SearchWorkings

Apache Lucene - a library retrieving data for millions of users

Simon Willnauer
Apache Lucene Core Committer & PMC Chair
simonw@apache.org / simon.willnauer@searchworkings.org

About me?

- Lucene Core Committer
- Project Management Committee Chair (PMC)
- Apache Member
- BerlinBuzzwords Co-Founder
- Addicted to OpenSource

Agenda

- Apache Lucene a historical introduction
- (Small) Features Overview
- The Lucene Eco-System
- Upcoming features in Lucene 4.0
- Maintaining superior quality in Lucene (backup slides)
- Questions

Apache Lucene - a brief introduction

- A fulltext search library entirely written in Java
- An ASF Project since 2001 (happy birthday Lucene)
- Founded by Doug Cutting
- Grown up - being the de-facto standard in OpenSource search
- Starting point for a other well known projects
- Apache 2.0 License

Where are we now?

- Current Version 3.4 (frequent minor releases every 2 - 4 month)
- Strong Backwards compatibility guarantees within major releases
- Solid Inverted-Index implementation
- large committer base from various companies
- well established community
- Upcoming Major Release is Lucene 4.0 (more about this later)

(Small) Features Overview

- Fulltext search
  - Boolean-, Range-, Prefix-, Wildcard-, RegExp-, Fuzzy-, Phase-, & SpanQueries
- Faceting, Result Grouping, Sorting, Customizable Scoring
- Large set of Language / Text-Processing Tools (Analyzers)
- High-Throughput incremental indexing (Create, Update, Delete)
- Schema free
- Query Suggestions, SpellChecking, Highlighting
- No Durability Guarantees - **Hey its not a database!**

Former Lucene Subprojects

- Apache Nutch

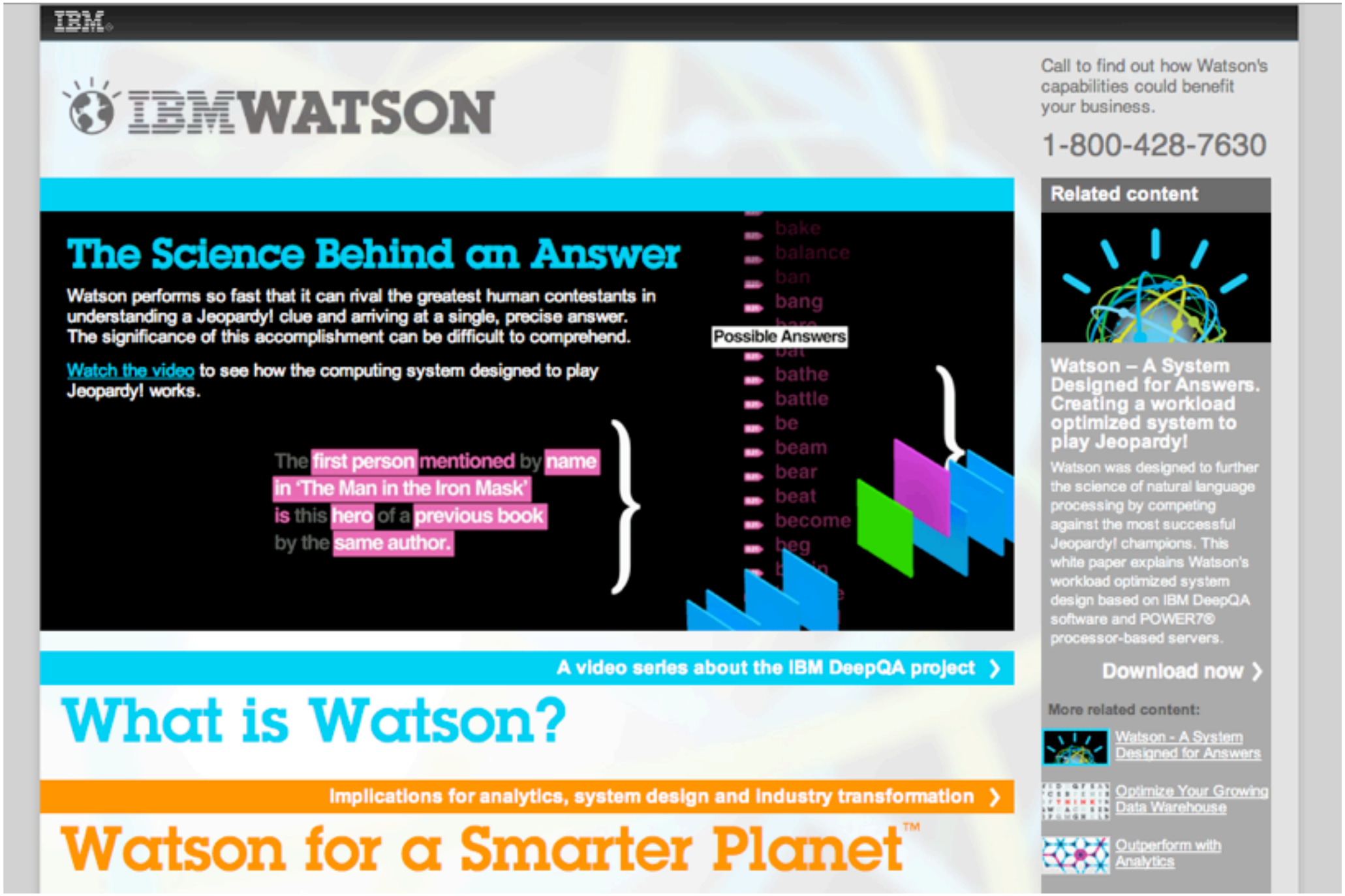
  - 2002 - 2004
    - web-scale, crawler based search engine on top of Lucene
    - distributed with sort / merge based processing
  - 2004 - 2006
    - added DFS & MapReduce to Nutch known as Nutch-DFS
    - two part-time devs, over two years
- Apache Hadoop (2006 - today)

- Apache Mahout (2008 - today)

