

Advanced Indexing Techniques with Lucene

Michael Busch

buschmi@{apache.org, us.ibm.com}



Advanced Indexing Techniques with Lucene

Agenda

- ▶ Introduction
 - Lucene's data structures 101
 - Payloads
 - Numeric Search
 - New TokenStream API
 - Outlook to Flexible Indexing



Introduction

What Lucene is really good at

- Simple and convenient APIs
- Incremental Indexing
- Managing files (consistency, merging, deletions)
- High performance
- Atomic commits with transactional semantics (commit, rollback, snapshots)

Wouldn't it be nice to utilize these features for custom data structures?



The (long) way to Flexible Indexing

- Lucene used to provide a single, fixed posting list format
- Payloads were introduced to provide a certain degree of flexibility
- Recently the option to avoid storing term positions was added
- Column-stride fields are being discussed and (hopefully) added soon
- Work on allowing custom encodings is ongoing



After this talk you should ...

- ... understand what an inverted index and a posting list are.
- ... understand the difference between stored fields and column-stride fields (and payloads).
- ... be able to use the new TokenStream API.
- ... know how to use the new TrieRange classes to accelerate numeric searches.
- ... be able to explore the new indexing chain.
- ... be excited about Lucene!



Advanced Indexing Techniques with Lucene

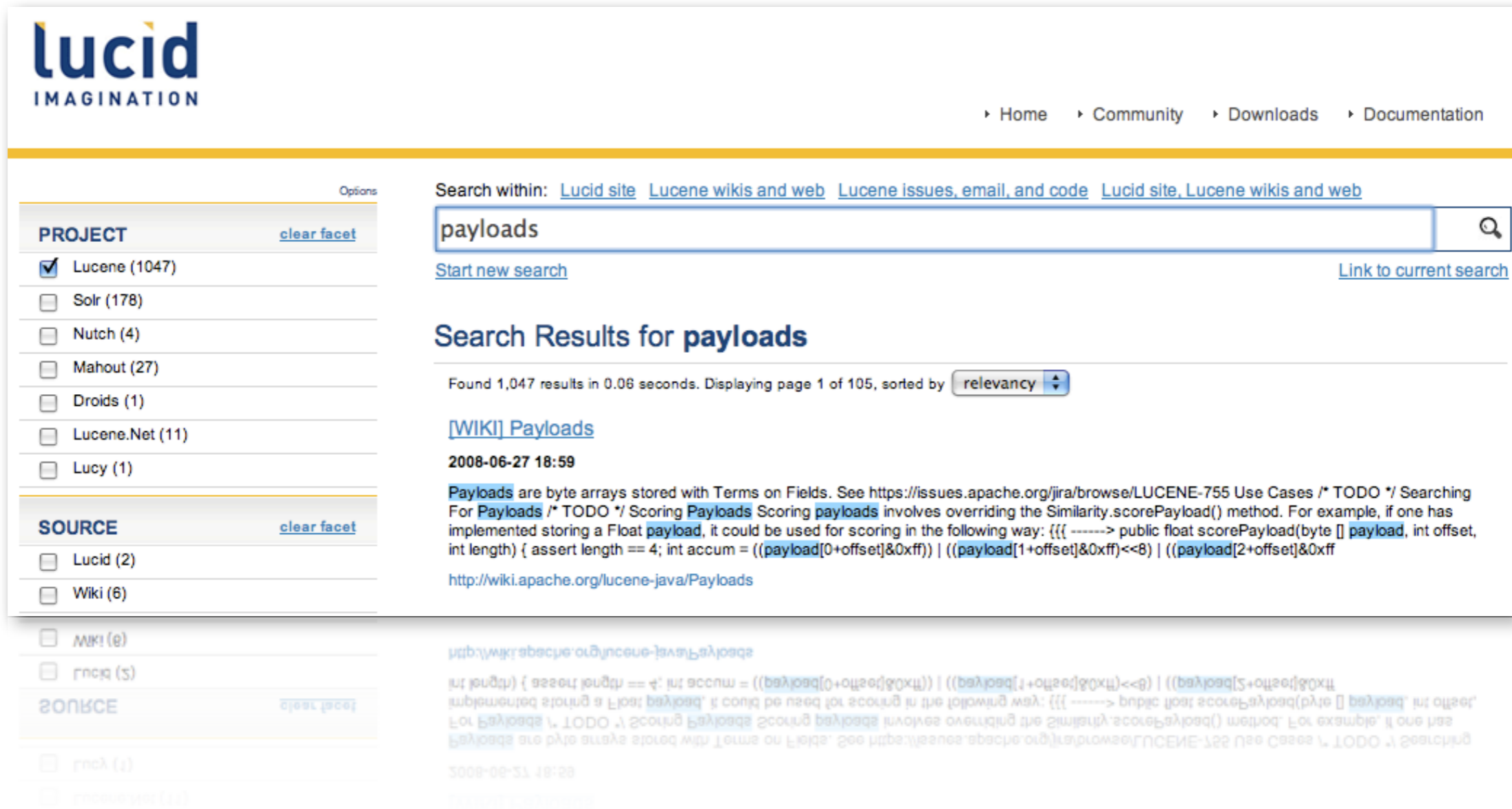
Agenda

- Introduction
- ▶ **Lucene's data structures 101**
- Payloads
- Numeric Search
- New TokenStream API
- Outlook to Flexible Indexing



