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

Michael Raskin, raskin@mccme.ru

TU Munich

March 21, 2022

The author is supported by project receiving funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS)

Disclaimer

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

- Why I care and why you might care probably differ
  … but I want to find other use-cases and add support!
- I like Common Lisp where it fits, even if nothing is perfect
  … but the same for Bash and SQL and UNIX process boundaries
- I even use Vim not Emacs
  … but Vim, Firefox, Emacs — FS API is universal

For me, something flexible enough for SQL as FS,
for you, exporting functionality in CL to whatever speaks POSIX FS API

Disclaimer

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

- Why I care and why you might care probably differ
  … but I want to find other use-cases and add support!
- I like Common Lisp where it fits, even if nothing is perfect
  … but the same for Bash and SQL and UNIX process boundaries
- I even use Vim not Emacs
  … but Vim, Firefox, Emacs — FS API is universal

For me, something flexible enough for SQL as FS,
for you, exporting functionality in CL to whatever speaks POSIX FS API

Disclaimer

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

- Why I care and why you might care probably differ
  … but I want to find other use-cases and add support!
- I like Common Lisp where it fits, even if nothing is perfect
  … but the same for Bash and SQL and UNIX process boundaries
- I even use Vim not Emacs
  … but Vim, Firefox, Emacs — FS API is universal

For me, something flexible enough for SQL as FS,
for you, exporting functionality in CL to whatever speaks POSIX FS API

Disclaimer

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

- Why I care and why you might care probably differ
  … but I want to find other use-cases and add support!
- I like Common Lisp where it fits, even if nothing is perfect
  … but the same for Bash and SQL and UNIX process boundaries
- I even use Vim not Emacs
  … but Vim, Firefox, Emacs — FS API is universal

For me, something flexible enough for SQL as FS,
for you, exporting functionality in CL to whatever speaks POSIX FS API

Disclaimer

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

- Why I care and why you might care probably differ
  … but I want to find other use-cases and add support!
- I like Common Lisp where it fits, even if nothing is perfect
  … but the same for Bash and SQL and UNIX process boundaries
- I even use Vim not Emacs
  … but Vim, Firefox, Emacs — FS API is universal

For me, something flexible enough for SQL as FS,
for you, exporting functionality in CL to whatever speaks POSIX FS API

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»

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: more than a boring FS

+1 filesystem, with large numbers handled on the fly:

```
(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

+1 filesystem, with large numbers handled on the fly:

