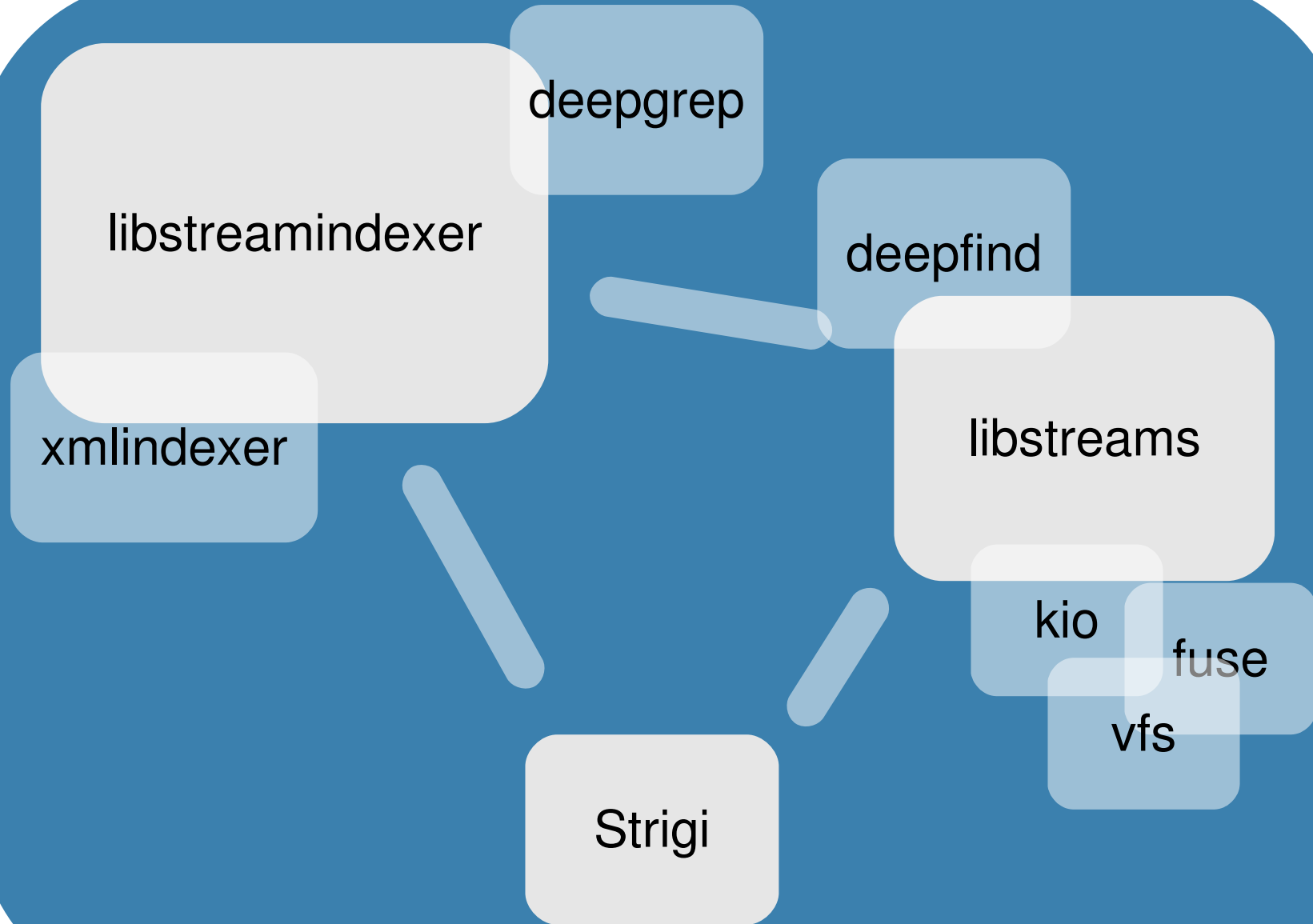




Strigi Internals

Fast libraries for text and metadata

Jos van den Oever



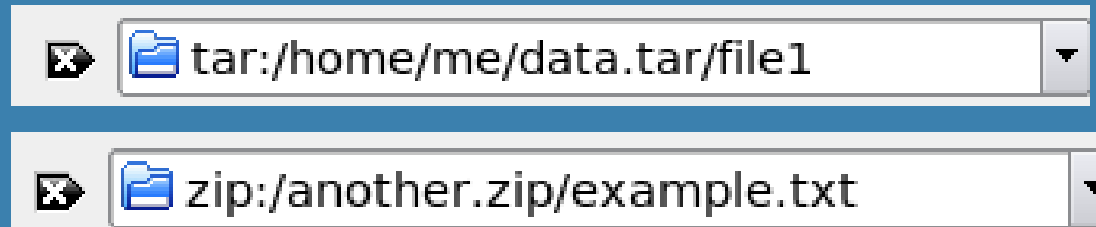
Reading nested files

*.gz	zcat
*.bz2	bzcat
*.tar	tar
*.zip, *.tar, *.jwe, *.ar, openoffice files	unzip
email	mail client
email attachment	mail client
*.pdf (?)	?
*.deb, *.ar, static libs	ar
*.cpio	cpio
*.rpm	rpm2cpio + cpio

many formats, many tools, many interfaces

Common API for nested files

Can we use kio or vfs?



zip:/

tar:/

gz:/

rpm:/

deb:/

commonapi:/

disadvantages:

- user has to figure out what kio or vfs is required

solution:

- make a clever kio/vfs that understands all
- alternative: fuse

Files nested in nested files

`tar:/home/me/data.tar/file1.zip#zip:example.txt`

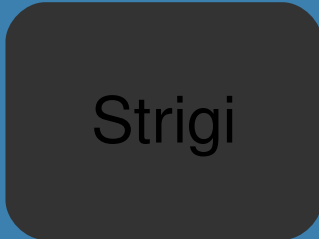
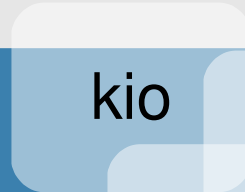
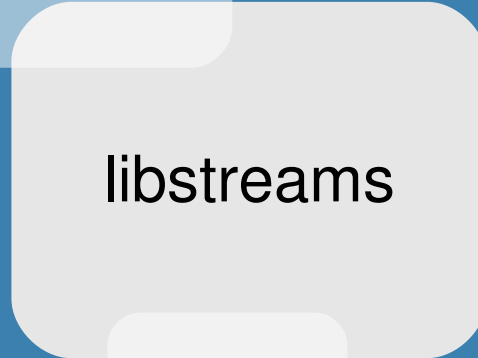
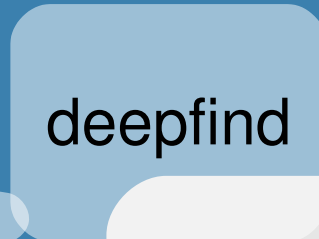
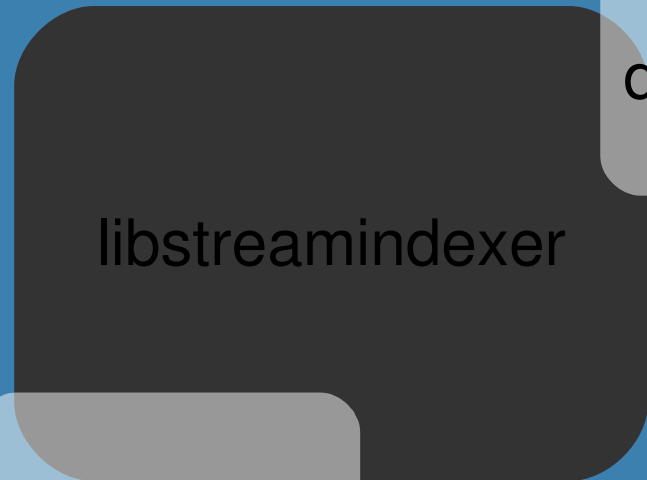
“None of the chained uri stuff (tar/zip/etc)
really work, and never did.”

Alexander Larsson,
Oct 2005 to gnome-vfs-list@gnome.org

“Bug 73821: Please "unchain" kioslaves.
Browsing a zip inside a zip should work.”

KDE bug since Jan 2004

cause: most implementations rely on random access



Java InputStream

Java has nice streaming base class

```
public StreamDemo(URL url) throws IOException {
    InputStream filestream = url.openStream();
    ZipInputStream zipstream =
        new ZipInputStream(filestream);
    ZipEntry entry = zipstream.getNextEntry();
    while (entry != null) {
        handleEntry(zipstream, entry);
        entry = zipstream.getNextEntry();
    }
}
```

StreamBase<char>

```
class StreamBase<T> {  
    ...  
public:  
    ...  
    virtual int32_t  
        read(const T*& start,  
             int32_t min,  
             int32_t max) = 0;  
    virtual int64_t  
        reset(int64_t pos) = 0;  
    ...  
};
```

```
void  
readdemo() {  
    int32_t nread;  
    const char* data;  
    nread = jstream->read(data, 1, 0); // read at least 1 byte  
    jstream->reset(0); // reset to start of stream  
    nread = jstream->read(data, 3, 3); // read exactly 3 bytes  
}
```

