



SearchWorkings

Lucene today, tomorrow and beyond

Simon Willnauer

Apache Lucene Core Committer & PMC Chair

simonw@apache.org / simon.willnauer@searchworkings.org

Who am I?



- Lucene Core Committer
- Project Management Committee Chair (PMC)
- Apache Member
- BerlinBuzzwords Co-Founder
- Addicted to OpenSource
- Apache Solr & Lucene User / Consultant / Promoter

- Community Portal targeting OpenSource Search



[SIGN UP](#) [Login](#)

- HOME
- PROJECTS
- RESOURCES
- FORUM
- BLOG

Welcome, we can see you are a newbie – let us show you around...
SearchWorkings.org is a community of search professionals looking for
a resource where they can discover, share and discuss the latest
technologies and topics.

[▶ Sign up!](#)

➔ [Home](#)

Featured Topics

Free Online Training

[Integrating Solr with JEE applications](#)



So you have downloaded Solr, configured it, indexed your data and are now ready to integrate it with the rest of your enterprise Java

application. For most situations, this process will begin with...

Featured Blog Entry

[The ManifoldCF authorization model](#)

Getting documents out of a repository and into Solr is only half of the problem, because it is a rare repository that does not attempt to restrict access to individual documents based on a user's...

[View in Context »](#)



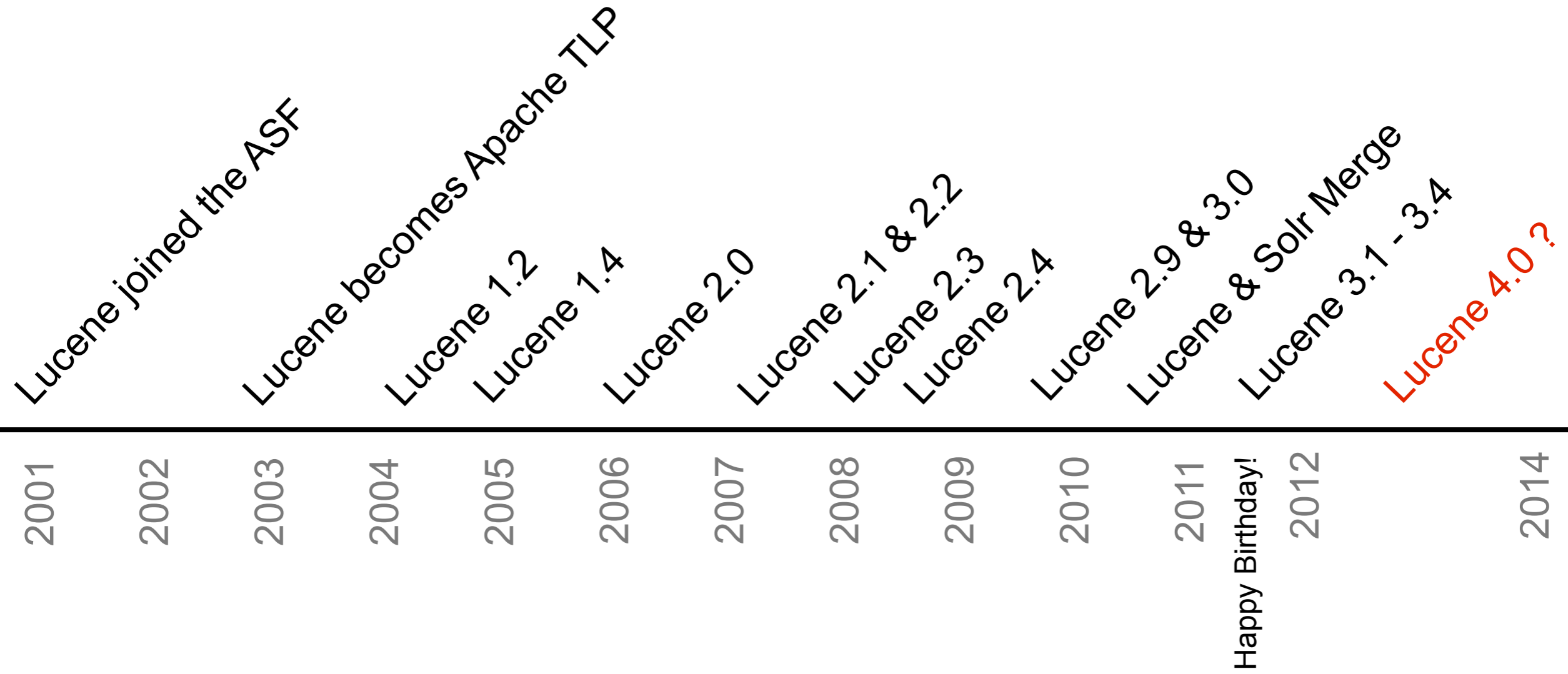
It's time for Apache Lucene EUROCON in Barcelona. A conference aimed at the European Apache Lucene / Solr open source search community. Two key contributors from SearchWorkings.org have been asked to participate and will be speakers at the event.

