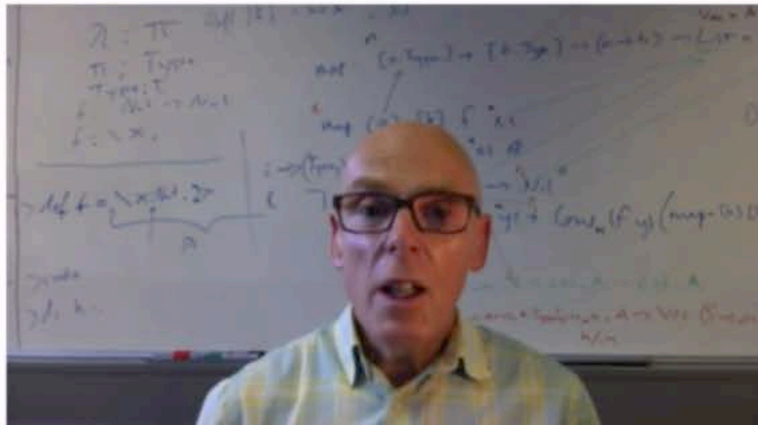

University of
Kent

University of Kent

Hot code loading

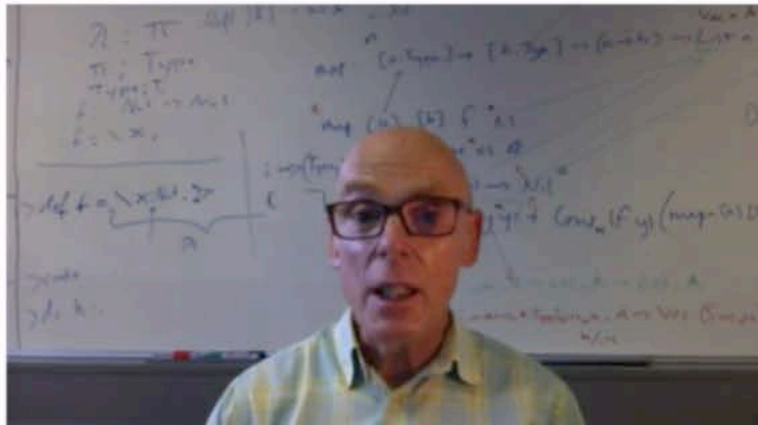


University of Kent

Hot code loading

New compiled code can be added to an Erlang system, one module at a time.

The system doesn't need to be taken down, and there is a mechanism to manage how the new code is used.

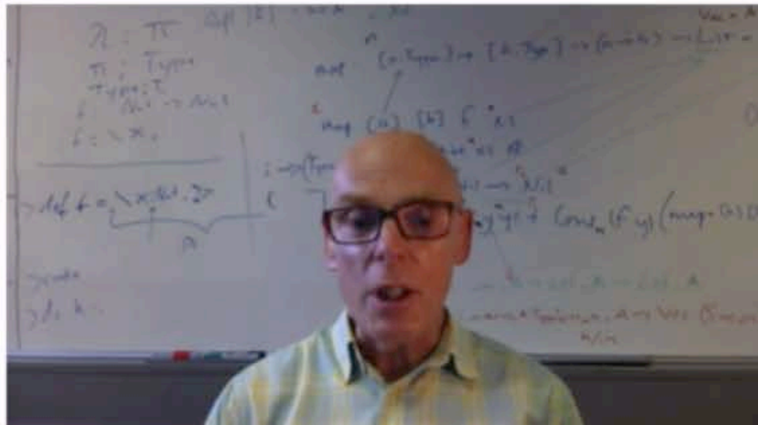


University of Kent

Loading code into the BEAM

How is the code for a module `Foo.erl` loaded into the BEAM?

- When a function from `Foo` is called, the `Foo.beam` file will be loaded, if it exists.
- When `Foo` is compiled, using `c(Foo)` in the shell or calling the Erlang function `compile:file(Foo)`, the module is loaded.
- When is is explicitly loaded by calling `code:load_file(Foo)`.



University of Kent

What is already loaded?

Is a module loaded? Call `code:is_loaded(Foo)` ...

