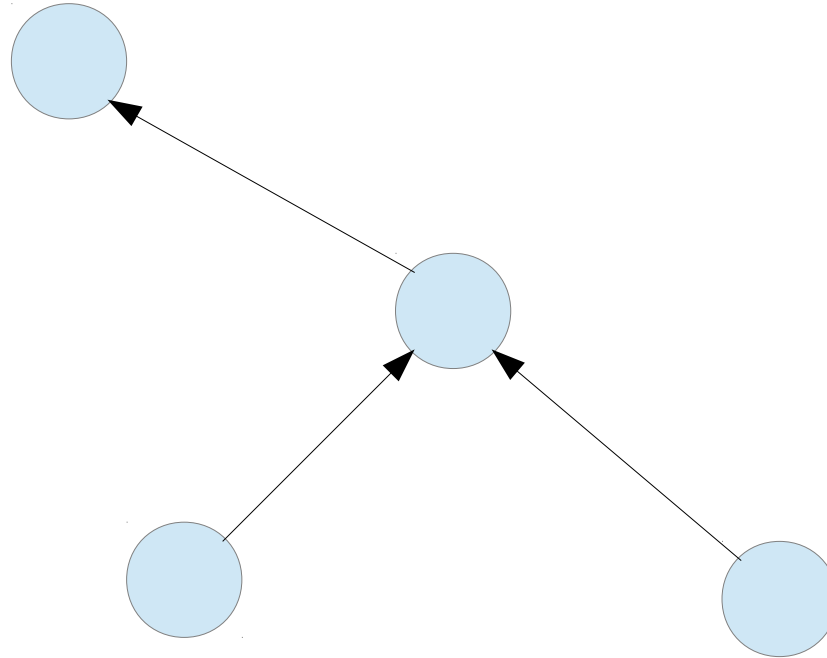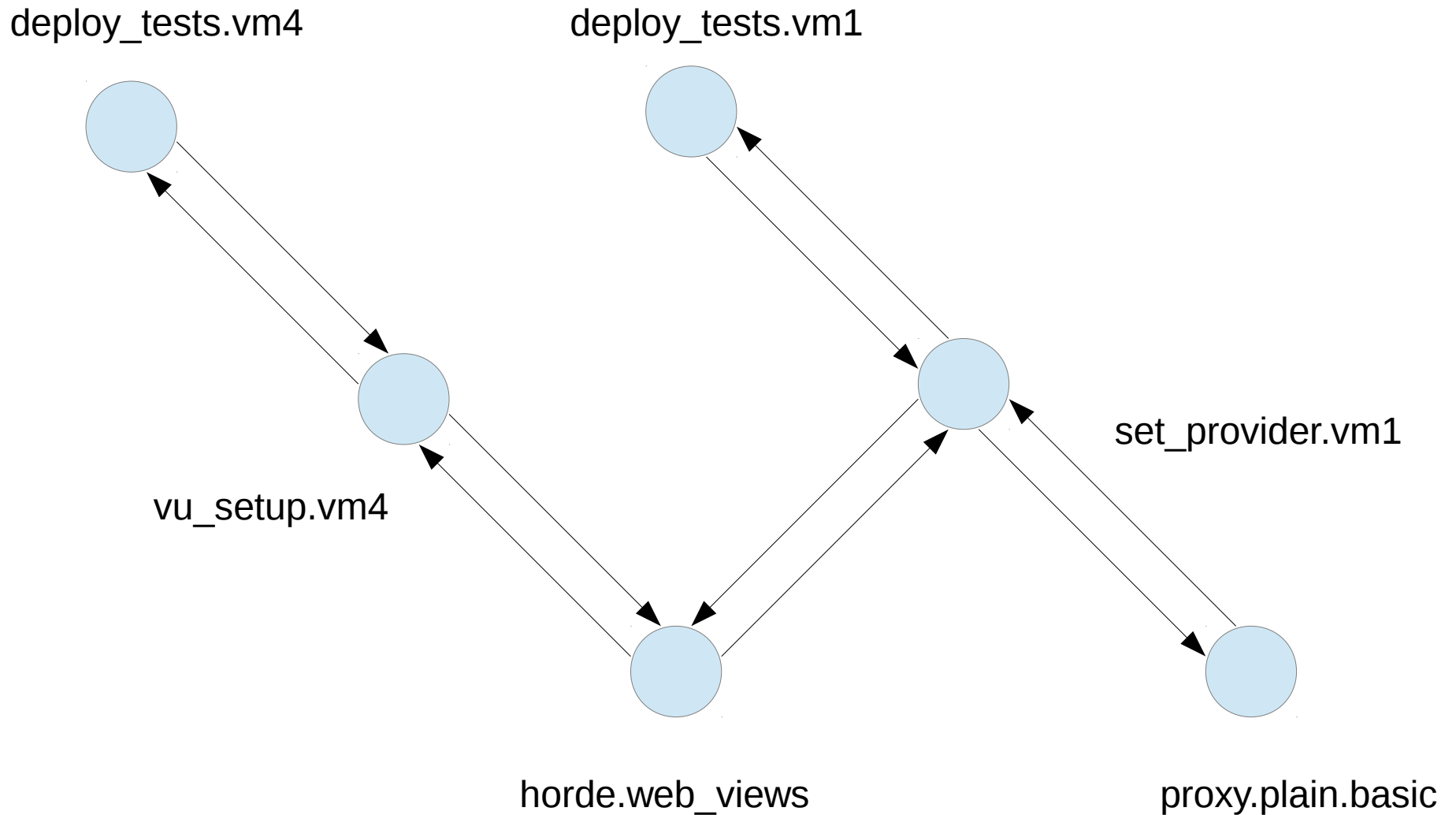# I2N-A Test traversal algorithm

Let's take a short look of how the algorithm used for automated setup and resolution of test dependencies works. It is the core algorithm of our test suite.

*** *Please keep in mind that in order to understand the algorithm you need to understand the graph structure behind it which is explained in detail in the document accompanying this one.*
*If you do so then I promise this will be a lot of fun.*

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

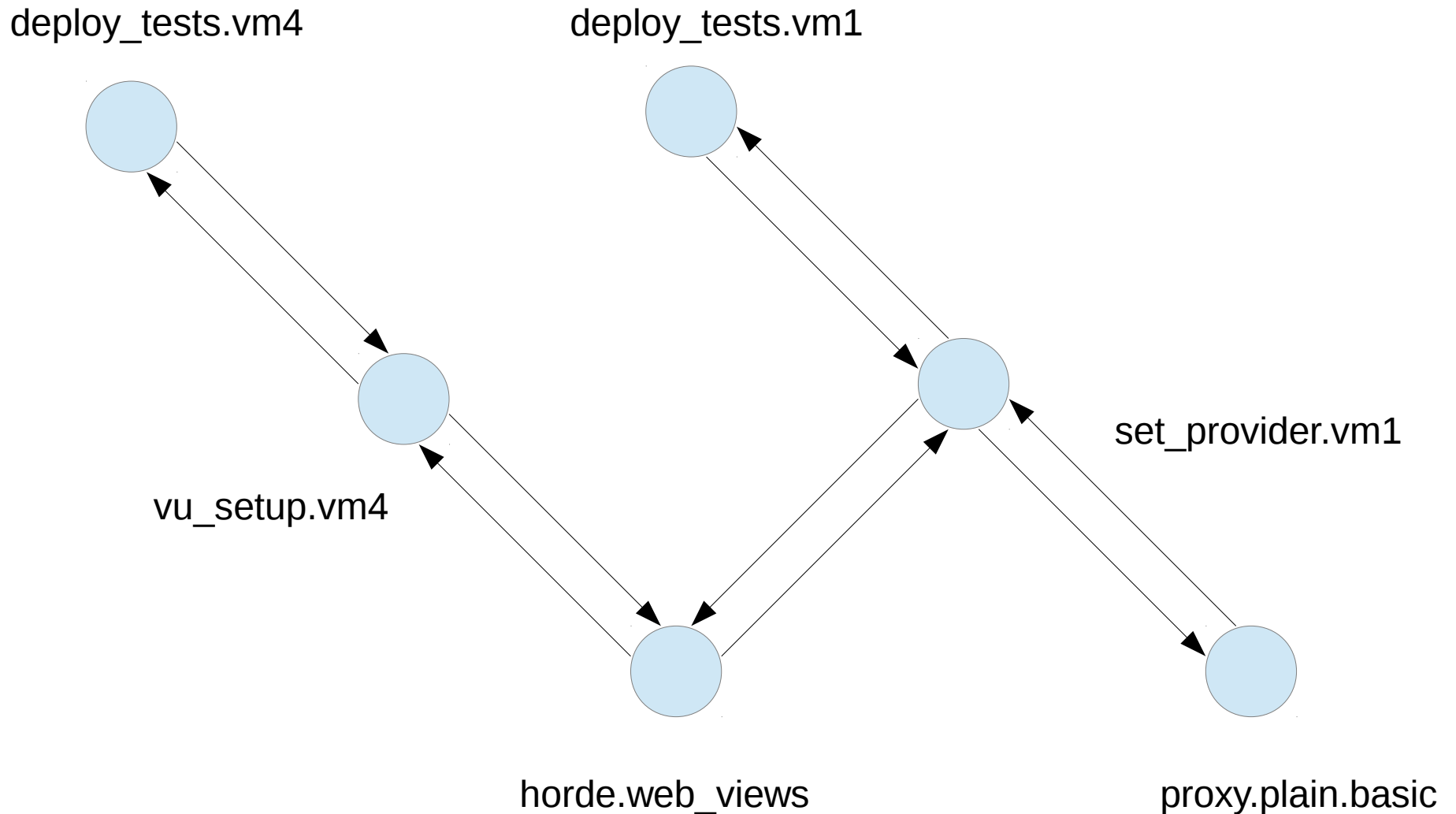set_provider.vm1

horde.web_views

proxy.plain.basic

Firstly, let's draw a circle for each test node.
Secondly, let's connect each test with an upward arrow to a test it depends on.
Lastly, let's connect each test with a downward arrow to a test depending on it.

Each test is then aware of the arrows that originate from it. No test is aware of the arrows that point to it.
This is important for the implementation of the algorithm where it is simply pointers to other nodes divided as
pointers to parents and pointers to children.

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

Here we take a few tests interconnected in this way giving them some example names. To differentiate among repeating tests but acting on different objects (vms), we will append ".vmX". The ones that don't have this appended are using multiple vms, e.g. *horde.web_views* uses *vm1* as well as *vm4* which can be seen by the dependencies.

Let's start from *deploy_tests.vm1* as our root test node...
//to simplify the default start from an actual root node which is analogical (in the last slides)

