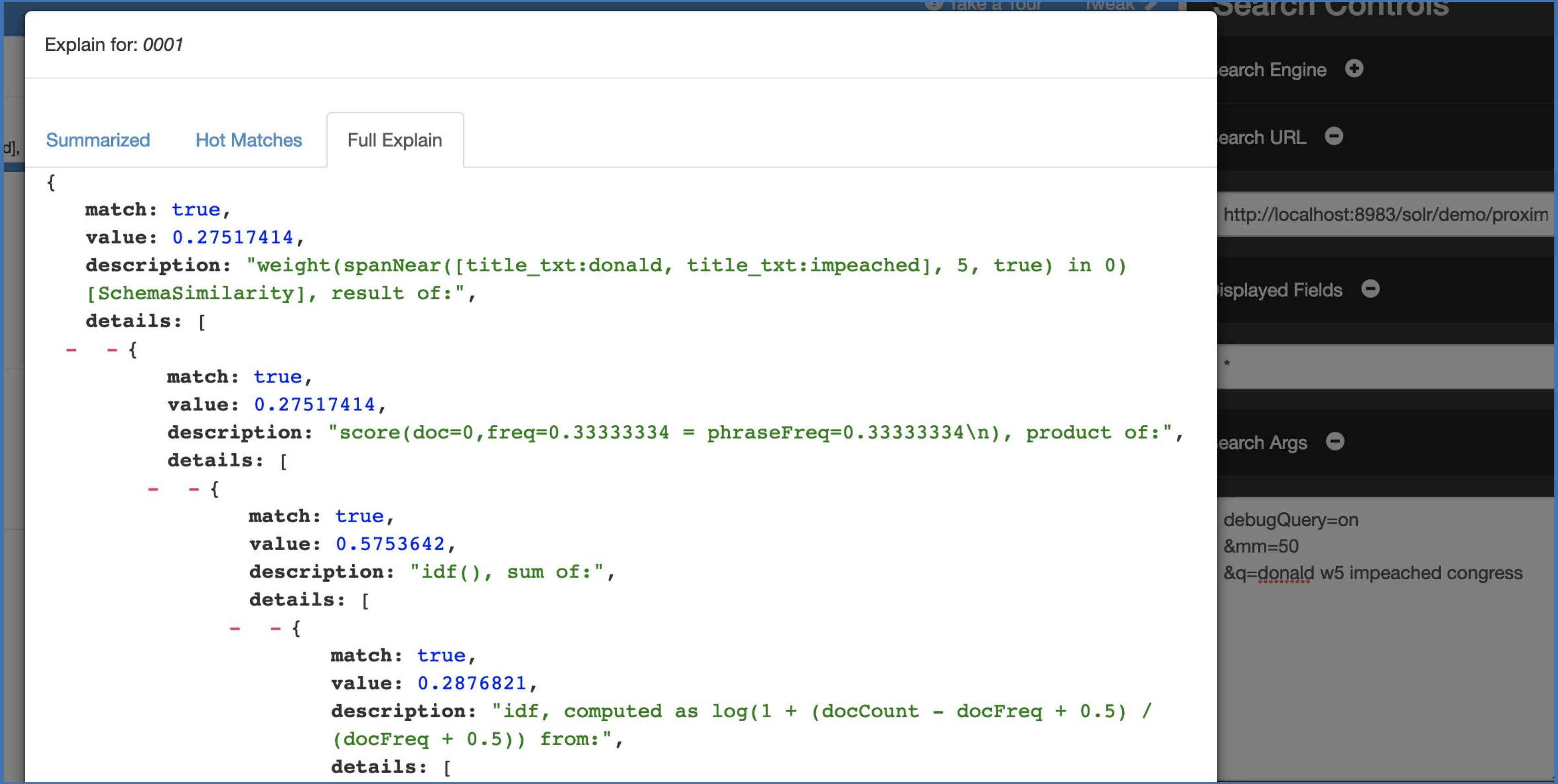
Query Parser's "Parse Flow"



