What Are All These Class Files Even About?

and Other Stories

by Gary Fredericks
at Clojure/conj 2018

About Me

- `@gfredericks_`
- DRW
- etc.

What Are We Doing?

Trying to understand the relationship between Clojure's bytecode and the dynamic runtime

Why Would We Do That?

- It's interesting to know how things work
- It's probably useful for faster debugging (stack traces might make more sense!)

Is This Hard?

Yes

- Clojure code can be compiled and loaded in lots of ways
- There's a parallel in-memory representation of things in the code
- Not even a clear line between compile-time and runtime

Caveats

- Lots of things are missing
- Some things are intentional lies
- Some things are accidental lies

Roadmap

- `require`
- `require`, but moreso
- `compile`
- `require` after `compile`
- `:reload`, etc.

Part 1: `require`

let's just load some code

Code

src/my/ns.clj

```
1  (ns my.ns
2    (:require
3     [my.other-ns :as other]))
4
5  (defn assoc-foo
6    [m]
7    (assoc m :foo other/foo))
```

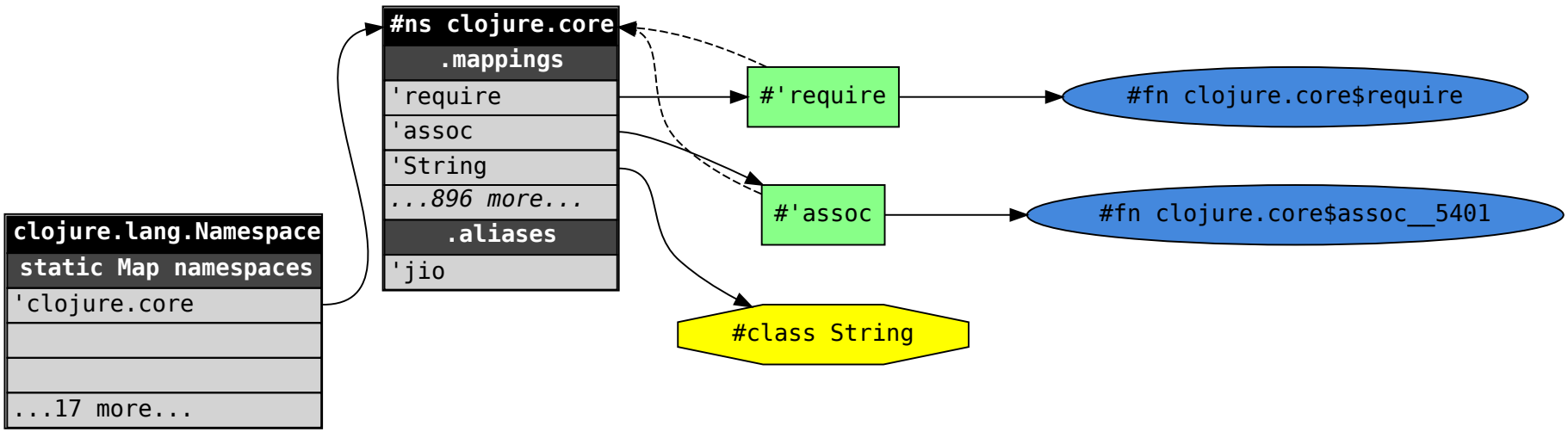
src/my/other_ns.clj

```
1  (ns my.other-ns)
2
3  (def foo "foo")
```

Let's start a repl

```
1 $ clj
2 Clojure 1.10.0-RC2
3 user=>
```

Namespace Graph



Require

```
1  user=> (require 'my.ns)
```

(ns my.ns ...)

```
1  (ns my.ns
2    (:require
3     [my.other-ns :as other]))
```

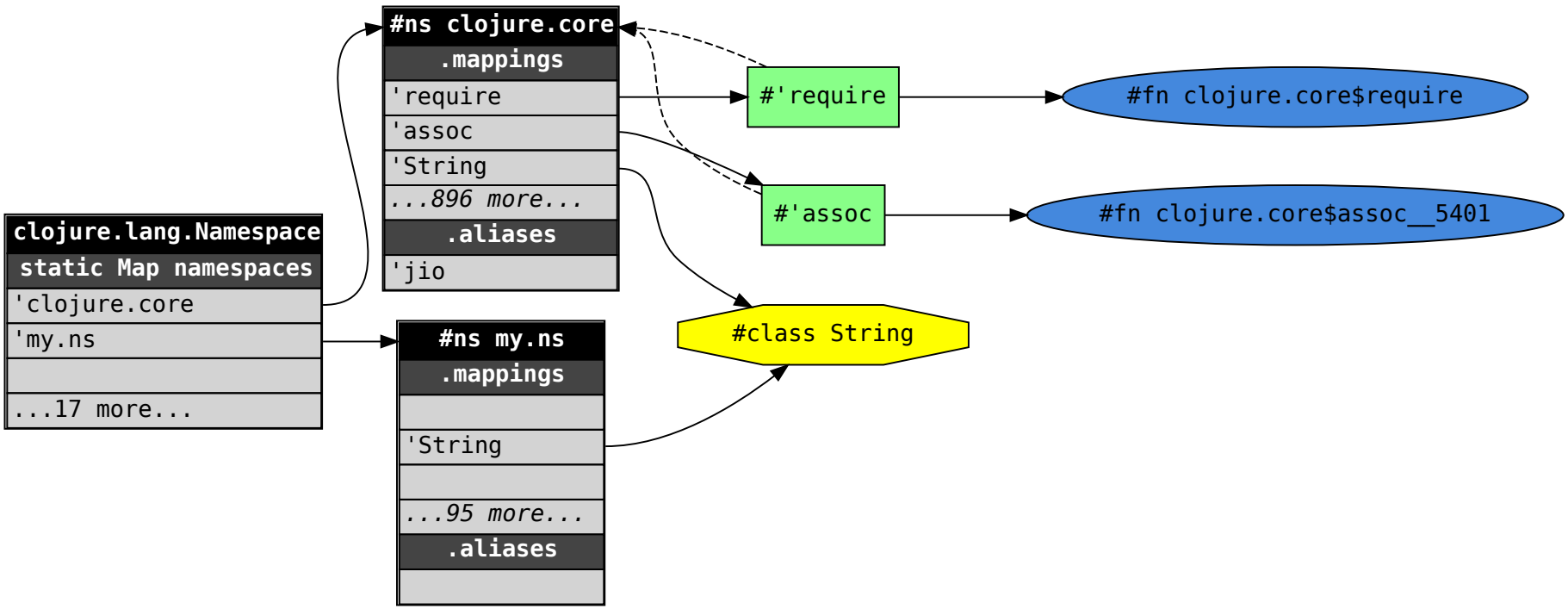
(macroexpand '(ns my.ns ...))

```clojure
;; mildly edited for readability
(do
  (in-ns 'my.ns)
  (with-loading-context
    (refer 'clojure.core)
    (require '[my.other-ns :as other]))
  (if (.equals 'my.ns 'clojure.core)
    nil
    (do
      (dosync
       (commute @#'*loaded-libs* conj 'my.ns))
      nil)))
```

(macroexpand '(ns my.ns ...))

```clojure
;; mildly edited for readability
(do
  ;; ------------
  (in-ns 'my.ns)
  ;; ^^^^^^^^^^^^
  (with-loading-context
    (refer 'clojure.core)
    (require '[my.other-ns :as other]))
  (if (.equals 'my.ns 'clojure.core)
    nil
    (do
      (dosync
       (commute @#'*loaded-libs* conj 'my.ns))
      nil)))
```

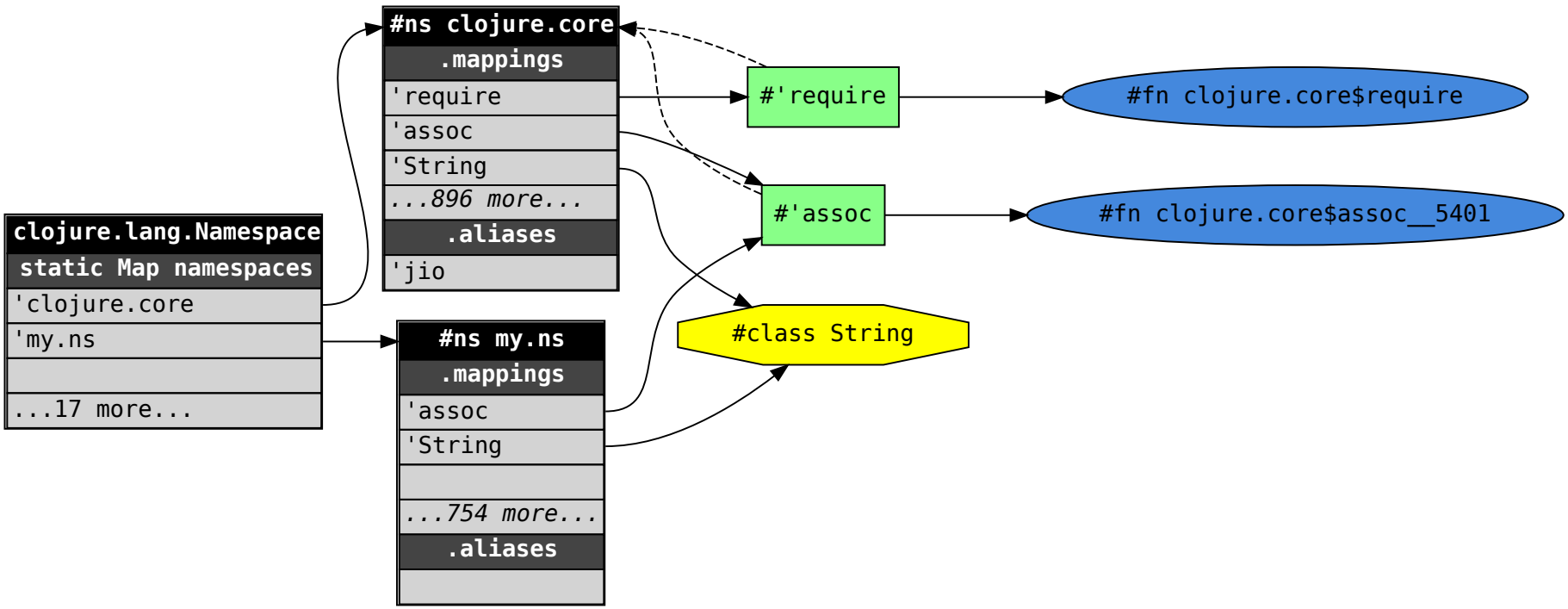
(in-ns 'my.ns)


#ns clojure.core
.mappings
'require
'assoc
'String
...896 more...
.aliases
'jio
clojure.lang.Namespace
static Map namespaces
'clojure.core
'my.ns
...17 more...
#ns my.ns
.mappings
'String
...95 more...
.aliases
#'require
#'assoc
#fn clojure.core$require
#fn clojure.core$assoc__5401
#class String

(macroexpand '(ns my.ns ...))

```
1  ;; mildly edited for readability
2  (do
3    (in-ns 'my.ns)
4    (with-loading-context
5      ;; ------------------
6      (refer 'clojure.core)
7      ;; ^^^^^^^^^^^^^^^^^^
8      (require '[my.other-ns :as other]))
9    (if (.equals 'my.ns 'clojure.core)
10     nil
11     (do
12       (dosync
13        (commute @#'*loaded-libs* conj 'my.ns))
14       nil)))
```

(refer 'clojure.core)



(macroexpand '(ns my.ns ...))

```
1  ;; mildly edited for readability
2  (do
3    (in-ns 'my.ns)
4    (with-loading-context
5      (refer 'clojure.core)
6      ;; ----------------------------------
7      (require '[my.other-ns :as other]))
8      ;; ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
9    (if (.equals 'my.ns 'clojure.core)
10     nil
11     (do
12       (dosync
13        (commute @#'*loaded-libs* conj 'my.ns))
14       nil)))
```