Lucene's data structures 101

Store vs inverted index



The screenshot shows the Lucid search engine interface. The top left features the 'lucid IMAGINATION' logo. A navigation bar includes links for Home, Community, Downloads, and Documentation. A search bar contains the query 'payloads' and a search icon. Below the search bar, there are links for 'Start new search' and 'Link to current search'. On the left side, there are two facet panels: 'PROJECT' and 'SOURCE'. The 'PROJECT' panel has a 'clear facet' link and a list of projects with checkboxes: Lucene (1047) [checked], Solr (178), Nutch (4), Mahout (27), Droids (1), Lucene.Net (11), and Lucy (1). The 'SOURCE' panel also has a 'clear facet' link and a list of sources with checkboxes: Lucid (2), Wiki (6), MKI (6), and GnuD (5). The main search results area displays 'Search Results for payloads' and indicates 'Found 1,047 results in 0.06 seconds. Displaying page 1 of 105, sorted by relevancy'. A dropdown menu for sorting is set to 'relevancy'. The first result is titled '[WIKI] Payloads' with a timestamp of '2008-06-27 18:59'. The result text describes payloads as byte arrays stored with Terms on Fields and provides a code snippet for scoring payloads. The code snippet is:

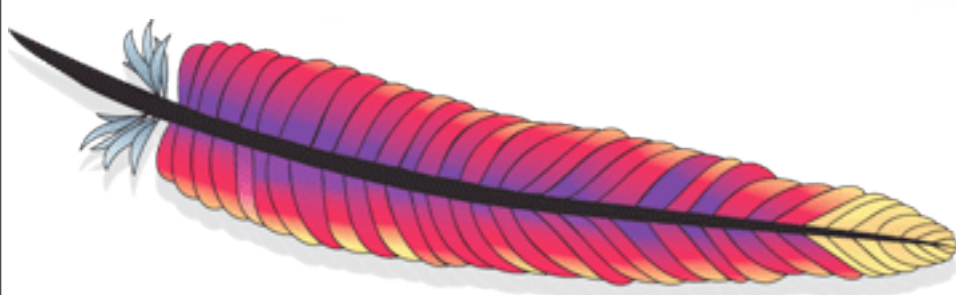
```
public float scorePayload(byte [] payload, int offset, int length) { assert length == 4; int accum = ((payload[0+offset]&0xff) | ((payload[1+offset]&0xff)<<8) | ((payload[2+offset]&0xff
```



Store vs inverted index

Inverted Index

The screenshot shows the Lucid search engine interface. At the top left is the logo "lucid IMAGINATION". A navigation bar contains links for Home, Community, Downloads, and Documentation. A search bar contains the text "payloads" and a search icon. Below the search bar, there are links for "Start new search" and "Link to current search". A red arrow points from the "Inverted Index" text box to the search bar. Below the search bar, the text "Search Results for payloads" is displayed. A summary box states: "Found 1,047 results in 0.06 seconds. Displaying page 1 of 105, sorted by relevancy". The results list includes a link to "[Wiki] Payloads" with a date of "2008-06-27 18:59". The snippet of the article text is visible, discussing how payloads are stored and scored in Lucene. On the left side, there are facet filters for PROJECT and SOURCE. The PROJECT filter shows "Lucene (1047)" selected, along with other projects like Solr, Nutch, Mahout, Droids, Lucene.Net, and Lucy. The SOURCE filter shows "Lucid (2)" and "Wiki (6)".