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
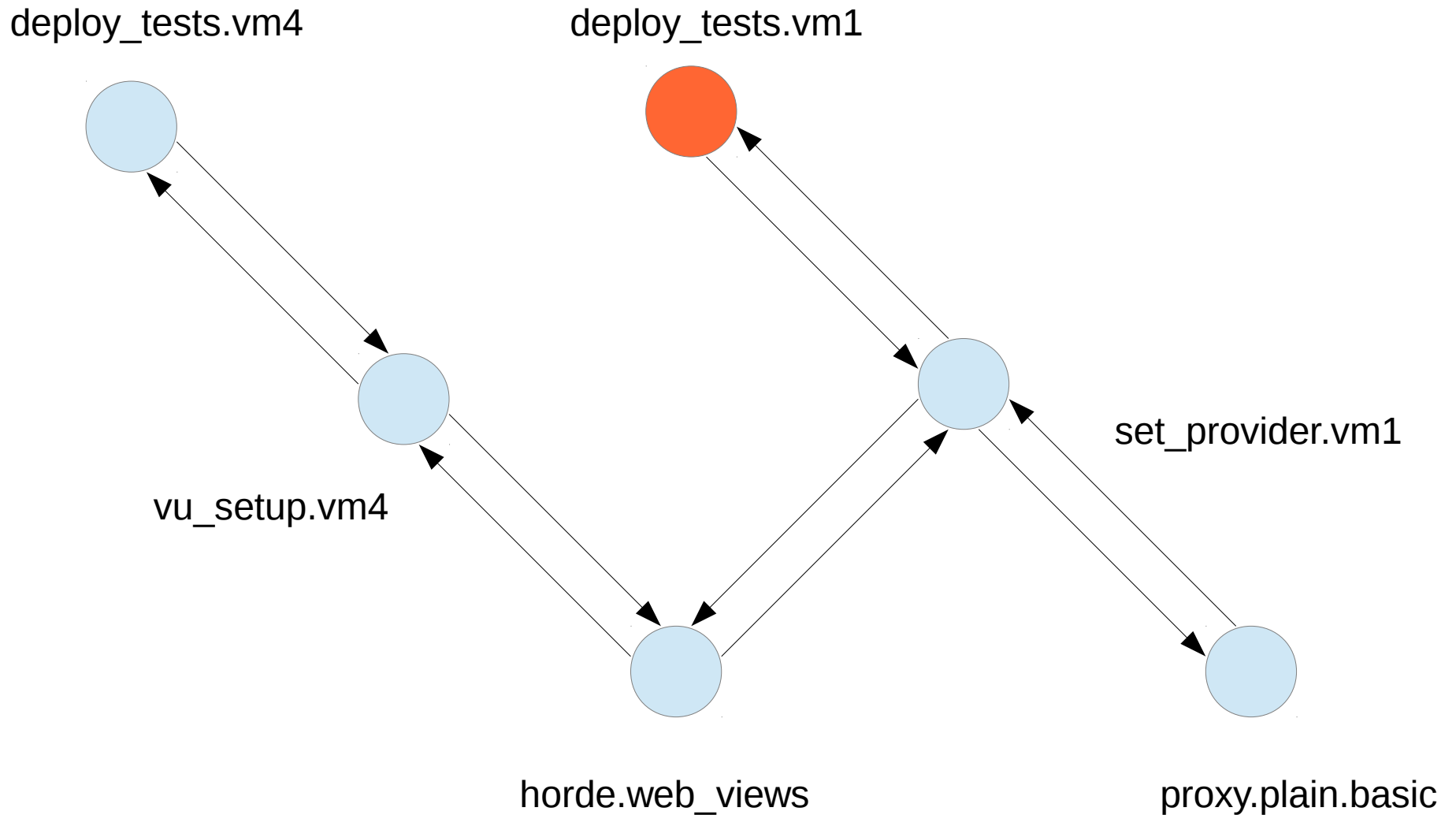
There are three qualities that affect our decision about deploy_tests.vm1:

- I**s it setup ready** = is the number of parent connections equal to 0
- **Is it cleanup ready** = is number of child connections equal to 0
- **Is it reusable** = does it create end states that can be reused so that we don't need to run it each time and if so was it run at least once so that the states are already available
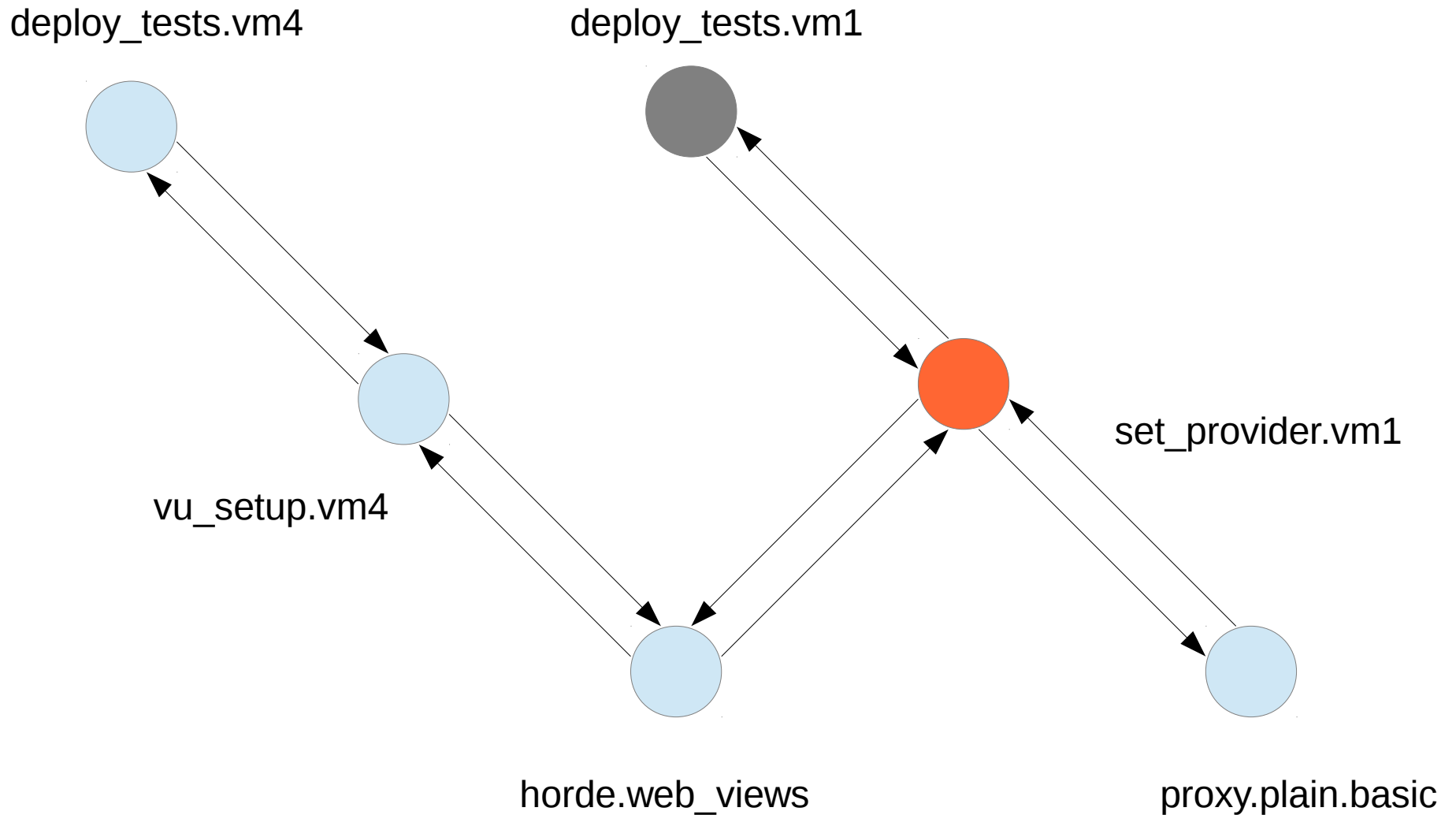
deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic
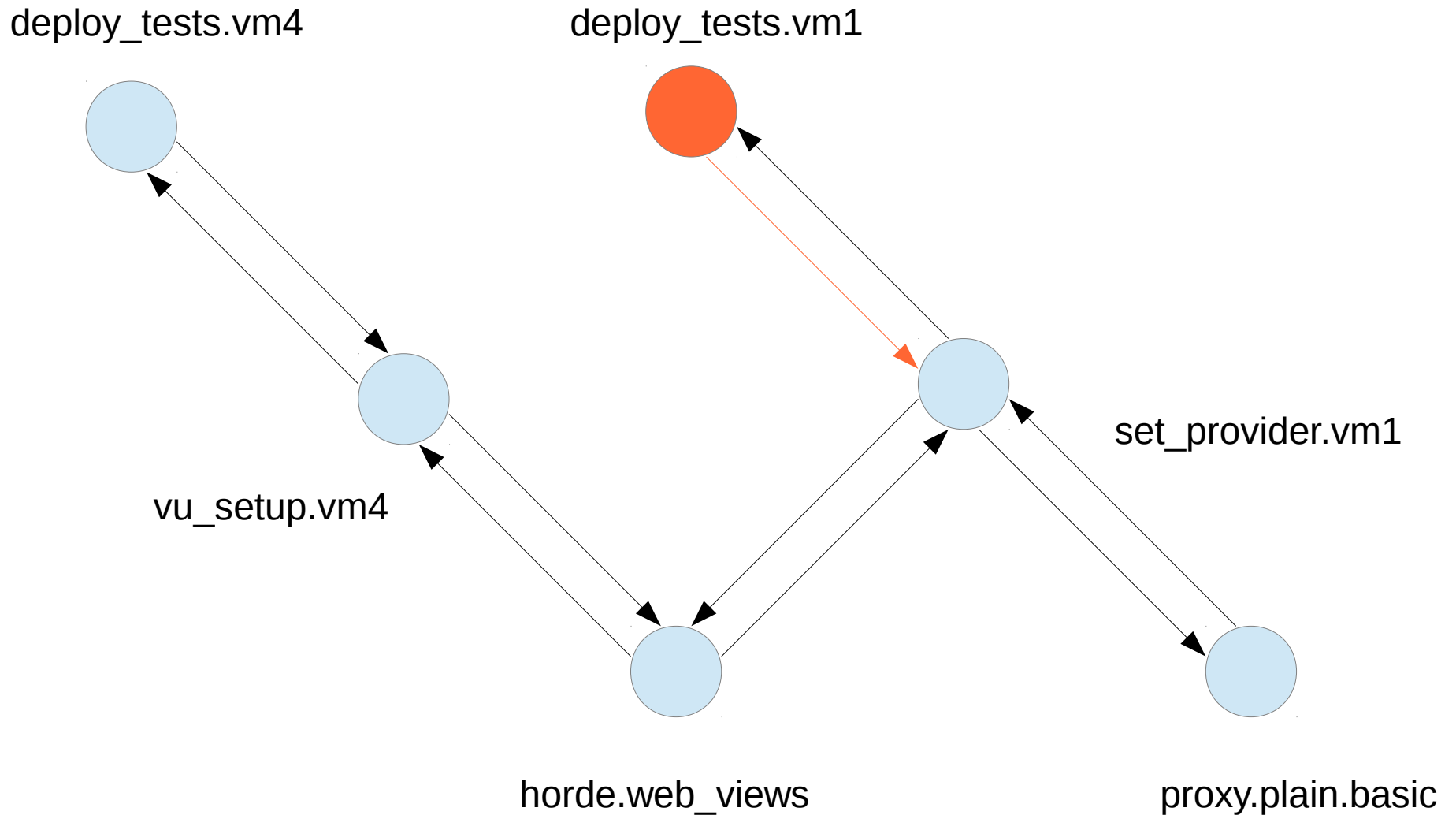
The conclusions are:

- I**s it setup ready** – True (always true for root state since it doesn't need setup by definition)
- **Is it cleanup ready** – False (one connection pending so no cleanup yet)
- **Is it reusable** – that depends on scanning for availability after previous runs, let's say it was run (True)
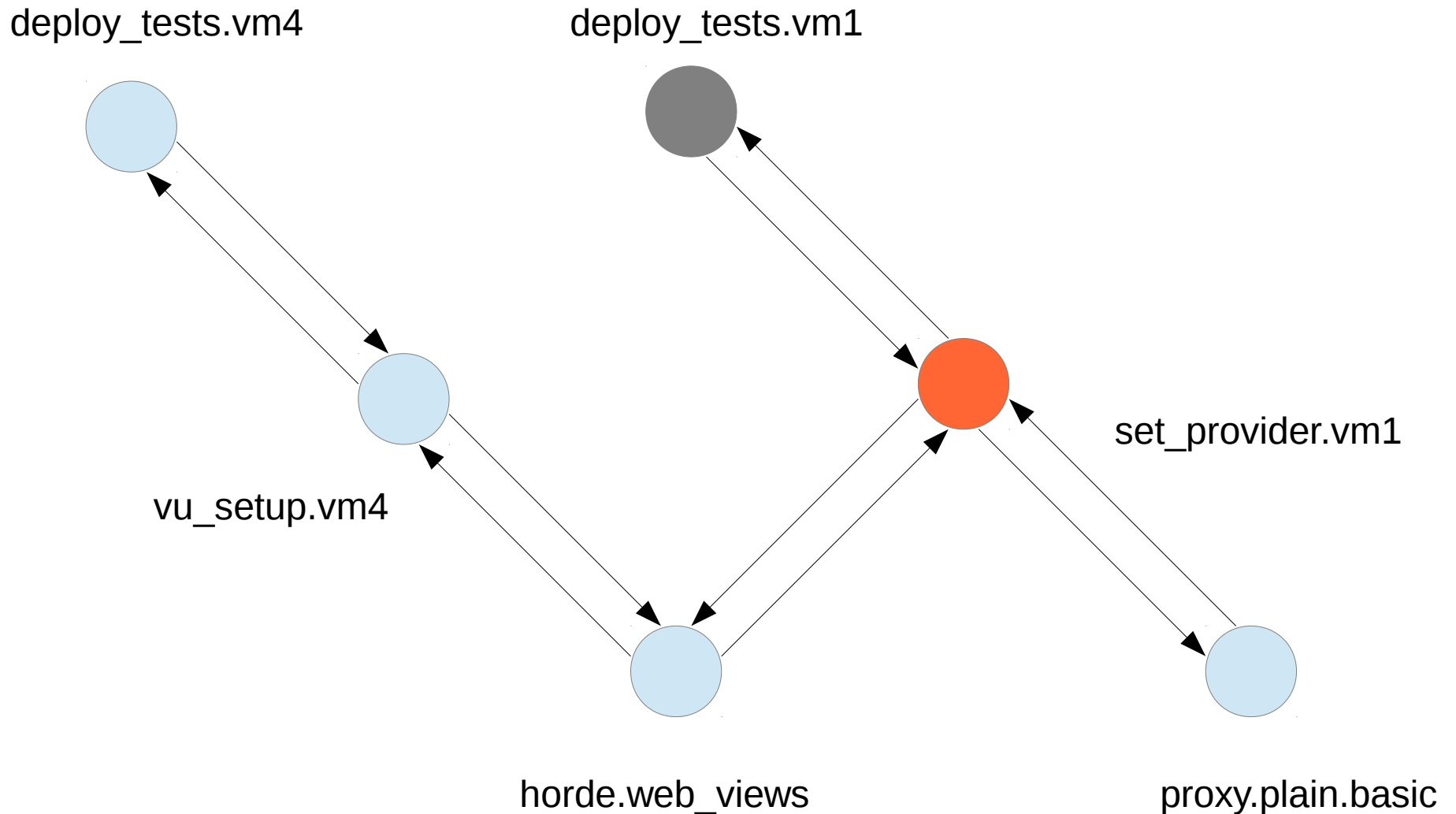
deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic
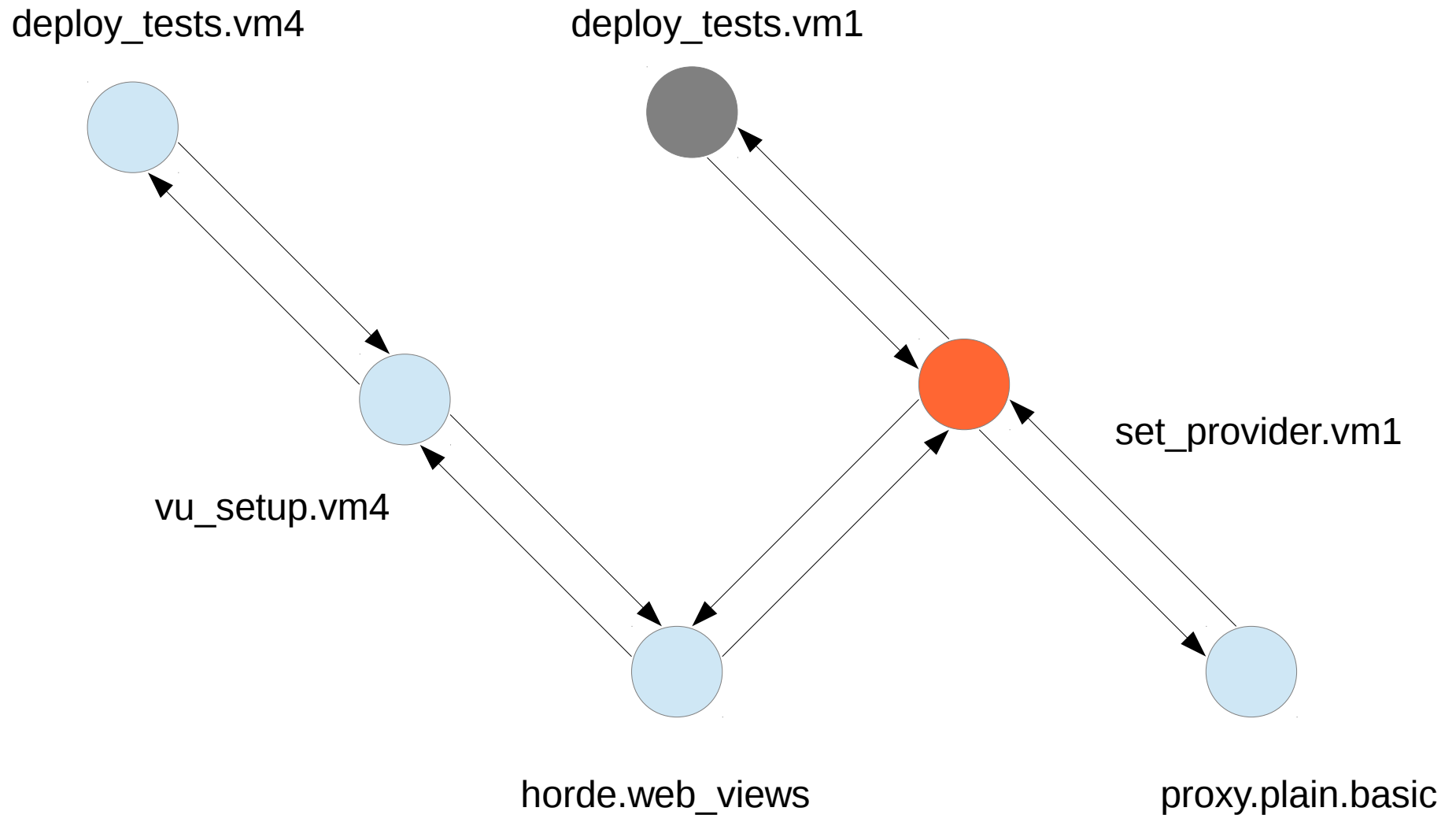
Moving to the next test node consists of two steps:

1) Push the node that is left behind to a stack of all visited nodes that are not cleanup ready
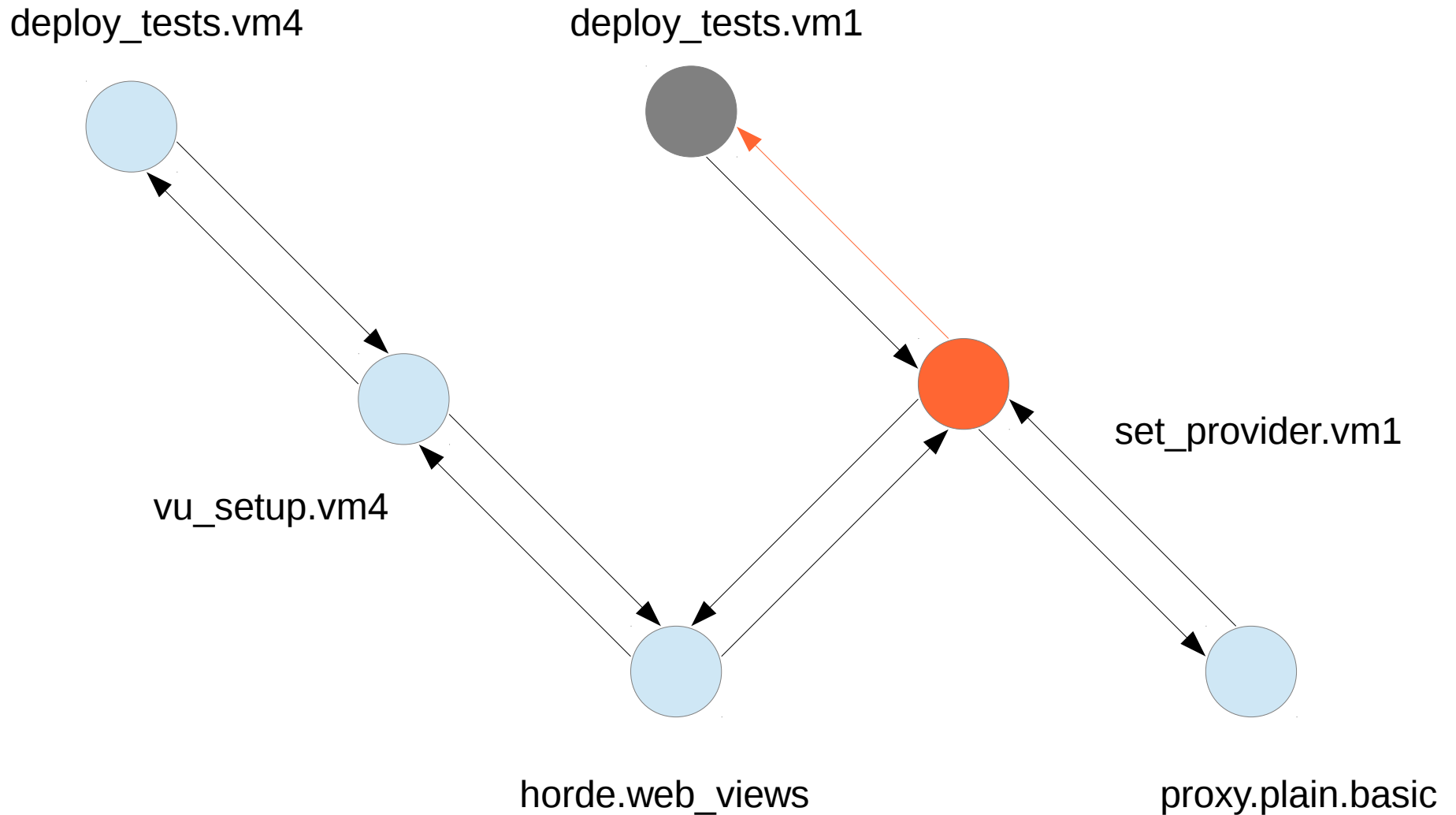2) Use this path to the root that the stack represents to determine whether you are moving down or up

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic

 The direction to determine the next action (moving down or up) as is a matter of a simple check:

1) If the previous node (in the stack) is one of this node's parents, direction is down
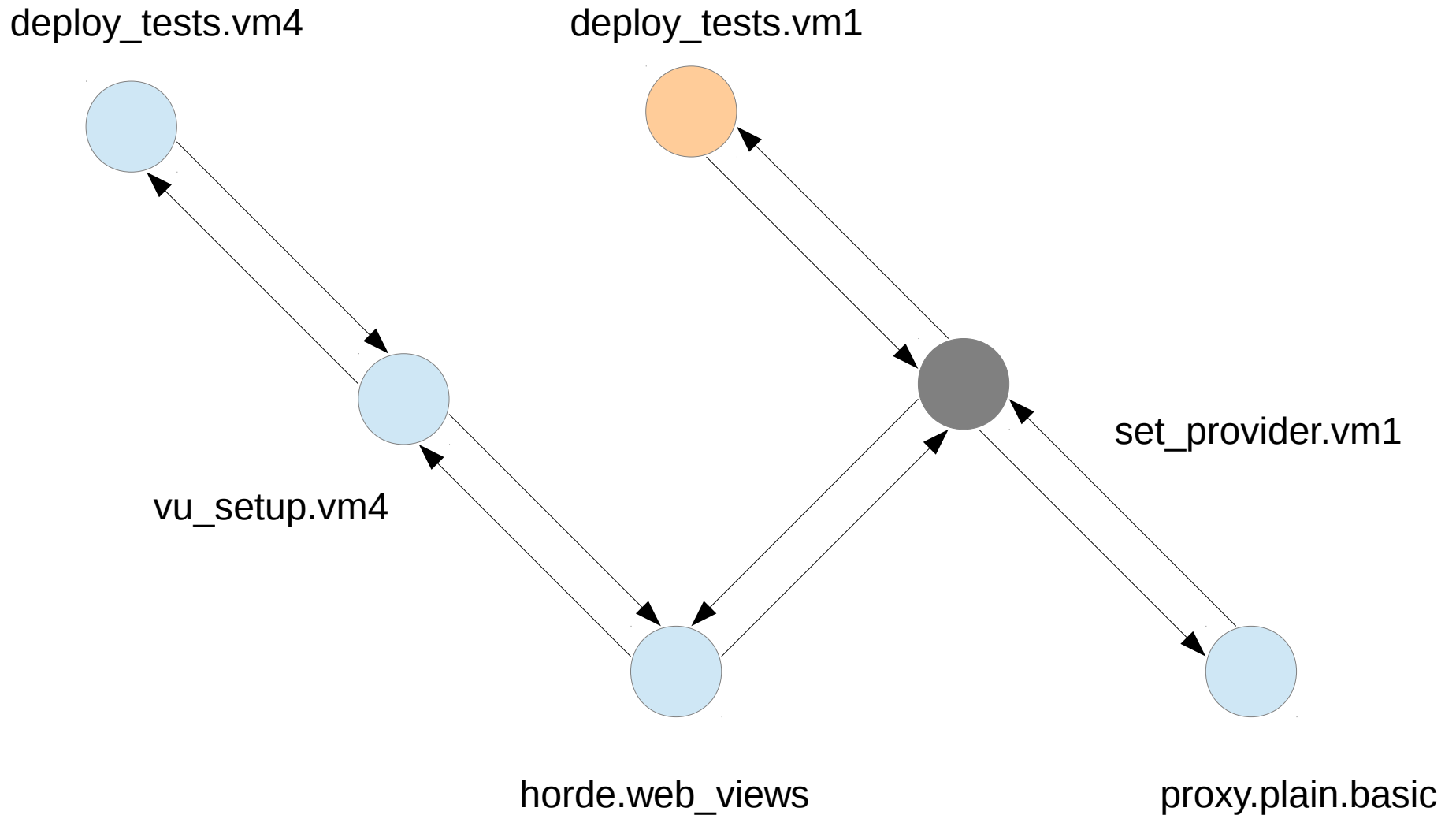2) If it is one of this node's children, direction is up (inverse traversal looking for missing setup)

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic

We can see above or even test with the previous condition that the direction is down, so we continue with the previously showed decision process:

- I**s it setup ready** – False (there is one arrow to parent, i.e. setup to check for availability)
- **Is it cleanup ready** – False (two connections pending so other tests wait to use this one)
- **Is it reusable** – depends on scanning, but let's say it wasn't run (False)

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

What is different now it that the node is not setup ready, so we can't run it or continue down. Instead, we prioritize the setup and reverse the direction to up.