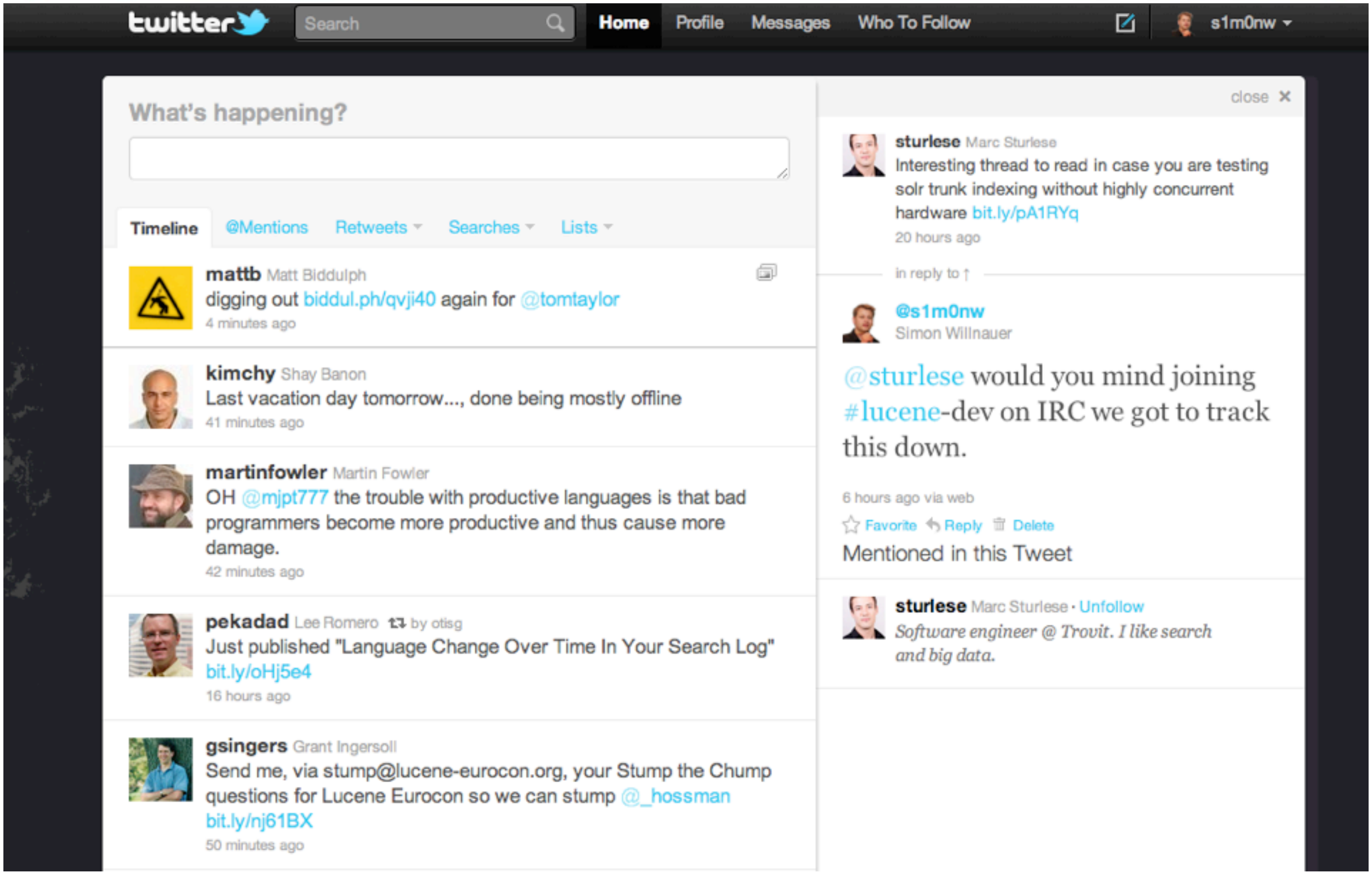
Lets look at some use-cases

I am always surprised what people do with Lucene...

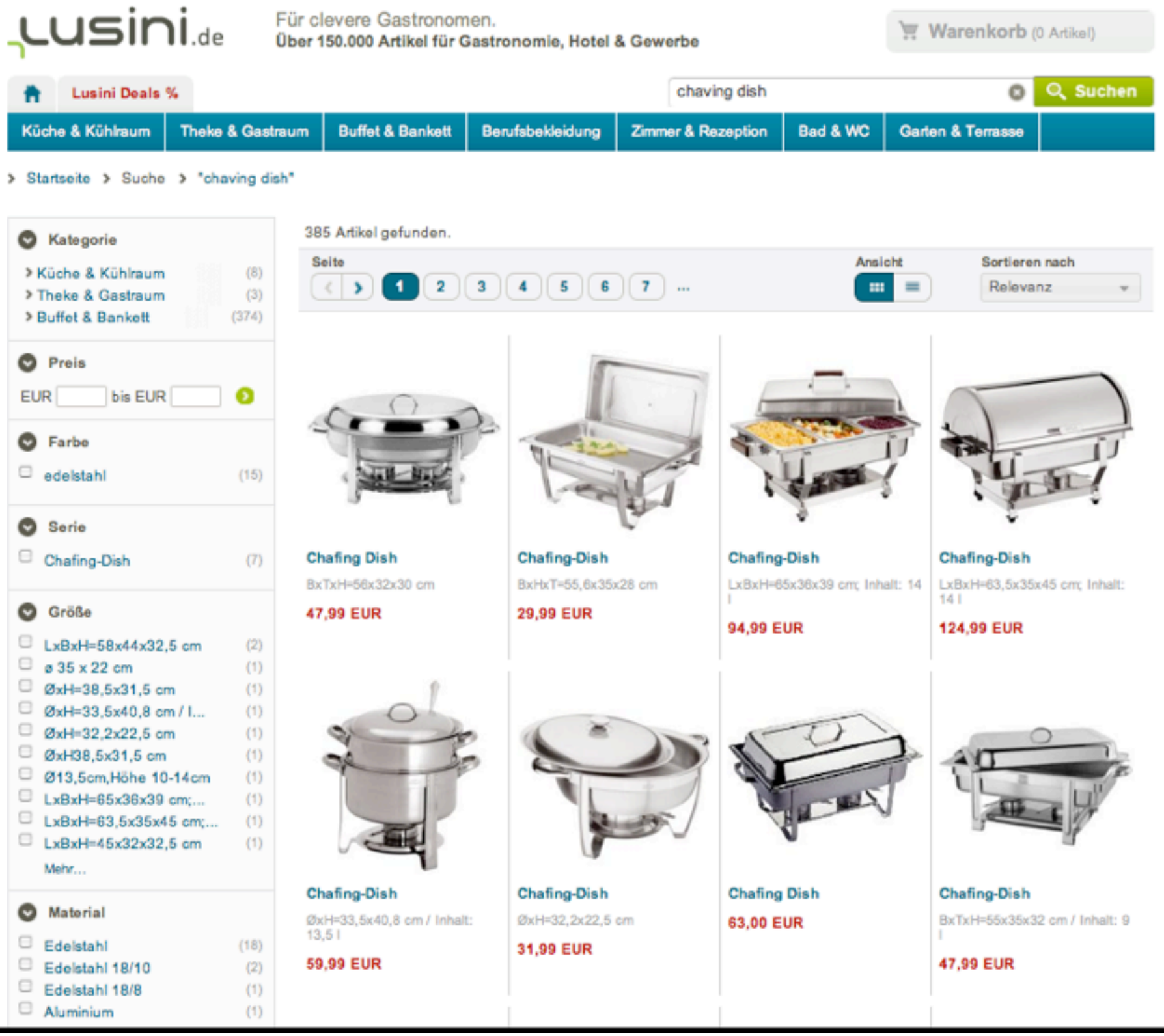
Answering Questions - IBM Watson



Realtime Search - Twitter



Search Driven Webshops



Scientific Map Search

Not logged in (log in or sign up)

PANGAEA®
Data Publisher for Earth & Environmental Science

Always quote citation when using data!