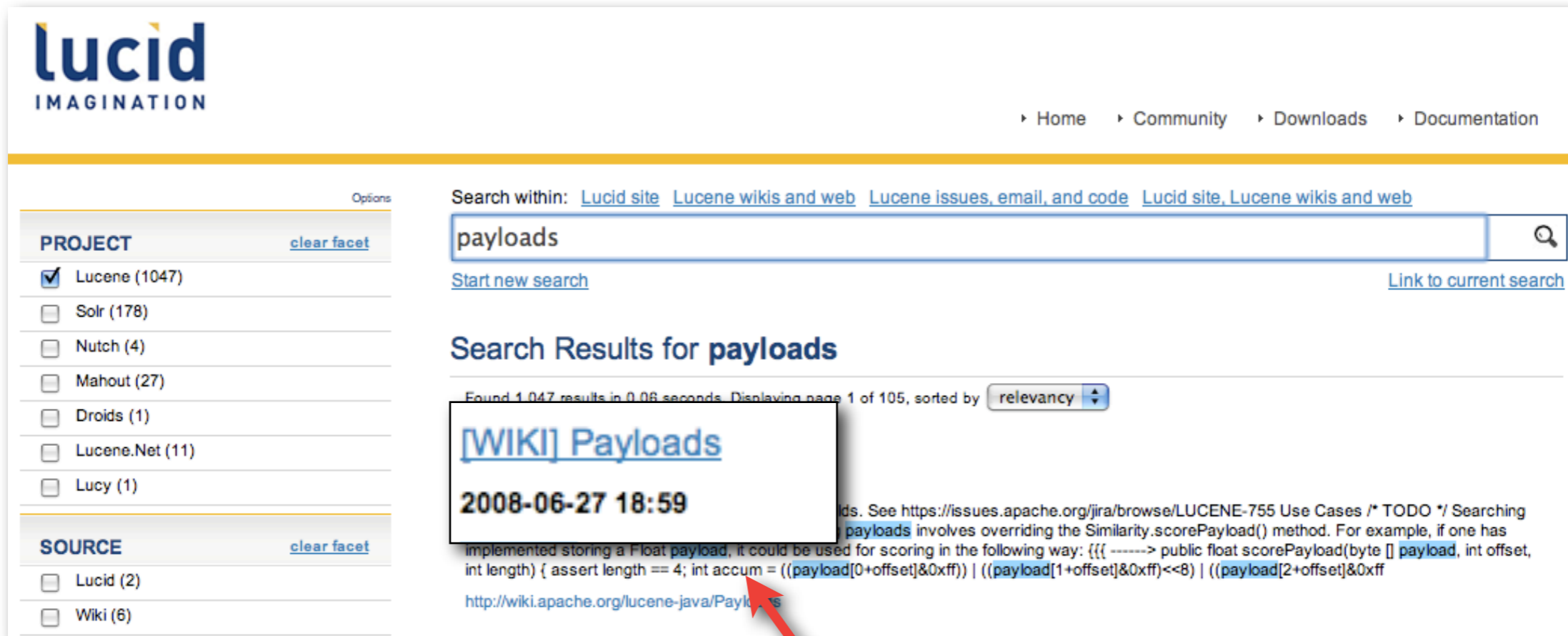


Store vs inverted index

The screenshot shows the Lucid search engine interface. At the top left is the logo "lucid IMAGINATION". To the right are navigation links: Home, Community, Downloads, and Documentation. Below the logo is a search bar with the text "payloads" and a search icon. To the left of the search bar is a facet menu with two sections: "PROJECT" and "SOURCE". The "PROJECT" section has a "clear facet" link and a list of projects with checkboxes: Lucene (1047) [checked], Solr (178), Nutch (4), Mahout (27), Droids (1), Lucene.Net (11), and Lucy (1). The "SOURCE" section also has a "clear facet" link and a list of sources with checkboxes: Lucid (2), Wiki (6), MKI (6), and GnuD (5). Below the search bar, there are links for "Start new search" and "Link to current search". The main content area shows "Search Results for payloads" with a sub-header "Found 1,047 results in 0.06 seconds. Displaying page 1 of 105, sorted by relevancy". Below this is a link to "[WIKI] Payloads" and a timestamp "2008-06-27 18:59". The main text of the result is a snippet from a wiki page, starting with "Payloads are byte arrays stored with Terms on Fields. See https://issues.apache.org/jira/browse/LUCENE-755 Use Cases /* TODO */ Searching For Payloads /* TODO */ Scoring Payloads Scoring payloads involves overriding the Similarity.scorePayload() method. For example, if one has implemented storing a Float payload, it could be used for scoring in the following way: {{{ -----> public float scorePayload(byte [] payload, int offset, int length) { assert length == 4; int accum = ((payload[0+offset]&0xff) | ((payload[1+offset]&0xff)<<8) | ((payload[2+offset]&0xff" and ending with "http://wiki.apache.org/lucene-java/Payloads".



Store vs inverted index



The screenshot shows the Lucid search engine interface. The search query is "payloads". The search results are sorted by relevancy. The top result is "[WIKI] Payloads" with a timestamp of "2008-06-27 18:59". A red arrow points from a green box labeled "Store" to the search results.

Lucid IMAGINATION

Home Community Downloads Documentation

Search within: Lucid site Lucene wikis and web Lucene issues, email, and code Lucid site, Lucene wikis and web

payloads

Start new search Link to current search

Search Results for **payloads**

Found 1,047 results in 0.06 seconds. Displaying page 1 of 105, sorted by relevancy

[WIKI] Payloads

2008-06-27 18:59

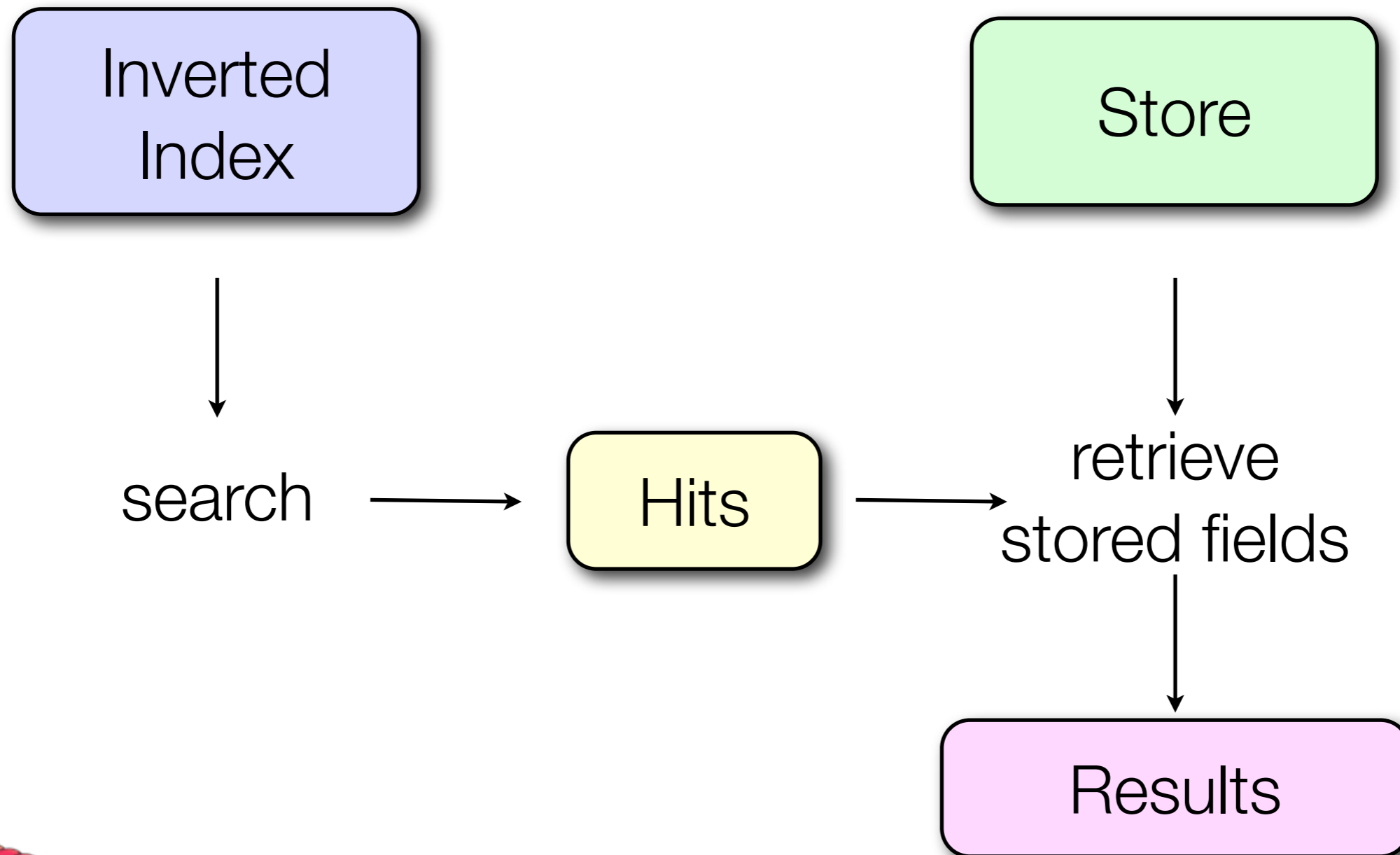
See <https://issues.apache.org/jira/browse/LUCENE-755> Use Cases /* TODO */ Searching **payloads** involves overriding the Similarity.scorePayload() method. For example, if one has implemented storing a Float **payload**, it could be used for scoring in the following way: {{{ -----> public float scorePayload(byte [] payload, int offset, int length) { assert length == 4; int accum = ((payload[0+offset]&0xff)) | ((payload[1+offset]&0xff)<<8) | ((payload[2+offset]&0xff