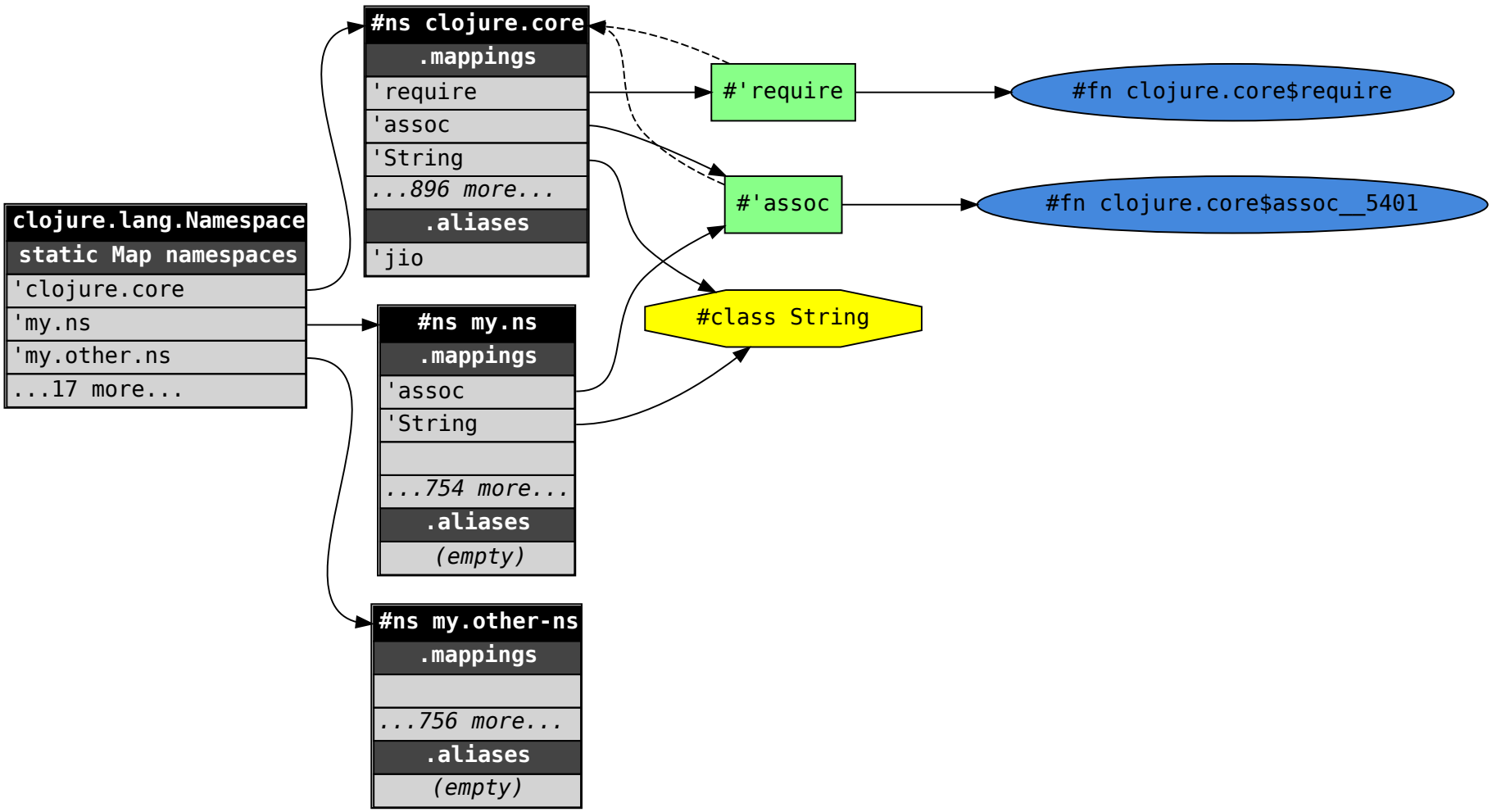
(ns other.ns ...)

```
1  (ns my.other-ns)
```

(macroexpand '(ns my.other-ns ...))

```clojure
;; mildly edited for readability
(do
  ;; ----------------------
  (in-ns 'my.other-ns)
  (with-loading-context
    (refer 'clojure.core))
  ;; ^^^^^^^^^^^^^^^^^^^^^^
  (if (.equals 'my.other-ns 'clojure.core)
    nil
    (do
      (dosync
       (commute @#'*loaded-libs* conj 'my.other-ns))
      nil)))
```

(in-ns 'my.other-ns) and (refer 'clojure.core)



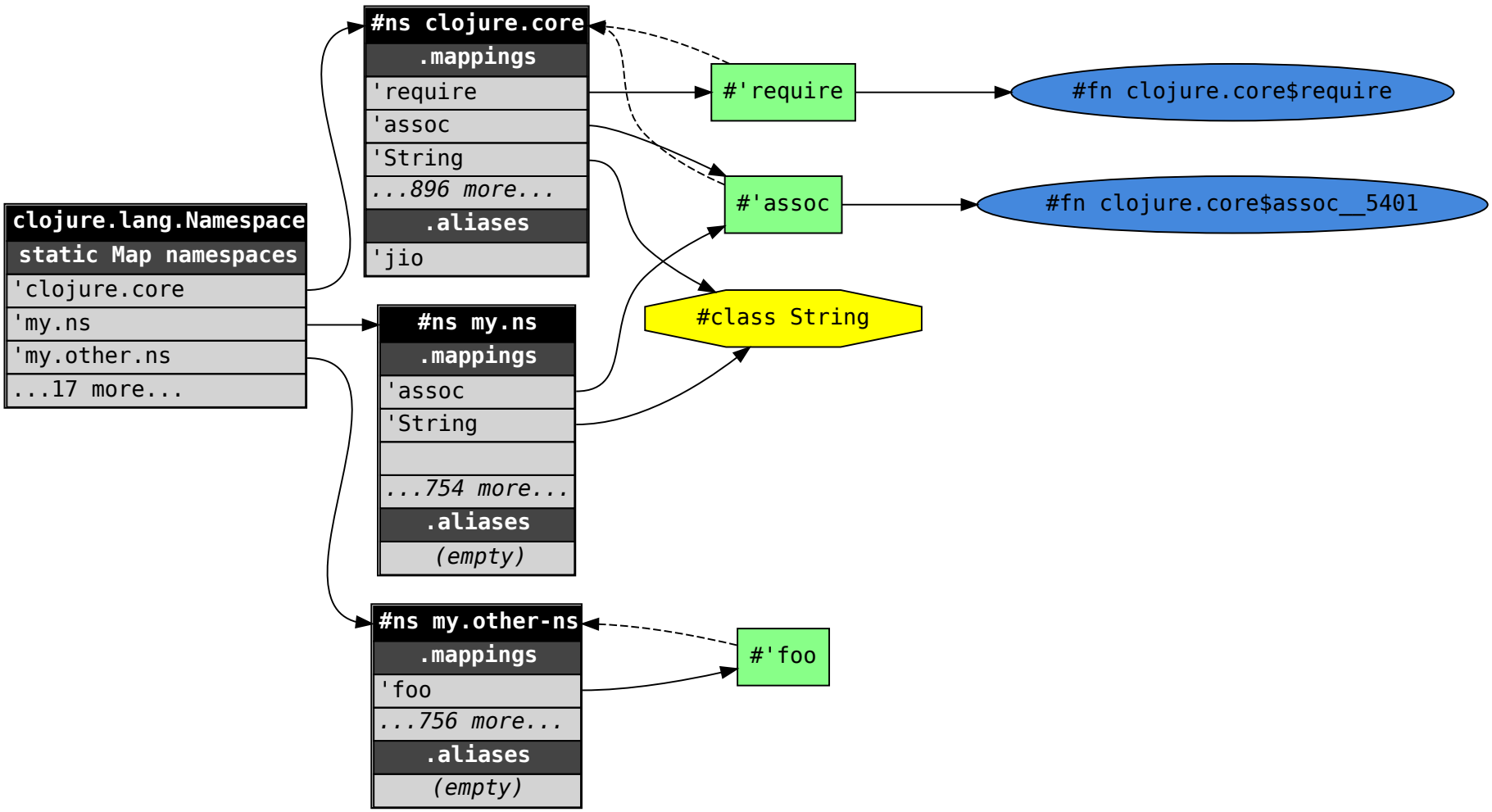
(macroexpand '(ns my.other-ns ...))

```
1  ;; mildly edited for readability
2  (do
3    (in-ns 'my.other-ns)
4    (with-loading-context
5      (refer 'clojure.core))
6    ;; ----------------------------------------
7    (if (.equals 'my.other-ns 'clojure.core)
8      nil
9      (do
10       (dosync
11        (commute @#'*loaded-libs* conj 'my.other-ns))
12       nil)))
13   ;; ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

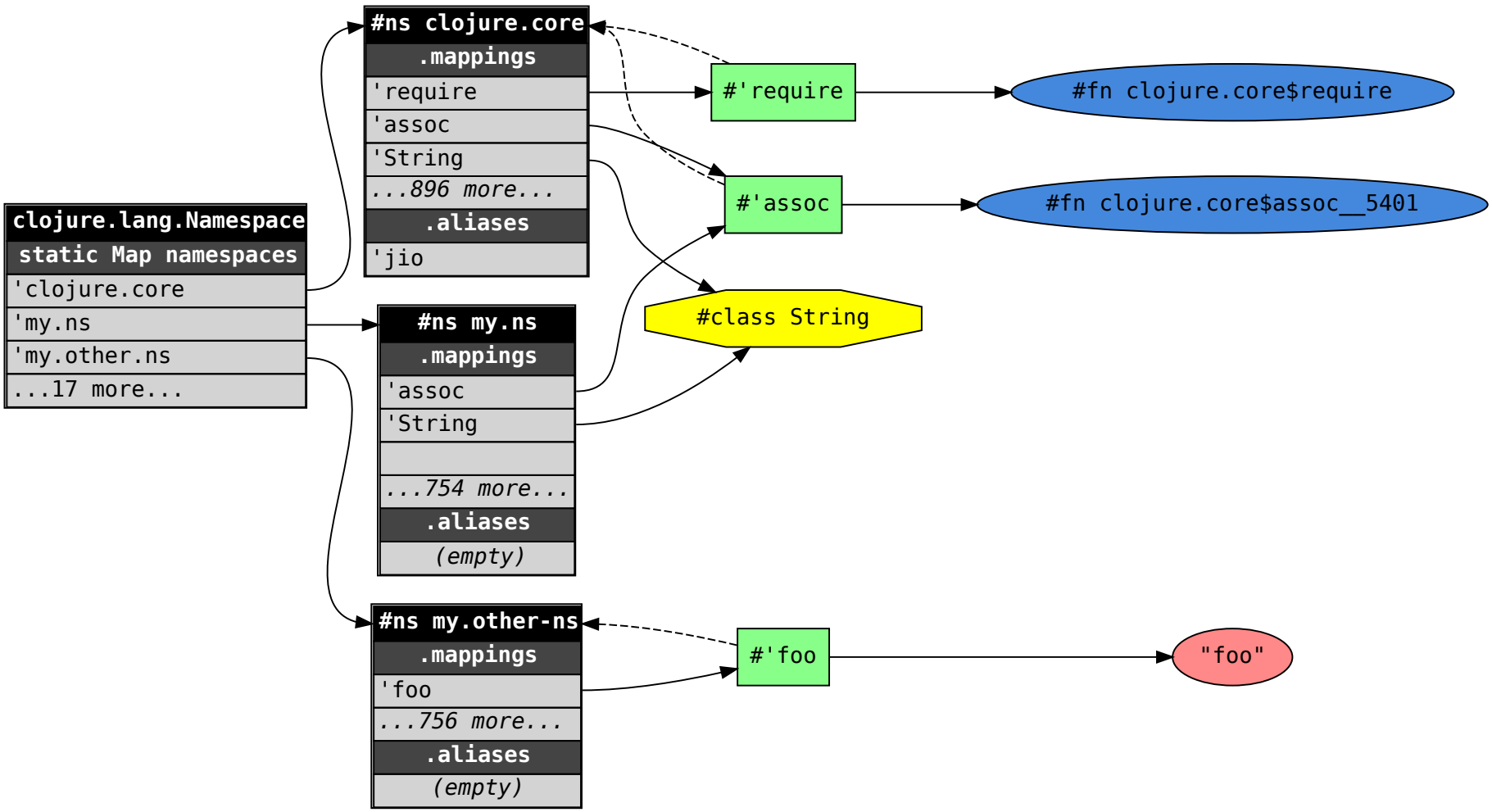
(def foo "foo")

