Namespace, Now and Then

The State of Namespace

Satoshi Tagomori (@tagomoris)
2024/08/31 RubyKaigi 2024 follow up event

@tagomoris

Satoshi Tagomori

Maintainer/Founder:

- **OSS: Fluentd, MessagePack, Norikra, Woothee, …**
- **Event: ISUCON**
- **Service: Pathtraq**


SAKURA internet

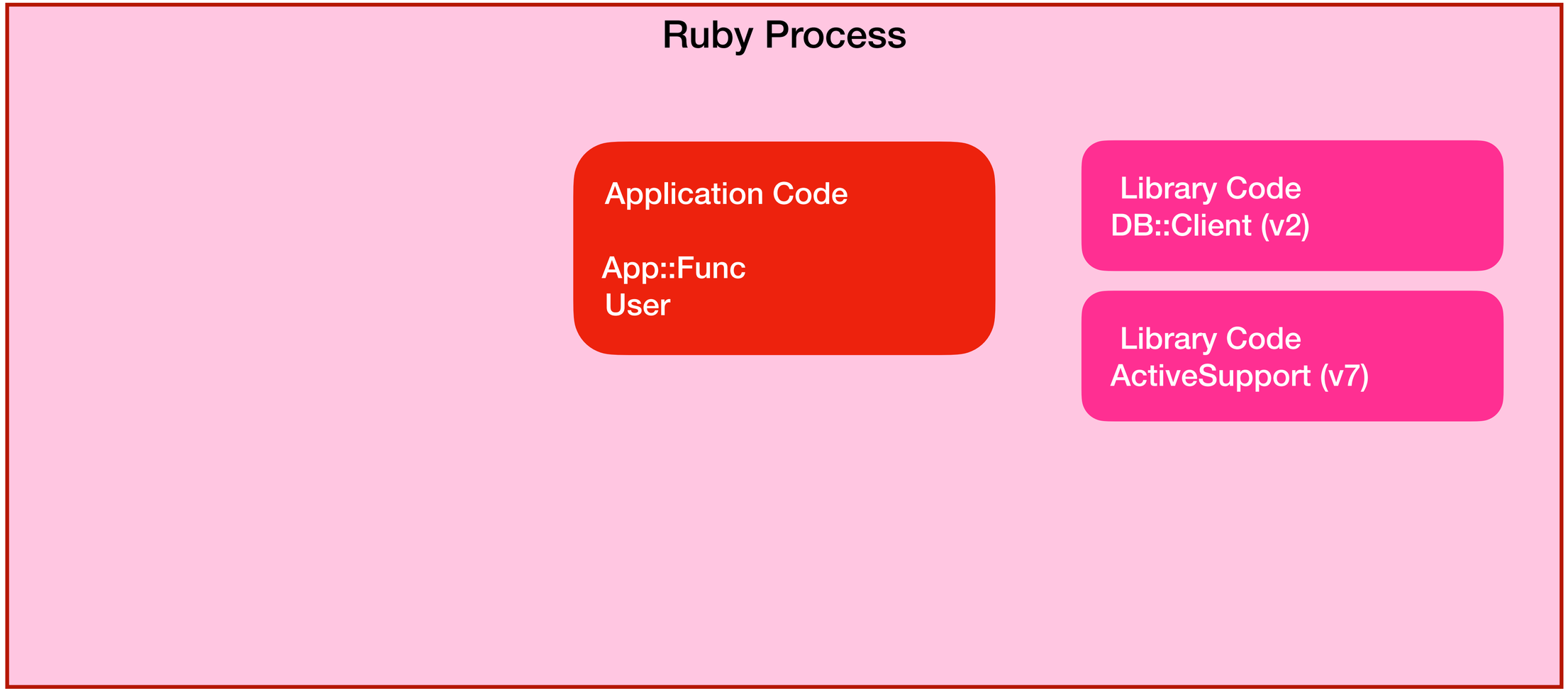
Namespace

Separating apps/libs into isolated spaces

- Load apps/libs in a space
- Hide changes from apps/libs in a namespace to other spaces
- Run methods defined in a space with definitions in the space

