Integrating OS Package Management and the Erlang VM

An Erlang Startup

- Write awesome Erlang code.
- ???
- Profit.

An Erlang Startup

- Write awesome Erlang code.
- ???
- Deploy to EC2.
- ???
- Profit.

An Erlang Startup

- Write awesome Erlang code.
- ???
- Deploy to EC2.
- ???
- Profit.

How do I launch my software

How do I launch my software

… to EC2

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

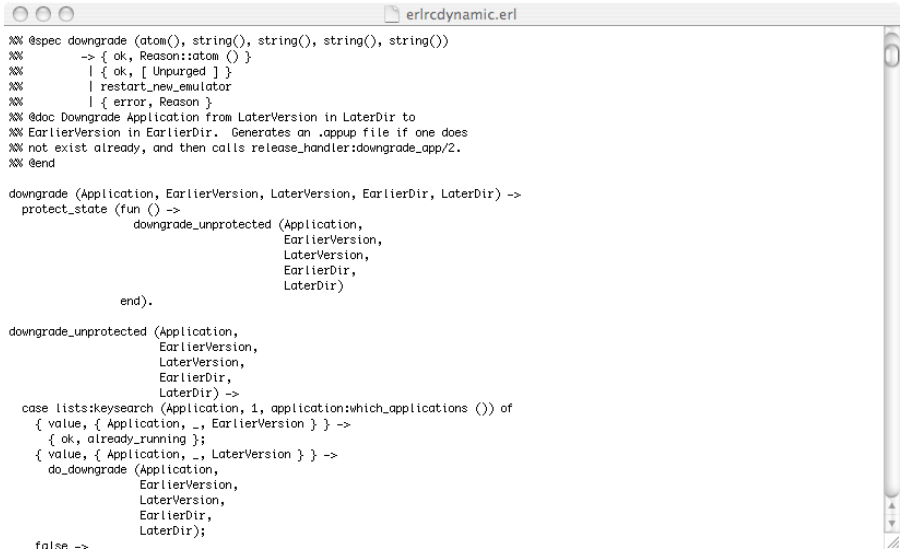
erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure



Erlang/OTP software structure

erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                 end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

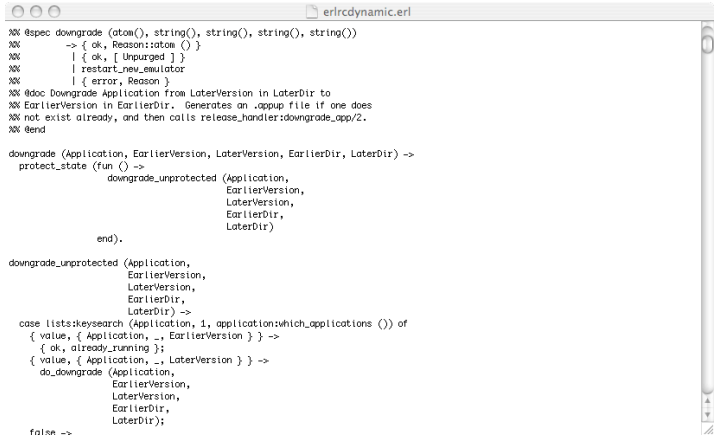
erlrcdynamic.erl

```
%% @spec downgrade (atom(), string(), string(), string(), string())
%%         -> { ok, Reason::atom () }
%%          | { ok, [ Unpurged ] }
%%          | restart_new_emulator
%%          | { error, Reason }
%% @doc Downgrade Application from LaterVersion in LaterDir to
%% EarlierVersion in EarlierDir.  Generates an .appup file if one does
%% not exist already, and then calls release_handler:downgrade_app/2.
%% @end

downgrade (Application, EarlierVersion, LaterVersion, EarlierDir, LaterDir) ->
  protect_state (fun () ->
                   downgrade_unprotected (Application,
                                          EarlierVersion,
                                          LaterVersion,
                                          EarlierDir,
                                          LaterDir)
                end).

downgrade_unprotected (Application,
                       EarlierVersion,
                       LaterVersion,
                       EarlierDir,
                       LaterDir) ->
  case lists:keysearch (Application, 1, application:which_applications ()) of
    { value, { Application, _, EarlierVersion } } ->
      { ok, already_running };
    { value, { Application, _, LaterVersion } } ->
      do_downgrade (Application,
                    EarlierVersion,
                    LaterVersion,
                    EarlierDir,
                    LaterDir);
    false ->
```

Erlang/OTP software structure

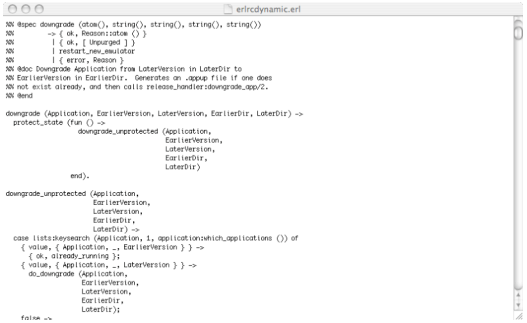


Erlang/OTP software structure

Erlang/OTP software structure

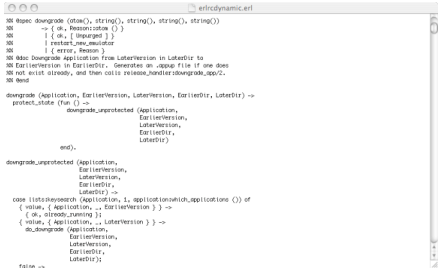
Erlang/OTP software structure

Erlang/OTP software structure

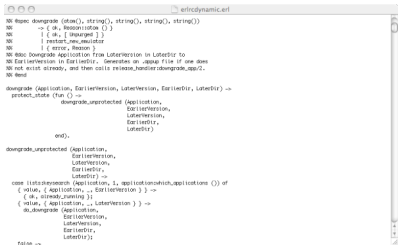


Erlang/OTP software structure

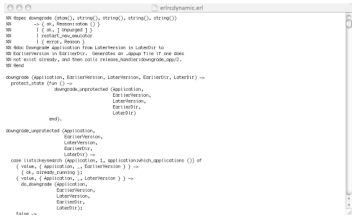
Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP software structure

Erlang/OTP software structure

cool.erl

Erlang/OTP software structure

cool.erl

Erlang/OTP software structure

cool.erl

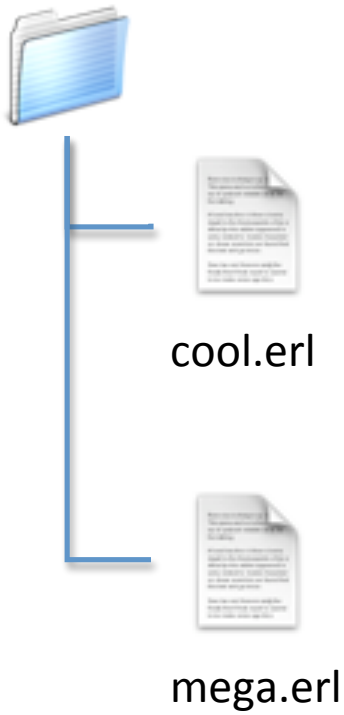
Erlang/OTP software structure

cool.erl

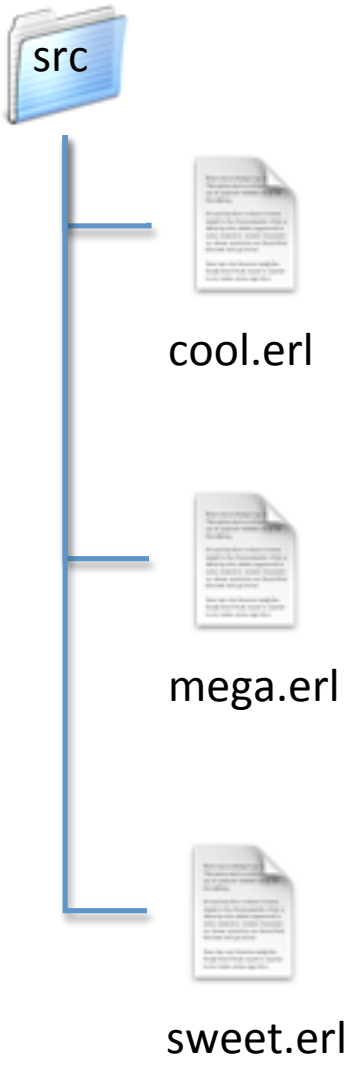
Erlang/OTP software structure

cool.erl

Erlang/OTP software structure



Erlang/OTP software structure



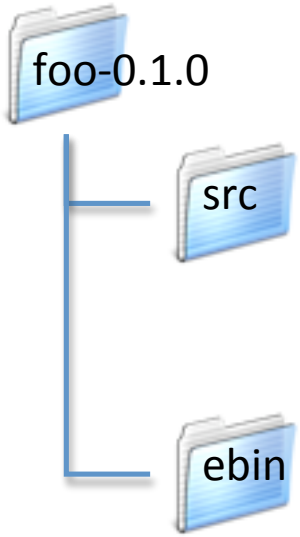
Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