Data Description Show Map Google Earth

Citation: **Lo Bue, Nadia (2010):** Physical oceanography and methane concentrations measured with the MEDUSA system during Maria S. Merian cruise MSM15/1 at station MSM15/1_339-1. *Istituto Nazionale di Geofisica e Vulcanologia*, doi:10.1594/PANGAEA.746135

Project(s): **In situ monitoring of oxygen depletion in hypoxic ecosystems of coastal and open seas and land-locked water bodies** (HYPOX)

Coverage: *Median Latitude:* 44.783989 * *Median Longitude:* 31.962154 * *South-bound Latitude:* 44.776580 * *West-bound Longitude:* 31.948263 * *North-bound Latitude:* 44.792838 * *East-bound Longitude:* 31.973925

Date/Time Start: 2010-04-20T09:33:52 * *Date/Time End:* 2010-04-20T13:39:12

Minimum DEPTH, water: 7.8 m * *Maximum DEPTH, water:* 215.8 m

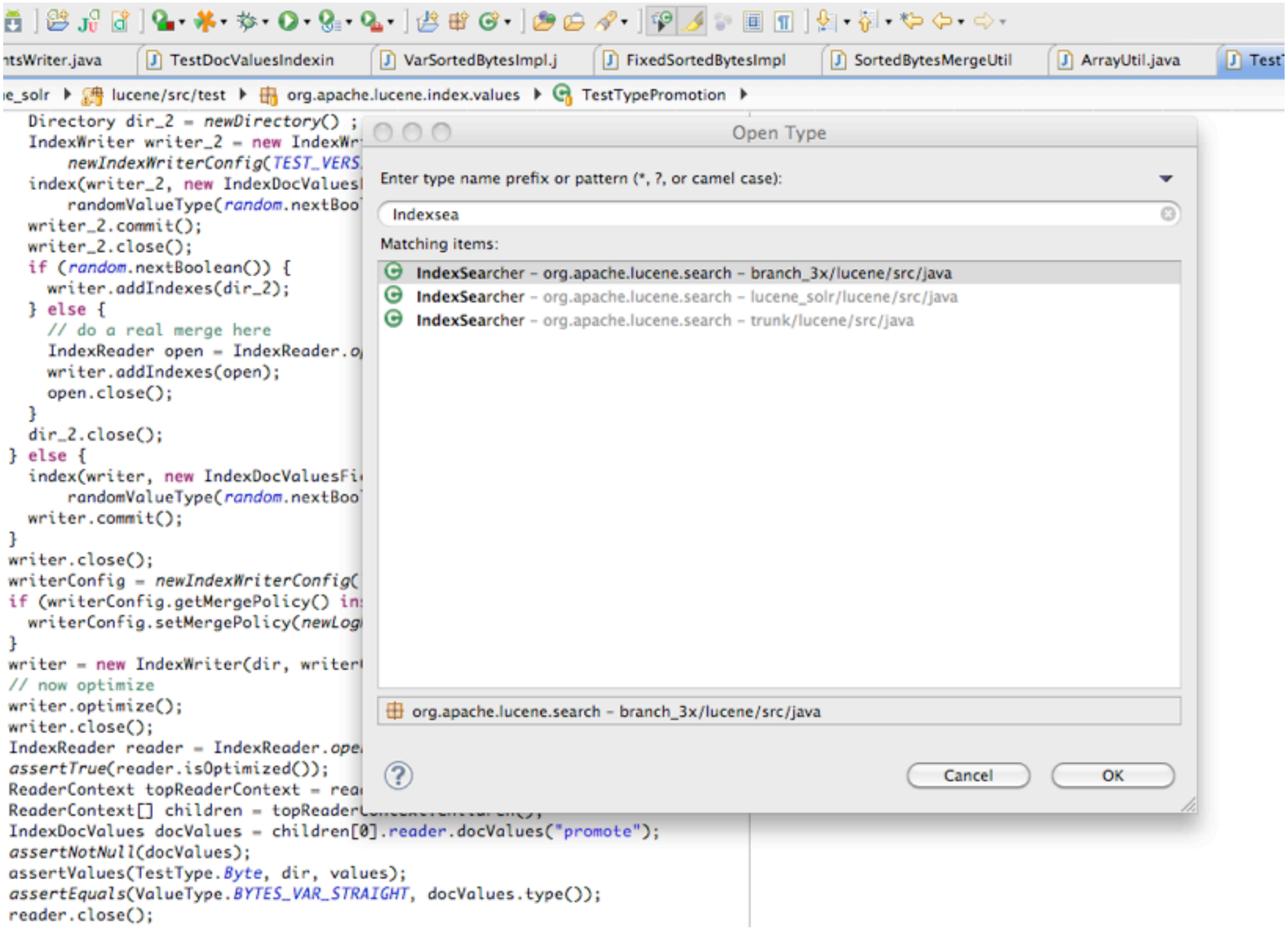
Event(s): **MSM15/1_339-1** * *Latitude Start:* 44.776668 * *Longitude Start:* 31.973700 * *Latitude End:* 44.792483 * *Longitude End:* 31.948148 * *Date/Time Start:* 2010-04-20T09:58:00 * *Date/Time End:* 2010-04-20T13:26:47 * *Location:* Black Sea * *Campaign:* MSM15/1 * *Basis:* Maria S. Merian * *Device:* Remote operated vehicle MEDUSA

Further details: hdl:10013/epic.36070.d001

Parameter(s):

#	Name	Short Name	Unit	Principal Investigator	Method	Comment
1	DATE/TIME	Date/Time				Geocode
2	LATITUDE	Latitude				Geocode
3	LONGITUDE	Longitude				Geocode
4	DEPTH, water	Depth water	m			Geocode
5	Pressure, water	Press	dbar	Lo Bue, Nadia		
6	Speed	Speed	kn	Lo Bue, Nadia		
7	Course	Course	deg	Lo Bue, Nadia		
8	Oxygen	O2	µmol/l	Lo Bue, Nadia		
9	Oxygen, saturation	O2 sat	%	Lo Bue, Nadia		

The Eclipse IDE



The Lucene Eco System

- Since Lucene by itself is only a library a rather small percentage of users are using Lucene directly
- Several Projects emerged on top of Lucene

Apache Solr

elasticsearch.

Katta

- But search needs data, right? And processing? Content Extraction?

hadoop

mahout

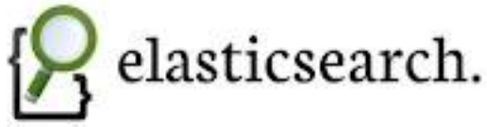
Apache™ Tika

openNLP

Apache Solr

- A full featured enterprise search server
  - Living in a ServletContainer or embedded
  - Exposing almost all Lucene features via HTTP (Json, XML, etc)
- Lucene's first class citizen - living in the same codebase since 2009
- Grown mature - showing its age!
- Very large community, very good support (commercial and free)
- Fixed Schema on top of Lucene
- Apache 2.0 Licensed

ElasticSearch



- Fairly new, scalable Search engine
- Simple and straight forward runtime system
- Targeted for cloud deployments
- Feature set is limited to distributed features (so far)
- Sharding is a first class citizen
- Rather small but growing community
- Apache 2.0 License

Apache Hadoop

- Framework for processing large dataset with the MapReduce programming model
- Very high latency - no, you can not use this for realtime processing
- Build Lucene indices from massive amounts of data
- Pre-process data for indexing
- Post-process data from searches (query logs, klick data, etc)
- Large community, Good support (commercial and free)
- Apache 2.0 License

Apache Mahout

- Scalable MachineLearning library / framework
- Provides tools for:
  - Recommendations / collaborative filtering
  - Classification
  - Clustering
- Pretty young project but growing
- Build on top of Hadoop for large scale

What is left?

