Heavy Committing: Flexible Indexing in Lucene 4.0

Uwe Schindler

SD DataSolutions GmbH,
uschindler@sd-datasolutions.de

My Background

- I am committer and PMC member of Apache Lucene and Solr. My main focus is on development of Lucene Java.

- Implemented fast numerical search and maintaining the new attribute-based text analysis API. Well known as Generics and Sophisticated Backwards Compatibility Policeman.

- Working as consultant and software architect for SD DataSolutions GmbH in Bremen, Germany. The main task is maintaining PANGAEA (Publishing Network for Geoscientific & Environmental Data) where I implemented the portal's geo-spatial retrieval functions with Lucene Java.

- Talks about Lucene at various international conferences like ApacheCon EU/US, Berlin Buzzwords, Lucene Revolution, Lucene Eurocon and various local meetups.

Agenda

- Motivation

- API changes in general

- Codecs

- Future

- Wrap up

Lucene up to version 3.2

- Lucene started > 10 years ago
  - Lucene's vInt format is old and not as friendly as new compression algorithms to CPU's optimizers *(exists since Lucene 1.0)*

- It's hard to add additional statistics for scoring to the index
  - IR researchers don't use Lucene to try out new algorithms

- Small changes to index format are often huge patches covering tons of files
  - Example from days of Lucene 2.4: final omit-TFaP patch of LUCENE-1340 is ~70 KiB covering ~25 files

Why Flexible Indexing?

- Many new compression approaches ready to be implemented for Lucene

- Separate index encoding from terms/postings enumeration API

- Make innovations to the postings format easier

**Targets to make Lucene extensible even on the lowest level**

Agenda

- Motivation

- API changes in general

- Codecs

- Future

- Wrap up

Quick Overview

- Will be **Apache Lucene ≥ 4.0** <u>only</u>!

- Allows to
  - store new information into the index
  - change the way existing information is stored

- Under heavy development
  - almost stable API, may break suddenly
  - lots of classes still in refactoring

- Replaces a lot of existing classes and interfaces → **Lucene 4.0** will not be backwards compatible *(API wise)*

Quick Overview
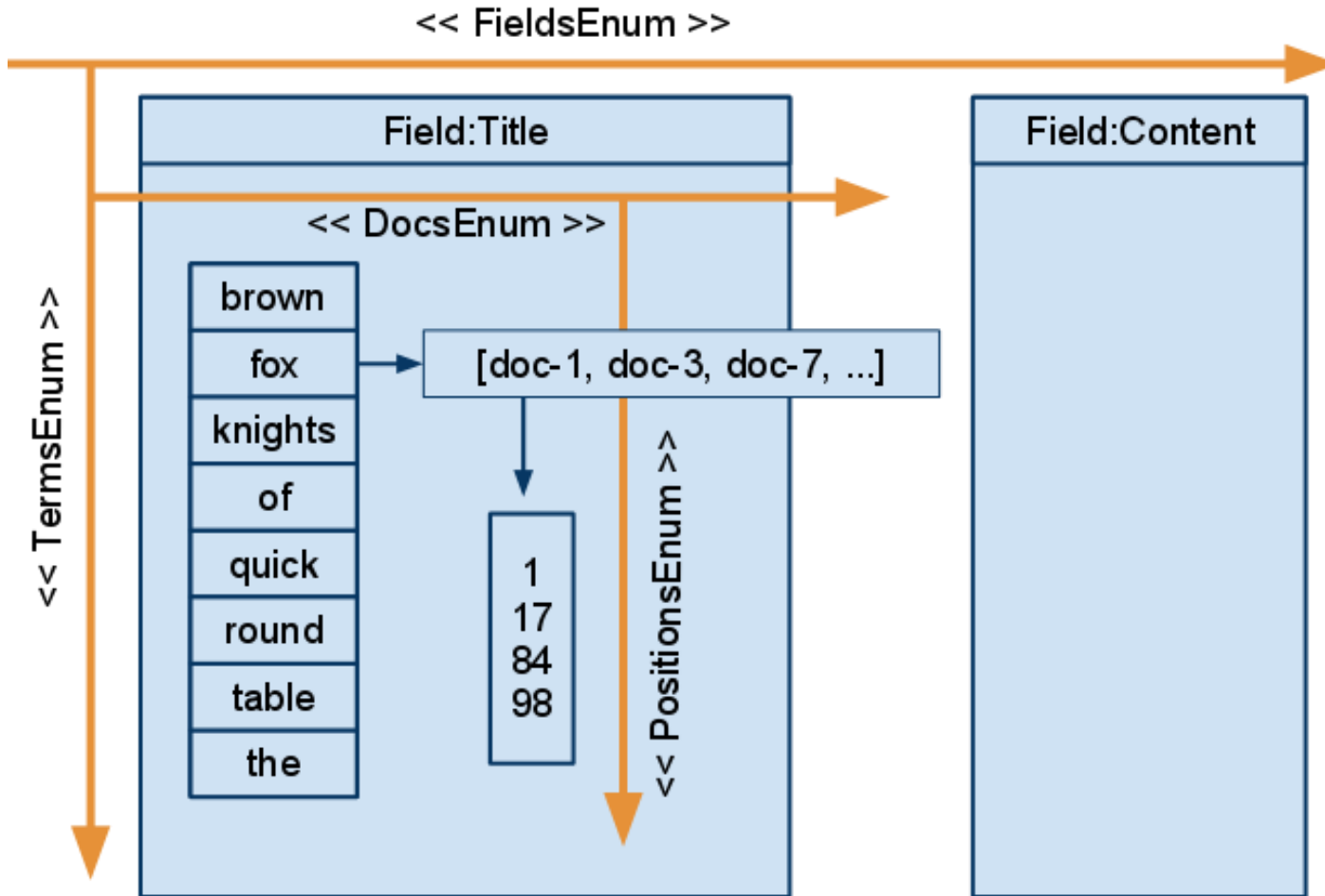
- **Pre-4.0** indexes are still **readable**, but **< 3.0 is no longer supported**