- Simple abstract class
 - templated class
 - one read function
 - passes a pointer to an internal buffer
 - only two functions need to be implemented

BufferedStream<char>

```
class BufferedStream<T> {  
    ...  
public:  
    ...  
    virtual int32_t  
        fillBuffer(T* start,  
                  int32_t space) = 0;  
    ...  
};
```

- Examples

- FileInputStream
- BZ2InputStream
- GZipInputStream
- InputStreamReader
- ProcessInputStream

- Stream with a buffer
 - most common use case
 - implement one simple function
 - called when the buffer is empty
 - ImapInputStream
 - HttpInputStream

Streams without buffer

```
class SubInputStream<T> {
    ...
public:
    SubInputStream(
        StreamBase<char>* input,
        int32_t size);
    ...
};
```

- SubInputstream

- a size limited version of another stream

```
class
StringTerminatedSubStream<T> {
    ...
public:
    StringTerminatedSubStream(
        StreamBase<char>* input,
        int32_t size);
    ...
};
```

- StringTerminatedSub Stream

- a string limited version of another stream

SubStreamProvider

```
class SubStreamProvider {  
    ...  
public:  
    SubStreamProvider(  
        StreamBase<char>*  
input);  
    virtual StreamBase<char>*  
        nextEntry() = 0;  
    const EntryInfo&  
        getEntryInfo() const;  
    ...  
};
```

- Examples

- TarInputStream,
ZipInputStream
- ArInputStream,
RpmInputStream
- MailInputStream

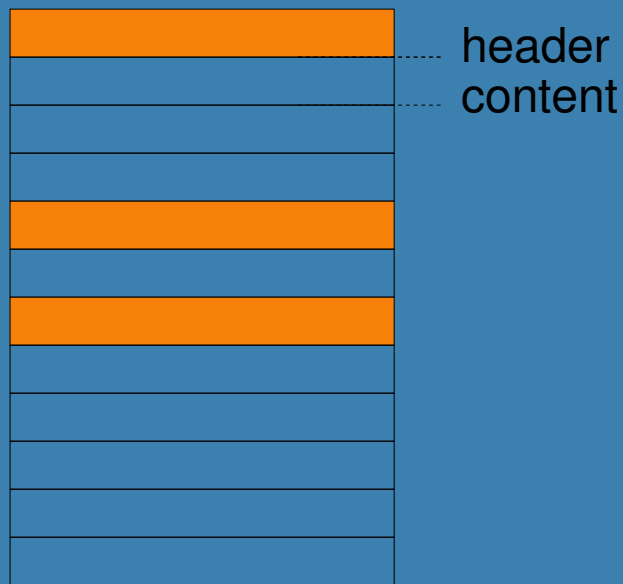
- Split a stream up

- access parts of
resources
- implement one simple
function
- called to get streams
one after the other

Example: TarInputStream

Tar file format

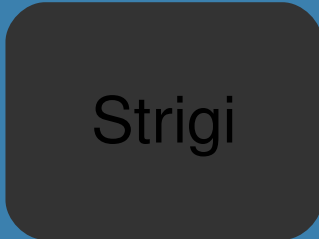
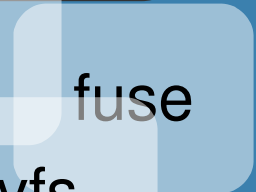
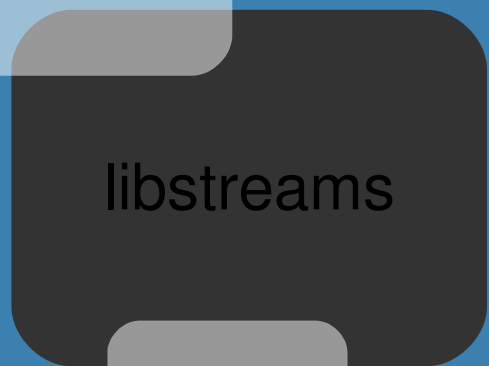
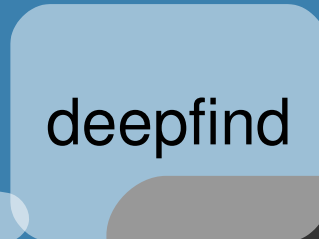
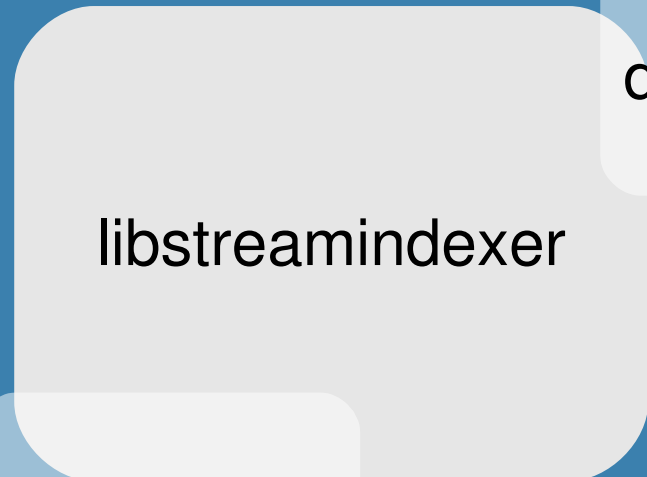
512 byte blocks