- We have tools for:
  - Distributed search
  - Large data processing
  - Machine learning
- What we need is:
  - Tools to extract data from “documents”
  - Do algorithmic processing of extracted data

Apache Tika & Apache OpenNLP

- Tika
  - Extracting text from common formats
  - Supports PDF, MS Office docs, OpenOffice, 20+ other formats
- OpenNLP
  - A machine learning toolkit tailored for Natural Language Processing
  - Sentence segmentation, part of speech tagging, named entity recognition, coreference resolution

Enough high level introductions... lets get a bit deeper into Apache Lucene

Upcoming features in Lucene 4.0

- Lucene 4 is the first major release since 2009
- In contrast to 3.0, Lucene 4.0 breaks Backwards Compatibility
  - New Redesigned APIs
  - Entirely new & customizable Index Format
  - Binary String Representation - we are back to Byte-Arrays!
  - Fixing long standing inconsistencies
- A similar way like Python 3k - at some point you need to get rid of ancient APIs and file formats for good.

Why breaking BW-Compatibility?

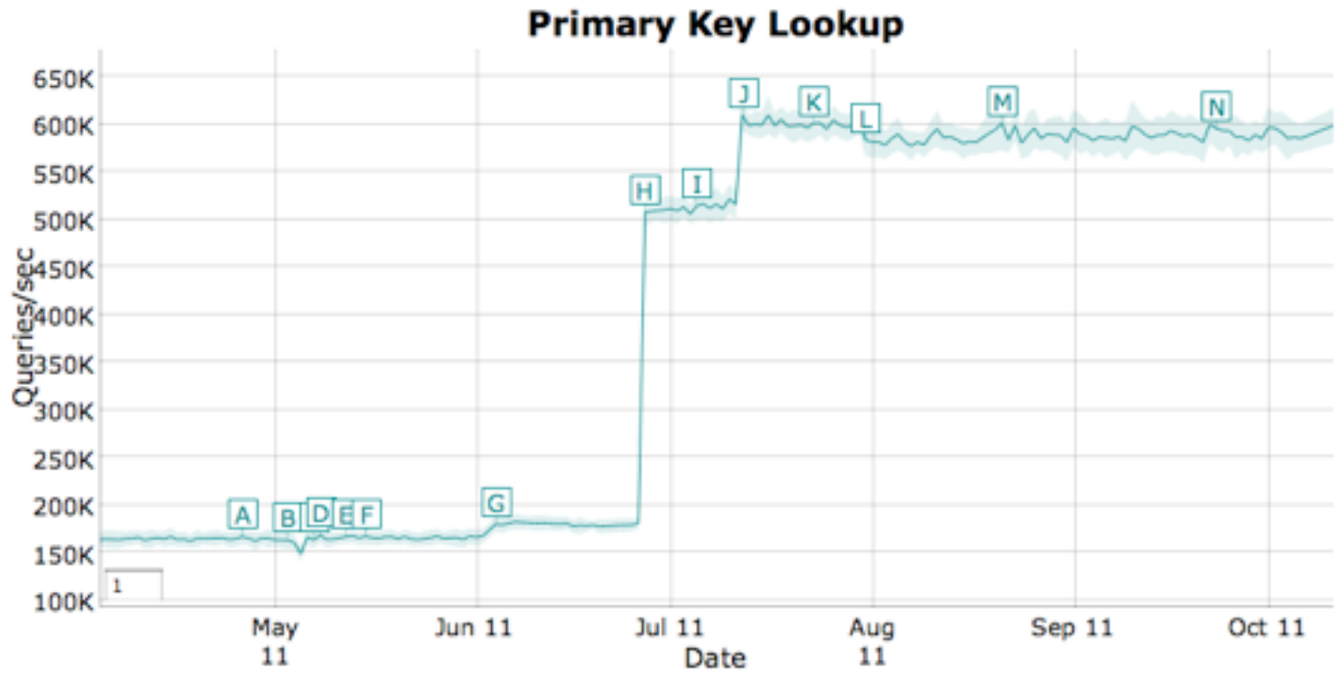
- Speed, Speed, Speed oh and Speed
- Lucene has grown and lost flexibility over time
- Lot of features required major API and algorithmic overhaul
  - New FuzzyQuery needed new features in the term dictionary
  - File Formats were pretty much set into stone once released
  - Lots of different users have very unique requirements and eventually its all about the user!
- It was time to “get it right”