... it returns the path of the `Foo.beam` file being used, or `false`.

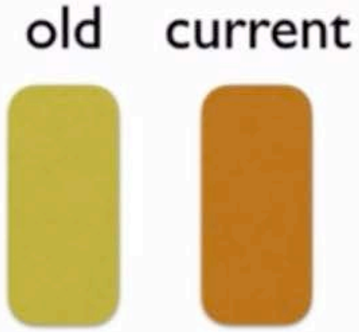
Hitting the `Tab` key in the shell will list all the modules loaded ...

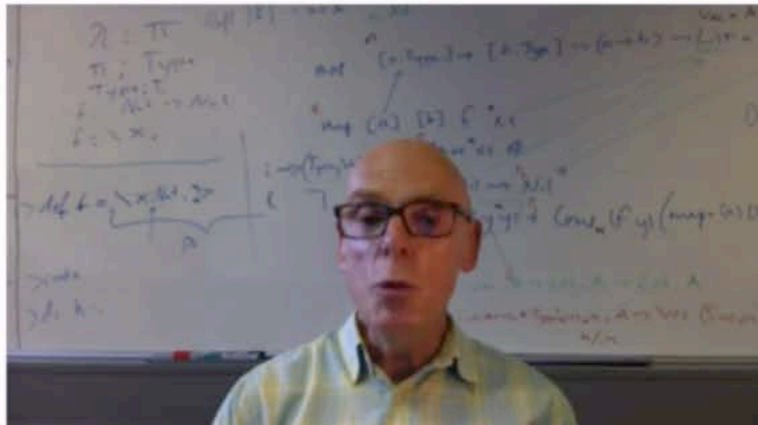
... give this a try: you'll see all the modules running the standard runtime system!

University of Kent

Old and current versions of a module

At any one time the system may contain up to two versions of a module: the *old* and the *current*.



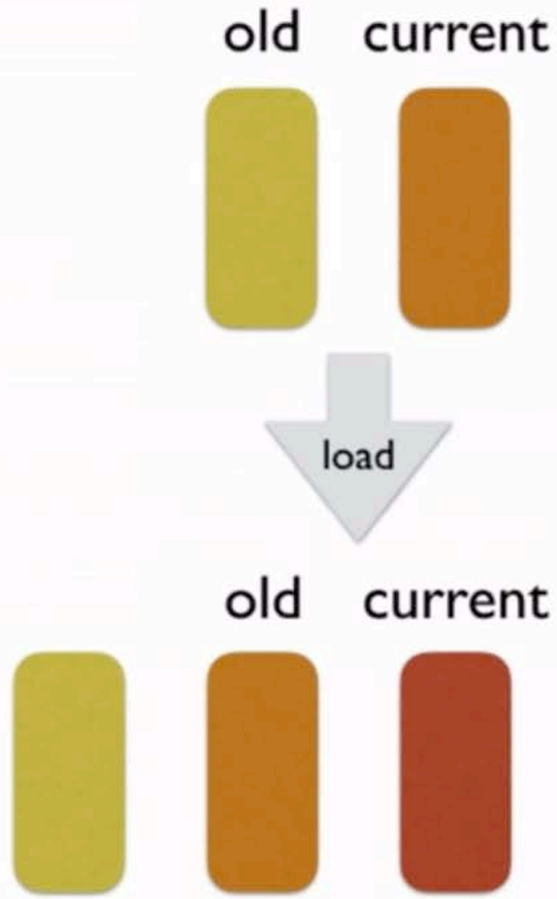


University of Kent

Old and current versions of a module

At any one time the system may contain up to two versions of a module: the *old* and the *current*.

Loading a new version makes that version the current, and the code it replaces becomes the old.