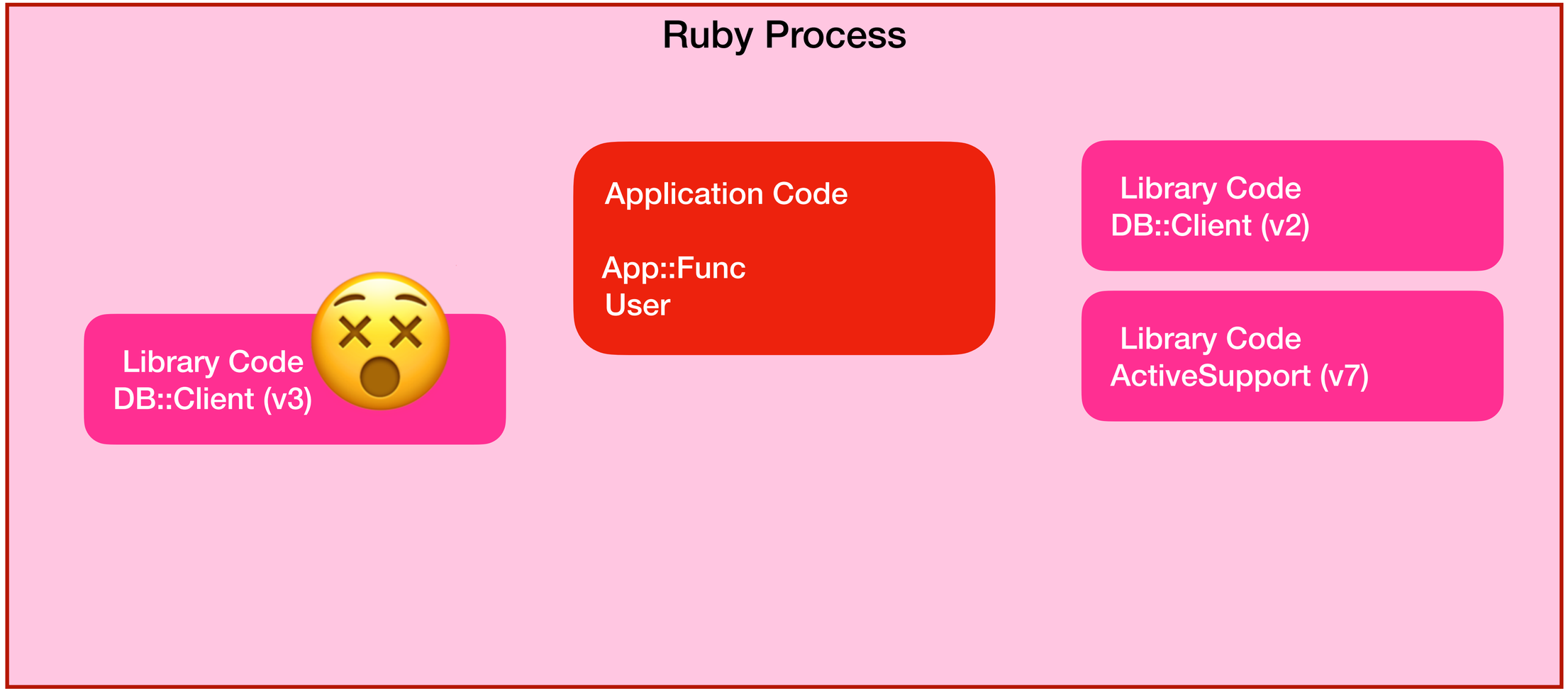
Before Namespace: Global Only

All classes/modules are shared in the entire Ruby process



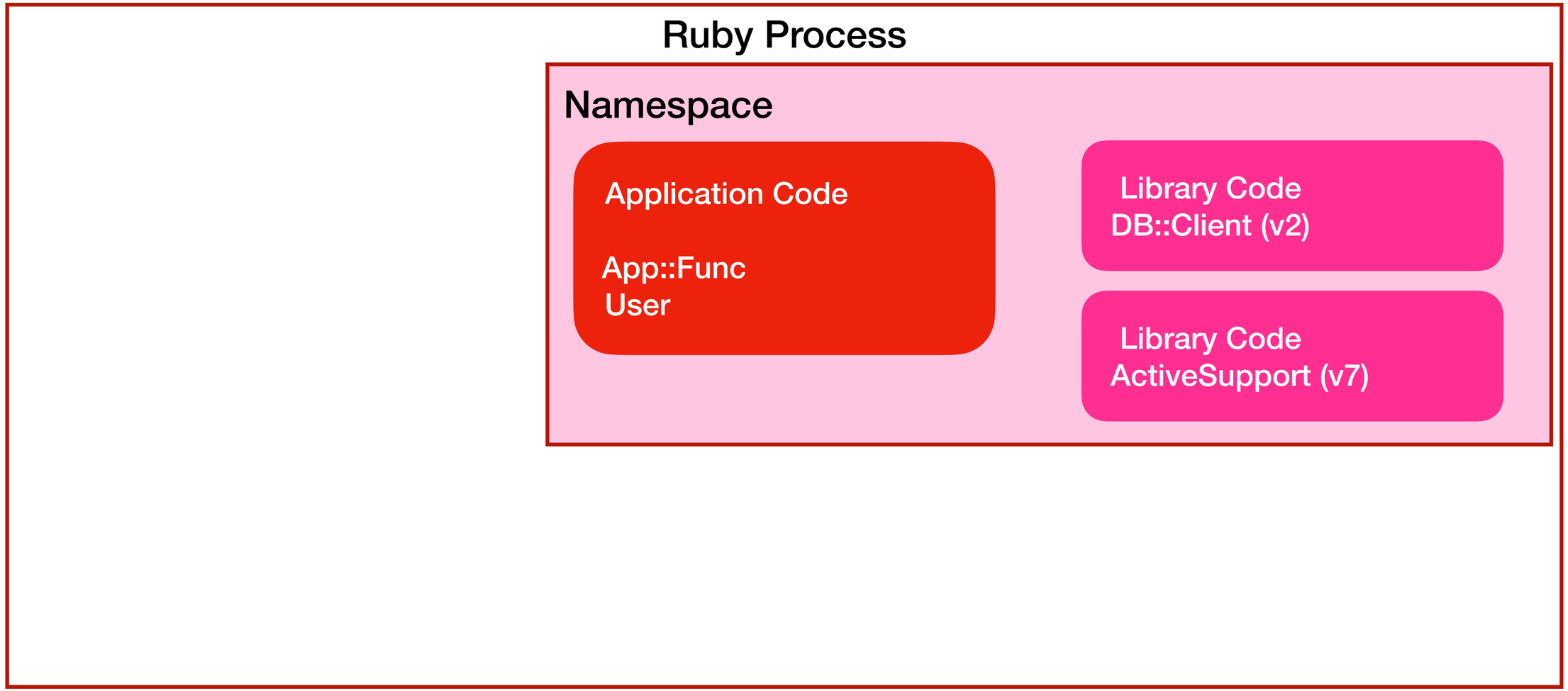
Before Namespace: Global Only

Collision: 2 versions of 1 library cause errors



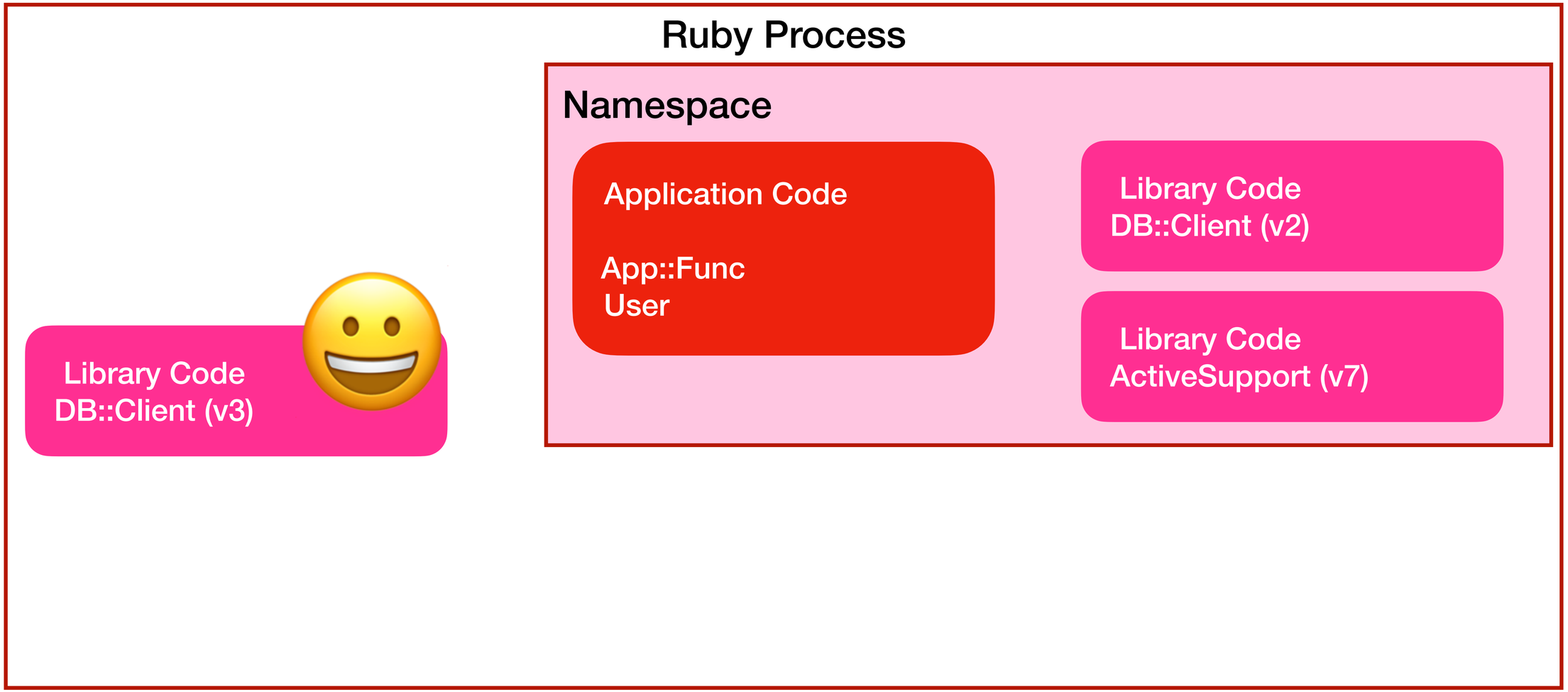
Namespace

Load apps/libs in a space



