Query-FS: Integrating with UNIX from Common Lisp via FS API

Michael Raskin, raskin@mccme.ru

TU Munich

May 24, 2020

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Disclaimer

- I actually like some of the UNIX ideas
- I might not be a Common Lisp programmer
  (just a programmer who uses Common Lisp for some things)
- I do not use Emacs
  ... I coordinate parts of my environment in other ways
- I sometimes prefer non-S-expression-based syntax
- I even like Bash! And plain SQL. For *some* tasks

Query-FS is a virtual POSIX FS, implemented using FUSE;
Lisp is used as a great tool, not something defining every choice

... Query-FS reflects what *I* wanted to use (and how),
but I am happy to extend it if you have a use case!

Overview

- Virtual filesystem
- File layout created by code — on the fly
- Queries in pluggable DSLs
- «A Lisp data structure as a directory»
- «SQL SELECT as a directory»

Overview

- Virtual filesystem
- File layout created by code — on the fly
- Queries in pluggable DSLs
- «A Lisp data structure as a directory»
- «SQL SELECT as a directory»

Overview

- Virtual filesystem
- File layout created by code — on the fly
- Queries in pluggable DSLs
- «A Lisp data structure as a directory»
- «SQL SELECT as a directory»

Goals

- Integrate with everything that speaks POSIX FS
  so just everything
- Provide content or symbolic links to content
- Easy access to SQL
  RelFS was nice! but too limited
- Flexibility to add vaguely «query-like» things

Goals

- Integrate with everything that speaks POSIX FS
  so just everything
- Provide content or symbolic links to content
- Easy access to SQL
  RelFS was nice! but too limited
- Flexibility to add vaguely «query-like» things

Goals

- Integrate with everything that speaks POSIX FS
  so just everything
- Provide content or symbolic links to content
- Easy access to SQL
  RelFS was nice! but too limited
- Flexibility to add vaguely «query-like» things

Goals

- Integrate with everything that speaks POSIX FS
  so just everything
- Provide content or symbolic links to content
- Easy access to SQL
  RelFS was nice! but too limited
- Flexibility to add vaguely «query-like» things

Demo

Install some native stuff...

```
$ package-manager install gcc libfuse-development
```

Get the latest update and dependencies

```
$ cd ~/quicklisp/local-projects
$ git clone https://gitlab.common-lisp.net/cl-fuse/query-fs
* (ql:quickload :query-fs)
```

Run it!

```
* (query-fs:run-fs :target "query-fs-test")
```

You now have `query-fs-test/results`

Demo

Install some native stuff...

```
$ package-manager install gcc libfuse-development
```

Get the latest update and dependencies

```
$ cd ~/quicklisp/local-projects
$ git clone https://gitlab.common-lisp.net/cl-fuse/query-fs
* (ql:quickload :query-fs)
```

Run it!

```
* (query-fs:run-fs :target "query-fs-test")
```

You now have `query-fs-test/results`

Demo (pointless)

Install some native stuff...

```
$ package-manager install gcc libfuse-development
```

Get the latest update and dependencies

```
$ cd ~/quicklisp/local-projects
$ git clone https://gitlab.common-lisp.net/cl-fuse/query-fs
* (ql:quickload :query-fs)
```

Run it!

```
* (query-fs:run-fs :target "query-fs-test")
```

You now have `query-fs-test/results`
... it is empty: no queries to represent

Demo (minimal)

Let's create query-fs-test/queries/short-doc.sexp

```
("describe-this-dir"
  "This is the mount point of a query returning some text.")
("what-is-used"
  ("SBCL" "Steel Bank Common Lisp compiler")
  ("FUSE" "File system in USEr space"))
```

... and try starting Query-FS again

```
* (query-fs:run-fs :target "query-fs-test")
$ ls query-fs-test/results/short-doc/
describe-this-dir  what-is-used/
$ cat query-fs-test/results/short-doc/describe-this-dir
This is the mount point of a query returning some text.
```

Demo (minimal)

Let's create query-fs-test/queries/short-doc.sexp

```
("describe-this-dir"
  "This is the mount point of a query returning some text.")
("what-is-used"
  ("SBCL" "Steel Bank Common Lisp compiler")
  ("FUSE" "File system in USEr space"))
```

... and try starting Query-FS again

```
* (query-fs:run-fs :target "query-fs-test")
$ ls query-fs-test/results/short-doc/
describe-this-dir  what-is-used/
$ cat query-fs-test/results/short-doc/describe-this-dir
This is the mount point of a query returning some text.
```

Demo (non-constant)

Let's create query-fs-test/queries/sum-numbers.cl

```
(defparameter *sum-numbers-range* 10)

(mk-splice
  (mk-file "README" "A POSIX interface to #'CL:+")
  (mk-pair-generator x
    (loop for k from 1 to *sum-numbers-range*
          collect (list (format nil "~a" k) k))
    (mk-dir (first x) :just
            (mk-pair-generator y
              (loop for k from 1 to *sum-numbers-range*
                    collect (list (format nil "~a" k)
                                  (+ k (second x))))
              (mk-file (first y)
                       (format nil "~a" (second y)))))))
```

Demo (non-constant)

```
$ ls query-fs-test/results/sum-numbers/
1/  10/  2/  3/  4/  5/  6/  7/  8/  9/  README
$ cat query-fs-test/results/sum-numbers/3/4
7
```

Demo (SQL)

SQL means a DB... I use PostgreSQL (and I have a local server)
Let's prepare a playground

```
$ echo "…" > /home/test/psql-pass
$ createdb test_queryfs
$ psql -d test_queryfs -c \
    "create table test_table (
        name varchar,
        content varchar
     );"
```

Demo (SQL)

Now let's install some stuff for Query-FS

```
$ package-manager install postgresql-client
* (ql:quickload :clsql-postgresql :esrap-peg)
```

And start filling `query-fs-test/queries/db.sql2`

```
set db-server="127.0.0.1"
set db-name="test_queryfs"
set db-type="postgresql"
set db-user="test"

read db-password < "/home/test/psql-pass"
```

Demo (SQL)

Now let's install some stuff for Query-FS

```
$ package-manager install postgresql-client
* (ql:quickload :clsql-postgresql :esrap-peg)
```

And start filling `query-fs-test/queries/db.sql2`

```
set db-server="127.0.0.1"
set db-name="test_queryfs"
set db-type="postgresql"
set db-user="test"

read db-password < "/home/test/psql-pass"
```

Demo (SQL)

Now some actual query

```
mkdir "all" do
  for x in "select name, content from test_table"
    with-file $name do
      on-read $x[1]
      on-write data "update test_table
                     set content = ${data}
                     where name = ${name}"
      on-remove "delete from test_table
                 where name = ${name}"
    done
  on-create-file name "insert into test_table
                       (name) values (${name})"
done
```

Demo (SQL)

It works

```
$ echo qwe > query-fs-test/results/db/all/123
$ echo asd > query-fs-test/results/db/all/12345
$ cat query-fs-test/results/db/all/123
qwe
```

