Lucene Forecast: Version, Unicode, Flex and Modules

Simon Willnauer & Uwe Schindler

Who we are

- Uwe Schindler (uschindler@apache.org)
  - Apache Lucene/Solr PMC Member and Committer. He implemented fast numerical search and is maintaining the new attribute-based text analysis API. Software architect and consultant for PANGAEA (Publishing Network for Geoscientific & Environmental Data) in Bremen, Germany.
- Simon Willnauer (simonw@apache.org)
  - Apache Lucene/Solr, OpenRelevance and Connectors Committer. Currently working as a freelancer on Search, Large Data Processing and Scalability topics. I'm a BerlinBuzzwords Co-organizer and located in Berlin.

What happens in the next 35 minutes?

- Current Community Developments
- Modularization
- Version – Tale of Backwards Compatibility
- Lucene, Java, Unicode
- State of the **Flex**

Two projects – One Codebase

- Merging Lucene and Solr development
  - Still two separate released “products”!!!
  - Share mailing list and code repository
  - Solr trunk code in sync with Lucene trunk code
- Benefits to both Lucene and Solr users
  - Lucene features exposed to Solr faster
  - Solr features available to Lucene users
  - Modules for common used components: one place for Analyzers, Tokenizers, TokenFilters

Lucene 3.1 vs. Lucene 4.0

- Lucene 3.1 aka "branch_3x":
  - Next stable release with Unicode 4.0 and supplementary character support in Lucene Core
  - Unicode 5.2 in contrib-icu using ICU 4.4, featuring rule-based tokenization (LUCENE-1343, LUCENE-2399, LUCENE-2409, LUCENE-2414 and others)
  - Full backwards compatibility using o.a.l.util.Version parameters to most Analyzers
- Lucene 4.0 aka "trunk" - Not Backwards-Compatible:
  - Flexible Indexing
  - Revised enumeration API for fields, terms, docs, positions
  - Binary terms
  - Attribute serialization support (unstructured payloads are gone)
  - Index conversion tool, as older indexes cannot be read anymore

Migration to new 4.0 version

- No longer a 3.9 version with all features (like flexible indexing), but also deprecated APIs and "sophisticated backwards layers" (like Attributes vs. Token in 2.9)
- If you want to move, upgrade your code first
- Binary index format changed, indexes can be converted to new format, **BUT:** Analyzer changes may require reindexing

Lucene / Solr Modularization

- Common used components are moved from Lucene and Solr into a shared place:
  - Lucene Core without analysis, only abstract TokenStream and Analyzer classes stay with a reduced set of Attributes
  - New analysis module containing TokenFilters, Tokenizers, Analyzers for various languages (moved out of Solr, Lucene Core and Lucene Contrib), lots of custom Attributes
  - Possibly separate JAR files for different language groups
- Solr's Facetting will be also available for Lucene-only use cases

Version - Tale of backwards compatibility

- A Released-Version constant passed to constructors
- Introduced in LUCENE-1684
- Already present in Lucene 2.9
  - Rarely used in released Lucene Versions
  - Extensively used in Lucene 3.1 branch
  - New configuration parameter in Solr's config and schema
  - Created to preserve Version by Version compatibility
- public StandardAnalyzer(Version matchVersion);

Version - Tale of backwards compatibility

Snippet from the StandardAnalyzer JavaDoc:

You must specify the required Version compatibility when creating StandardAnalyzer:

- As of 3.1, StopFilter correctly handles Unicode 4.0 supplementary characters in stopwords
- As of 2.9, StopFilter preserves position increments
- As of 2.4, Tokens incorrectly identified as acronyms are corrected (see LUCENE-1068)