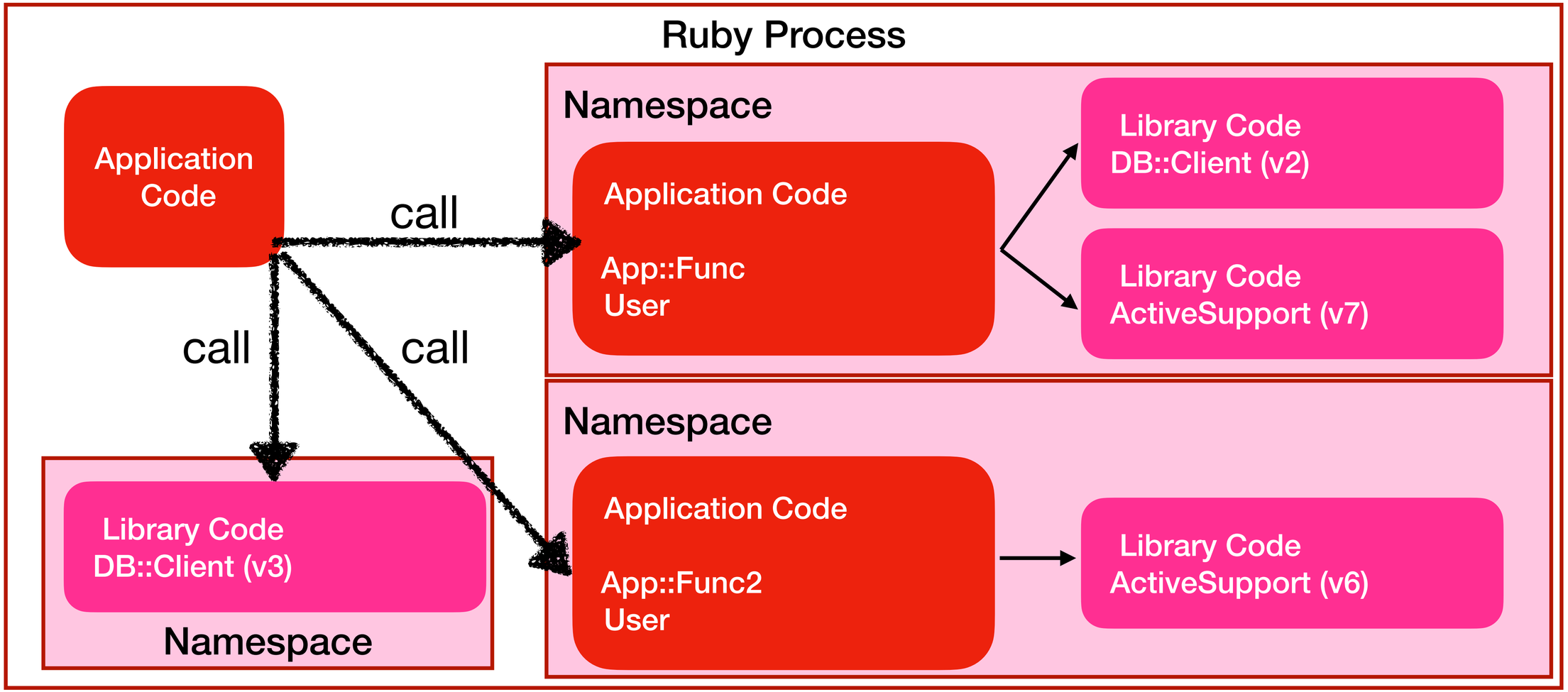
Namespace

Hide changes from apps/libs in a namespace to other spaces



Namespace

Run methods defined in a space with definitions in the space



at RubyKaigi 2024

Did it work?

- Demo code failed with SEGV (just once! 🤪)
- ‘test-all’ stopped in the middle with SEGV