```
1  (ns my.other-ns)
2
3  (def foo "foo")
```

(def foo "foo")


clojure.lang.Namespace
static Map namespaces
'clojure.core
'my.ns
'my.other.ns
...17 more...
#ns clojure.core
.mappings
'require
'assoc
'String
...896 more...
.aliases
'jio
#ns my.ns
.mappings
'assoc
'String
...754 more...
.aliases
(empty)
#ns my.other-ns
.mappings
'foo
...756 more...
.aliases
(empty)
#'require
#'assoc
#class String
#'foo
#fn clojure.core$require
#fn clojure.core$assoc__5401

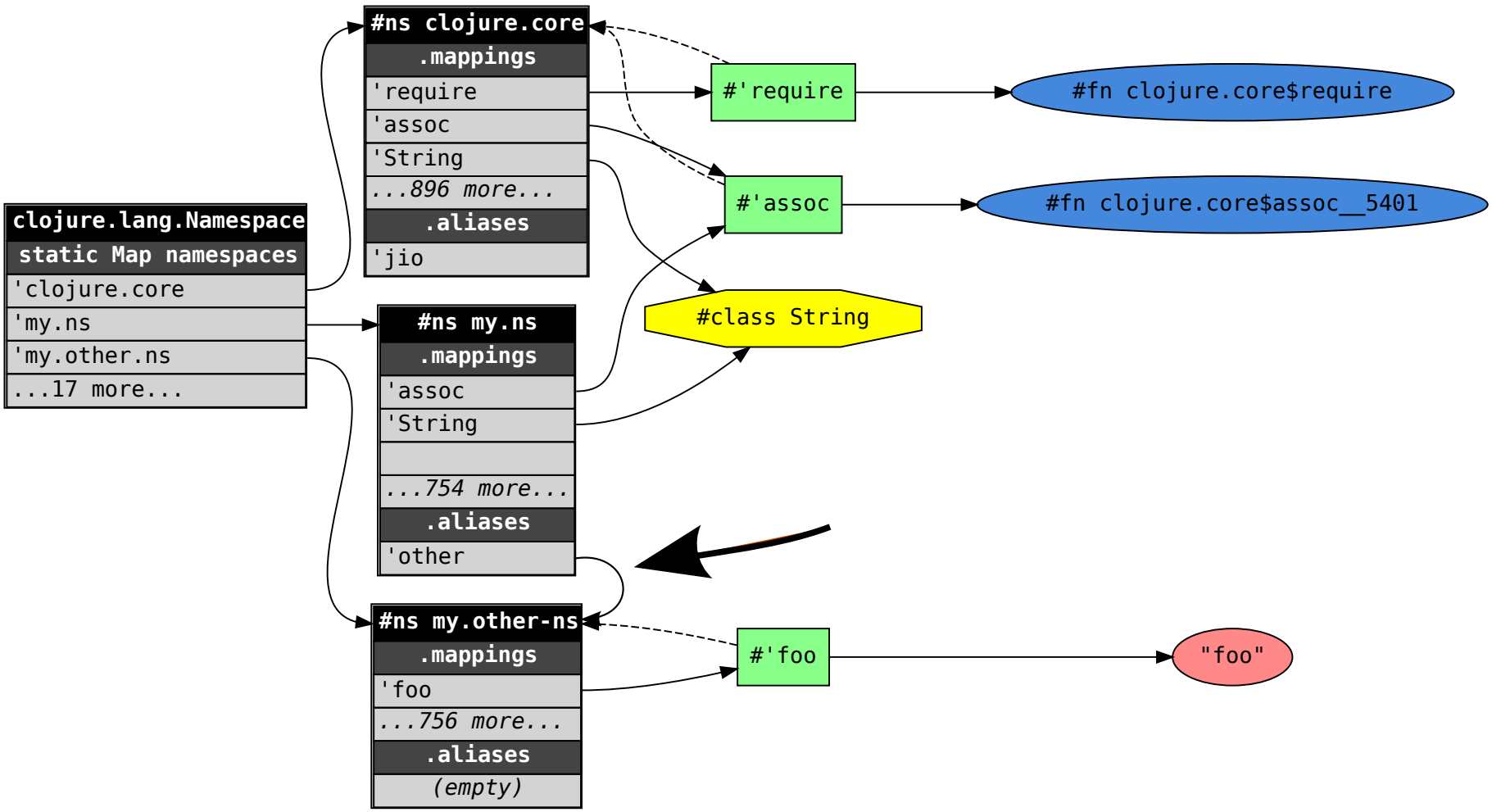
(def foo "foo")


clojure.lang.Namespace
static Map namespaces
'clojure.core
'my.ns
'my.other.ns
...17 more...
#ns clojure.core
.mappings
'require
'assoc
'String
...896 more...
.aliases
'jio
#ns my.ns
.mappings
'assoc
'String
...754 more...
.aliases
(empty)
#ns my.other-ns
.mappings
'foo
...756 more...
.aliases
(empty)
#'require
#'assoc
#class String
#'foo
#fn clojure.core$require
#fn clojure.core$assoc__5401
"foo"

(macroexpand '(ns my.ns ...))

```
1  ;; mildly edited for readability
2  (do
3    (in-ns 'my.ns)
4    (with-loading-context
5      (refer 'clojure.core)
6      ;; ----------------------------------
7      (require '[my.other-ns :as other]))
8      ;; ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
9    (if (.equals 'my.ns 'clojure.core)
10     nil
11     (do
12       (dosync
13        (commute @#'*loaded-libs* conj 'my.ns))
14       nil)))
```

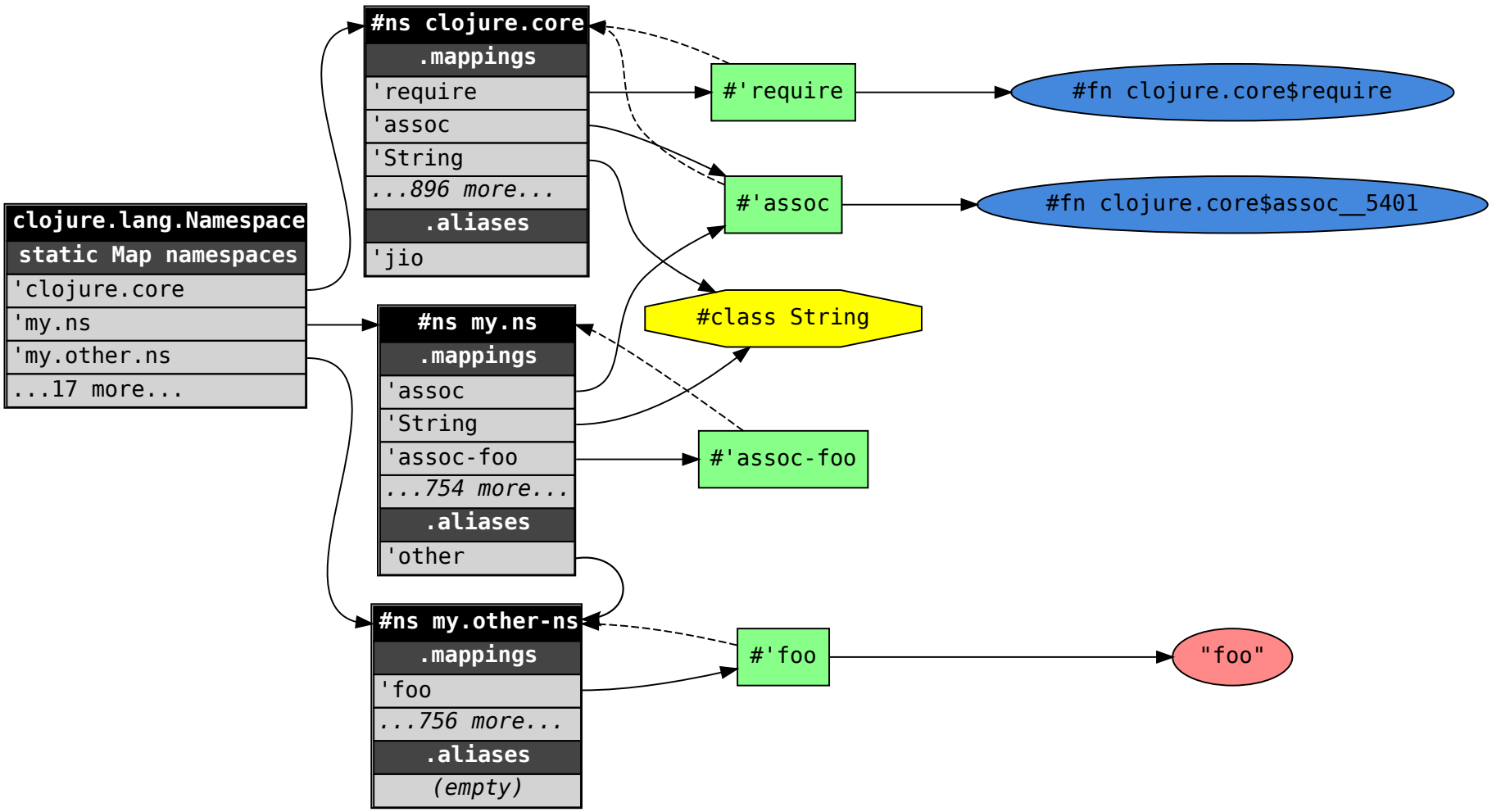
:as other


clojure.lang.Namespace
static Map namespaces
'clojure.core
'my.ns
'my.other.ns
...17 more...
#ns clojure.core
.mappings
'require
'assoc
'String
...896 more...
.aliases
'jio
#ns my.ns
.mappings
'assoc
'String
...754 more...
.aliases
'other
#ns my.other-ns
.mappings
'foo
...756 more...
.aliases
(empty)
#'require
#'assoc
#class String
#'foo
#fn clojure.core$require
#fn clojure.core$assoc__5401
"foo"

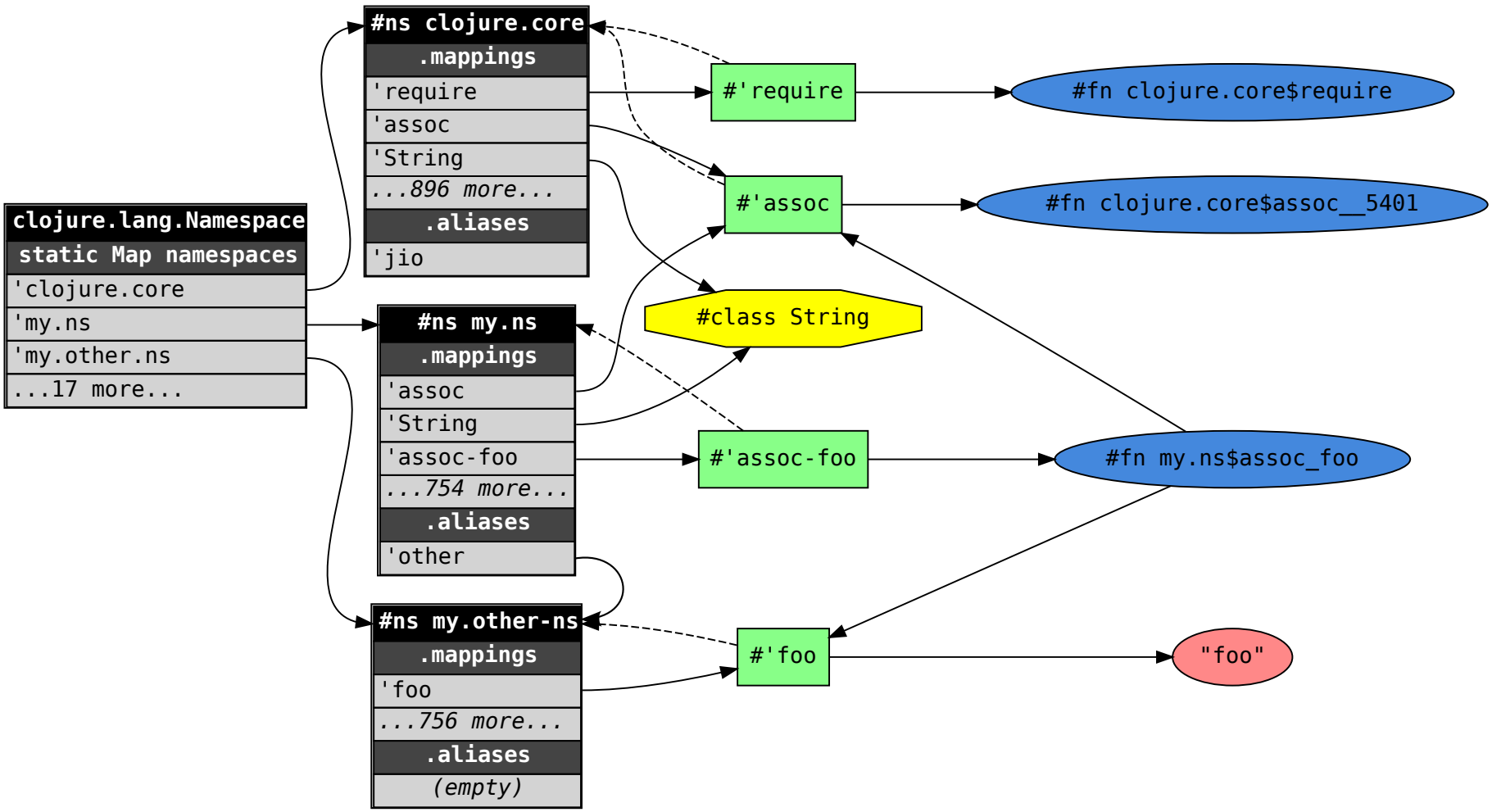
(defn assoc-foo ...)