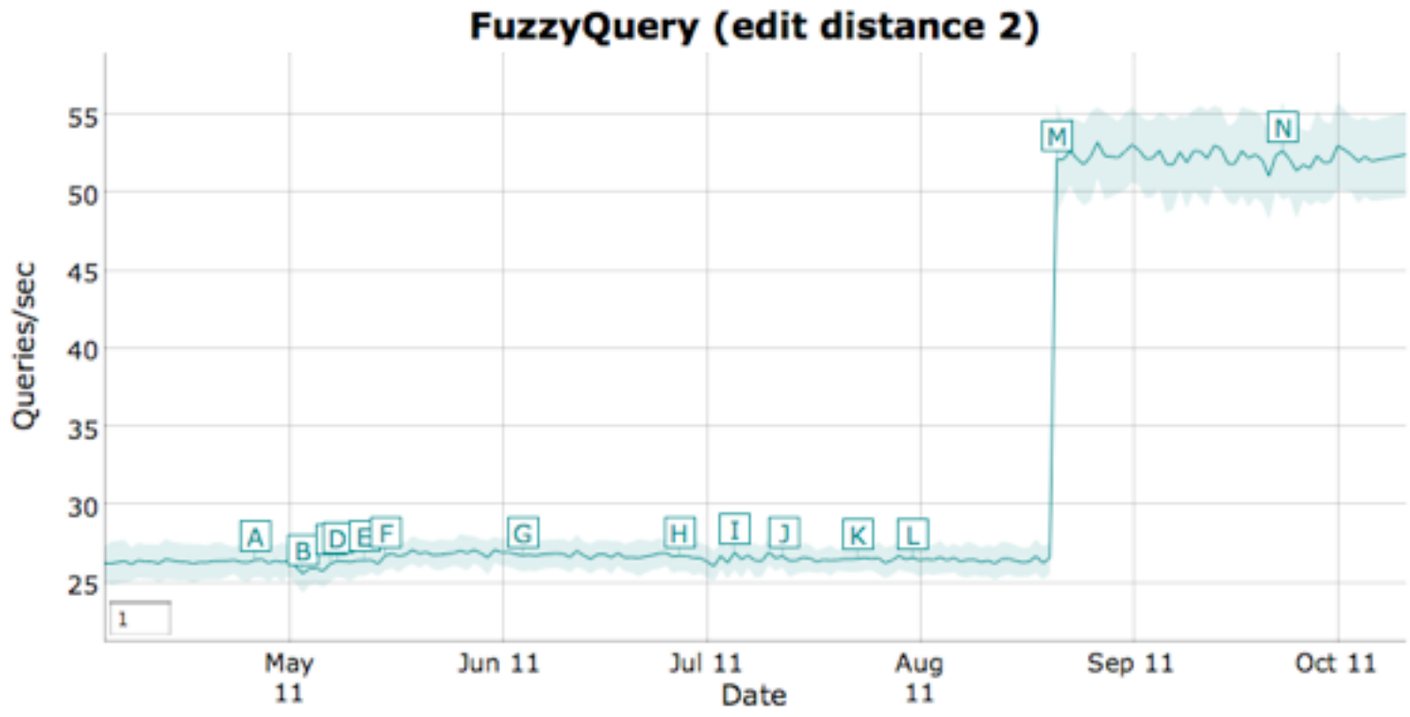
Some random improvements

- 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

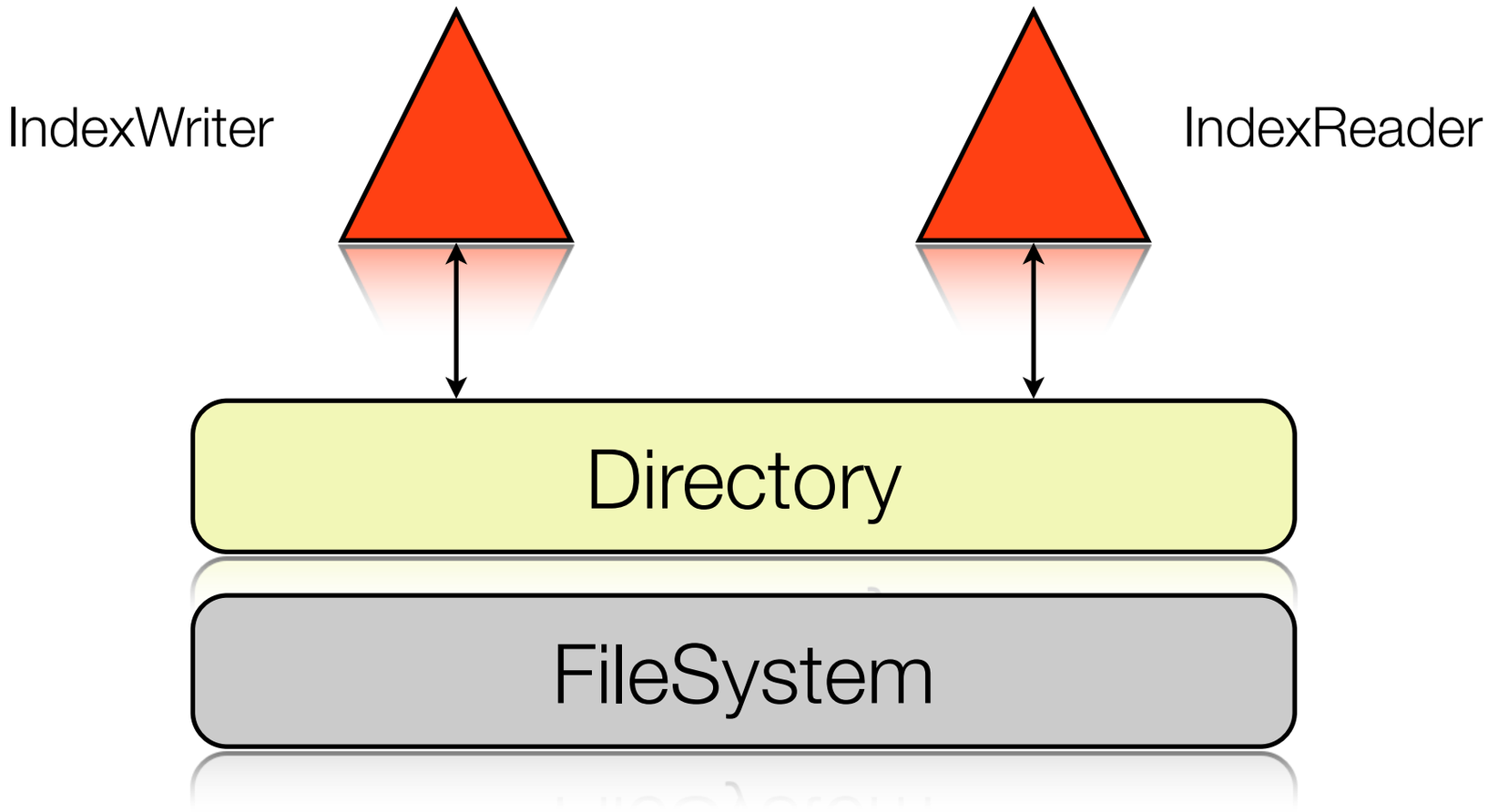
Your mileage may vary!

