T15

Cfengine 3
An unveiling

Mark Burgess

1. Introduction

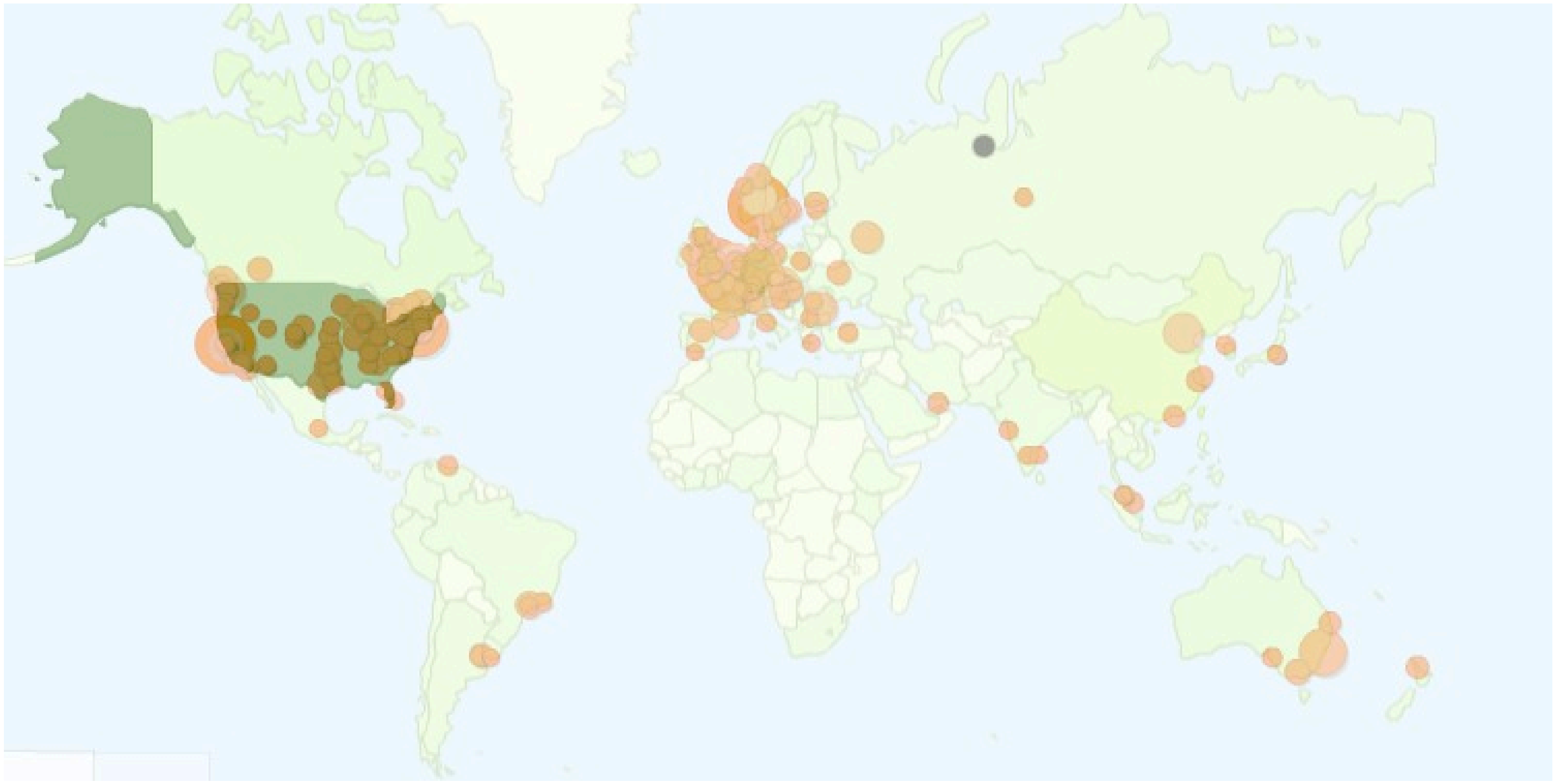
The start of a next generation tool

- Complete rewrite of “the best of” cfengine
- Complete rationalization
- All known limitations removed
- Many historical quirks redesigned
- Consistent syntax
- Everything from a single conceptual model
- Integrated knowledge management

What is cfengine?

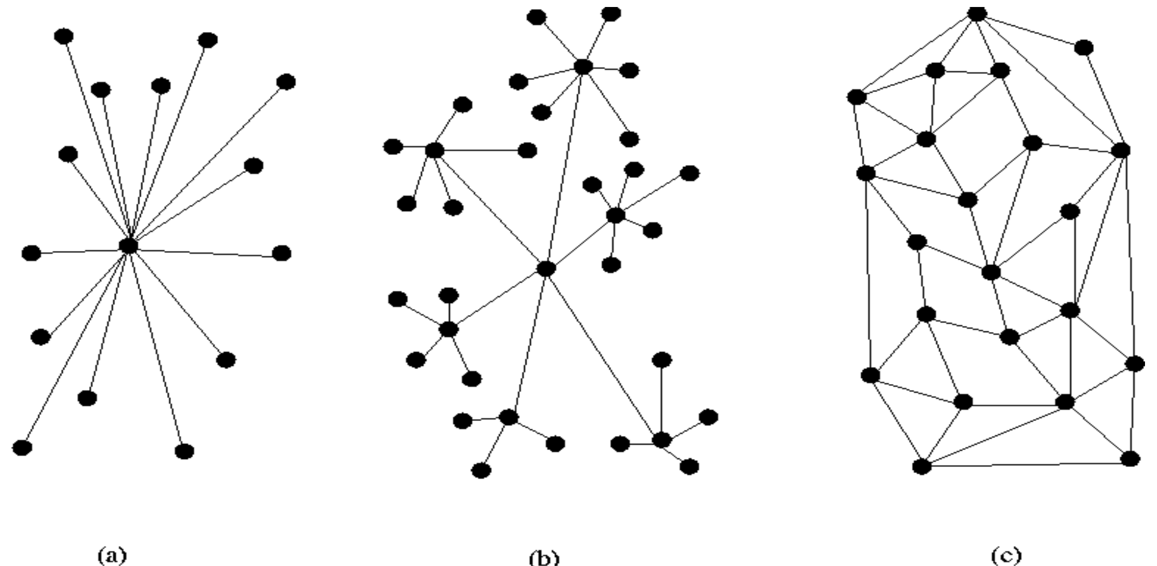
- Automation for the data-centre
  - Software for installing, maintaining and auto-monitoring networked computers (unix-like)
  - Free, open and GPL2
  - Unix and some Windows
  - Good for 1 or 30,000 hosts
  - 15 years and a million computers

Who is using it?



All management models

- There are various “theories” of administration:
  - Everything is centrally controlled from one place
  - Some or all systems remain independently controlled (autonomous)
  - Systems are federated (P2P) – offer each other services and help each other out, but no one is “boss”.