```
1  (ns my.ns
2    (:require
3     [my.other-ns :as other]))
4
5  (defn assoc-foo
6    [m]
7    (assoc m :foo other/foo))
```

(defn assoc-foo ...)



(defn assoc-foo ...)



So what was all that then?

All top-level forms in clojure code are for side effects, often in the namespace graph

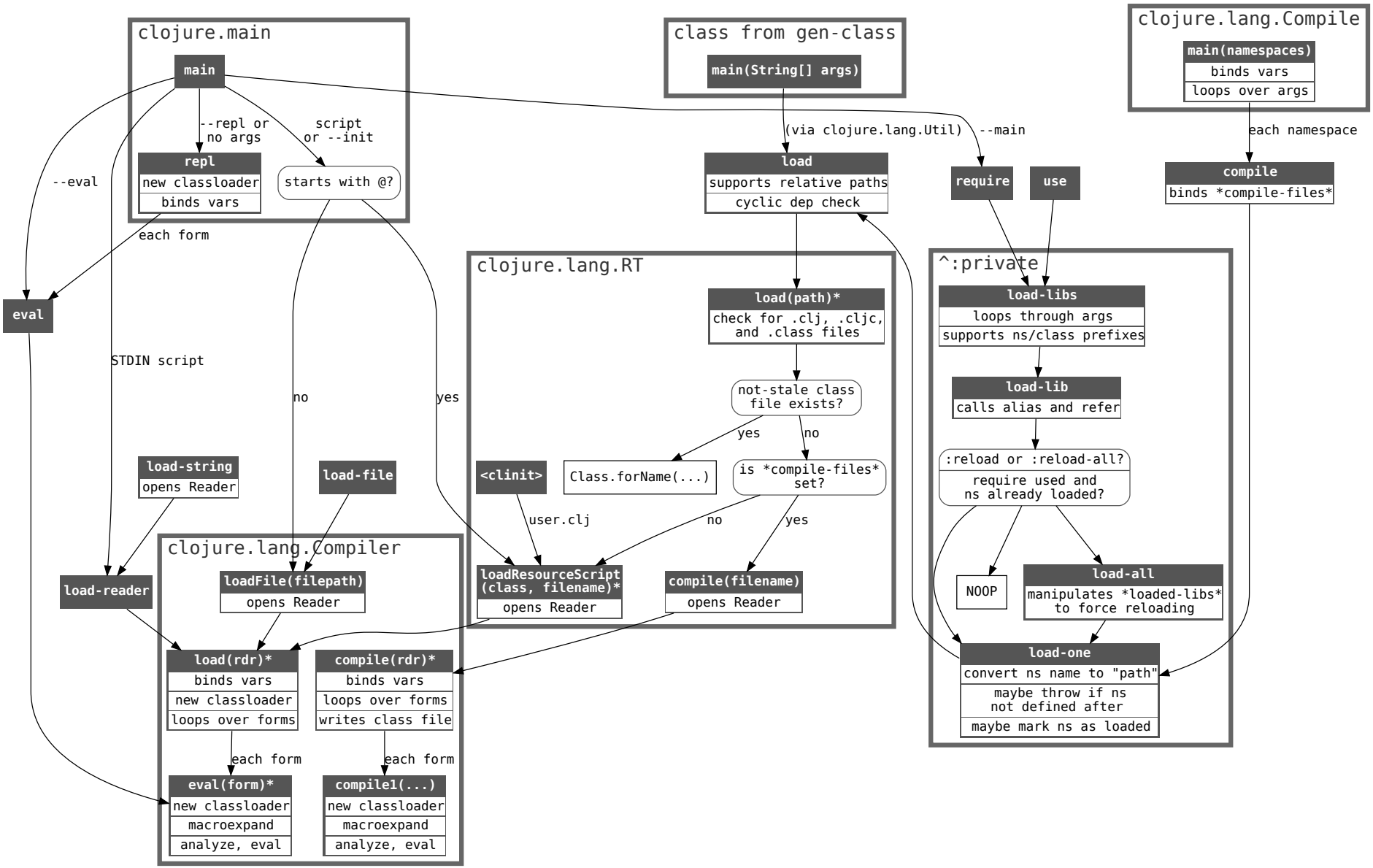
Part 2: `require`, but moreso

where was the bytecode there?

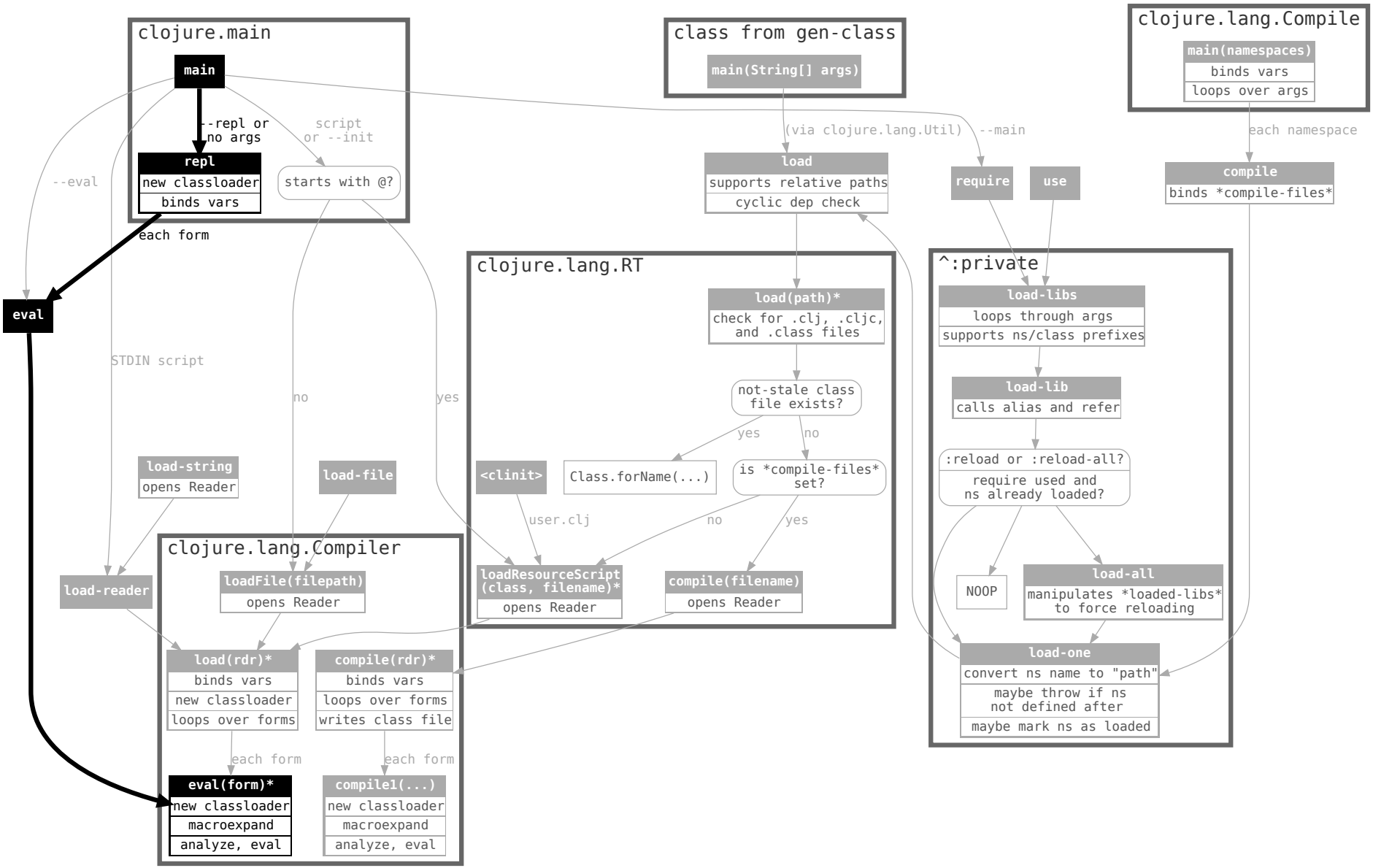
The Repl

```
1 $ clj
2 Clojure 1.10.0-RC2
3 user=> (require 'my.ns)
```

The API


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
load-string
opens Reader
load-file
load-reader
no
yes
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

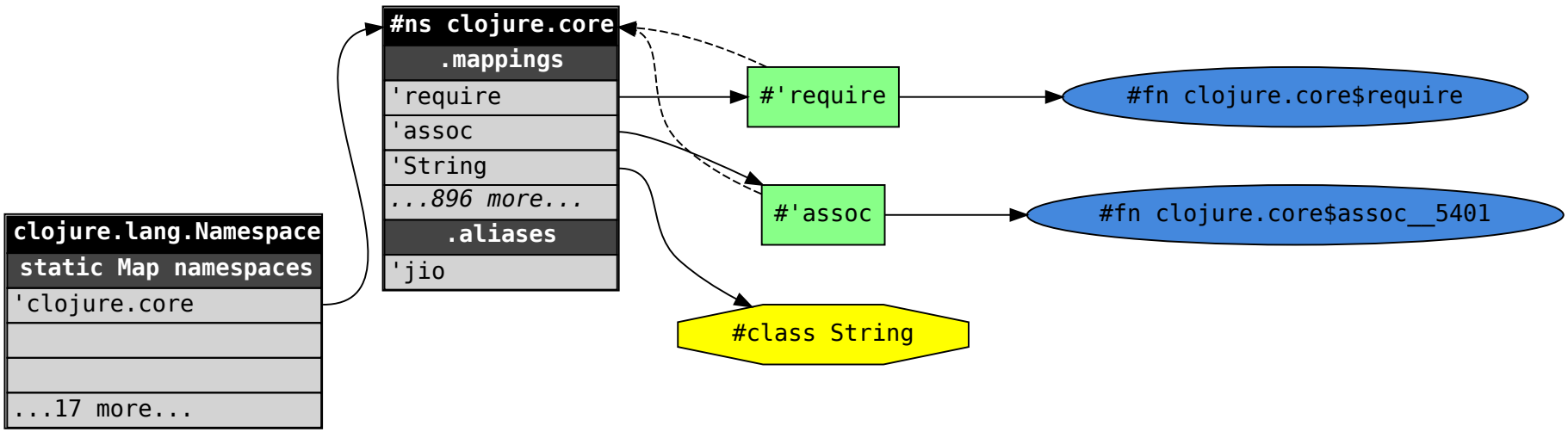
main -> repl


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
no
yes
load-string
opens Reader
load-file
load-reader
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

(require 'my.ns) as a compiled class

```
1  public final class user$eval1
2      extends clojure.lang.AFunction {
3    // #'clojure.core/require
4    public static final clojure.lang.Var const__0;
5    // 'my.ns
6    public static final clojure.lang.AFn const__1;
7    // const__0 = RT.var("clojure.core","require");
8    // const__1 = Symbol.intern(null,"my.ns");
9    public static {};
10   // super()
11   public user$eval1();
12   // const__0.getRawRoot().invoke(const__1);
13   public static java.lang.Object invokeStatic();
14   // call .invokeStatic()
15   public java.lang.Object invoke();
16 }
```

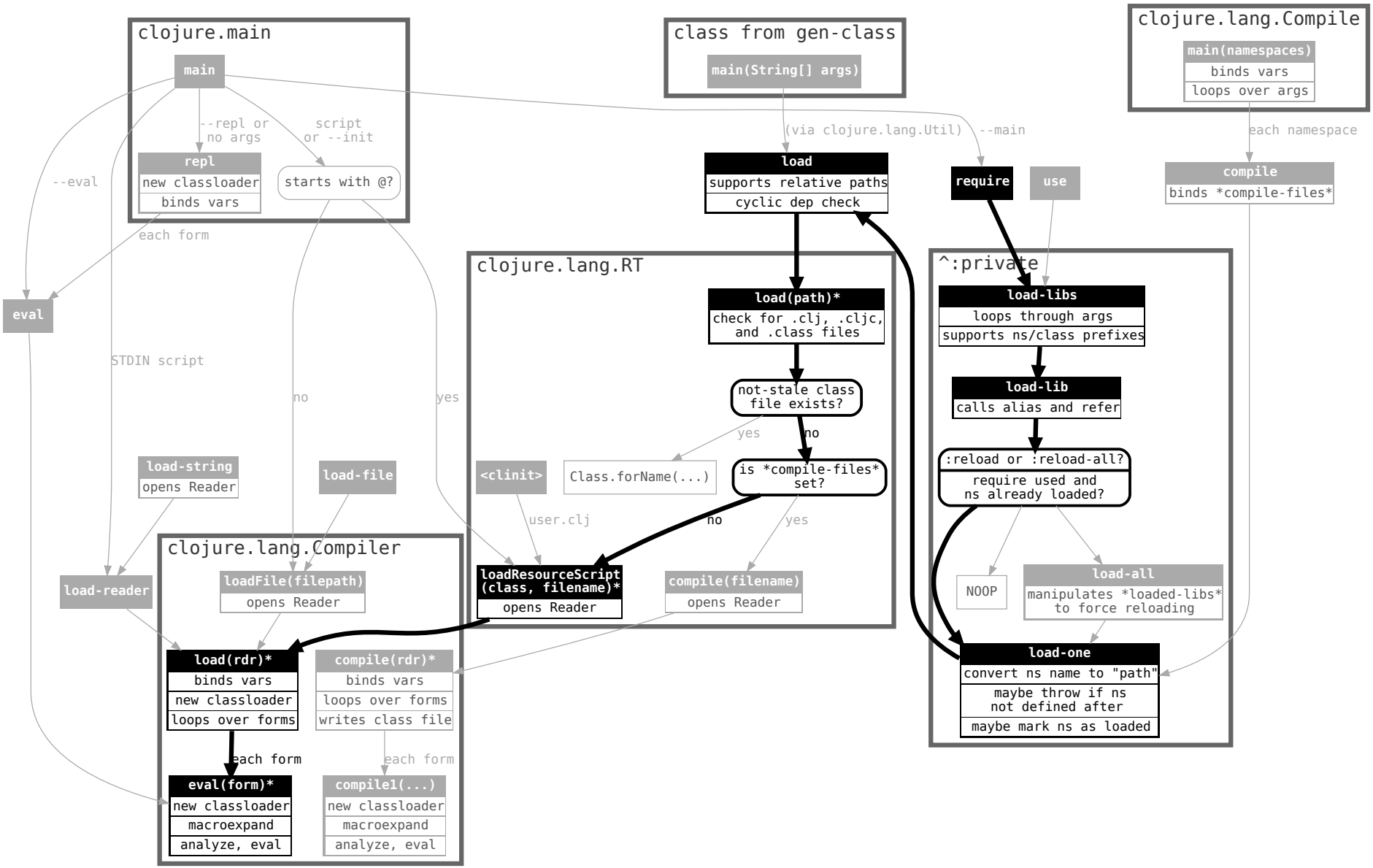
RT.var("clojure.core","require");



Classloading

```
1  package clojure.lang;
2
3  import java.net.URLClassLoader;
4  // ...etc.
5
6  public class DynamicClassLoader
7    extends URLClassLoader{
8
9    static ConcurrentHashMap<String, Reference<Class>>
10     classCache =
11         new ConcurrentHashMap<String, Reference<Class>>();
12
13
14   // ... lots of overridden methods
15 }
```

require


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
no
yes
load-string
opens Reader
load-file
load-reader
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

Evaling src/my/ns.clj