- Index upgrade tool is available since recently released **Lucene 3.2**
  - *Supports upgrade of pre-3.0 indexes **in-place** → two-step migration to 4.0* (LUCENE-3082)
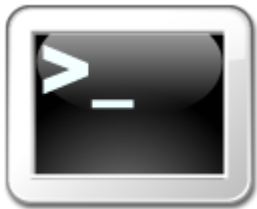
Architecture
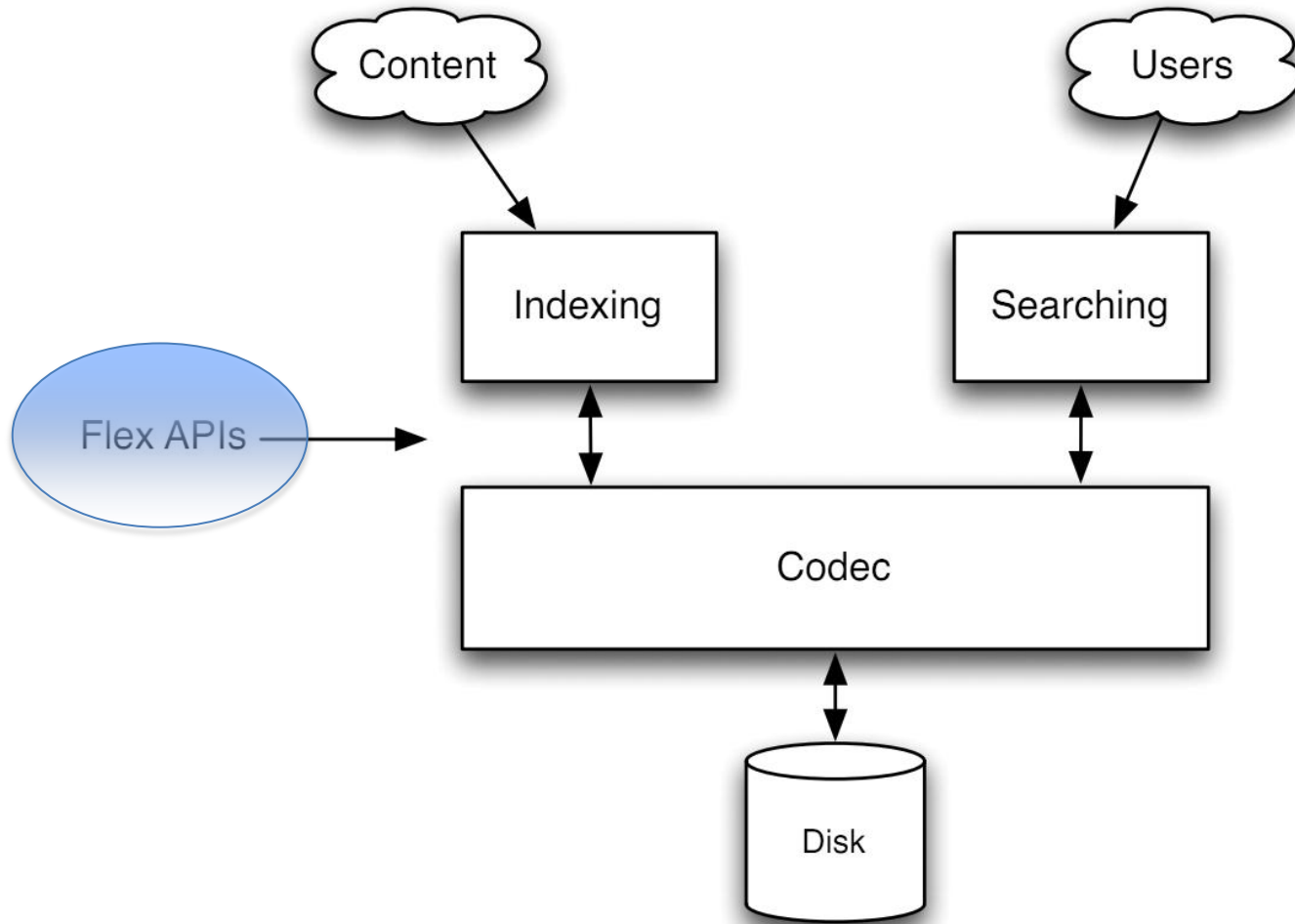
New 4-dimensional Enumeration-API