at Jul 1

- 完走したが失敗は増えた
  - ```
    Finished tests in 2273.469071s, 14.2364 tests/s, 2835.4193 assertions/s.
    32366 tests, 6446238 assertions, 21 failures, 19 errors, 192 skips
    ```
  - まあしょうがないね


    ```

at the End of Aug, 2024

Does it work?

- Demo code SHOULD run successfully!
- ‘test-all’ runs to the end!
  - 2 failures, 1 error
  - 2 failures: new failures after the last rebase
  - 1 error: SEGV about (probably) subclasses with GC.compact

Namespace design updates

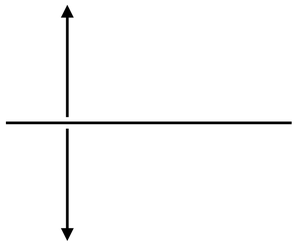
Types of Namespaces

- Before RubyKaigi 2024
  - Root namespace
  - Main namespace
  - (Other) namespaces
- After RubyKaigi 2024
  - Root namespace
  - Builtin namespace
  - (User namespaces)
    - Main namespace
    - Optional namespaces

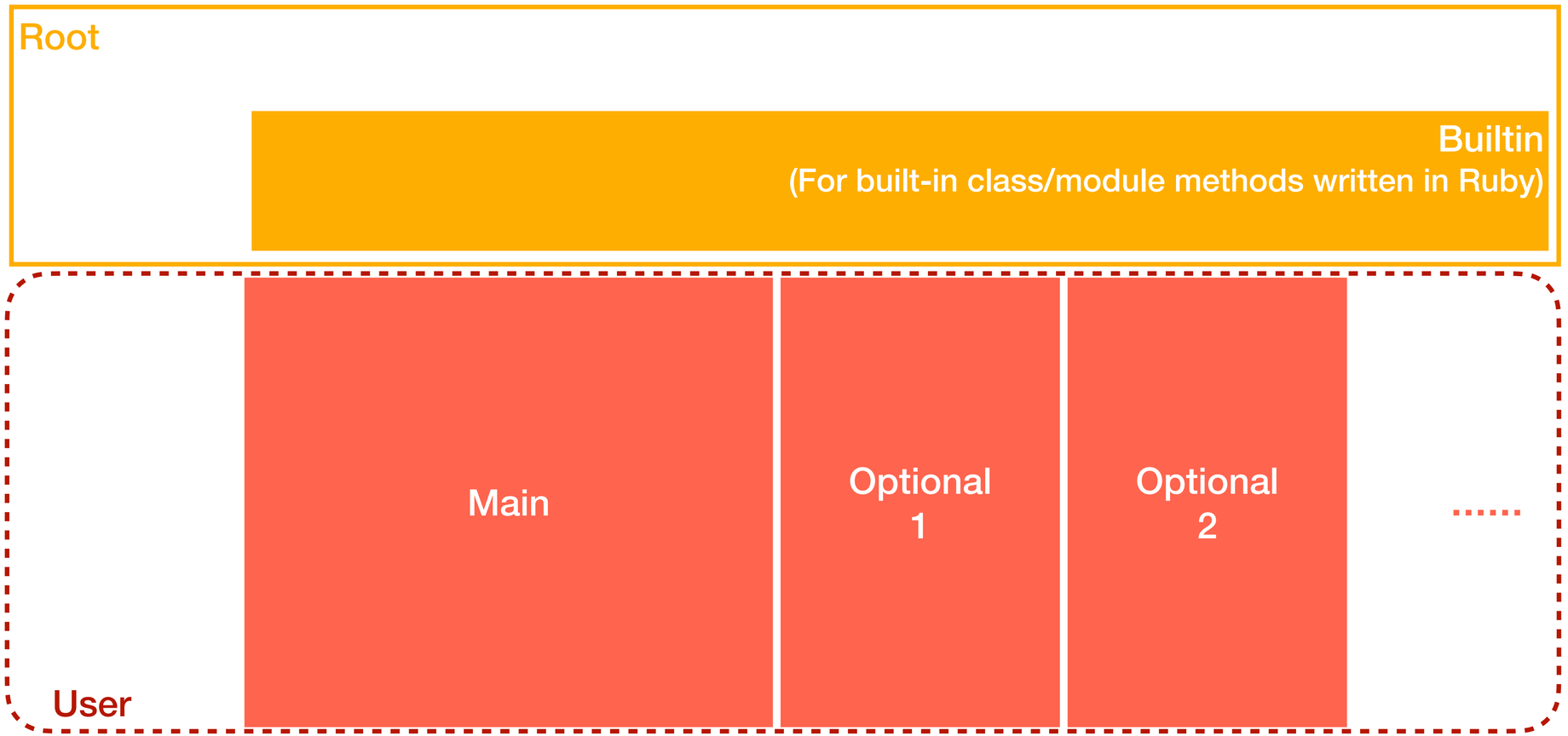
Types of Namespaces (cont.)

Seeing is Believing

for built-in classes/modules



for user scripts



Loading .rb into builtin namespace

RubyGems is in the box!

- Classes/Modules available without any #require

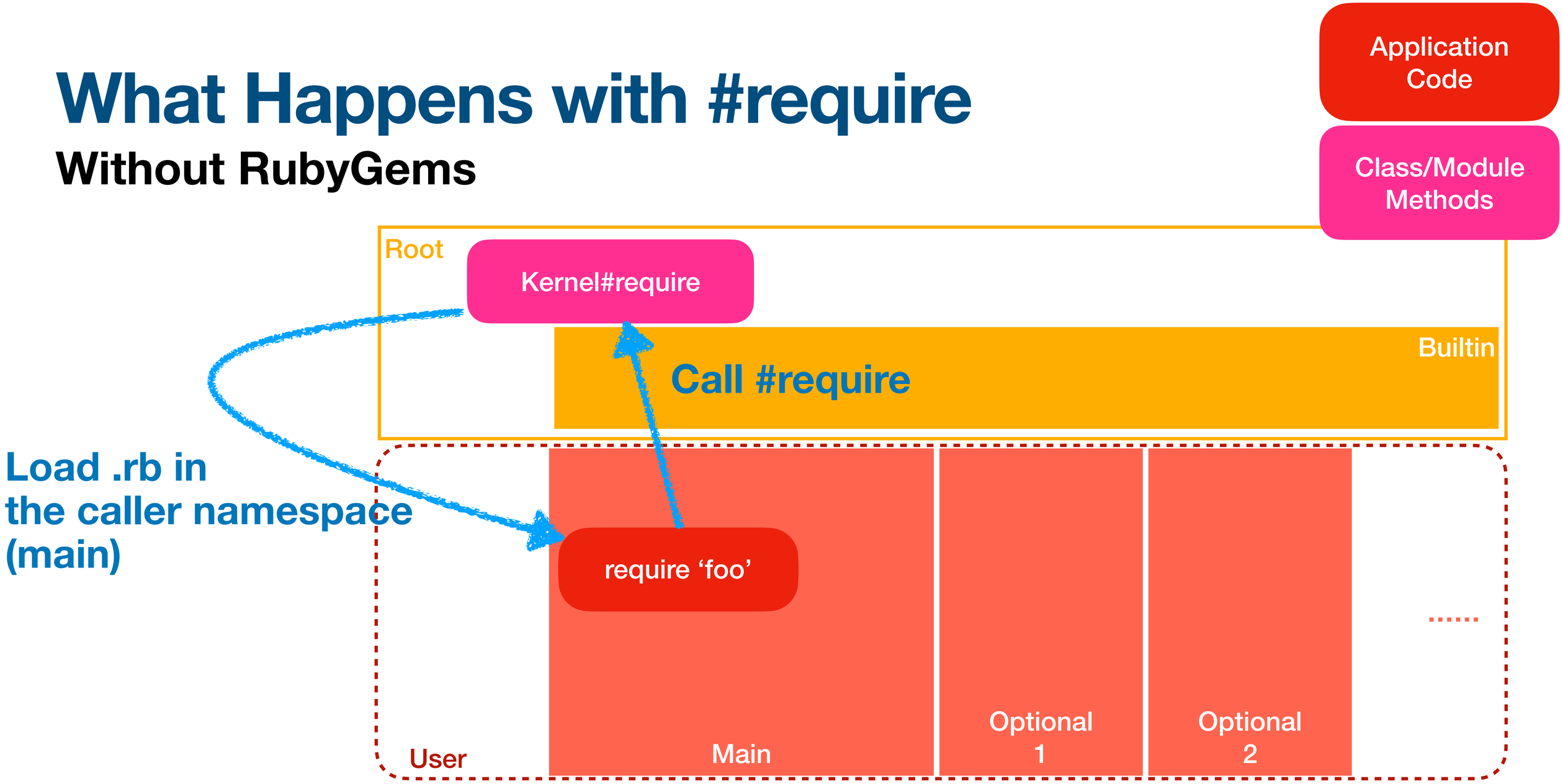
  SHOULD be in the root (or builtin) namespace

- RubyGems should be in the root/builtin namespace

- RubyGems requires its .rb files … What happens?