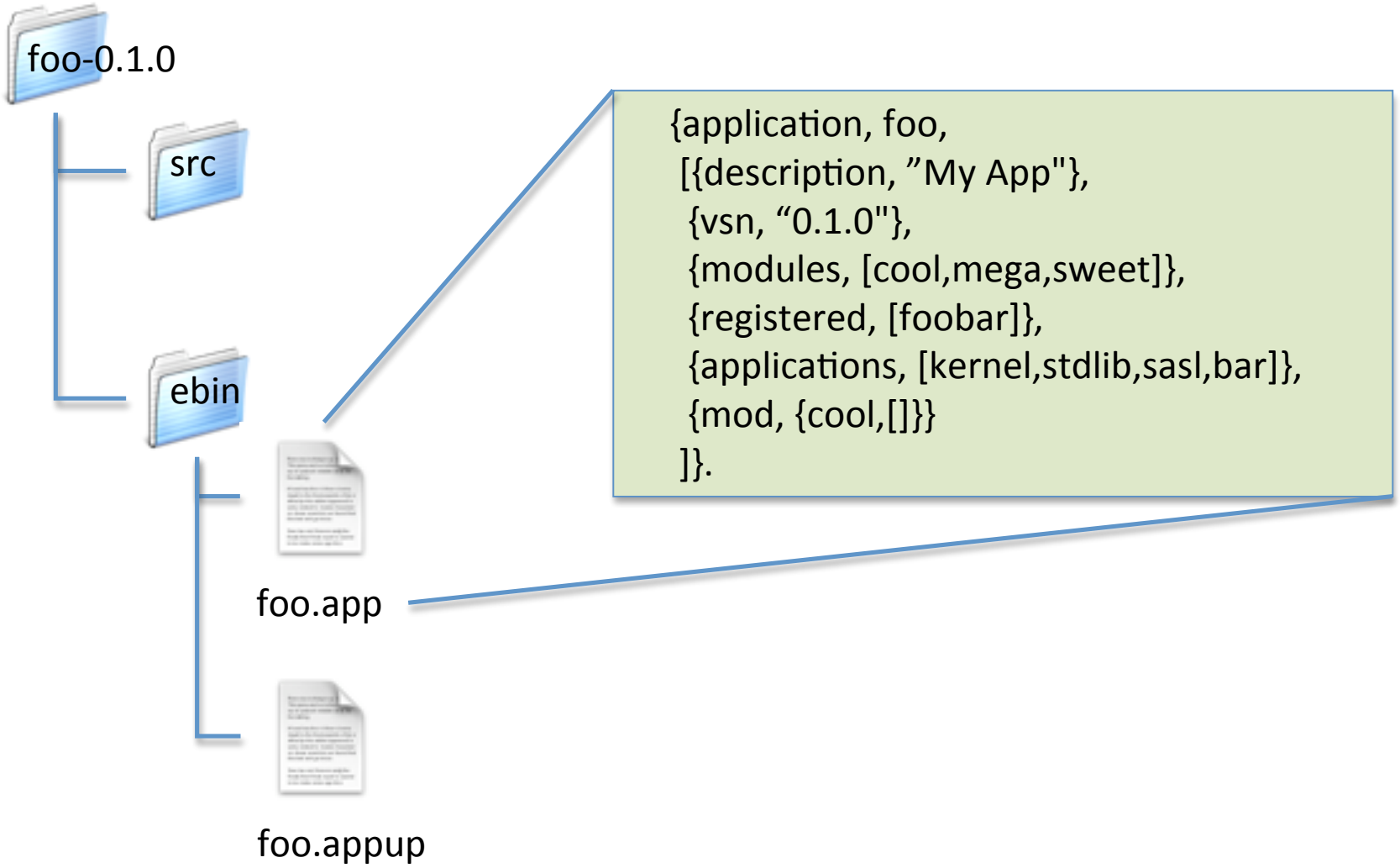
Erlang/OTP software structure

Erlang/OTP software structure

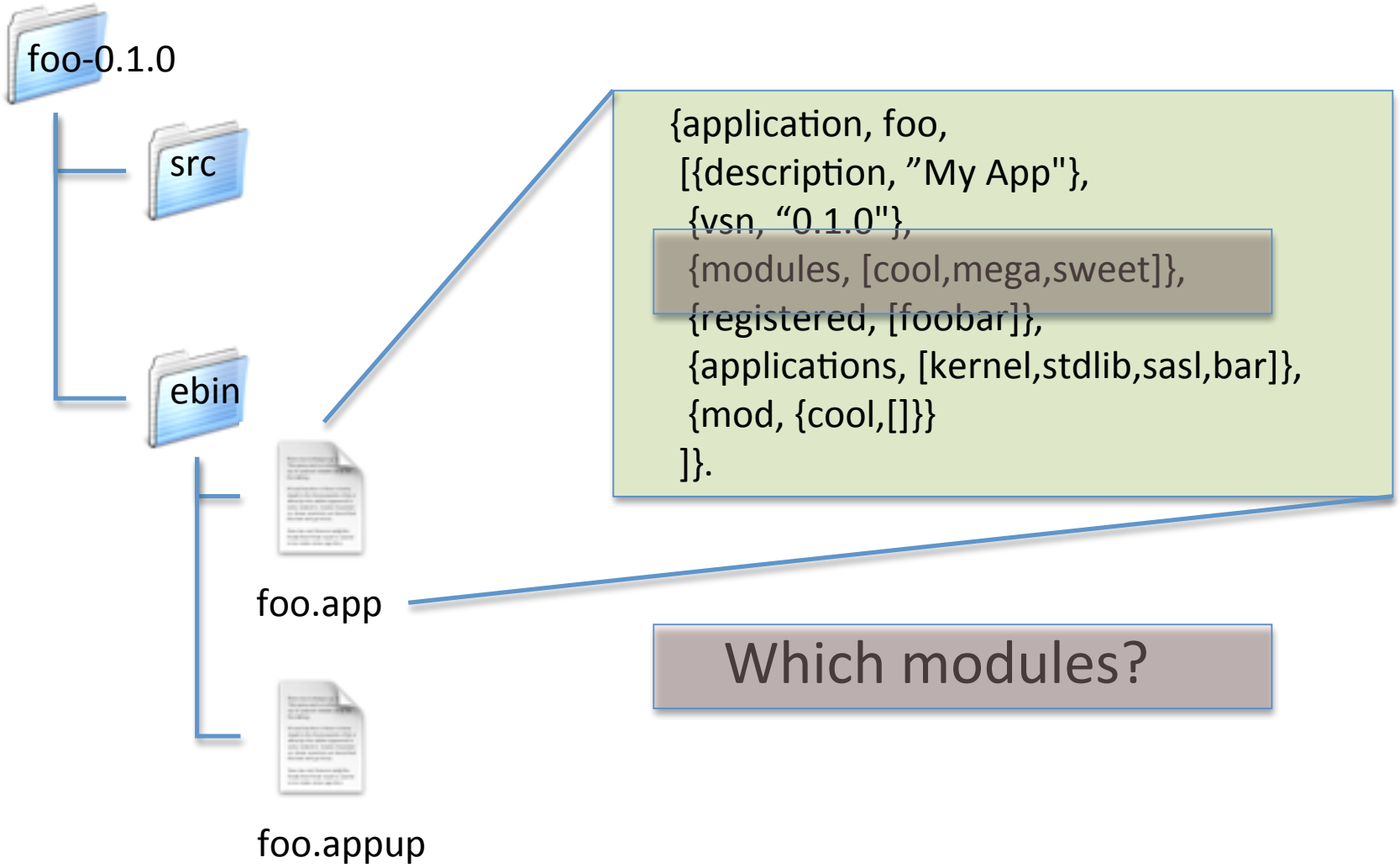


Erlang/OTP software structure

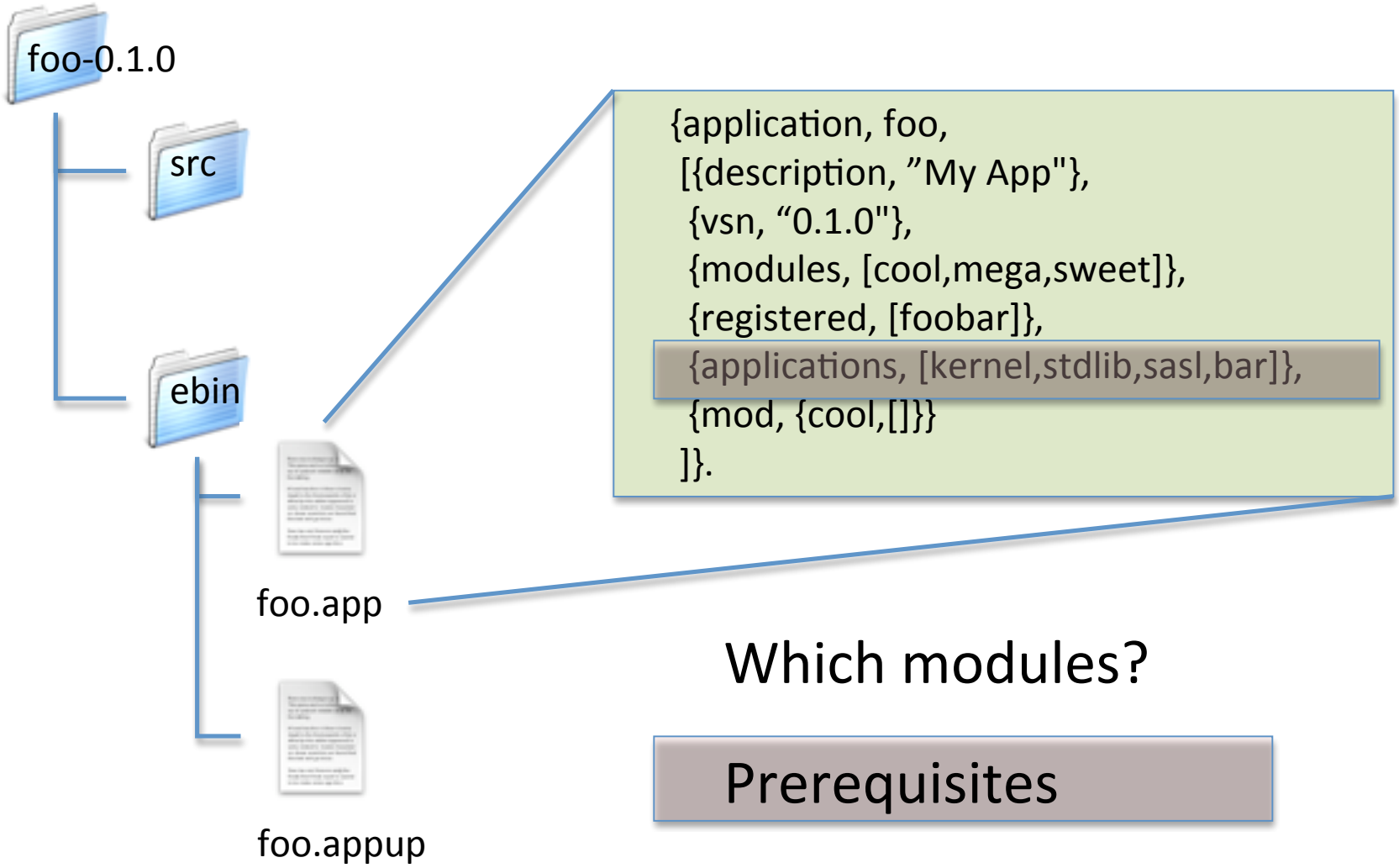
Erlang/OTP software structure



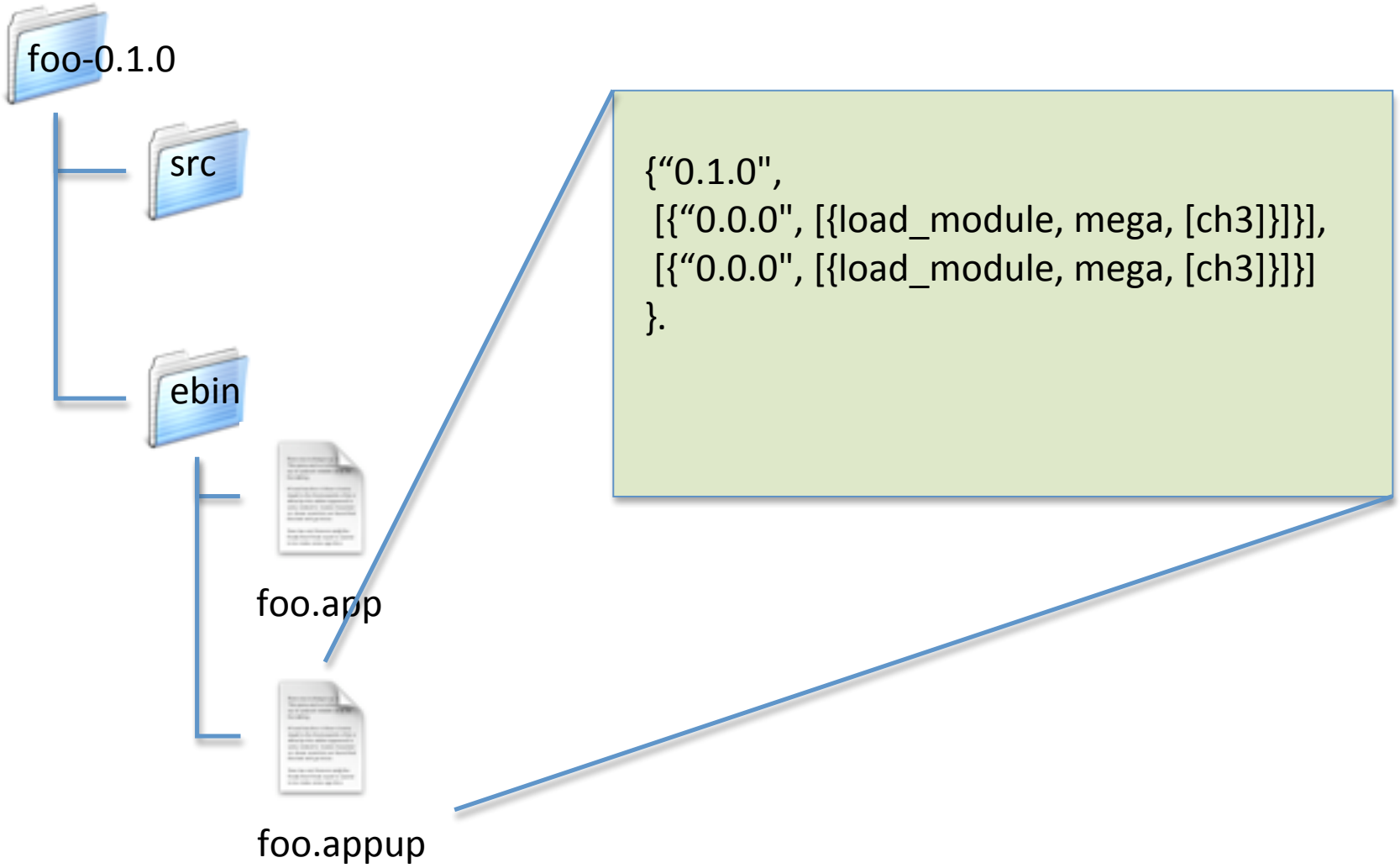
Erlang/OTP software structure



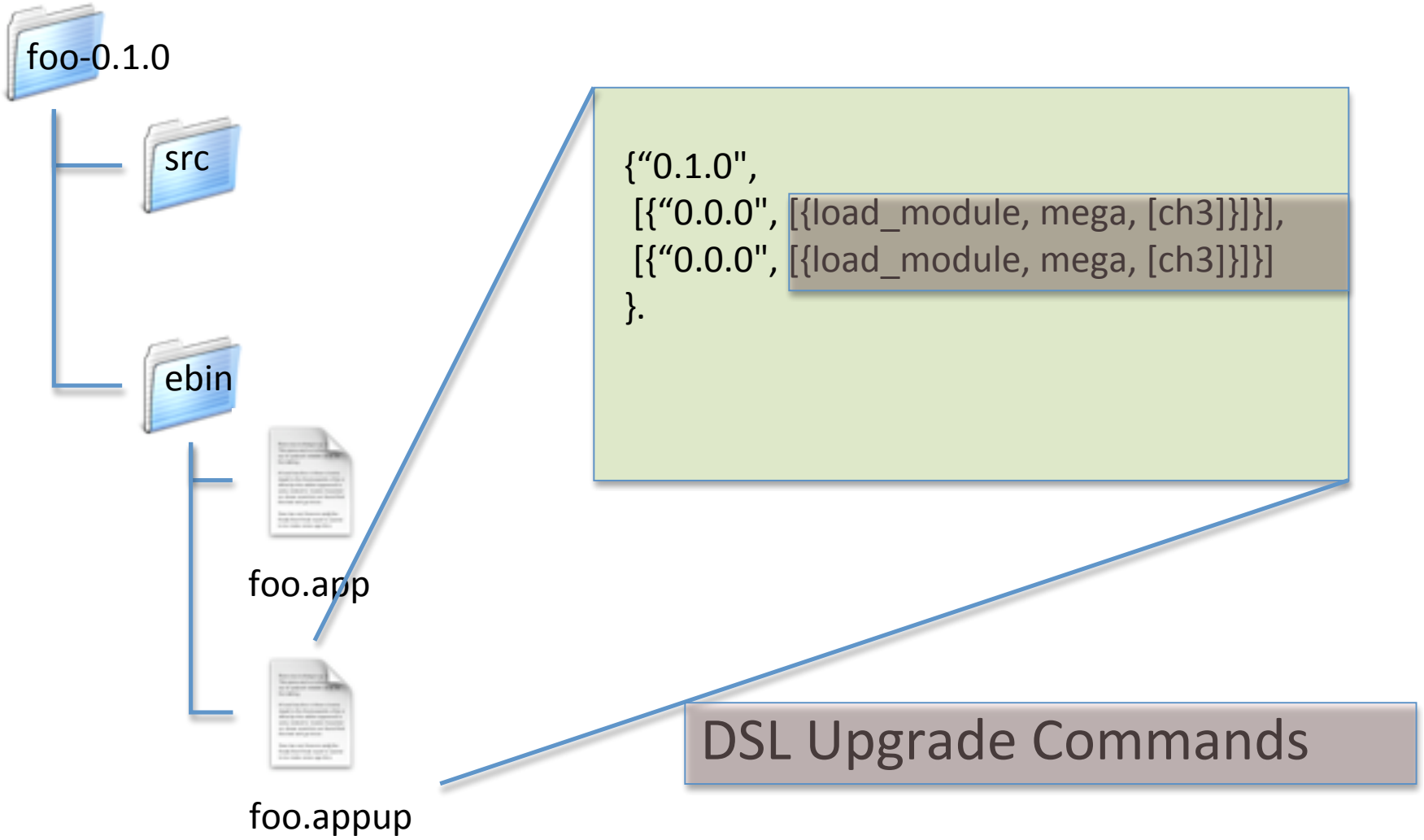
Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