<http://wiki.apache.org/lucene-java/Payloads>

Store



Lucene's data structures



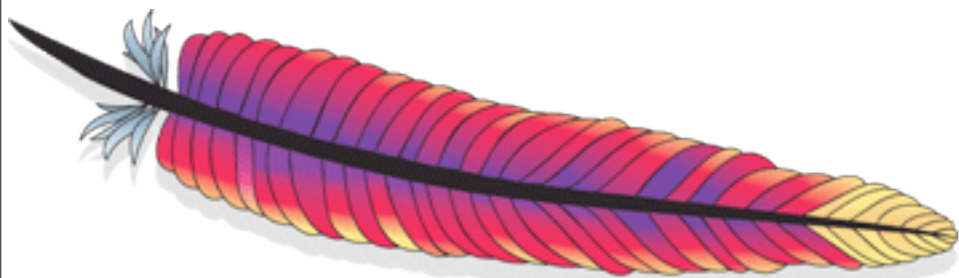
Inverted Index

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.

Table with 6 documents

Example from:

*Justin Zobel , Alistair Moffat,
Inverted files for text search engines,
ACM Computing Surveys (CSUR)
v.38 n.2, p.6-es, 2006*



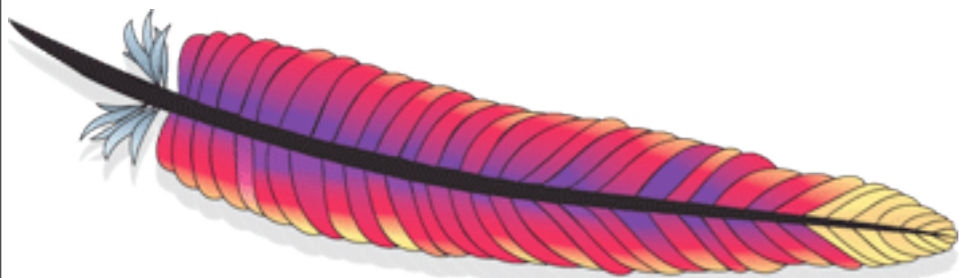
Inverted Index

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.

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: keeper

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.

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: keeper

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.

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: “in the night”

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.

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: "in the night"

1	The old night keeper keeps the keep in the town
2	In the
3	The ho
4	Where t
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

1 should not be a match
- we need more data!

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: “in the night”

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 <u>1</u> night <u>2</u> keeper <u>3</u> keeps <u>4</u> the <u>5</u> keep <u>6</u> in <u>7</u> the <u>8</u> night <u>9</u>
6	And keeps in the dark and sleeps in the light.

Table with 6 documents

term	freq	
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>

Dictionary and posting lists



Inverted Index

Query: "in the night"

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 <u>1</u> night <u>2</u> keeper <u>3</u> keeps <u>4</u> the <u>5</u> keep <u>6</u> in <u>7</u> the <u>8</u> night <u>9</u>
6	And keeps in the dark and sleeps in the light.

term	freq	
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>

Table with 6 documents

term	freq	Posting lists with term positions
in	5	<1 [8]> <2 [1, 6]> <3 [3]> <5 [7]> <6 [3, 8]>
...		
night	3	<1 [3]> <4 [4]> <5 [2, 9]>
...		
the	6	<1 [1, 6, 9]> <2 [2, 7]> <3 [1, 4, 7]> <4 [2]> <5 [1, 5, 8]> <6 [4, 9]>

sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists



Inverted Index

Query: "in the night"

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.

term	freq	
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>

Table with 6 documents

term	freq	Posting lists with term positions
in	5	<1 [8]> <2 [1, 6]> <3 [3]> <5 [7]> <6 [3, 8]>
...		
night	3	<1 [3]> <4 [4]> <5 [2, 9]>
...		
the	6	<1 [1, 6, 9]> <2 [2, 7]> <3 [1, 4, 7]> <4 [2]> <5 [1, 5, 8]> <6 [4, 9]>

sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists



Inverted Index

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

term	freq	
and	1	<6>
big	2	<2> <3>
dark	1	<6>
did	1	<4>
gown	1	<2>
in	5	<1> <2> <3> <5> <6>
keep	3	<1> <3> <5>
keeper	3	<1> <4> <5>
never	1	<3>
old	3	<2> <3>
sleeps	1	<4>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Boost this occurrence of 'night'

Table with 6 documents

term	freq	Posting lists with term positions
in	5	<1 [8]> <2 [1, 6]> <3 [3]> <5 [7]> <6 [3, 8]>
...		
night	3	<1 [3]> <4 [4]> <5 [2, 9]>
...		
the	6	<1 [1, 6, 9]> <2 [2, 7]> <3 [1, 4, 7]> <4 [2]> <5 [1, 5, 8]> <6 [4, 9]>

sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists



Inverted Index

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

term	freq	
and	1	<6>
big	2	<2> <3>
dark	1	<6>
did	1	<4>
gown	1	<2>
in	5	<1> <2> <3> <5> <6>
keep	3	<1> <3> <5>
keeper	3	<1> <4> <5>

