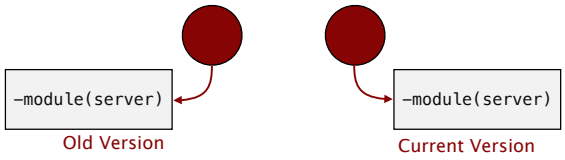
Erlang Solutions Ltd.

Code Updating

Overview: code updating

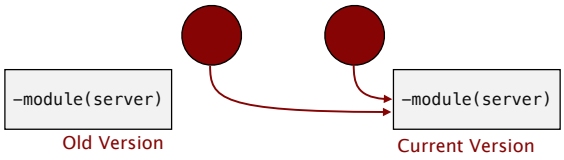
- Software Upgrade
- Code Server
- The .erlang File

Software Upgrade



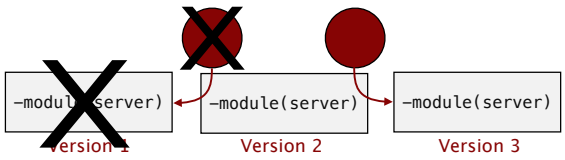
- Two versions of a module may be loaded in the run time system at any one time
- A process may run either version

Software Upgrade



- If the process runs the old version and a fully qualified function call is made, the module reference is updated to point to the latest version of the code

Software Upgrade



- When a third version of a module is loaded, the oldest version is purged and any process still running it is killed

Software Upgrade: example

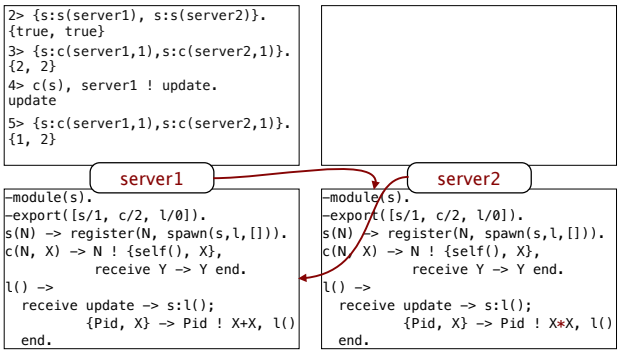
```
1> c(s).
{ok, s}
2> {s:s(server1), s:s(server2)}.
{true, true}
3> {s:c(server1,1),s:c(server2,1)}.
{2, 2}
```

server1

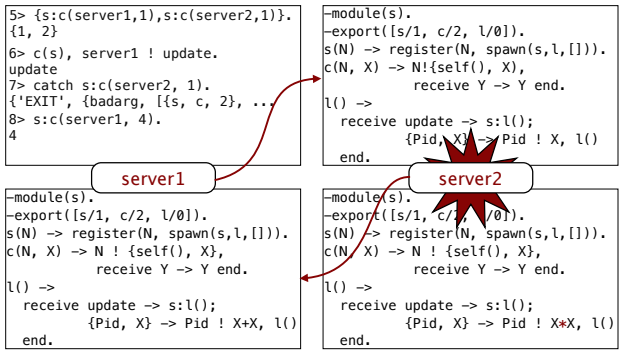
server2

```
-module(s).
-export([s/1, c/2, l/0]).
s(N) -> register(N, spawn(s,l,[])).
c(N, X) -> N ! {self(), X},
           receive Y -> Y end.
l() ->
  receive update -> s:l();
          {Pid, X} -> Pid ! X+X, l()
  end.
```

Software Upgrade: example



Software Upgrade: example



Software Upgrade

- Code is loaded in the run time system by:
  - Calling a function in a module which is not loaded
  - Compiling the module using **c(Module)**
  - Explicitly loading it with **code:load_file(Module)**
  - From the shell, use the **l(Module)** command
- Function calls where the module is prefixed are called **fully qualified function calls** (**M:F(Args)**)
- If the function call is not fully qualified, the process will continue running the old version of the code

Code Server

- The code server handles the dynamic loading of modules during run time
- A module is loaded in the system the first time a fully qualified call is made to it
- The code server will search the code path sequentially for a compiled version of the module

Code Server

- The code search path consists of a list of directories
- The directory elements are sequentially searched for the module we want to load
- Search paths can be viewed with **code:get_path()**
- Default directories include:
  - "." (Current working directory), $ERLANGROOT/lib/
- Directories can be added:
  - At the beginning with code:add_patha(Dir)
  - At the end by using code:add_pathz(Dir)

Code Server

```
1> code:add_patha("/Users/ferd/erlang").
true
2> code:get_path().
["/Users/ferd/erlang",
 ".",
 "/opt/local/lib/erlang/lib/kernel-2.14/ebin",
 "/opt/local/lib/erlang/lib/stdlib-1.17/ebin",
 "/opt/local/lib/erlang/lib/xmerl-1.2.5/ebin",
 "/opt/local/lib/erlang/lib/wx-0.98.6/ebin",
 "/opt/local/lib/erlang/lib/odbc-2.10.8",
 "/opt/local/lib/erlang/lib/observer-0.9.8.3/ebin",
 [...]|...]
```

Code Server

- The code server can remove the old version of a module
- **code:purge(Module)** will remove the old version and kill all processes running it, returning true if any process was killed
- **code:soft_purge(Module)** will remove the old version if no process is running it, returning true if the old version was removed

Software Upgrade: example

```
1> c(s), c(s), code:soft_purge(s).
true
2> s:s(server).
true
3> c(s), code:soft_purge(s).
false
4> s:c(server, 1).
2
5> code:purge(s).
true
6> catch s:c(server, 1).
{'EXIT', {badarg, [{s, c, 
7> c(s), code:purge(s).
false
```

server1 = s:l()

```
-module(s).
-export([s/1, c/2, l/0]).
s(N) -> register(N, spawn(s,l,[])).
c(N, X) -> N!{self(), X},
           receive Y -> Y end.
l() ->
  receive update -> s:l();
          {Pid,X} -> Pid ! X+X, l()
  end.
```


The .erlang File

- Placing valid Erlang expressions in a **.erlang** file will result in these expressions being executed every time the ERTS (Erlang Run Time System) is started
- The file is placed in the user's home directory
- It is useful for setting paths to your own modules and tools, or other common expressions such as:
  - `code:addpatha(Path)`
  - `erlang:set_cookie(node(), Cookie)`
  - `io:format(Message)`

Summary: code updating

- Software Upgrade
- Code Server
- The .erlang File