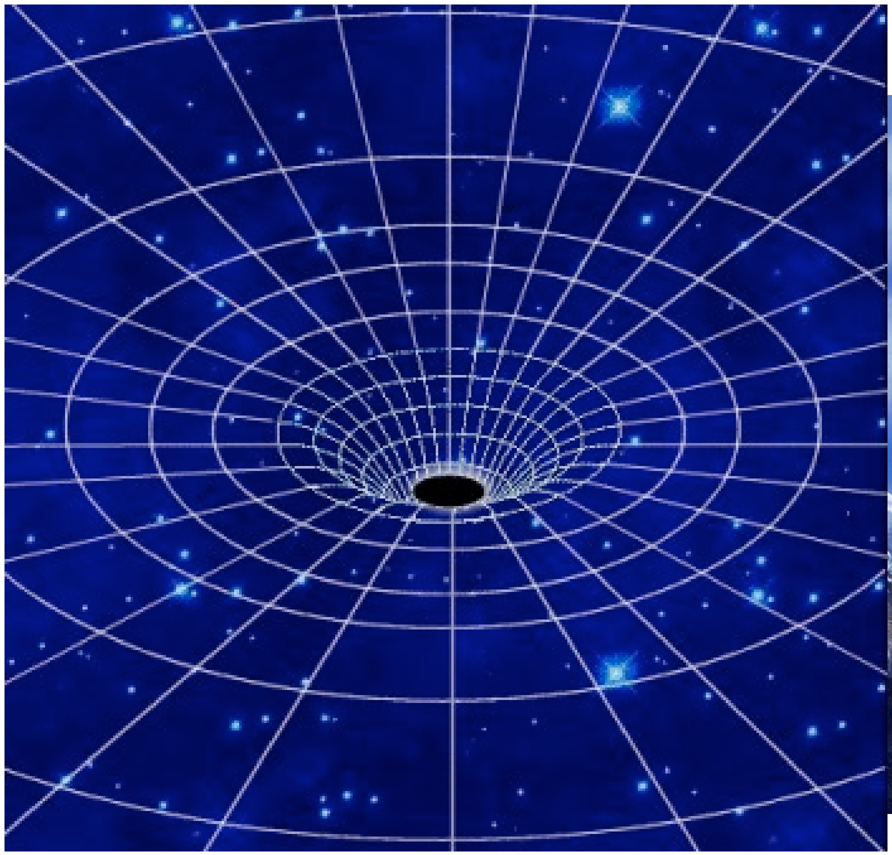


Move on from scripts

- Procedural scripts can be clumsy
  - Humans make many mistakes
- Descriptive language
  - Like “style sheets” for operating systems
  - Separate description from implementation

```
processes:
  “/important/process”  signal=hup
                        restart “/etc/init.d/process restart”
                        ifelapsed=240
```

Two kinds of change process



"Convergence" to end state

Cfengine way

Baseline and grow

Scripting approach

Rationalization

- Consistent data-model
  - All components share the same configuration
- Files, editfiles, copy, tidy, links, etc
  - Now just “files”
- Disks, misc_mounts
  - Now just “volumes”
- Processes and shellcommands
  - Processes and commands

Stricter syntax

- Allows consistent powerful parser
  - Semi-colons and quotes – get used to it!

```
body attribute_type template(params)

{
sub_attr1 => “value 1”;
sub_attr2 => “value 2”;

class::

sub_attribute3 => “value 3”;
}
```

New components

- cf-agent (cfagent)
- cf-serverd (cfservd)
- cf-monitord (cfenvd)
- cf-execd (cfexecd)
- cf-promises – promise checker
- cf-know – integrated knowledge management

Major changes 1

- Control parameters separated from variable definition
  - Control parameters represent attributes to implicit promises
  - Variables are explicit user-promises

```
body control agent
{
ifelapsed => “120”;
}

bundle agent mybundle

{
vars:
          “name” string => “value .....”
}
```

Major changes 2

- Variables now have types:
  - string
  - int
  - real
  - slist
  - ilist
  - rlist
- No more lists by character separators

Major changes 3

- Lists:

```
vars:
 “scalar”  string =>  “scalar values”;

 “listvar”  slist => { “one”, “two”, @(otherlist) };

 “otherlist” slist => { “three”, “four” }


commands:

  “/bin/echo”

        args => “hello $(listvar)”;
```

Major changes 4

- No actionsequence required
  - bundlesequence instead

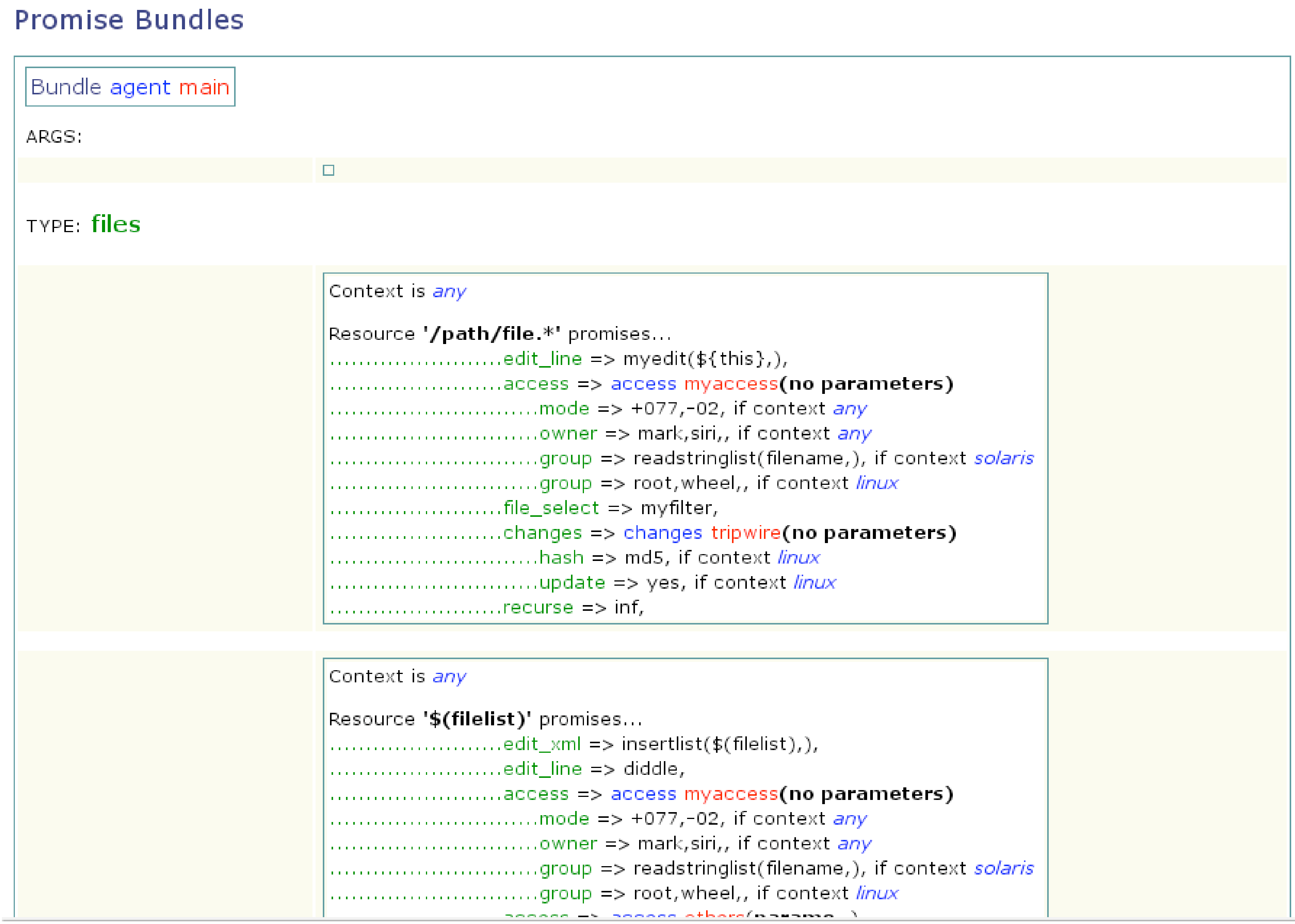
```
body control common

{
bundlesequence => {
                    “bundle1”,
                    bundle2(param)
                    };
}
```

The simplest configuration

```
body common control
{
bundlesequence  => { "test" };
}

bundle agent test
{
reports:

 Yr2008::
    "Hello world";
}
```

2. The Promise Model

Promise oriented configuration



Promise Bundles

Bundle agent main

ARGS:

☐

TYPE: files

Context is *any*

Resource **'/path/file.*'** promises...
.......................edit_line => myedit(${this},),
.......................access => access myaccess**(no parameters)**
............................mode => +077,-02, if context *any*
............................owner => mark,siri,, if context *any*
............................group => readstringlist(filename,), if context *solaris*
............................group => root,wheel,, if context *linux*
.......................file_select => myfilter,
.......................changes => changes tripwire**(no parameters)**
............................hash => md5, if context *linux*
............................update => yes, if context *linux*
.......................recurse => inf,