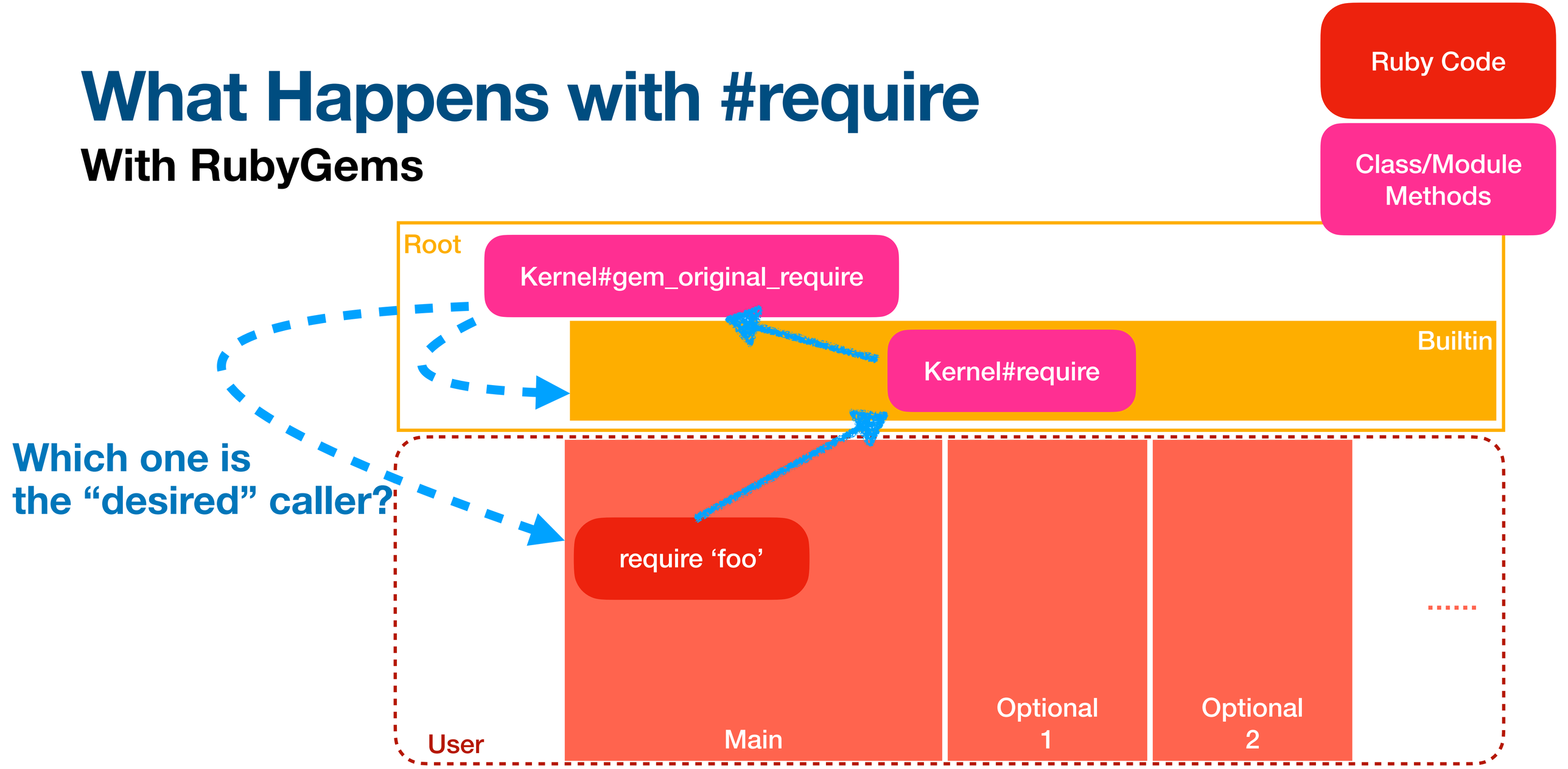
What Happens with #require

Without RubyGems



What Happens with #require

With RubyGems



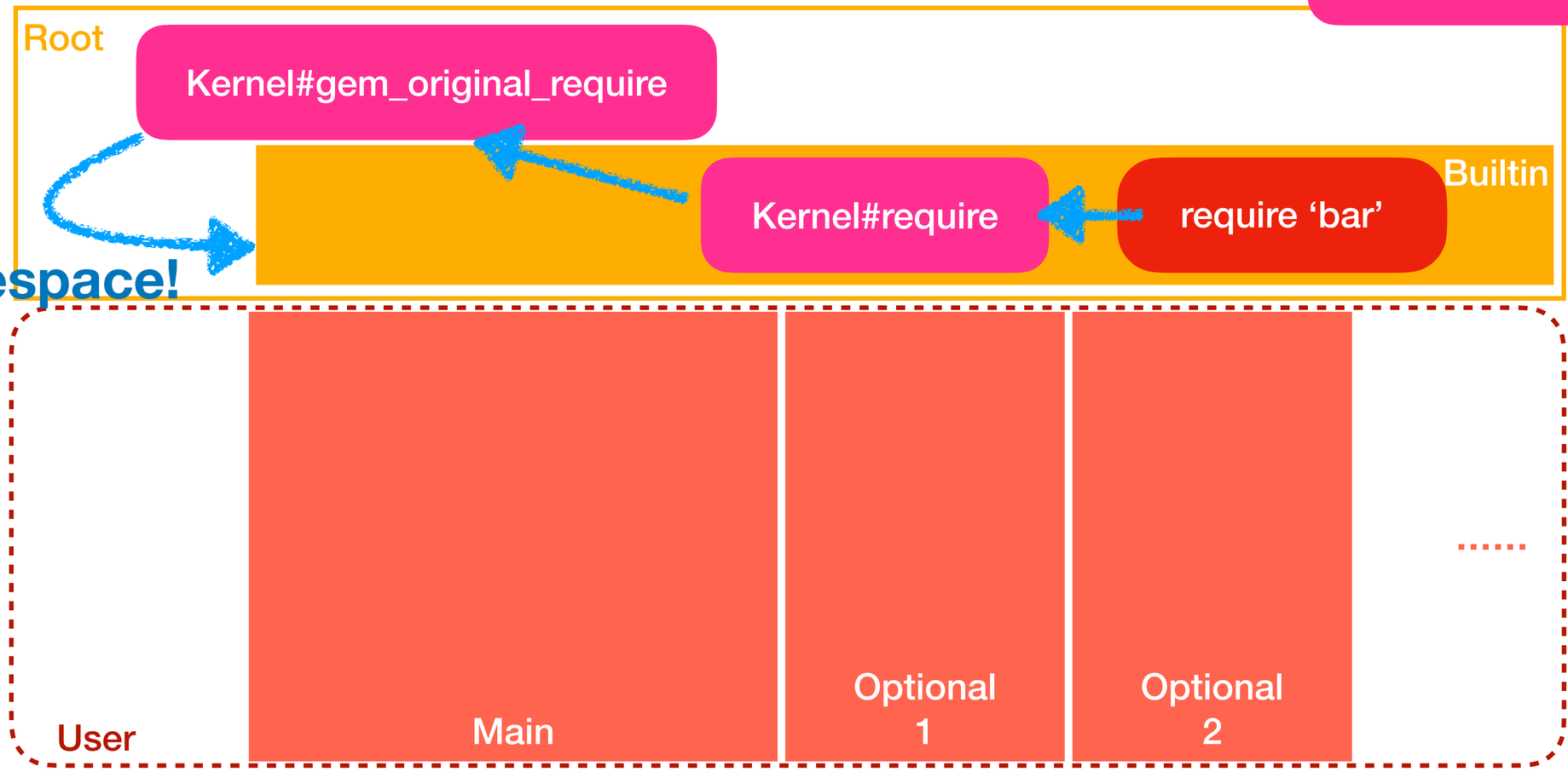
What Happens with #require

With RubyGems, by RubyGems .rb code

Ruby Code

Class/Module Methods

RubyGems requires files to be loaded into the builtin namespace!



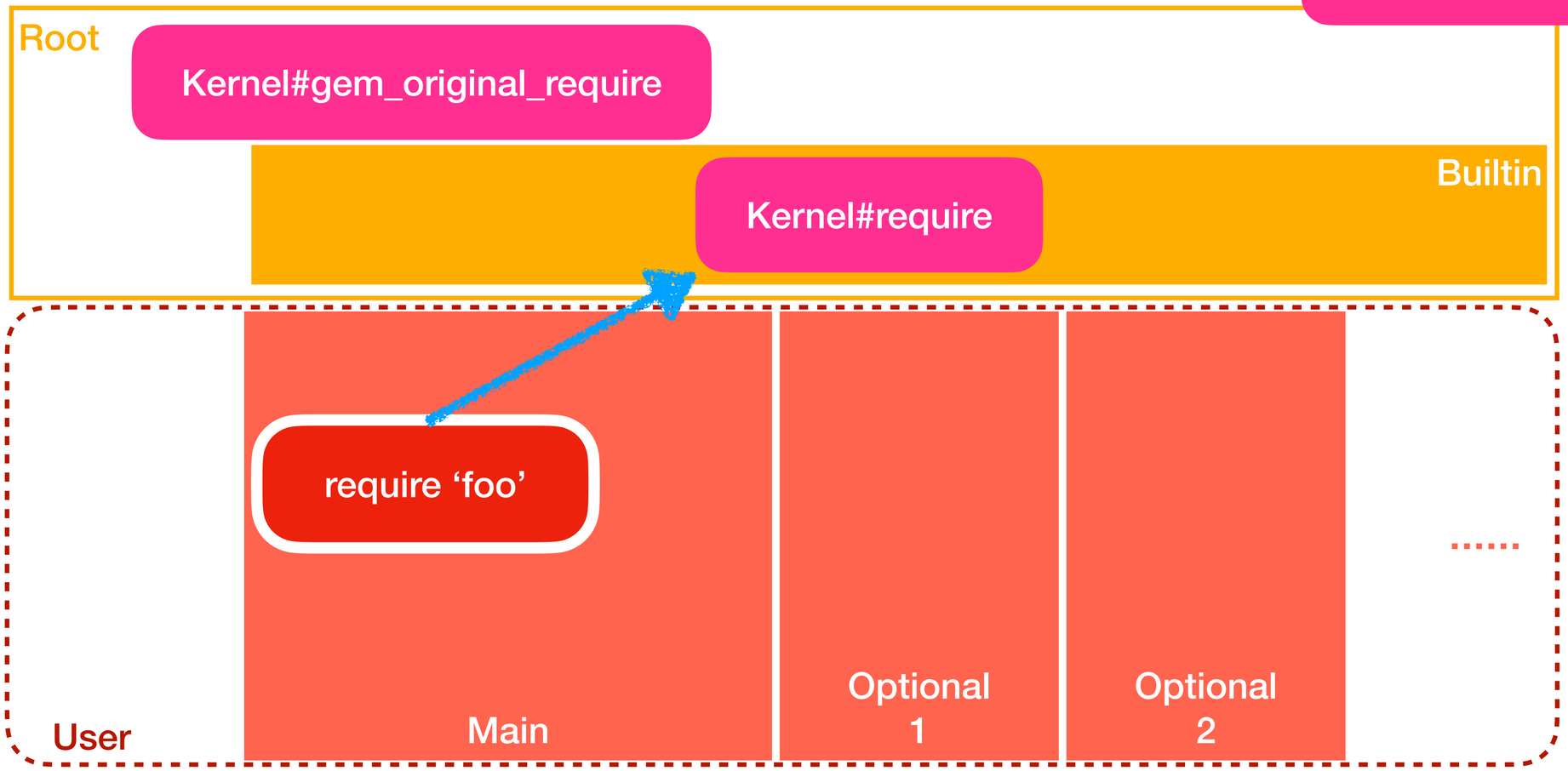
What Happens with #require

With RubyGems, and RubyGems .rb code

Ruby Code

Class/Module Methods

Call #require (caller is main)



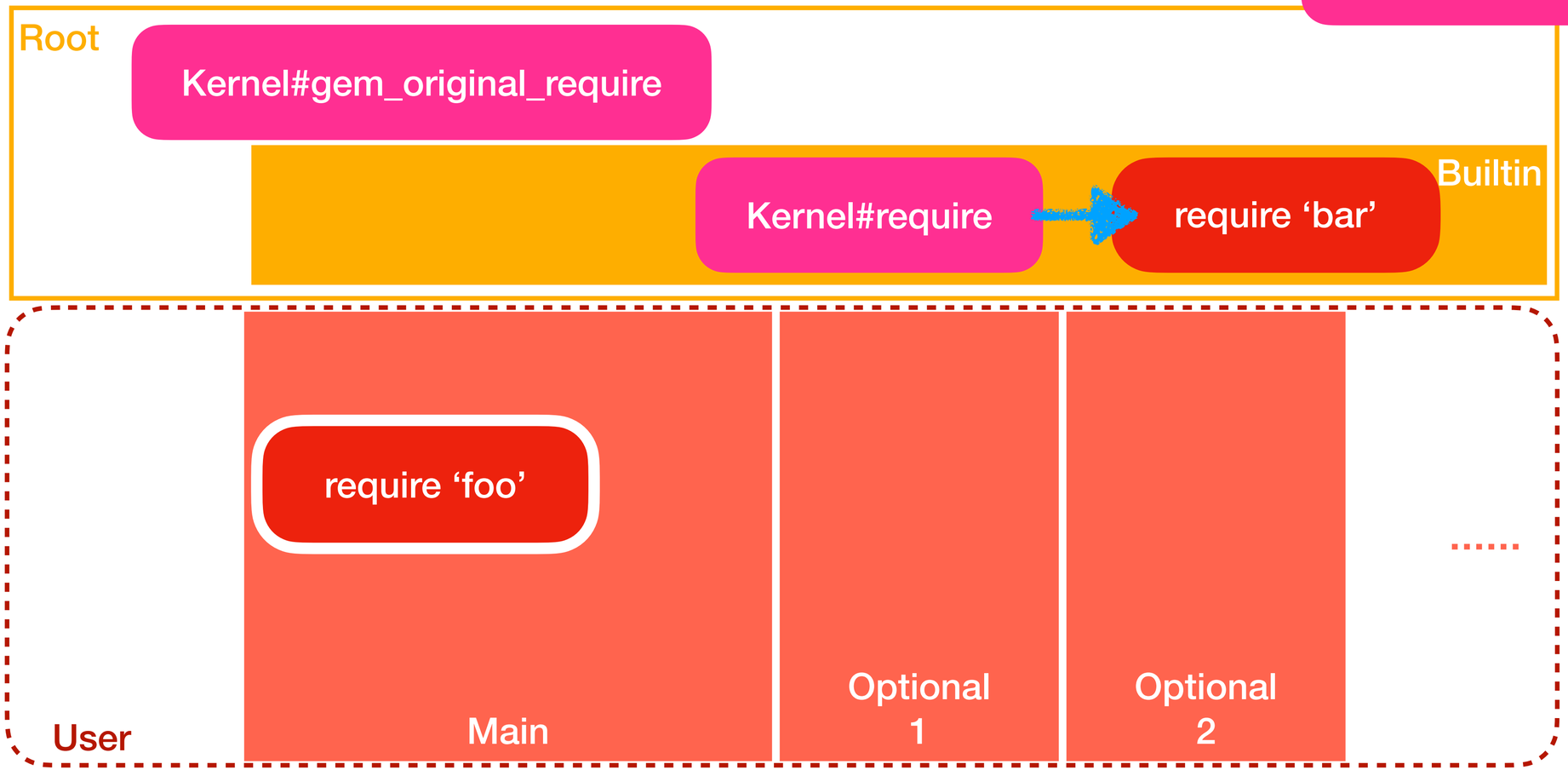
What Happens with #require

With RubyGems, and RubyGems .rb code