Fuzzy & PK Lookup over time

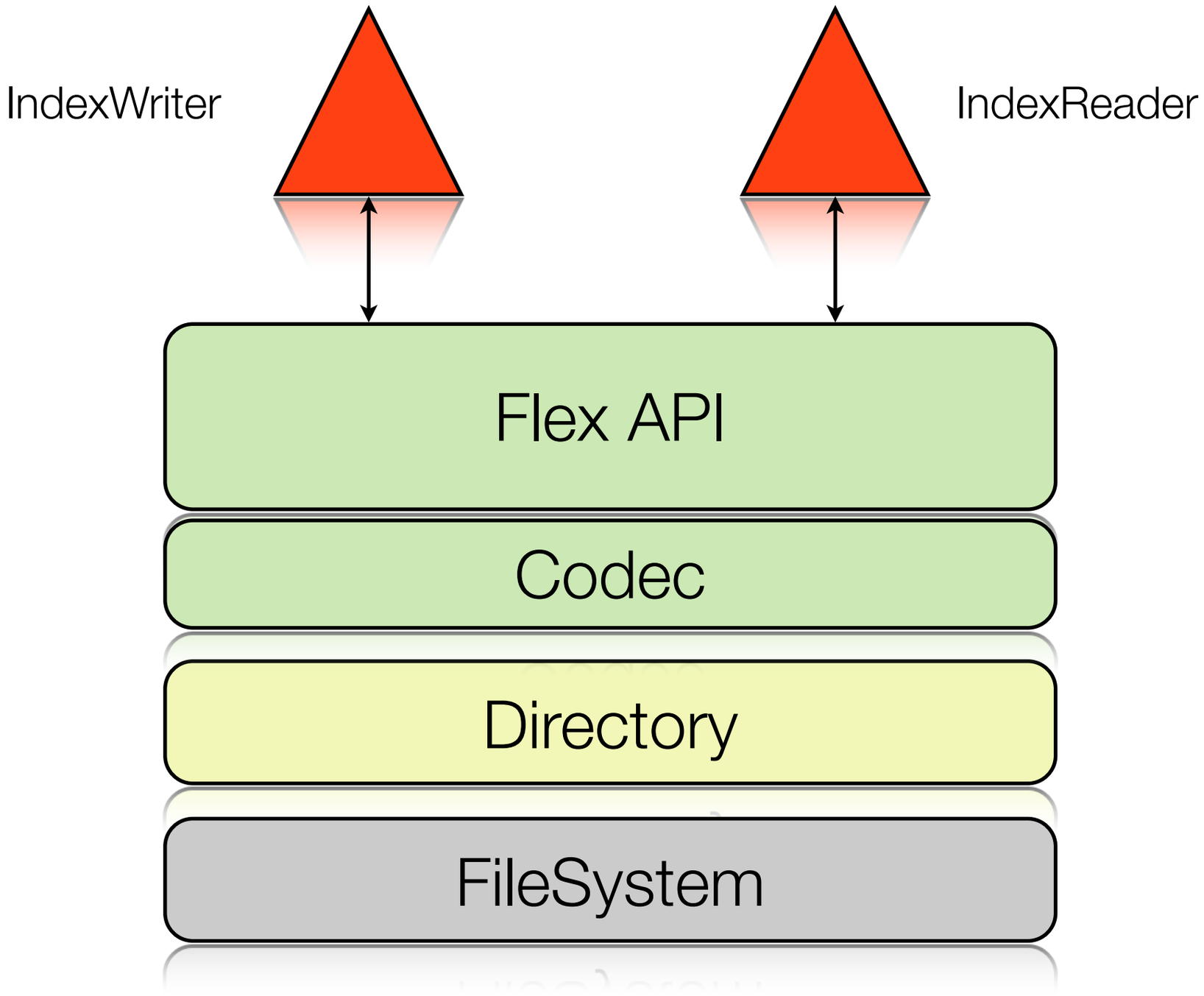




Index Access API in 3.x



Flexible Indexing in 4.0



Flexible Indexing in 4.0

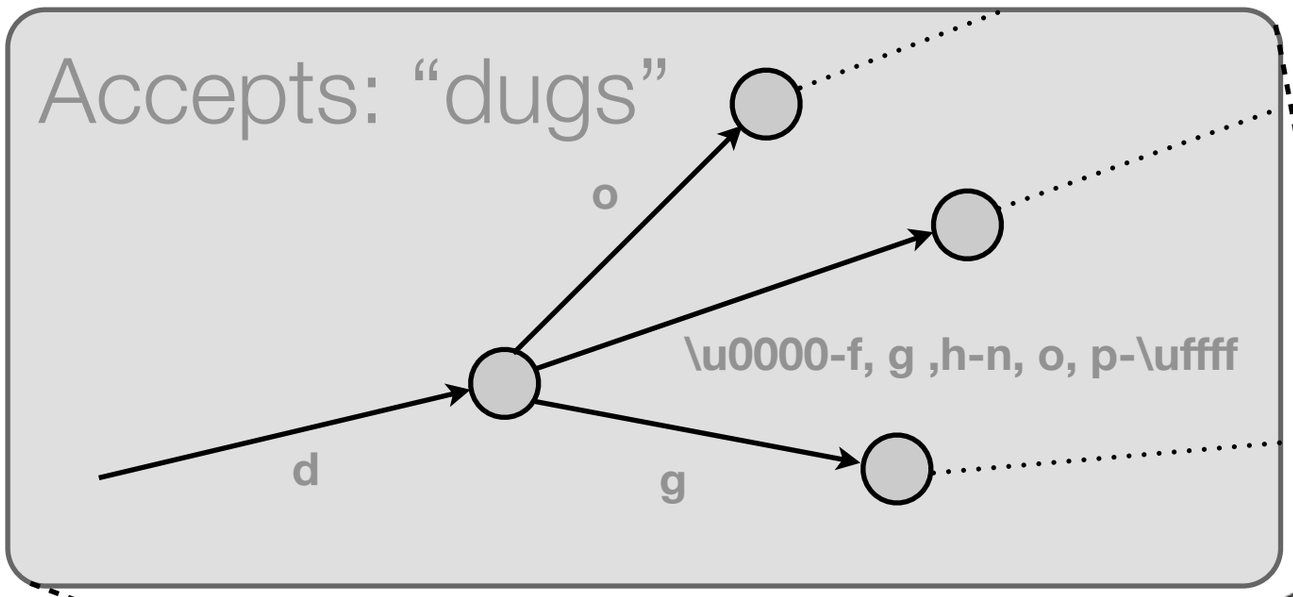
- Allows to customize low level reading and writing
- Performance optimizations and flexibility are provided per index field
- Each **Codec** can be versioned and evolve over time
- 3.x indices are simply a dedicated codec
- Conversion / Index upgrade happens transparently in the background

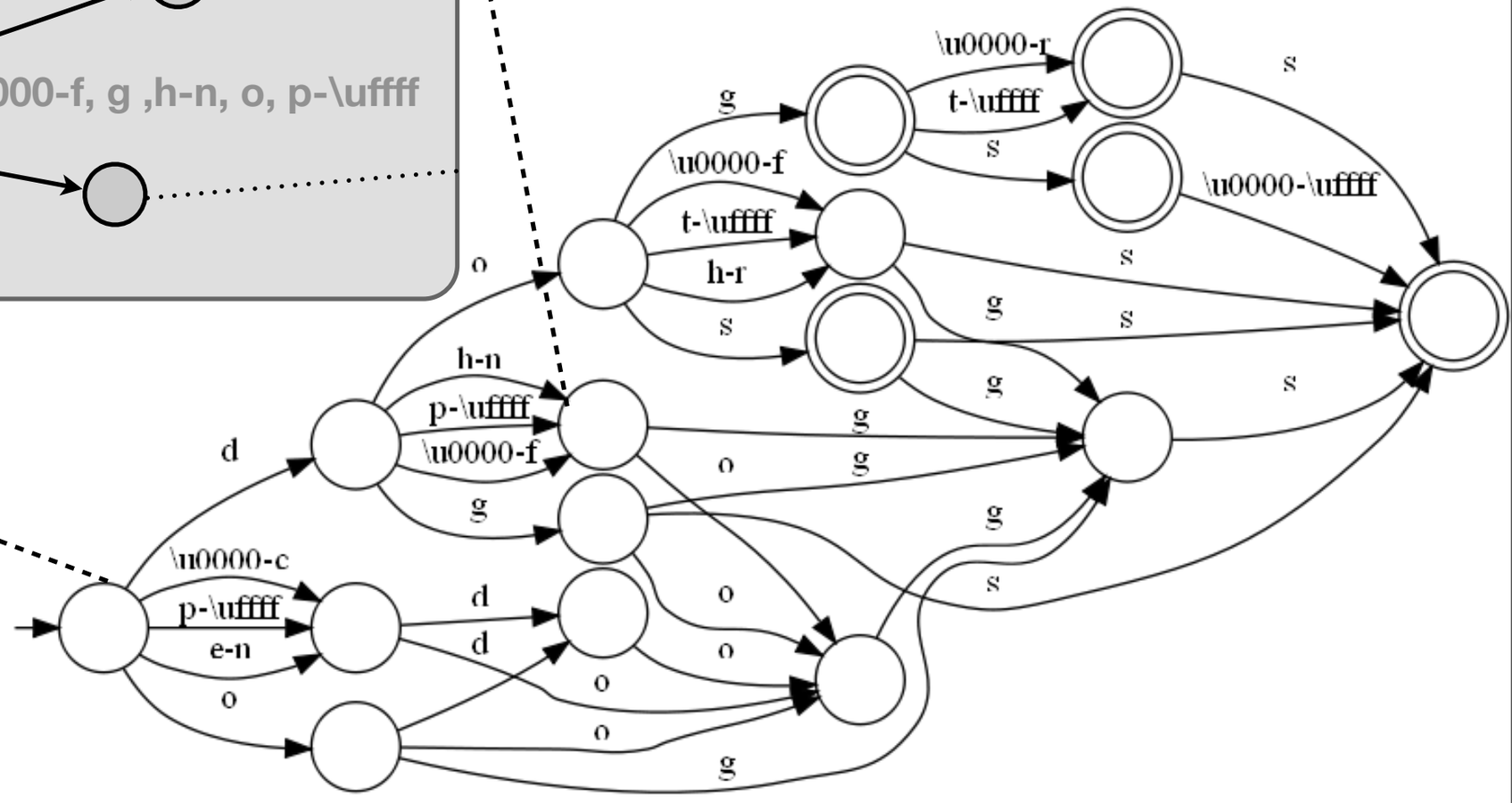
Automation Queries

- Complex Queries matching more than one terms are historically expensive.
- *FuzzyQuery* for instance required to examine O(T) terms (T = # terms in all documents in the search field)
- New Lucene API semantics allow major optimizations over 3.x
- Some Term-Dictionary implementation offer efficient intersection procedures
- Query as a DFA (Deterministic Finite Automaton)