- Simple SubStreamProvider
 - fixed size blocks
 - no additional buffer required
 - parse the header into the `EntryInfo` object
 - Position the stream at the start of the content and create a `SubInputStream` with the given size of the stream

directory | file





xmlindexer

libstreamindexer

deepgrep

deepfind

libstreams

kio

fuse

vfs

Strigi



Harvesting information

- Extracting text
 - get text from different file formats
- Semantic Desktop
 - extract data from many files and other resources
 - filesize, email addresses, titles, authors, links
- Efficient tools required
- Extensible system
 - common library for performing the harvesting

libstreamindexer

- Extensible library
- Almost no dependencies (z, bz2, crypto)
- Plugin architecture
- Flexible interface for use in different application types

the naïve method

- 1 Determine *the* mimetype with library or external application from filename and/or part of the contents
- 2 Use the mimetype to choose one file handler from the list of available ones
- 3 Call the library or application on the file
- 4 Convert the result into something useable

- 1 **mimetype as key**
- 2 **external applications**
- 3 **works on files**
- 4 **only one program per file**

how is libstreamindexer better?

- 1 no keys for choosing analyzers
- 2 use plugins instead of applications
- 3 work on streams (but not exclusively)
- 4 possibly many plugins per file

analysis pipeline



IndexerConfiguration



document stream



through analyzers



end analyzer



Indexable




IndexWriter

analysis pipeline

- 1 Create an IndexerConfiguration
- 2 Create an IndexWriter
- 3 Create an Indexable from a stream
- 4 Start indexing
- 5 Initialize all available throughanalyzers
- 6 Try all endanalyzers until one works
- 7 Let the endanalyzer handle the stream
- 8 Get the stream size

for each
substream



IndexerConfiguration

- determines whether to index subfiles, if text should be extracted
- determines how to index each field (binary, compressed, indexed, stored, tokenized)
- can filter on file- and pathname
- determines which analyzers to use
- abstract class

IndexWriter

- abstract class for outputting the harvesting results
- can be used with IndexManager and IndexReader

Writing analyzers

- implement a factory for introspection of the field name and instanciating the analyzer
- for `EndStreamAnalyzer`:
 - implement `checkHeader()` dfd
 - implement `analyze()`
- for `ThroughStreamAnalyzer`:
 - implement `connectInputStream()`

Use case 1: deepfind

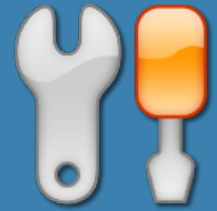
- IndexerConfiguration
 - index subfiles: yes
 - extract text and metadata: no
- IndexWriter
 - write complete pathnames