Erlang/OTP software structure

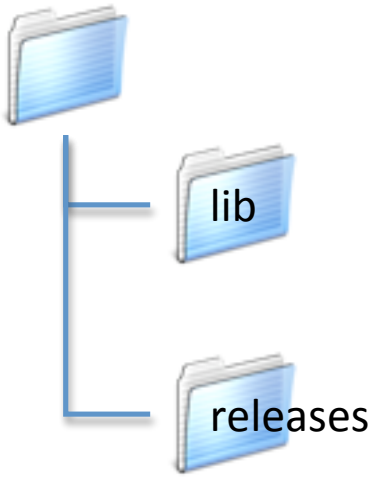
Erlang/OTP software structure

Erlang/OTP software structure

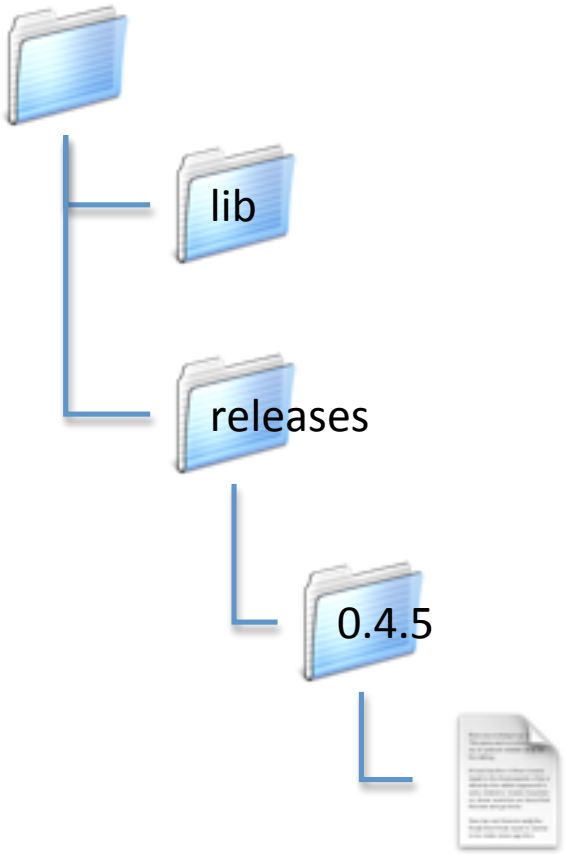
Erlang/OTP software structure

Erlang/OTP software structure

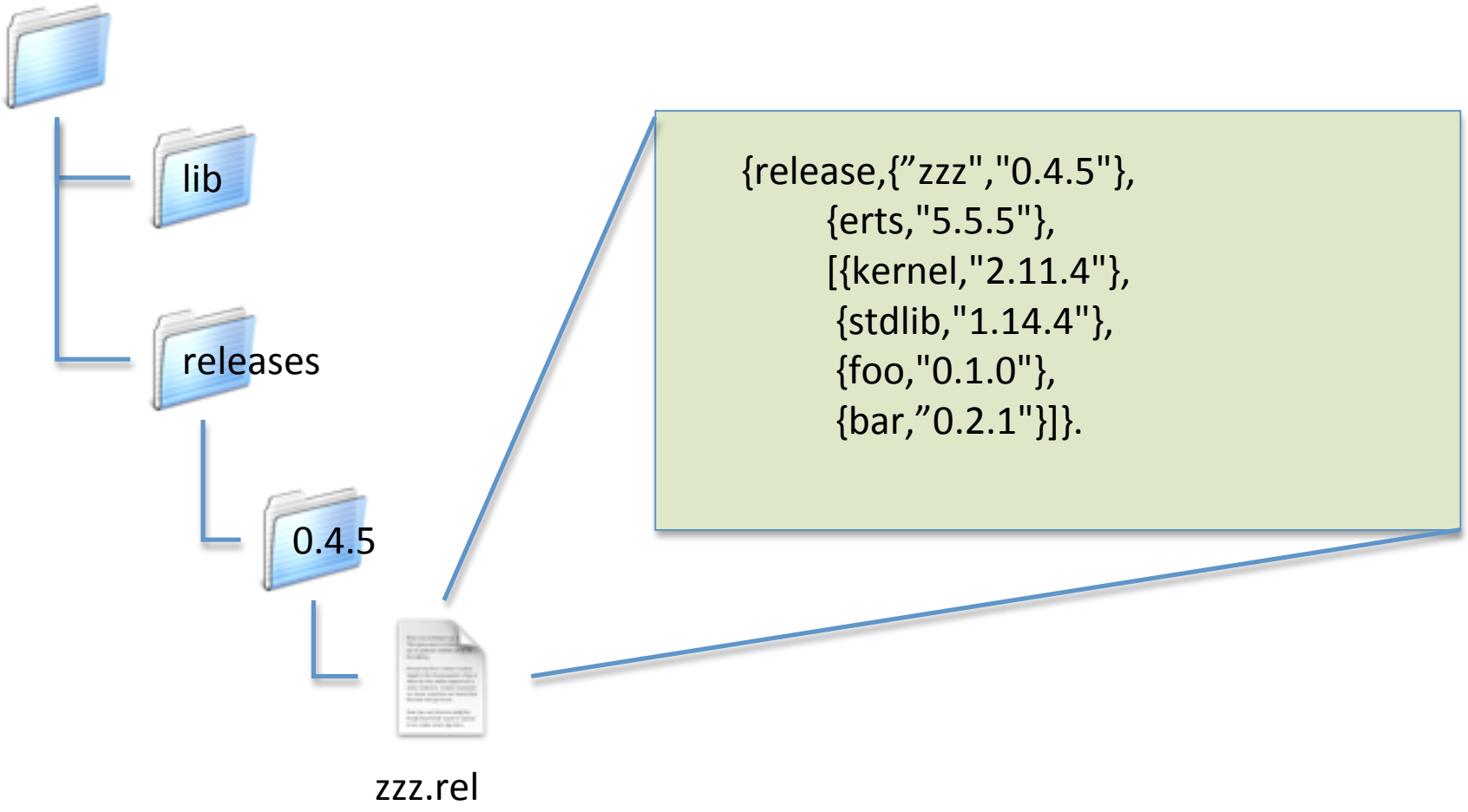
Erlang/OTP software structure



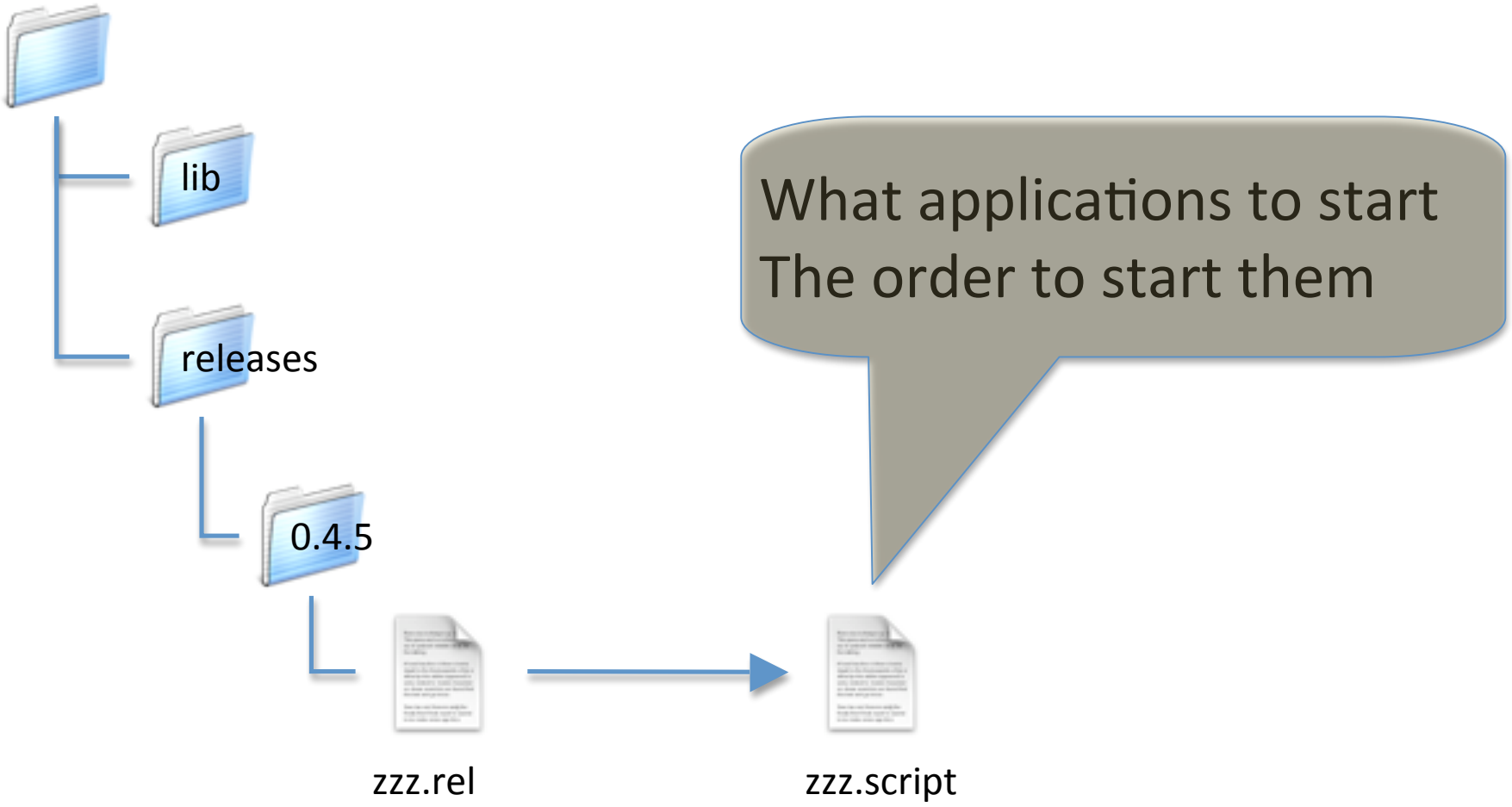
Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP software structure



Erlang/OTP Software Structure

- A module is a unit of code.
- An application is a collection of modules.
  - Defines how to start and stop.
  - Defines how to upgrade/downgrade.
- A release is a collection of applications.
  - Defines what constitutes the system.
  - Defines how to start the system.
  - Also can define upgrade/downgrade.

Erlang/OTP Launch Strategy

- Collect versioned applications together.
  - **Defines** the new state of the system.
  - Complete, not incremental.
- Create the release using OTP tools.
- Make the release accessible.
  - The **deployment** problem.
- Upgrade using release handler tools.

Large Companies

- Infrequent complete releases.
- Heavy integrative QA.

~~Large Companies~~

Start Up

- Infrequent complete releases
- Heavy integrative QA

~~Large Companies~~

Start Up

- ~~Infrequent complete releases~~

Incremental

- Heavy integrative QA

~~Large Companies~~

Start Up

- ~~Infrequent complete releases~~

Incremental

- ~~Heavy integrative QA~~

Light Pointwise

Startup

- “Agile” launch processes

Large Companies

- Infrequent complete releases
- Heavy integrative QA

Use the OS Package Manager

- Uniformity.
- Dependency Specification.
  - Between Erlang applications.
  - Between Erlang and non-Erlang components.
- Installation hooks.
  - Hot code management.
- Provisioning.
- Deployment.

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

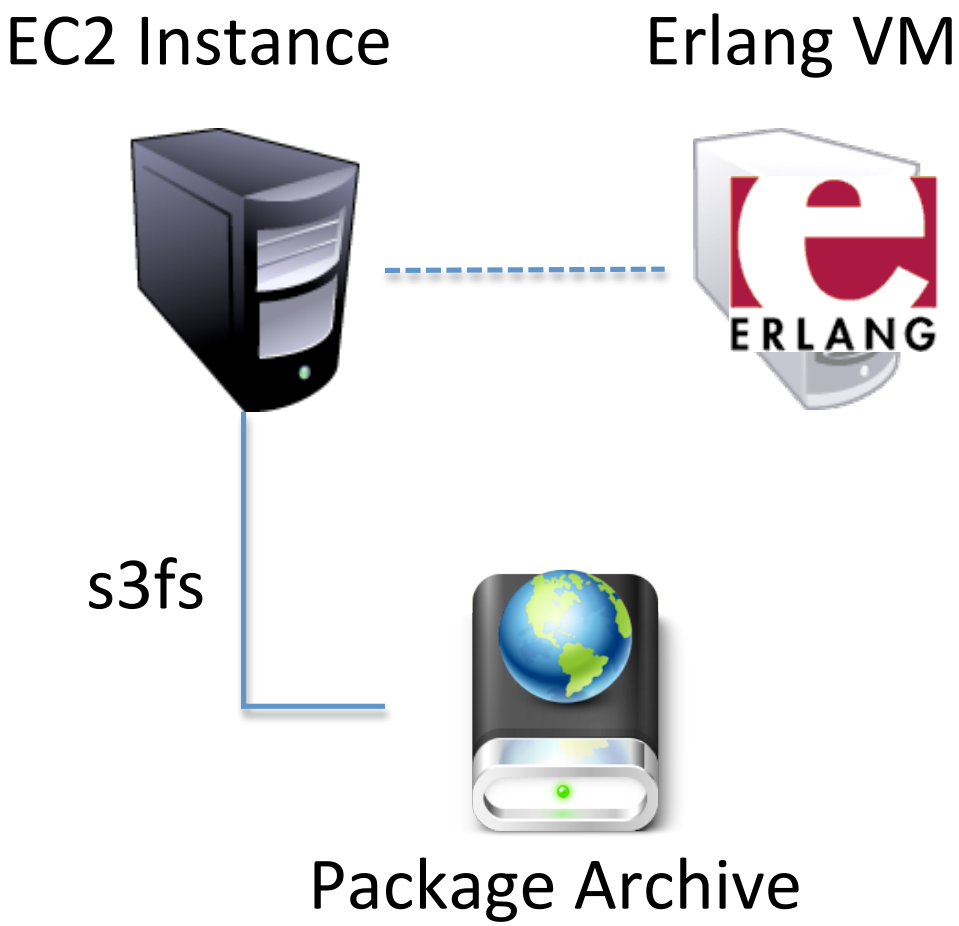
1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM

1 Xen Instance = 1 Erlang VM



1 Package = 1 Application

- Small enough to launch frequently.
  - No need to assemble all application changes.
  - “Release” = set of packages.
- Large enough to be interesting.
  - Supervision hierarchy.
  - Upgrade semantics.

Our Launch Process

- Create application OS package.
  - Province of the build system.
- Upload to package archive.
  - Standard tools available.
- Install the package on server.
  - Automatically starts, stops, upgrades, downgrades applications as appropriate.

Our Launch Process

- Create application OS package.
  - Province of the build system.
- Upload to package archive.
  - Standard tools available.
- Install the package on server.
  - Automatically starts, stops, upgrades, downgrades applications as appropriate.

Erlrc: The Vision …

```
pmineiro@ub32srvvmw-199% sudo apt-get install egerl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  egerl
1 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
Need to get 0B/113kB of archives.
After unpacking 0B of additional disk space will be used.
WARNING: The following packages cannot be authenticated!  egerl
Install these packages without verification [y/N]? y
(Reading database ... 48221 files and directories currently installed.)
Preparing to replace egerl 4.0.1 (using .../archives/egerl_4.1.0_all.deb) ...
Unpacking replacement egerl ...
erlrc-upgrade: Upgrading 'egerl': (cb8eec1a1b85ec017517d3e51c5aee7b) upgraded
Setting up egerl (4.1.0) ...
erlrc-start: Starting 'egerl': (cb8eec1a1b85ec017517d3e51c5aee7b) already_running
```