Version - Tale of backwards compatibility

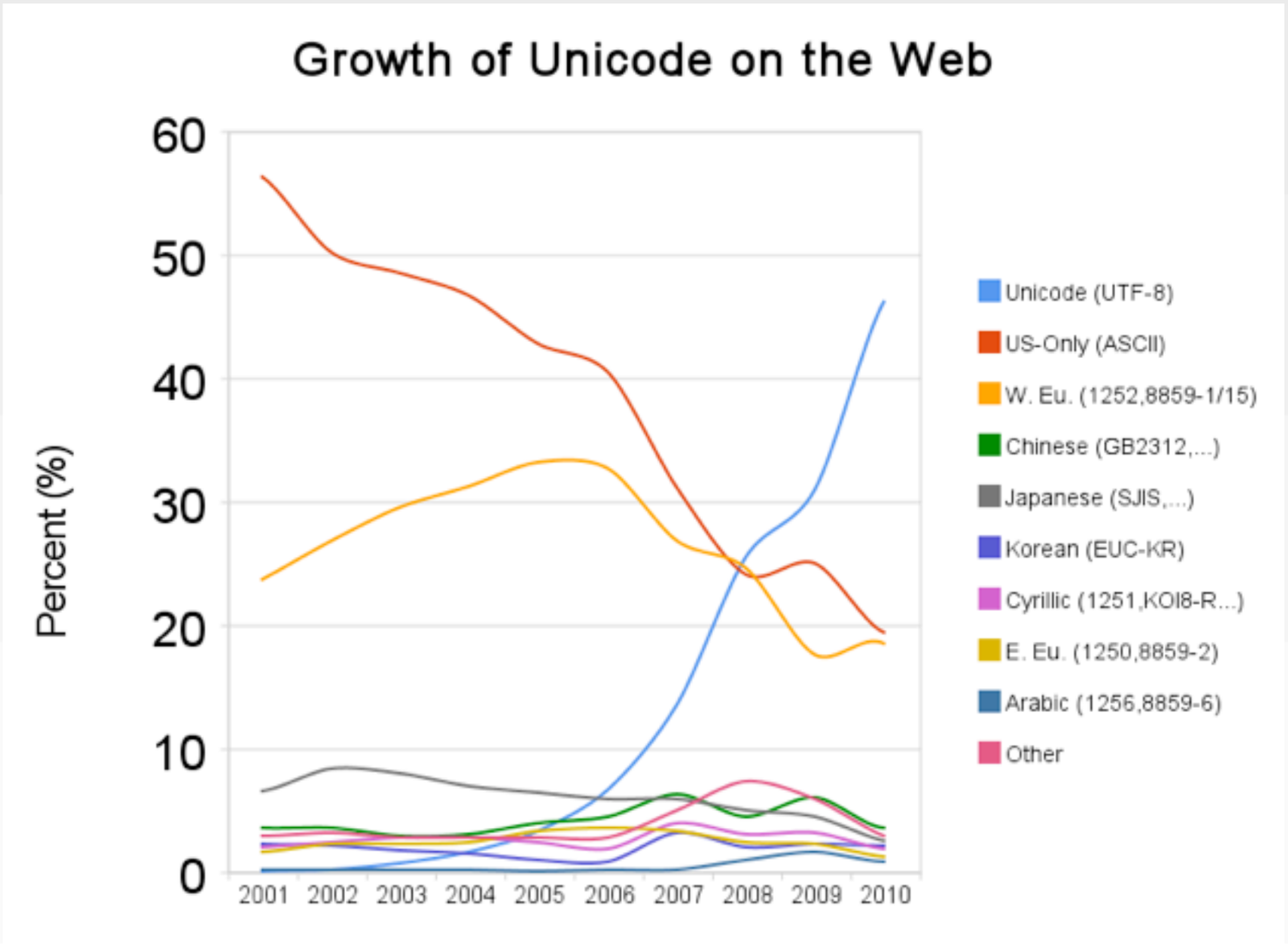
- Version constants trigger:
  - different runtime - behavior
  - different APIs
  - old buggy code :)
  - different defaults

Upgrade with Version

- Upgrades to newer Lucene Releases became easier!
  - re-indexing not absolutely necessary
  - old behavior can be preserved where necessary
  - custom code can be adopted incrementally
  - get the best or both worlds
    - use compatible improvements
    - stick to old behavior if changes are not compatible
- **Important:** Don't use **Version.LUCENE_CURRENT**, if you want to reuse your indexes with later Lucene versions!

Lucene, Java & Unicode

50% of the web uses Unicode



Limited support for Unicode in Lucene

- Bound to Java 1.4 until Lucene 2.9
  - Java 1.4 supported Unicode 3.0
  - `char` type was created as a 16-bit entity
  - each `char` represented a complete codepoint
  - Unicode 3 - 0x0000 through 0xFFFF

Unicode – Why should I care?

- The most of you wouldn't!
- Unless you need to index:
  - Japanese
  - Korean
  - Places in Hong Kong
  - Traditional Chinese
  - Mormon books
  - Ancient Greek
  - ...

Unicode 4.0 support since Java 1.5

- Unicode 4 – 0x0000 through 0x10FFFF
  - `char` is now a UTF-16 code unit, not a code point
  - Unicode code points are represented as an `int`
  - low-level APIs use `int` instead of `char`
  - high level APIs now respect surrogate pairs

Unicode – What is this all about?

Can you read Deseret?

Lucene 2.9s LowerCaseFilter can't!