Flex - Enum API Properties (1)

- Replaces the `TermEnum` / `TermDocs` / `TermPositions`

- Unified iterator-like behavior: no longer strange `do...while` vs. `while`

- Decoupling of term from field
  - `IndexReader` returns enum of fields, each field has separate `TermsEnum`
  - No interning of field names needed anymore (will be removed soon)

- Improved RAM efficiency:
  - efficient re-usage of byte buffer with the `BytesRef` class

Flex - Enum API Properties (2)

- Terms are arbitrary byte slices, no 16bit UTF-16 chars anymore

- Term now using byte[] instead of char[]
  - compact representation of numeric terms

- Can be any encoding
  - Default is UTF-8 → changes sort order for supplementary characters
  - Compact representation with low memory footprint for western languages
  - Support for e.g. BOCU1 (LUCENE-1799) possible, but some queries rely on UTF-8 (especially `MultiTermQuery`)

- Indexer consumes `BytesRefAttribute`, which converts the good old `char[]` to UTF-8

Flex - Enum API Properties (3)

- All Flex Enums make use of `AttributeSource`
  - Custom Attribute deserialization *(planned)*
  - E.g., `BoostAttribute` for `FuzzyQuery`

- `TermsEnum` supports seeking *(enables fast automaton queries like `FuzzyQuery`)*

- `DocsEnum` / `DocsAndPositionsEnum` now support separate `skipDocs` property: Custom control of excluded documents *(default implementation by IndexReader returns deleted docs)*

FieldCache

- FieldCache consumes the flex APIs

- Terms (previously strings) field cache more RAM efficient, low GC load
  - Used with `SortField.STRING`

- Shared byte[] blocks instead of separate String instances
  - Terms remain as `byte[]` in few big `byte[]` slices

- `PackedInts` for ords & addresses (LUCENE-1990)