```
1  (ns my.ns
2    (:require
3     [my.other-ns :as other]))
4
5  (defn assoc-foo
6    [m]
7    (assoc m :foo other/foo))
```

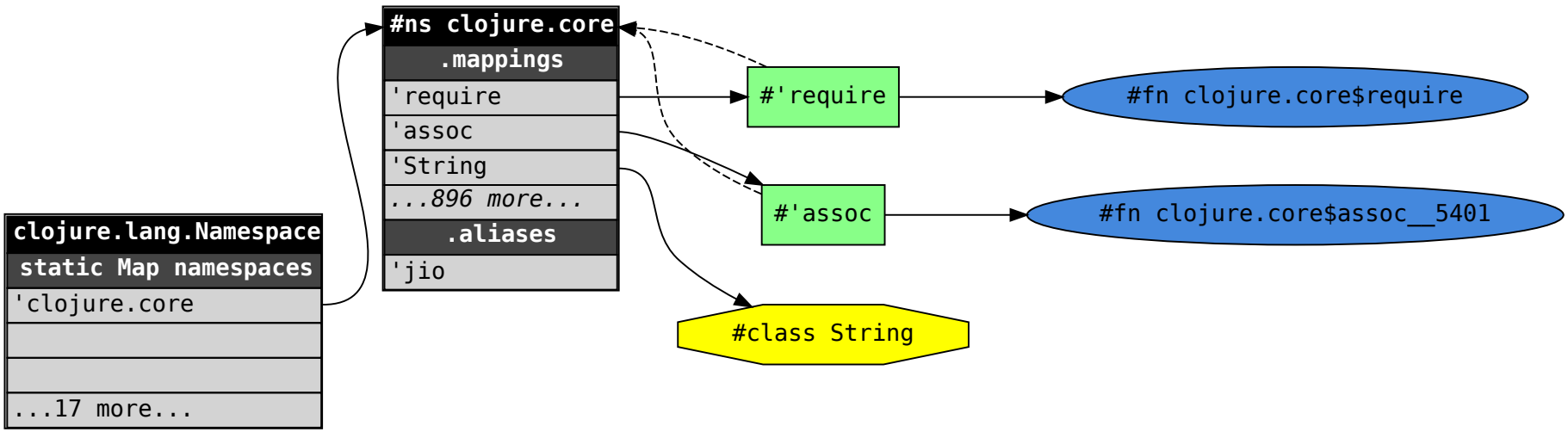
Further macroexpand (ns my.ns ...)

```
1  ;; mildly edited for readability
2  (do
3    (in-ns 'my.ns)
4    ((fn loading__6621__auto__ []
5       (clojure.lang.Var/pushThreadBindings
6        {clojure.lang.Compiler/LOADER
7         (.getClassLoader
8          (.getClass loading__6621__auto__))})
9       (try
10        (refer 'clojure.core)
11        (finally
12          (clojure.lang.Var/popThreadBindings)))))
13   (if (.equals 'my.ns 'clojure.core)
14     nil
15     (do
16       (clojure.lang.LockingTransaction/runInTransaction
17        (fn [] (commute @#'*loaded-libs* conj 'my.ns)))
18       nil)))
```

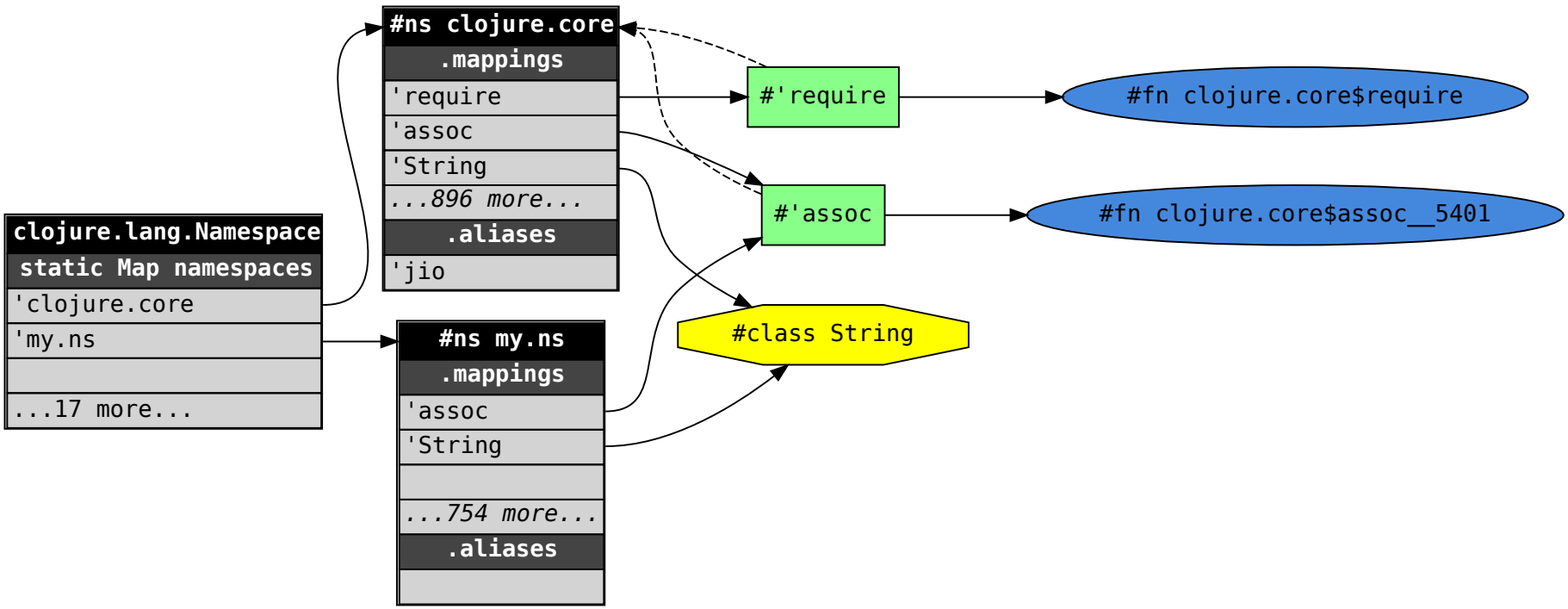
Functions calling functions