Ruby Code

Class/Module Methods

The .rb code for Kernel#require executes any .rb code
(caller is main)



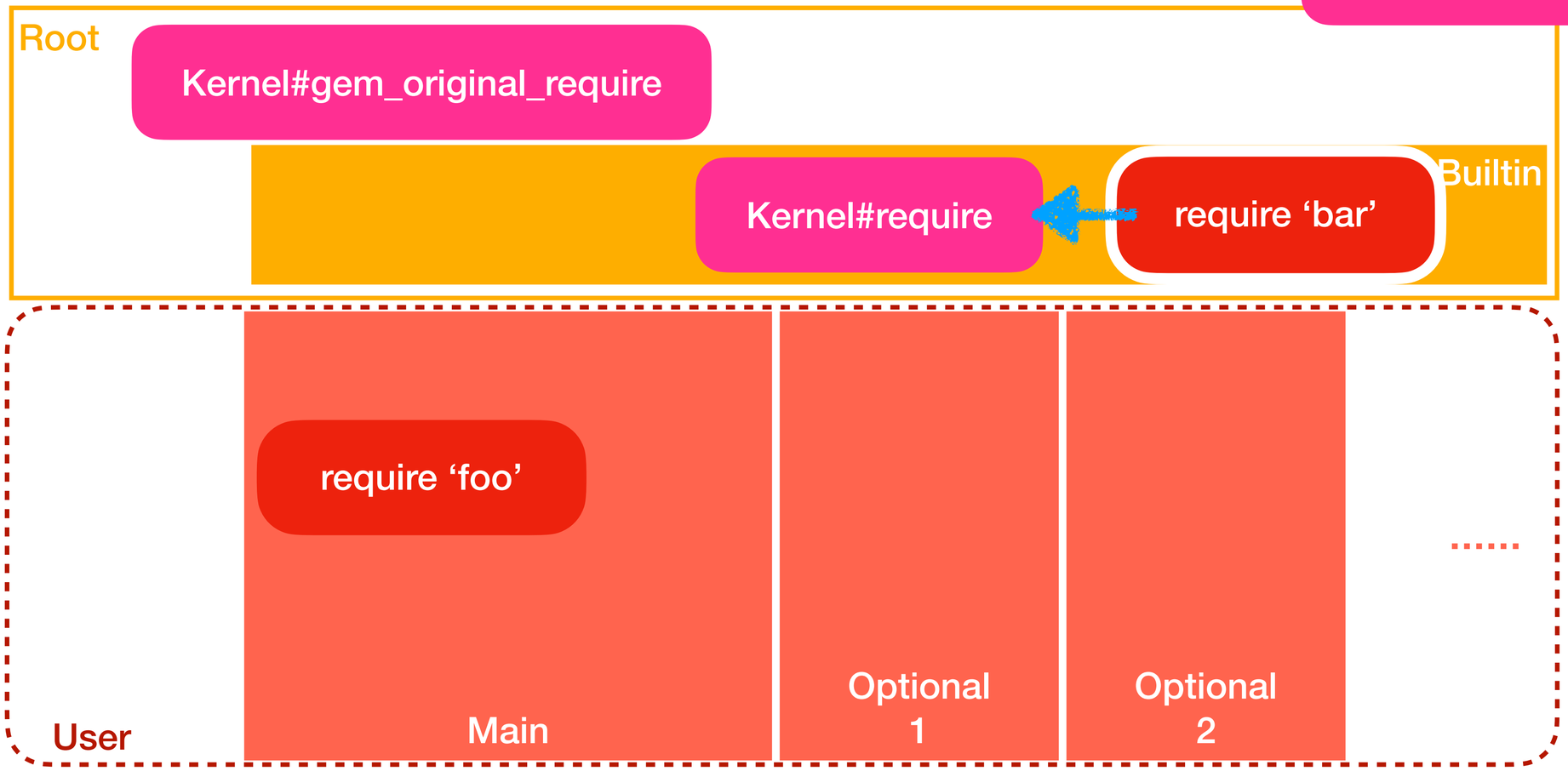
What Happens with #require

With RubyGems, and RubyGems .rb code

Ruby Code

Class/Module Methods

The .rb code requires any file (caller is builtin)



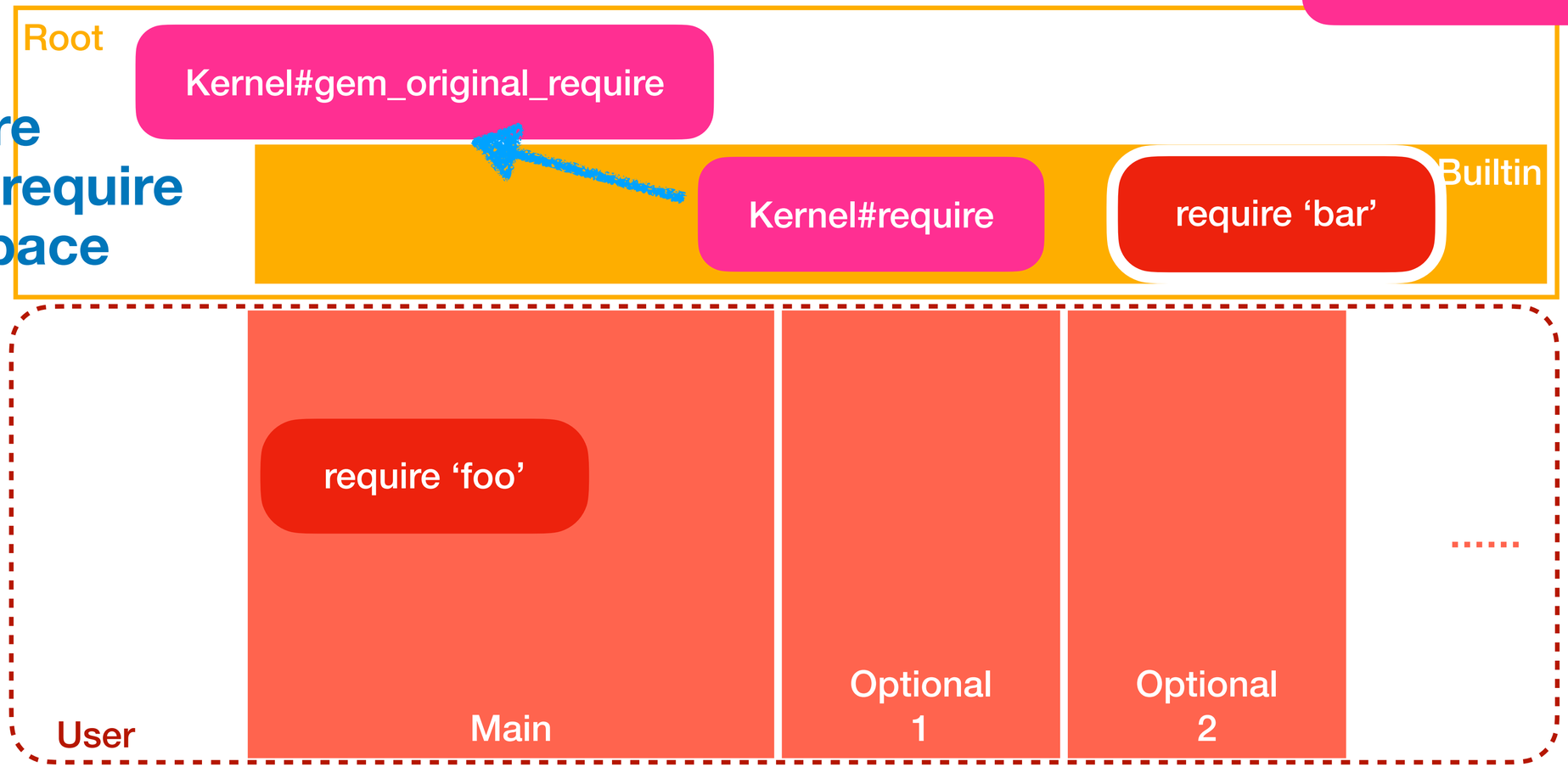
What Happens with #require

With RubyGems, and RubyGems .rb code

Ruby Code

Class/Module Methods

RubyGems #require calls the original #require in the root namespace (caller is builtin)

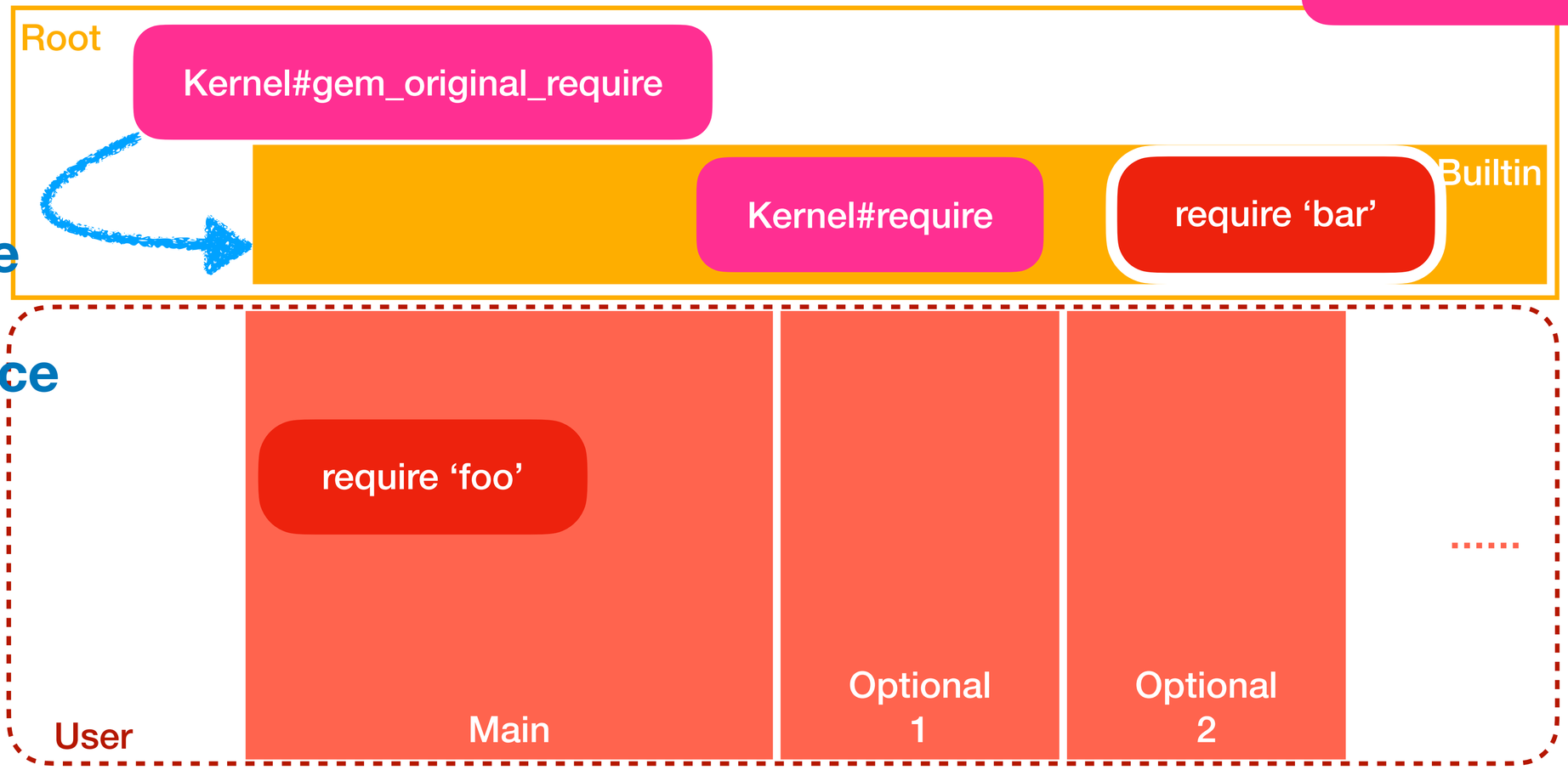


What Happens with #require