```
(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 (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!
```

Structure

- CL-FUSE
  - CFFI bindings for FUSE
  - Direct use of FUSE medium-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 `1plus.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 medium-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 `1plus.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 medium-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 `1plus.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 medium-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 `1plus.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

PEG parsing

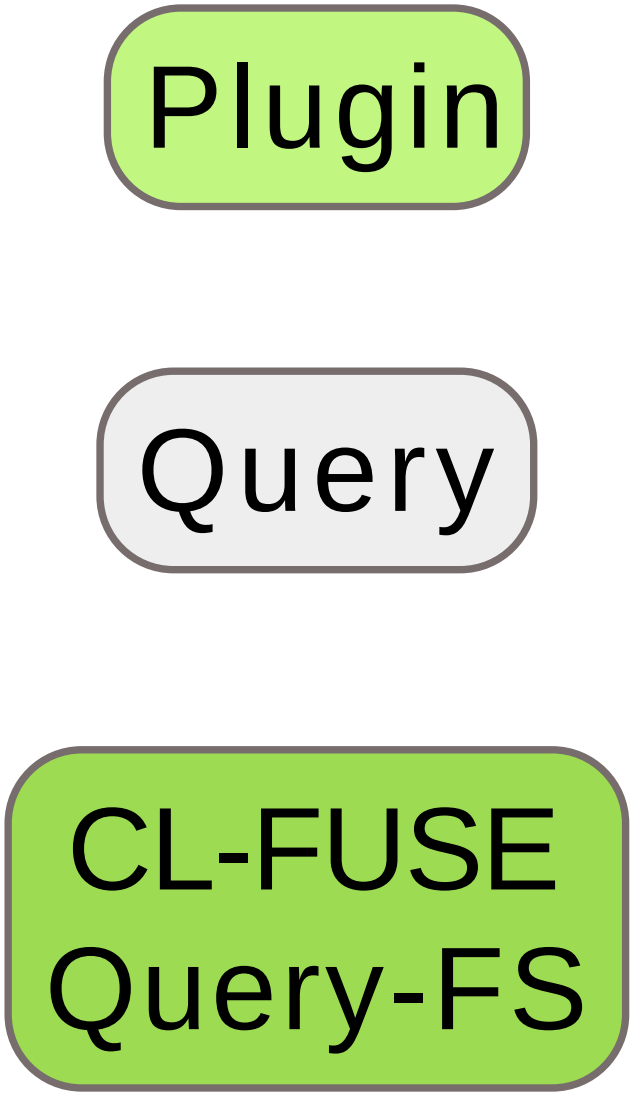
Esrap-PEG: frontend for Esrap
An abstract grammar

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

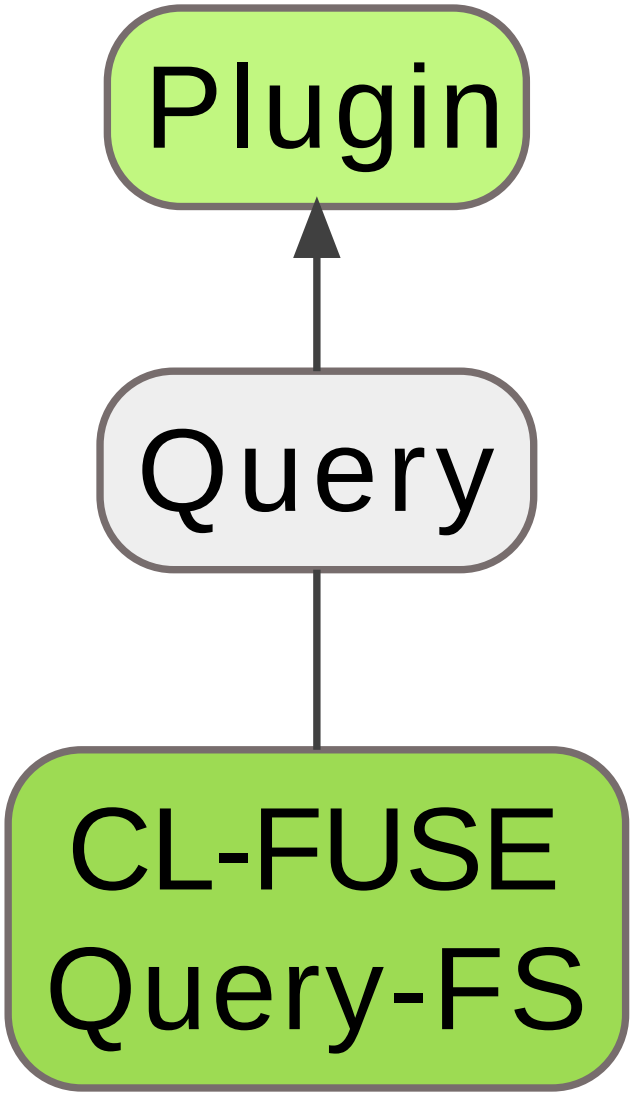
And pattern-matching to process the AST

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

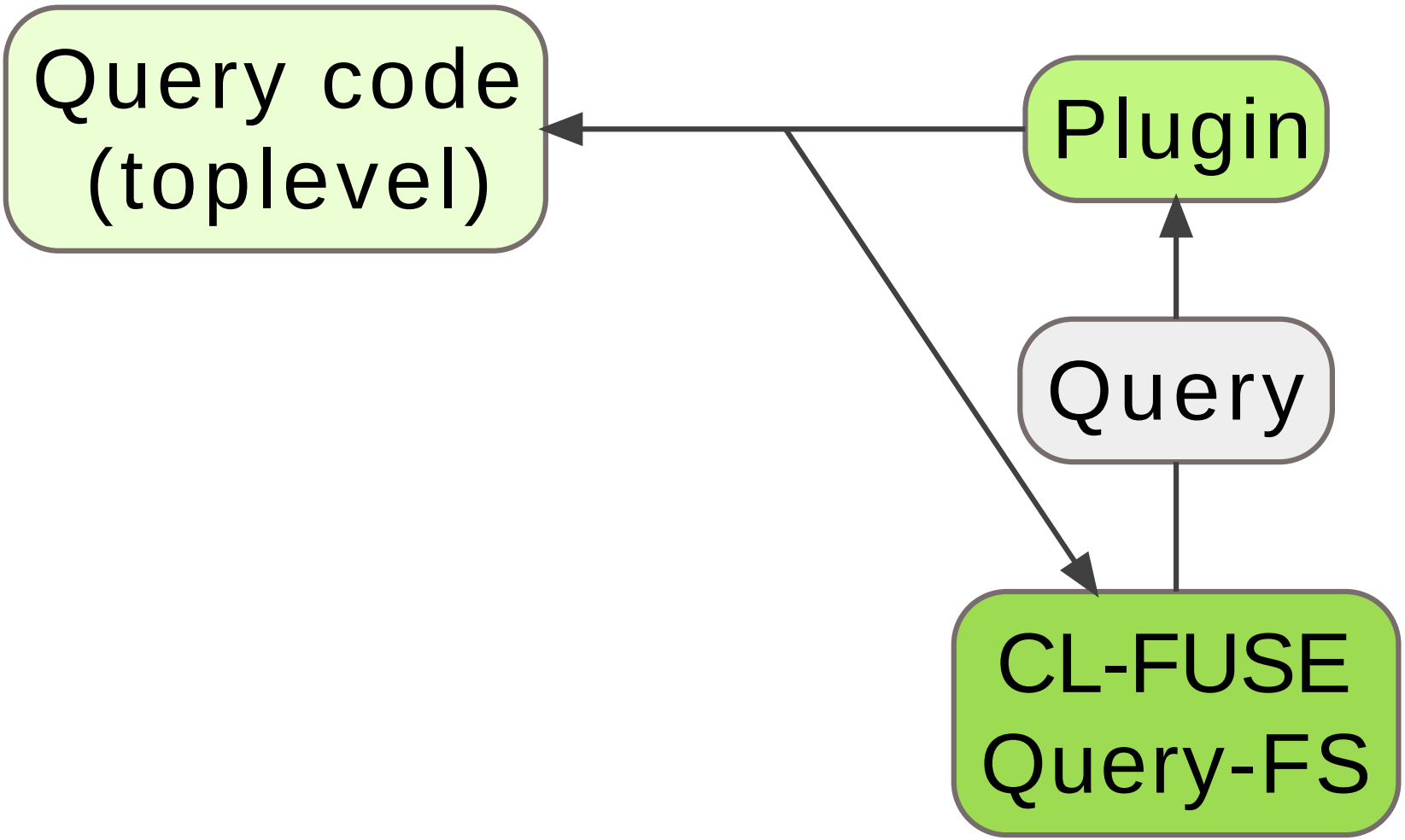
Query-FS request



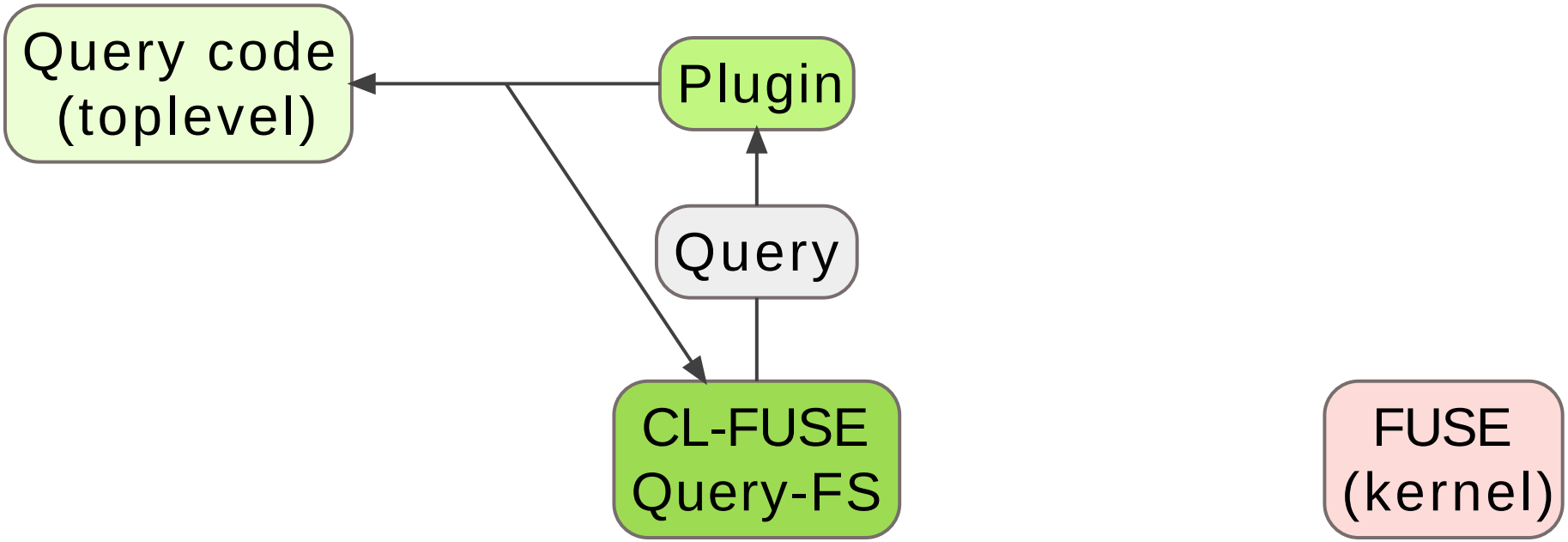
Query-FS request



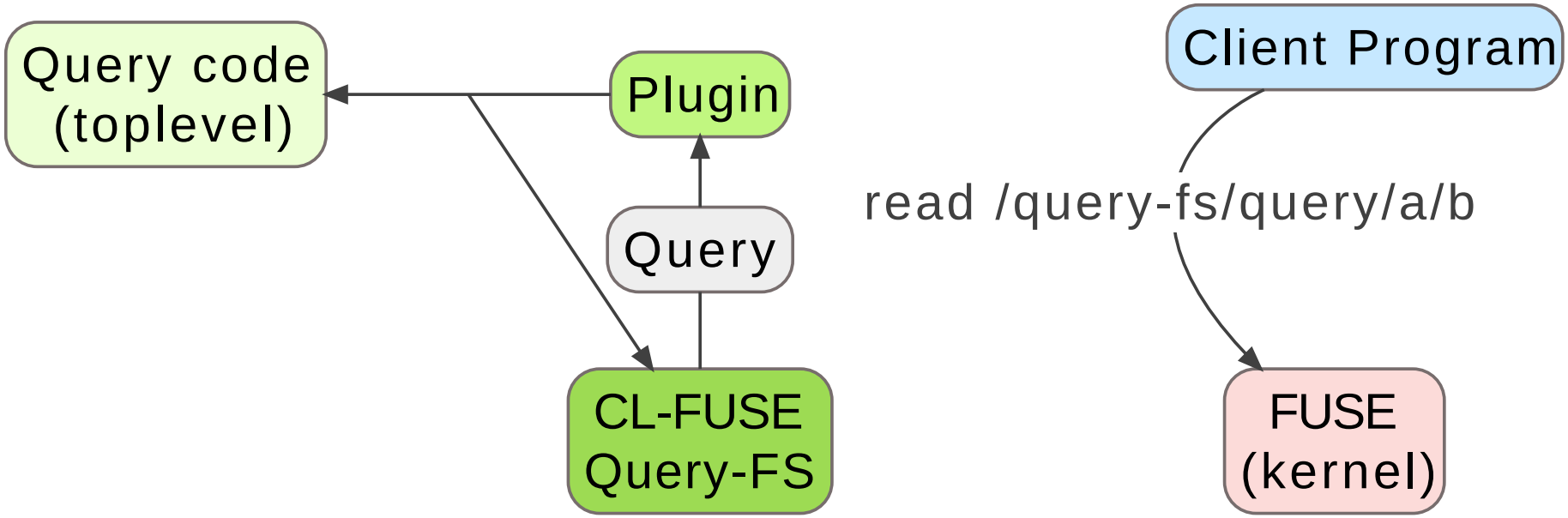
Query-FS request