Demo

1. Overview of the Java code
2. Run unit tests
3. Deploy the plugin jar
4. Run test queries
5. Examine scoring

Score



Explain for: *0001*

Summarized | Hot Matches | Full Explain

```
{
   match: true,
   value: 0.27517414,
   description: "weight(spanNear([title_txt:donald, title_txt:impeached], 5, true) in 0)
   [SchemaSimilarity], result of:",
   details: [
 -   - {
          match: true,
          value: 0.27517414,
          description: "score(doc=0,freq=0.33333334 = phraseFreq=0.33333334\n), product of:",
          details: [
       -   - {
                match: true,
                value: 0.5753642,
                description: "idf(), sum of:",
                details: [
             -   - {
                      match: true,
                      value: 0.2876821,
                      description: "idf, computed as log(1 + (docCount - docFreq + 0.5) /
                      (docFreq + 0.5)) from:",
                      details: [
```

earch Engine

earch URL

http://localhost:8983/solr/demo/proxim

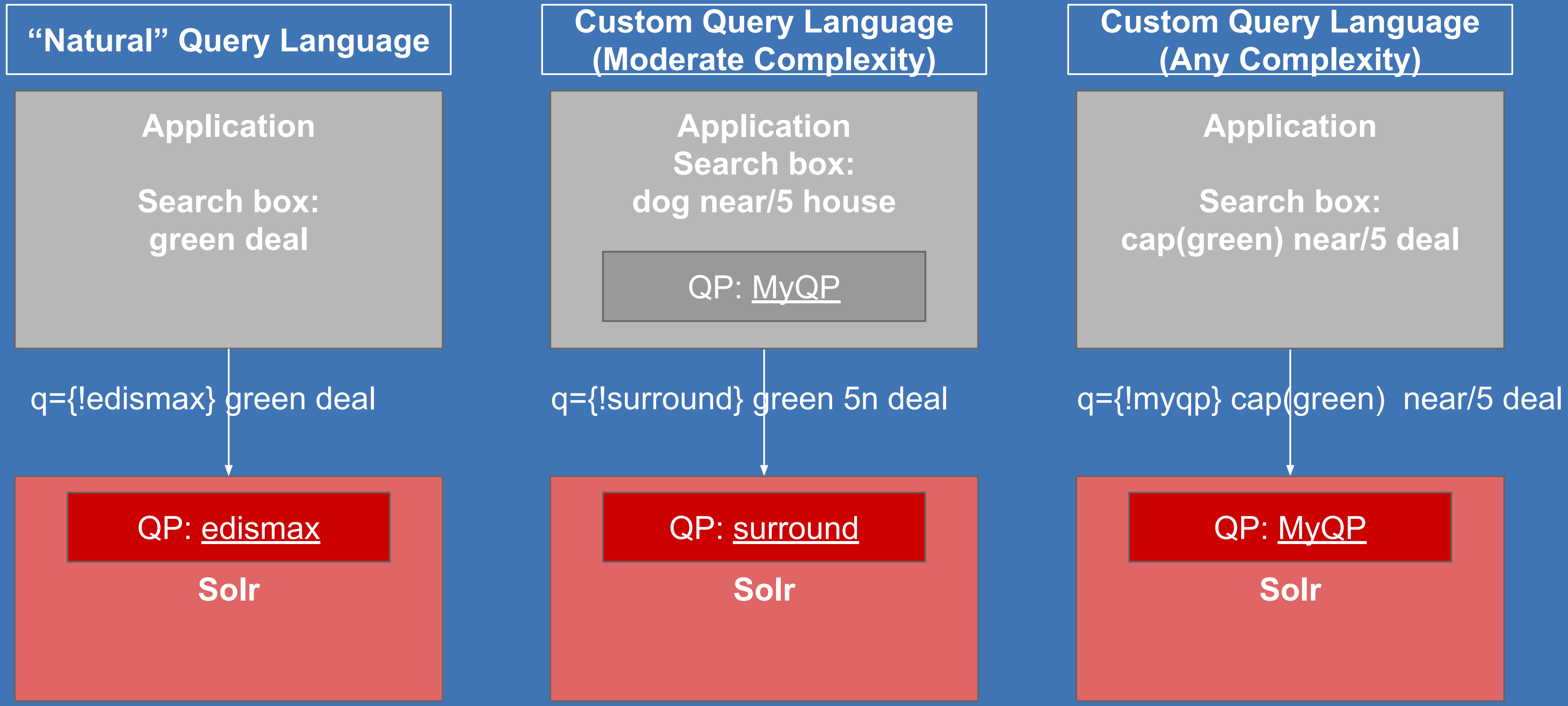
isplayed Fields

*

earch Args

debugQuery=on
&mm=50
&q=donald w5 impeached congress

Query Parser Strategies

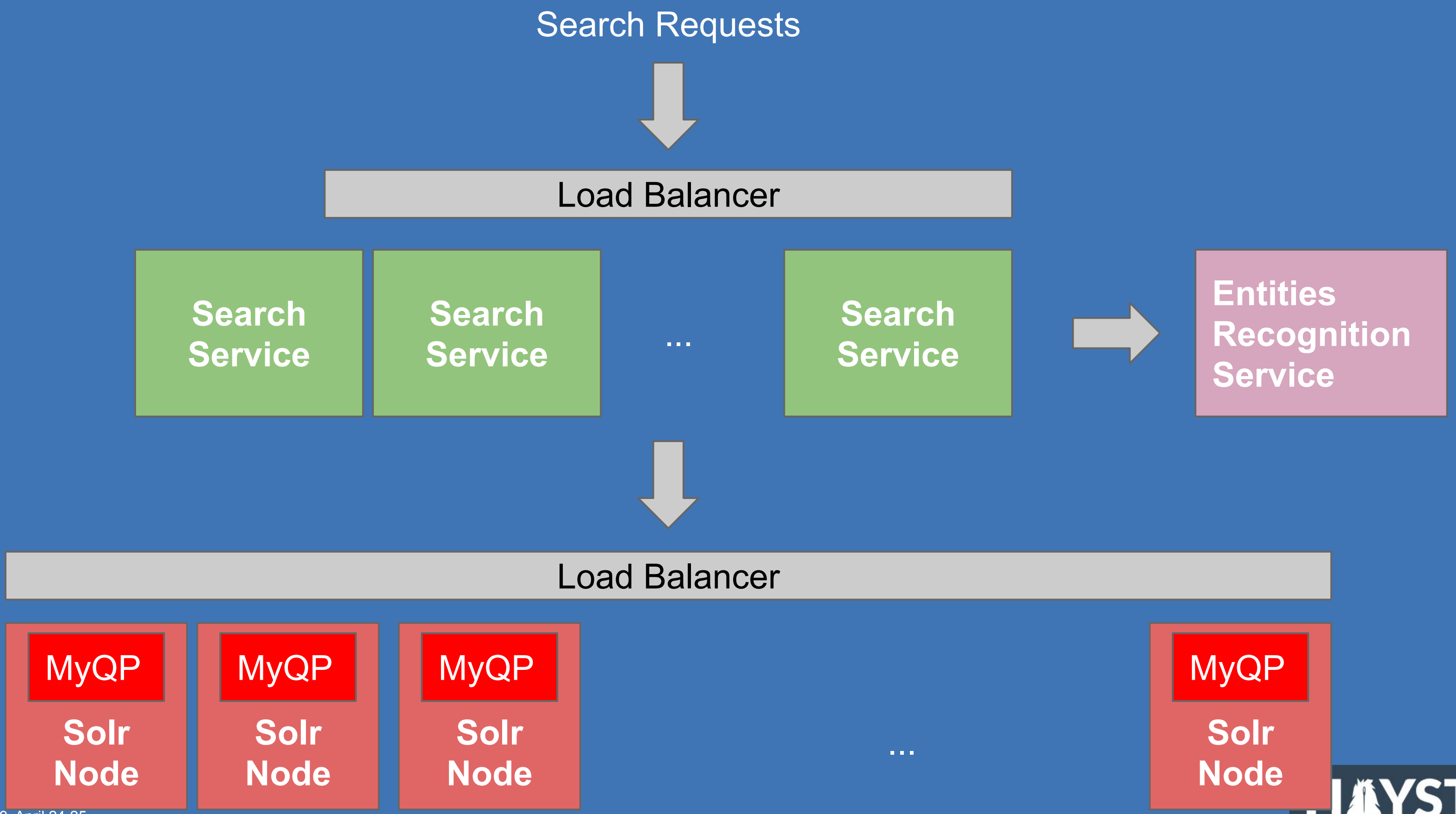


Query Parser Strategies Comparison

Criteria	edismax	Solr QPs Composition	Custom QP
Software R&D	No	Moderate	High
Ease of Solr upgrade	Very Good	Good	To be managed
Performance	Good	Good	Better vs. Solr QPs composition But be careful!
Deployment	-	-	Plugin jar(s)
Ease of Relevancy Tuning	The good ol' edismax	Individual QPs' knobs and dials	More software to write!

HAYSTACK

Entities Recognition vs. Query Parsing



Closing Remarks

- QPs are a lot of fun, BUT:
  - Make sure you *really* need to go beyond the out-the-box features!
  - Great power comes with great responsibility. Careful what you write!
  - Relevancy knobs and dials can be tricky to re-implement: Multi-field, term- vs. fields-centric, mm, field boosting, etc.
- The next frontier: Custom Lucene queries
  - Multi-terms synonyms w/ equalized scoring
  - Frequency operators