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

I picked light orange instead of orange only to remind you that the stack now contains three nodes (the current path) with repeating *deploy_tests.vm1*:

deploy_tests.vm1 -> set_provider.vm1 -> deploy_tests.vm1

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
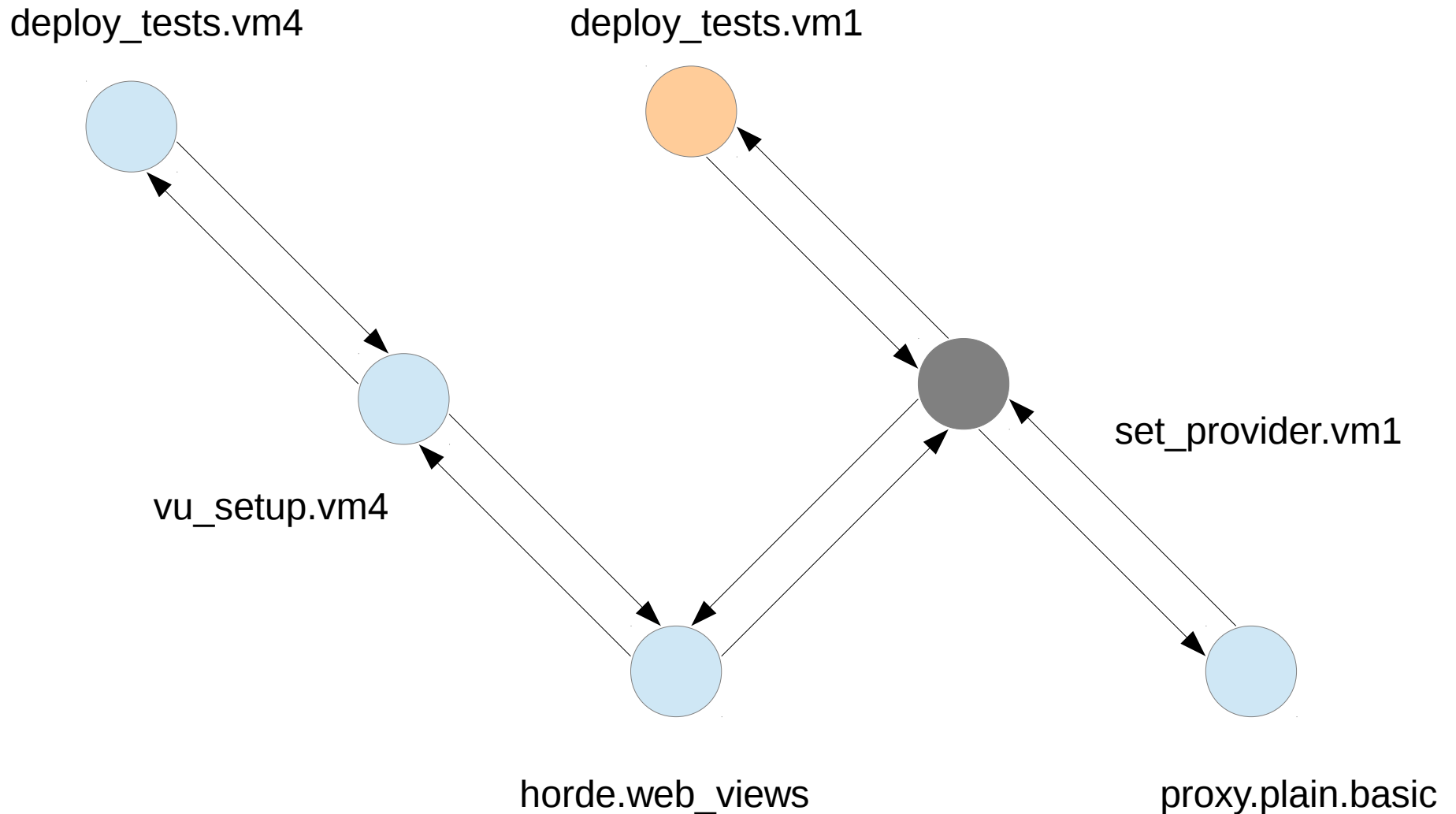
 Of course we use this to detect the we have reversed direction and it is now up, so we make a different set of decisions based on the same three criteria:

• **If node is not setup ready** –> go to its next parent (lowest test count) - inverse DFS
• **If node is setup ready** –> run it if not reusable or skip it if reusable, pop the stack and remove the arrow that led to here
• **Since deploy_tests.vm1 node is setup ready** –> let's observe the second case!

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
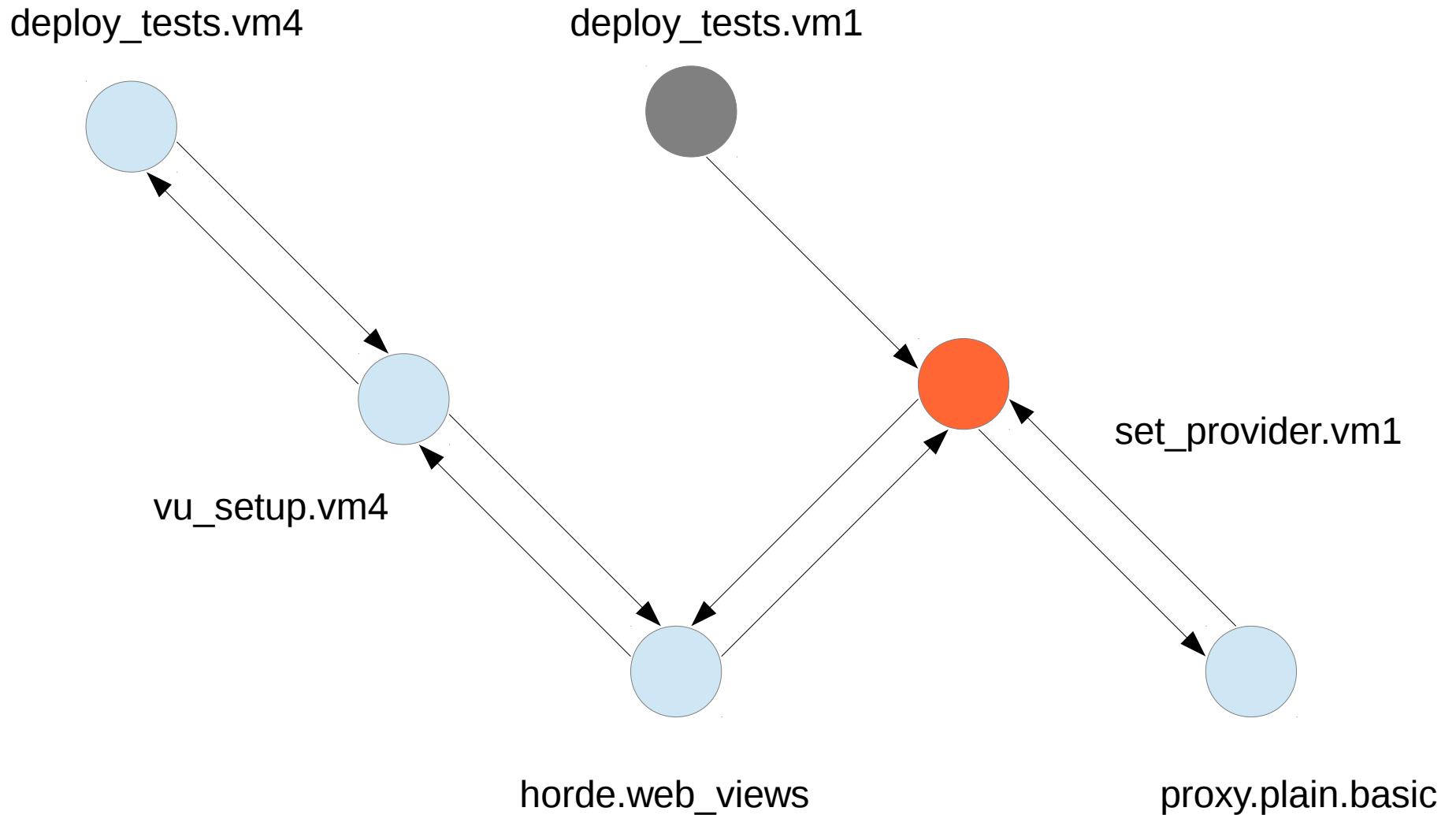
So as we just said, three things happen:

1) Run the test, however rather skip it since we said that it is *reusable*
2) Pop the stack, i.e. go to the last test before this one and make a step backward in the path
3) Remove the arrow from the previous test node to this one (or rather consider it visited)

deploy_tests.vm4

deploy_tests.vm1



set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
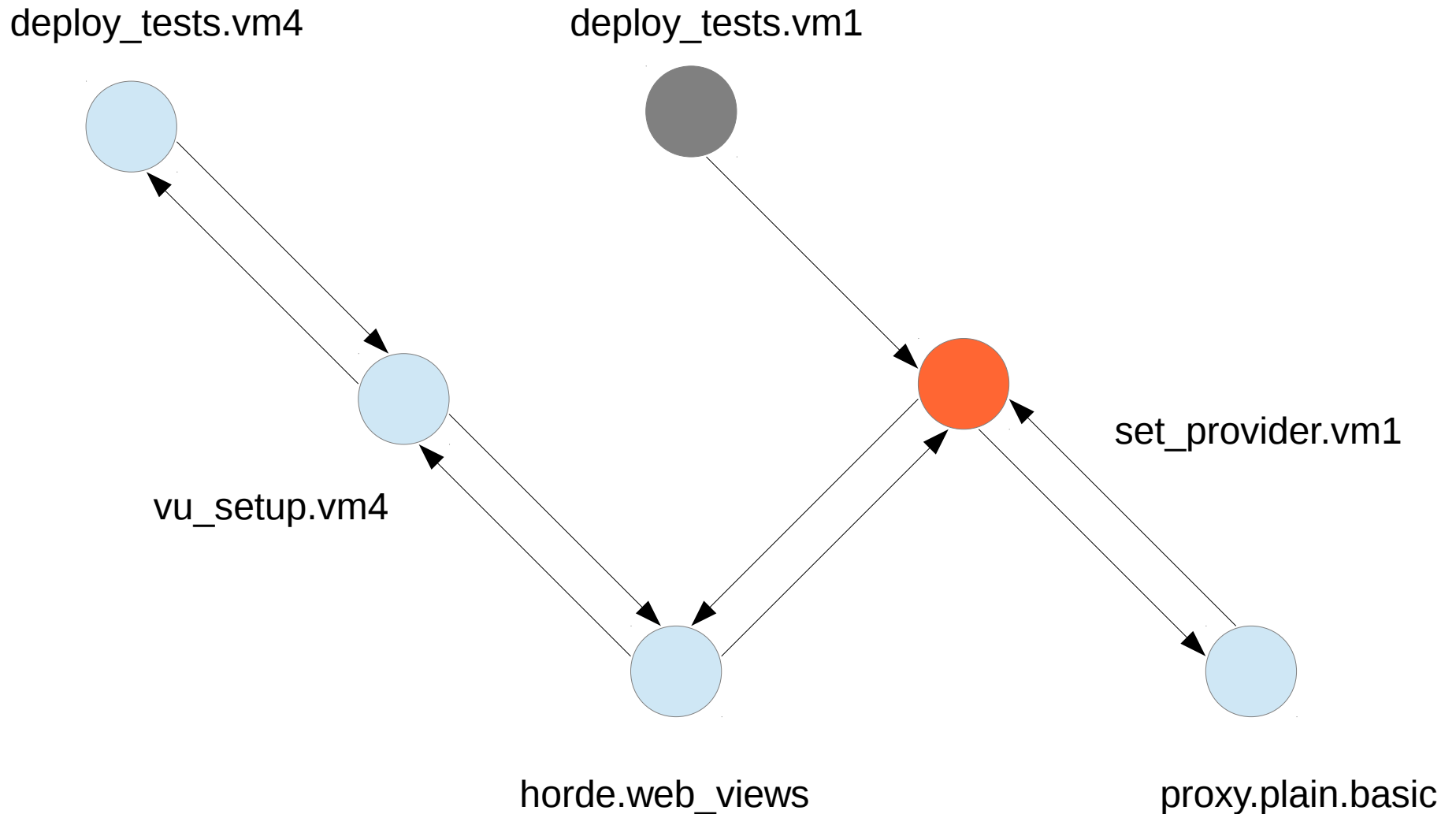
What now? Well checking the direction now it seems we are going down, so our check for these three gives us updated results:

- I**s it setup ready** – True (there are no arrows to parent, i.e. all setup is available and can be run)
- **Is it cleanup ready** – False (two connections are still pending so other tests keep waiting)
- **Is it reusable** – we said it wasn't run (False) so it's time to run it!