LowerCaseFilter	Input	Output
Version.LUCENE_30	𐐷𐐲𐐻	𐐷𐐲𐐻
Version.LUCENE_31	𐐷𐐲𐐻	𐐯𐐲𐐻

Unicode – It is getting worse!

Try LetterTokenizer with: **"the '£' semuncia symbol"**

LetterTokenizer	Output
Version.LUCENE_30	"the", "semuncia", "symbol"
Version.LUCENE_31	"the", "£", "semuncia", "symbol"

Unicode – What did change?

- TokenFilter and Tokenizer take Version for compatibility
  - upgrading to 3.1 requires re-indexing in some cases
  - CharTokenizer uses Version to switch API
- I/O code is aware of 16 bit code units
  - buffer boundaries check for high / low surrogate pairs

Unicode – What did change? #2

- Most code is ported to handle supplementary characters correctly, but:
  - StandardTokenizer still not fixed, will be renamed to:

    SmartENWithSmartProductNumbersAndStupidURLDetectionWithPossesiveSMarkerTokenizer

    in the future

- "RevisedStandardTokenizer" in preparation that uses Unicode Standard Annex #29 (LUCENE-2167)

When will Lucene support Unicode 5.2?

Here you go!

- Lucene contrib contain new ICU based Analysis tools
  - ICU - Folding Filter (LUCENE-1343)
    - case-folding
    - accent-removal
  - ICU - Normalization Filter (LUCENE-2399)
    - Standard normalization modes
    - custom mappings
  - ICU - Transformation Filter (LUCENE-2409)
    - Conversion from Traditional to Simplified Chinese characters
    - Script conversions, for example Serbian Cyrillic to Latin
  - See: http://svn.apache.org/repos/asf/lucene/dev/trunk/lucene/contrib/icu

Unicode based segmentation

- ICUTokenizer finds boundaries between certain significant text elements: user-perceived characters, words, and sentences.
  - Recently added through LUCENE-2414
  - Defaults to Unicode Standard Annex #29
  - Thai (uses dictionary-based word breaking)
  - Khmer, Myanmar, Lao (uses custom rules for syllabification)
  - Details can be found here:
    - http://unicode.org/reports/tr29/
    - https://issues.apache.org/jira/browse/LUCENE-2414

ICUFoldingFilter + ICUTokenizer

“The Quick Brown Föx. Θε Κυικκ Βρουν Φοξ. ته قُكك بروُن فُكس. טהֶ קִקךְ ברֶון פֶּכס. เทกฺจกบโรวนโฟอ. Тхе Куицк Броун Фокс. てへ くいっく ぶろうん ふぉくす。 Տհե Qուիծկ Բրուն Ֆոխ.

Tჰე Qუიcქ Bროwნ Fოx.”

"the", "quick", "brown", "fox", "θε", "κυικκ", "βρουν", "φοξ", "ته", "قكك", "برون", "فكس", "טה", "קקך", "ברון", "פכס", "เท", "กฺ", "จกบ", "โร", "วนโฟอ", "тхе", "куицк", "броун", "фокс", "て", "へ", "く", "い", "っ", "く", "ぶ", "ろ", "う", "ん", "ふ", "ぉ", "く", "す", "տհե", "q", "ուիծկ", "բրուն", "ֆոխ", "t", "ჰე", "q", "უი", "c", "ქ", "b", "რო", "w", "ნ", "f", "ო", "x"