What makes this talk different?

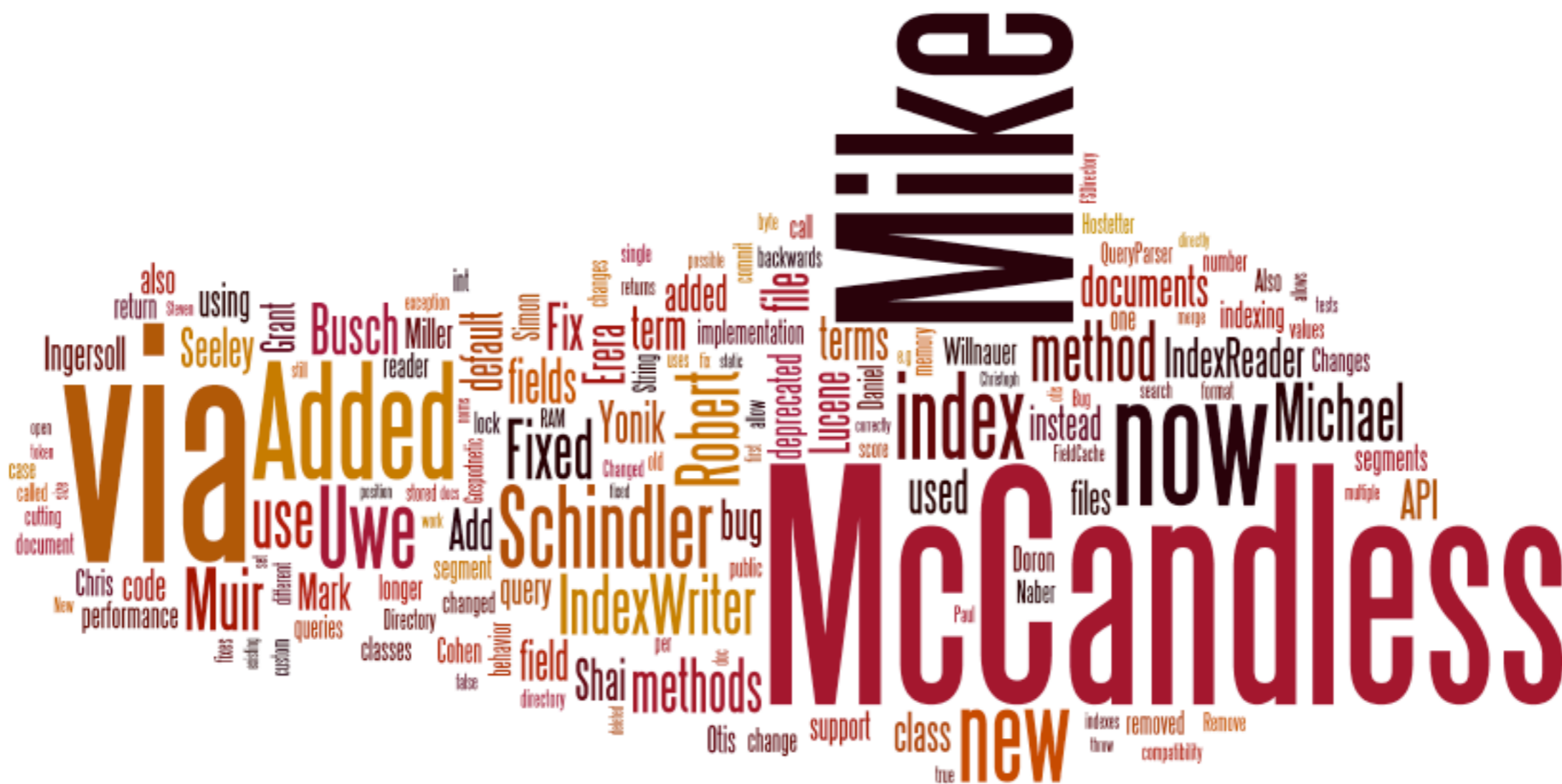


- The most of the talks here are presenting what Lucene **can** do or what people do with Lucene, right?
- This talk will show what Lucene **can't** do today (**trunk**) but might be doing in the future.
- I **won't** talk about what people going to do in the future - maybe next time :)

Let's go back in time a bit



And who did all the work?



Created from Lucene core CHANGES.TXT

Especially “via” is interesting since we use this for contributions from non-committers (FooBar via \$committer_name)

Lets make this a fair game!





Where are we now - once 4.0 is out?

- Lucene 4.0 contains a ton of smallish improvements
- Lots of refined APIs
- Large speed improvements
- New modules
- **And lots of paths to explore for the future!**

Some random improvements

- FuzzyQuery speedup by **20000%** (yes 20k!)
- Indexing throughput improvements **200% to 280%**
- Document Filtering speedup up to **480%**
- Loading term dictionaries up to **30x** faster using **10%** of the memory compared to 3.x
- **600000** key-value lookups/second
- Tremendous reduction of GC needs at runtime

Your mileage may vary!

Flexible Indexing & Codecs



- Allows to customize low level index structure per field
- Yields significant performance gains depending on the use-case
- Highly optimized data-structures
- Allows future improvements due to per codec Backwards Compatibility
- Lets you decide on memory consumption

- Value per field & document - similar to FieldCache
- Type-safe and efficient on-disk & in-memory access
- Soon update-able
- More flexible than FieldCache
- Fast loading times

Flexible Scoring



- New ranking models in addition to VSM