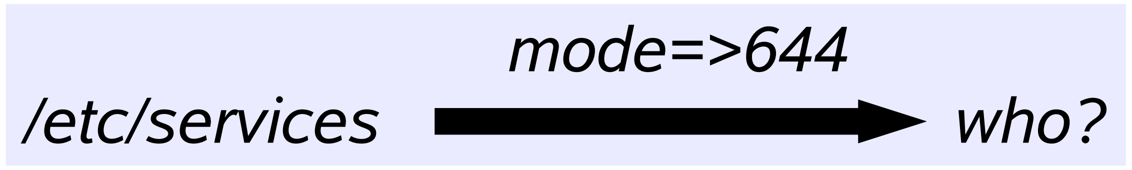
Context is *any*

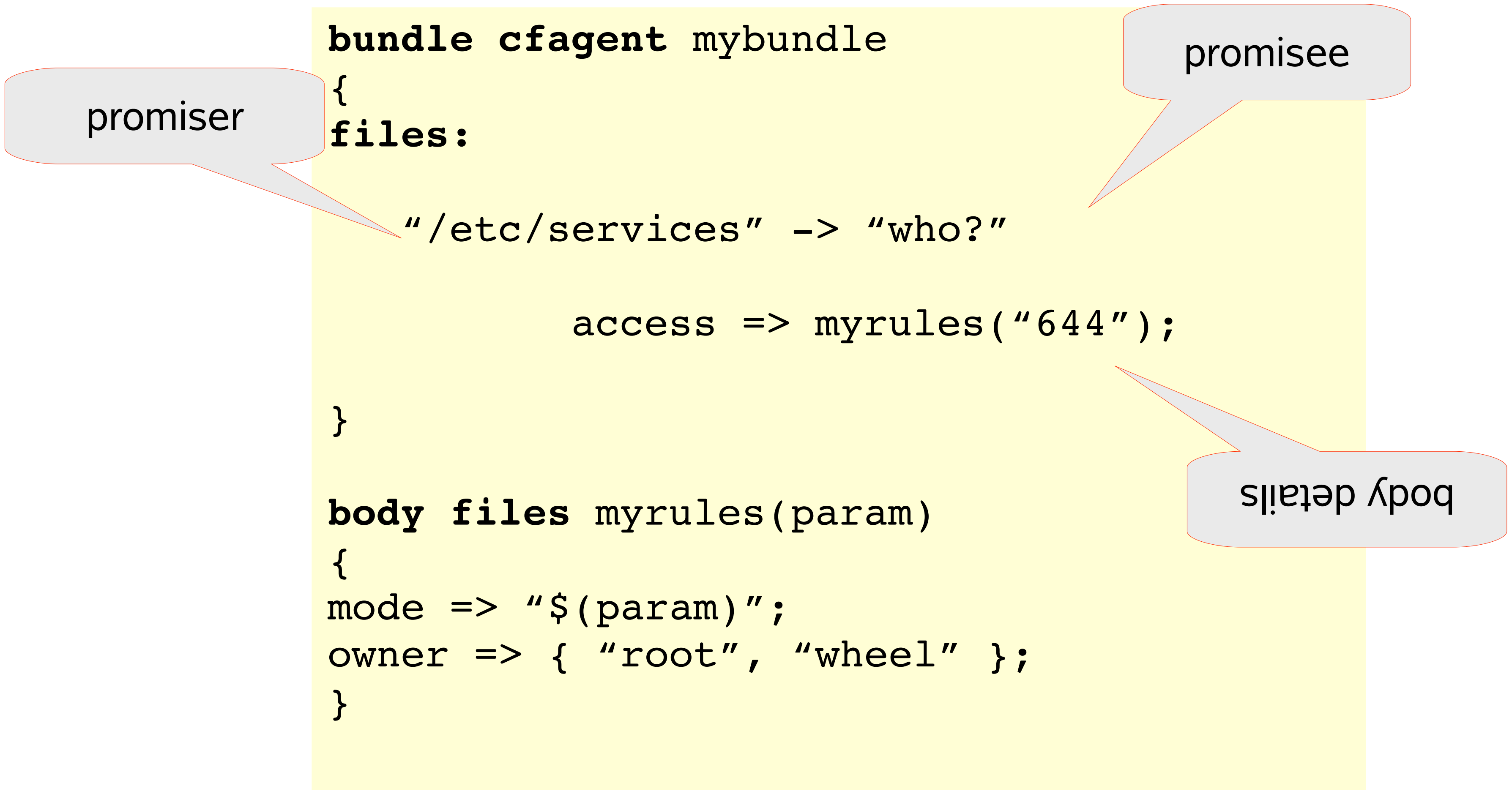
Resource **'$(filelist)'** promises...
.......................edit_xml => insertlist($(filelist),),
.......................edit_line => diddle,
.......................access => access myaccess**(no parameters)**
............................mode => +077,-02, if context *any*
............................owner => mark,siri,, if context *any*
............................group => readstringlist(filename,), if context *solaris*
............................group => root,wheel,, if context *linux*
.......................access => access others**(params..)**

Why promises?

- Promises describe the best we can achieve of intended state
- Every promise corresponds to a convergent maintenance method to “keep” the promise
- This is a simple model that fits all cases

How does it look in cfengine 3?





```
bundle cfagent mybundle
{
files:

   "/etc/services" -> "who?"

           access => myrules("644");

}

body files myrules(param)
{
mode => "$(param)";
owner => { "root", "wheel" };
}
```

How does it look in cfengine?

```
control:
actionsequence = (
                  files
                  tidy

shellcommands
                  )
files:
   /etc/services
          mode=0644
          owner=root,wheel
          action=fixall

tidy:

  /tmp pattern=* age=7

shellcommands:

  "/usr/bin/updatedb"
```

```
bundle cfagent mybundle
{
files:

   "/etc/services"
      access => myrules;

   "/tmp"
      tidy   => mymask;

executables:

  "/usr/bin/updatedb";
}

body files myrules()
{
mode => "0644";
owner => { "root", "wheel" };
}
```

After a cfengine promise check

- Count up the compliance of the configuration service

```
cfengine:enterprise: Outcome of version (1.0.1):
Promises still kept 92%, Promises repaired 8%,
Promises not kept 0%
```

Input files

- Cfengine looks by default for `promises.cf`
- An input file contains promises
- Some promises are implicit
  - e.g. about a cfengine's behaviour
- Some promises are explicit
  - e.g. about configuration entity properties

Promise syntax

```
type:

 classes::

  “promiser” -> { “promisee1”, “promisee2” },

      attribute_1 => body_or_template1,
      attribute_2 => body_or_template2;
```

Parts

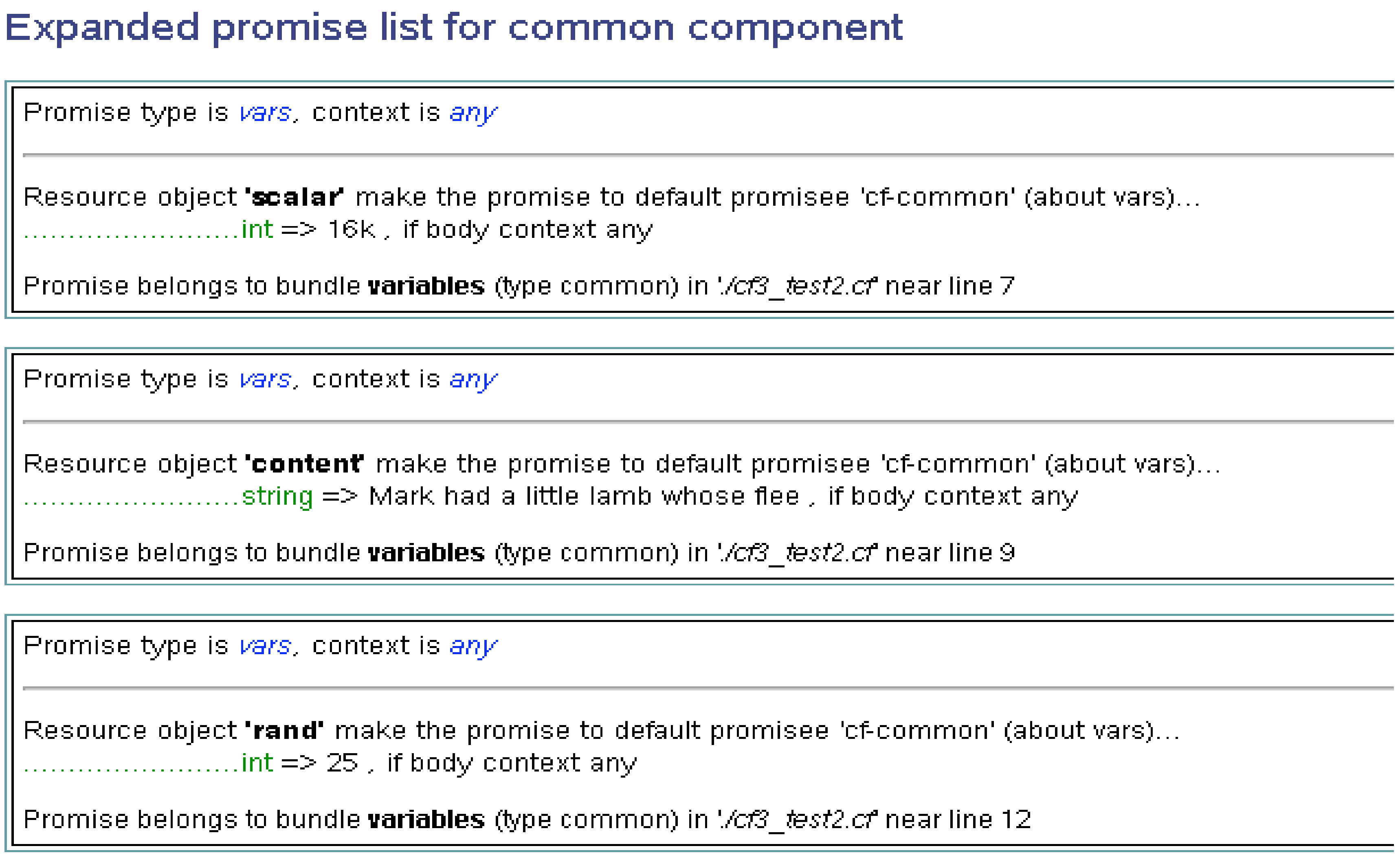
- The type is the type of entity
- Promiser is an independent object or entity of the system with managed properties
- The promisees are those who receive the promise – who verifies the promise?
  - Presently for documentation only
- Attributes represent the subject of the promise
- Bodies – the content of the promise