Automaton Queries (Fuzzy)

Example DFA for “dogs” Levenshtein Distance 1





Automaton Queries

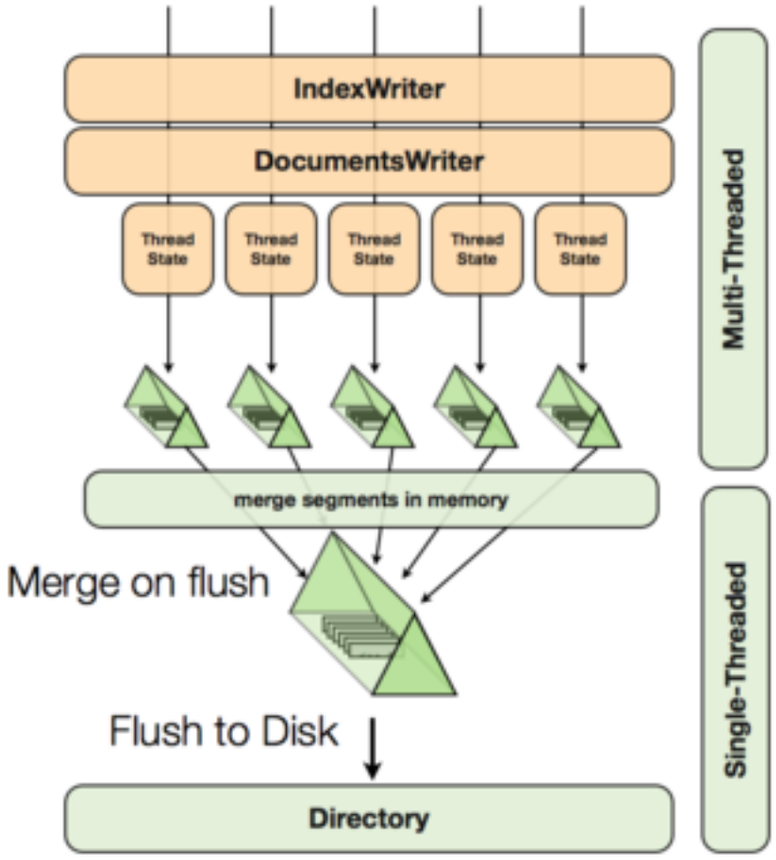
- Provides a very flexible & powerful language to retrieve data
- Automatons can be combined
  - *FuzzyPrefixQuery* for instance
- Opens the door for further improvements
  - Query Expansion vs. Stemming
- Can be used on large corpuses

```
// a term representative of the query, containing the field.
// term text is not important and only used for toString() and such
Term term = new Term("body", "dogs~1");

// builds a DFA for all strings within an edit distance of 2 from "bla"
Automaton fuzzy = new LevenshteinAutomata("dogs").toAutomaton(1);

// concatenate this with another DFA equivalent to the "*" operator
Automaton fuzzyPrefix = BasicOperations.concatenate(fuzzy, BasicAutomata
    .makeAnyString());

// build a query, search with it to get results.
AutomatonQuery query = new AutomatonQuery(term, fuzzyPrefix);
```

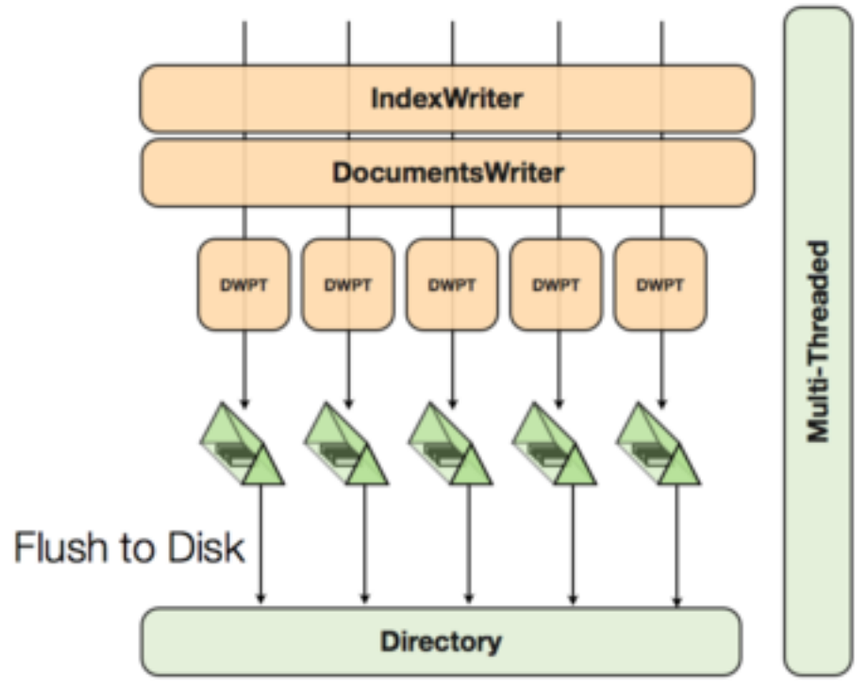
DocumentsWriterPerThread

- Incremental indexing offers concurrent flushing
  - Efficiently utilizes IO systems
  - Large performance gains for high concurrent systems
  - Less impact if IO is slow
- Non-Blocking Indexing process
- Up to 280% throughput improvements