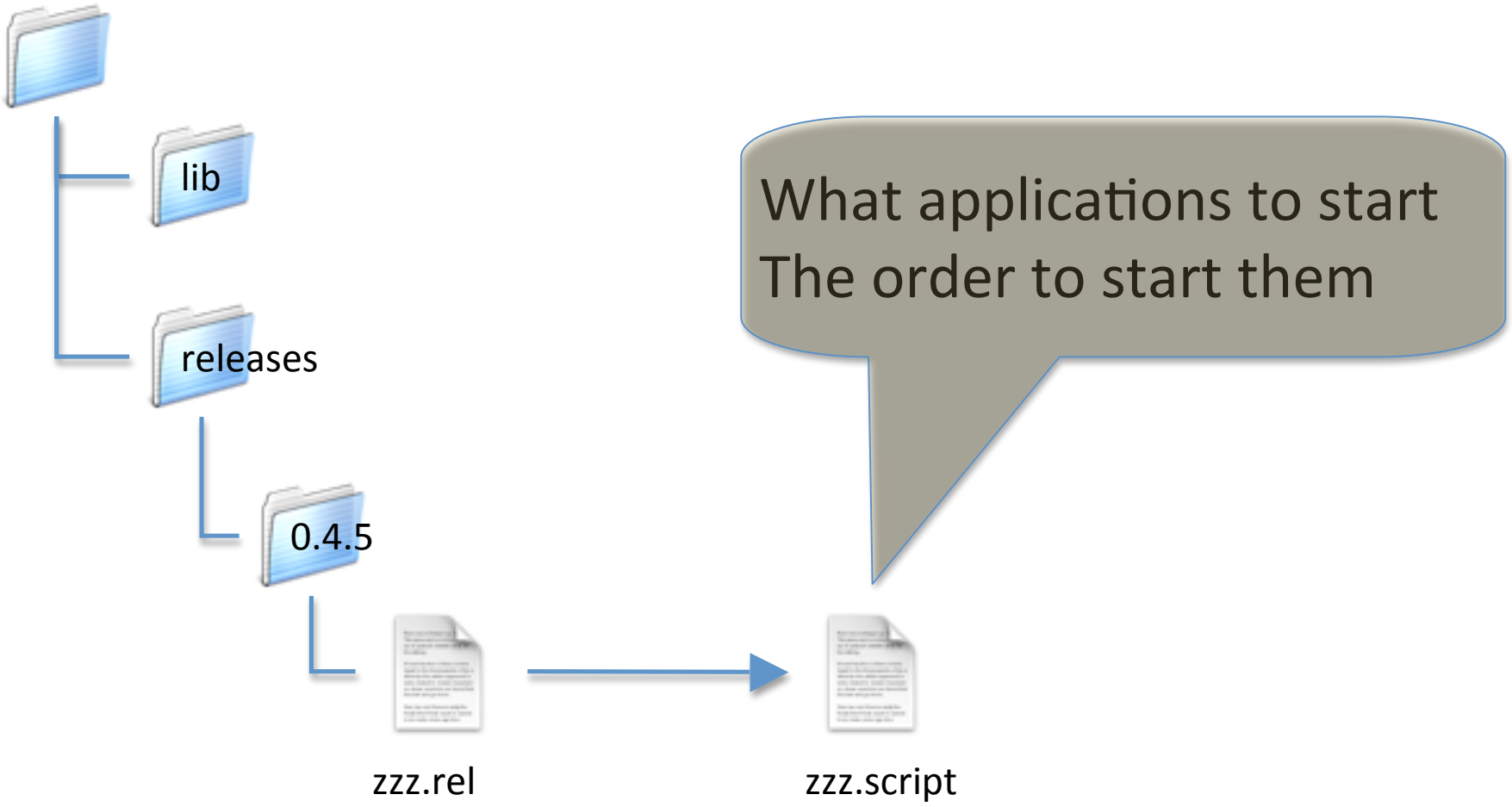
Erlrc: The Vision …

```
pmineiro@ub32srvvmw-199% sudo apt-get install egerl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  egerl
1 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
Need to get 0B/113kB of archives.
After unpacking 0B of additional disk space will be used.
WARNING: The following packages cannot be authenticated!  egerl
Install these packages without verification [y/N]? y
(Reading database ... 48221 files and directories currently installed.)
Preparing to replace egerl 4.0.1 (using .../archives/egerl_4.1.0_all.deb) ...
Unpacking replacement egerl ...
```

Problems to solve

- Static: booting the VM properly
- Dynamic: hot install/upgrade/remove

Erlang VM Boot

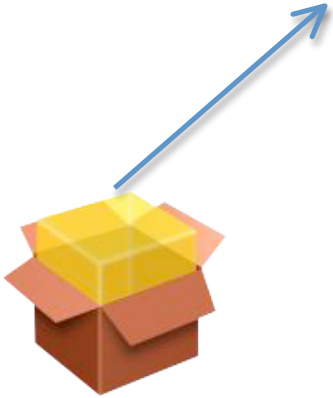


Config File vs. Config Directory

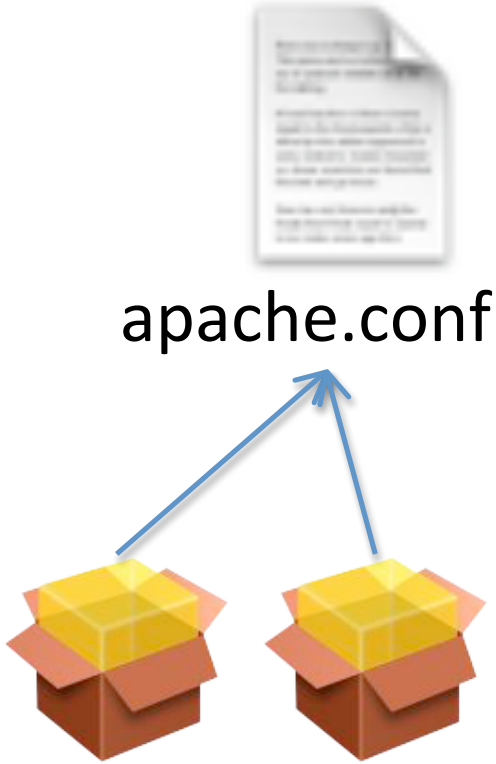
apache.conf

Config File vs. Config Directory

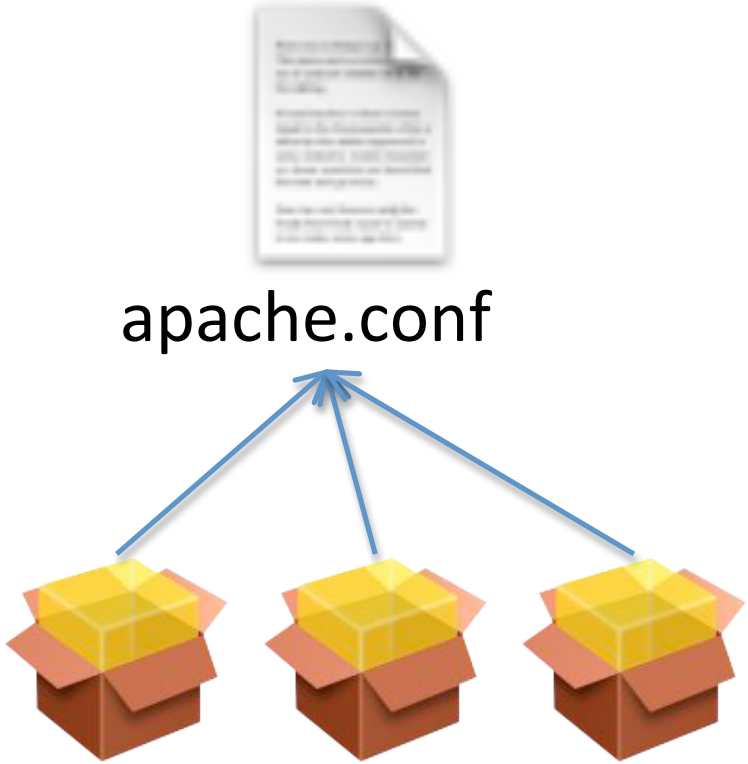
apache.conf



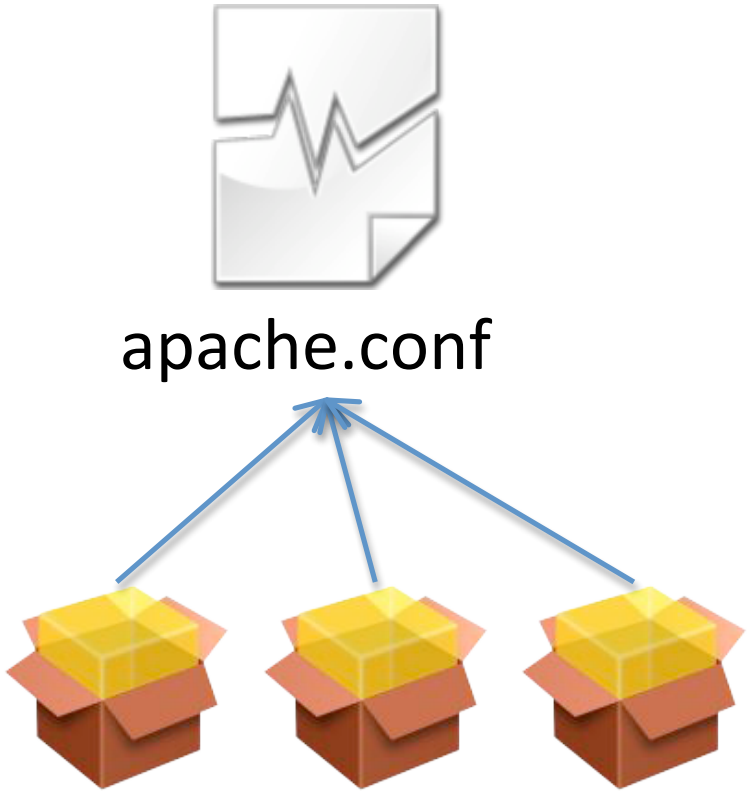
Config File vs. Config Directory



Config File vs. Config Directory



Config File vs. Config Directory



Config File vs. Config Directory

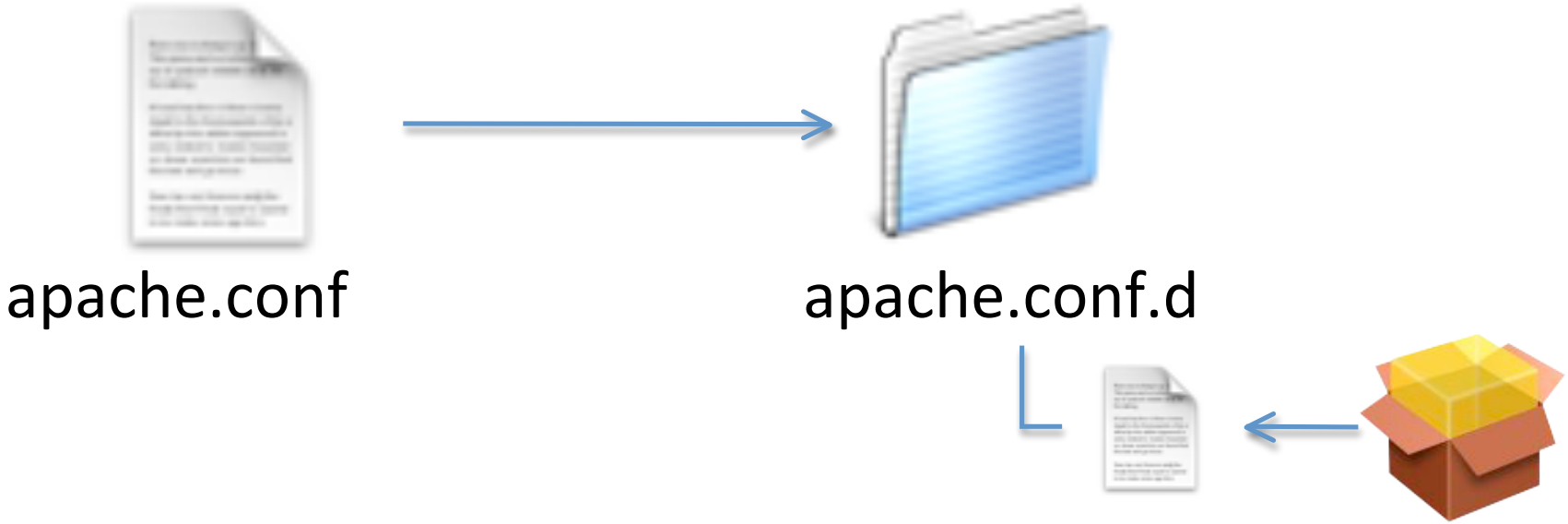
apache.conf

Config File vs. Config Directory

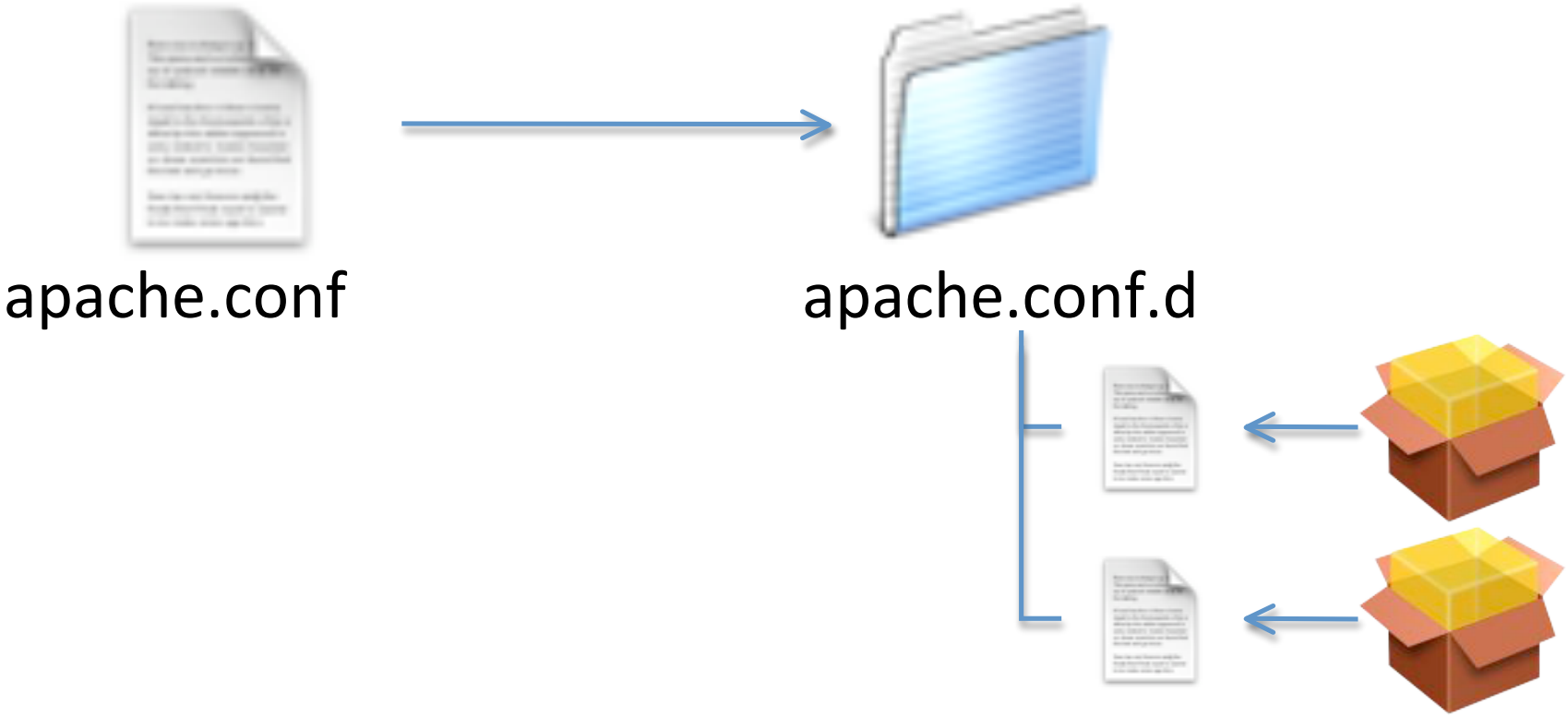
apache.conf

apache.conf.d

Config File vs. Config Directory



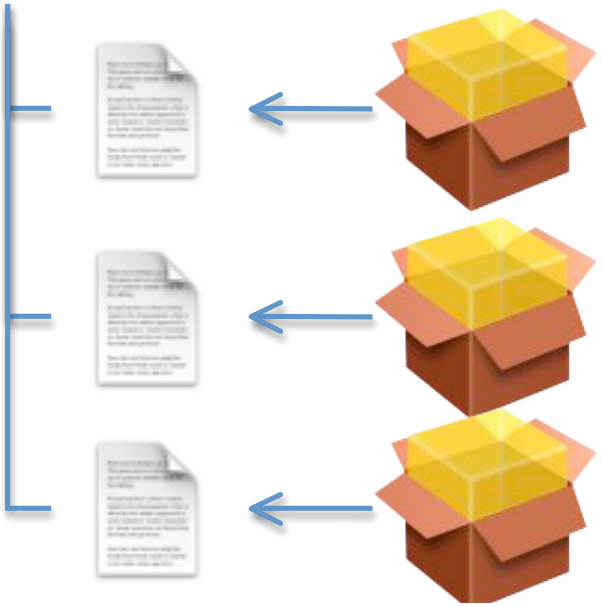
Config File vs. Config Directory



Config File vs. Config Directory

apache.conf

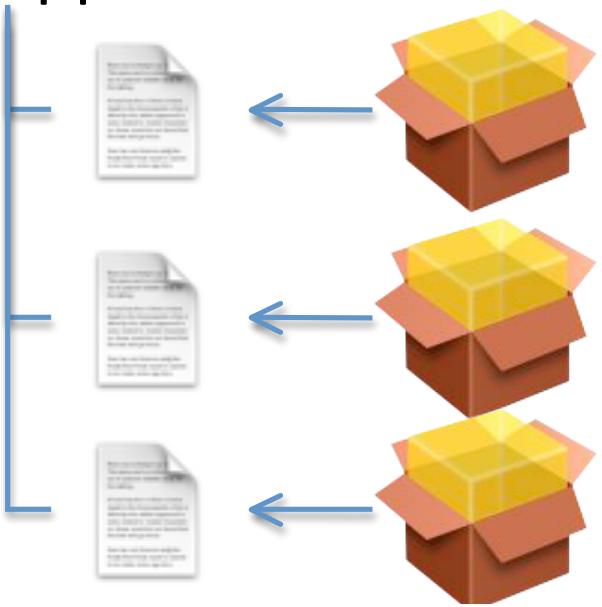
apache.conf.d



Config File vs. Config Directory

zzz.script

/etc/erlrc.d/applications



Actual Applications Directory

```
% ls /etc/erlrc.d/applications
appinspect       egerl   fragmentron  genherd       nodefinder    zfile
combonodefinder  erlrc   fuserl       loggins       schemafinder
ec2nodefinder    erlsom  gencron      n54etsbugfix  virtuerl
```