- Adds key statistics to Lucene index to support other scoring models
- Decoupled matching from ranking
- Powerful Similarity API (can use IndexDocValues)

What else?



- DocumentWriterPerThread
 - High throughput incremental indexing
 - Preparation for RT-Search
- AutomatonQuery (FuzzyQuery)
 - Query as s Deterministic Finite Automata (DFA)
 - Levenshtein Automata for fast Fuzzy Queries (up to 20000% improvement over 3.x)
 - Flexible Automata concatenation

This was what we get with Lucene 4.0 (roughly)



- What is missing in this picture?
- Where are we going?
- What comes after 4.0?
- What is not going to make it into 4.0?

All this boils down to: “What do **WE & YOU** want Lucene to become in the future?”



CORE SEARCH FEATURES! - LIMITATIONS?

Positions - not a first class citizen

- We have:
 - Spans (Near, First, MultiTerm...)
 - PhraseQuery (sloppy & strict)
- The Problem:
 - Either use “common” query hierarchy or Spans
 - Score **ALL** or **NOTHING**
 - Scoring lots of documents takes ages

Positions - not a first class citizen

- Solutions?
 - Multi-Phase searches
 - Collect documents without positions
 - Re-score top N based on position data
 - Query hierarchy can be complex
 - We need an API with the same granularity as Scorer
 - Span semantics should not be bound to a query
 - Divorce scoring & matching for positions

Positions - not a first class citizen

- What about highlighting?
 - The implementation is a mess
 - Tons of `If (query instanceof FooQuery)`
 - Hard to extend for custom queries
- First steps are already taken!
 - <http://svn.apache.org/repos/asf/lucene/dev/branches/positions/>
 - Scorer allows to pull positions for any query - Help Wanted!

Updates - Huh? Incremental you know!