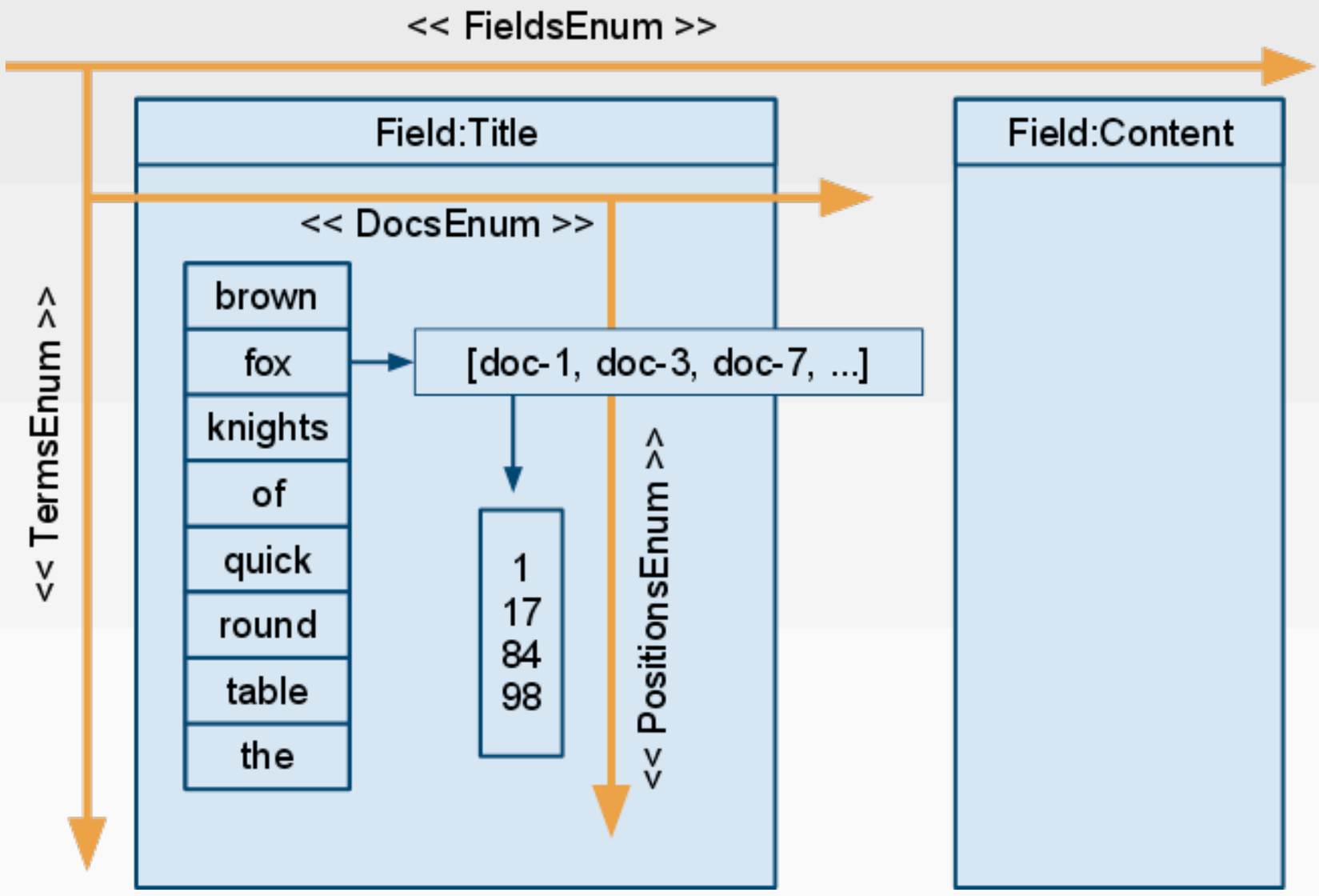
Friday, June 11, 2010

Flexible Indexing – aka. Flex – API

Flexible Indexing

- Targets to make Lucene extensible even on the lowest level
- Will be >= 4.0 ONLY!
- allows to
  - store new information into the index
  - change the way existing information is stored
- Under heavy development - No stable API yet!
- Replaces a lot or existing classes and interfaces

The new 4 Dimensional Enumeration-API



Flex – Enum API Properties

- Replaces the TermEnum / TermDocs / TermPositions
- Unified iterator-like behavior: no longer strange **do..while** vs. **while**
- Improved RAM efficiency
  - using `byte[ ]` instead of `char[ ]`
  - compact representation of Numeric-Terms (Trie) and ASCII chars (UTF-8 bytes)
  - efficient re-usage of byte buffer with the `BytesRef` – Class
- All Flex Enums make use of AttributeSource
  - Custom Attribute deserialization
  - BoostAttribute for Fuzzy Query

Flex Enums – an example

Pre-Flex	Flex

TermEnum enum = ...;
do {
  Term t = enum.term();
} while(enum.next());
	BytesRef termRef;
	Fields fields = ...;
	TermsEnum termsEnum = fields.terms("fieldname").iterator();
	AttributeSource attrSrc = termsEnum.attributes();
	BoostAttribute boostAttr = attrSrc.addAttribute(BoostAttribute.class)
	Term t = new Term("title");
	while ((termRef = termsEnum.next()) != null) {
	  Term t = t.createTerm(termRef.utf8ToString())
	  float termBoost = boostAttr.getBoost();
	}