Variables

```
vars:

 "scalar"  int    => "16k";

 "content" string =>
 readfile("/home/mark/tmp/testfile","33");

 "rand"    int    => randomint(4,88);

 “list” slist => { “one”, “two”, “three”};
```

Compile promises

```
host$ cf-promises -f ./cf3_test2.cf
host$ webbrowser promise_output_common.html
```



Expanded promise list for common component

Promise type is *vars*, context is *any*

Resource object **'scalar'** make the promise to default promisee 'cf-common' (about vars)...
........................int => 16k , if body context any

Promise belongs to bundle **variables** (type common) in *'./cf3_test2.cf'* near line 7

Promise type is *vars*, context is *any*

Resource object **'content'** make the promise to default promisee 'cf-common' (about vars)...
........................string => Mark had a little lamb whose flee , if body context any

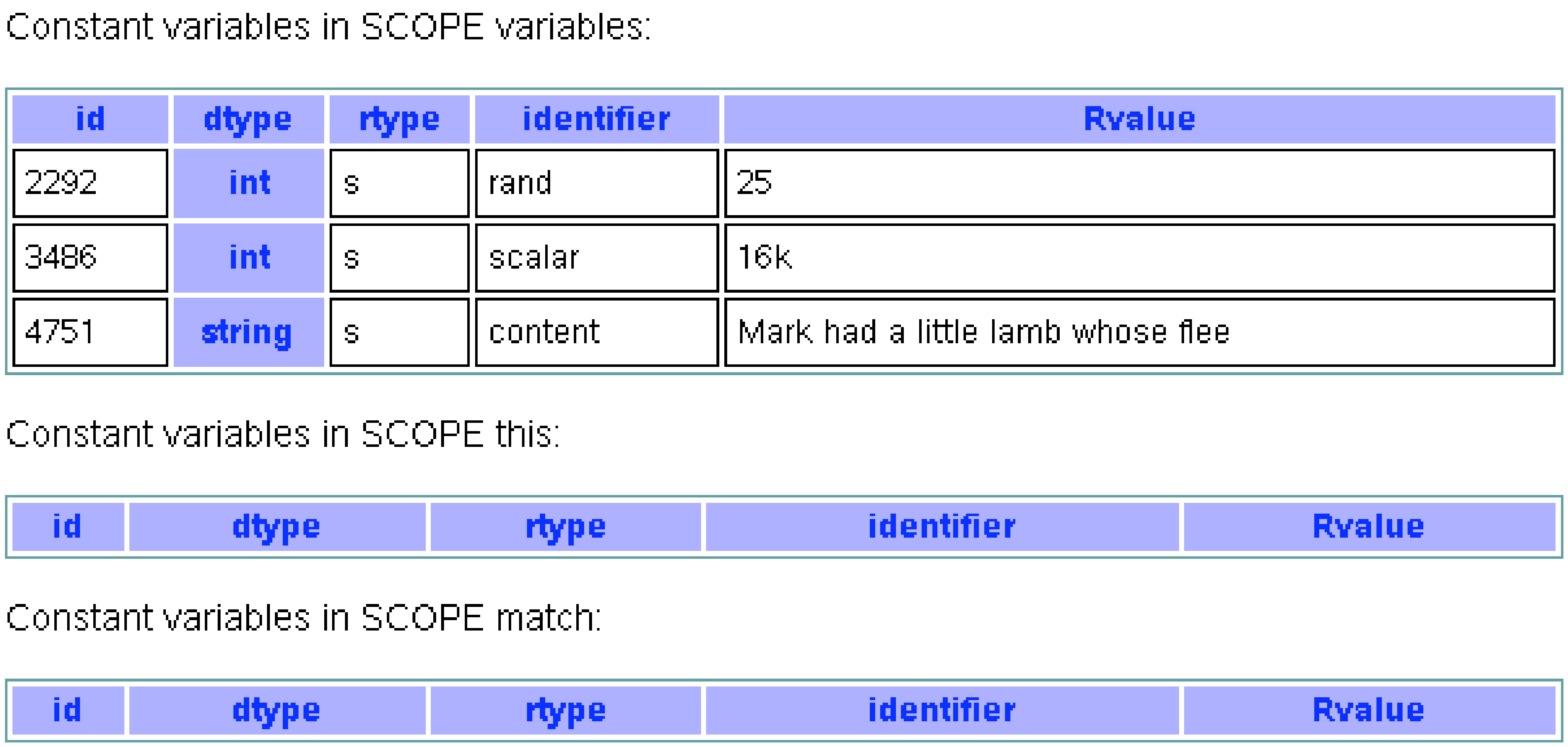
Promise belongs to bundle **variables** (type common) in *'./cf3_test2.cf'* near line 9

Promise type is *vars*, context is *any*

Resource object **'rand'** make the promise to default promisee 'cf-common' (about vars)...
........................int => 25 , if body context any

Promise belongs to bundle **variables** (type common) in *'./cf3_test2.cf'* near line 12

Promise kept...



Constant variables in SCOPE variables:

id	dtype	rtype	identifier	Rvalue
2292	int	s	rand	25
3486	int	s	scalar	16k
4751	string	s	content	Mark had a little lamb whose flee

Constant variables in SCOPE this:

id	dtype	rtype	identifier	Rvalue

Constant variables in SCOPE match:

id	dtype	rtype	identifier	Rvalue

Lists

- Variables representing lists: `@(listname)`
- If we use a scalar reference to a list, this implies iteration over each scalar member

```
vars:
 “hostlist” slist => { “host1”, “host2”, ... };
 “filelist” slist => { “/etc/passwd”, “/etc/shadow” };

files:

 “/backup/$(hostlist)/$(filelist).copied_at_$(cdate)"

   copy_from => c("$(filelist)",”$(hostlist)”);
```

Classes

```
classes:

"myclass" or => { "solaris", "linux" };

"my_dist" dist => { "10","20","40","50" };

"summary" expression => classmatch(“web.*”);
```

Class promises are kept at runtime

```
cf3 ***************************************************************
cf3  BUNDLE test
cf3 **************************************************************
cf3    =========================================================
cf3    classes in bundle test
cf3    =========================================================
cf3
cf3      +  Private classes augmented:
cf3      +       my_dist
cf3      +       my_dist_10
cf3      +       myclass
cf3
cf3      -  Private classes diminished:
cf3
cf3      ?  Public class context:
cf3      ?       any
cf3      ?       Tuesday
cf3      ?       Hr15
cf3      ?       Min35
cf3      ?       Min35_40
cf3      ?       Q3
```