DocumentsWriterPerThread

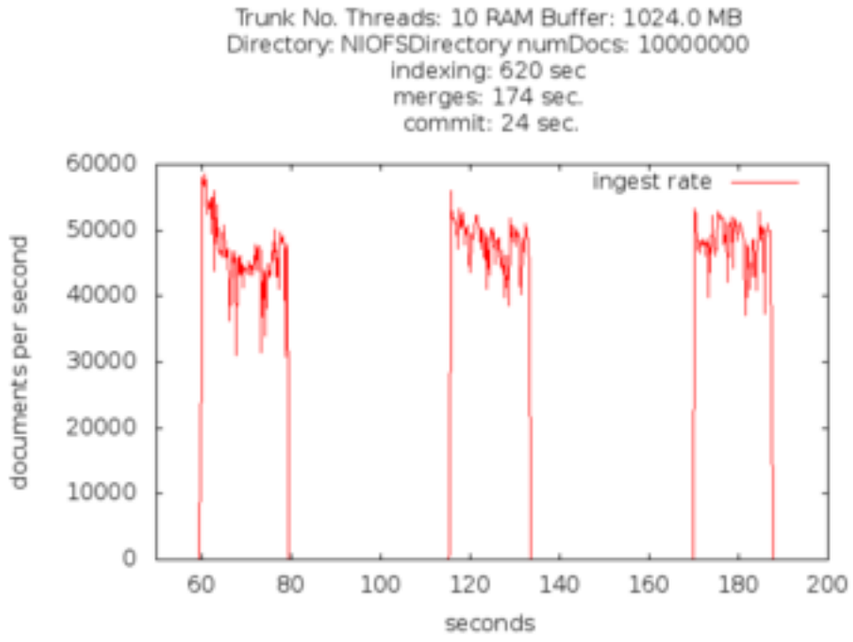


Indexing with Lucene 3.x

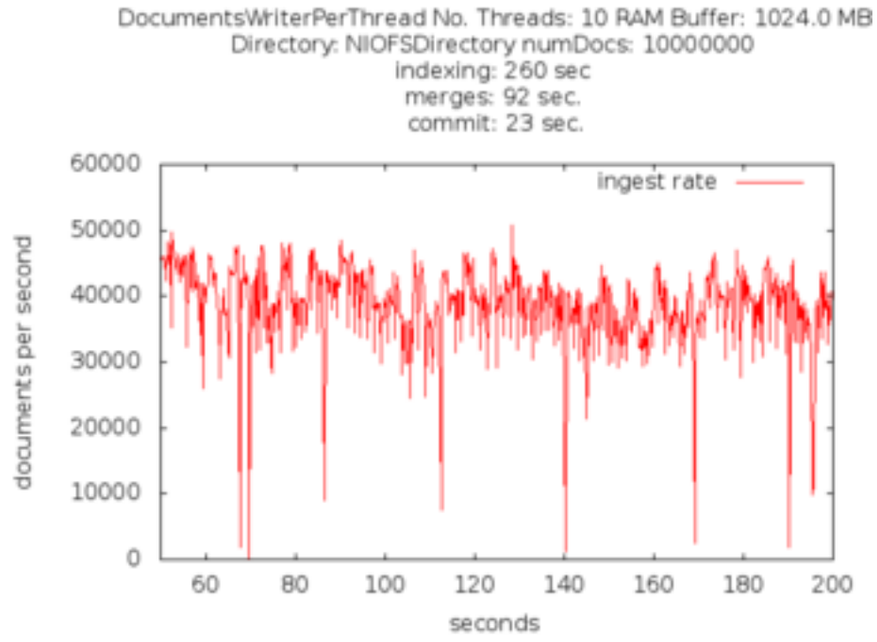


Indexing with Lucene 4.0

DocumentsWriterPerThread

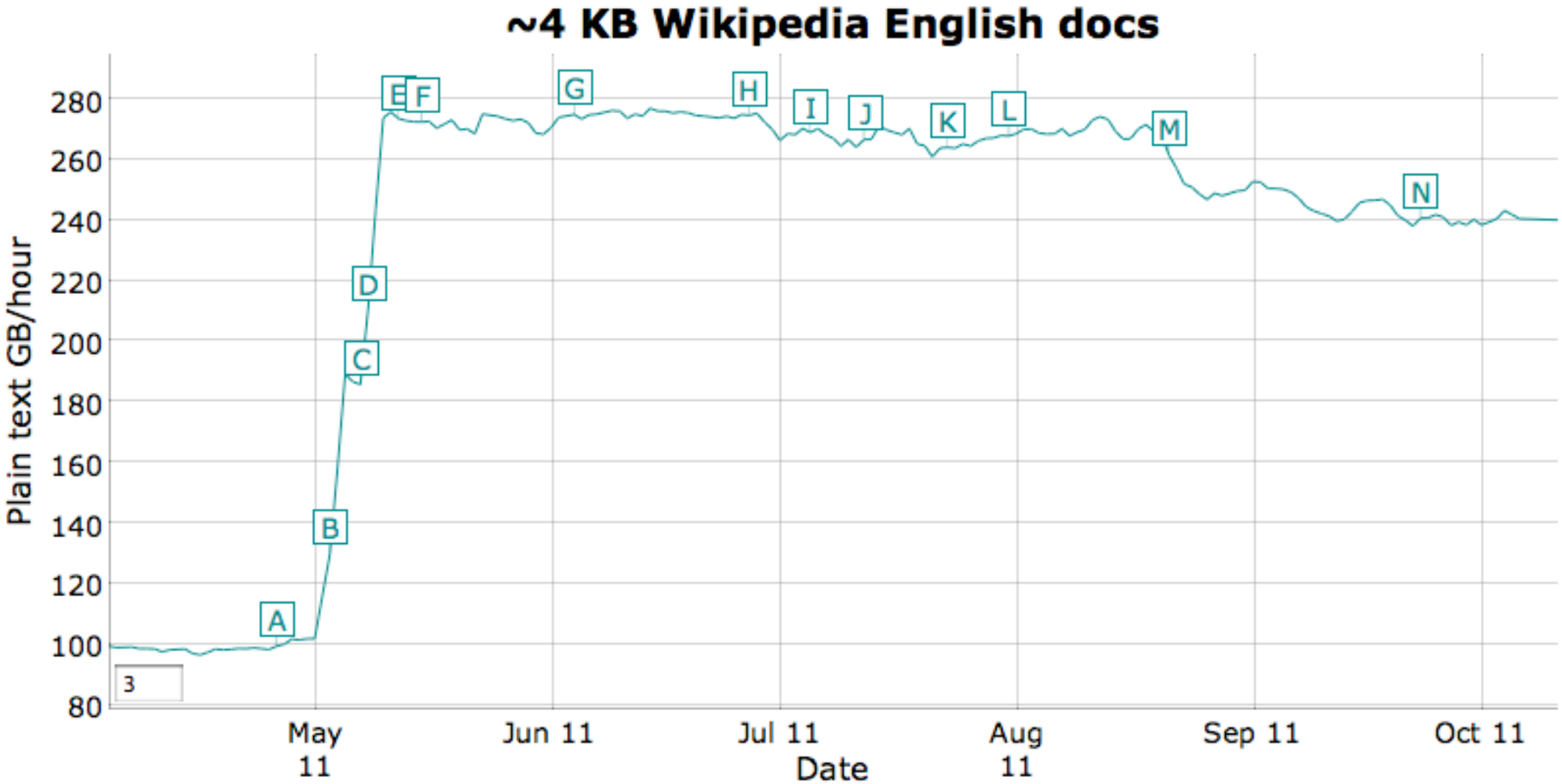


Indexing with Lucene 3.x



Indexing with Lucene 4.0

Indexing Throughput over time



When can we get this?

- Lucene 4.0 is supposed to be released rather sooner than later :)
- We all hope to release within the next 6 month or even this year.
- Since this is a major break in BW Compat and we don’t plan to do this very often is crucial to get things right.

Some time left?

Questions? or Backup-Slides?

Maintaining Superior Quality in Lucene

- Maintaining a Software Library used by thousands of users comes with responsibilities
- Lucene has to provide:
  - Stable APIs
  - Backwards Compatibility
- Needs to prevent performance regression
- Lets see what Lucene does about this.

Tests getting complex in Lucene

- Lucene needs to test
  - 10 different Directory Implementations
  - 8 different Codec Implementation
  - tons of different settings on IndexWriter
  - Unicode Support throughout the entire library
  - 5 different MergePolicies
  - Concurrency & IO

Solution: Randomized Testing

- Each test is initialized with a random seed
- Most tests run with:
  - A random Directory, MergePolicy, IndexWriterConfig & Codec
- # iterations and limits are selected at random
- Open file handles are tracked and test fails if they are not closed
- Tests use Random Unicode Strings (we broke several JVM already)
- On failure, test prints a random seed to reproduce the test

Randomized Testing - the Problem

- You still need to write the test :)
- Your test can fail at any time
  - Well better than not failing at all!
- Failures in concurrent tests are still hard to reproduce even with the same seed

Investing in Randomized testing

- Lucene gained the ability to rewrite large parts of its internal implementations without much fear!
- Found 10 year old bugs in every day code
- Prevents leaking file handles (random exception testing)
- Gained confidence that if there is a bug we gonna hit it one day

Thank you for you attention!

Questions?