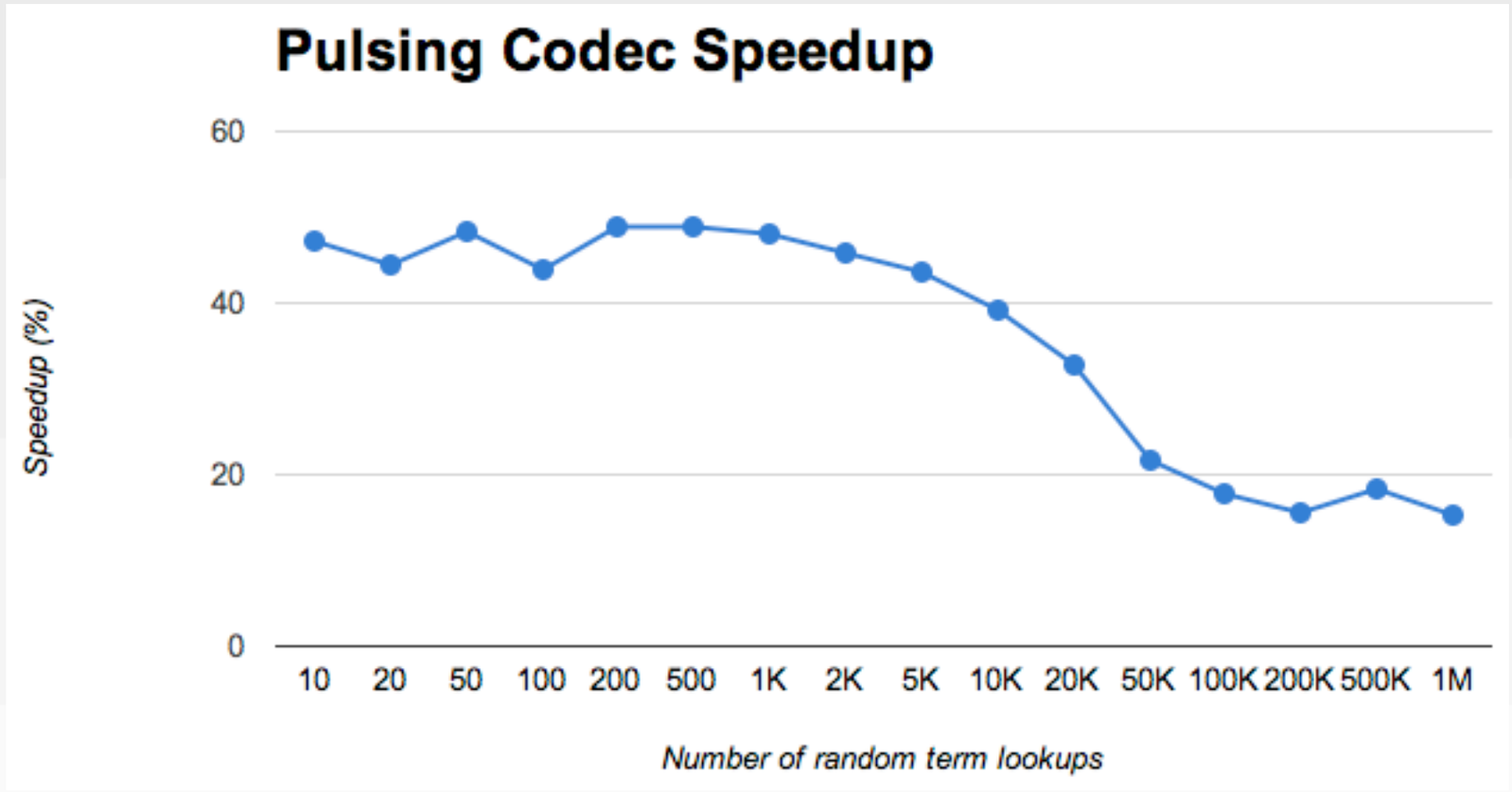
Extending Flex with Codecs

- A Codec represents the primary Flex-API extension point
- Directly passed to SegmentsReader to decode the index format
- Provides reader and writer for posting files

Flex Build-In Codecs

- Pre-Flex - Indexes will be Read Only with Lucene 4.0 (this codec is only needed for index conversion tool - slow!)
- Standard Index Codec is 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
- Additional codecs are in development (experimental)
  - PForDeltaCodec
  - PulsingCodec

Speedup with Pulsing-Codec



Flex – Current State

- Current State:
  - with StandardCodec all tests pass
  - many more tests and documentation is needed
  - community feedback is highly appreciated
- Future:
  - Serialize custom attributes to the index
  - More RAM savings
  - Improved index compression
  - Faster Near-Real-Time performance
  - Convert all remaining queries to use internally BytesRef-terms

Help Wanted!

- Flex API is still experimental
- Extensible APIs need to be implemented to improve!
- Low-Level Code including IO code is tricky
  - use different OS
  - use different file systems
- Spend time porting your existing applications to Flex and report back your:
  - experiences
  - bugs
  - speed improvements :)

Questions?

Thank you for your attention!