Demo: more than a boring FS

Extend the query

```
mkdir "silly" do
  for x in "select ${x[0]},
                   'Indeed, we have '|| ${x[0]} ||' here!'
              where ${x[0]} is not null"
    with-file $name do
      on-read $x[1]
    done
done
```

And now...

```
$ ls query-fs-test/results/db/silly/
$ cat query-fs-test/results/db/silly/code
Indeed, we have code here!
```

Demo: more than a boring FS

Extend the query

```
mkdir "silly" do
  for x in "select ${x[0]},
                   'Indeed, we have '|| ${x[0]} ||' here!'
              where ${x[0]} is not null"
    with-file $name do
      on-read $x[1]
    done
done
```

And now...

```
$ ls query-fs-test/results/db/silly/
$ cat query-fs-test/results/db/silly/code
Indeed, we have code here!
```

Demo: more than a boring FS

«We have everything» works in every syntax

```
(mk-pair-generator x
  (let ((xn (ignore-errors (parse-integer (first x)))))
    (if xn `((,(first x) ,(1+ xn)))
        (loop for k from 1 to 10
              collect `(,(format nil "~a" k) ,(1+ k)))))
  (mk-file (first x) (format nil "~a" (second x))))
```

```
$ ls query-fs-test/results/1plus/
1  10  2  3  4  5  6  7  8  9
$ cat query-fs-test/results/1plus/3
4
$ cat query-fs-test/results/1plus/33
34
$ cat query-fs-test/results/1plus/no
cat: /home/raskin/queries/plus1/no: No such file or directory
```

Demo: more than a boring FS

«We have everything» works in every syntax

```
(mk-pair-generator x
  (let ((xn (ignore-errors (parse-integer (first x)))))
    (if xn `((,(first x) ,(1+ xn)))
        (loop for k from 1 to 10
              collect `(,(format nil "~a" k) ,(1+ k)))))
  (mk-file (first x) (format nil "~a" (second x))))
```

```
$ ls query-fs-test/results/1plus/
1  10  2  3  4  5  6  7  8  9
$ cat query-fs-test/results/1plus/3
4
$ cat query-fs-test/results/1plus/33
34
$ cat query-fs-test/results/1plus/no
cat: /home/raskin/queries/plus1/no: No such file or directory
```

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE low-level API
  - A slightly lispy wrapper on top
- CL-FUSE-Meta-FS
  - Produce list-based layout instead of callbacks
  - A set of macros to define layouts
    Used in `sum-numbers.cl`
  - Missing: CLOS-based API
- Query-FS
  - Plugins to parse queries
  - For each query, plugin outputs lisp code
    CL-FUSE-Meta-FS layout descriptions
  - Complete FS definition composed of translated queries
  - Queries can be updated while FS is mounted

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE low-level API
  - A slightly lispy wrapper on top
- CL-FUSE-Meta-FS
  - Produce list-based layout instead of callbacks
  - A set of macros to define layouts
    Used in `sum-numbers.cl`
  - Missing: CLOS-based API
- Query-FS
  - Plugins to parse queries
  - For each query, plugin outputs lisp code
    CL-FUSE-Meta-FS layout descriptions
  - Complete FS definition composed of translated queries
  - Queries can be updated while FS is mounted

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE low-level API
  - A slightly lispy wrapper on top
- CL-FUSE-Meta-FS
  - Produce list-based layout instead of callbacks
  - A set of macros to define layouts

    Used in `sum-numbers.cl`
  - Missing: CLOS-based API
- Query-FS
  - Plugins to parse queries
  - For each query, plugin outputs lisp code

    CL-FUSE-Meta-FS layout descriptions
  - Complete FS definition composed of translated queries
  - Queries can be updated while FS is mounted

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE low-level API
  - A slightly lispy wrapper on top
- CL-FUSE-Meta-FS
  - Produce list-based layout instead of callbacks
  - A set of macros to define layouts

    Used in `sum-numbers.cl`
  - Missing: CLOS-based API
- Query-FS
  - Plugins to parse queries
  - For each query, plugin outputs lisp code

    CL-FUSE-Meta-FS layout descriptions
  - Complete FS definition composed of translated queries
  - Queries can be updated while FS is mounted

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE low-level API
  - A slightly lispy wrapper on top
- CL-FUSE-Meta-FS
  - Produce list-based layout instead of callbacks
  - A set of macros to define layouts
    Used in `sum-numbers.cl`
  - Missing: CLOS-based API
- Query-FS
  - Plugins to parse queries
  - For each query, plugin outputs lisp code
    CL-FUSE-Meta-FS layout descriptions
  - Complete FS definition composed of translated queries
  - Queries can be updated while FS is mounted

Query-FS lifecycle

- First, find and load plugins
  - Plugins can register parsers for query file extensions
  - Current plugins: Lisp reader, or PEG-defined parsers
  - Parsing returns Lisp code
- Second, find and translate queries
- Third, build a combined definition — and run it
  - A handler can update the definition on the fly
- Exit once done

Query-FS lifecycle

- First, find and load plugins
  - Plugins can register parsers for query file extensions
  - Current plugins: Lisp reader, or PEG-defined parsers
  - Parsing returns Lisp code
- Second, find and translate queries
- Third, build a combined definition — and run it
  - A handler can update the definition on the fly
- Exit once done

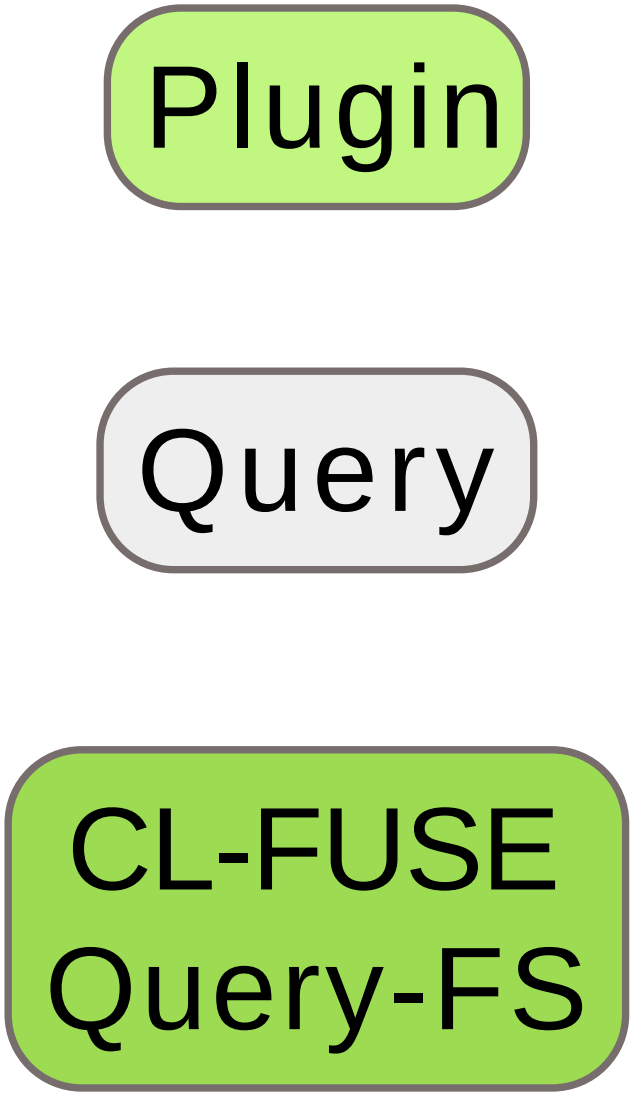
Query-FS lifecycle

- First, find and load plugins
  - Plugins can register parsers for query file extensions
  - Current plugins: Lisp reader, or PEG-defined parsers
  - Parsing returns Lisp code
- Second, find and translate queries
- Third, build a combined definition — and run it
  - A handler can update the definition on the fly
- Exit once done