Actual Applications Directory

```
% ls /etc/erlrc.d/applications
appinspect       egerl   fragmentron  genherd       nodefinder    zfile
combonodefinder  erlrc   fuserl       loggins       schemafinder
ec2nodefinder    erlsom  gencron      n54etsbugfix  virtuerl
```

Filename = Application Name

Files are Empty (for Now)

Actual Applications Directory

```
% ls /etc/erlrc.d/applications
appinspect        egerl   fragmentron  genherd        nodefinder    zfile
combonodefinder   erlrc   fuserl       loggins        schemafinder
ec2nodefinder     erlsom  gencron      n54etsbugfix   virtuerl
```

Filename = Application Name

Files are Empty (for Now)

Start Up Order

Actual Applications Directory

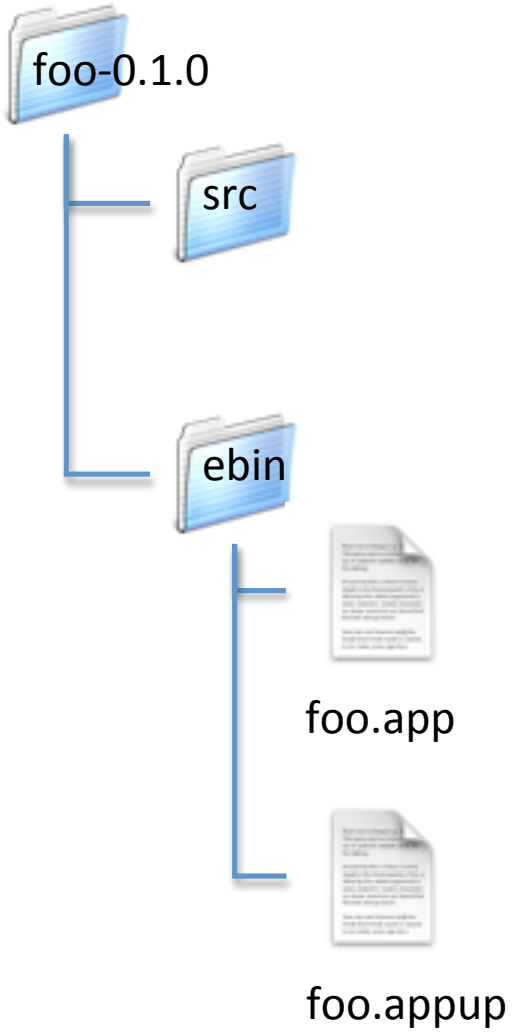
```
% ls /etc/erlrc.d/applications
appinspect        egerl   fragmentron  genherd        nodefinder    zfile
combonodefinder   erlrc   fuserl       loggins        schemafinder
ec2nodefinder     erlsom  gencron      n54etsbugfix   virtuerl
```

Filename = Application Name

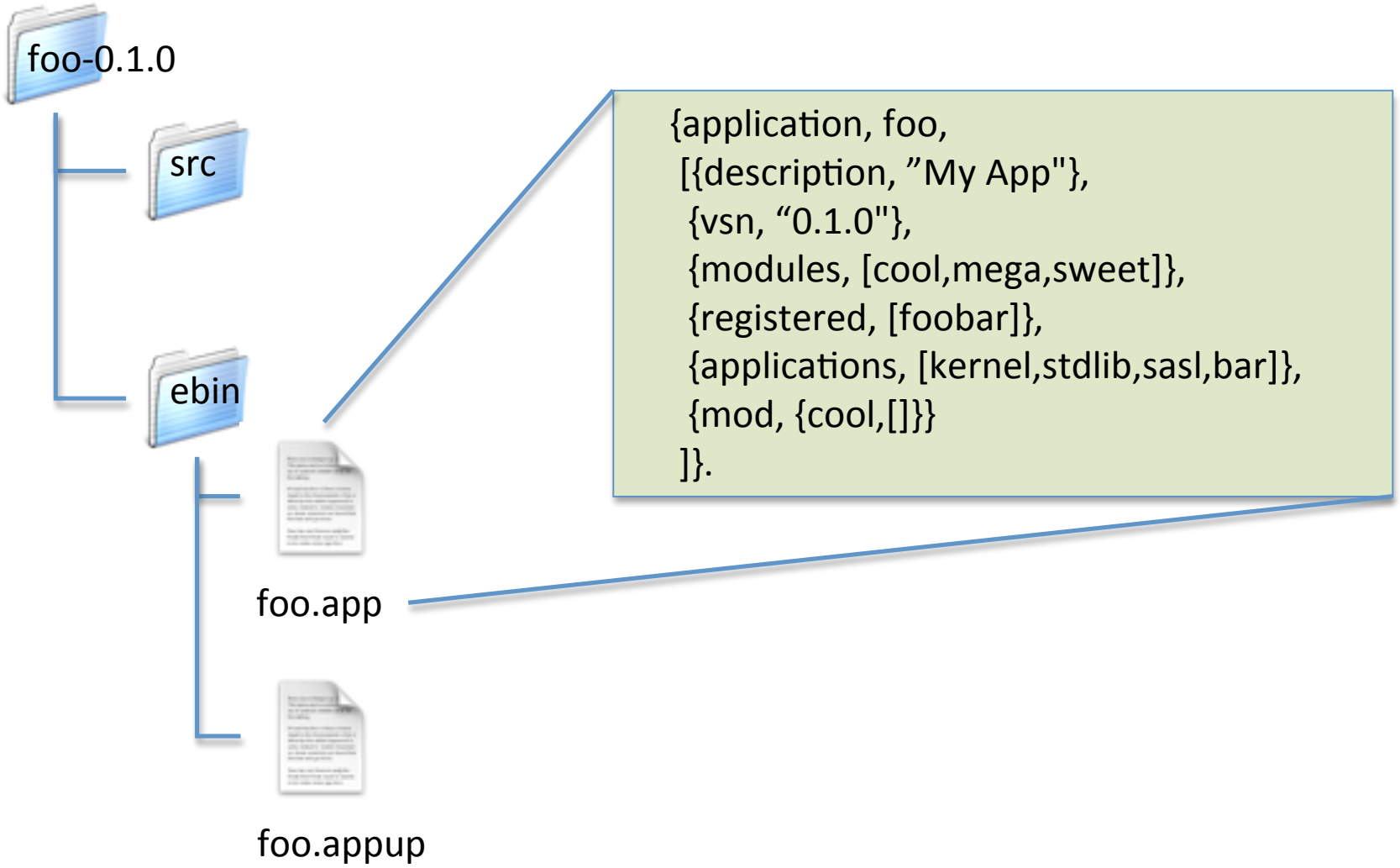
Files are Empty (for Now)

Start Up Order?