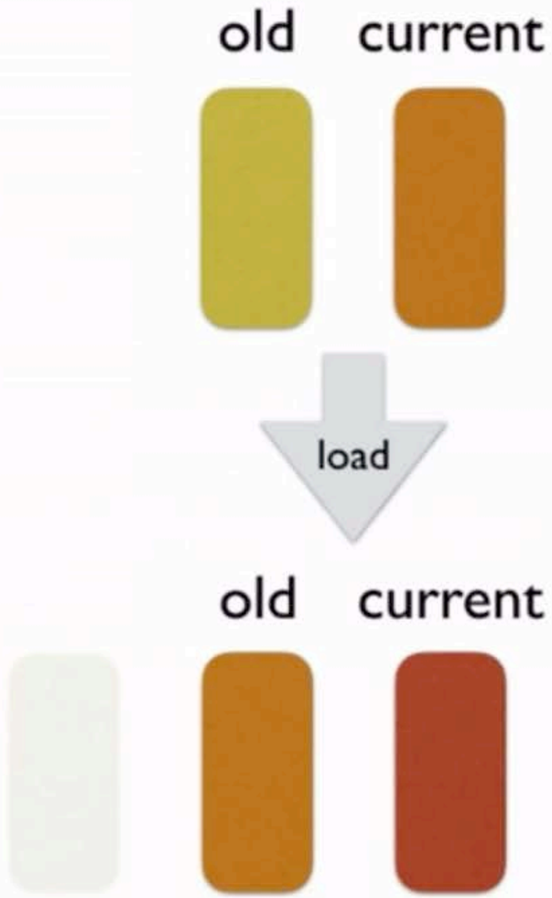
University of Kent

Old and current versions of a module

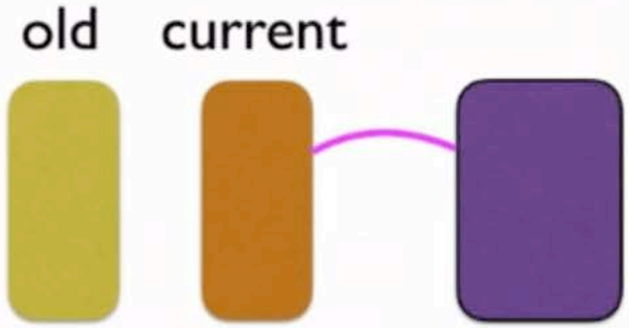
At any one time the system may contain up to two versions of a module: the *old* and the *current*.

Loading a new version makes that version the current, and the code it replaces becomes the old.

The previous old code is *purged*, and any module that is running that version will be terminated.

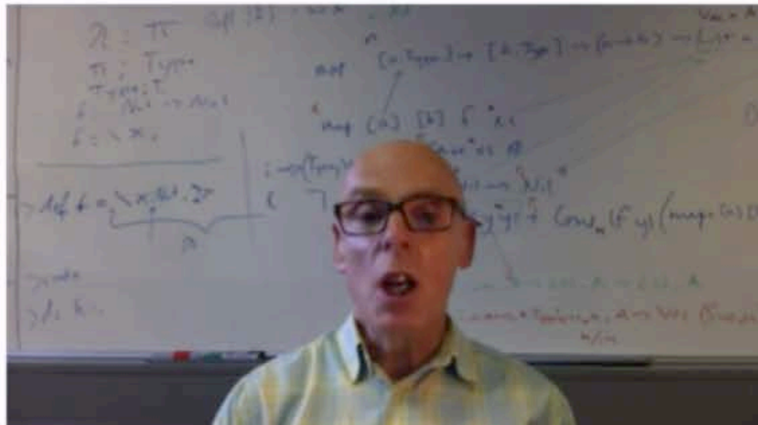


$64,000 question: using the new code

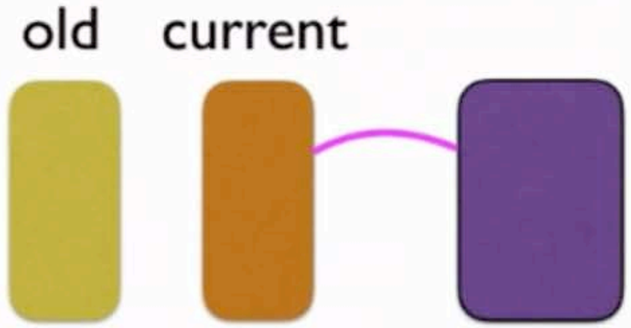


After the new code for `Foo` is loaded, the module using it will use the same code …