- Everybody wants it, right?
 - Updating a field without reindexing the entire doc? Yeah!
 - Watch out, this comes not for free!
- You can't simply update a field - it's a reverse index!
 - Term -> [(docID, freq)] (how to update this)
 - Lucene is write once - no in-place updates (which is good!)
- We have write per field per segment deltas and merge them on IndexReader open?! - seems tricky?
- Lots of paths need to be explored - maybe "appending fields"?

Updates - Huh? Incremental you know!

| term | fre | Posting list |
|--------|-----|--------------|
| and | 1 | 6 |
| big | 2 | 2 3 |
| dark | 1 | 6 |
| did | 1 | 4 |
| gown | 1 | 2 |
| had | 1 | 3 |
| house | 2 | 2 3 |
| in | 5 | 1 2 3 5 6 |
| keep | 3 | 1 3 5 |
| keeper | 3 | 1 4 5 |
| keeps | 3 | 1 5 6 |
| light | 1 | 6 |
| never | 1 | 4 |
| night | 3 | 1 4 5 |
| old | 4 | 1 2 3 4 |
| sleep | 1 | 4 |
| sleeps | 1 | 6 |
| the | 6 | 1 2 3 4 5 6 |
| town | 2 | 1 3 |
| where | 1 | 4 |

| | |
|---|--|
| 1 | The old night keeper keeps the keep in the town |
| 2 | In the big old house in the big old gown. |
| 3 | The house in the town had the big old keep |
| 4 | Where the old night keeper never did sleep. |
| 5 | The night keeper keeps the keep in the night |
| 6 | And keeps in the dark and sleeps in the light. |

update freq & postings

| | |
|---|--|
| 2 | In the small old house in the big old gown. |
|---|--|

insert new term

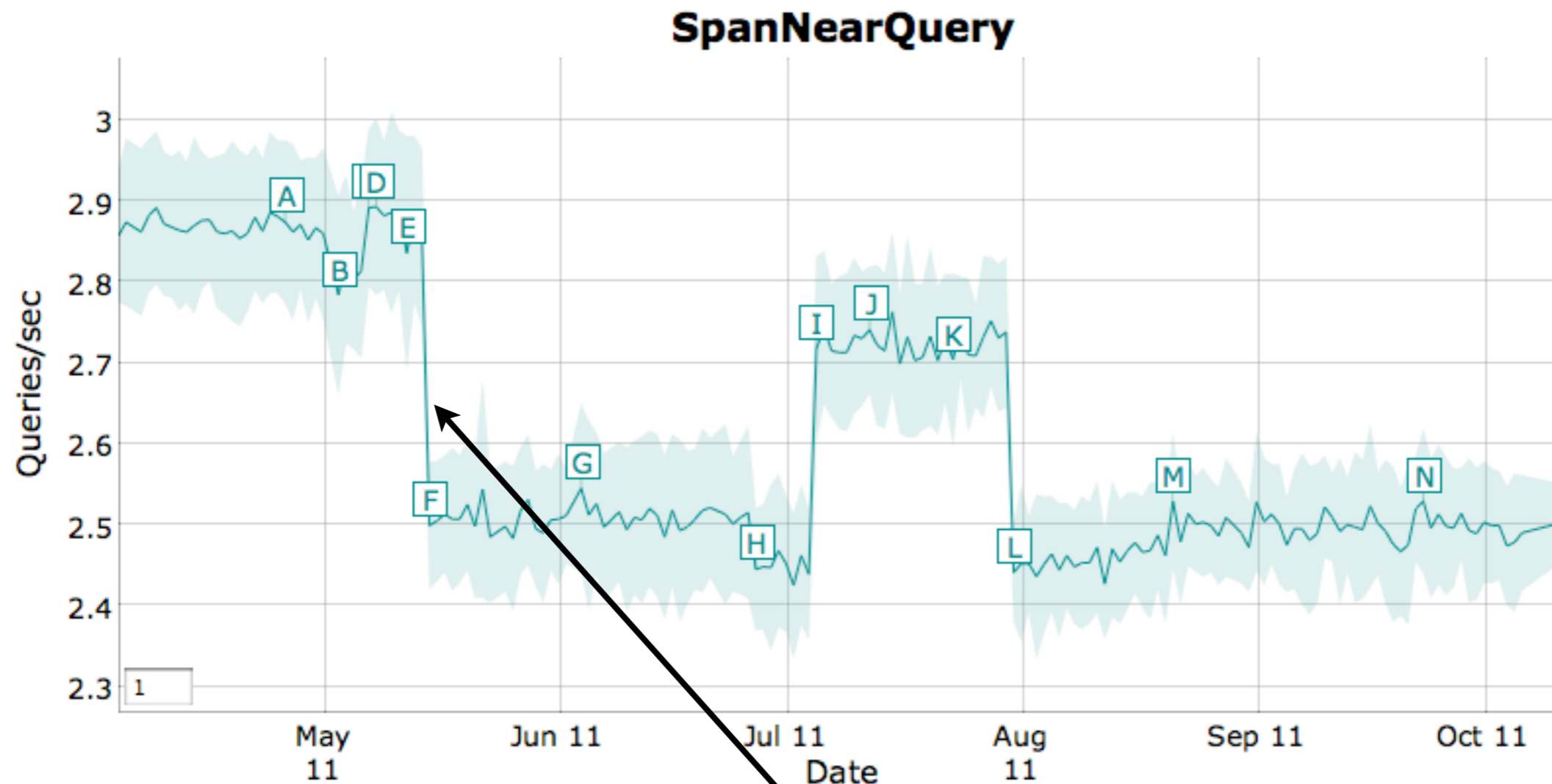
Updates - Hu? Incremental you know!