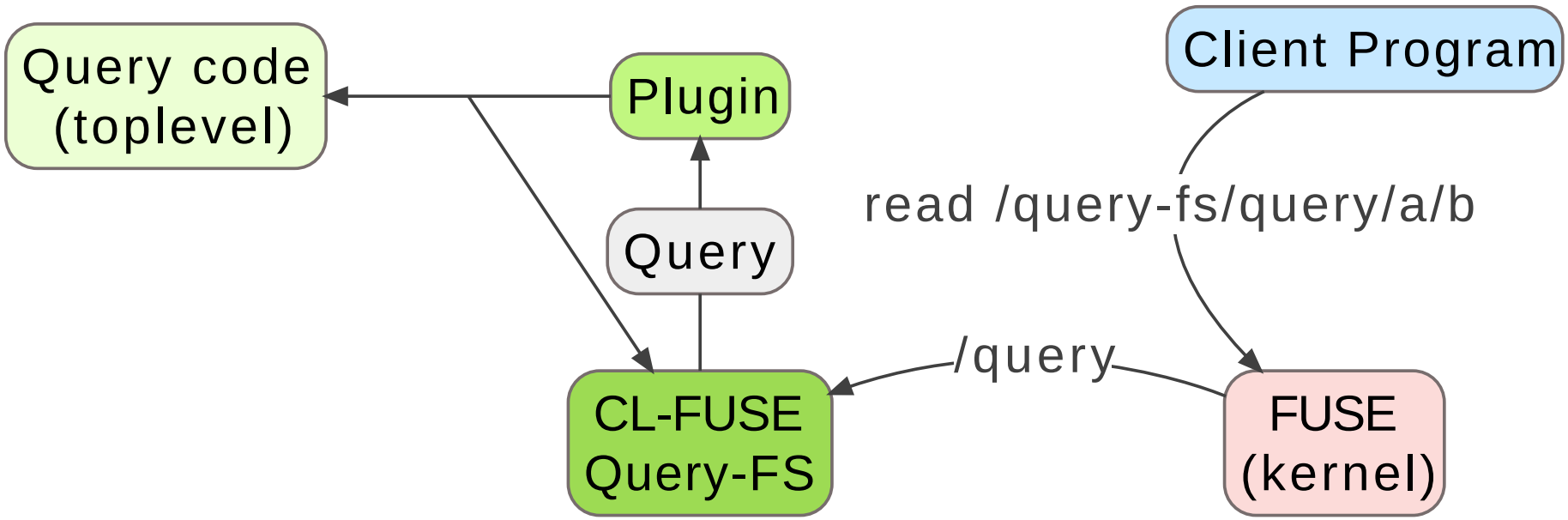
Query-FS request



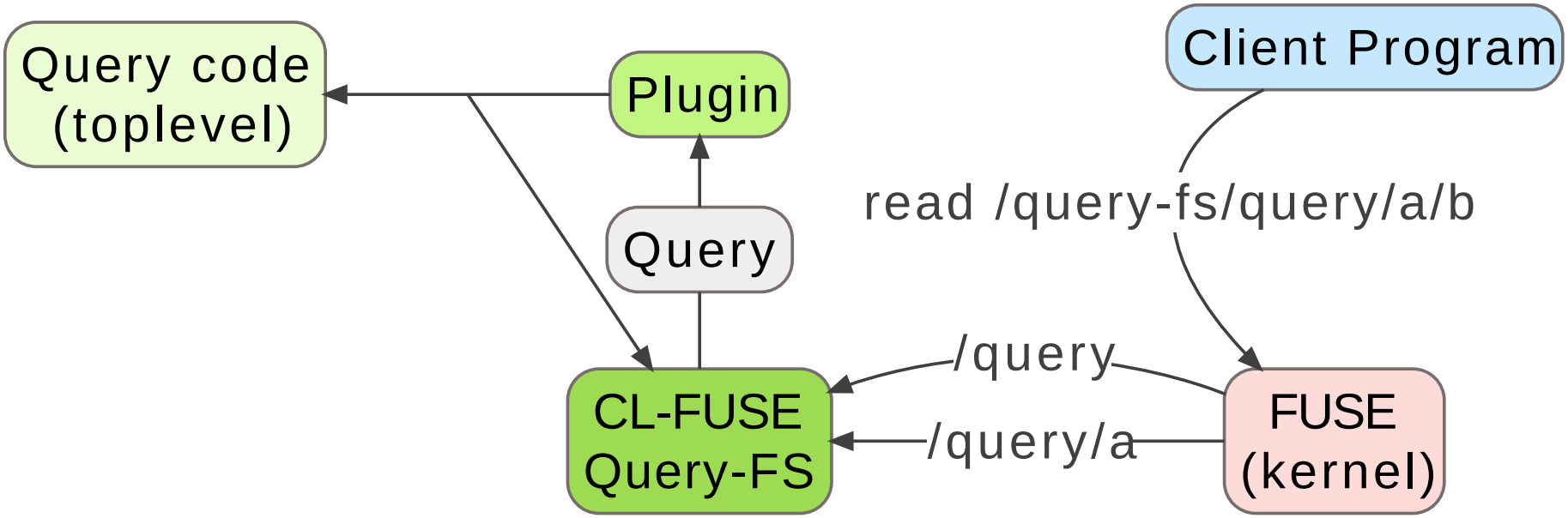
Query-FS request



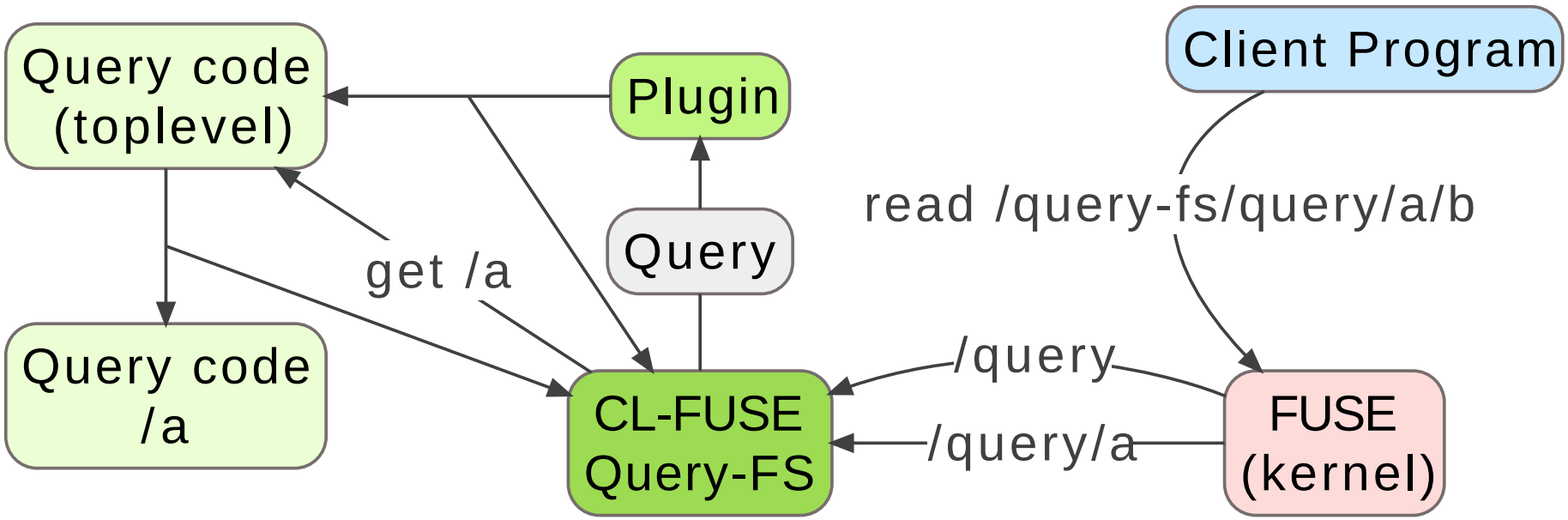
Query-FS request



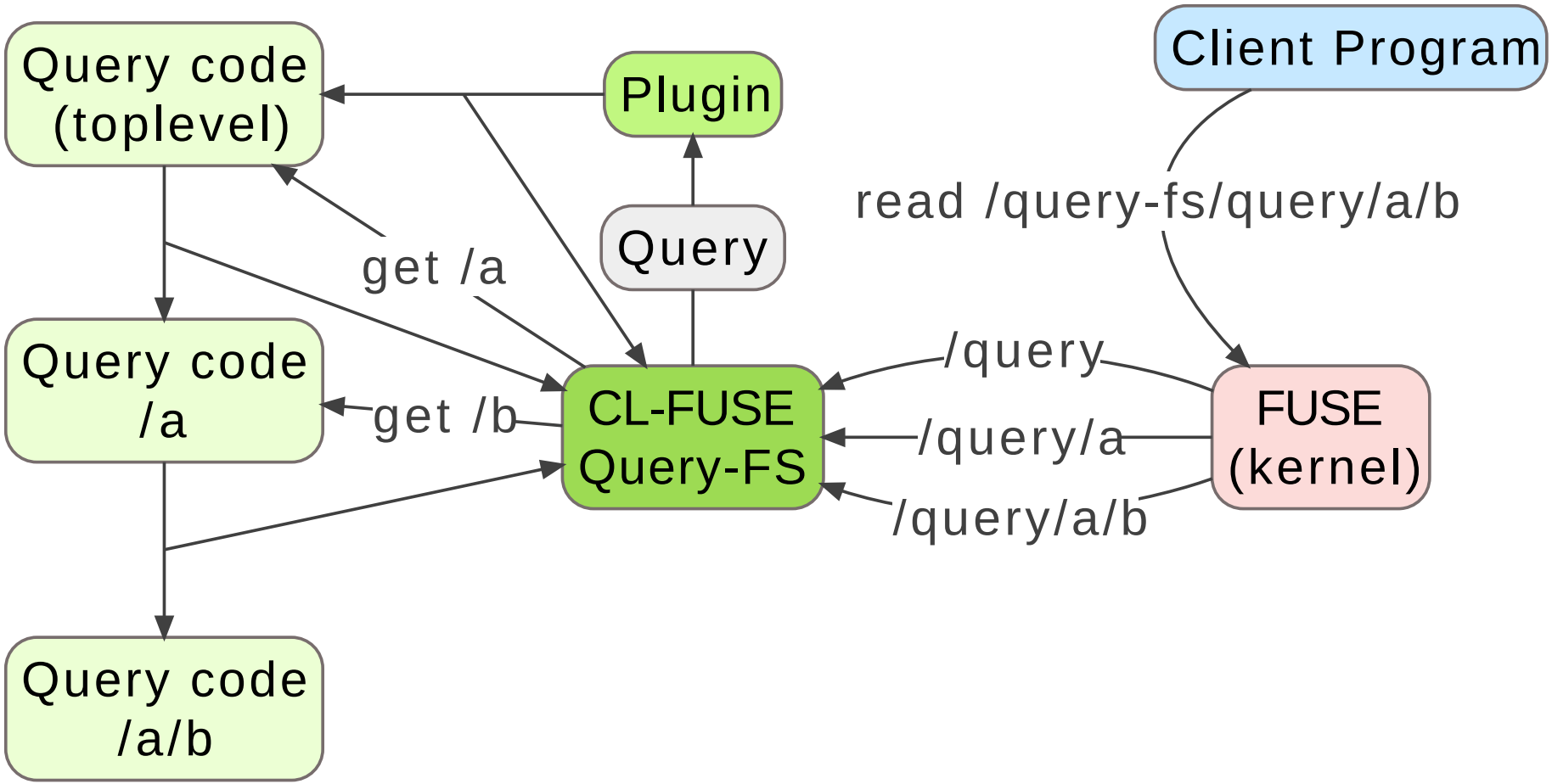
Query-FS request



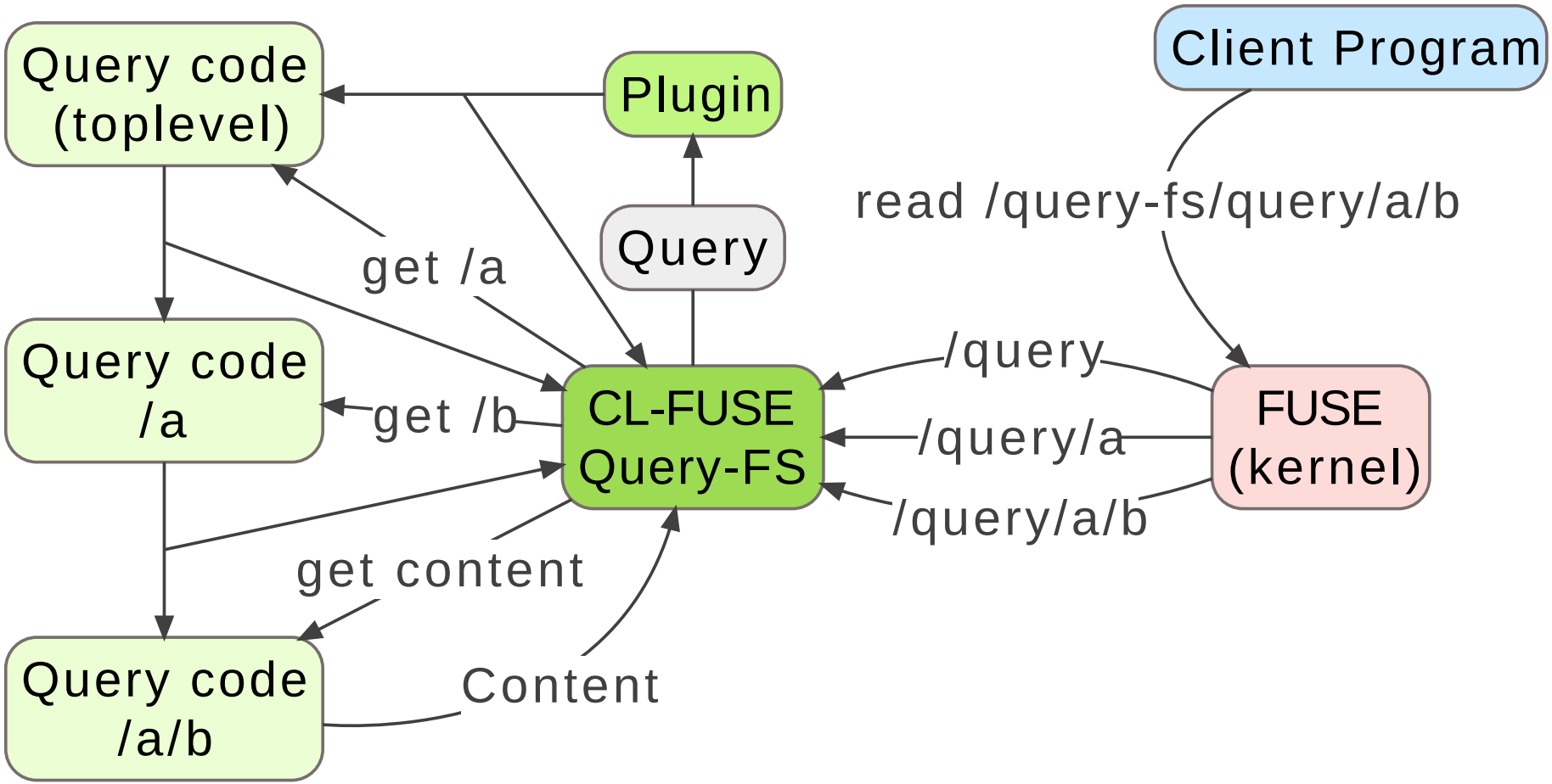
Query-FS request



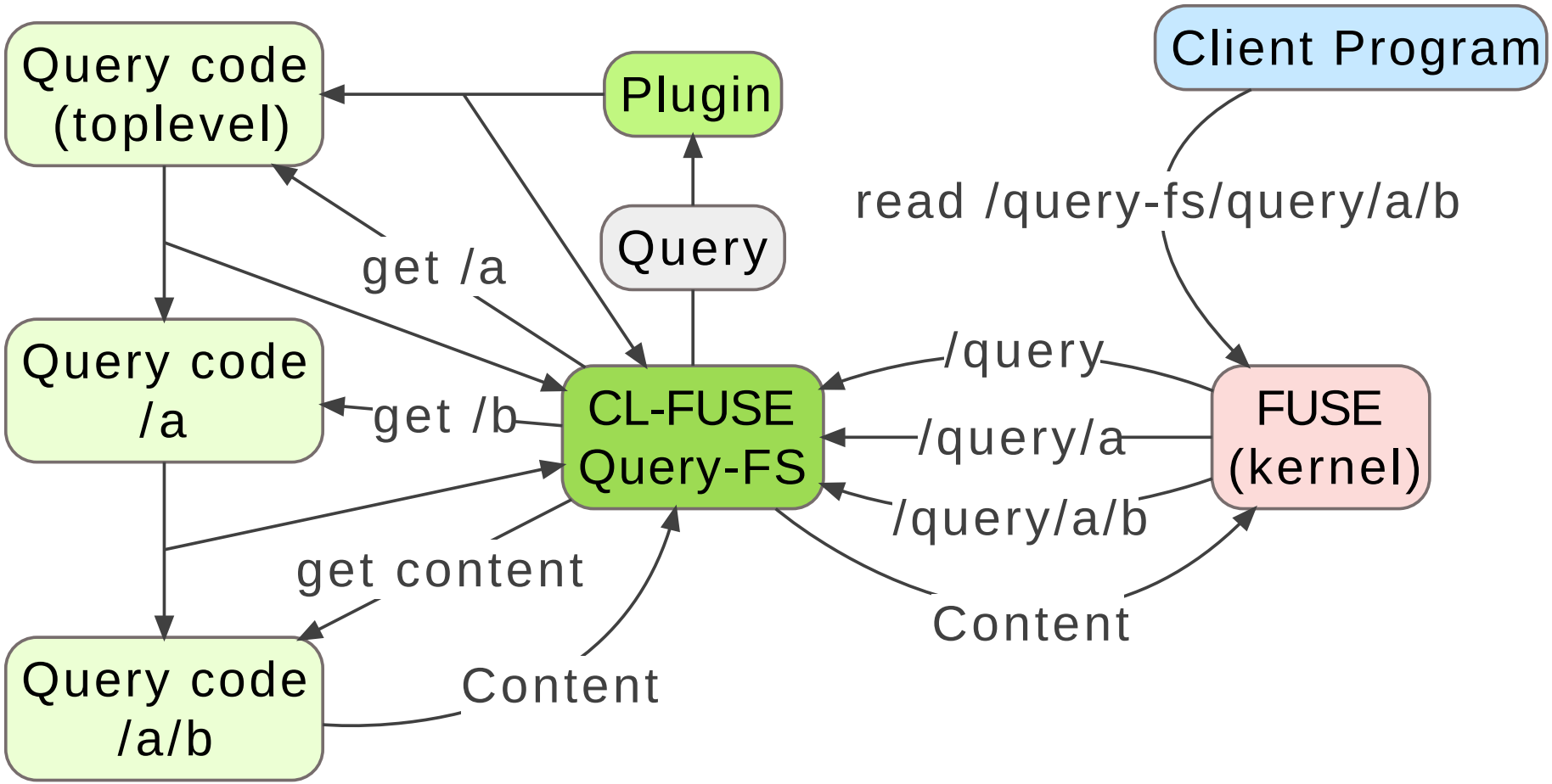
Query-FS request



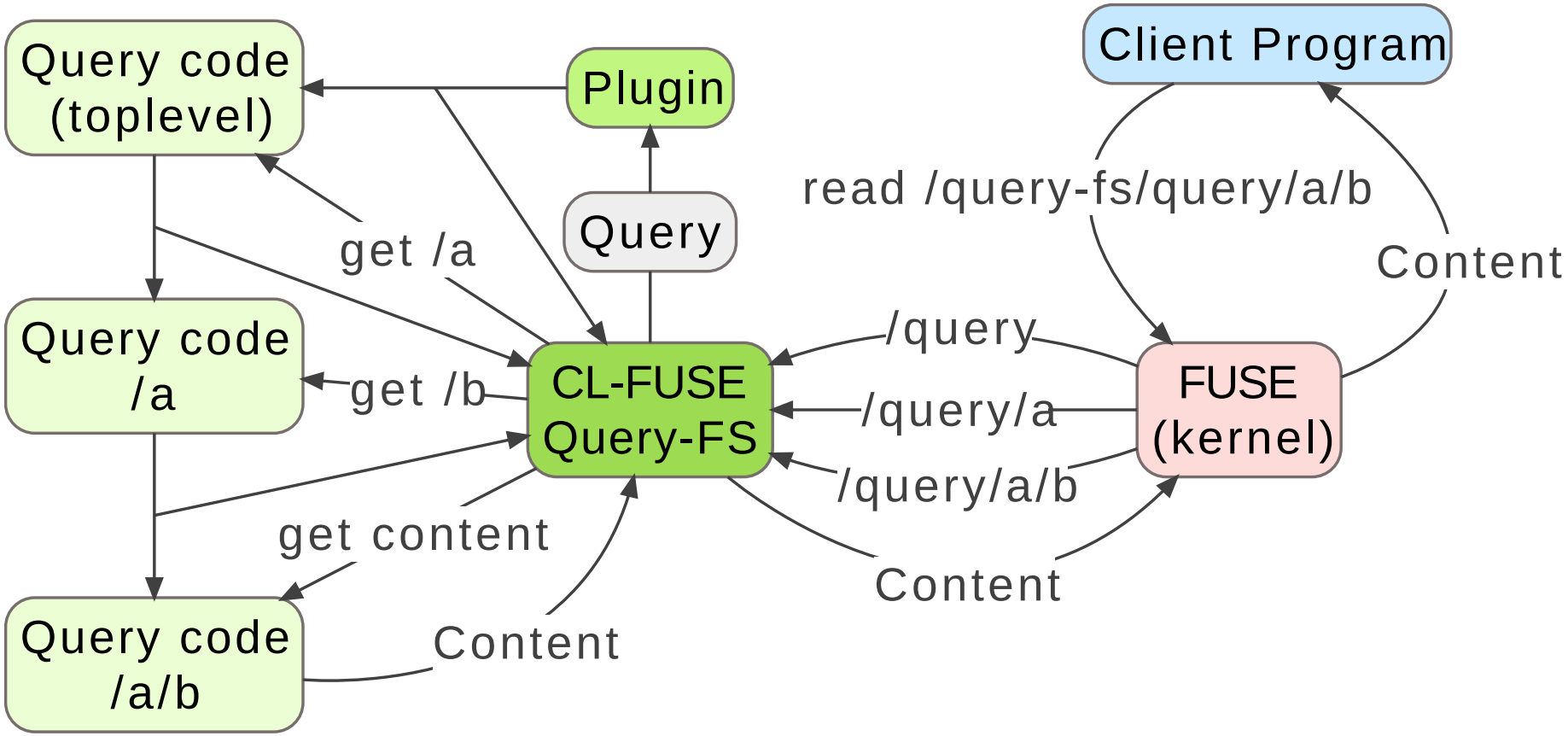
Query-FS request



Query-FS request



Query-FS request



Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on FS interface

Many of us value stability and flexibility

