```ruby
def test_trace
  clipp(CONFIG.merge(
    default_site_config: <<-EOS
     Action id:1 phase:REQUEST_HEADER clipp_announce:a "predicate:(eq 'foo' (sub 'x' (var 'ARGS')))"
     Action id:2 phase:REQUEST_HEADER clipp_announce:b "predicate:(eq 'bar' (sub 'x' (var 'ARGS')))"
     Action id:3 clipp_announce:c "predicate:(or (eq 'bar' (sub 'x' (var 'ARGS'))) (eq 'bar' (sub 'y' (var 'ARGS'))))"
     PredicateTrace "/tmp/ptrace"
    EOS
  )) do
    transaction do |t|
      t.request(
        raw: 'POST /a/b/c?x=bar&y=foo HTTP/1.1',
        headers: {
          "Content-Type" => "application/x-www-form-urlencoded",
          "Content-Length" => 5
        },
        body: "y=bar"
      )
    end
  end
  assert_no_issues
  assert_log_match /PredicateTrace/
end
```

1: Here is a clipp test with a couple predicate rules.

2: This directive tells IronBee to write a predicate trace to `/tmp/ptrace`.

3: You can also pass in "" to have it output the trace to stderr. This can be useful with Clipp.

4: There are many caveats:

- Will **append** to file.
- Will behave poorly in multithreaded IronBee.
- Will kill performance.
- Uses lots of memory.
- Does not scale well to large Predicate graphs.

5: But, it can be useful for debugging small sets of Predicate rules; for understanding Predicate; and for understanding IronBee.

6: The `PredicateTrace` directive is scoped to the current context, so you can trace some contexts but not others.

7: To process, run the `predicate/render_ptrace.rb` script, passing in `/tmp/ptrace` as the first argument.

The output is HTML; this is the first part.

```
PredicateTrace REQUEST_HEADER context=default:location:/ consider=3 inject=2
```

8: This header indicates a PredicateTrace block. There is a PredicateTrace block for any phase and context that Predicate did something in. It also shows the number of rules (root nodes) considered and the number injected (fires). You can click on it to toggle display of the block.

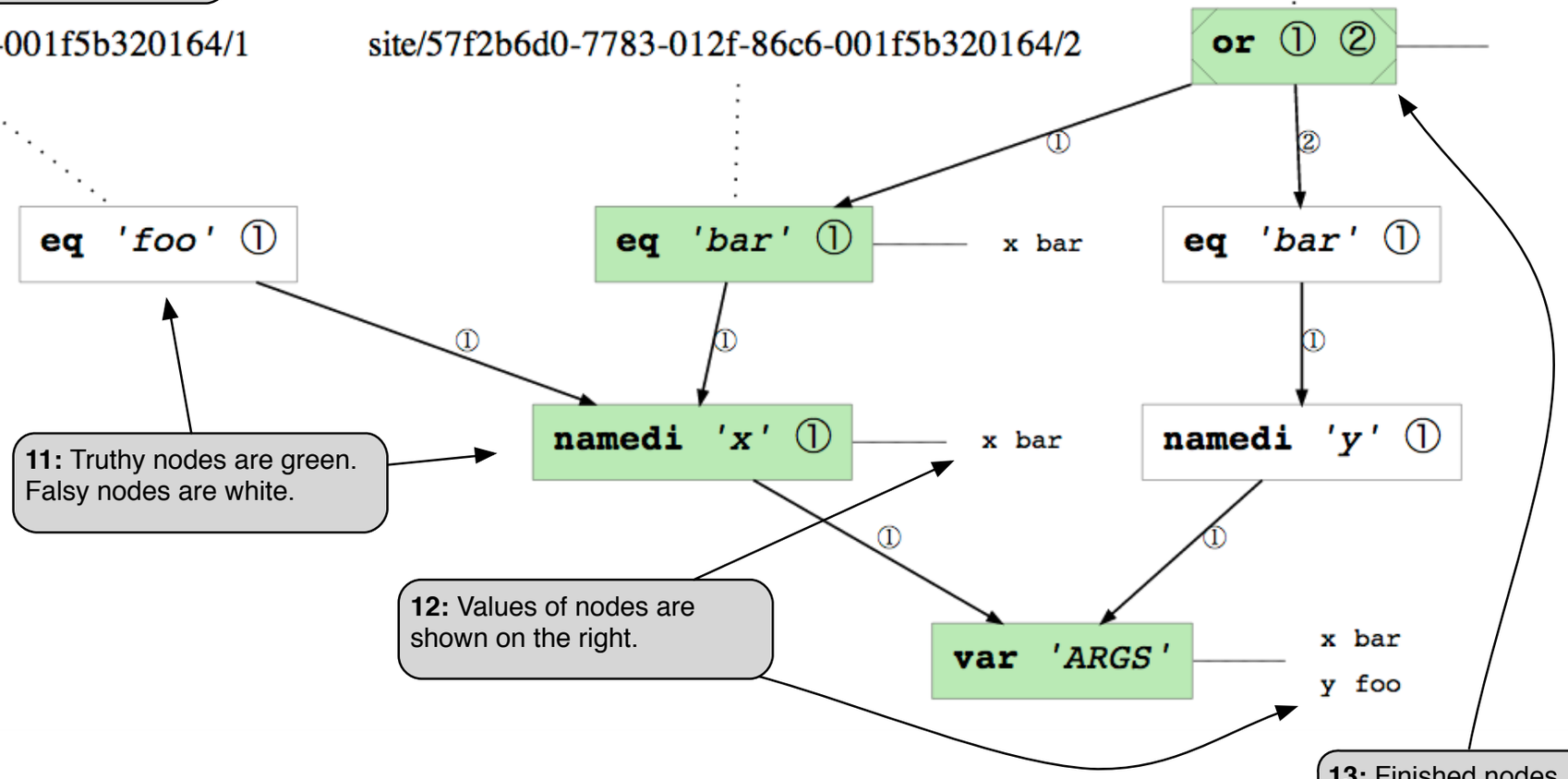
9: Each block is shown with the portion of the DAG relevant to that block, i.e., all root nodes for that phase and context along with their descendants.