- Much easier (and closer) for not-indexed values
 - IndexDocValues
- Assumption:
 - Document Title OR Body changes are low frequent
 - PageRank OR User-Ratings change very frequently
- Maybe available in 4.0
- **Bottom Line:** this is still far away but on the list!

The JVM - or is it the JIT?



- Unpredictable Mr. JIT

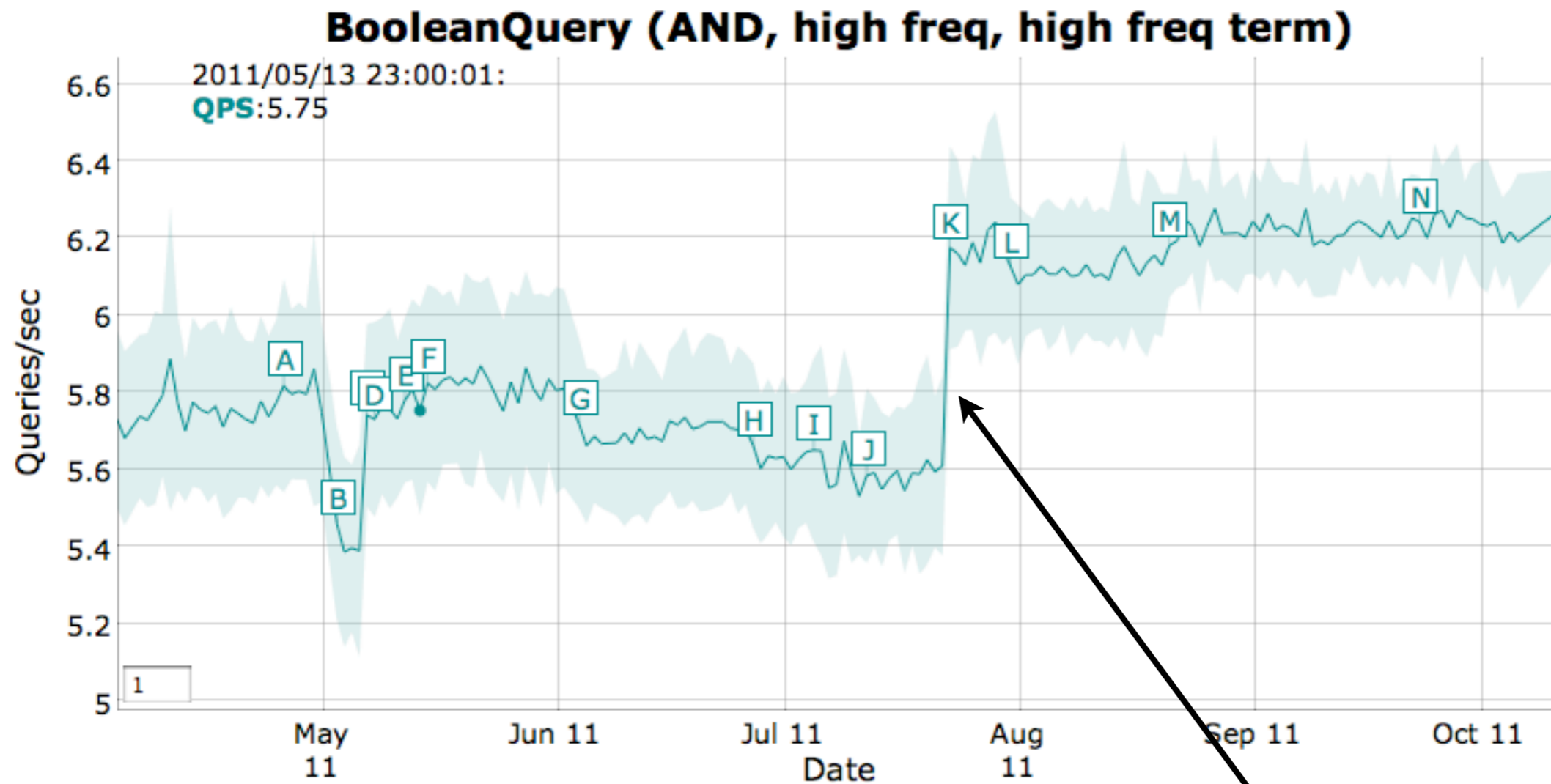


Grouping benchmark changes Spans? WTF?

The JVM - or is it the JIT?



- The cost of a virtual method call



ConjunctionScorer Code Specialization

The JVM - or is it the JIT?

- Lucene has a lot of **HOT** loops
 - Each TermScorer needs DocID & TermFreq for every possible hit
 - Calling DocsEnum#next() & #freq() adds up
 - Inlining seems unreliable

- **Solutions?**

Possible Solutions / Paths to explore

- Native Code / Generation (thats gonna be fun!)
- Code Specialization
 - Can bring 50% to 100% performance improvements
- ByteCode Generation & Query Compilation
 - Prototypes for FunctionQuery yields 300% speed improvements
- Bulk Reading APIs - *BulkPostings branch - watch out its hairy*
 - Reading more than one DocID / TermFreq at a time
 - More than one step **backwards** - API wise

- Specializing Queries at Runtime?
 - Might bring nice speed improvements per use-case
 - Problems arise with testing and correctness?
- Could help tremendously with bulk postings
 - Some people say the API is unusable (Uwe?)
 - Maybe you don't need to use it at all?
 - Would be nice if you could specify your query on a very high level and Lucene generates optimal code for you?

The Future beyond the core



- Users have two options
 - **Nothing** - plain Lucene (well its a lot already - a lot to code)
 - **All** - Solr / ElasticSearch etc.