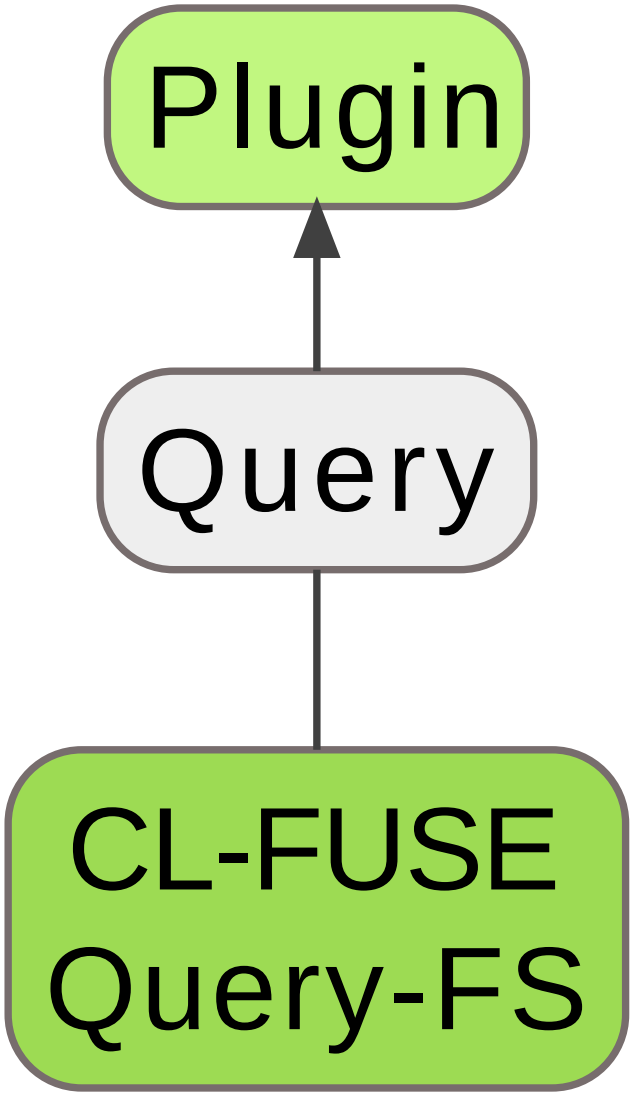
Query-FS lifecycle

- First, find and load plugins
  - Plugins can register parsers for query file extensions
  - Current plugins: Lisp reader, or PEG-defined parsers
  - Parsing returns Lisp code
- Second, find and translate queries
- Third, build a combined definition — and run it
  - A handler can update the definition on the fly
- Exit once done