Boost this occurrence of 'night'

Table with 6 documents

term	freq	Posting lists with term positions and payloads
in	5	<1 [8]> <2 [1, 6]> <3 [3]> <5 [7]> <6 [3, 8]>
...		
night	3	<1 [3]> <4 [4 (P)]> <5 [2, 9]>
...		
the	6	<1 [1, 6, 9]> <2 [2, 7]> <3 [5, 8]>

Payload
arbitrary byte array associated with a term position

where	1	<4>
-------	---	-----

Dictionary and posting lists

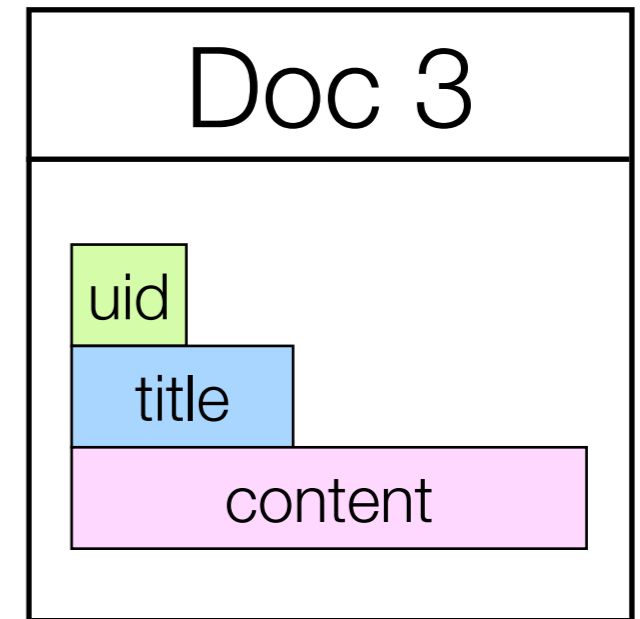
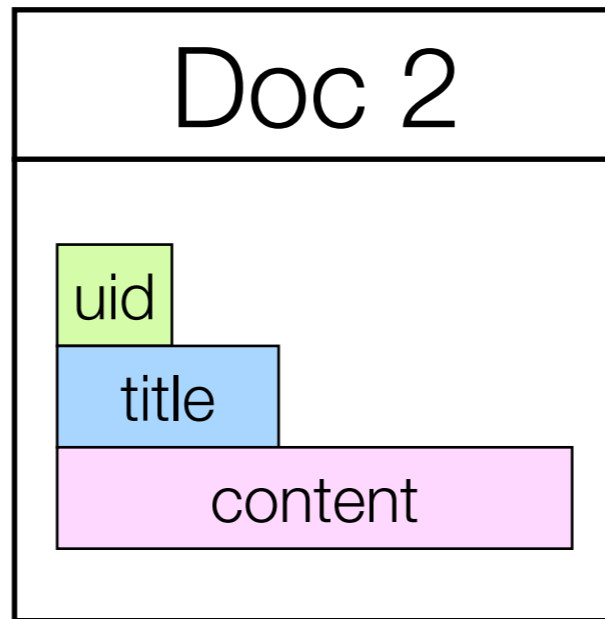
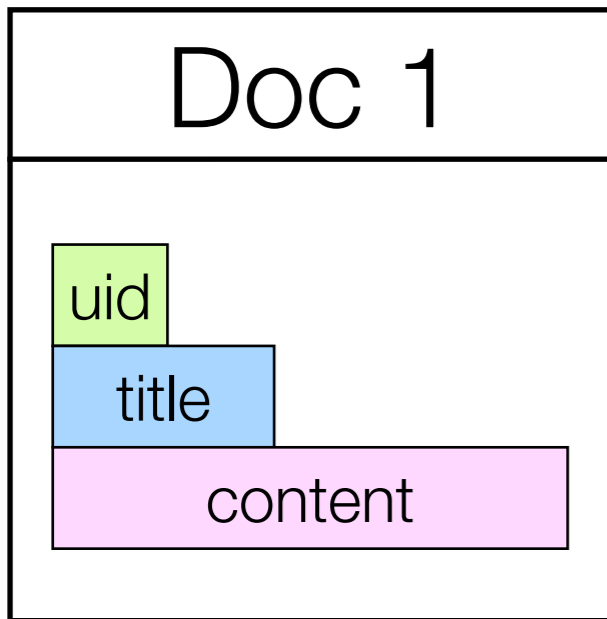


New feature: Boolean postings

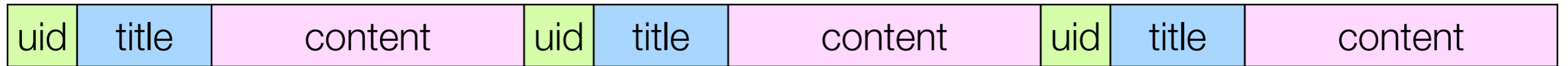
- Allows storing posting lists w/o positions for better decoding speed and space efficiency
- Usage: `field.setOmitTf(true);`
- Will be renamed before the 2.9 release
- Use for content that doesn't have position information, such as dates, email metadata (e.g. sender), etc.