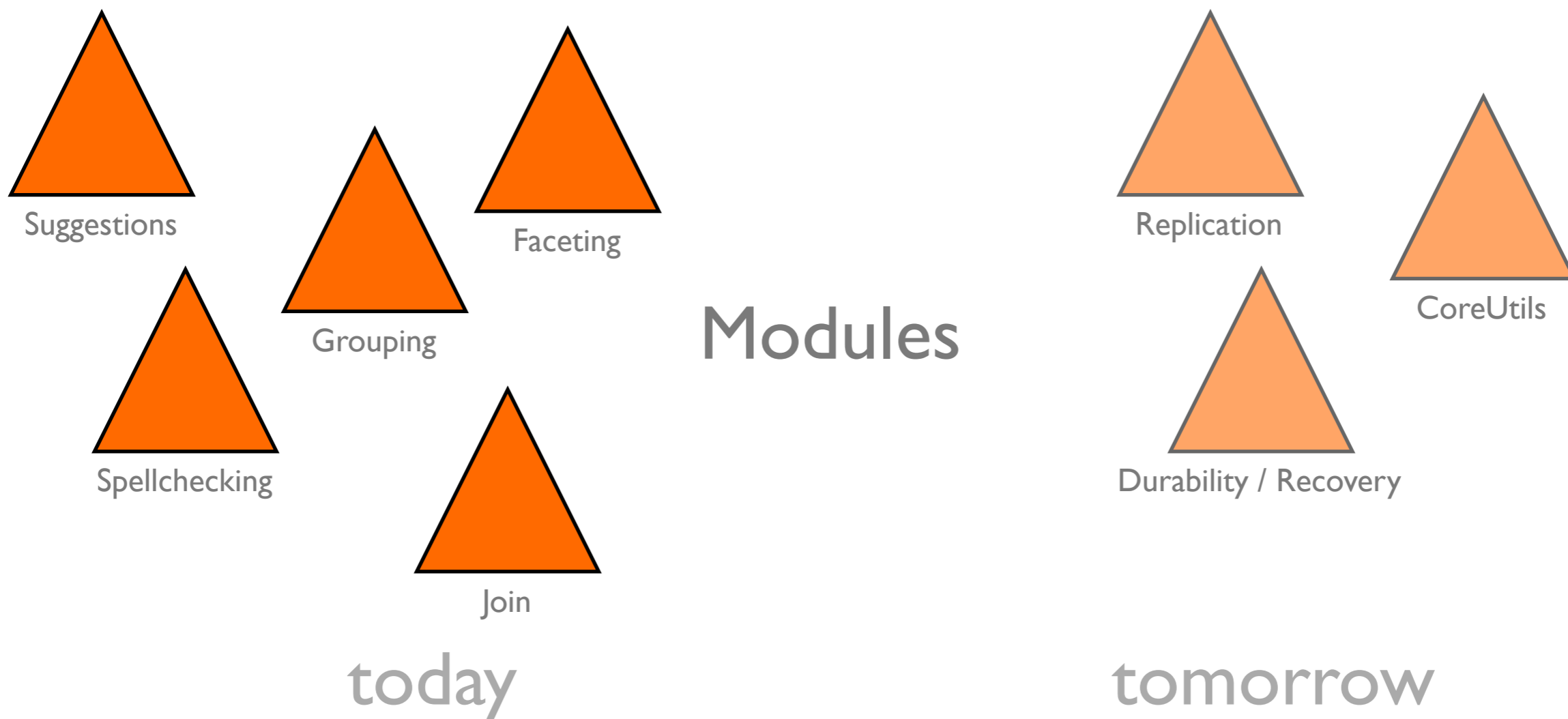
- **I'd like something in between, you?**

<dream>Lucene 5.0</dream>

- actually, XML is backwards: { “dream” : “Lucene 5.0” }
- Solr has grown, grown large and is showing its age!
- 95% of the time I only want one or two “services” Solr provides
 - still I got to use it - **all** or **nothing**!
 - I have to setup a (to me) heavy weight container (5 years ago Jetty / Tomcat was lightweight - times ‘r changing)
 - I got to figure out this documentation - fair enough!