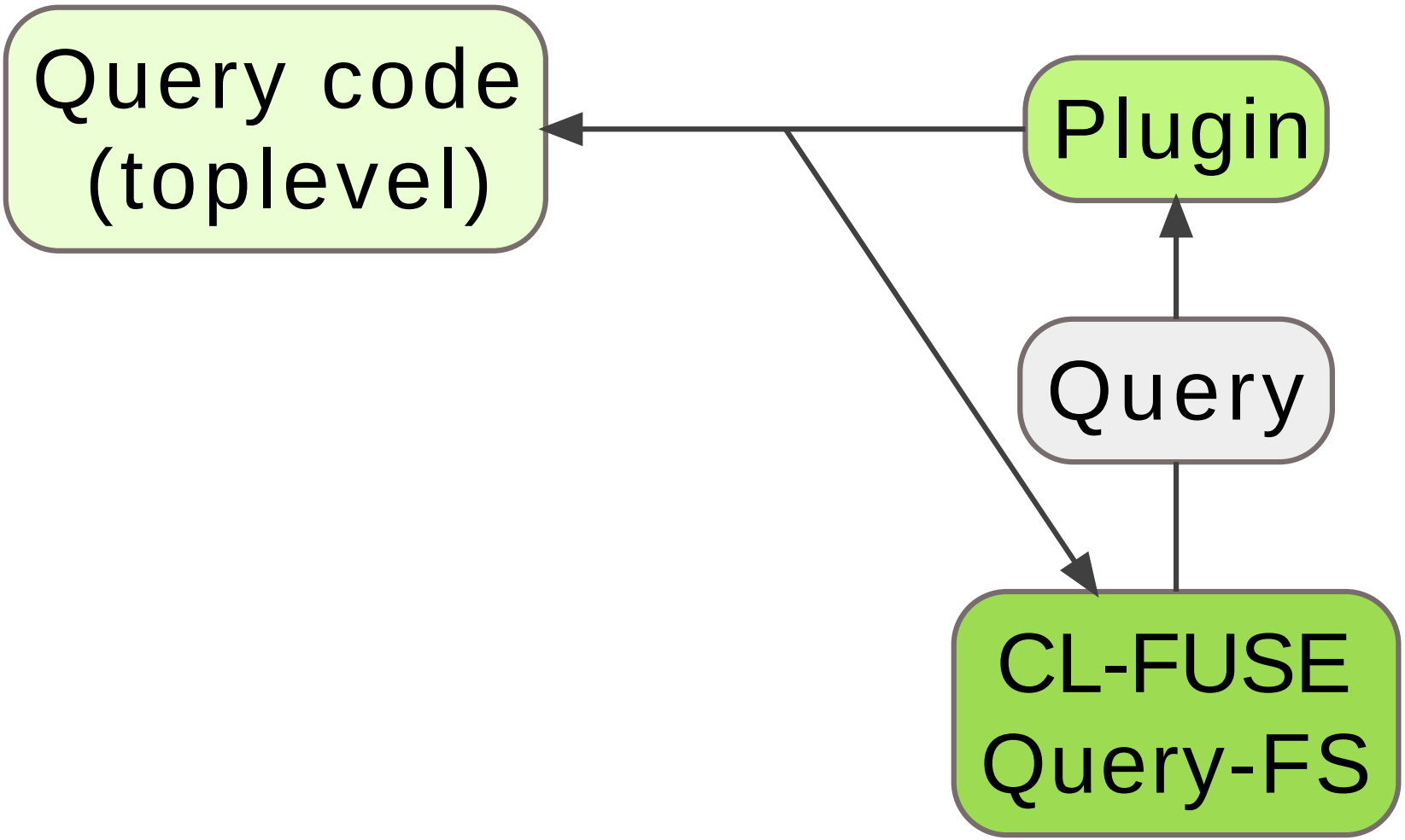
Query-FS request



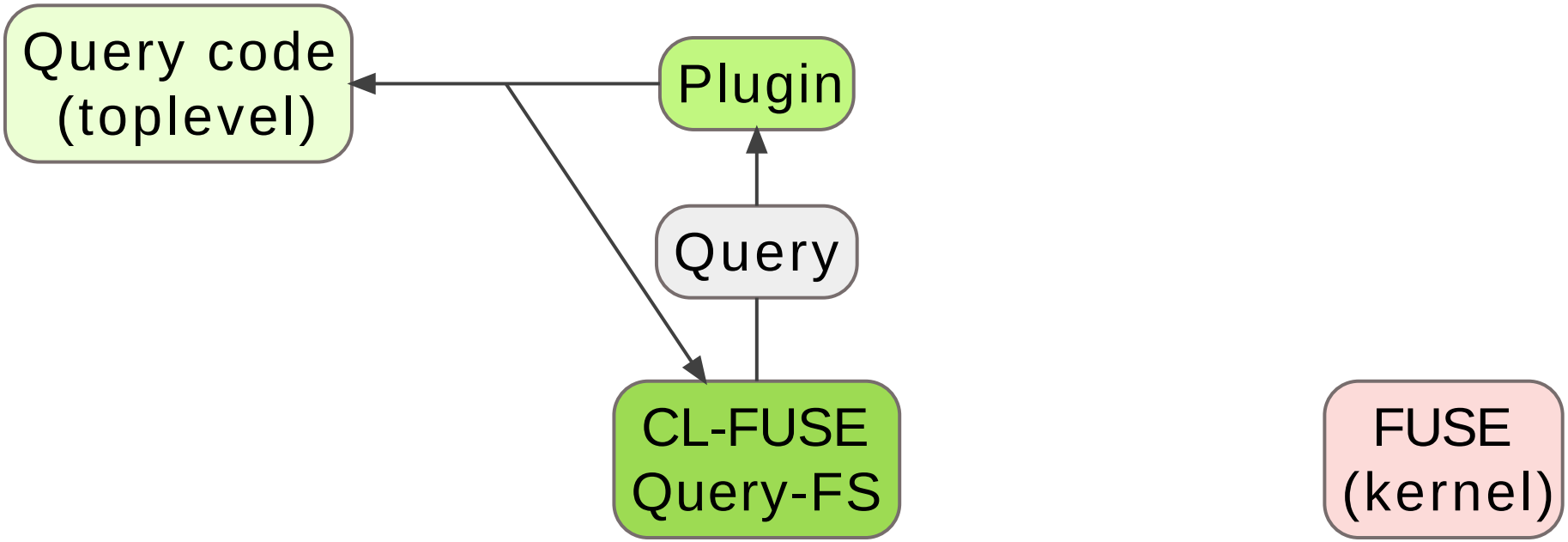
Query-FS request



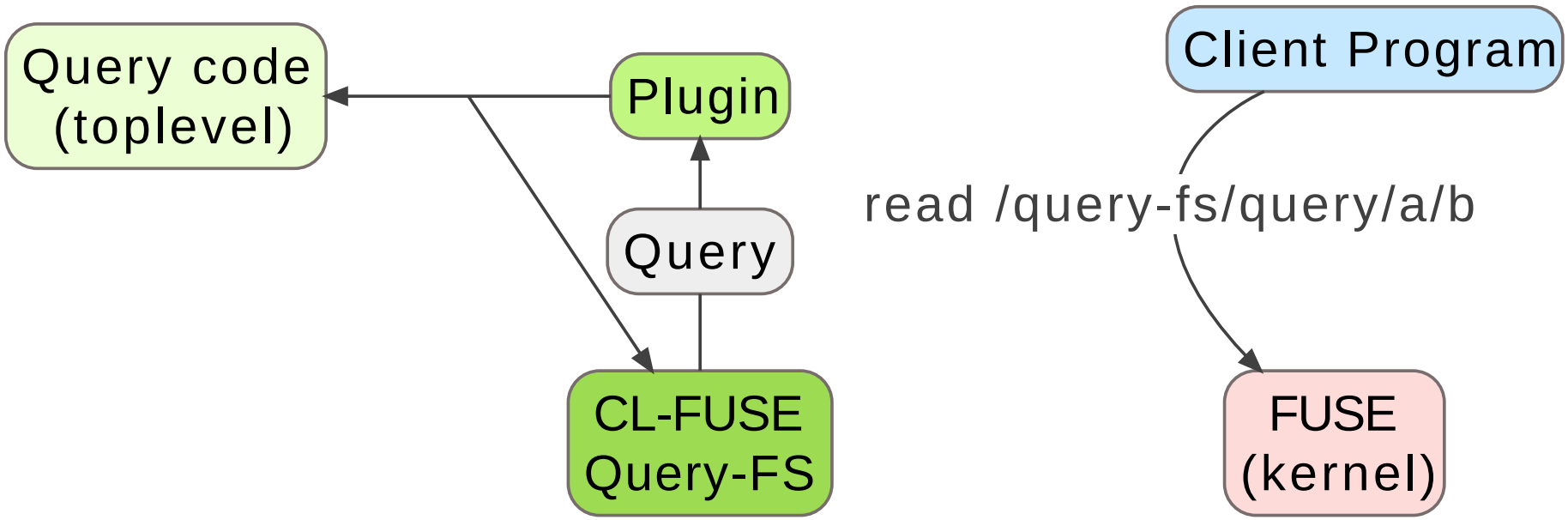
Query-FS request



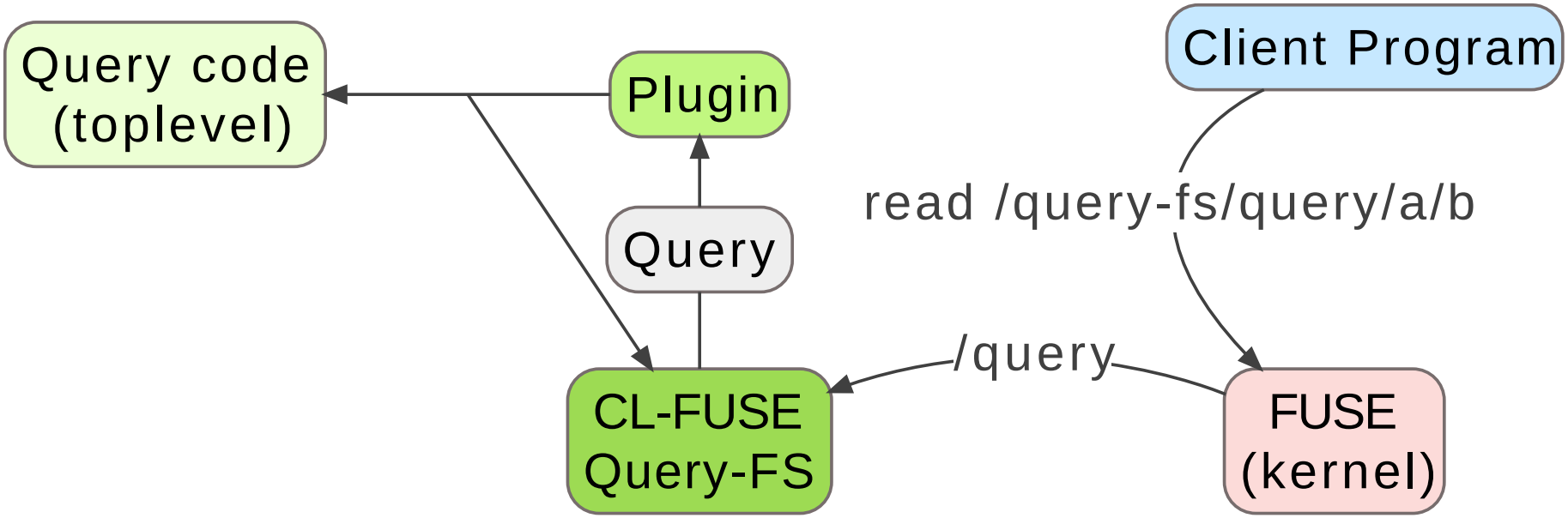
Query-FS request



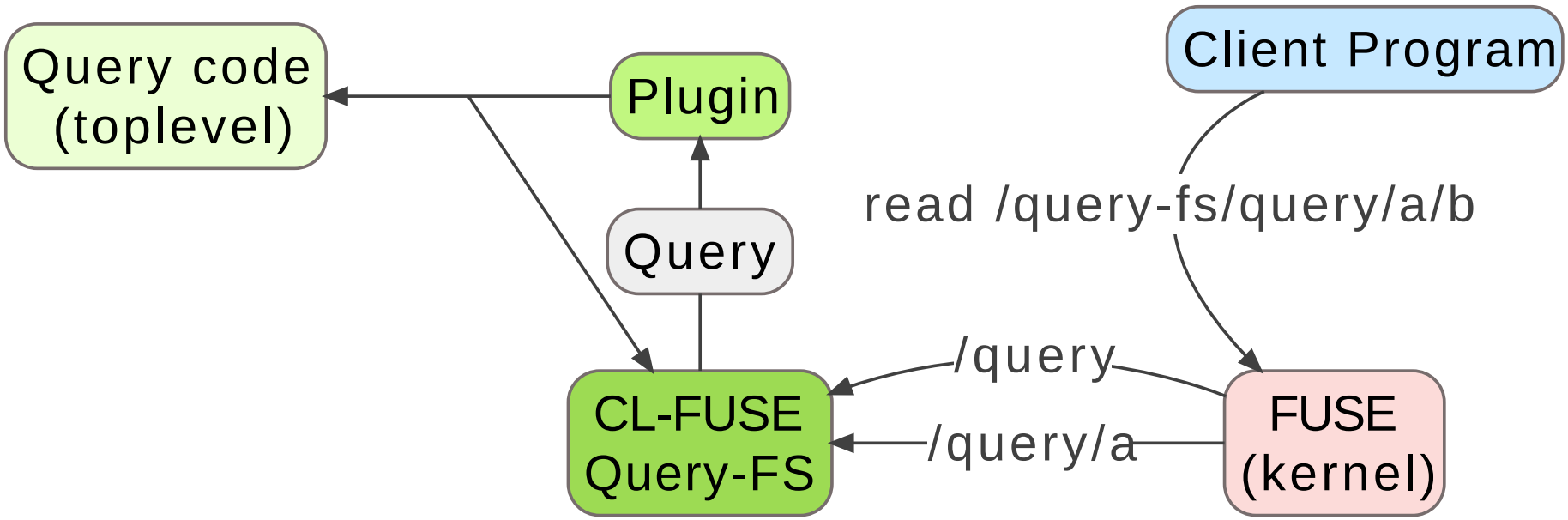
Query-FS request



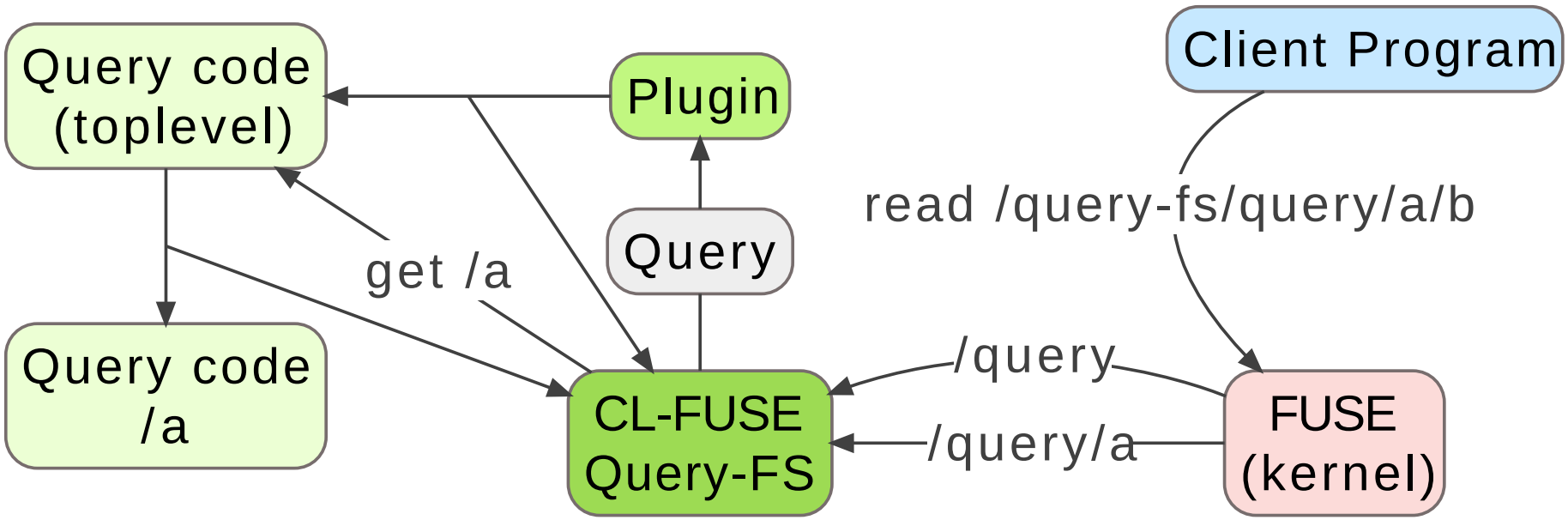
Query-FS request



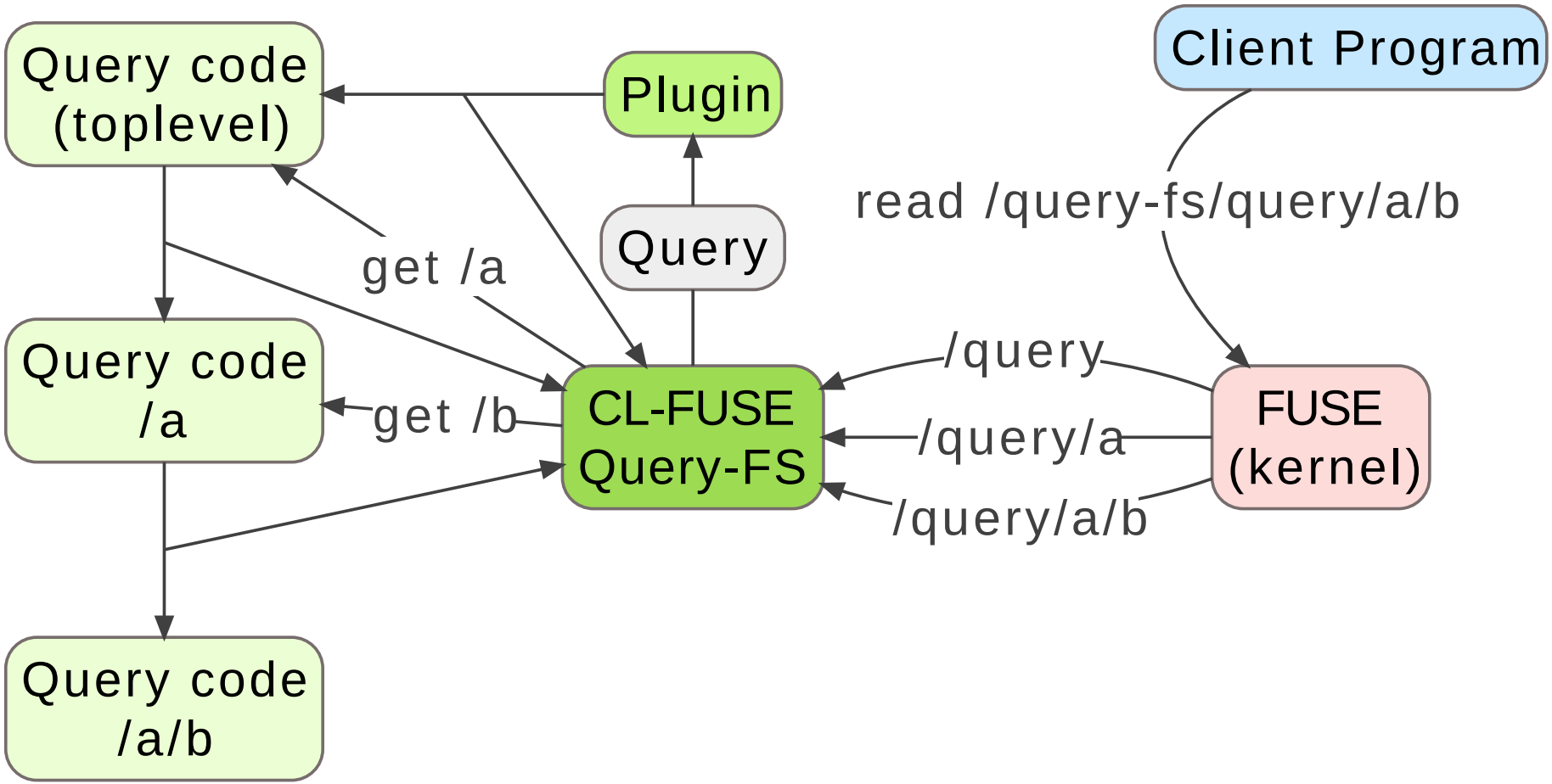
Query-FS request



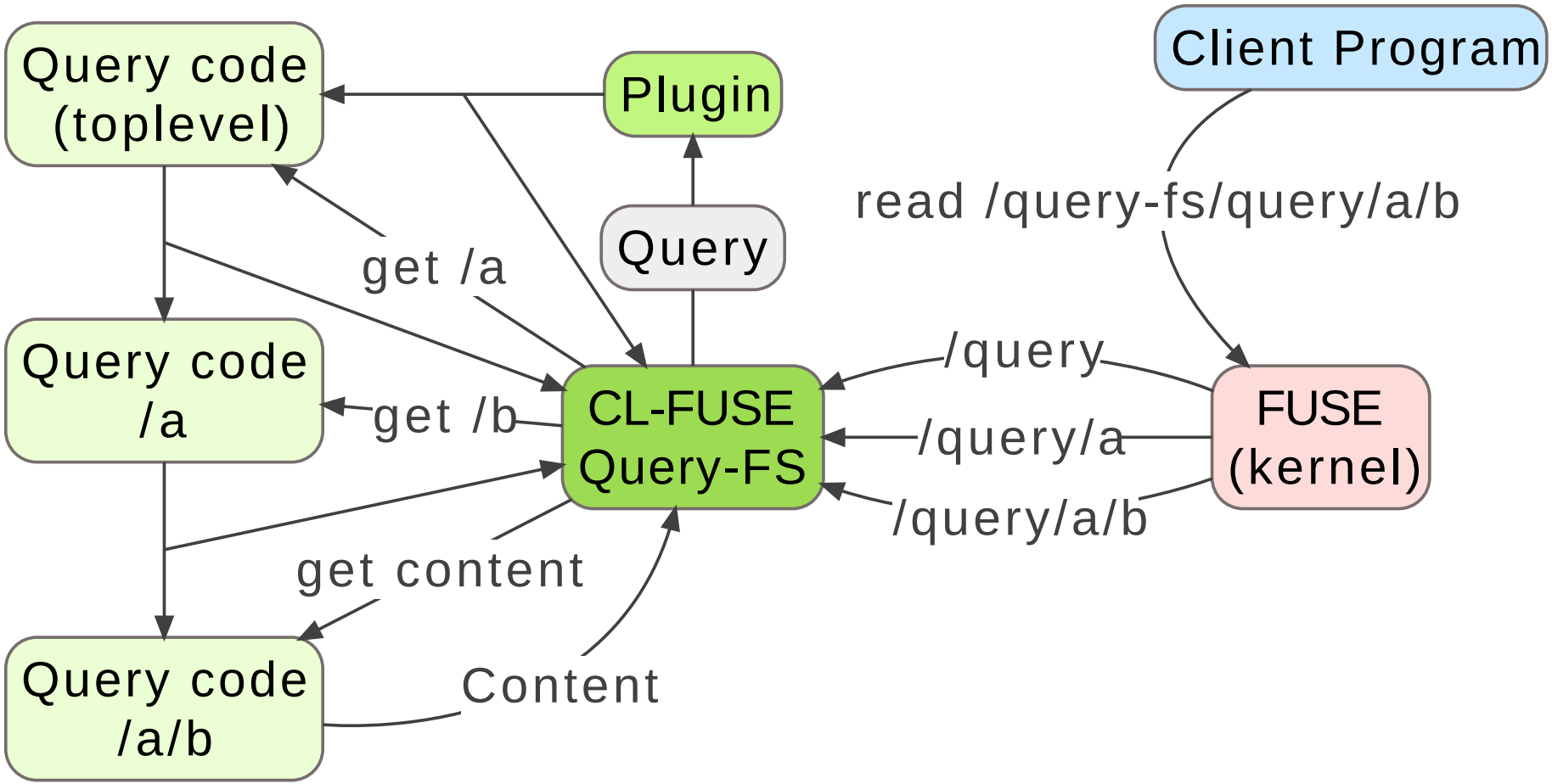
Query-FS request



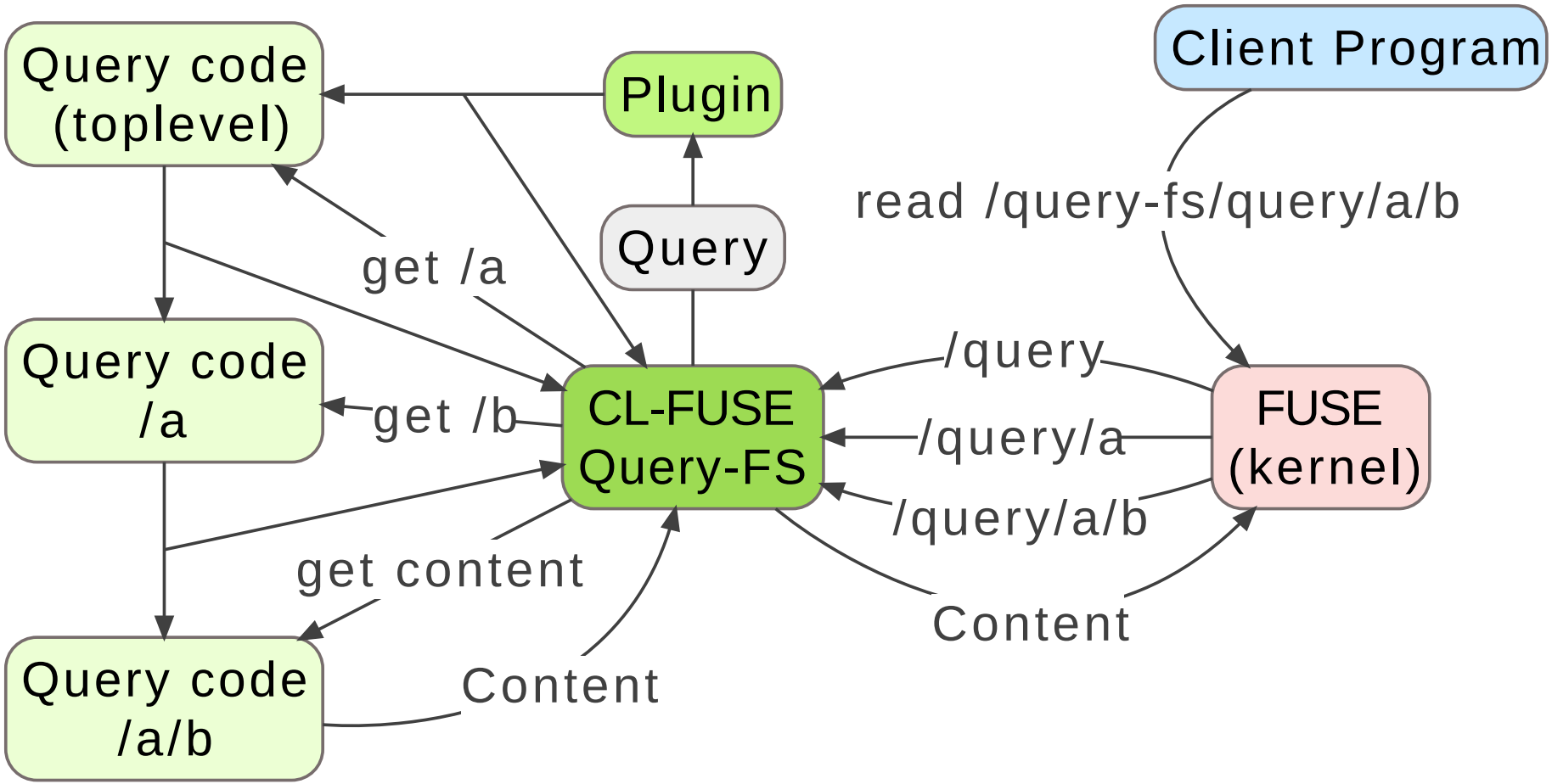
Query-FS request



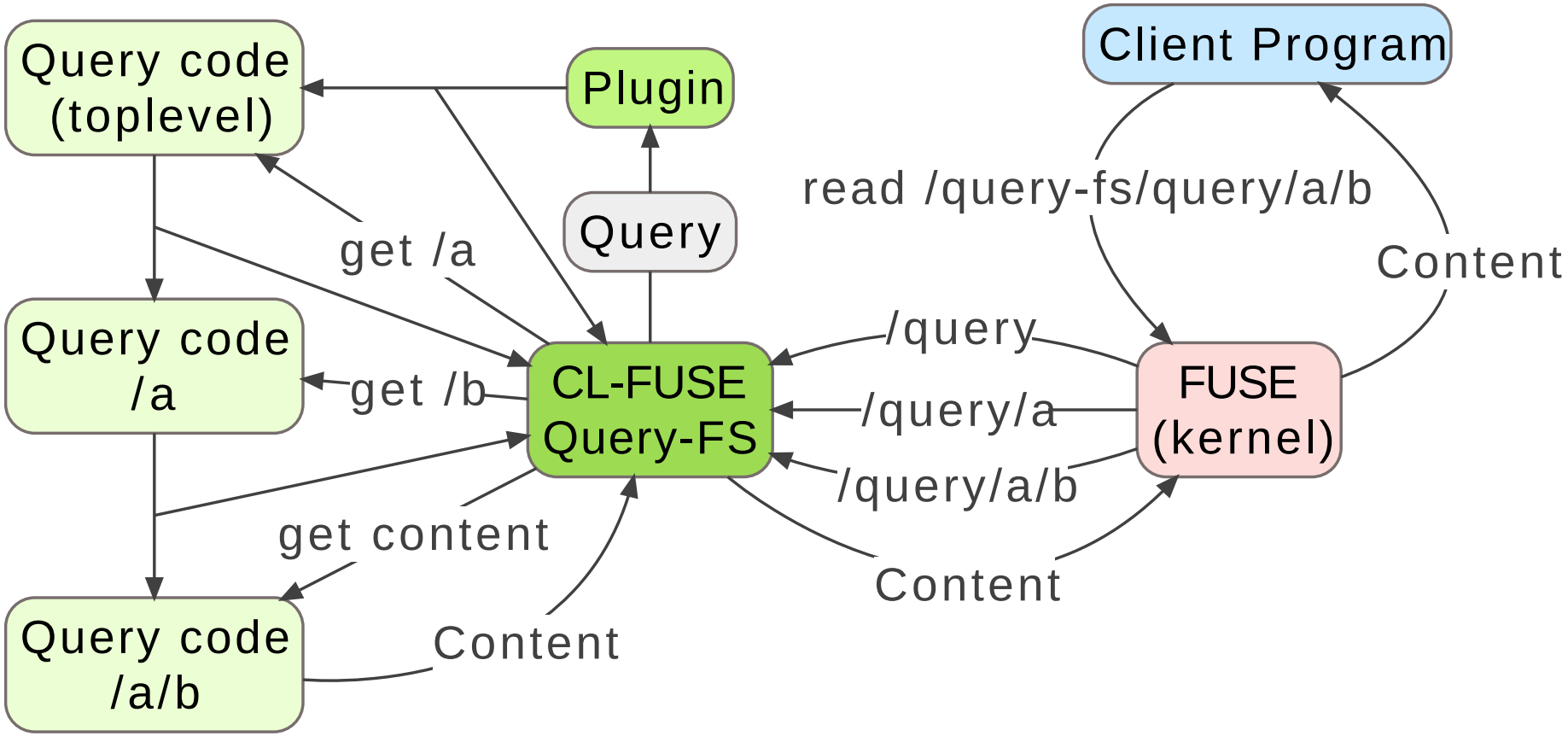
Query-FS request



Query-FS request



Query-FS request



Query-FS: directory contents

- Enumeration vs. lookup
- Full listing can be expensive
- Listing and enumeration are similar — code duplication?
- Combined enumeration/lookup: string means lookup, NIL means list
  CL-FUSE-Meta-FS supports separate lookup and enumeration
- Not as clean, but simpler queries
- Also allows handling entries we do not enumerate

Query-FS: directory contents

- Enumeration vs. lookup
- Full listing can be expensive
- Listing and enumeration are similar — code duplication?
- Combined enumeration/lookup: string means lookup, NIL means list
  CL-FUSE-Meta-FS supports separate lookup and enumeration
- Not as clean, but simpler queries