site/57f2b6d0-7783-012f-86c6-001f5b320164/3

site/57f2b6d0-7783-012f-86c6-001f5b320164/1

site/57f2b6d0-7783-012f-86c6-001f5b320164/2

10: This is a root label indicating that this node is the root node for rule 1.



11: Truthy nodes are green. Falsy nodes are white.

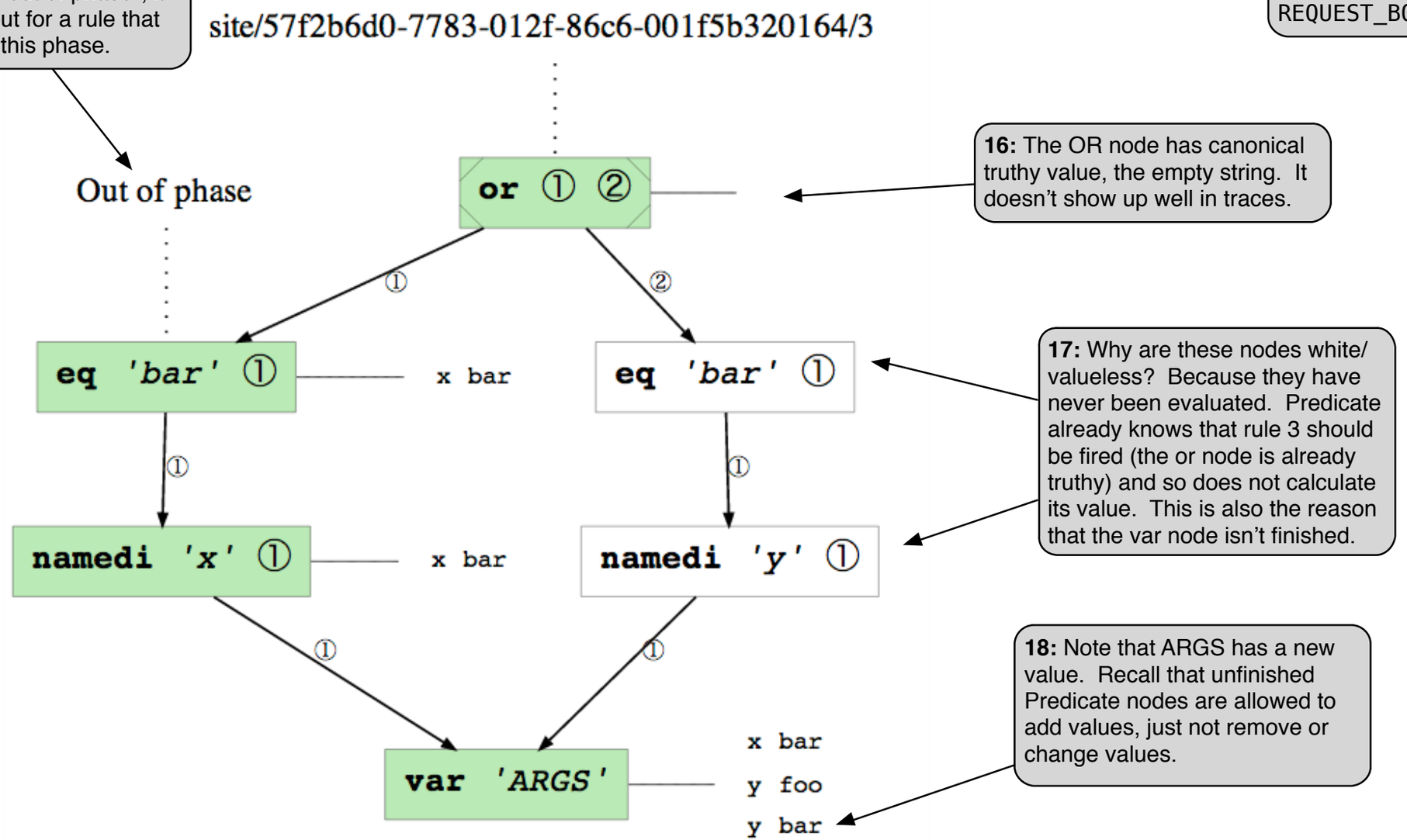
12: Values of nodes are shown on the right.

13: Finished nodes have triangles.

```
PredicateTrace REQUEST_BODY context=default:location:/ consider=1 inject=1
```

14: Here is the second part of the trace, the block for REQUEST_BODY.

15: This root is “out of phase”; it is a root node but for a rule that does not run in this phase.



16: The OR node has canonical truthy value, the empty string. It doesn’t show up well in traces.

17: Why are these nodes white/valueless? Because they have never been evaluated. Predicate already knows that rule 3 should be fired (the or node is already truthy) and so does not calculate its value. This is also the reason that the var node isn’t finished.

18: Note that ARGS has a new value. Recall that unfinished Predicate nodes are allowed to add values, just not remove or change values.