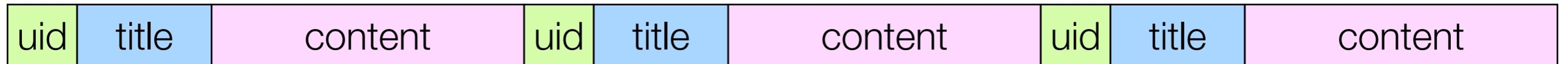
Stored fields vs. Payloads (column-stride fields)



Stored fields



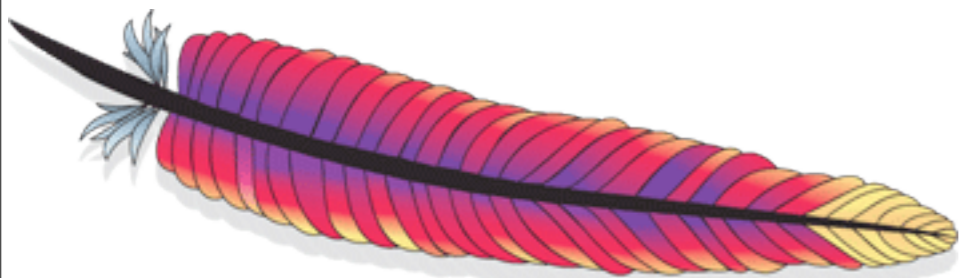
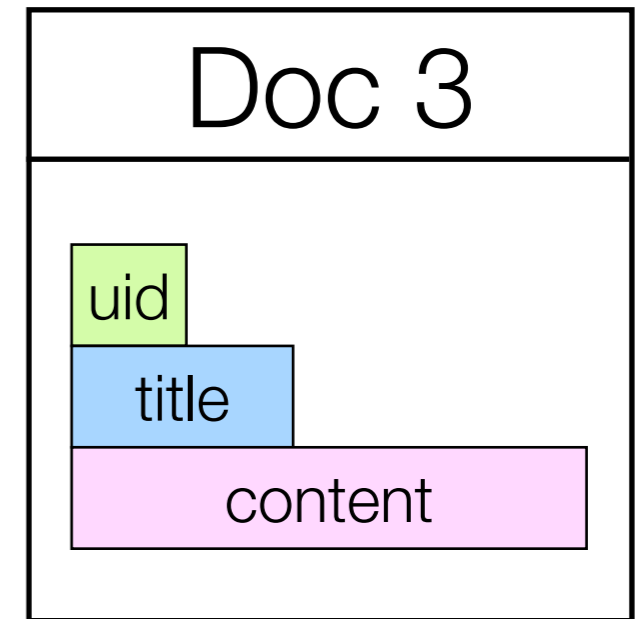
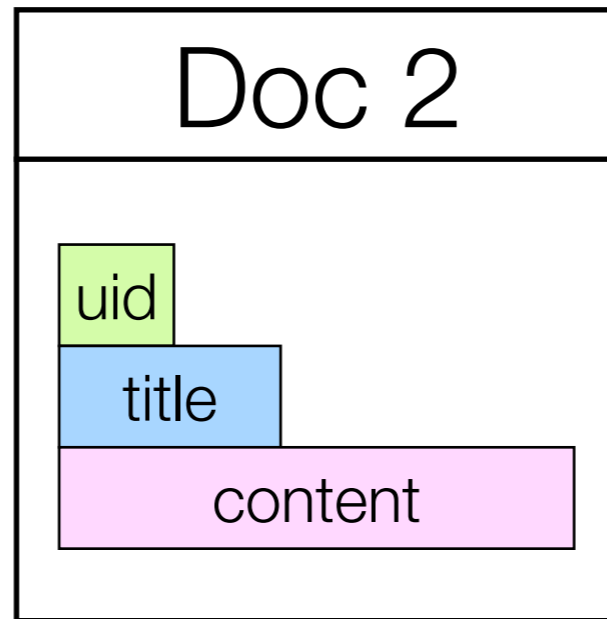
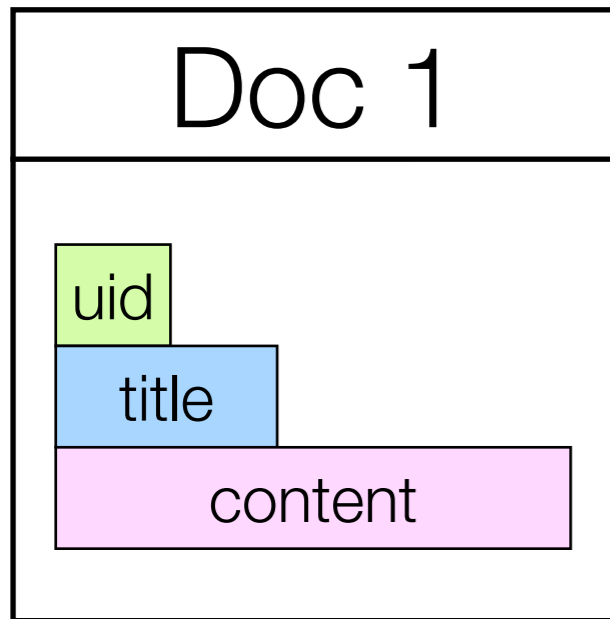
Stored fields



- Optimized for random access: loading all stored fields for a document is fast
- However, to read all 'uid' values from a large number of documents is slow, because the 'title' and 'content' fields have to be skipped



Stored fields vs. Payloads (column-stride fields)



Payloads and column-stride fields



Payloads and column-stride fields



- Optimized for scanning and skipping
- Should be used for fields that are needed during hit collection, so probably in this example only 'uid' would be a column-stride field
- Posting lists with payloads are very similar to column-stride fields - but they have overhead for encoding term positions
- Column-stride fields will probably improve performance compared to payloads by a factor of 2



Advanced Indexing Techniques with Lucene

Agenda

- Introduction
- Lucene's data structures 101
- ▶ **Payloads**
- Numeric Search
- New TokenStream API
- Outlook to Flexible Indexing



Payloads

Payloads - API

- org.apache.lucene.analysis.tokenattributes.PayloadAttribute

```
void setPayload(Payload payload);
```

- org.apache.lucene.index.Payload

```
Payload(byte[] data);
```

```
Payload(byte[] data,  
        int    offset,  
        int    length);
```