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic
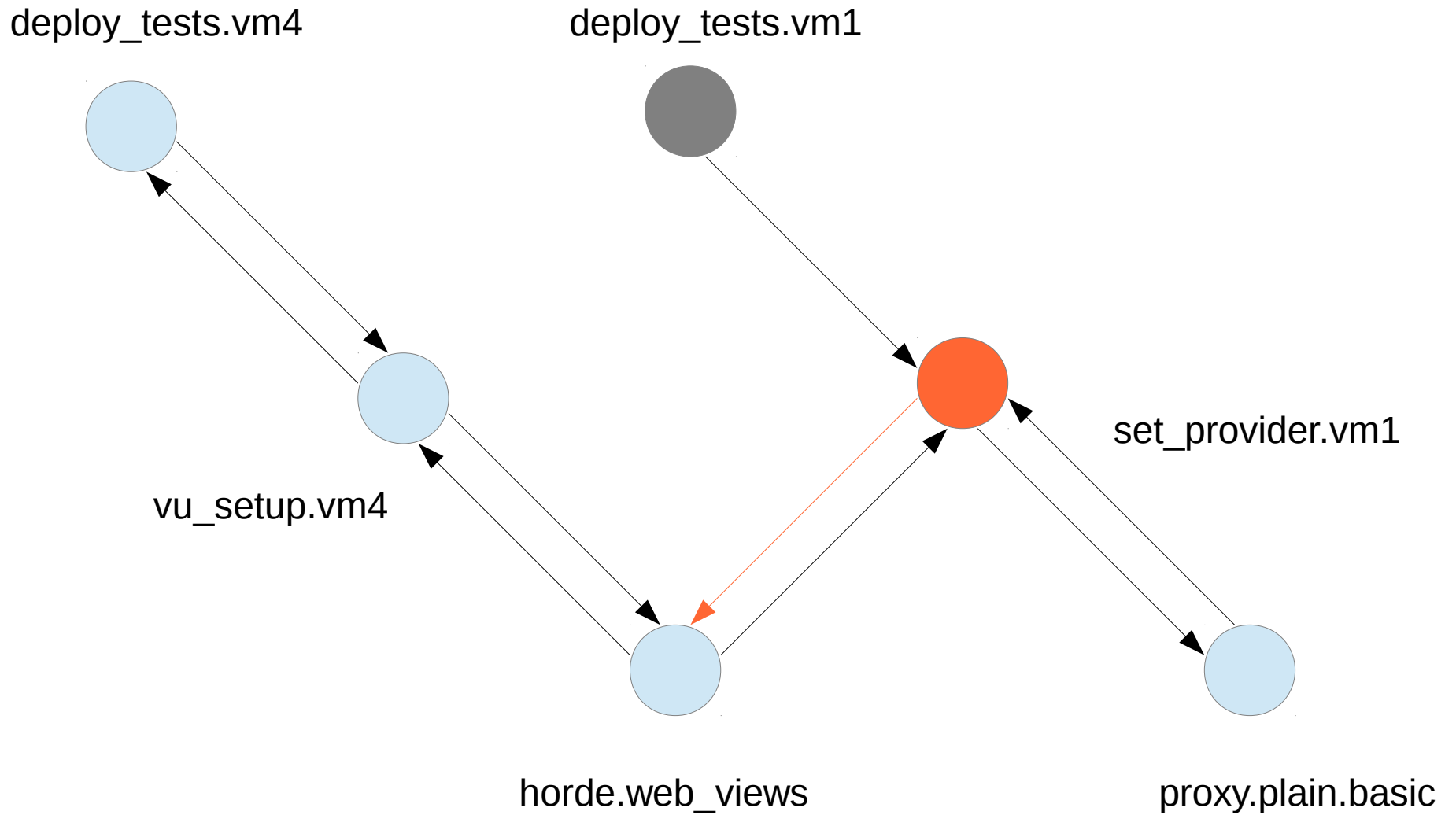
Decisions are as expected – we run the set_provider.vm1 and go to the child with the lowest count (in the future this could be any other scheduling criterion)

deploy_tests.vm4

deploy_tests.vm1
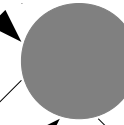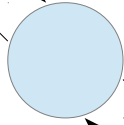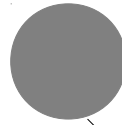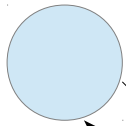
set_provider.vm1

vu_setup.vm4
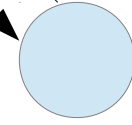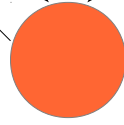
horde.web_views

proxy.plain.basic

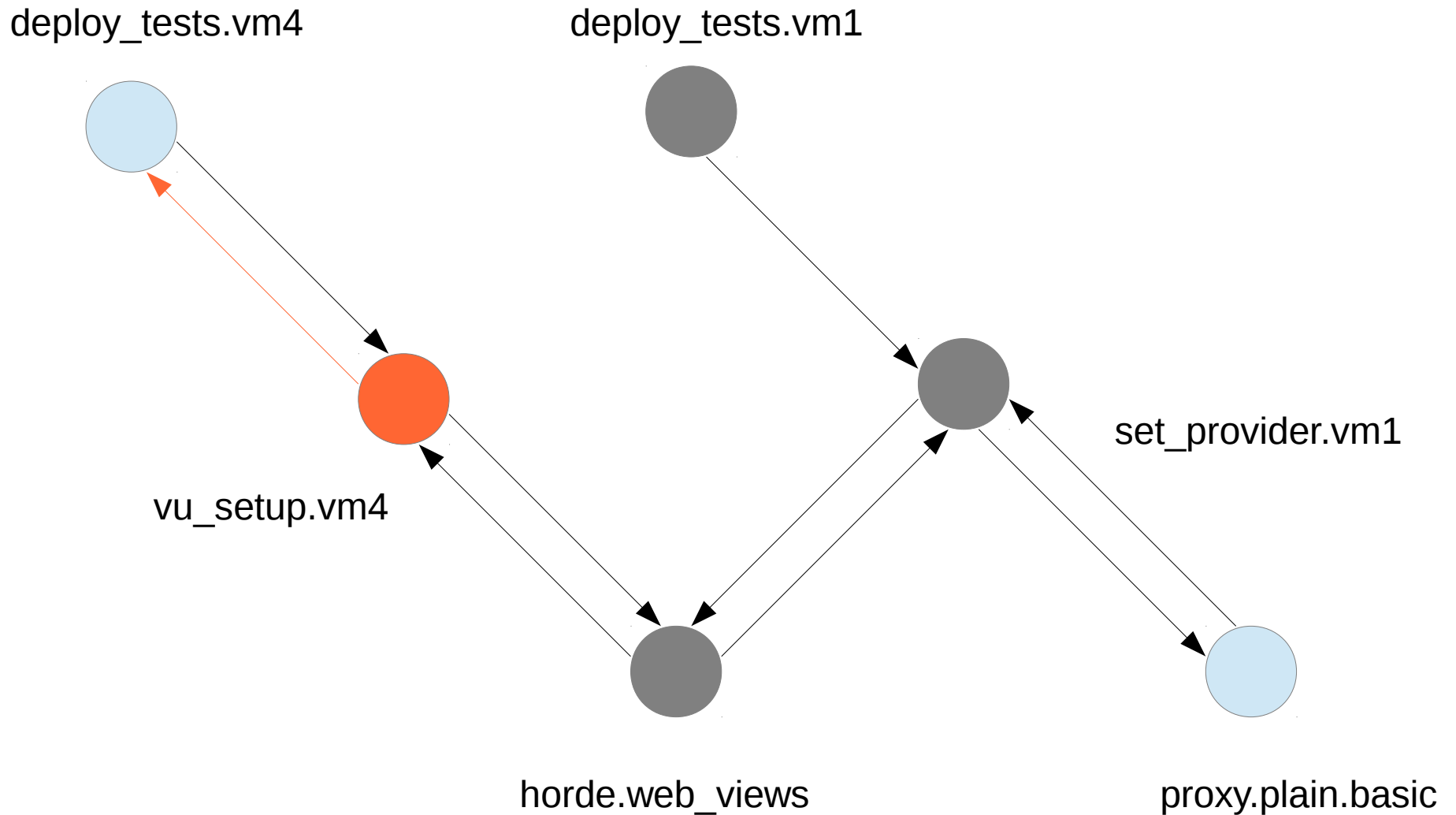It's just about to get boring, but not yet. I promise it it will become a rather tedious process in a few slides ;)

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic
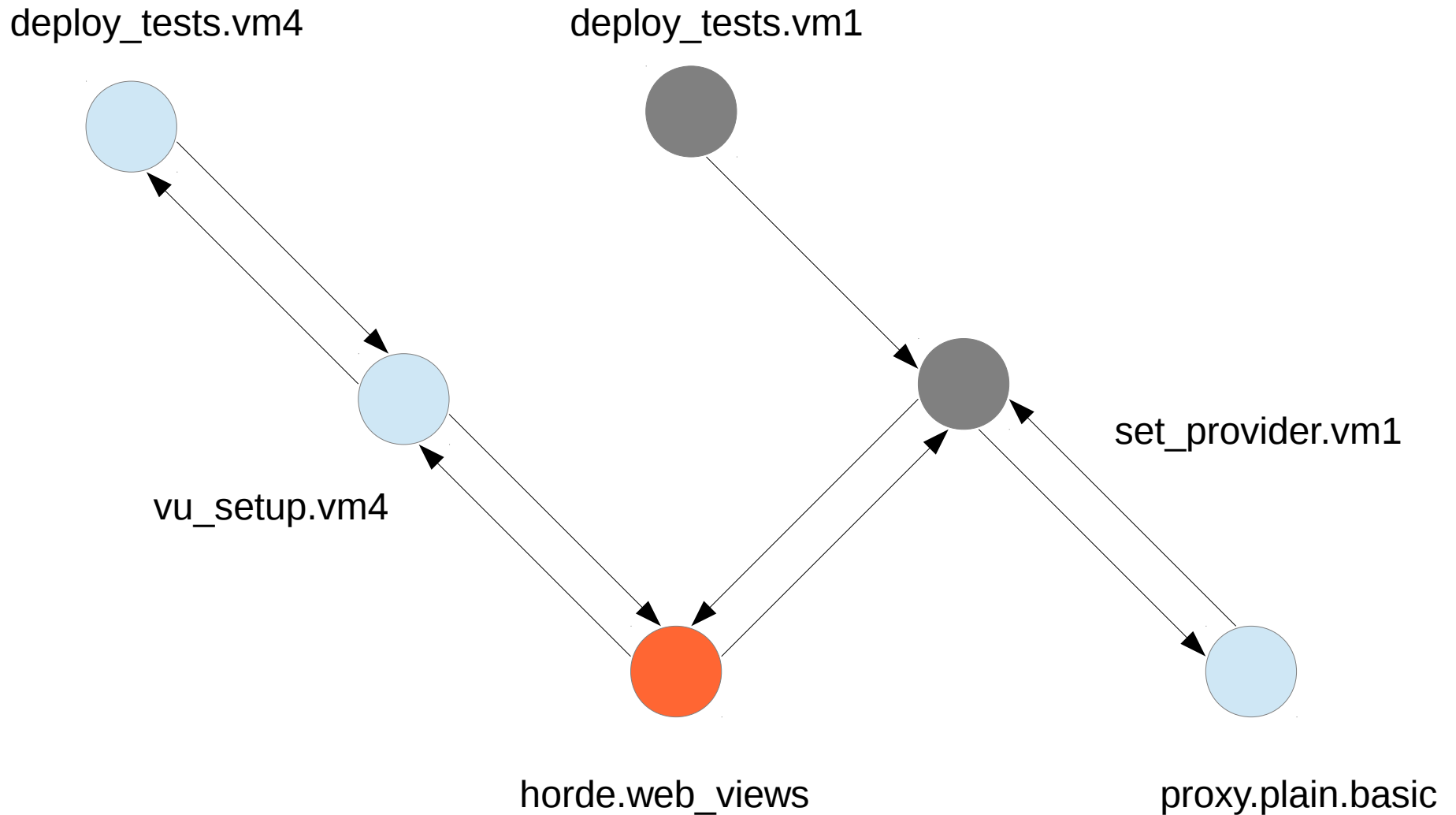
Checking the conditions tells us what to do next:

- I**s it setup ready** – False (there are two arrows to parent, so multiple objects have to be prepared)
- **Is it cleanup ready** – True (no other tests wait and depend on this one, it is a "leaf" test node)
- **Is it reusable** – as a leaf node, this is expected to leave no end states, i.e. it is not reusable (False)

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
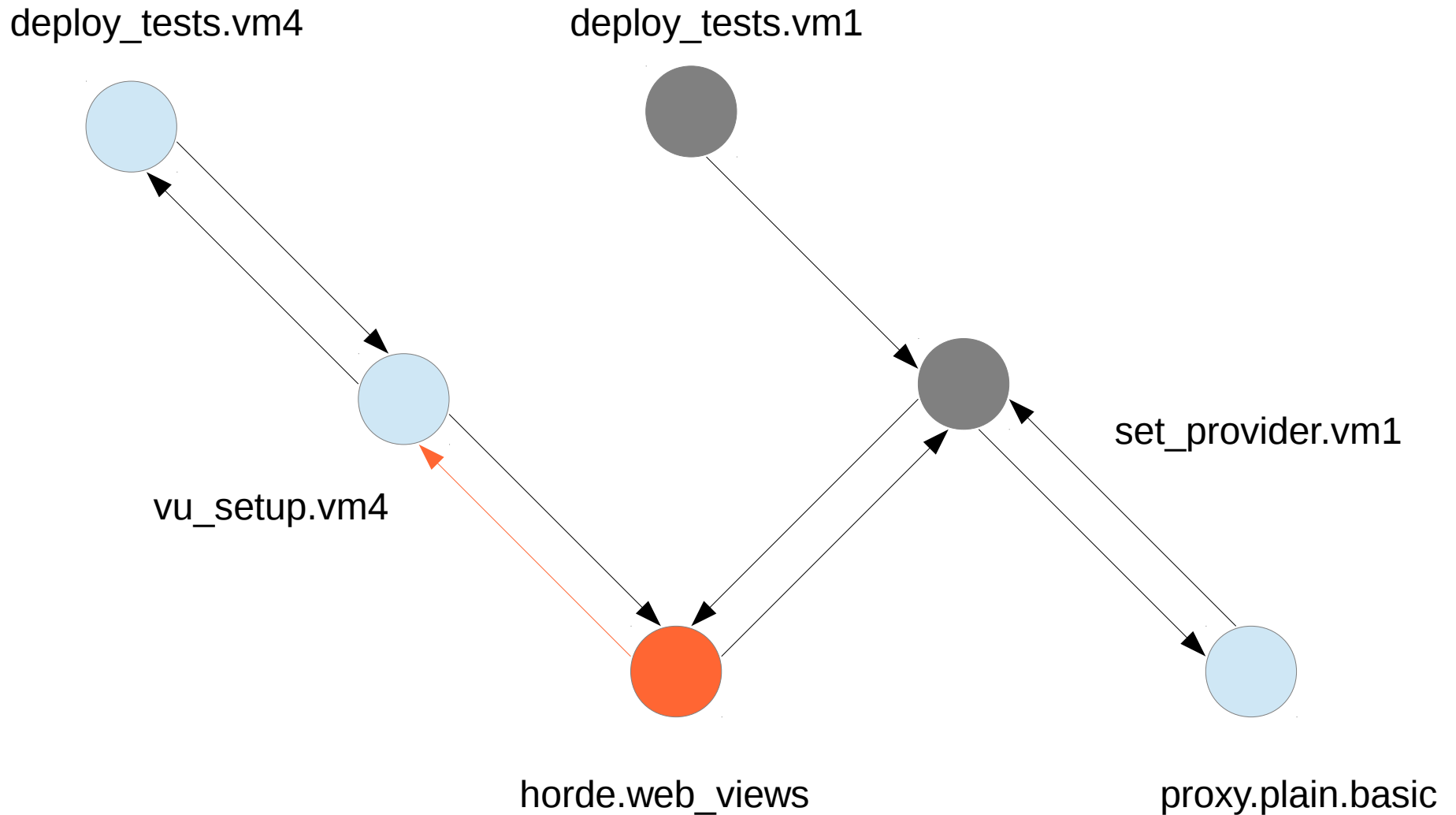
   The test node not being reusable ever means that no matter how many times we run it, we should run it again. However, not just yet - just like before we should inverse the direction and go to the next parent based on count or other priority.

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

Inverse direction makes us check only if the node is setup ready. No. So we go to the next parent based on count (inverse DFS as promised).

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic
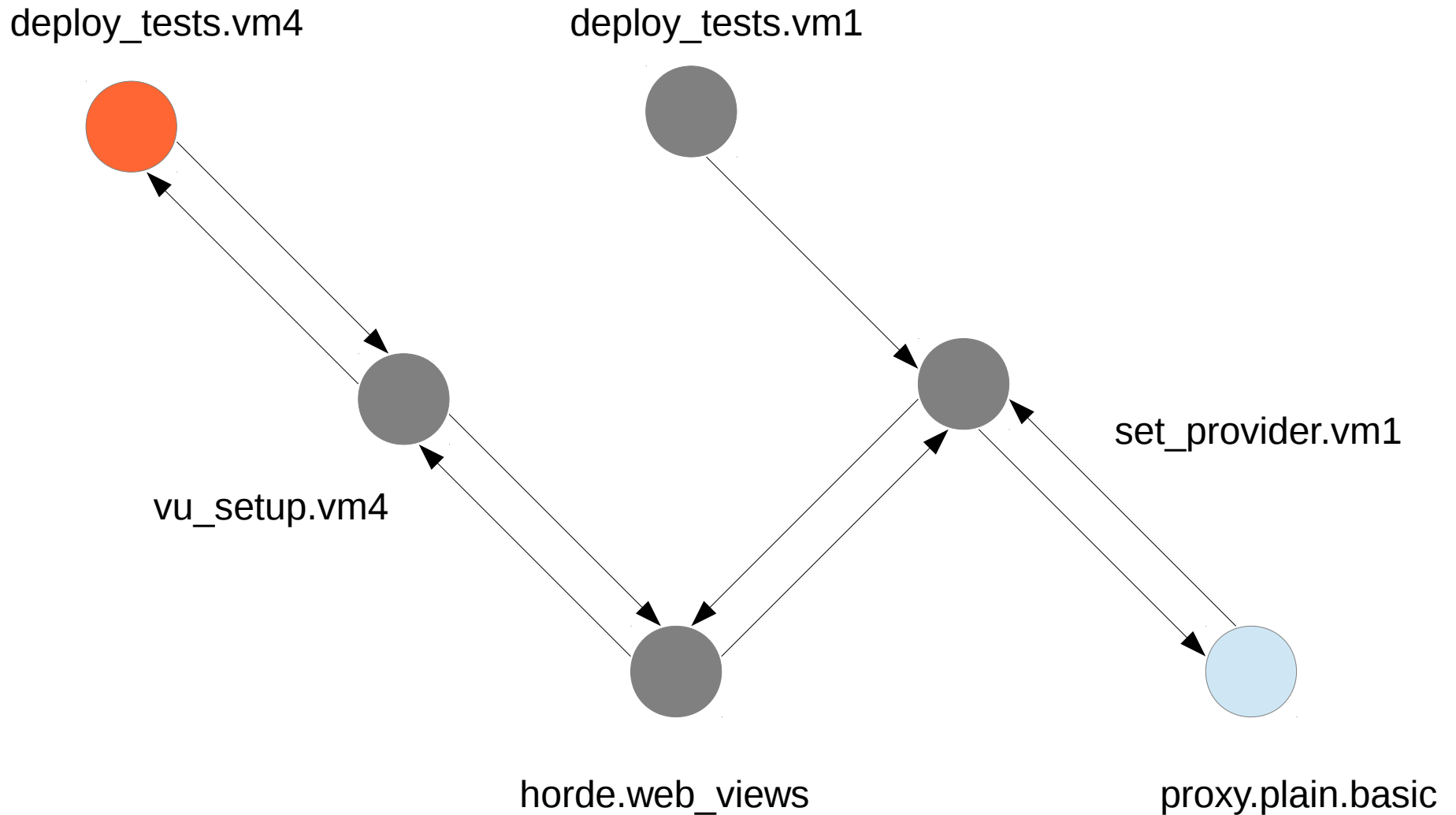
As this seems to be another root state (not really but for the sake of simplicity yes yes yes), we repeat the same thing like *deploy_tests.vm1* so it should lead to...

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views
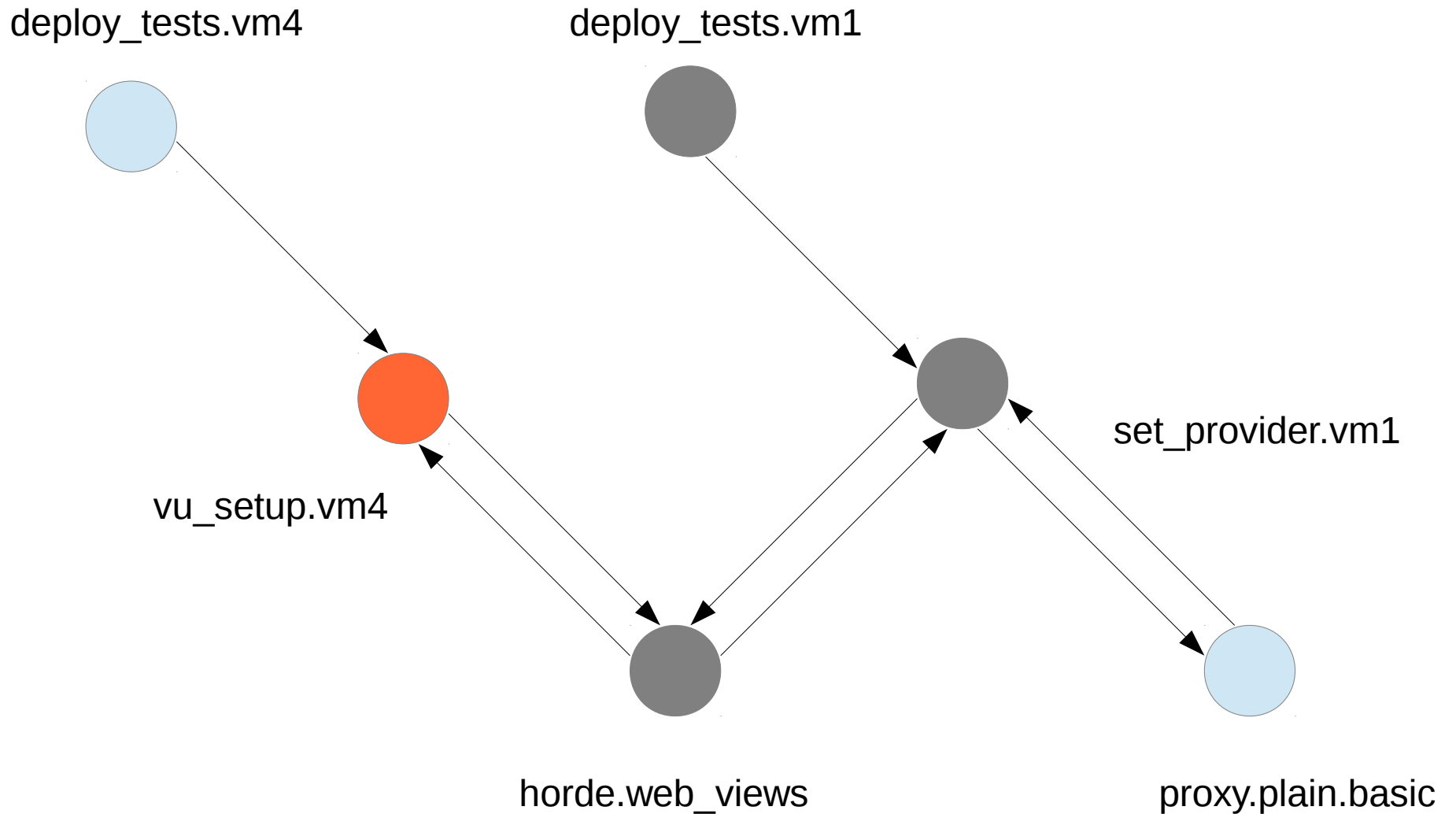
proxy.plain.basic