Files

```
files:

 any::

  “/etc/passwd” -> “security group”,

     # We can hide all defaults

     perms => privileged;
```

Files #2

```
files:

 any::

  “/etc/passwd” -> “security group”,

     # We can make parameterized bodies

     perms => priv(“0644”,”root”);
```

Files #3

```
files:

  "/home/mark/tmp/test_create"

      comment => "explain me!",
      rename => disable;

  "/home/mark/tmp/rotateme"

      rename => rotate("4");
```

Files #4

```
files:

 # Make a link of the password file

 "/home/mark/tmp/passwd"

  link_from => dolink("/etc/passwd"),
  move_obstructions => "true";
```

Processes

```
processes:

 "cfservd" -> “operations group”

      process_count => up("cfservd");

 cfservd_out_of_control::

  "cfservd"

   signals => { "stop" , "term" },
   restart_class => "start_cfserv"
```

Commands

```
commands:

  "/bin/sleep 10"
     action  => background;

  "/bin/sleep"
     args => "20",
     action  => background;
```

reports

```
reports:

  activation_class::

   "Hello world, and mum" -> “conscience”,

   comment => “Will this make up for
forgetting her birthday?”;
```

Reprise: promise syntax

```
type:

 classes::

  “promiser” -> { “promisee1”, “promisee2” },

      attribute_1 => body_or_template1,
      attribute_2 => body_or_template2;
```

3. Bundles and bodies

Containers - bundles

- A system is composed of many promises
- We need a way to bundle promises together in meaningful ways
  - A bundle is an arbitrary collection of promises
  - A bundle has a name
  - Represents an “aspect” of system behaviour
  - A bundle can be parameterized for re-use
  - Allows config for different componets in same file
  - Bundles are ordered

Promise bundles

- Division into typed bundles of promises

```
bundle agent basics(param)
{
files:

 classes::

   "/file/name" attribute => value();
}

bundle server access_stuff
{
access:
   "/file" accesslist => values();
}
```

Promise bodies

- Like the body of a document, or contract
- Contains the details of a promise
  - Every constraint has the form `lval => rval`
  - Some rvals can be bodies with multiple values
  - If an lval takes a body, you can make any number of templates of the matching type

Bodies - parameter templates

- Collect together defaults and hide parameters
  - Improves readability, reusability, type-safety

```
body attribute_type template(params)

{
sub_attr1 => “value 1”;
sub_attr2 => “value 2”;

class::

sub_attribute3 => “value 3”;
}
```

Example

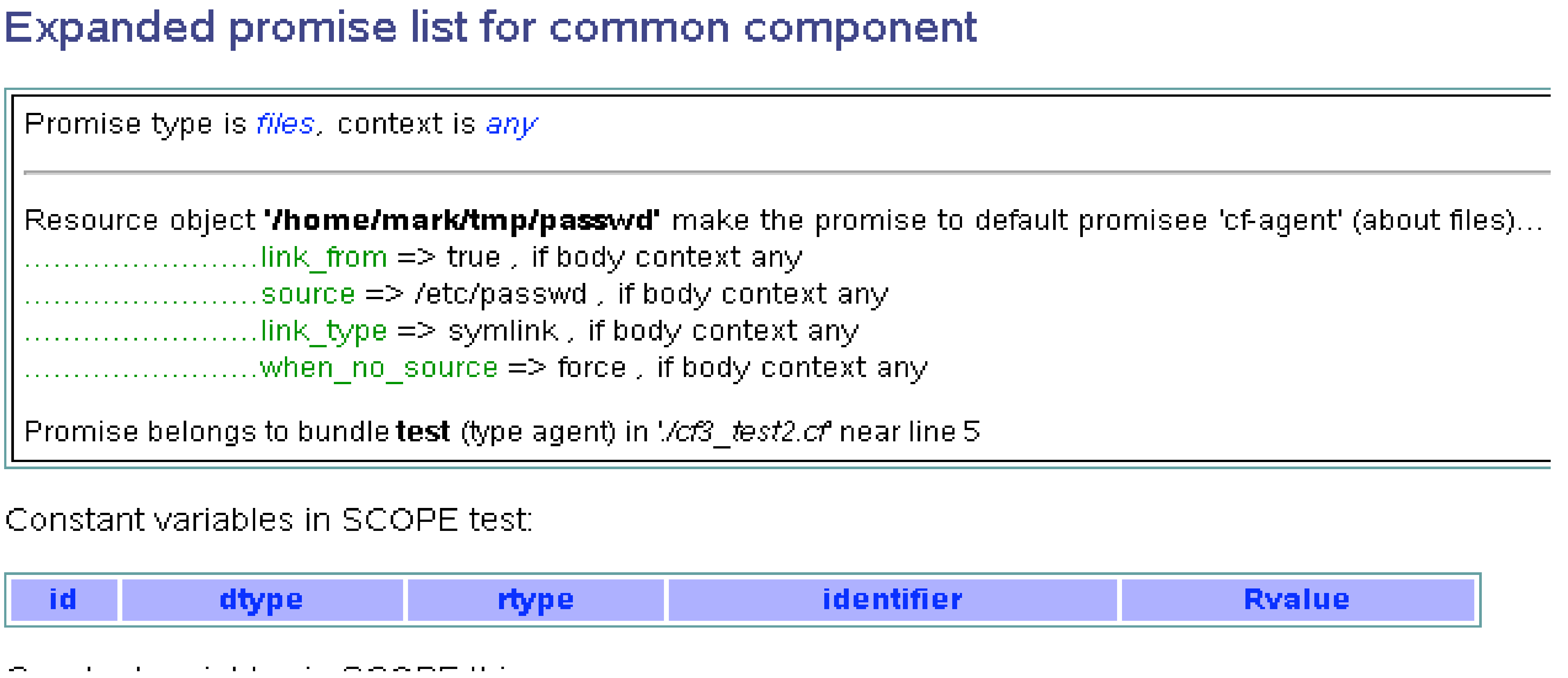
```
body link_from dolink(tofile)

{
# We can pass parameters
source              => "$(tofile)";

# Or set constants – to hide detail

link_type           => "symlink";
when_no_source      => "force";
}
```

Complete example

```
bundle agent test
{
files:
 "/home/mark/tmp/passwd"
     link_from => dolink("/etc/passwd");
}
body link_from dolink(tofile)
{
source           => "$(tofile)";
link_type        => "symlink";
when_no_source   => "force";
}
```

Checking output

```
host$ cf-promises -f cf3_test2.cf
host$ webbrowser promise_output_common.html
```



Expanded promise list for common component

Promise type is *files*, context is *any*

Resource object **'/home/mark/tmp/passwd'** make the promise to default promisee 'cf-agent' (about files)...
........................link_from => true , if body context any
........................source => /etc/passwd , if body context any
........................link_type => symlink , if body context any
........................when_no_source => force , if body context any

Promise belongs to bundle **test** (type agent) in *'./cf3_test2.cf'* near line 5

Constant variables in SCOPE test:

id	dtype	rtype	identifier	Rvalue

4. Quickstart configuration

Organizing files

- Various ways to organize the files
  - `WORKDIR/promises.cf` is the default
  - `WORKDIR/failsafe.cf` is the backup in case former does not compile
  - `WORKDIR` is `/var/cfengine` for root
  - `WORKDIR` is `~/.cfagent` for other users

Including files

```
#
# Failsafe file
#

body control common

{
bundlesequence = { "update" };

inputs => { "update.cf" };
}
```

Including same files

```
#
# promises.cf config file
#

body control common

{
bundlesequence = { "update", “other” };

inputs => { "update.cf" , “other.cf” };
}
```

The update bundle

```
bundle agent update
{
files:
   "/var/cfengine/inputs"

       perms => system,

       copy_from =>
        mycopy("/masterfiles/cfengine_inputs",
                "gold_source.domain.tld"),

       depth_search => recurse("inf");
}
```

The update bodies

```
body perms system
{
mode   => "0600";
}

body depth_search recurse(d)
{
depth => "$(d)";
}

body copy_from mycopy(from,server)
{
source          => "$(from)";
servers         => {"$(server)","failover.domain.tld"};
#backup          => "true";
#trustkey        => "true";
encrypt         => "true";
}
```