Payloads - API

- org.apache.lucene.index.TermPositions

```
boolean next();
```

```
int freq();
```

```
int nextPosition();
```

```
boolean isPayloadAvailable();
```

```
int getPayloadLength();
```

```
byte[] getPayload(byte[] data, int offset);
```



Example: BoostingTermQuery

Use case:

- Score certain occurrences of a term higher than others



Example: BoostingTermQuery

- TokenFilter:

```
final byte BoldBoost = 5;
...
boolean incrementToken() {
    if (!input.incrementToken()) return false;
    if (isBold) {
        payloadAttribute.setPayload(
            new Payload(new byte[] {BoldBoost}));
    }
    return true;
}
```



Example: BoostingTermQuery

- Similarity:

```
Similarity boostingSimilarity =
  new DefaultSimilarity() {
    // @override
    public float scorePayload(byte[] payload,
                              int offset, int length) {
      if (length == 1) {
        return byteToFloat(payload[offset]);
      }
    }
  };
```



Example: BoostingTermQuery

- BoostingTermQuery:

```
Query btq = new BoostingTermQuery(  
    new Term("field", "searchterm"));
```

- Searching:

```
Searcher searcher = new IndexSearcher(...);  
searcher.setSimilarity(boostingSimilarity);  
...  
TopScoreDocCollector c = new TopScoreDocCollector();  
searcher.search(btq, c);  
ScoreDoc[] hits = collector.topDocs().scoreDocs;
```



Example: Unique doc ids

Use case:

- Store a unique document id (UID) that e.g. maps to a row in a database table
- Retrieve UID at search time to influence matching/scoring
- FieldCache takes too long to load



Example: Unique doc ids

Solution:

- Index one special term for each document, e.g. ID:UID
- Index **one** occurrence for each document
- Store UID in the Payload of the occurrence

This is a workaround until we have column-stride fields.



Example: Unique doc ids

Performance for loading UIDs for 2M docs into memory:

- FieldCache: 16.5 s
- Payloads: 430 ms



Advanced Indexing Techniques with Lucene

Agenda

- Introduction
- Lucene's data structures 101
- Payloads
- ▶ **Numeric Search**
- New TokenStream API
- Outlook to Flexible Indexing



Numeric Search

New in Lucene: TrieRange

- Accelerate most numeric range queries by orders of magnitudes
- Initially developed for geospatial search by Uwe Schindler
- Idea: Store the numeric values multiple times in different precisions
- Intelligently select the least amount of terms to process for a given range query



New in Lucene: TrieRange

- Example: dates (the “real” TrieRange works on 32 or 64 bit numbers and computes terms by performing fixed-size bit shifts [=”precisionStep”])
- Normally Lucene indexes one term (and posting list) for each numeric value
- E.g. if you have documents from 2005 to 2007, then you will have 3×365 terms in the inverted index
- A query [02/15/05 TO 11/15/07] needs to visit roughly 1000 terms



New in Lucene: TrieRange

- With TrieRange we store additional terms in the index
- E.g. 'Jan 05', 'Feb 05' - 'Dec 07'; '2005', '2006', '2007'
- Query [02/15/05 TO 11/15/07] can now has only to visit these terms:
 - '02/15/05'-'02/28/05'; '11/01/07'-'11/15/07' (~30 terms)
 - 'Mar 05'-'Dec 05'; 'Jan 07'-'Oct 07' (~20 terms)
 - '2006' (1 term)
- TrieRange processes about 50 terms (vs. ~1000 before)



New in Lucene: TrieRange

- TrieRange is in the contrib/queries folder
- Good overview: [org/apache/lucene/search/trie/package.html](http://org.apache.lucene/search/trie/package.html)
- Simple usage:

```
long lvalue = 121345L;  
TrieUtils.addIndexedFields(doc, "exampleLong",  
                           TrieUtils.trieCodeLong(lvalue, precisionStep));
```

```
Query q = new LongTrieRangeFilter("exampleLong", precisionStep,  
                                 123L, 999999L, true, true).asQuery();
```



Advanced Indexing Techniques with Lucene

Agenda

- Introduction
- Lucene's data structures 101
- Payloads
- Numeric Search
- ▶ **New TokenStream API**
- Outlook to Flexible Indexing



New TokenStream API

TokenStream history

- TokenStream used to define this method:

```
public abstract class TokenStream {  
    public Token next();  
}
```

- Performance bottleneck: for each token of a document a new instance of Token was created; solution: re-usage of single Token instance:

```
public abstract class TokenStream {  
    public Token next(Token reusableToken);  
}
```



TokenStream history

- Token has a fixed number of members:
 - TermText, Type, Offset, PositionIncrement, Payload, Flags
- Problem: hard to add additional data to a Token



New TokenStream API

- New class that holds different (e.g. custom) attributes:

```
public class AttributeSource {  
    public <T extends Attribute> T addAttribute(  
        Class<T> attClass);  
}
```

- TokenStream extends AttributeSource and uses Attributes to stream data instead of the now deprecated Token:

```
public abstract class TokenStream  
    extends AttributeSource {  
    public boolean incrementToken();  
}
```



Example: PartOfSpeech tagging

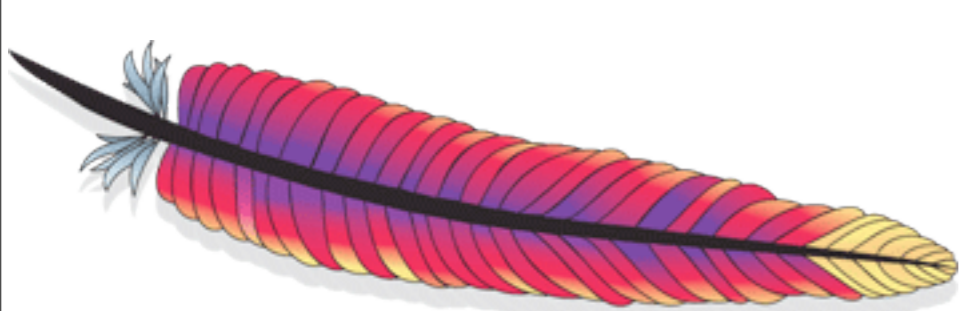
1. Define custom Attribute

```
public class POSAttribute extends Attribute {
    public static enum PartOfSpeech {
        ProperNoun, Noun, Verb, Adverb, Adjective
    }

    private PartOfSpeech pos;

    public void setPartOfSpeech(PartOfSpeech pos) {
        this.pos = pos;
    }

    public PartOfSpeech getPartOfSpeech() { return pos; }
}
```