{“dream” : “Lucene 5.0”}

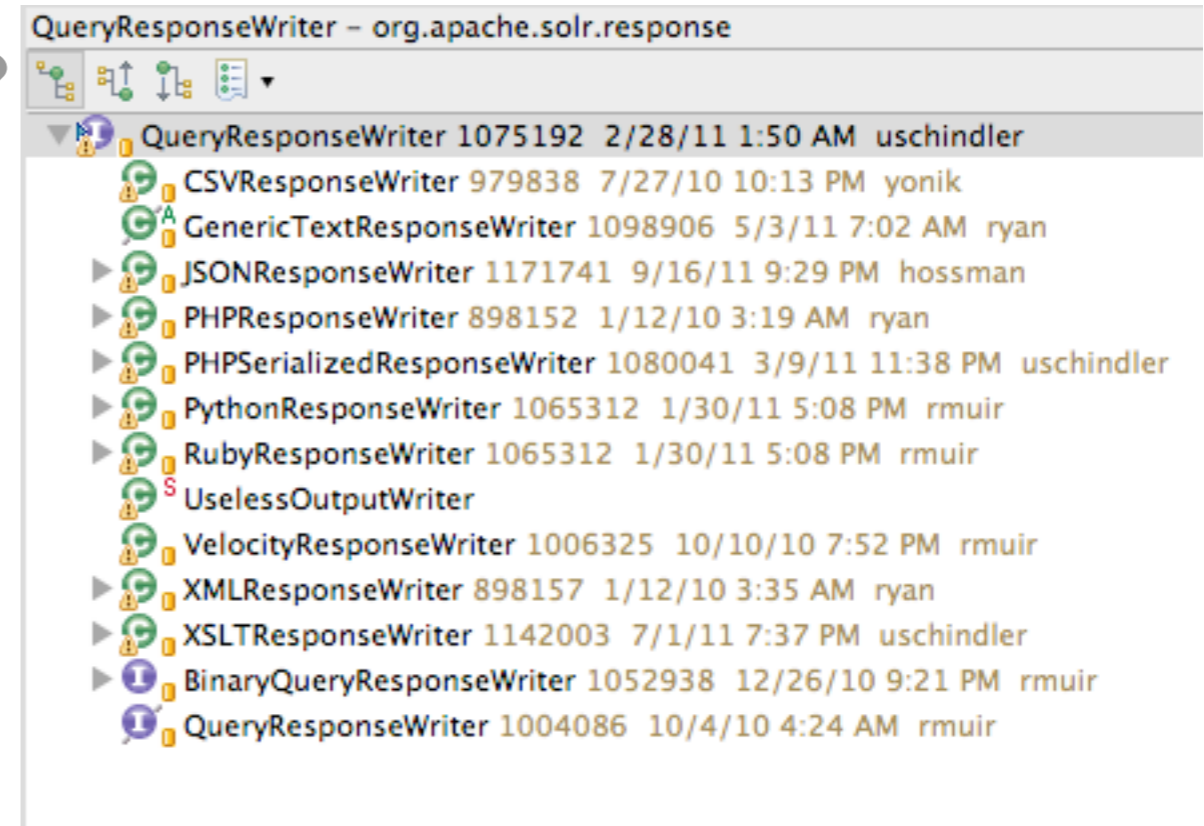
- Can we get this more modular, lightweight & lean?
 - I rather do some coding than configure 2 lines of XML, you?



Isn't this what Solr is?



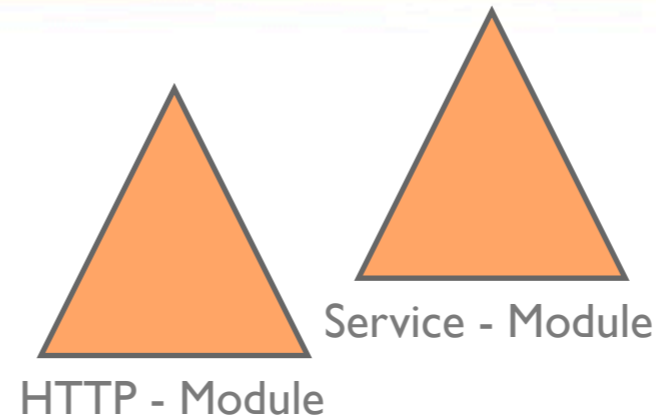
- Not quiet!
 - Lucene tries to provide APIs where you hardly can't take anything away
 - When I think of Solr, you can hardly add anything
- Everybody should be able to build their own \$Solr
- **How hard will it be to draw the line?**
- Who is going to benefit?



Back to {"dream" : "Lucene 5.0"}



- Can we go one step further?



- Elasticsearch did a great job making things dead simple!
 - we should follow this example and *less might be more eventually!*
- Taking it as far as Elasticsearch (all or nothing again) seems not the right path for Lucene but simple is good, no?

-
- This was my personal vision maybe not the one other people have.
 - Lets see what the community wants / needs - It's all about the users!

Thank you!