Create files / dirs

```
files:

  "/home/mark/tmp/test_plain"

       perms => system,
       create => "true";

  "/home/mark/tmp/test_dir/."

       perms => system,  # for dirs +x
       create => "true";
```

Disabling and rotating

```
files:
  "/home/mark/tmp/test_create"
      rename => disable;

 "/home/mark/tmp/rotateme"
      rename => rotate("4");
```

```
body rename disable
{
disable => "true";
disable_suffix => "_blownaway";
}
body rename rotate(level)
{
rotate => "$(level)";
}
```

Hashing

```
"/home/mark/tmp" -> "me"
     changes      => tripwire,
     depth_search => recurse("inf"),
     action       => background;

"/home/mark/LapTop/words" -> "you"
     changes      => tripwire,
     depth_search => recurse("inf");
```

```
body changes tripwire
{
hash           => "md5";
report_changes => "content";
update         => "yes";
}
```

Customizing multiple copies

```
bundle agent virtualhosts
{
vars:
 "vmbase"          string => "/path/vm";
 "src_files"  string => "/path/src";

 "vmlist" slist => { "host1","host2","host3",
                     "host4", "host5", ...  };

files:
 "$(vmbase)/$(vmlist)/config_$(vmlist).vm"

 copy_from => buildvm("$(src_files)/in_$(vmlist)" );
}
```

Check filesystems

```
storage:

  "/usr" volume   => mycheck("10%");
```

```
body volume mycheck(free)
{
check_foreign  => "false";
freespace      => "$(free)";
sensible_size  => "10000";
sensible_count => "2";
}
```

Mount filesystems

```
storage:

 "/home/mark/server_home"

  mount => nfs("myserver","/home/mark");
```

```
body mount nfs(server,source)
{
mount_type => "nfs";
mount_source => "$(source)";
mount_server => "$(server)";
#mount_options => { "rw" };
edit_fstab => "true";
}
```

Server copy -single file example

```
body common control
{
bundlesequence  => { "testbundle"  };
version => "1.2.3";
}

bundle agent testbundle
{
files:

  "/home/mark/tmp/testcopy"

    copy_from    => mycopy("/home/mark/LapTop/words","127.0.0.1"),
    perms        => system,
    depth_search => recurse("inf");
}

body perms system
{
mode  => "0644";
}
```

```
body copy_from mycopy(from,server)
{
source      => "$(from)";
servers     => { "$(server)" };
copy_backup => "true";                #/false/timestamp
purge       => "true";
}
```

```
body server control
{
allowconnects          => { "127.0.0.1" };
allowallconnects       => { "127.0.0.1" };
trustkeysfrom          => { "127.0.0.1" };
}

bundle server access_rules()
{
access:

  "/home/mark/LapTop"

    admit   => { "127.0.0.1" };
}
```

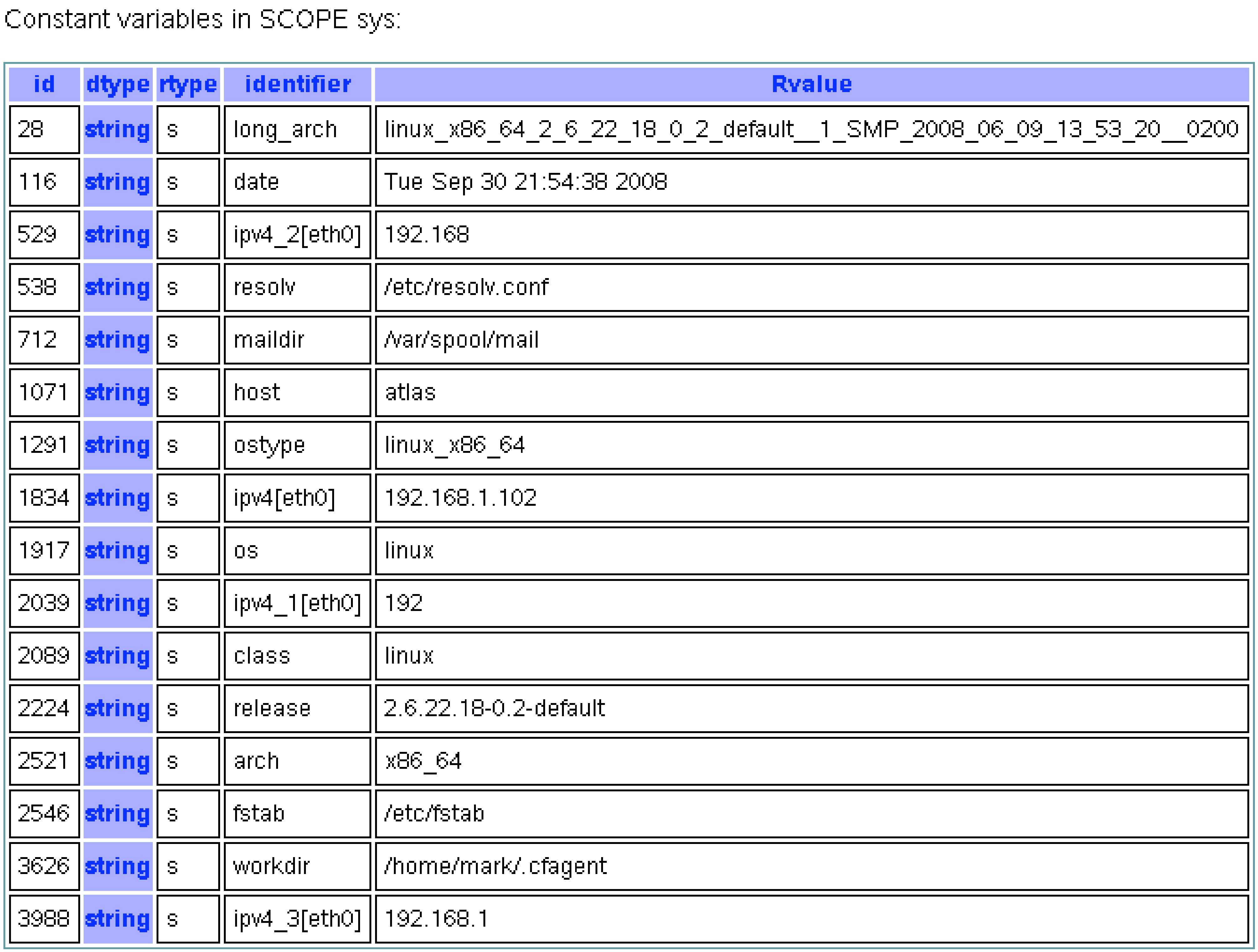
5. Creating libraries

Code simplification or re-use

- Iterate rules over lists (asserted patterns)
- Implicit iteration over regular expression patterns
- Parameterized templates with type protection
- Parameterized bundles
- Use classes to adapt to different contexts
- Create libraries that are ready-adapted to multi-operating systems

Body parts galore

- Make as many bodies as you like
  - As many names as you like
  - As many defaults as you like
- Create a “middleware” language of these templates
- Use the system variables and classes to adapt to the different operating environments
  - edit `$(sys.resolv)`



Constant variables in SCOPE sys:

id	dtype	rtype	identifier	Rvalue
28	string	s	long_arch	linux_x86_64_2_6_22_18_0_2_default__1_SMP_2008_06_09_13_53_20__0200
116	string	s	date	Tue Sep 30 21:54:38 2008
529	string	s	ipv4_2[eth0]	192.168
538	string	s	resolv	/etc/resolv.conf
712	string	s	maildir	/var/spool/mail
1071	string	s	host	atlas
1291	string	s	ostype	linux_x86_64
1834	string	s	ipv4[eth0]	192.168.1.102
1917	string	s	os	linux
2039	string	s	ipv4_1[eth0]	192
2089	string	s	class	linux
2224	string	s	release	2.6.22.18-0.2-default
2521	string	s	arch	x86_64
2546	string	s	fstab	/etc/fstab
3626	string	s	workdir	/home/mark/.cfagent
3988	string	s	ipv4_3[eth0]	192.168.1

library.cf – roll your own

```
body rename disable
{
disable => "true";
disable_suffix => "_blownaway";
}

body rename rotate(level)
{
rotate => "$(level)";
}

body process_select proc_finder(p)
{
process_owner  => { "avahi", "bin" };
command        => "$(p)";
pid            => "100,199";
vsize          => "0,1000";
process_result => "command.
(process_owner|rsize)";
}
```