Example: PartOfSpeech tagging

2. POSTokenFilter that tags tokens:

```
public class POSTokenFilter extends TokenFilter {
    private PartOfSpeechAttribute posAtt;
    private TermAttribute          termAtt;

    POSTokenFilter(TokenFilter in) {
        super(in);
        posAtt = addAttribute(PartOfSpeechAttribute.class);
        termAtt = addAttribute(TermAttribute.class);
    }
}
```



Example: PartOfSpeech tagging

2. POSTokenFilter that tags tokens:

```
public boolean incrementToken() throws IOException {
    if (!input.incrementToken()) return false;

    PartOfSpeech pos = determinePOS(termAtt);
    posAtt.setPartOfSpeech(pos);
    return true;
}
}
```



New TokenStream API

- Max. one single instance of one Attribute class
- Attribute instances are reused for all tokens of a document; no downcasting in `incrementToken()` necessary
- Default and custom attributes can be added to the TokenStream
- TokenFilters and consumers must keep a local reference to all attributes they're using and update its' values in `TokenStream#incrementToken()`



Advanced Indexing Techniques with Lucene

Agenda

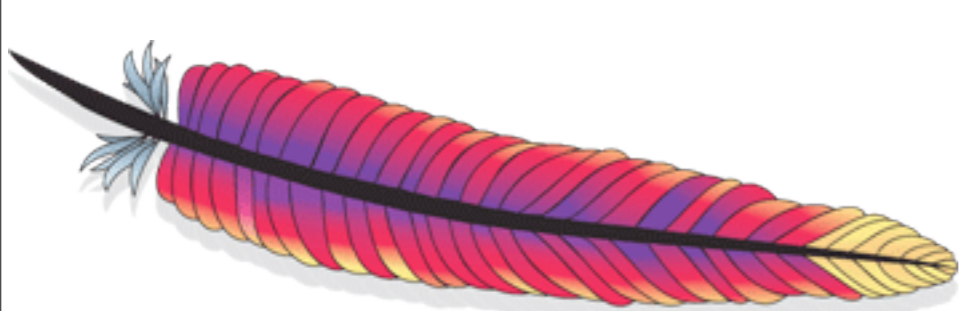
- Introduction
- Lucene's data structures 101
- Payloads
- Numeric Search
- New TokenStream API
- ▶ Outlook to Flexible Indexing



Outlook to Flexible Indexing

New indexing chain

- DocumentsWriter split up into several classes
- Consumer model
- Different consumers can be plugged into the indexing chain
- Currently: factoring out codecs for the different data structures (LUCENE-1458)
- Consumers, Codecs and new TokenStream API will make indexing flexible and extendable
- Column-stride fields will most likely be implemented as a codec



Example: PartOfSpeech tagging

3. Implement consumer that uses PartOfSpeechAttribute

```
void writeProx(FreqProxTermsWriter.PostingList p,  
              int proxCode) {  
    if (payload != null && payload.length > 0) {  
        termsHashPerField.writeVInt(1, (proxCode<<1)|1);  
        termsHashPerField.writeVInt(1, payload.length);  
        termsHashPerField.writeBytes(1, payload.data,  
                                     payload.offset, payload.length);  
        hasPayloads = true;  
    } else {  
        termsHashPerField.writeVInt(1, proxCode<<1);  
    }  
    p.lastPosition = fieldState.position;  
}
```



Example: PartOfSpeech tagging

3. Implement consumer that uses PartOfSpeechAttribute

```
void writeProx(FreqProxTermsWriter.PostingList p,  
              int proxCode) {  
    if (payload != null && payload.length > 0) {  
        termsHashPerField.writeVInt(1, (proxCode<<1)|1);  
        termsHashPerField.writeVInt(1, payload.length);  
        termsHashPerField.writeBytes(1, payload.data,  
                                     payload.offset, payload.length);  
        hasPayloads = true;  
    } else {  
        termsHashPerField.writeVInt(1, proxCode<<1);  
    }  
    p.lastPosition = fieldState.position;  
}
```



Example: PartOfSpeech tagging

3. Implement consumer that uses PartOfSpeechAttribute

```
void writeProx(FreqProxTermsWriter.PostingList p,  
              int proxCode) {  
    if (posAtt != null) {  
        termsHashPerField.writeVInt(1, (proxCode<<1)|1);  
        byte posAsByte =  
            posToByte(posAtt.getPartOfSpeech());  
        termsHashPerField.writeByte(1, posAsByte);  
    } else {  
        termsHashPerField.writeVInt(1, proxCode<<1);  
    }  
    p.lastPosition = fieldState.position;  
}
```



Example: PartOfSpeech tagging

4. Implement search side

- APIs not defined yet
- LUCENE-1458 changes posting list accessor APIs to use AttributeSource
- Currently being developed



New posting list formats

- PFOR Delta compression (LUCENE-1410)
- Column-stride fields (LUCENE-1231)
- Custom formats...
- `org.apache.lucene.index.DocumentsWriter` is a good starting point to explore the indexing chain



So ...

- Stay tuned.
- Be brave.
- Contribute!



Questions?

Advanced Indexing Techniques with Lucene

Michael Busch

buschmi@{apache.org, us.ibm.com}