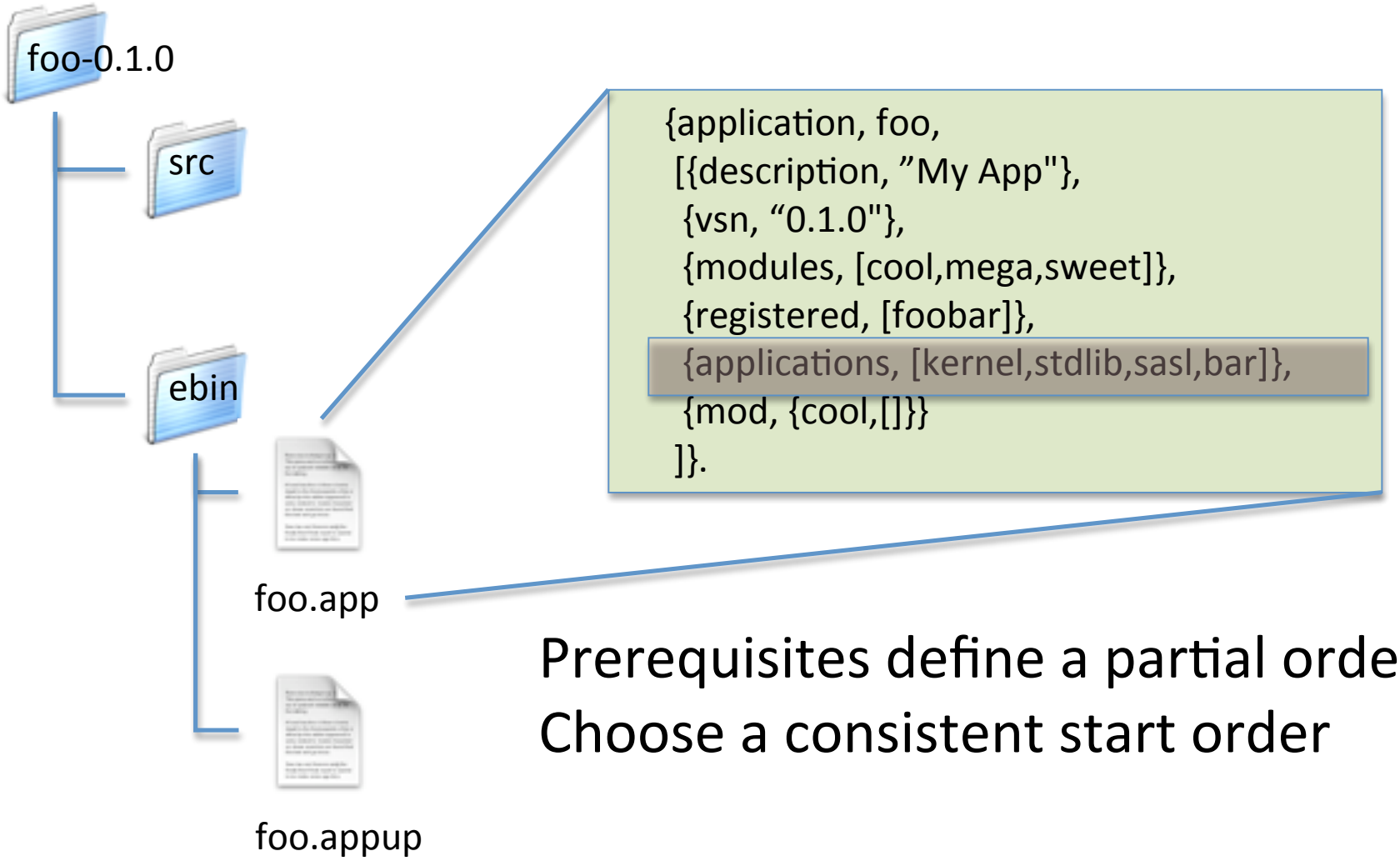
Determining the Startup Order



Determining the Startup Order



Determining the Startup Order



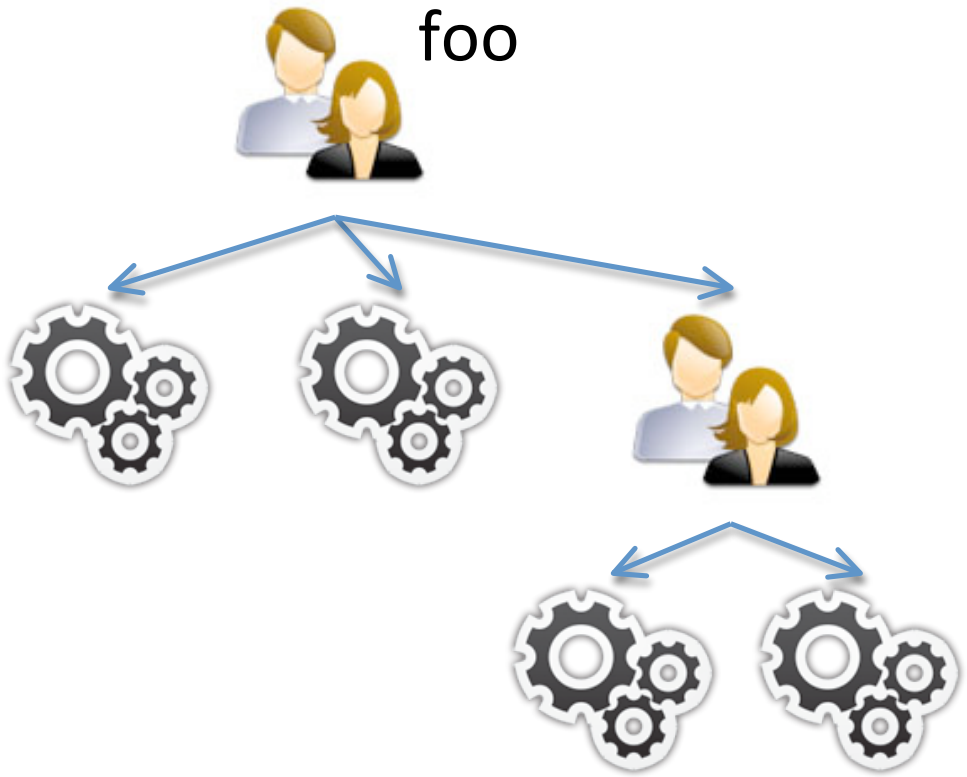
Prerequisites define a partial order

Choose a consistent start order

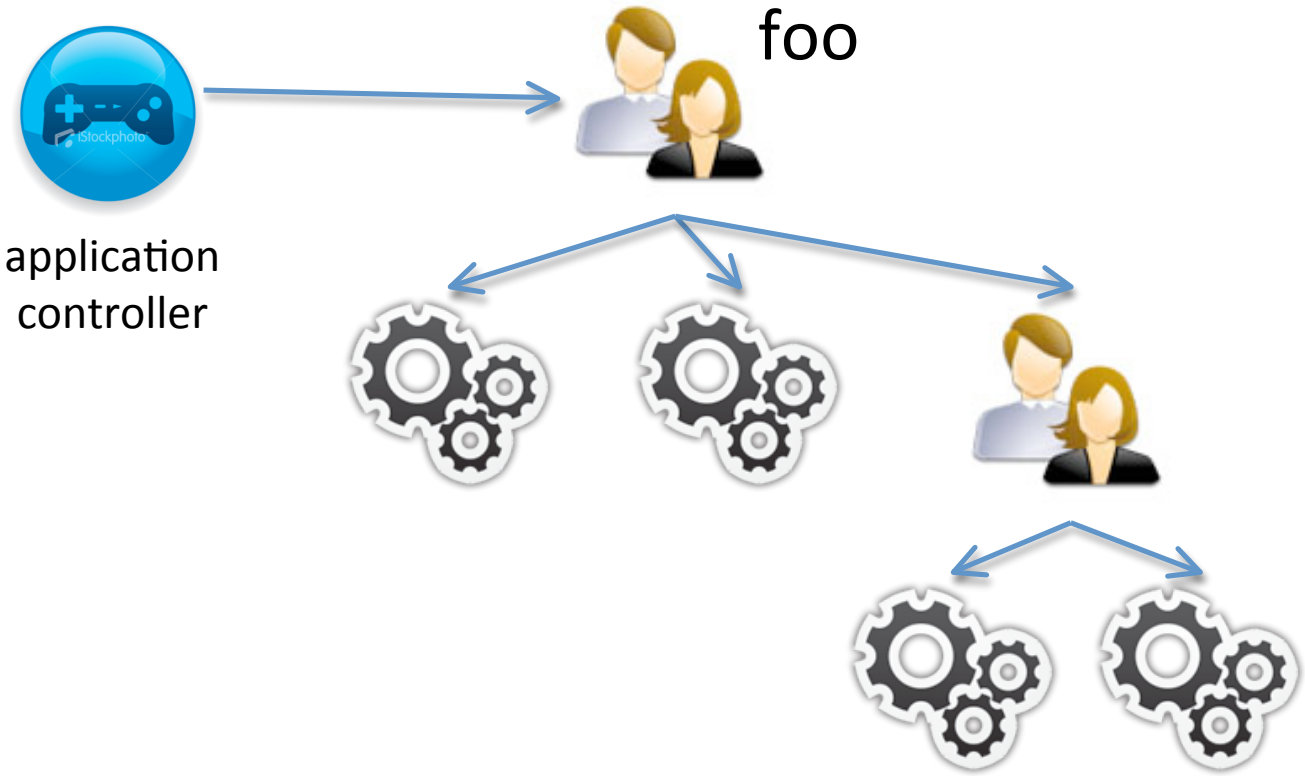
Regular Application

foo

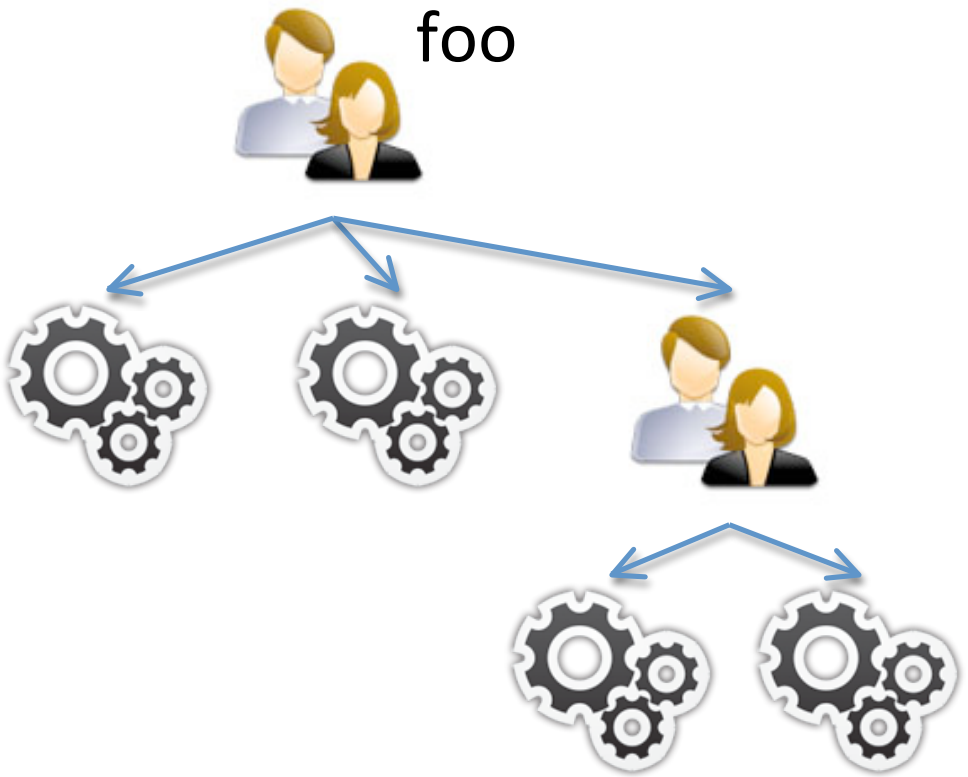
Regular Application



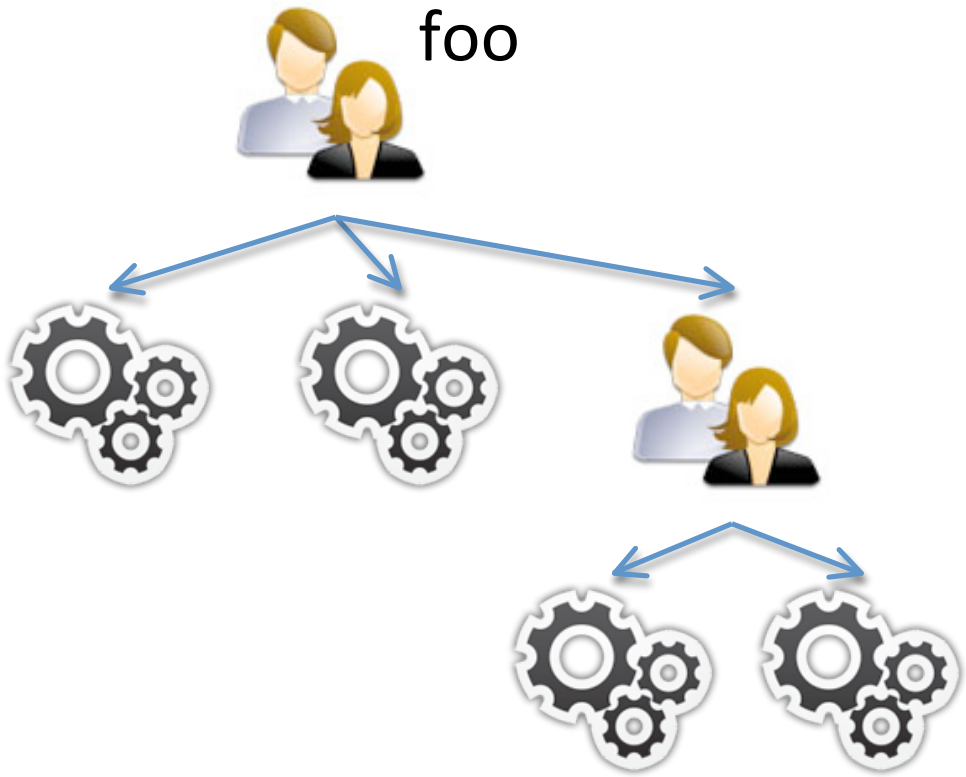
Regular Application



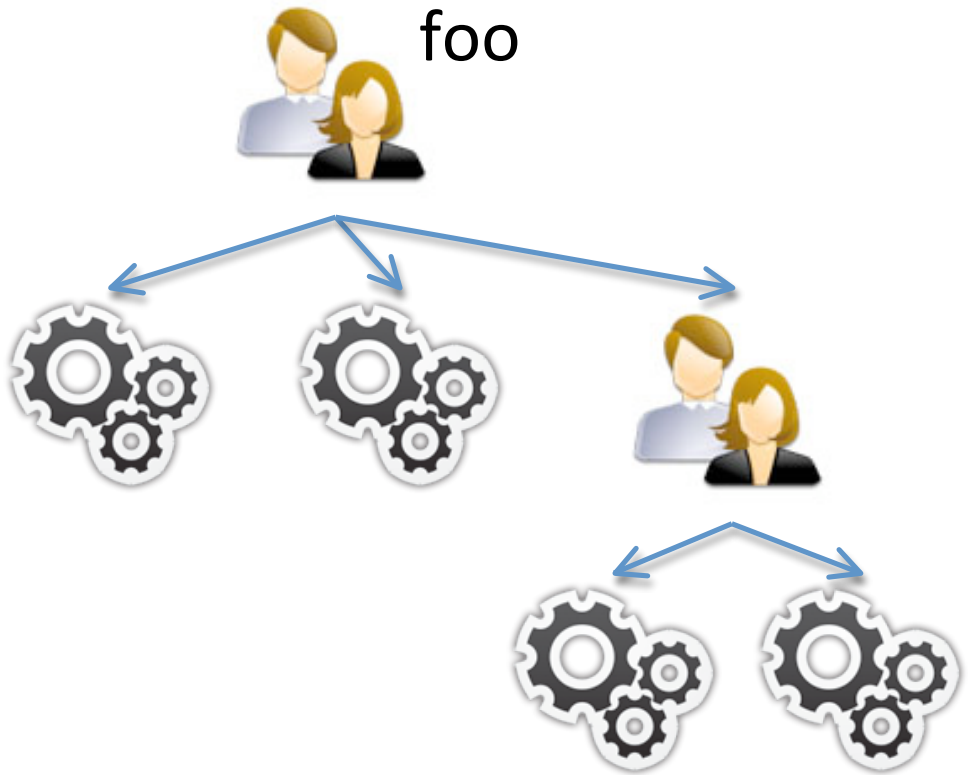
Included Application



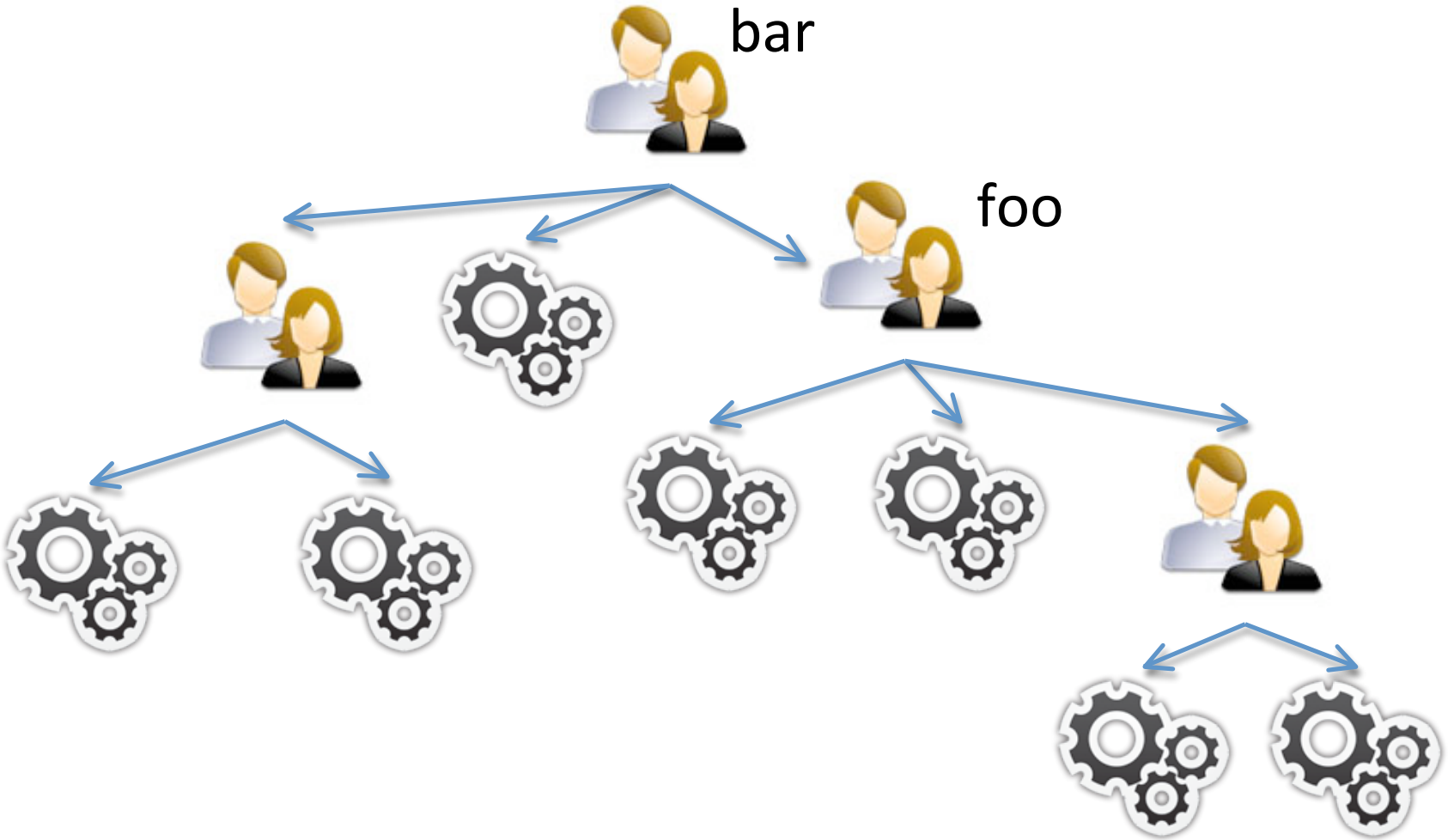
Included Application



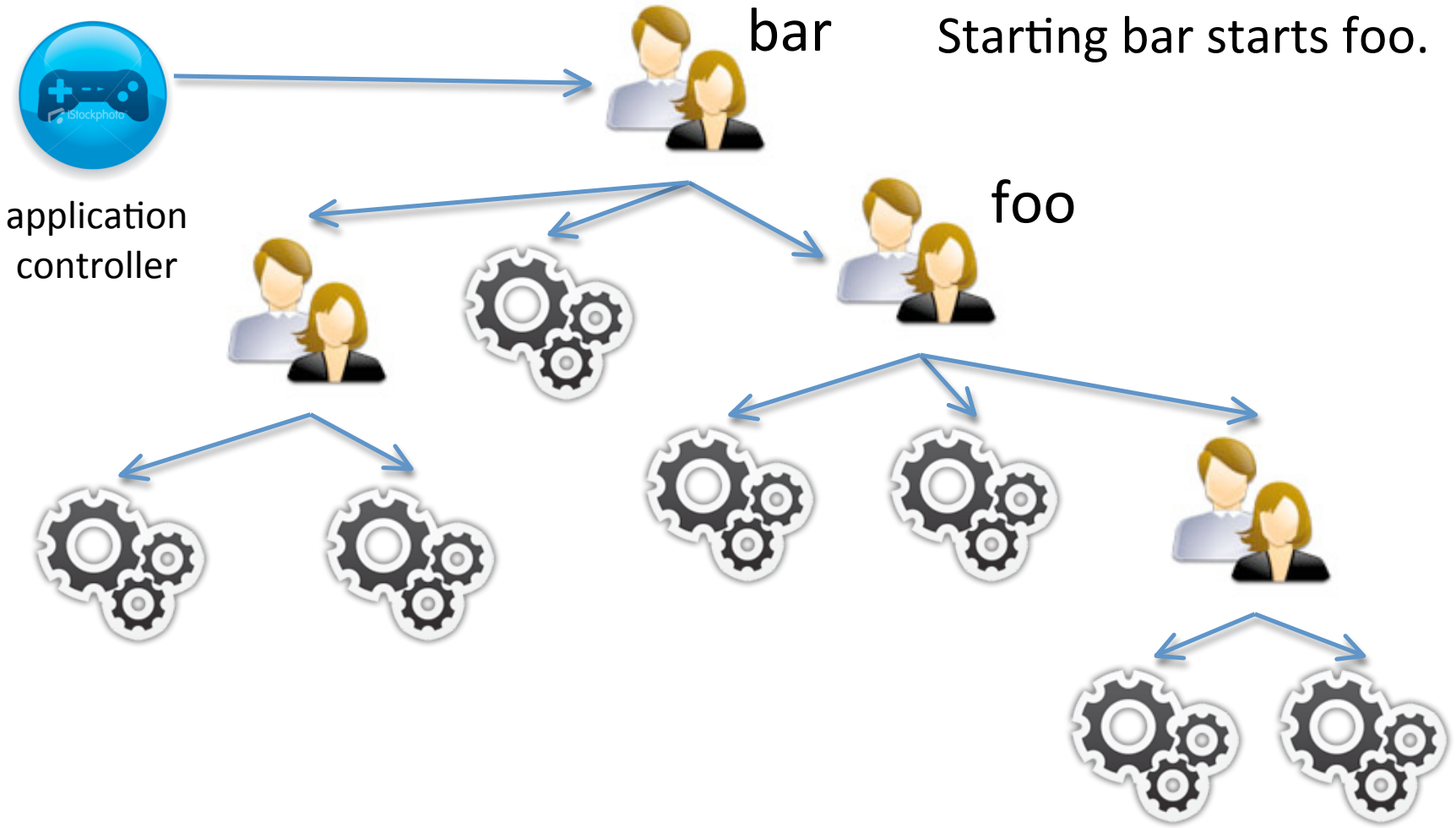
Included Application



Included Application



Included Application



Included Application Logic

- If bar includes foo, and
- both bar and foo are to be started, then
- only start bar.
- If baz depends upon foo, and
- bar is to be started, then
- baz depends upon bar.

Boot Procedure

- Packages contain additional empty file.

  /etc/erlrc/applications/<appname>
- .app prerequisites are correct.
- Start using any boot script, e.g., start_sasl.
- Pass control to `erlrc_boot:boot/0`.

  erl –boot start_sasl –s erlrc_boot boot

Problems to solve

- ✔ Static: booting the VM properly
- Dynamic: hot install/upgrade/remove

Erlrc upgrade components

- `erlrcdynamic` module methods.
  - `erlrcdynamic:downgrade/3`
- shell scripts
  - erlrc-downgrade
- rendezvous at /etc/erlrc.d/nodes

```
% ls /etc/erlrc.d/nodes
cb8eec1a1b85ec017517d3e51c5aee7b
```

Package hook correspondence

Shell script	`erlrcdynamic` method	Description	When (debian)
erlrc-start	`start/2`	Idempotent start	postinst configure ; postinst abort-remove
erlrc-stop	`stop/2`	Idempotent stop	prerm remove
erlrc-upgrade	`upgrade/3`	Upgrade	postinst upgrade
erlrc-downgrade	`downgrade/3`	Downgrade application	postinst abort_upgrade

Example: prerm

```
#! /bin/sh
package_name="example"
package_version="0.0.0”
operation="$1”
case "$operation" in
remove)
  which erlrc-stop >/dev/null 2>/dev/null
  test $? -ne 0 || erlrc-stop "${package_name}" "${package_version}" || exit 1
;;
*)
;;
esac
exit 0
```

Example: prerm

```
#! /bin/sh
package_name="example"
package_version="0.0.0”
operation="$1”
case "$operation" in
remove)
  which erlrc-stop >/dev/null 2>/dev/null
  test $? -ne 0 || erlrc-stop "${package_name}" "${package_version}" || exit 1
;;
*)
;;
esac
exit 0
```

Best practices for build system

- Automatically generate erlrc_boot file.
  - `/etc/erlrc.d/applications/<appname>`
- Generate both dependencies from single specification.
  - .app prerequisites
  - OS package dependencies
- Automatically generate package hooks.

Upgrade Logic

- If there is an appup file in the newer application version, use it; otherwise, generate one automatically.
- Ensure all modules listed in the current application specification are loaded.
- Find any added and removed included applications by examining the current and target version OTP app files.
- Stop any added included applications.
- Execute `release_handler:eval_appup_script/4`.
- Start any removed included applications, if they are listed in $ERLRC_ROOT/applications.