With RubyGems, and RubyGems .rb code

Ruby Code

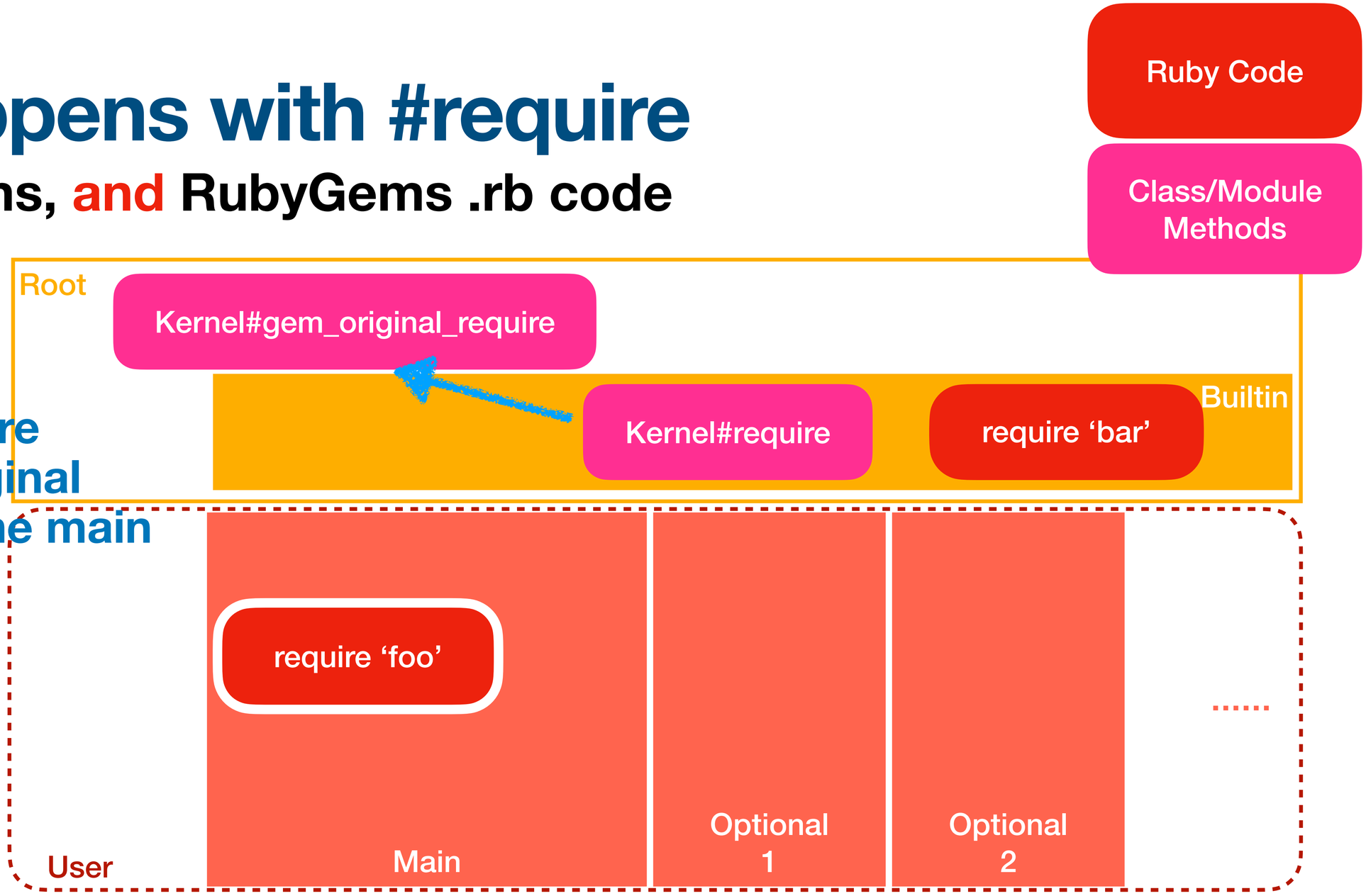
Class/Module Methods



The original #require loads the file into the builtin namespace (caller is builtin)

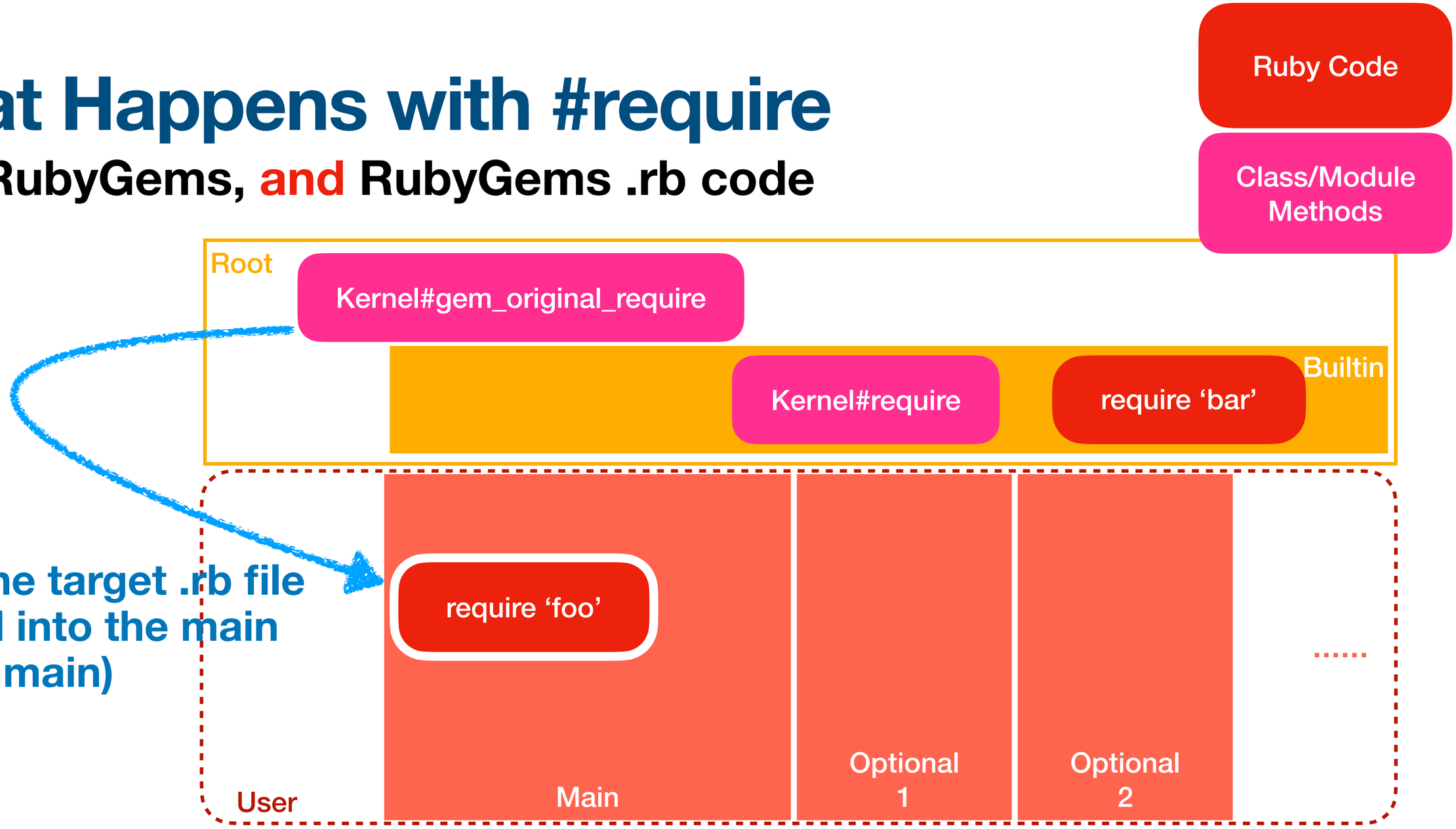
What Happens with #require

With RubyGems, and RubyGems .rb code



What Happens with #require

With RubyGems, and RubyGems .rb code



Finally, the target .rb file is loaded into the main (caller is main)

User Namespace
Implicit Mix-ins

Builtin Namespace
Implicit Refinements

Namespace is still WIP,
Mostly about just 1 SEGV.

Stay tuned til Ruby 3.4! (or …?)