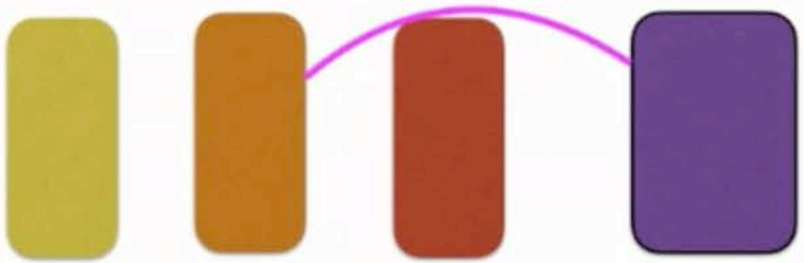
…until there's a call to any function in `Foo`, when it switches (for *all of* `Foo`).



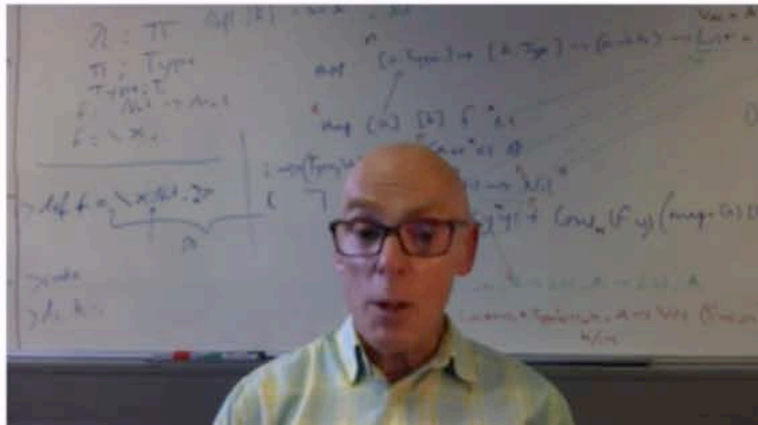
$64,000 question: using the new code



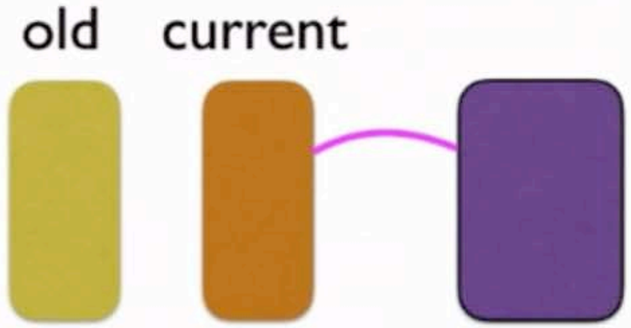
After the new code for `Foo` is loaded, the module using it will use the same code …



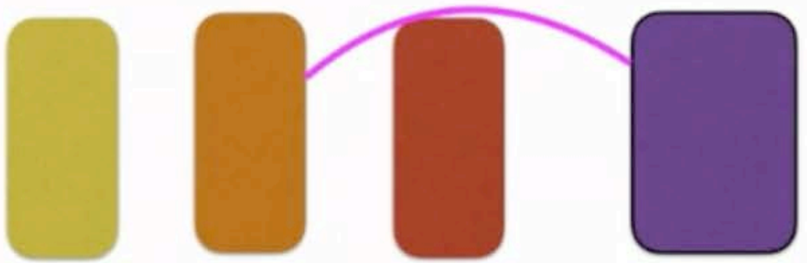
…until there's a call to any function in `Foo`, when it switches (for *all of* `Foo`).



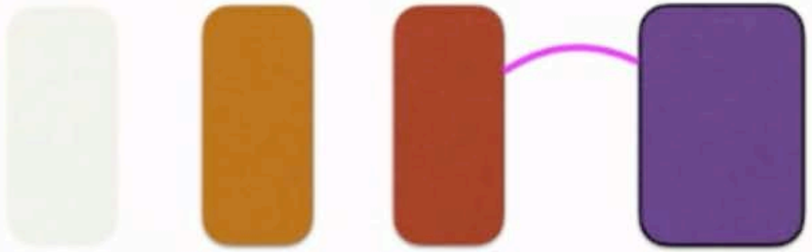
$64,000 question: using the new code



After the new code for `Foo` is loaded, the module using it will use the same code …



…until there's a call to any function in `Foo`, when it switches (for *all of* `Foo`).