```
1  public class my.ns$eval154
2    extends clojure.lang.AFunction {
3    // ...
4    public static Object invokeStatic(){
5      if(!clojureCoreSym.equals(myNsSym)){
6        LockingTransaction.runInTransaction(
7          new my.ns$eval154$fn__155();
8        );
9      }
10   }
11 }
```

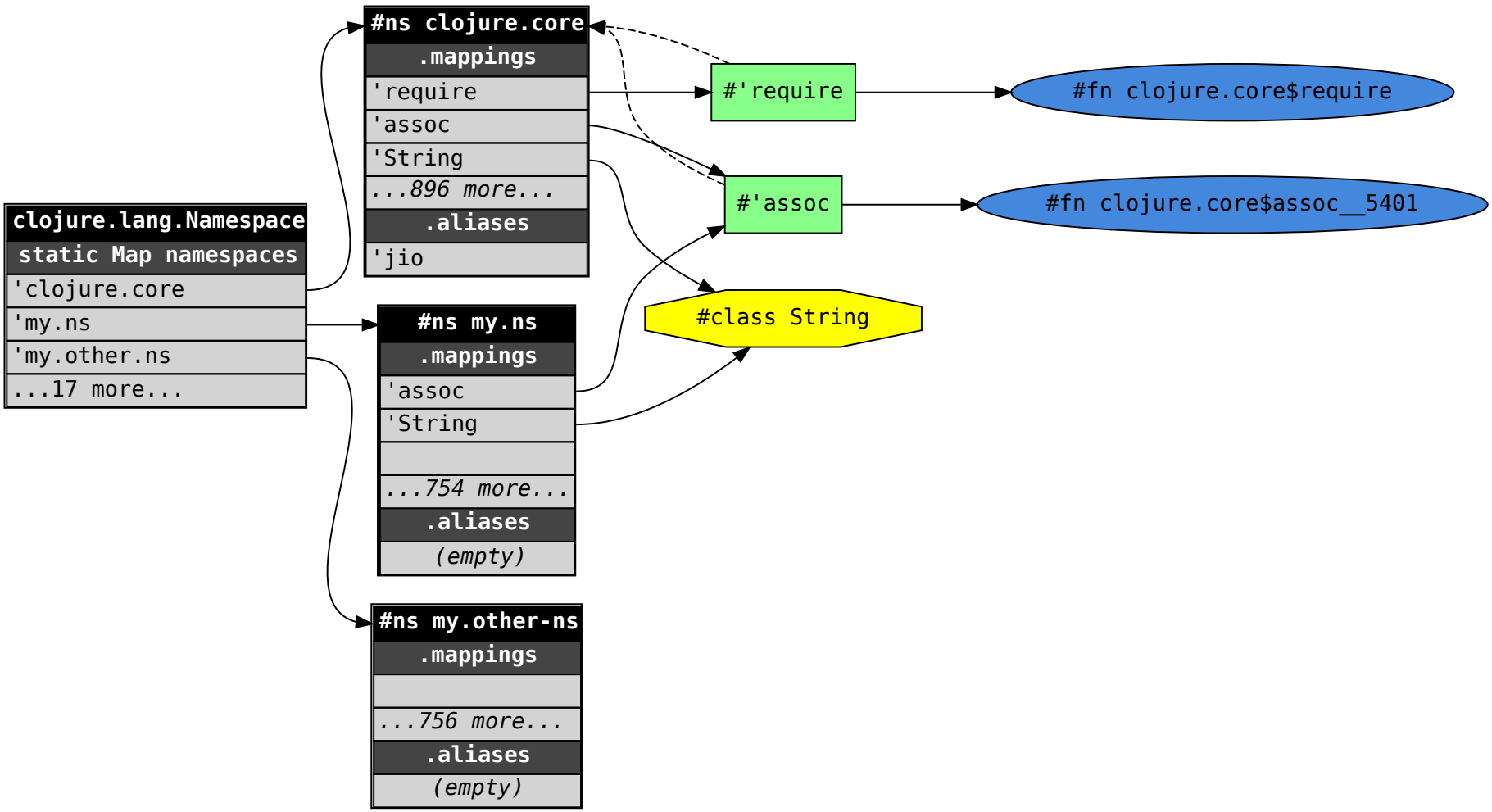
Side Effects



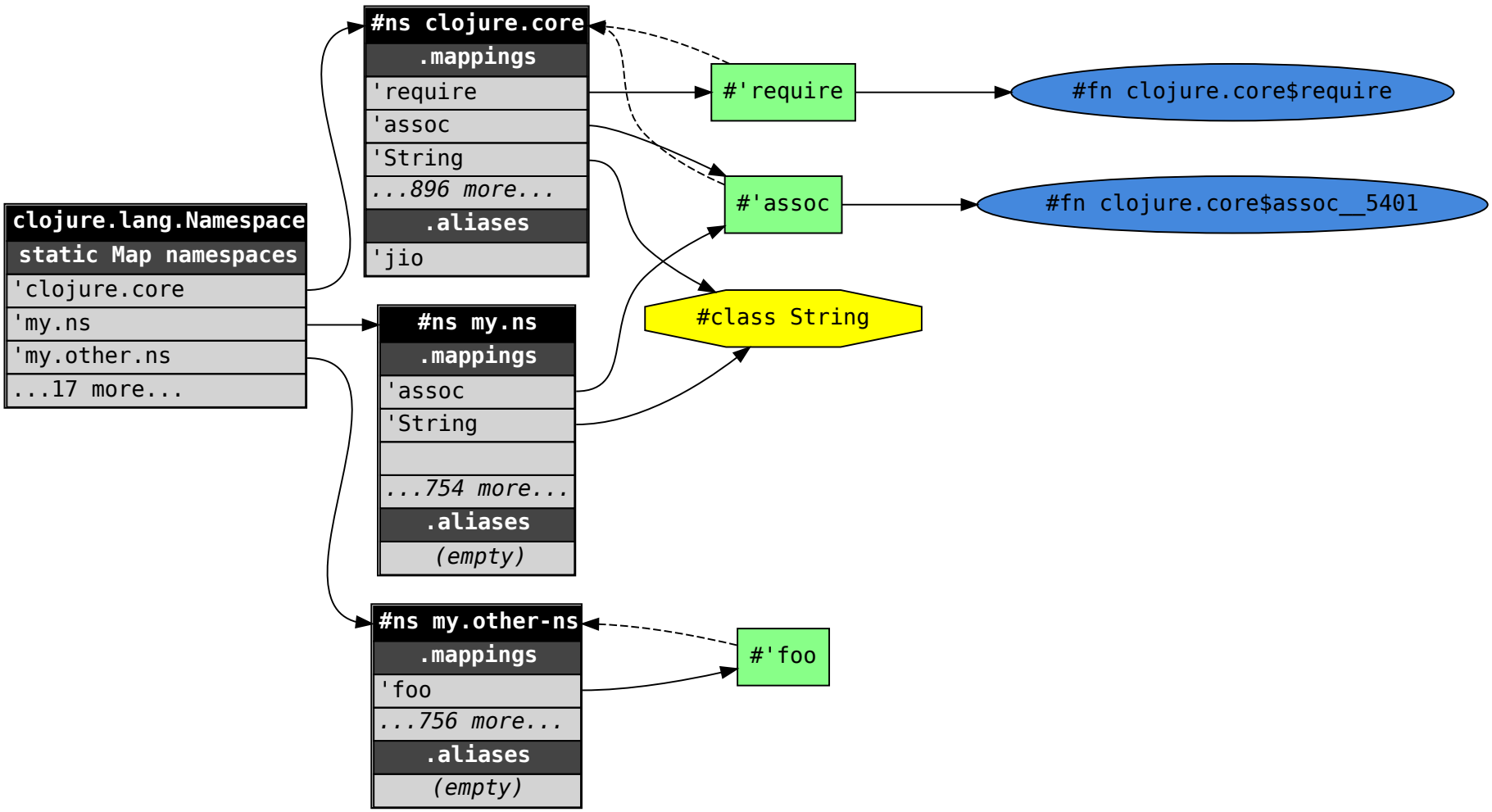
Side Effects



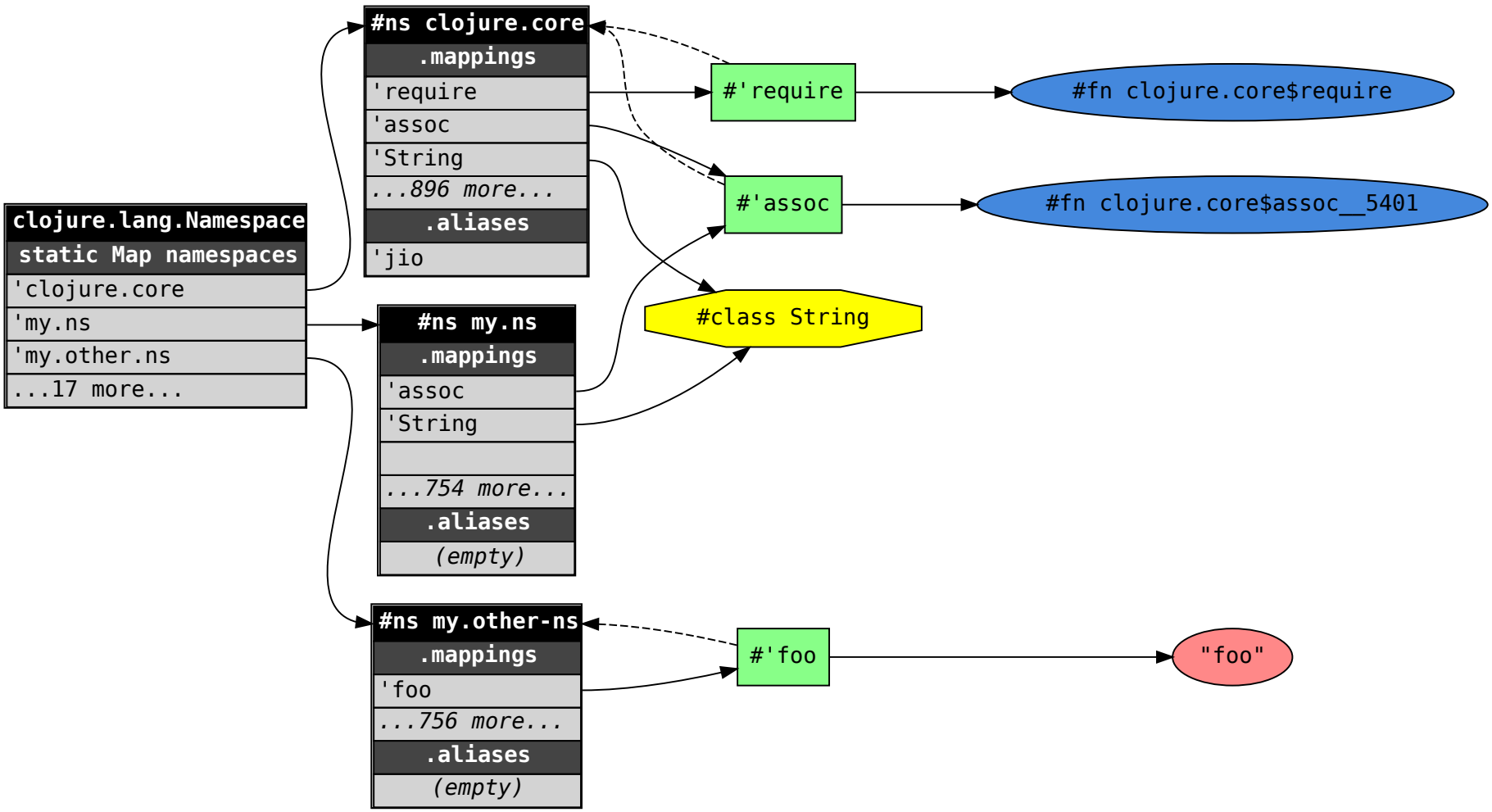
Side Effects



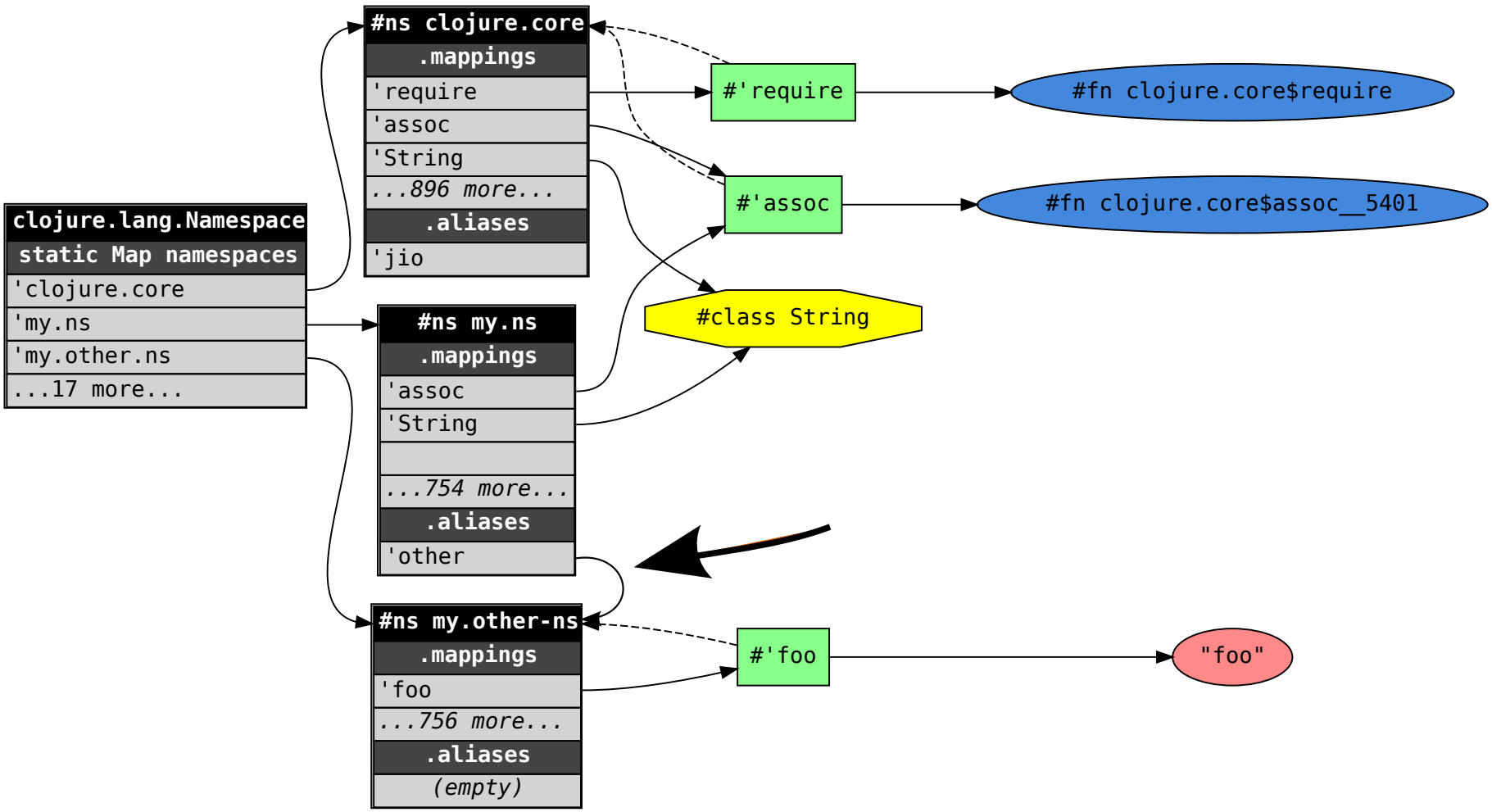
Side Effects



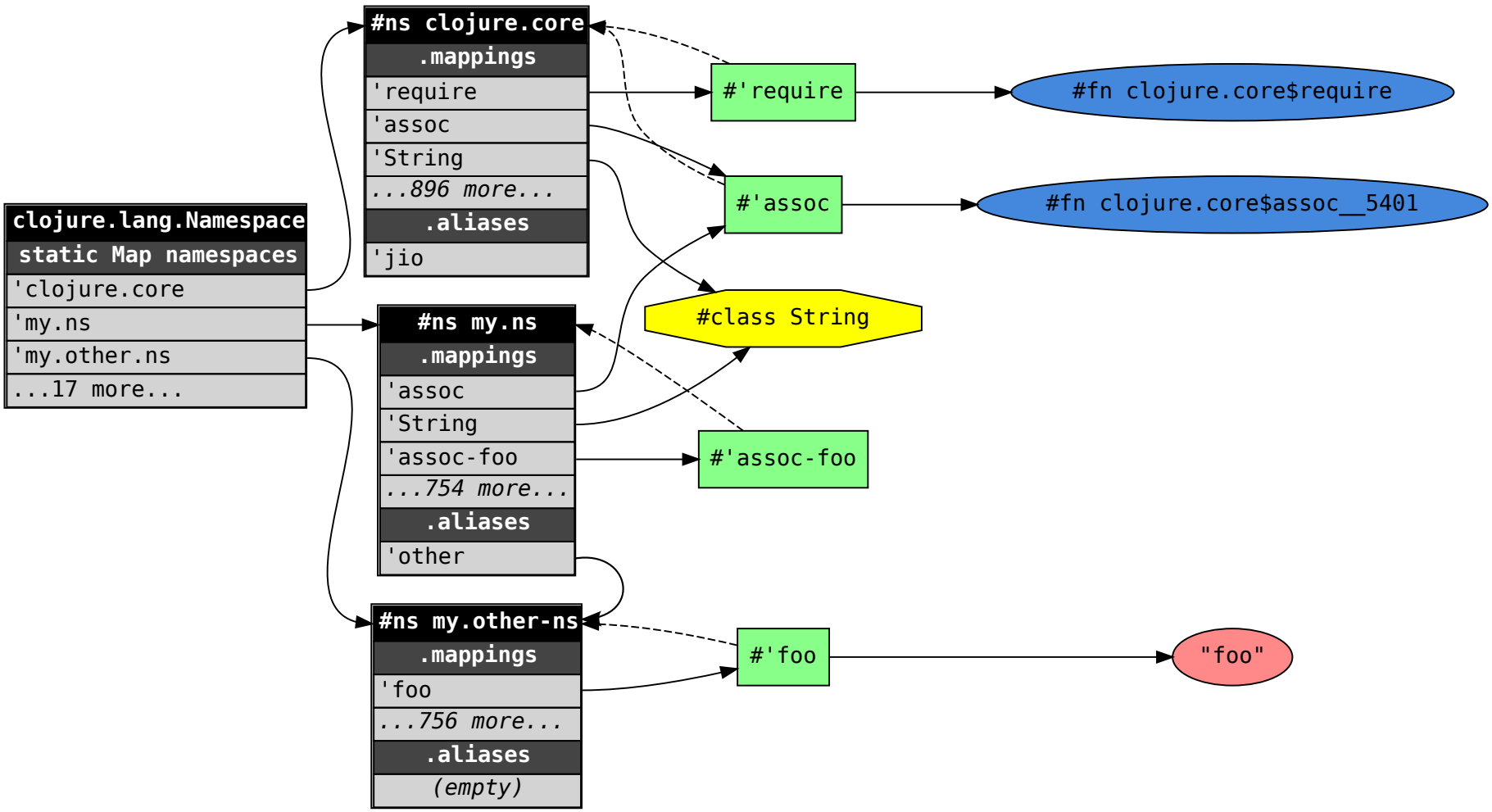
Side Effects



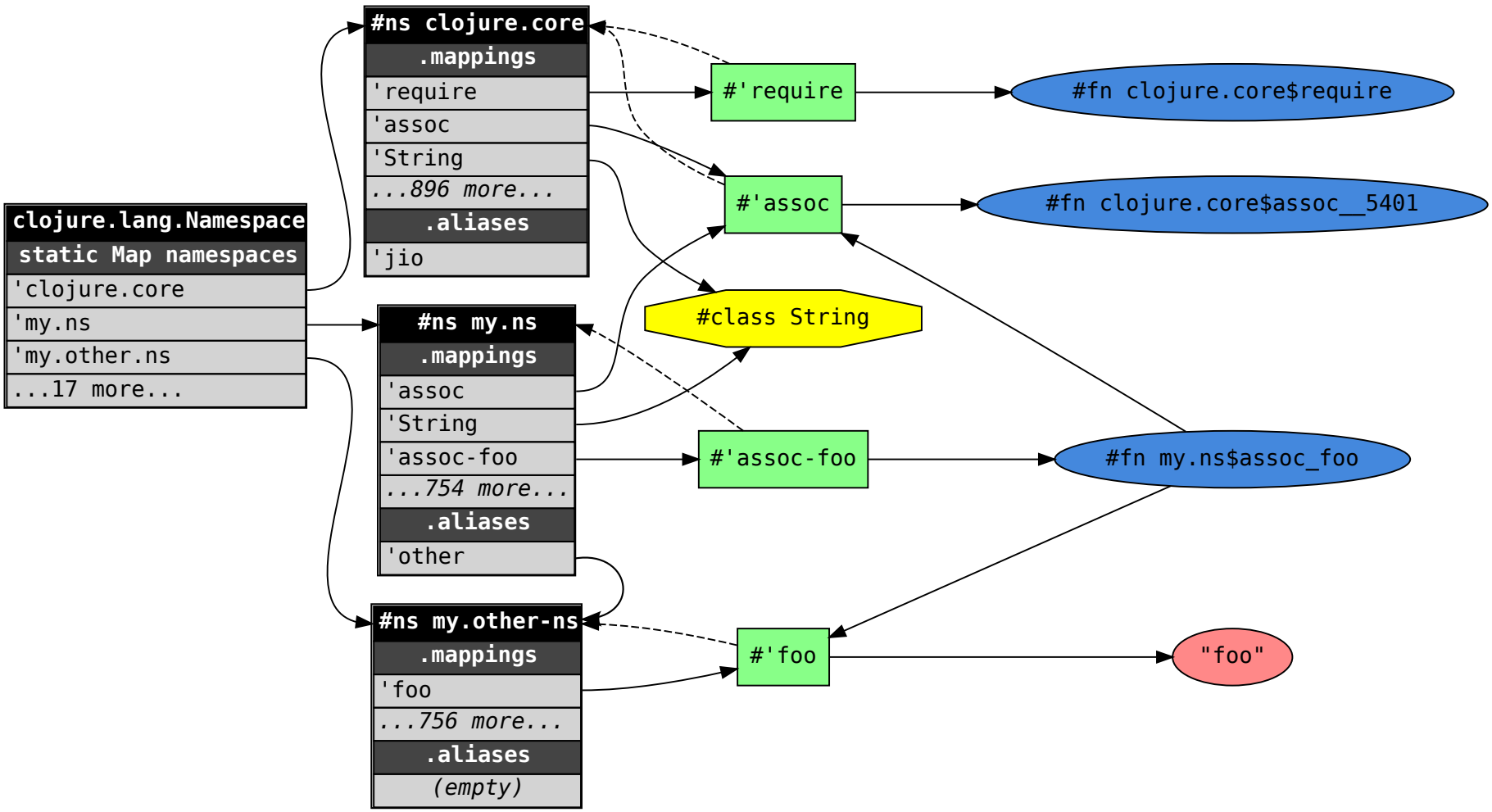
Side Effects



Side Effects



Side Effects



So what was all that then?

- Each function literals is compiled to a dedicated class that can be instantiated into functions
- At the repl and during normal code loading, top-level forms are
  - compiled to bytecode for a 0-arg function
  - loaded via the `DynamicClassLoader`
  - instantiated and `.invoke()`'d, likely performing side effects on the namespace graph or returning a value to the repl

Part 3: `compile`

let's do some AOT

(compile 'my.ns)