- Also allows handling entries we do not enumerate

Query-FS: directory contents

- Enumeration vs. lookup
- Full listing can be expensive
- Listing and enumeration are similar — code duplication?
- Combined enumeration/lookup: string means lookup, NIL means list
  CL-FUSE-Meta-FS supports separate lookup and enumeration
- Not as clean, but simpler queries
- Also allows handling entries we do not enumerate

Query-FS: directory contents

- Enumeration vs. lookup
- Full listing can be expensive
- Listing and enumeration are similar — code duplication?
- Combined enumeration/lookup: string means lookup, NIL means list
  CL-FUSE-Meta-FS supports separate lookup and enumeration
- Not as clean, but simpler queries
- Also allows handling entries we do not enumerate

Query-FS: directory contents

- Enumeration vs. lookup
- Full listing can be expensive
- Listing and enumeration are similar — code duplication?
- Combined enumeration/lookup: string means lookup, NIL means list
  CL-FUSE-Meta-FS supports separate lookup and enumeration
- Not as clean, but simpler queries
- Also allows handling entries we do not enumerate

PEG parsing

Esrap-PEG: frontend for Esrap
An abstract grammar

```
WhiteSpace <- " " / "\r" / "\n" / "\t"
S <- WhiteSpace +
OnWrite                  <- "on-write" S Identifier S SQLCommand
```

And pattern-matching to process the AST

```
…
  (OnWrite
    ((_ _ ?var _ ?body)
     `(:on-write
        (,(! ?var)
        ,(! ?body)))))
…
```

Remarks on integration

Filesystem API in general

- Encodings: decide whether (or when) filenames must be valid UTF8...

FUSE

- The simple way to use FUSE: a framework
  - Makes assumptions about threads...
  - FUSE-managed threads + callbacks + GC... no good
- One step below: actual functions... and tons of callbacks
  - Works fine with CFFI
- You do want to exit once the FS is unmounted
- Symbol versioning in `libfuse.so`
  - Hard to handle
  - One-function shared library just to wrap this

Remarks on integration

Filesystem API in general

- Encodings: decide whether (or when) filenames must be valid UTF8...

FUSE

- The simple way to use FUSE: a framework
  - Makes assumptions about threads...
  - FUSE-managed threads + callbacks + GC... no good
- One step below: actual functions... and tons of callbacks
  - Works fine with CFFI
- You do want to exit once the FS is unmounted
- Symbol versioning in `libfuse.so`
  - Hard to handle
  - One-function shared library just to wrap this

Remarks on integration

Filesystem API in general

- Encodings: decide whether (or when) filenames must be valid UTF8...