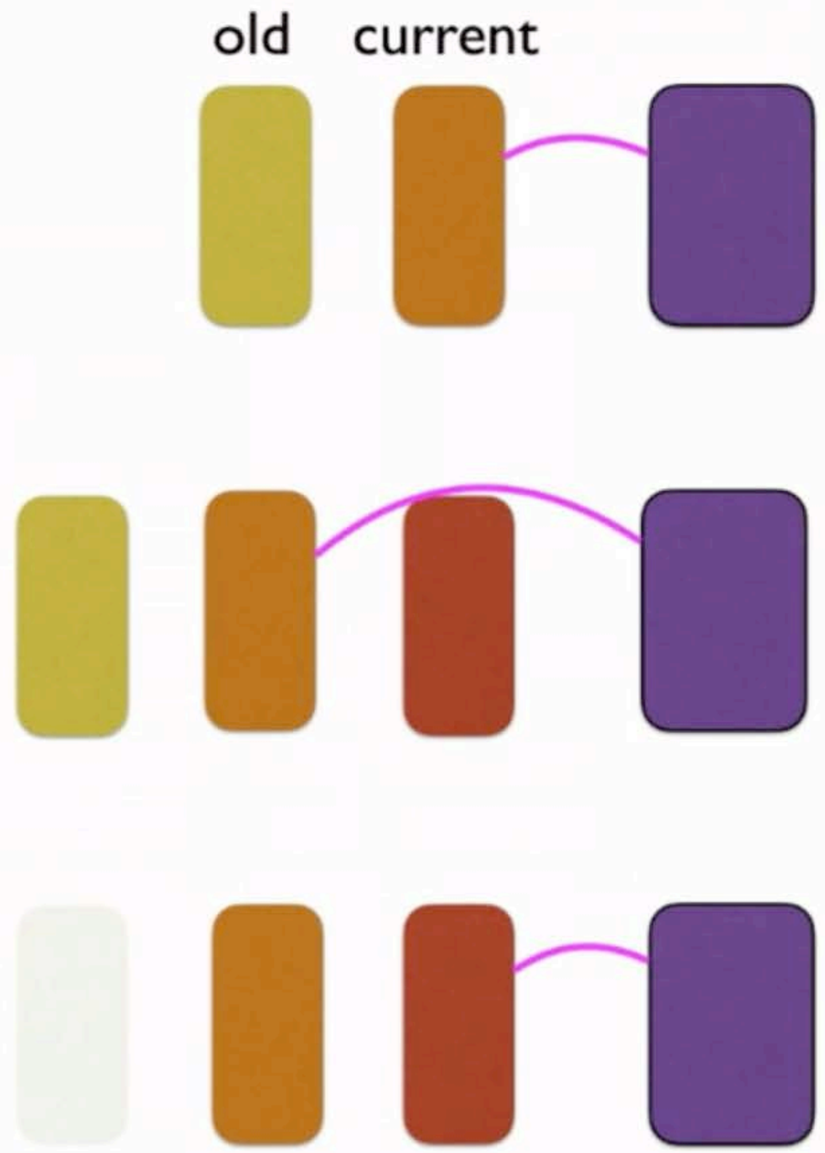


What about in the module itself?

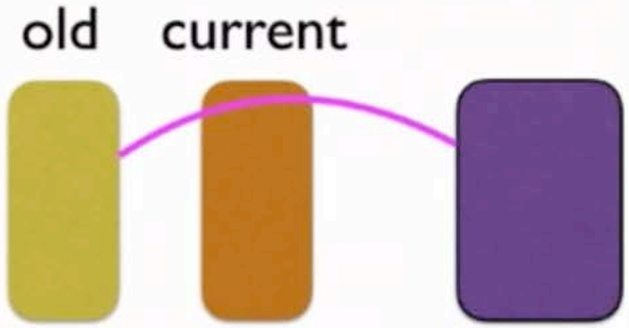
Same as before, except that

When there is a `Foo:foo` fully qualified call to any function in `Foo`, it switches to the new code for `Foo`.