```
1 $ clj
2 Clojure 1.10.0-RC2
3 user=> (compile 'my.ns)
4 Syntax error macroexpanding
5   clojure.core/ns at (ns.clj:1:1).
6 No such file or directory
```

(compile 'my.ns)

```
1 $ mkdir classes
2
3 $ clj
4 Clojure 1.10.0-RC2
5 user=> (compile 'my.ns)
6 my.ns
```

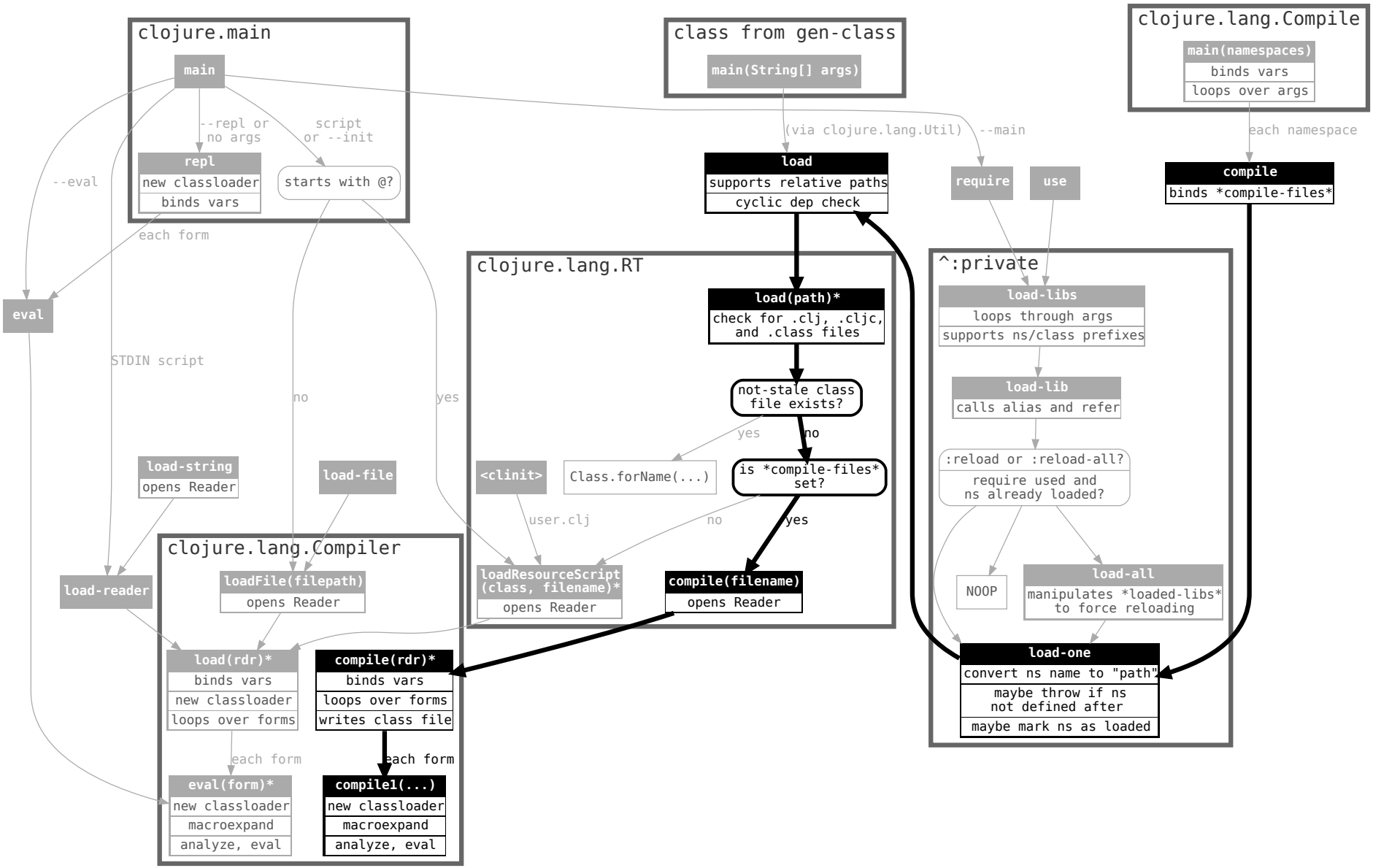
All These Class Files

```
$ find classes/ -type f

classes/clojure/core/specs/alpha$fn__222.class
classes/clojure/core/specs/alpha$fn__235.class
classes/clojure/core/specs/alpha$fn__285$fn__289.class
classes/clojure/core/specs/alpha__init.class
classes/clojure/core/specs/alpha$fn__249$fn__255.class
classes/clojure/core/specs/alpha$fn__180.class
classes/clojure/core/specs/alpha$fn__237$fn__241.class
classes/clojure/core/specs/alpha$fn__249$fn__253.class
classes/clojure/core/specs/alpha$fn__212.class
classes/clojure/core/specs/alpha$fn__197.class
... 47 more ...
```

All These Class Files

```
$ find classes/ -type f | grep my

classes/my/ns$assoc_foo.class
classes/my/ns$fn__304.class
classes/my/ns$loading__6706__auto____298.class
classes/my/ns__init.class
classes/my/other_ns$fn__302.class
classes/my/other_ns$loading__6706__auto____300.class
classes/my/other_ns__init.class
```

The API


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
no
yes
load-string
opens Reader
load-file
load-reader
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

All These Class Files

```
$ find classes/ -type f | grep my

classes/my/ns$assoc_foo.class
classes/my/ns$fn__304.class
classes/my/ns$loading__6706__auto____298.class
classes/my/ns__init.class
classes/my/other_ns$fn__302.class
classes/my/other_ns$loading__6706__auto____300.class
classes/my/other_ns__init.class
```

Top Level Side Effects of `my.ns`

- `(in-ns 'my.ns)`
- call ns-helper-fn-1
- pass ns-helper-fn-2 to that transaction locking thing
- instantiate `my.ns$assoc_foo`
- call `.setMeta` on `#'assoc-foo`
- call `.bindRoot` on `#'assoc-foo`

my.ns__init

- public class my.ns__init
  - static fields
    - 2 vars (#'in-ns, #'assoc-foo),
    - 3 constants (metadata, 'my.ns, 'clojure.core)
  - static init
    - calls __init0()
    - calls load()
  - __init0
    - initializes the five static fields
  - load
    - call (in-ns 'my.ns)
    - initialize my.ns$loading__6706__auto____298 and .invoke()
    - check (= 'clojure.core 'my.ns)
    - call LockingTransaction.runInTransaction(new my.ns$fn__304())
    - set metadata on #'assoc-foo
    - initialize my.ns$assoc_foo, set root binding of #'assoc-foo

Order of Events

```
1  - (compile 'my.ns)
2    - Compiler.compile
3      - Compiler.compile1('(ns my.ns ...))
4        - writes two helper classes
5        - (require 'my.other-ns)
6          - Compiler.compile
7            - Compiler.compile1('(ns my.other-ns ...))
8              - writes two helper classes
9            - Compiler.compile1('(def foo "foo"))
10           - writes init class for my.other-ns
11     - Compiler.compile1('(defn assoc-foo ...))
12       - writes assoc_foo class
13     - writes init class for my.ns
```

So What Was All That Then?

`compile`

- operates on namespaces
- evals all of the code into the current JVM
- writes `.class` files for all classes, including an `*__init` class that performs all the top-level side effects

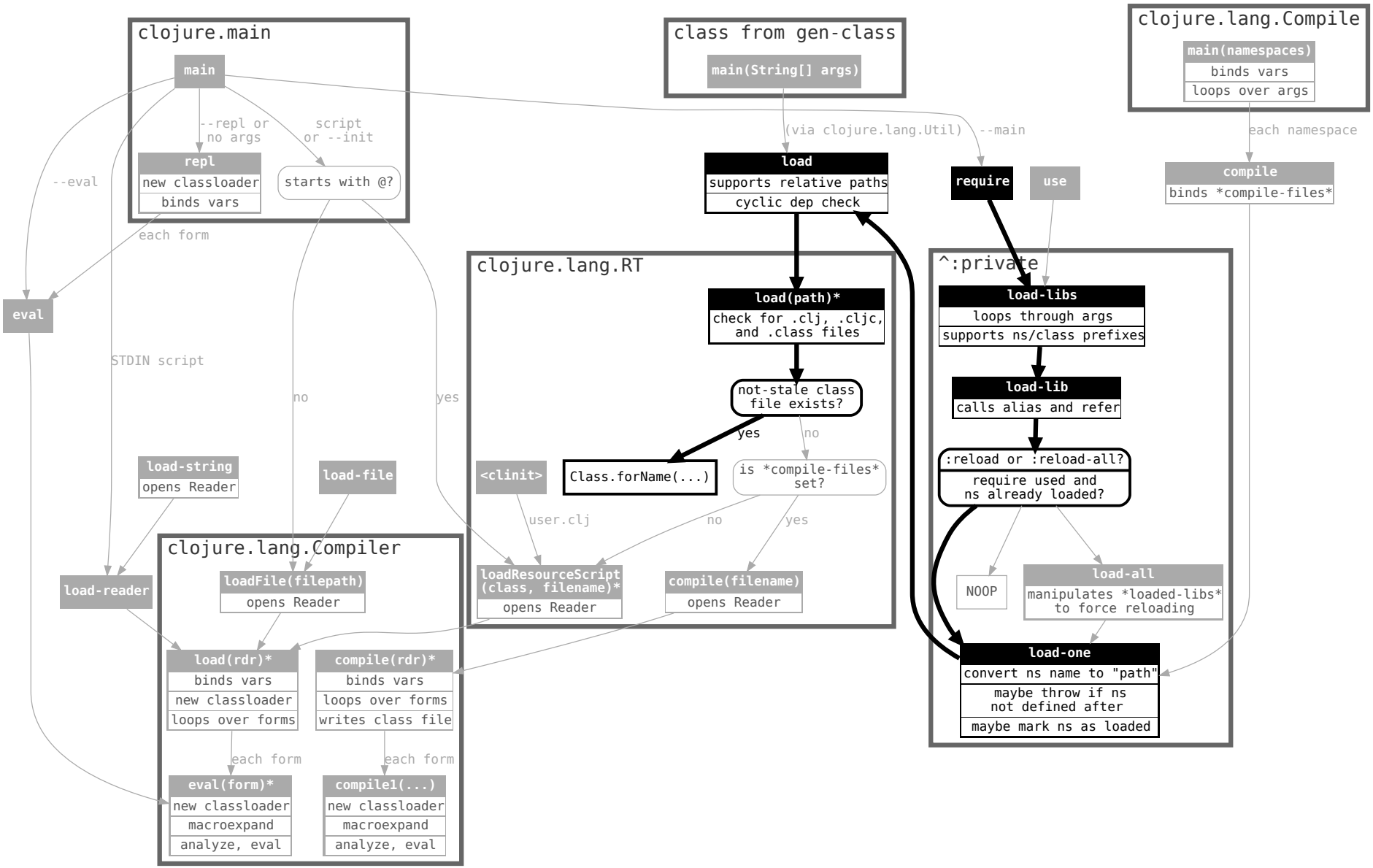
Part 4: `require` after `compile`

doing it all over again

The Repl