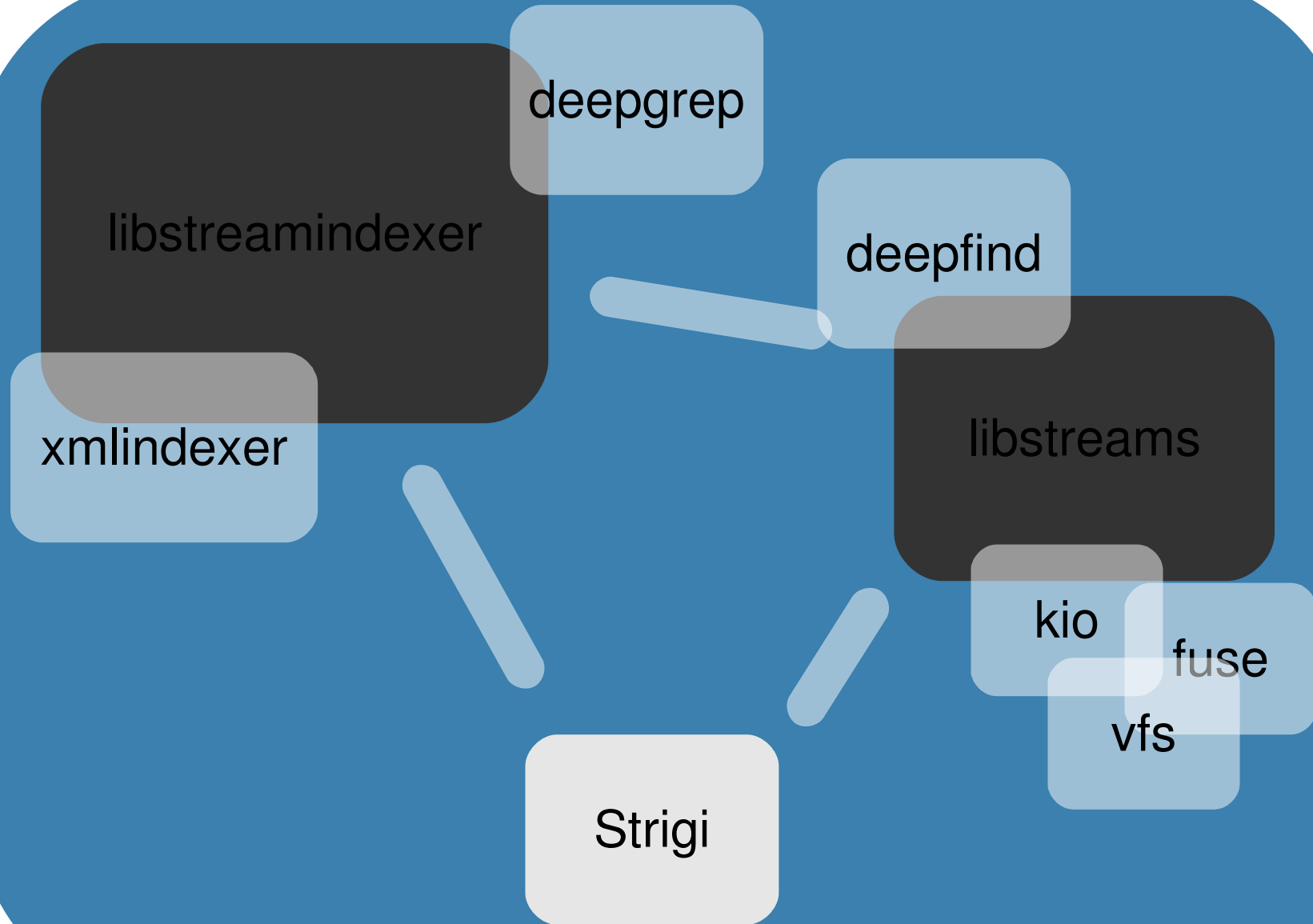
Result: a 'find' that can list embedded files

Use case 2: deepgrep

- IndexerConfiguration
 - index subfiles: yes
 - extract text and metadata: yes
- IndexWriter
 - write the lines that match the search expression



Result: a 'grep' that can search in embedded files and non-text files



Use case 3: Strigi

- IndexerConfiguration
 - index subfiles: yes
 - extract text and metadata: yes
- IndexWriter
 - write into one of the Strigi backend indexes



Result: the smallest and fastest desktop search engine

Speed comparison

Indexing 10 000 text files (168 MB)

Beagle	2h18	12m
Jindex	3h02	9m
Tracker	3h03	142m
Strigi	0h04	>4m

Source: Comparison of indexers

November, 2006

Michal Pryc, Xusheng Hui

Sun Microsystems

Use case 4: xmlindexer

- IndexerConfiguration
 - index subfiles: yes
 - extract text and metadata: yes
- IndexWriter
 - configurable XML format



Result: speed and support for embedded files for everyone

Use case 5: KDE4

- IndexerConfiguration
 - index subfiles: configurable
 - extract text and metadata: configurable
- IndexWriter
 - write into KFileMetaInfo



Result: metadata consistent with search tools and compatible with the semantic desktop

Metadata description standard

- use URIs as keys
- provide translations of keyword and description
- allow for inheritance
- use typed fields
- describe cardinality
- describe if property is embedded
- describe if property is derived