6. Integrating with cfengine2

Integration with cf 2

- Run cfagent as a program within cf3
- All daemons are 100% backward compatible

```
bundle agent cfengine

{
commands:

   “/var/cfengine/bin/cfagent”;
}
```

7. Patterns and searching

Search and compress files

```
files:

  "/home/mark/tmp/testcopy"

    file_select => pdf_files,
    transformer => "/usr/bin/gzip $(this.promiser)",
    depth_search => recurse("inf");
```

```
body file_select pdf_files
{
leaf_name => { ".*.pdf" , ".*.fdf" };
file_result => "leaf_name";
}
```

Patterns now with PCRE

- Pattern matching is regularized to Perl Compatible Regular Expression

```
files:

   /home/.*/.ssh/authorized_keys
        edit_line => add_key(“$(somekey)”);

   "/home/(.*)/testfile"
       create    => "true",
       edit_line => AppendIfNoLine("key_$(match.1)");
```

Back-references $(match.n)

```
bundle edit_line myedit(parameter)
  {
  replace_patterns:

  # replace shell comments with C comments

   "#(.*)"  # () make backrefs

     replace_with => C_comment,
     select_region => MySection("New section");
  }
```

```
body replace_with C_comment
{
replace_value => "/* $(match.1) */";
occurrences => "all";
}
```

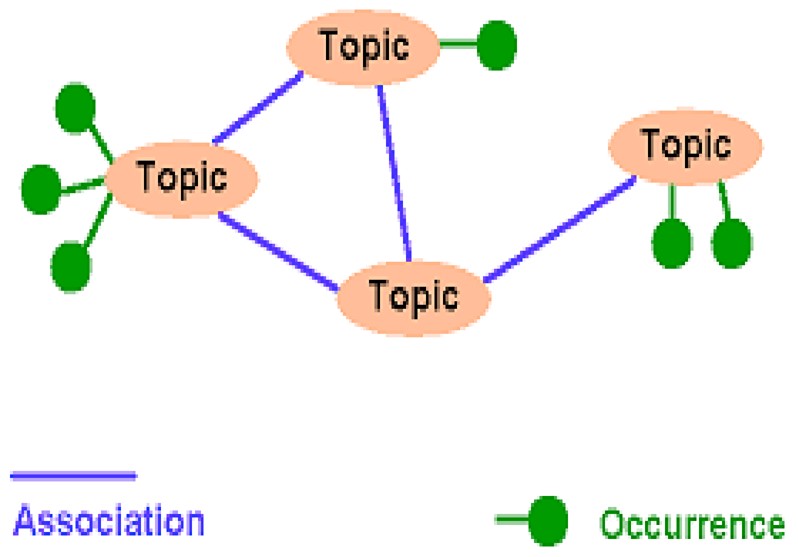
Selecting regions

```
body select_region MySection(x)

{
select_start => "\[$(x)\]";
select_end   => "\[.*\]";
}
```

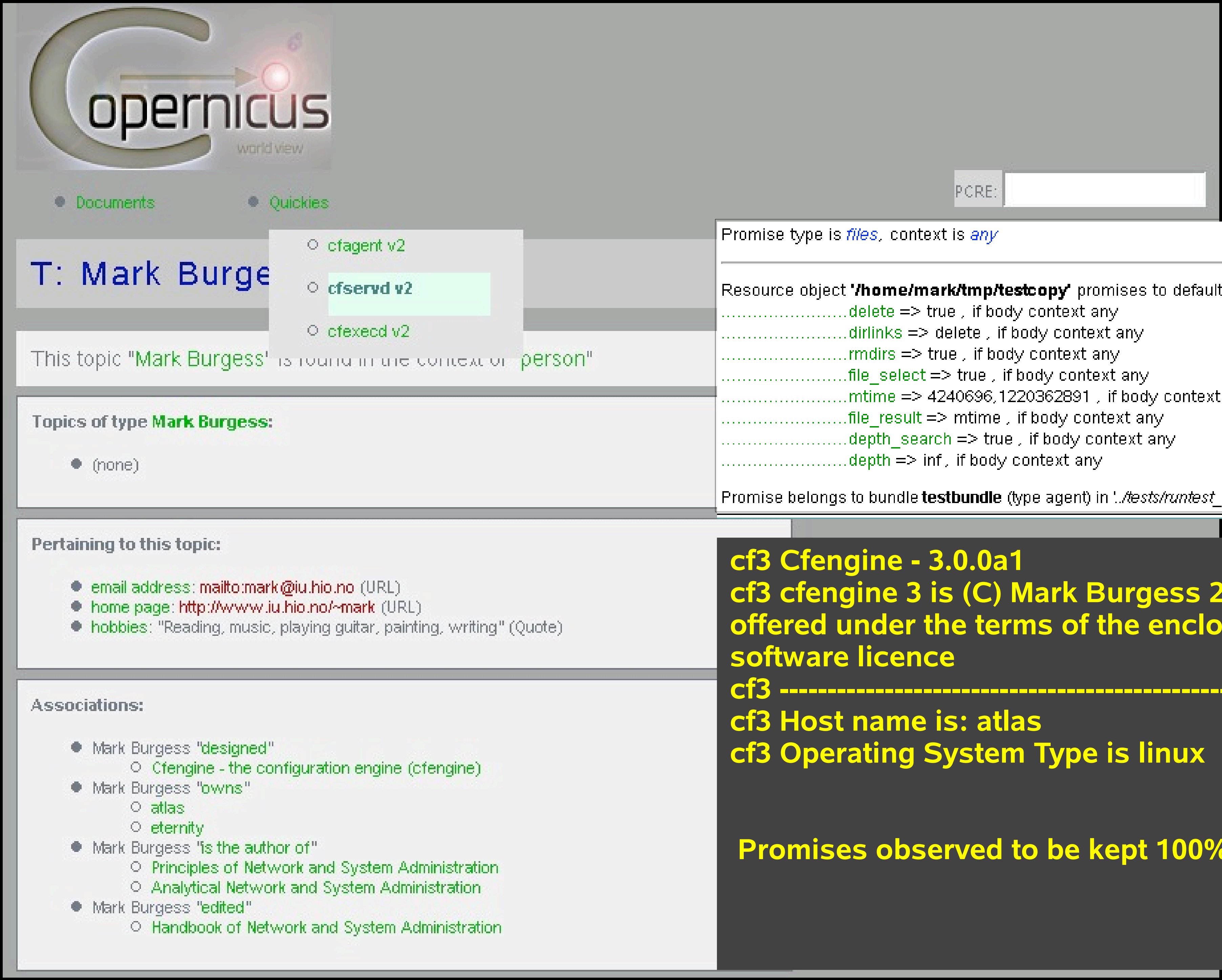
```
[Section Name]

# comments
Stuff

[Next Section]

more stuff
```

Epilogue: back to the future

Topic maps and promises



- Relational understanding of knowledge
  - A semantic web
  - Meaningful use of ontology
- Can embed topic maps in promises
- Promise theory is cfengine 3's central model
  - integration of all management issues



T: Mark Burge

This topic "Mark Burgess" is found in the context of "person"

Topics of type Mark Burgess:

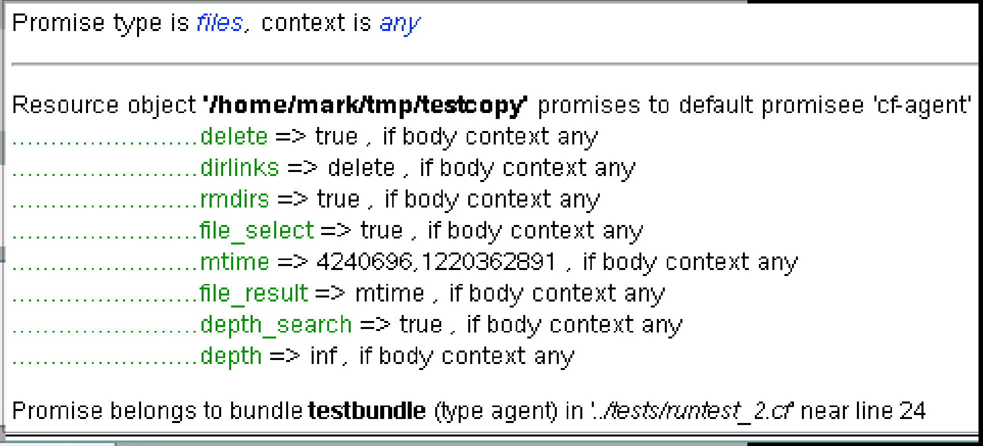
- (none)