Upgrade Logic

- If there is an appup file in the newer application version, use it; otherwise, generate one automatically.
- Ensure all modules listed in the current application specification are loaded.
- Find any added and removed included applications by examining the current and target version OTP app files.
- Stop any added included applications.
- Execute `release_handler:eval_appup_script/4`.
- Start any removed included applications, if they are listed in /etc/erlrc.d/applications.

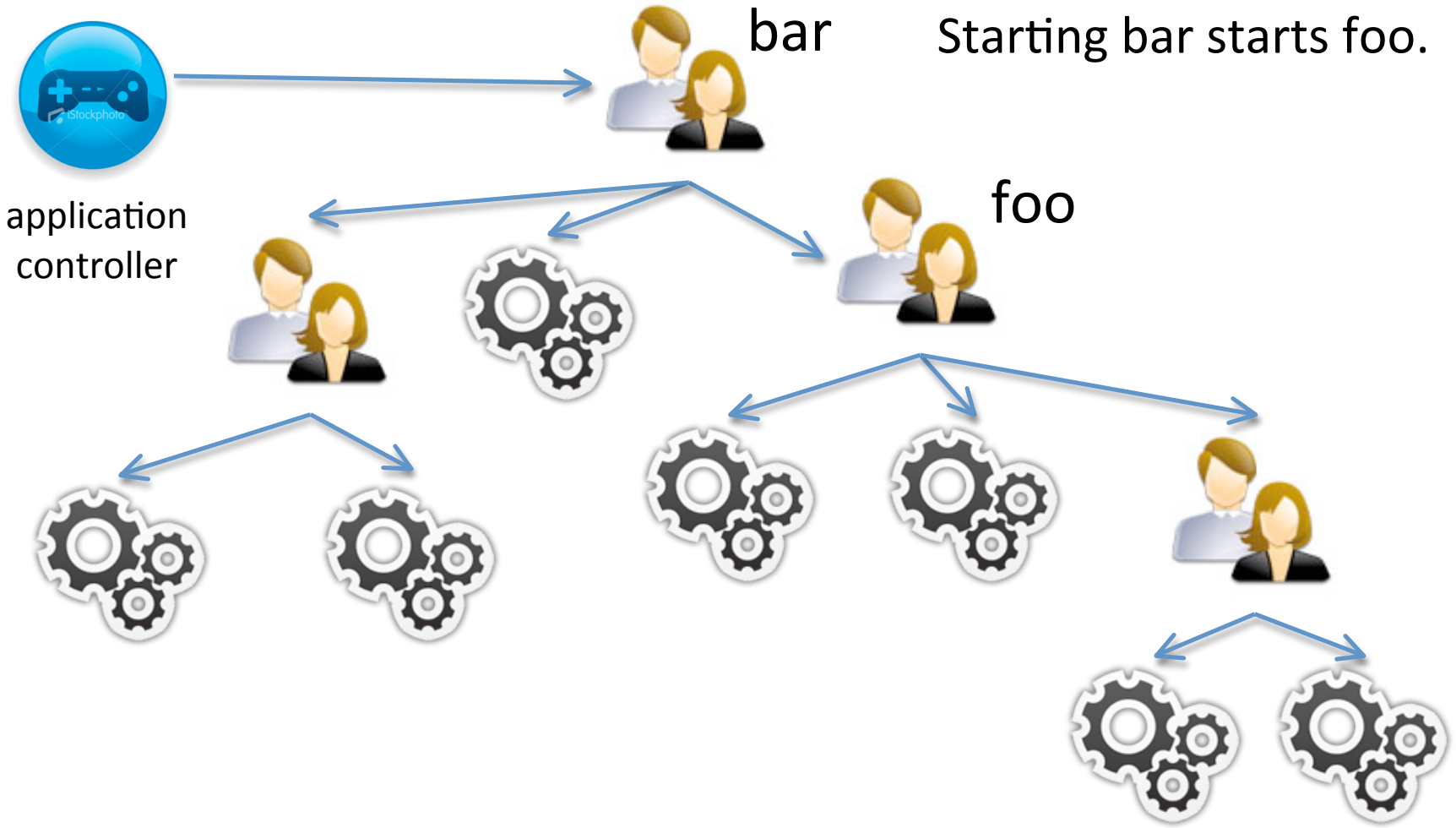
Upgrade Logic

- If there is an appup file in the newer application version, use it; otherwise, generate one automatically.
- Ensure all modules listed in the current application specification are loaded.
- Find any added and removed included applications by examining the current and target version OTP app files.
- Stop any added included applications.
- Execute `release_handler:eval_appup_script/4`.
- Start any removed included applications, if they are listed in /etc/erlrc.d/applications.

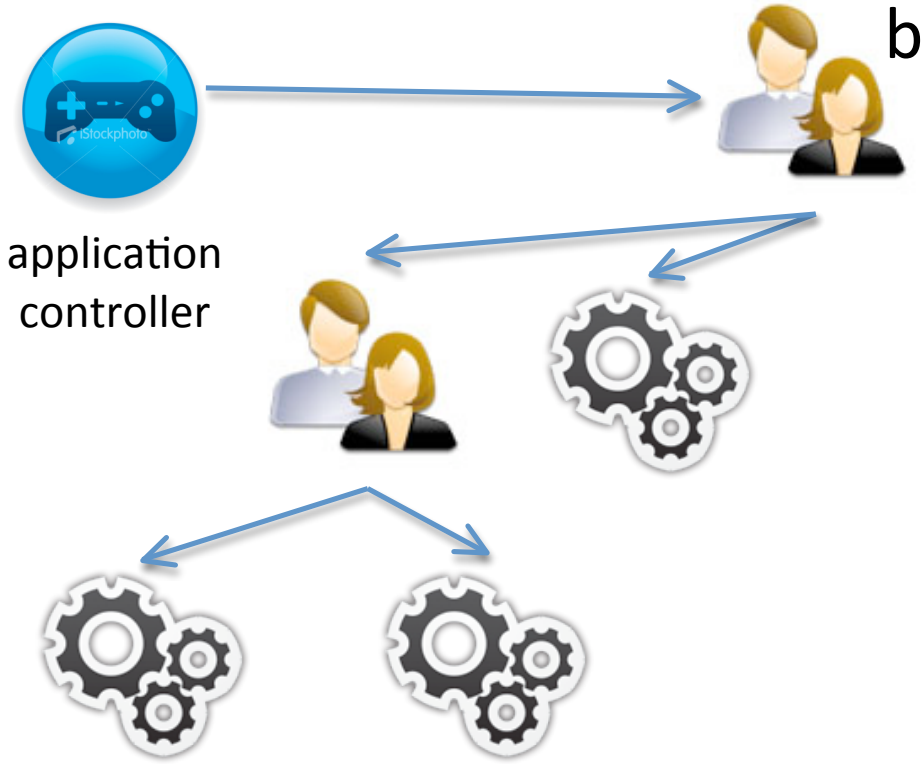
Upgrade Logic

- If there is an appup file in the newer application version, use it; otherwise, generate one automatically.
- Ensure all modules listed in the current application specification are loaded.
- Find any added and removed included applications by examining the current and target version OTP app files.
- Stop any added included applications.
- Execute `release_handler:eval_appup_script/4`.
- Start any removed included applications, if they are listed in /etc/erlrc.d/applications.

Included Application



Included Application



does not start

Starting bar ~~starts~~ foo.

Included Application Invariants

- If an application is listed in the `/etc/erlrc.d/applications` directory, then it should be running.
- When an application is running, all of its included applications are considered running.
- An application cannot both run itself and be included.

Included Application Logic

- If X is included by Y, then before starting Y, X is stopped.
- If X is included by Y, then after stopping Y, X is started.
- If Y is upgraded or downgraded such that X becomes included when it was not previously, before Y is upgraded or downgraded, X is stopped.
- If Y is upgraded or downgraded such that X becomes no longer included when it was previously, after Y is upgraded or downgraded, X is started.

Included Application Logic

- If X is included by Y, then before starting Y, X is stopped.
- If X is included by Y, then after stopping Y, X is started.
- If Y is upgraded or downgraded such that X becomes included when it was not previously, before Y is upgraded or downgraded, X is stopped.
- If Y is upgraded or downgraded such that X becomes no longer included when it was previously, after Y is upgraded or downgraded, X is started.

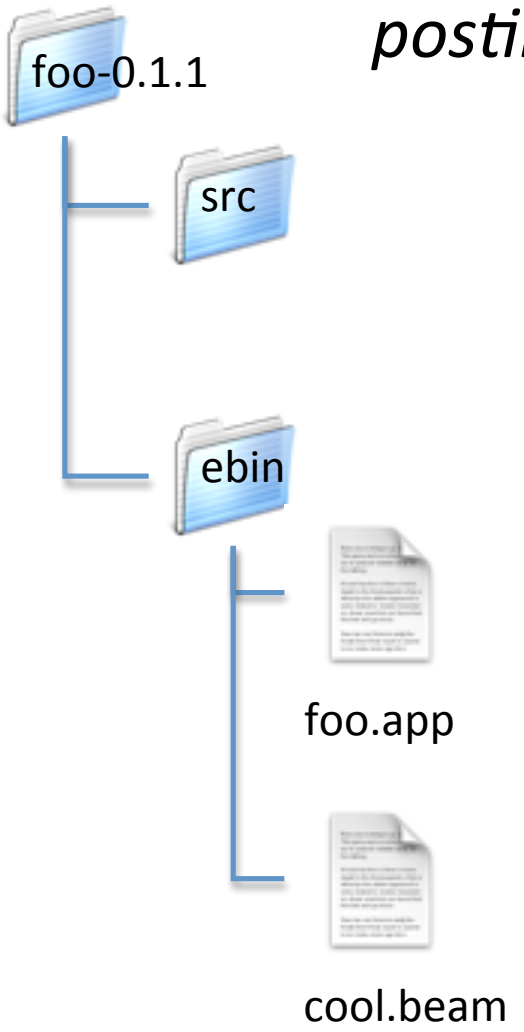
.appup files

- Manual generation: tedious, error-prone.
- Typically can be automatically generated.
- Strategy: automation with override.
  - Use .appup file if provided (rare).
  - Otherwise automatically generate (common).
- Key tool: `beam_lib:chunks/2`.