Non-qualified calls `foo` *don't trigger the switch.*

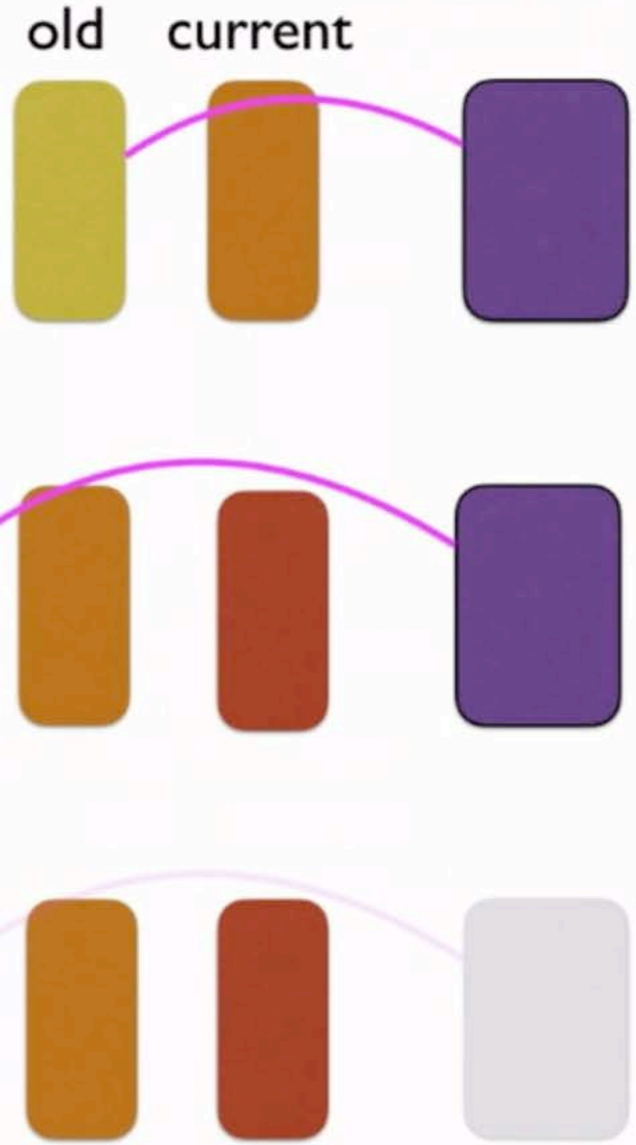


What about modules using the old code?



The old code will be purged, and the client module will be terminated.

What about modules using the old code?



The old code will be purged, and the client module will be terminated.

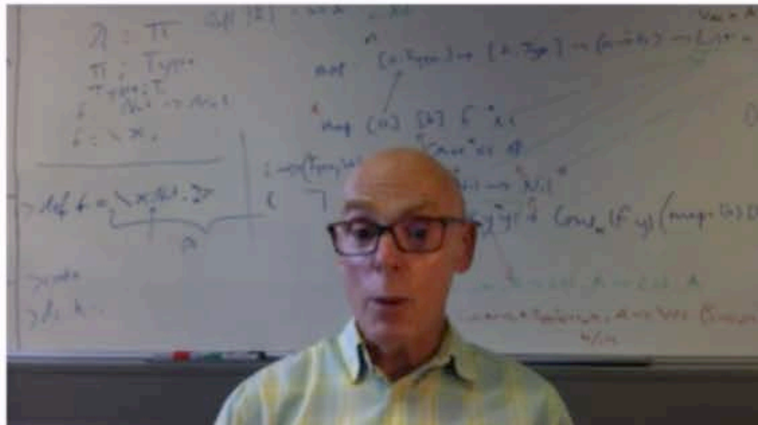
University of Kent

Removing the old version

Old versions of code can be purged by calling `code:purge(Foo)` or `code:soft_purge(Foo)`.

A call to `soft_purge/1` only succeeds if the old version of the code isn't being used; `purge/1` always succeeds but may cause a failure!

Why might it be useful to purge the old version?




University of
Kent