...this. Now the vu_setup.vm4 is setup ready so we make another step back running yet another test (I forgot to say we consider it not reusable).
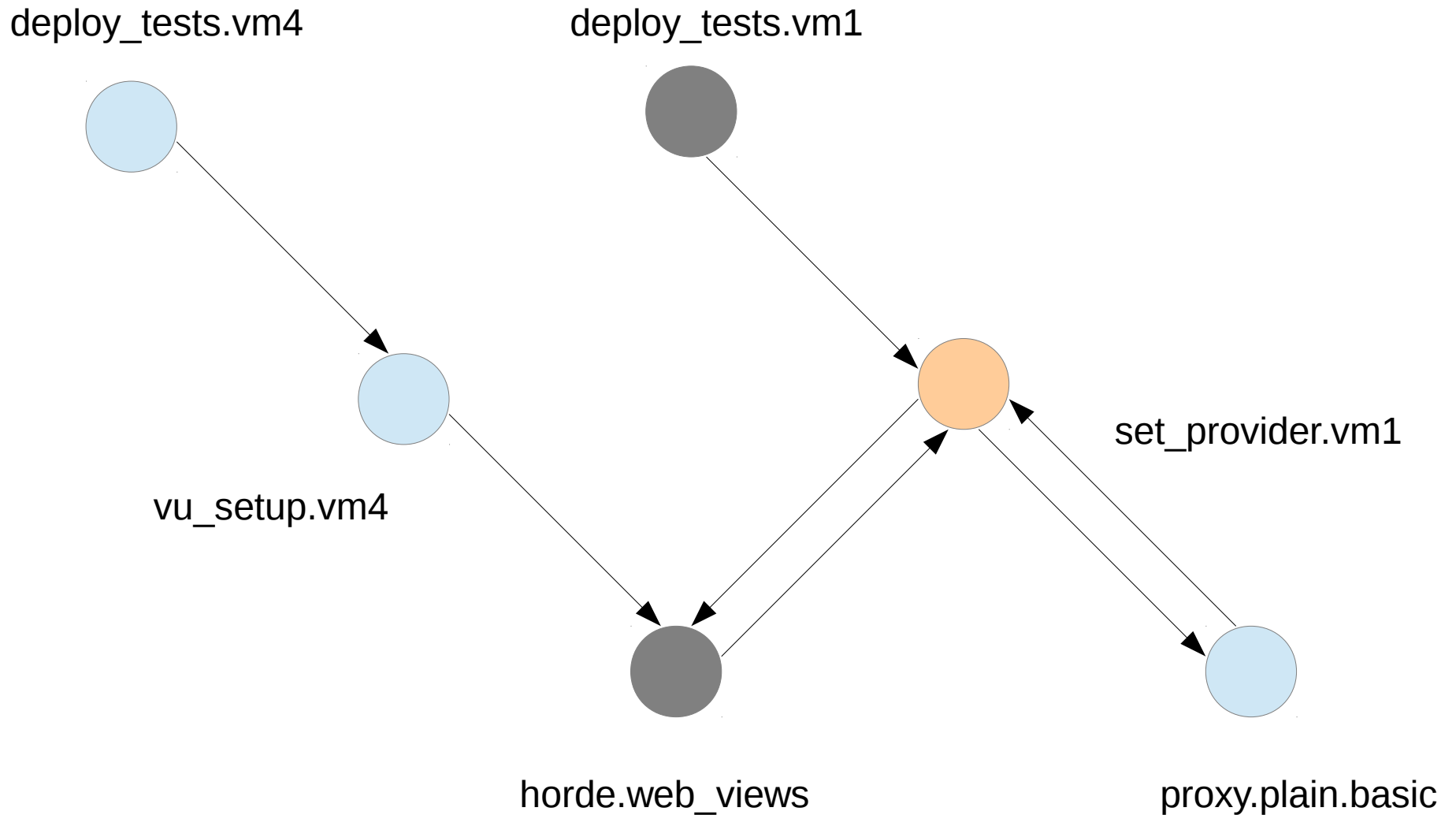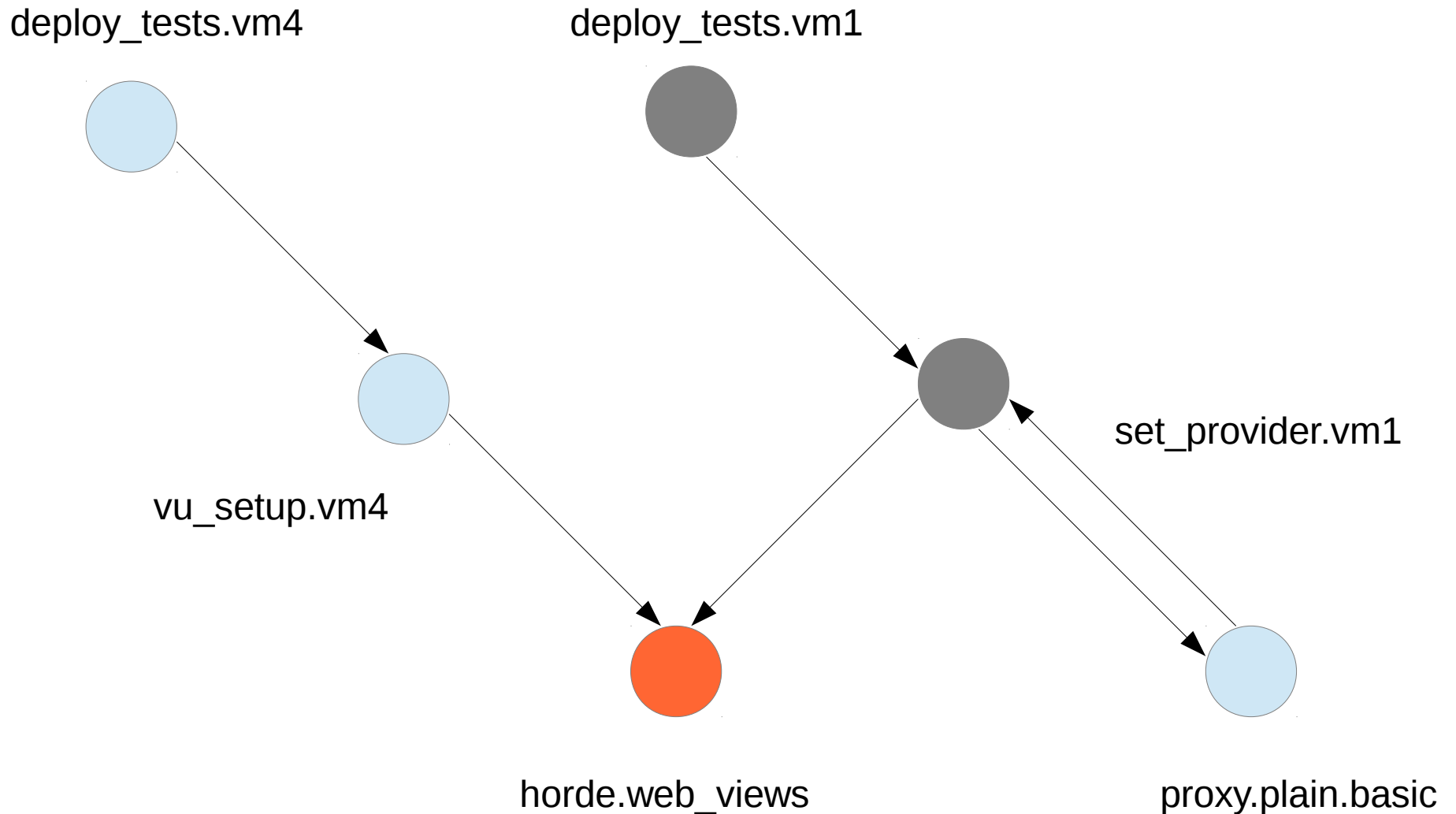
deploy_tests.vm4
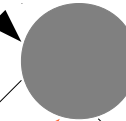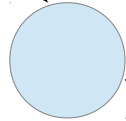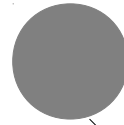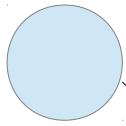
deploy_tests.vm1

vu_setup.vm4
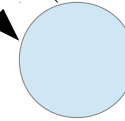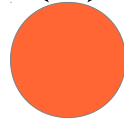
set_provider.vm1

horde.web_views

proxy.plain.basic

Is horde.web_views setup ready now? Not yet - one more object needs a setup but this one should be easier, as set_provider was run and left behind an end state to be reused.

Since set_provider.vm1 is reusable (True) we skip it. Good but a question here would be where now? The direction is up, so the three steps apply: run it (done), remove the arrow (done in next slide), and pop the node. Oh, that should be it then – we pop the node and go to the previous node, i.e. horde.web_views.

deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic

Is horde.web_views setup ready now? Finally it is. So we run the test. What after we run it? Now we get the chance to finally handle cases when the node is cleanup ready. And how to do it? Well it almost the same like handling the setup ready case in upward direction, but mirrored for the down direction.

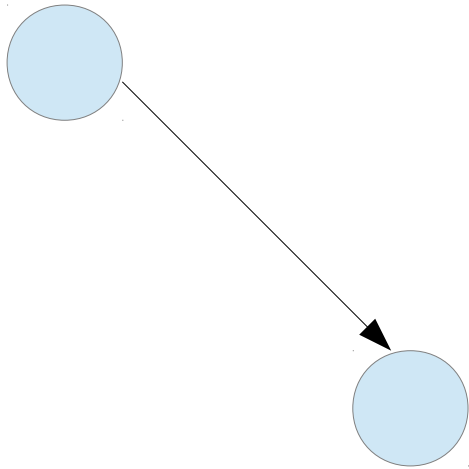deploy_tests.vm4
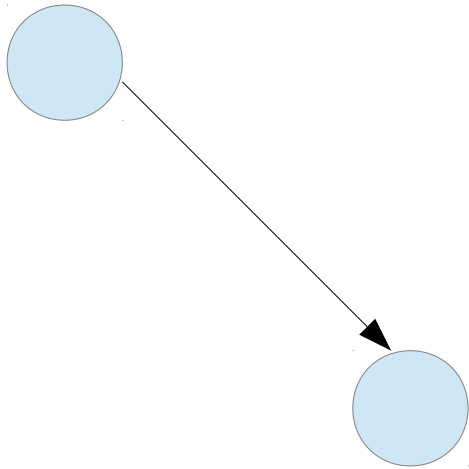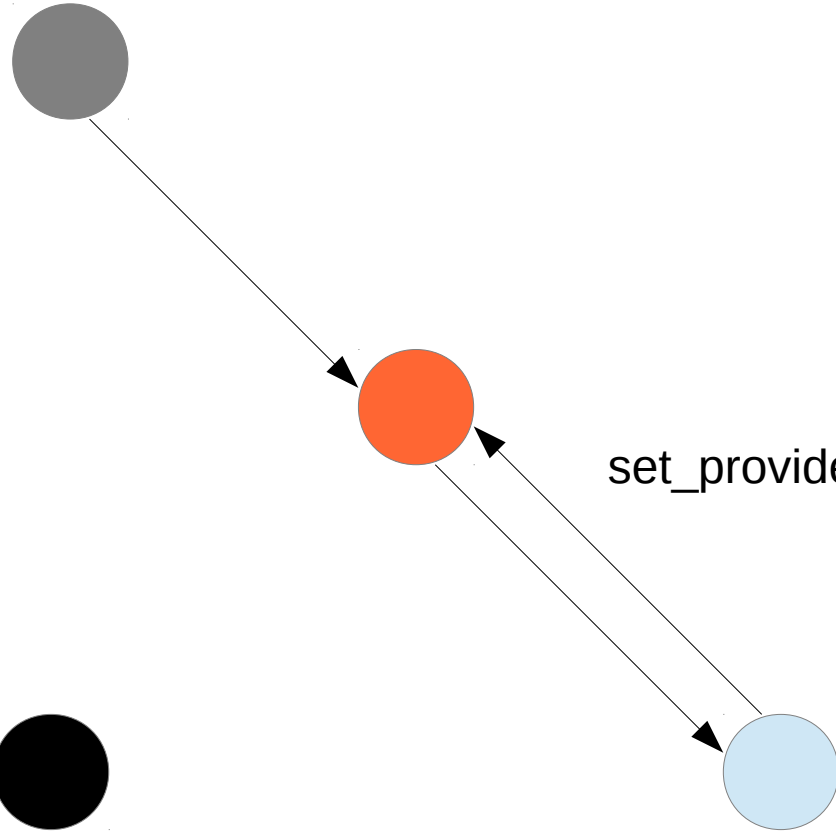
deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic

So as we just said, three things happen:
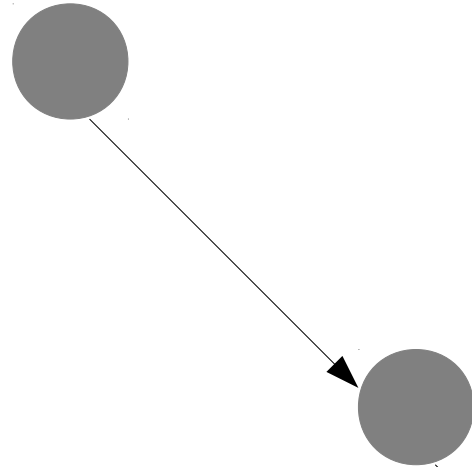
1) Run the test, we said that already
2) Pop the stack, i.e. go to the last test before this one and make a step backward in the path
3) Remove all arrows from the parent nodes to this one (or rather consider it visited child for all)
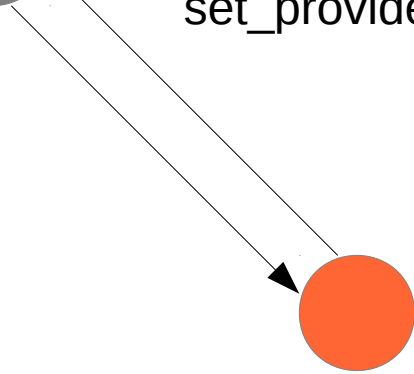
deploy_tests.vm4

deploy_tests.vm1
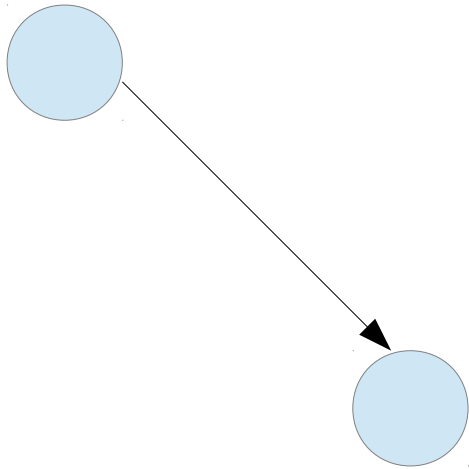
vu_setup.vm4

set_provider.vm1
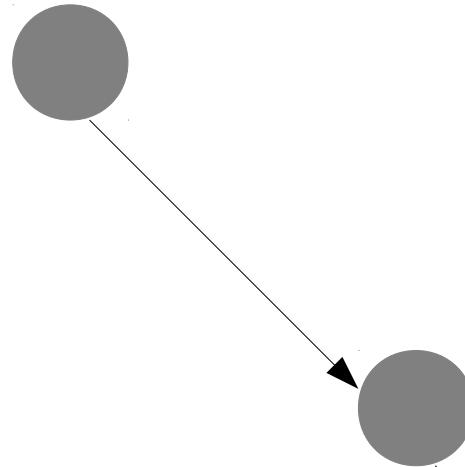
horde.web_views

proxy.plain.basic

Hold on, we are almost there! We now continue our regular DFS with the next child which turns to be not setup ready, cleanup ready, and not reusable (leaf).
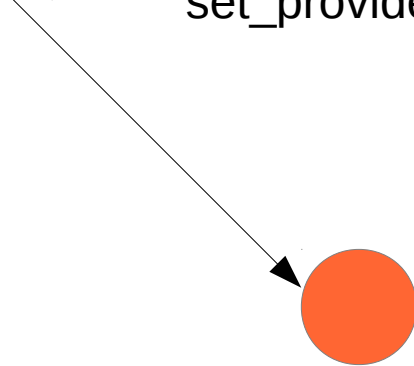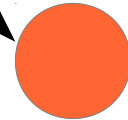
deploy_tests.vm4

deploy_tests.vm1

vu_setup.vm4

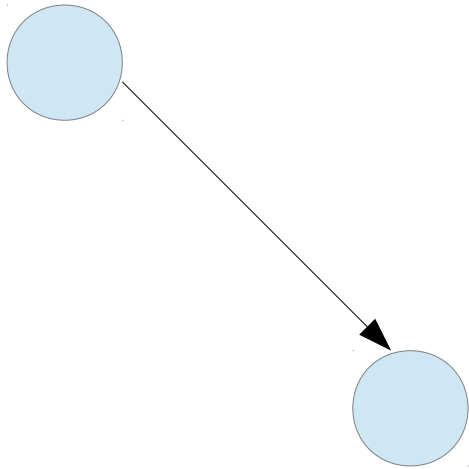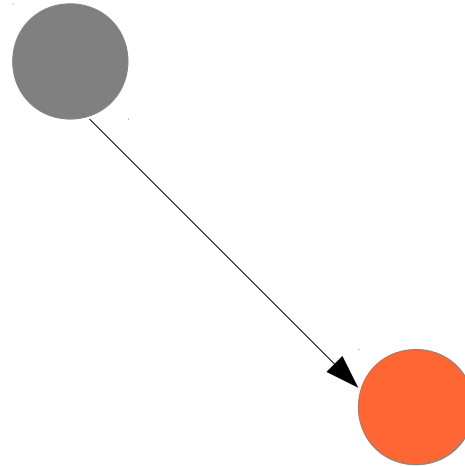set_provider.vm1

horde.web_views

proxy.plain.basic

I promised it will become tedious although I am a few slides down that promise.
The setup for this node was reusable/skipped, the inwards arrow was removed
and now we are about to make the three cleanup ready steps again…

deploy_tests.vm4

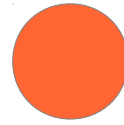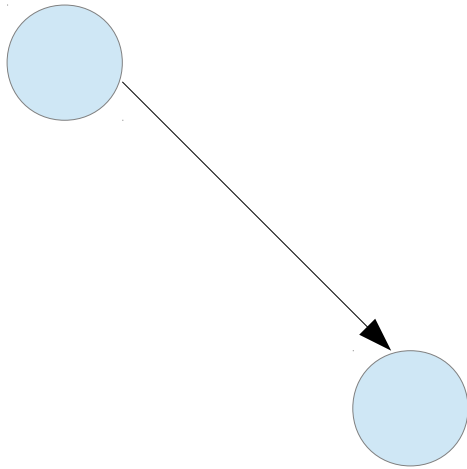deploy_tests.vm1

vu_setup.vm4

set_provider.vm1

horde.web_views

proxy.plain.basic

Finally the set_provider.vm1 node is cleanup ready! As you can see the test is no longer needed for anybody so this would be the perfect place to perform other cleanup steps like removing object states, etc. and not just popping.

deploy_tests.vm4

deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

The size of the stack has reduced to one... I leave the nodes incinerated for visual effects. All to show that there is a branch that will never be traversed and it's child tests never to be run. How can we set it on fire? Well, with a single real root test node which we skipped here for simplicity. If our current node was that root, it would have continued down its children, reaching these nodes.

deploy_tests.vm4
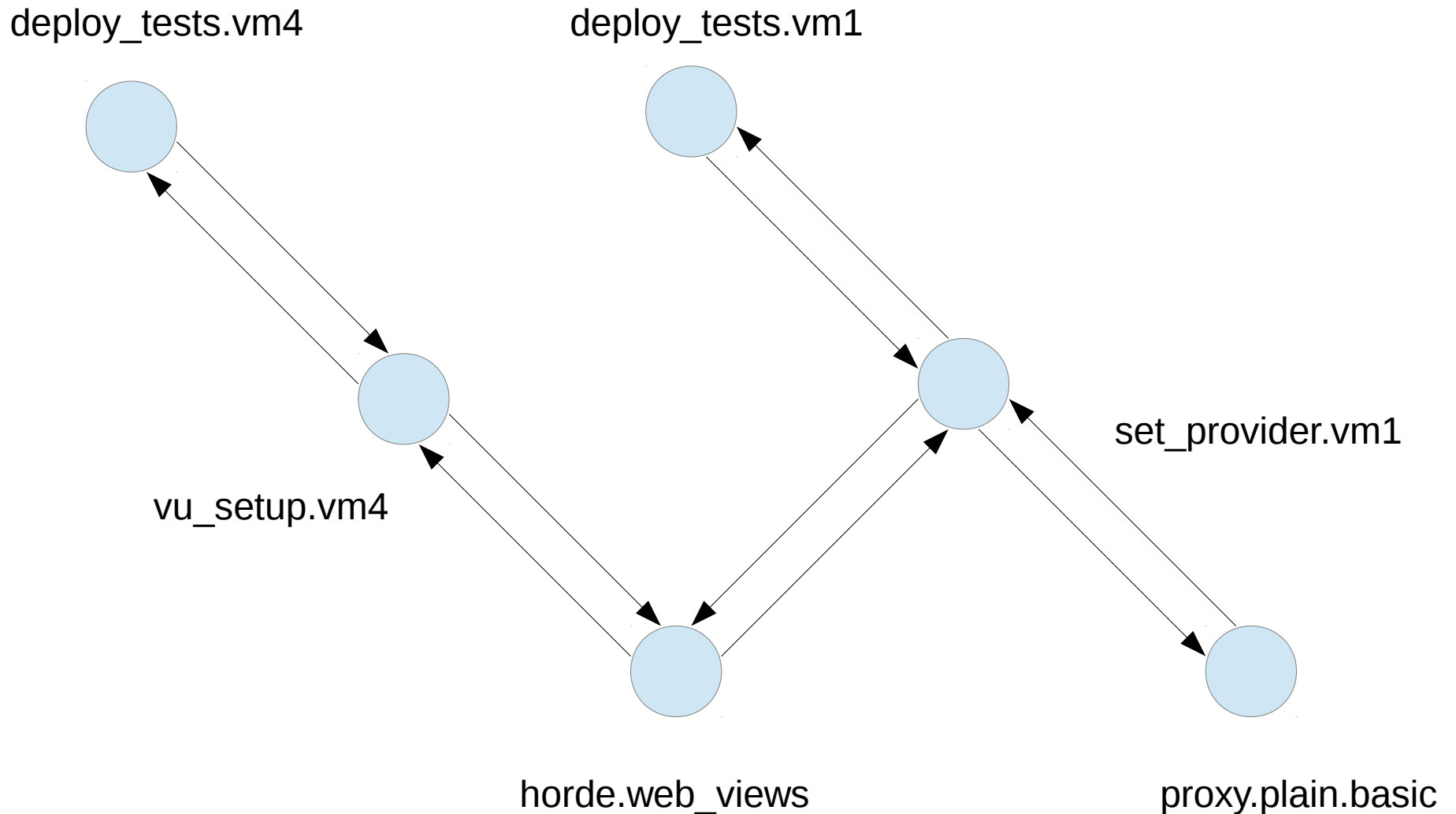
deploy_tests.vm1

set_provider.vm1

vu_setup.vm4

horde.web_views

proxy.plain.basic

As you might have already guessed the exit condition is:

root_node.is_cleanup_ready() == True.

Instead of the classical test-defined "setup" and "cleanup" procedures, it is so much nicer for a fellow test to just put all its mess in one object state and allow other to reuse it or simply forget it. This structure is trying to make the development of each test easier taking care of both parts but also of scheduling of the tests so that maximum mess is reused :) *Happy testing!*