FUSE

- The simple way to use FUSE: a framework
  - Makes assumptions about threads...
  - FUSE-managed threads + callbacks + GC... no good
- One step below: actual functions... and tons of callbacks
  - Works fine with CFFI
- You do want to exit once the FS is unmounted
- Symbol versioning in `libfuse.so`
  - Hard to handle
  - One-function shared library just to wrap this

Remarks on integration

Filesystem API in general

- Encodings: decide whether (or when) filenames must be valid UTF8...

FUSE

- The simple way to use FUSE: a framework
  - Makes assumptions about threads...
  - FUSE-managed threads + callbacks + GC... no good
- One step below: actual functions... and tons of callbacks
  - Works fine with CFFI
- You do want to exit once the FS is unmounted
- Symbol versioning in `libfuse.so`
  - Hard to handle
  - One-function shared library just to wrap this

Remarks on integration

Filesystem API in general

- Encodings: decide whether (or when) filenames must be valid UTF8...

FUSE

- The simple way to use FUSE: a framework
  - Makes assumptions about threads...
  - FUSE-managed threads + callbacks + GC... no good
- One step below: actual functions... and tons of callbacks
  - Works fine with CFFI
- You do want to exit once the FS is unmounted
- Symbol versioning in `libfuse.so`
  - Hard to handle
  - One-function shared library just to wrap this

Query-FS: my usage

- Email: indexed in PostgreSQL DB, read in Vim using Query-FS
- Planet.Lisp.org (and many other feeds): same
- Password manager: same, with master key to encrypt entries
  Probably wasn't a very good idea...
- File tagging: implemented... but I don't use it

Query-FS: my usage

- Email: indexed in PostgreSQL DB, read in Vim using Query-FS
- Planet.Lisp.org (and many other feeds): same
- Password manager: same, with master key to encrypt entries
  Probably wasn't a very good idea...
- File tagging: implemented... but I don't use it

Query-FS: my usage

- Email: indexed in PostgreSQL DB, read in Vim using Query-FS
- Planet.Lisp.org (and many other feeds): same
- Password manager: same, with master key to encrypt entries
  Probably wasn't a very good idea...
- File tagging: implemented... but I don't use it

Query-FS: my usage

- Email: indexed in PostgreSQL DB, read in Vim using Query-FS
- Planet.Lisp.org (and many other feeds): same
- Password manager: same, with master key to encrypt entries
  Probably wasn't a very good idea...
- File tagging: implemented... but I don't use it

Thanks!

Thanks for your attention!

Questions?

`https://gitlab.common-lisp.net/cl-fuse/query-fs`