- RAM reduction ~ 40-60%

Agenda

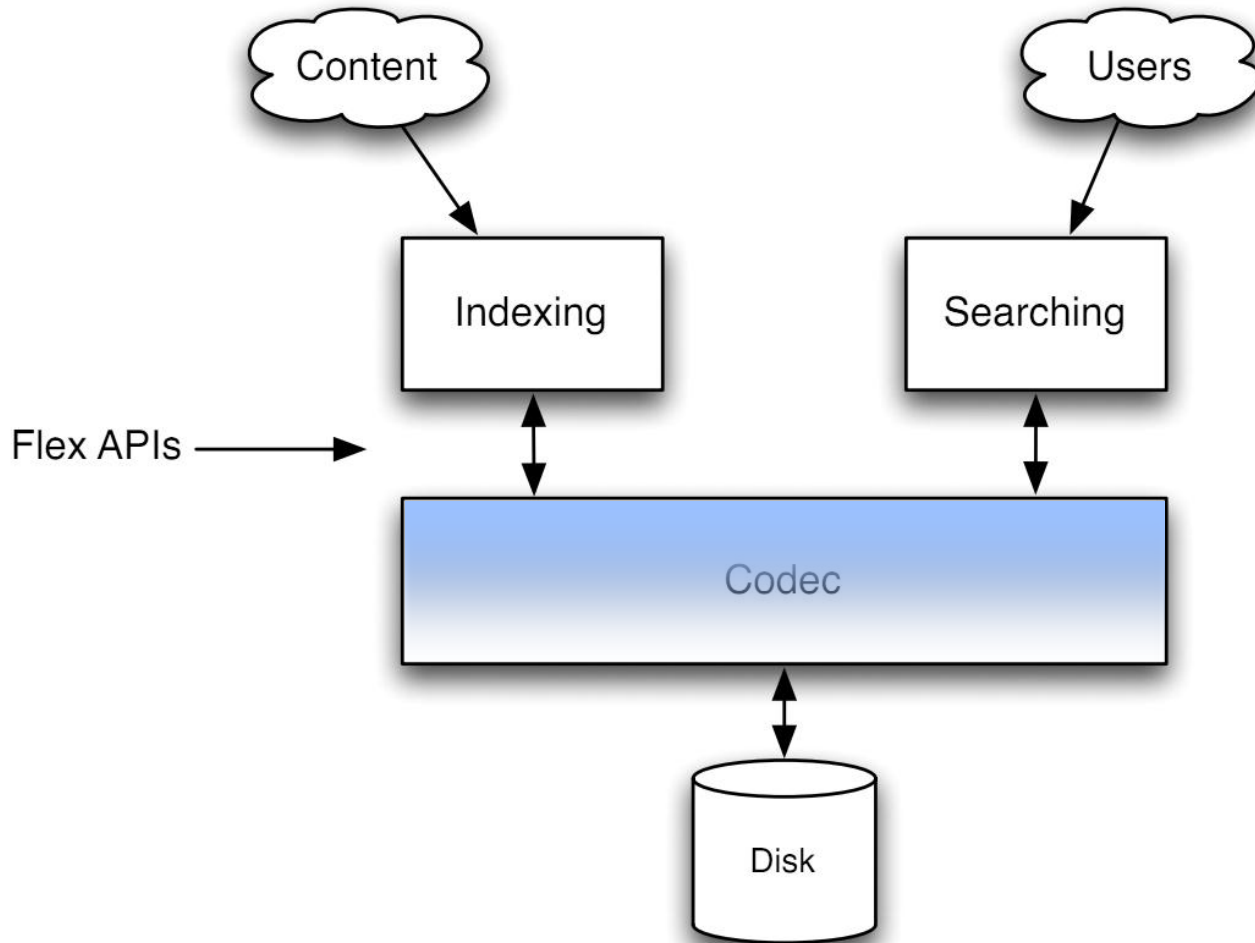- Motivation

- API changes in general

- Codecs

- Future

- Wrap up

Extending Flex with Codecs

- A Codec represents the primary Flex-API extension point

- Directly passed to `SegmentReader` to decode the index format

- Provides implementations of the enum classes to `SegmentReader`

- Provides writer for index files to `IndexWriter`

Architecture

Components of a Codec

- `Codec` **provides read/write for one segment**
  - Unique name (String)
  - FieldsConsumer (for writing)
  - FieldsProducer is 4D enum API + close

- `CodecProvider` **creates** `Codec` **instance**
  - **Passed to** `IndexWriter/IndexReader`
  - Supports per field codec configuration

- You can override merging

- Reusable building blocks
  - Terms dict + index, postings

Build-In Codecs: **PreFlex**

- Pre-Flex-Indexes will be "read-only" in Lucene 4.0

- all additions to indexes will be done with `CodecProvider`'s default codec

Warning: `PreFlexCodec` needs to reorder terms in term index on-the-fly: „surrogates dance"
→ use index upgrader (since Lucene 3.2, LUCENE-3082)

Default: **StandardCodec** (1)

Default codec was moved out of
`o.a.l.index`:

- lives now in its own package
  `o.a.l.index.codecs.standard`
- is the default implementation
- similar to the pre-flex index format
- requires far less ram to load term index

Default: StandardCodec (2)