```
1 $ clj
2 Clojure 1.10.0-RC2
3 user=> (require 'my.ns)
4 nil
```

The API


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
no
yes
load-string
opens Reader
load-file
load-reader
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

(Class/forName "my.ns__init")

`my.ns__init<clinit>`

- looks up vars and constants
- calls `(in-ns)`
- calls `(require 'my.other-ns)`
  - `my.other_ns__init<clinit>`
    - looks up vars and constants
    - calls `(in-ns)`
    - sets meta and root of `#'foo`
- initializes `my.ns$assoc_foo`
- sets meta and root of `#'assoc-foo`

So What Was All That Then?

- `require`-ing a namespace when there's a class file available results in loading the `*__init` class, which triggers all the top-level side effects that mutate the namespace graph

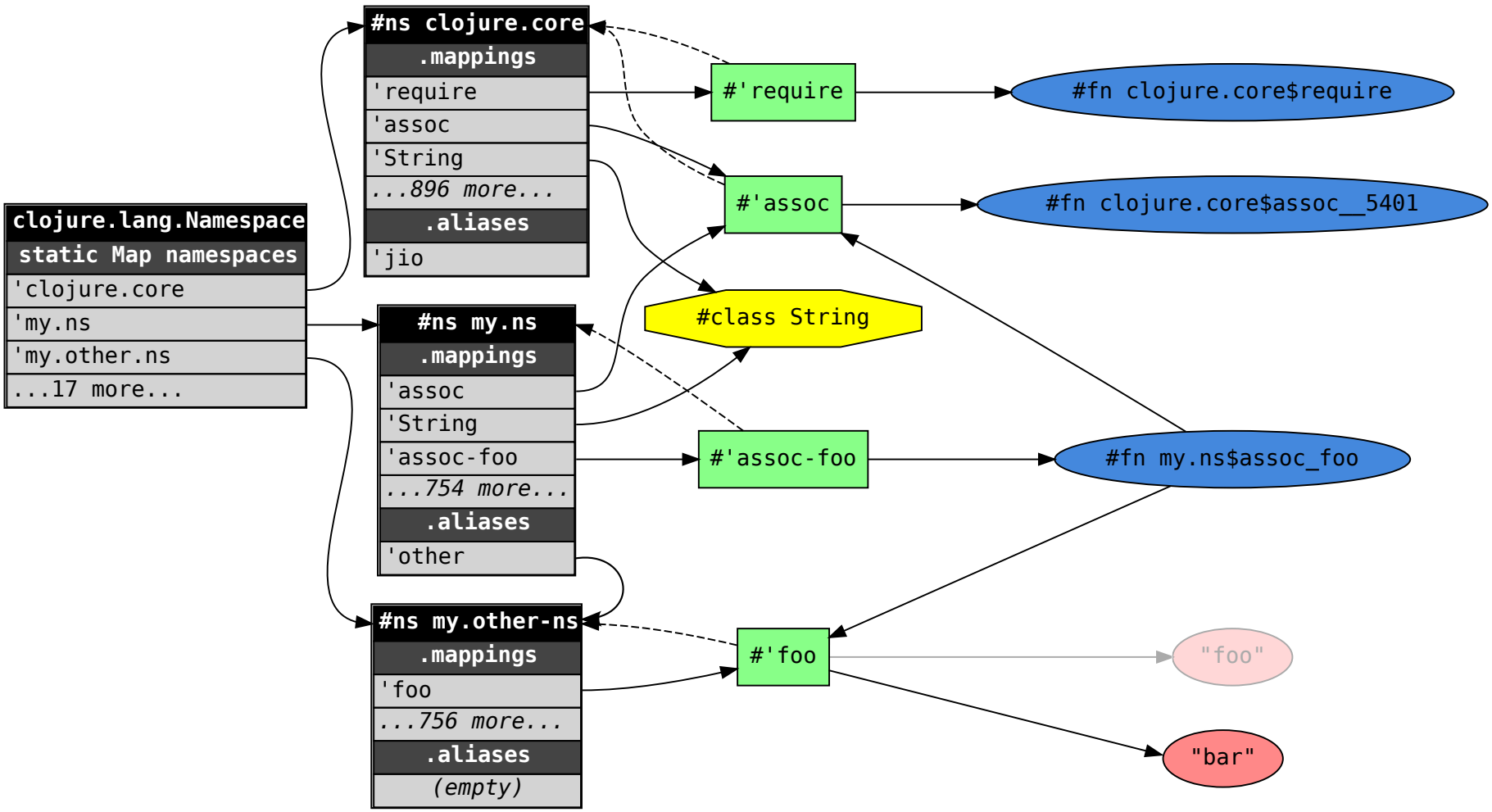
Part 5: `:reload`, etc.

let's change something

user=> (def foo "bar")

```
1  user=> (require 'my.ns)
2  nil
3  user=> (in-ns 'my.other-ns)
4  #object[clojure.lang.Namespace 0x4ee37ca3 "my.other-ns"]
5  my.other-ns=> (def foo "bar")
6  #'my.other-ns/foo
```

(def foo "bar")


clojure.lang.Namespace
static Map namespaces
'clojure.core
'my.ns
'my.other.ns
...17 more...
#ns clojure.core
.mappings
'require
'assoc
'String
...896 more...
.aliases
'jio
#ns my.ns
.mappings
'assoc
'String
'assoc-foo
...754 more...
.aliases
'other
#ns my.other-ns
.mappings
'foo
...756 more...
.aliases
(empty)
#'require
#'assoc
#class String
#'assoc-foo
#'foo
#fn clojure.core$require
#fn clojure.core$assoc__5401
#fn my.ns$assoc_foo
"foo"
"bar"

(def foo "bar")

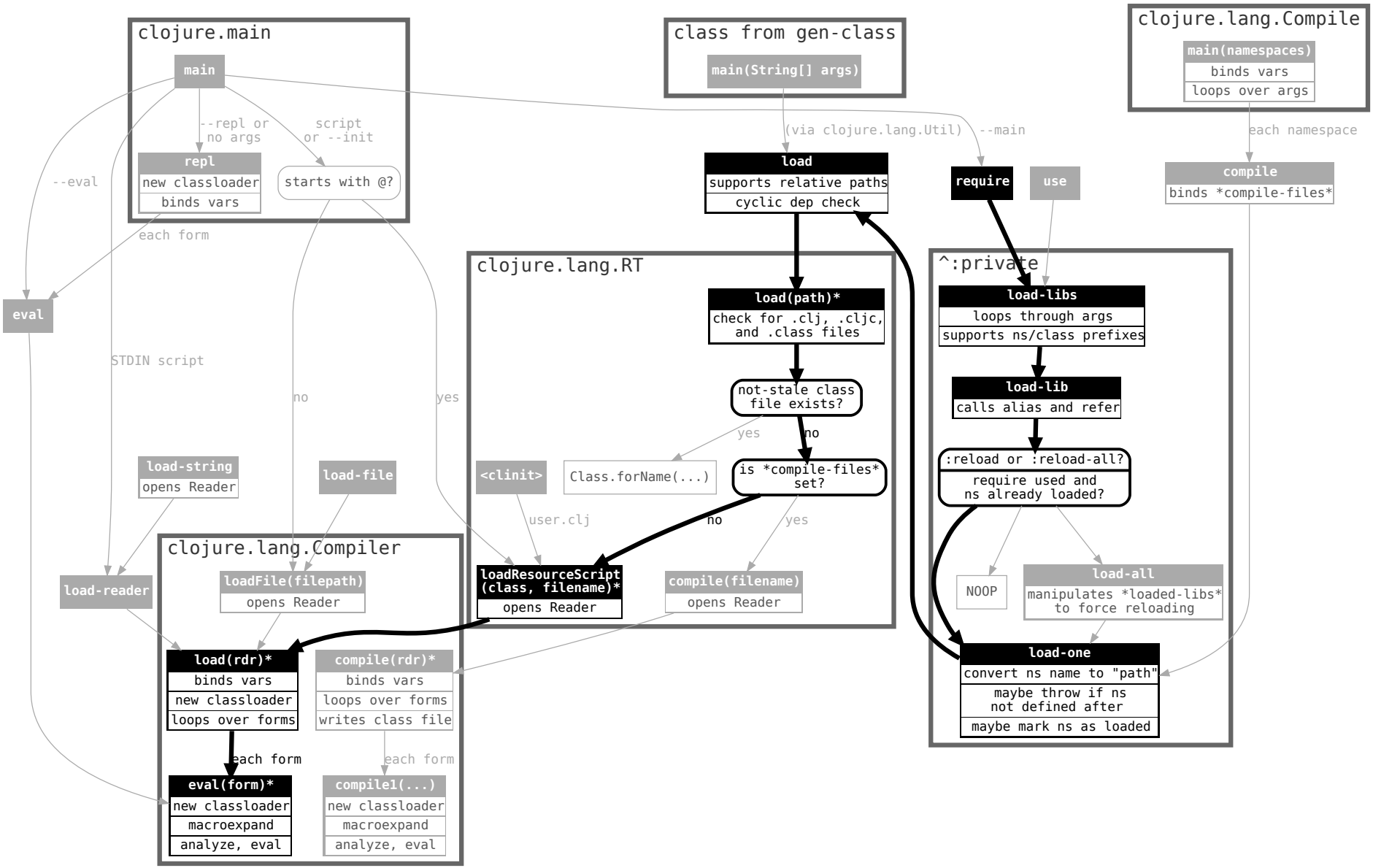
```
1  my.other-ns=> (my.ns/assoc-foo {})
2  {:foo "bar"}
```

(defn assoc-foo ...) (require 'my.ns :reload)

src/my/ns.clj

```
1  (ns my.ns
2    (:require
3     [my.other-ns :as other]))
4
5  (defn assoc-foo
6    [m]
7    (assoc m :foo other/foo
8             :another "map-entry"))
```

The API


clojure.main
main
--repl or no args
script or --init
repl
new classloader
binds vars
starts with @?
--eval
each form
eval
STDIN script
load-string
opens Reader
load-file
load-reader
clojure.lang.Compiler
loadFile(filepath)
opens Reader
load(rdr)*
binds vars
new classloader
loops over forms
compile(rdr)*
binds vars
loops over forms
writes class file
each form
each form
eval(form)*
new classloader
macroexpand
analyze, eval
compile1(...)
new classloader
macroexpand
analyze, eval
class from gen-class
main(String[] args)
(via clojure.lang.Util)
--main
load
supports relative paths
cyclic dep check
clojure.lang.RT
load(path)*
check for .clj, .cljc, and .class files
not-stale class file exists?
yes
no
<clinit>
Class.forName(...)
is *compile-files* set?
user.clj
no
yes
loadResourceScript (class, filename)*
opens Reader
compile(filename)
opens Reader
no
yes
require
use
^:private
load-libs
loops through args
supports ns/class prefixes
load-lib
calls alias and refer
:reload or :reload-all?
require used and ns already loaded?
NOOP
load-all
manipulates *loaded-libs* to force reloading
load-one
convert ns name to "path"
maybe throw if ns not defined after
maybe mark ns as loaded
clojure.lang.Compile
main(namespaces)
binds vars
loops over args
each namespace
compile
binds *compile-files*

What Happens?

- `(in-ns)` (already exists)
- `(require 'my.other-ns)` (NOOP)
- Compile the function, load into DynamicClassLoader as `my.ns$assoc_foo`
- Compiler looks up `#'assoc-foo` (already exists)
- `(.setMeta #'assoc-foo ...)` (no problem)
- `(.bindRoot #'assoc-foo (new my.ns$assoc_foo))` perfect!

(defn assoc-foo ...) (require 'my.ns :reload)

```
1  my.other-ns=> (my.ns/assoc-foo {})
2  {:foo "bar" :another "map-entry"}
```

So What Was All That Then?

- the repl and `(require :reload)` do largely the same thing:
  - Use `Compile.eval`
    - creates classes
    - loads them into the dynamic classloader
    - invokes them, so that they perform modifications to the namespace graph, existing namespaces and vars

Epilogue

Edge Cases

- Weird things can happen with name collisions
- The unit of compilation isn't quite a form – files have scope for certain dynamic vars, which the repl can't honor

Bottom Lines

- There's a namespace graph!
- `ns`, `def`, and other top level "commands" mutate the namespace graph
- Functions are compiled to classes in memory or in the file system
- in AOT, files can be compiled to `*__init` classes that perform the same mutations as evaling the file

This is it, it's over now

Thanks to (alphabetically)

- Conj organizers
- DRW
- Nicola Mometto
- You

I apologize for all of the