- C FFI integration needs support from both sides
- FFI is fragile: memory layout of arguments, ABIs
  that's before the unavoidable meaningful part
- In-process compatibility: threads, signals, allocations

- Maybe their file browsing/loading is enough?
  ...better chances than supporting HTTP just right
- Still need serialisation; at least breaking FS API is discouraged
- Different processes, diferent rules
- Well, some overhead; not so bad compared to SQL DB request
  for large files, generate symlinks to them

Remarks on implementation

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 implementation

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 implementation

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 implementation

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 implementation

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

Not there (yet?):

- Security model
  by kernel default only same UID allowed
- Advanced FS functionality: inotify, mmap, etc.
- Smarter caching based on lower-level FUSE API
- In-Lisp FUSE protocol parsing instead of libfuse
- 9p

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

Not there (yet?):

- Security model
  by kernel default only same UID allowed
- Advanced FS functionality: inotify, mmap, etc.
- Smarter caching based on lower-level FUSE API
- In-Lisp FUSE protocol parsing instead of libfuse
- 9p

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

Not there (yet?):

- Security model
  by kernel default only same UID allowed
- Advanced FS functionality: inotify, mmap, etc.
- Smarter caching based on lower-level FUSE API
- In-Lisp FUSE protocol parsing instead of libfuse
- 9p

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

Not there (yet?):

- Security model
  by kernel default only same UID allowed
- Advanced FS functionality: inotify, mmap, etc.
- Smarter caching based on lower-level FUSE API
- In-Lisp FUSE protocol parsing instead of libfuse
- 9p

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

Not there (yet?):

- Security model
  by kernel default only same UID allowed
- Advanced FS functionality: inotify, mmap, etc.
- Smarter caching based on lower-level FUSE API
- In-Lisp FUSE protocol parsing instead of libfuse
- 9p

Thanks!

Thanks for your attention!

Questions?

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