- Terms index:
  `VariableGapTermsIndexWriter/Reader`
  - uses finite state transducers from new FST package
  - removes useless suffixes
  - reusable by other codecs

> Talk today, 13:30h:
> "Finite State Automata in Lucene"
> by Dawid Weiss

- Terms dict: `BlockTermsWriter/Reader`
  - more efficient metadata decode
  - faster TermsEnum
  - reusable by other codecs

- Postings: `StandardPostingsWriter/Reader`
 - Similar to pre-flex postings

Build-In Codecs: **PulsingCodec** (1)

- Inlines low doc-freq terms into terms dict

- Saves extra seek to get the postings

- Excellent match for primary key fields, but also "normal" field (Zipf's law)

- Wraps any other codec

- Likely default codec will use Pulsing

See Mike McCandless' blog:
http://s.apache.org/XEw

Build-In Codecs: PulsingCodec (2)

Pulsing Codec Speedup

Speedup (%) vs Number of random term lookups
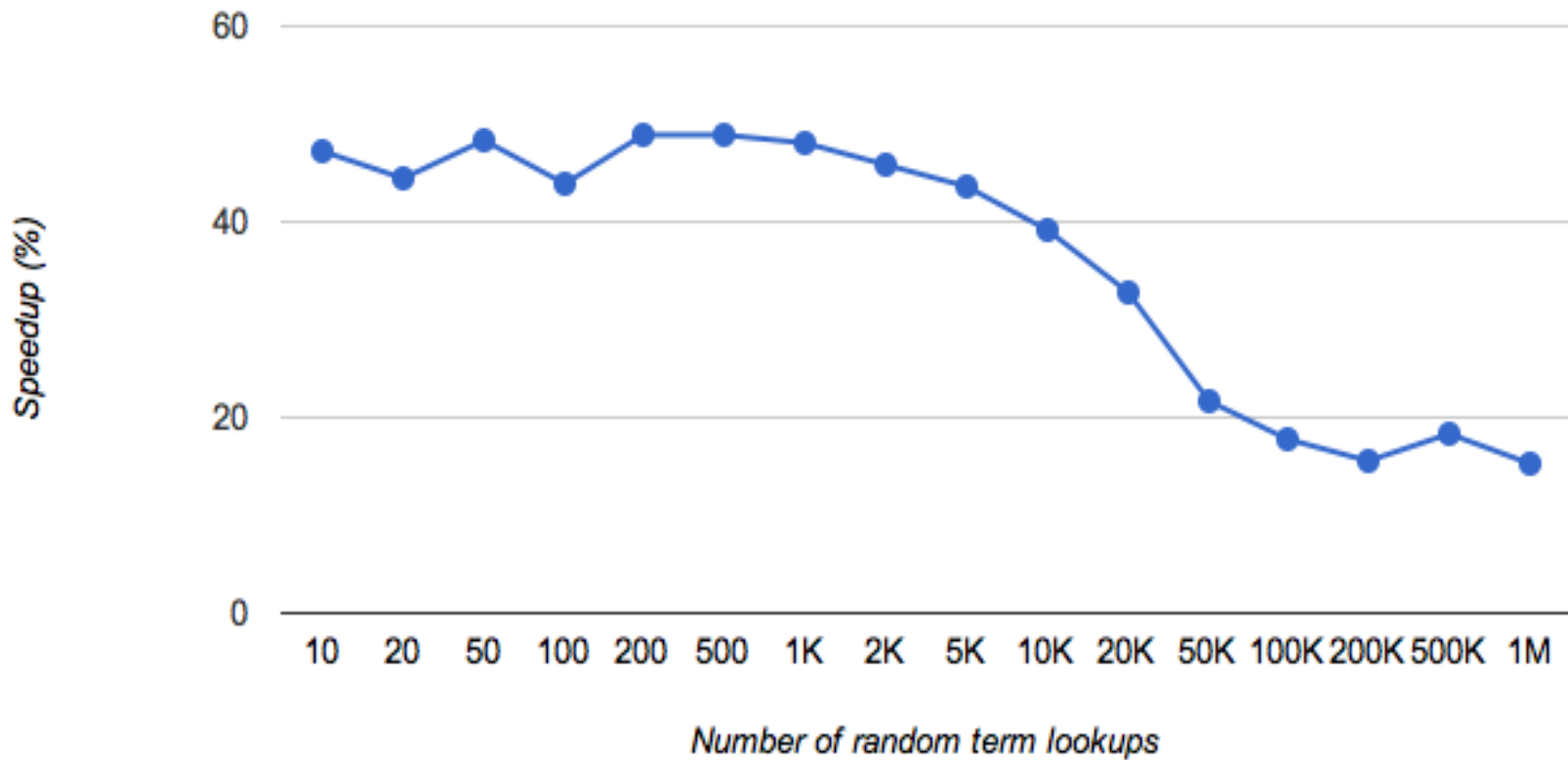
Build-In Codecs: SimpleText

```
field contents
  term file
    doc 0
      pos 5
  term is
    doc 0
      pos 1
  term second
    doc 0
      pos 3
  term test
    doc 0
      pos 4
  term the
    doc 0
      pos 2
  term this
    doc 0
      pos 0
END
```