Pertaining to this topic:

- email address: mailto:mark@iu.hio.no (URL)
- home page: http://www.iu.hio.no/~mark (URL)
- hobbies: "Reading, music, playing guitar, painting, writing" (Quote)

Associations:

- Mark Burgess "designed"
  - Cfengine - the configuration engine (cfengine)
- Mark Burgess "owns"
  - atlas
  - eternity
- Mark Burgess "is the author of"
  - Principles of Network and System Administration
  - Analytical Network and System Administration
- Mark Burgess "edited"
  - Handbook of Network and System Administration



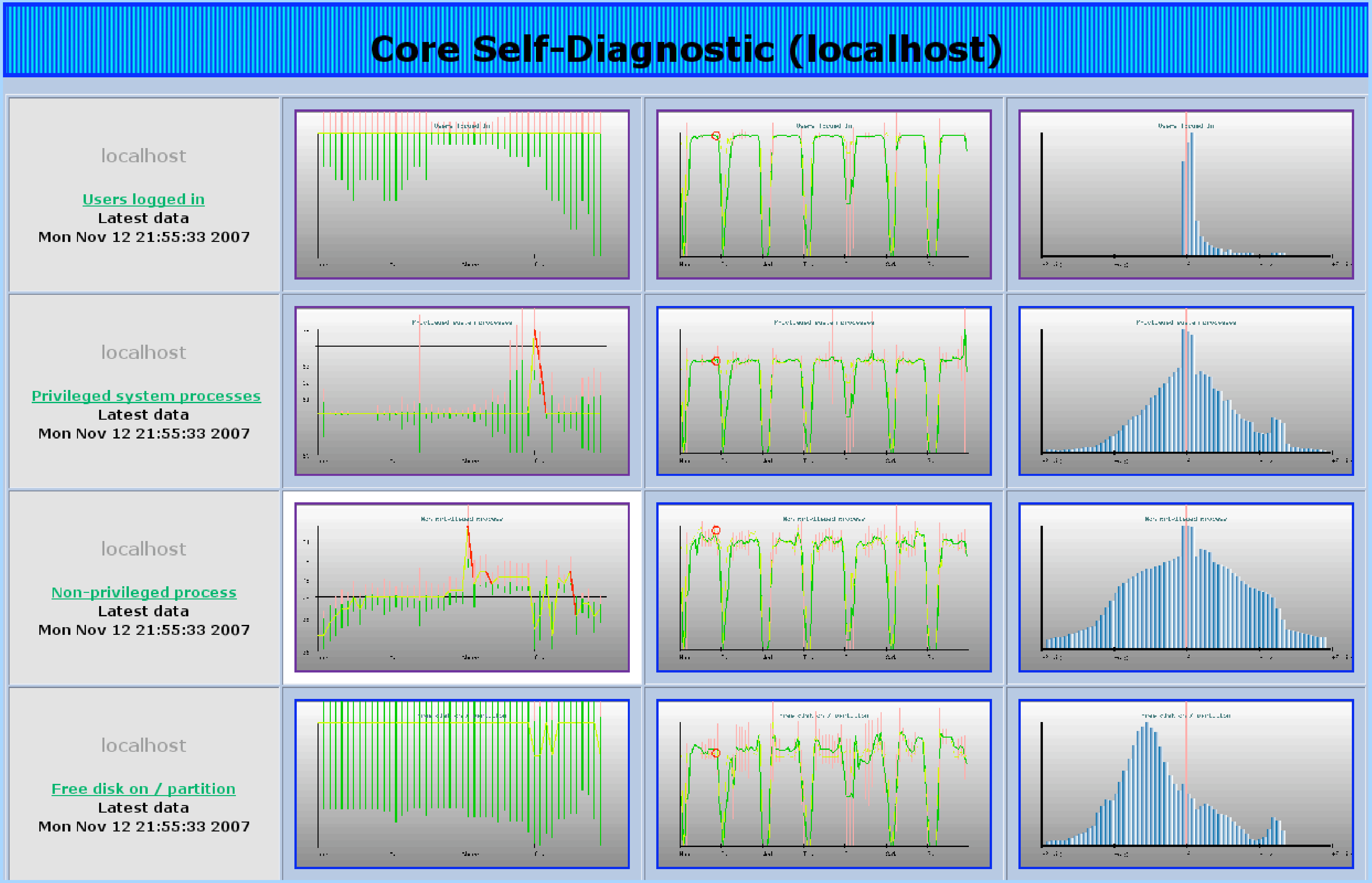
```
Promise type is files, context is any

Resource object '/home/mark/tmp/testcopy' promises to default promisee 'cf-agent'
......................delete => true , if body context any
......................dirlinks => delete , if body context any
......................rmdirs => true , if body context any
......................file_select => true , if body context any
......................mtime => 4240696,1220362891 , if body context any
......................file_result => mtime , if body context any
......................depth_search => true , if body context any
......................depth => inf , if body context any

Promise belongs to bundle testbundle (type agent) in './tests/runtest_2.cf' near line 24
```

```
cf3 Cfengine - 3.0.0a1
cf3 cfengine 3 is (C) Mark Burgess 2008, and
offered under the terms of the enclosed free
software licence
cf3 ---------------------------------------------------
cf3 Host name is: atlas
cf3 Operating System Type is linux


 Promises observed to be kept 100%
```

Afterword...

- Too much for a single tutorial!
  - A more powerful language
  - A more precise language
- The combination of pattern matching and promises
- Code re-use
- Variable contexts, local and global

CfBrain



Core Self-Diagnostic (localhost)

localhost

Users logged in
Latest data
Mon Nov 12 21:55:33 2007

localhost

Privileged system processes
Latest data
Mon Nov 12 21:55:33 2007

localhost

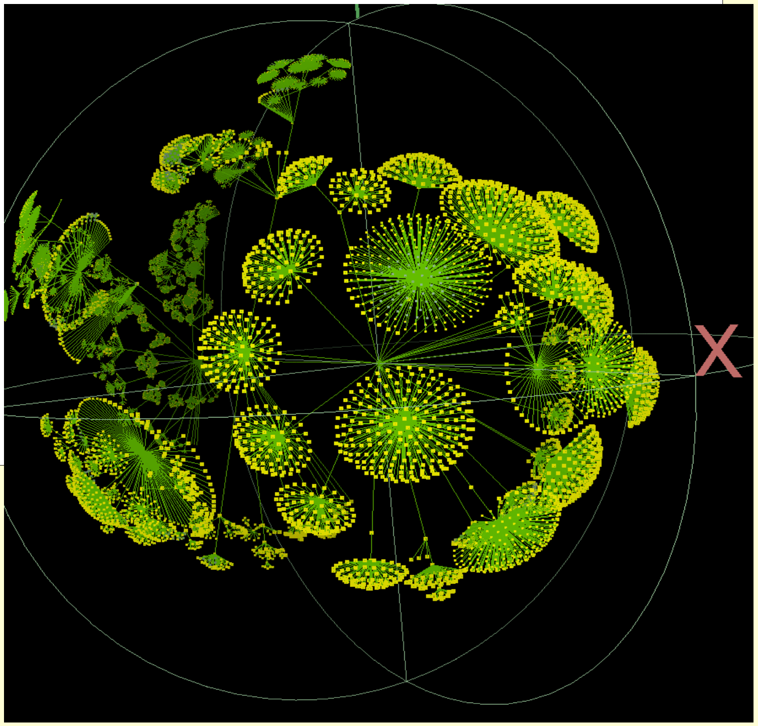
Non-privileged process
Latest data
Mon Nov 12 21:55:33 2007

localhost

Free disk on / partition
Latest data
Mon Nov 12 21:55:33 2007

Summary

- Promises are like "atomic theory"
  - A way of getting to the basics of a problem
  - Need to develop the model of outcomes
- Future – molecular and material computing
- Knowledge mgt



Questions?

visit http://www.cfengine.org
http://www.cfengine.com