Automatic .appup generation

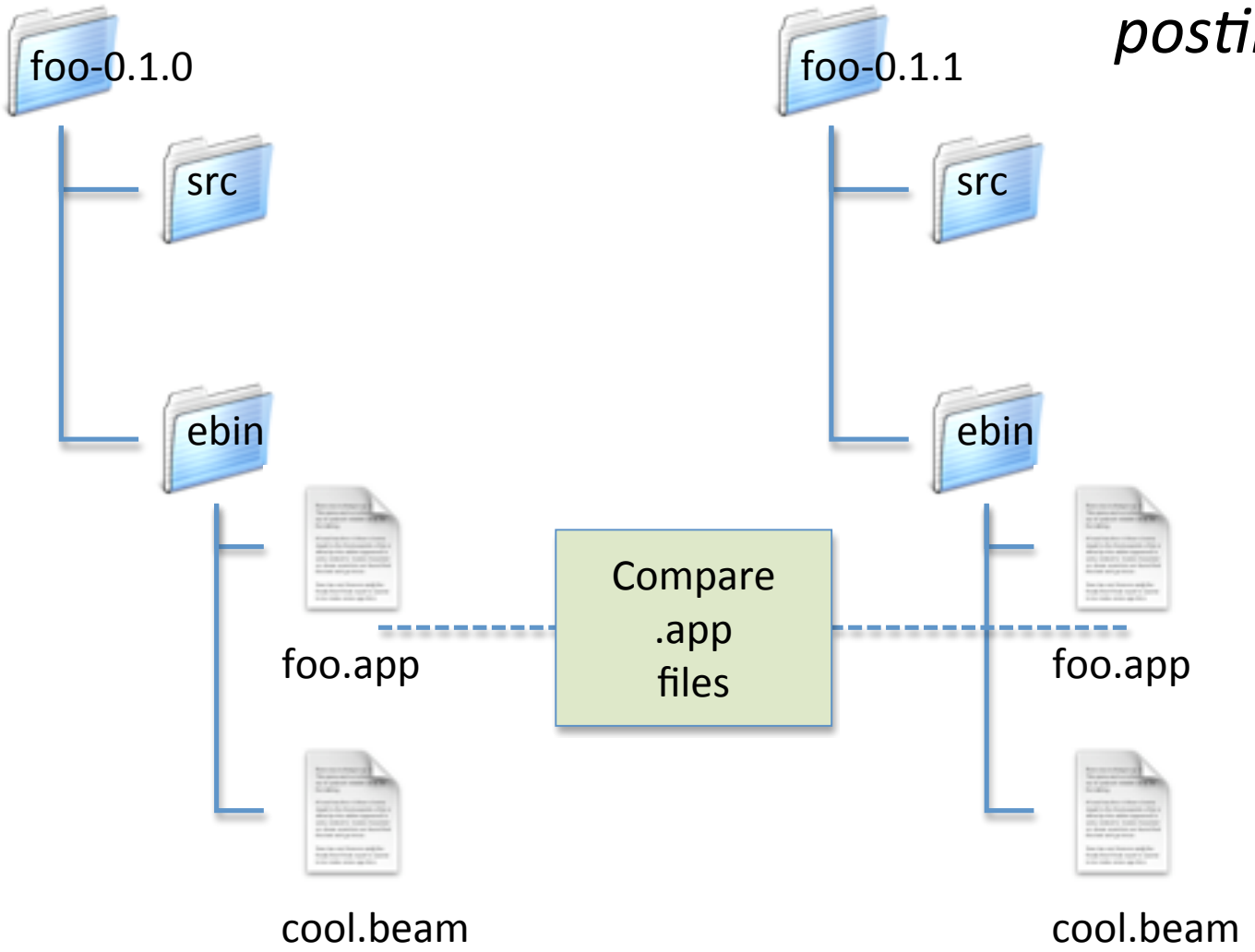
Automatic .appup generation

postinst upgrade

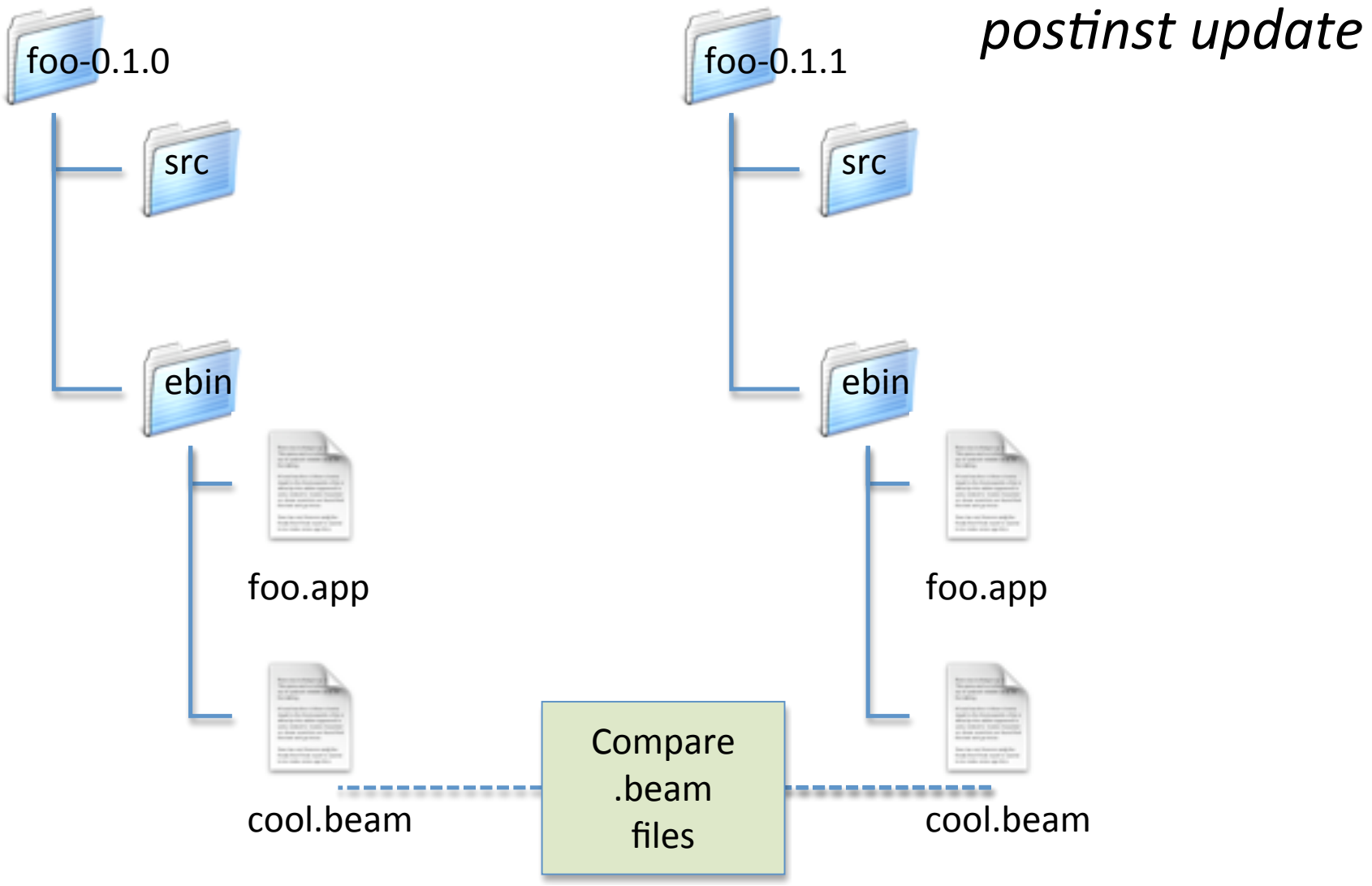


Automatic .appup generation

postinst upgrade



Automatic .appup generation



Example: load_module

```
{application, foo,
 [{description, ”My App"},
  {vsn, “0.1.0"},
  {modules, [cool,mega,sweet]},
  {registered, [foobar]},
  {applications, [kernel,stdlib,sasl,bar]},
  {mod, {cool,[]}}
 ]}.
```

```
{application, foo,
 [{description, ”My App"},
  {vsn, “0.1.1"},
  {modules, [cool,mega,sweet,ultra]},
  {registered, [foobar]},
  {applications, [kernel,stdlib,sasl,bar]},
  {mod, {cool,[]}}
 ]}.
```

Example: load_module

```
{application, foo,
 [{description, ”My App"},
  {vsn, “0.1.0"},
  {modules, [cool,mega,sweet]},
  {registered, [foobar]},
  {applications, [kernel,stdlib,sasl,bar]},
  {mod, {cool,[]}}
 ]}.
```

```
{application, foo,
 [{description, ”My App"},
  {vsn, “0.1.1"},
  {modules, [cool,mega,sweet,ultra]},
  {registered, [foobar]},
  {applications, [kernel,stdlib,sasl,bar]},
  {mod, {cool,[]}}
 ]}.
```

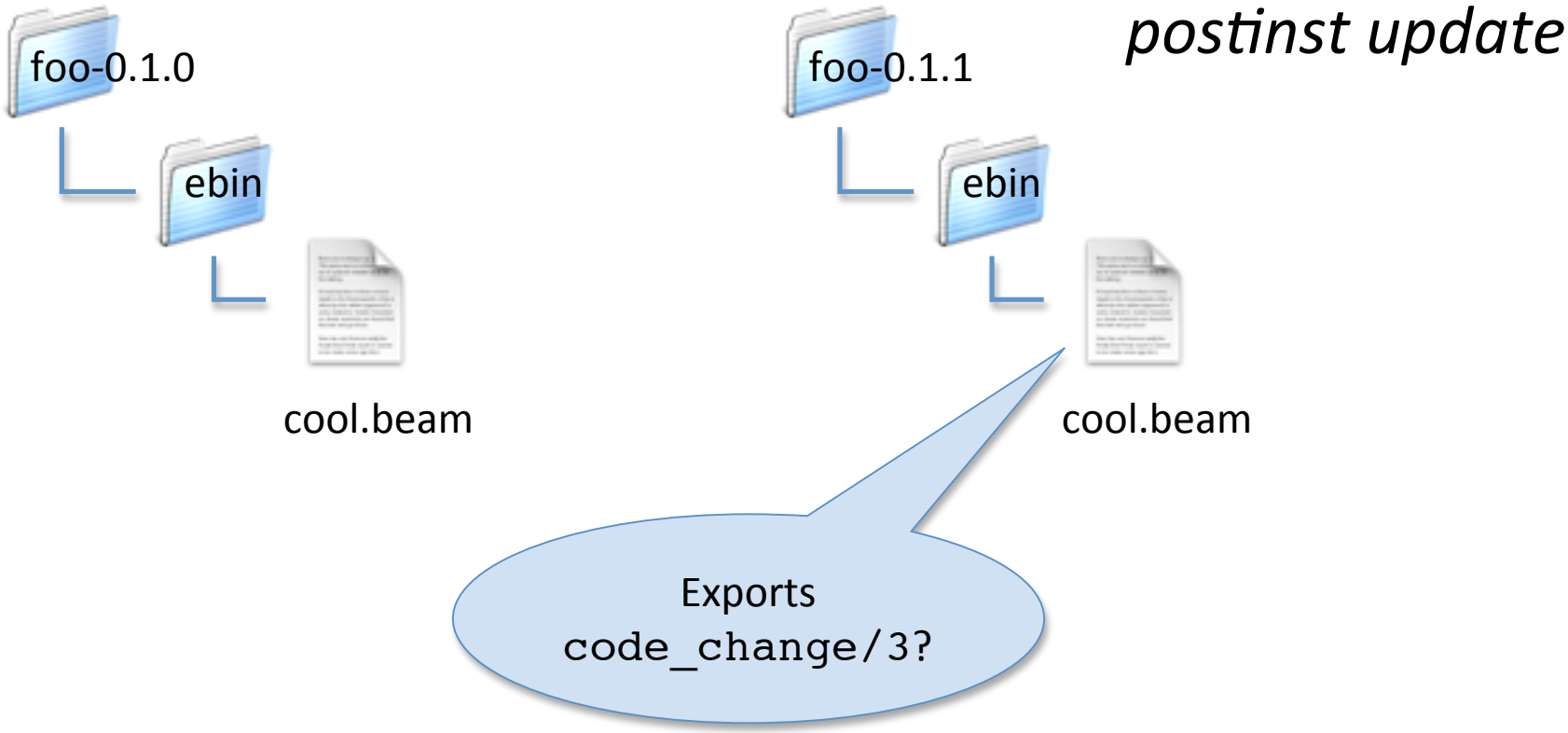
ultra is a new module

emit `load_module` directive

Example: update

- `update` directive is used to upgrade the code and the state of an active process, e.g., gen_server.
- All of the gen_XXX family export `code_change/3` in their behaviour.

Example: update



If yes, issue `update` directive.

.appup file generation

- Compute added and removed modules by looking at the old and new OTP app files.
- Emit a load_module directive for every added module.
- For each module which is in both versions of the application:
  - If the module implements the supervisor behaviour
    - Emit an "upgrade for supervisors" instruction.
    - If the module exports a sup_upgrade_notify/2 function, emit an instruction to call it.
  - Else if the module exports a code_change/3 function, emit a update directive for that module.
  - Otherwise, emit a load_module directive for that module.
- If there is a start module for the application:
  - Determine if the new beam for the start module exports a version_change/2 function. If so, emit a directive to call it.
- Emit a delete_module directive for every removed module.

.appup file generation

- Compute added and removed modules by looking at the old and new OTP app files.
- Emit a load_module directive for every added module.
- For each module which is in both versions of the application:
  - If the module implements the supervisor behaviour
    - Emit an "upgrade for supervisors" instruction.
    - If the module exports a sup_upgrade_notify/2 function, emit an instruction to call it.
  - Else if the module exports a code_change/3 function, emit a update directive for that module.
  - Otherwise, emit a load_module directive for that module.
- If there is a start module for the application:
  - Determine if the new beam for the start module exports a version_change/2 function. If so, emit a directive to call it.
- Emit a delete_module directive for every removed module.

Custom .appup hooks

- `<supervisor>:sup_upgrade_notify/2`
  - Used to start or stop newly added or removed children.
- `<application>:version_change/2`
  - Used to notify the application start module that the application version has changed.

Distribution and provisioning

- ec2-do
  - escript for rolling window execution across an ec2 group.

```
% ec2-do -g production -m 3 apt-get install egerl
```

- apt-snapshot
  - provisioning tool based upon .deb packages

```
% apt-snapshot create <snapshot>
% apt-snapshot restore host:<snapshot>
```

An Erlang Startup

- Write awesome Erlang code.
- ???
- Deploy to EC2.
- ???
- Profit.

An Erlang Startup

- Write awesome Erlang code.
- Use erlrc and make OS packages.
- Deploy to EC2.
- ???
- Profit.

More Information

http://dukesoferl.blogspot.com/

Google

http://code.google.com/p/erlrc/

n54
Target.Engage.Convert

http://n54.com/