- All postings stored in _X.pst text file

- Read / write

- Not performant
  – Do not use in production!

- Fully functional
  – Passes all Lucene/Solr unit tests (slowly…)

- Useful / fun for debugging

See Mike McCandless' blog:
http://s.apache.org/eh

Other Codecs shipped with Lucene Contrib / Tests

- `AppendingCodec` (contrib/misc)
  - Never rewinds a file pointer during write
  - Useful for Hadoop

- `MockRandomCodec` (test-framework)
  - used in tests to select random codec configurations

- `PreFlexRW` (test-framework)
  - Used by tests to build pre-flex indexes
  - Standard pre-flex codec is read-only

Heavy Development: Block Codecs

- Separate branch for developing block codecs:
  - http://svn.apache.org/repos/asf/lucene/dev/branches/**bulkpostings**
- All codecs use default terms index, but different encodings of postings
- Support for recent compression developments
  - FOR, PFORDelta, Simple64, BulkVInt
- Framework for blocks of encoded ints *(using* `SepCodec`*)*
  - Easy to plugin a new compression by providing a encoder/decoder between a block `int[]` ↔ `IndexOutput/IndexInput`

Block Codecs: Use Case

- Smaller index & better compression for certain index contents by replacing vInt
  - Frequent terms (stop words) have low doc deltas
  - Short fields (city names) have small position values

- `TermQuery`
  - High doc frequency terms

- `MultiTermQuery`
  - Lots of terms (possibly with low docFreq) in order request doc postings

Block Codecs: **Disadvantages**

- Crazy API for consumers, e.g. queries

- Removes encapsulation of DocsEnum
  - Consumer needs to do loading of blocks
  - Consumer needs to do doc delta decoding

- Lot's of queries may need duplicate code
  - Impl for codecs that support „bulk postings"
  - Classical DocsEnum impl

Agenda

- Motivation

- API changes in general

- Codecs

- Future

- Wrap up

Flexible Indexing - Current State

- All tests pass with a pool of random codecs
  - If you find a bug, post the test output to mailing list!
  - Test output contains random seed to reproduce

- many more tests and documentation is needed

- community feedback is highly appreciated

Flexible Indexing - TODO

- Serialize custom attributes to the index
- Convert all remaining queries to use internally `BytesRef`-terms
- Remove interning of field names
- Die, Term (& Token) class, die! ☺
- Make block-/bulk-postings API more user-friendly and merge to SVN trunk
- Merge DocValues branch

    (Next talk: "DocValues & Column Stride Fields in Lucene" by Simon Willnauer)

- More heavy refactoring!

Agenda

- Motivation

- API changes in general

- Codecs

- Future

- Wrap up

Summary

- New 4D postings enum APIs

- Pluggable codec lets you customize index format
  - Many codecs already available
- State-of-the-art postings formats are in progress

- Innovation is easier!
  - Exciting time for Lucene...
- Sizable performance gains, RAM/GC

We need YOU!

- Download nightly builds or check out SVN trunk

- Run tests, run tests, run tests,...

- Use in your own developments

- Report back experience with developing own codecs

Contact

Uwe Schindler

uschindler@sd-datasolutions.de

http://www.thetaphi.de

@thetaph1

*Many thanks to **Mike McCandless** for slides & help on preparing this talk!*

SD DataSolutions GmbH
Wätjenstr. 49
28213 Bremen, Germany
+49 421 40889785-0
http://www.sd-datasolutions.de

BERLINBUZZWORDS

~ THE CONFERENCE OF HIGH SCALABILITY ~