T8

Integrating cfengine into Organizational Service Management

Mark Burgess

What's service managament all about – a new paradigm?

- How do I define a service catalogue?
  - Web services, DNS, NFS, DB
  - Also System administration / datacentre
- How do I keep services running without incident? (keep my service promises)
- How do I upgrade and release new versions of my service catalogue?
- **ITIL – IT infrastructure library**

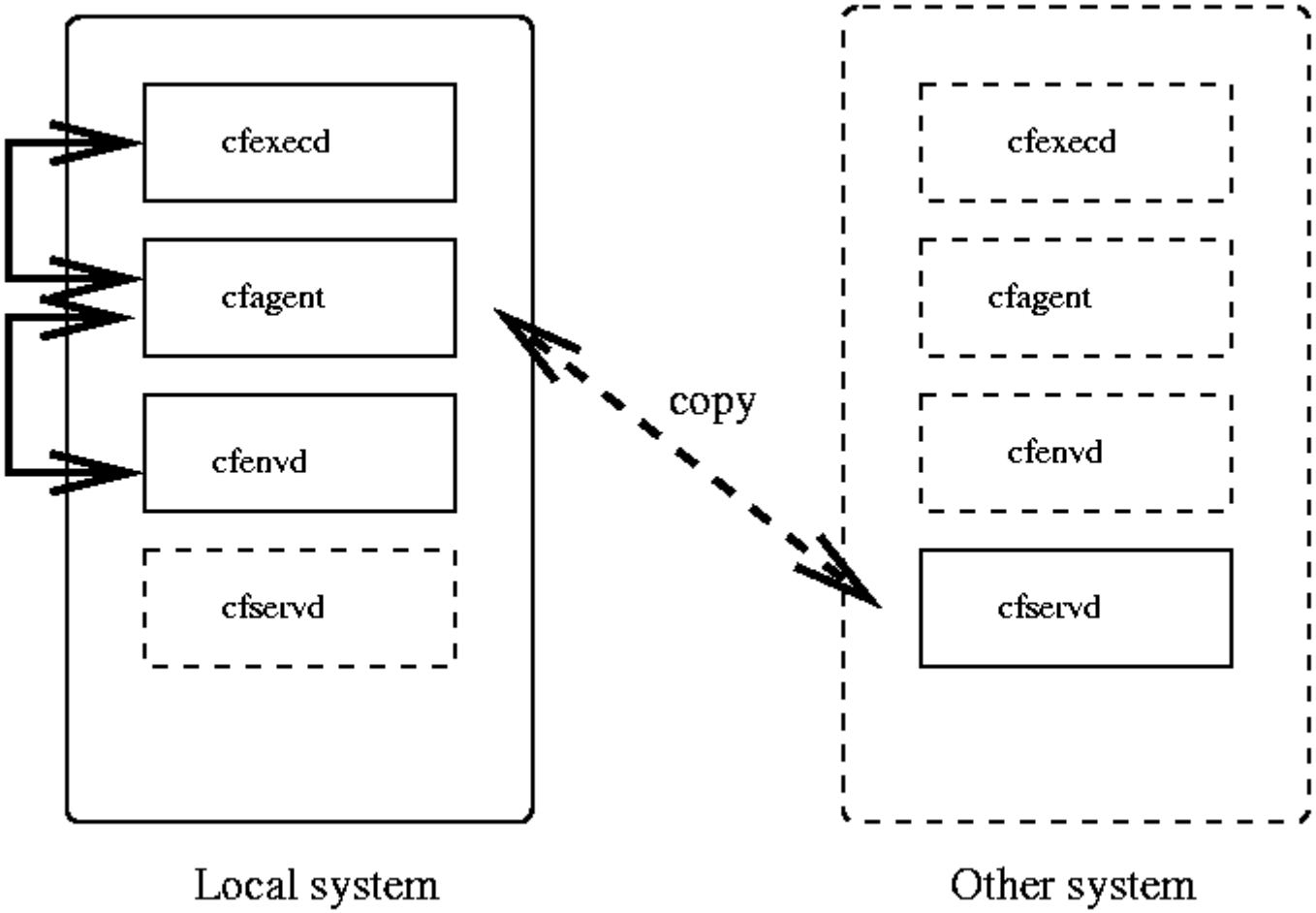
The service paradigm

- Today everything is considered a service
  - Functional rather than hierarchical view
  - Peer to peer
- System administration is a service to the users or the computers
  - How do we make the IT services deliver the results we want?
  - What is the impact on the goals of the organization?

What's cfengine all about?

- How do I manage my servers/workstations?
- How do I get maintenance jobs like backup, security scans, and software updates, done at the right times in the right places?
- How do I ensure that important system files are properly protected against unauthorized access and modification.
- Check compliance with expectations

Components


cfexecd
cfagent
cfenvd
cfservd
copy
cfexecd
cfagent
cfenvd
cfservd
Local system
Other system

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
```

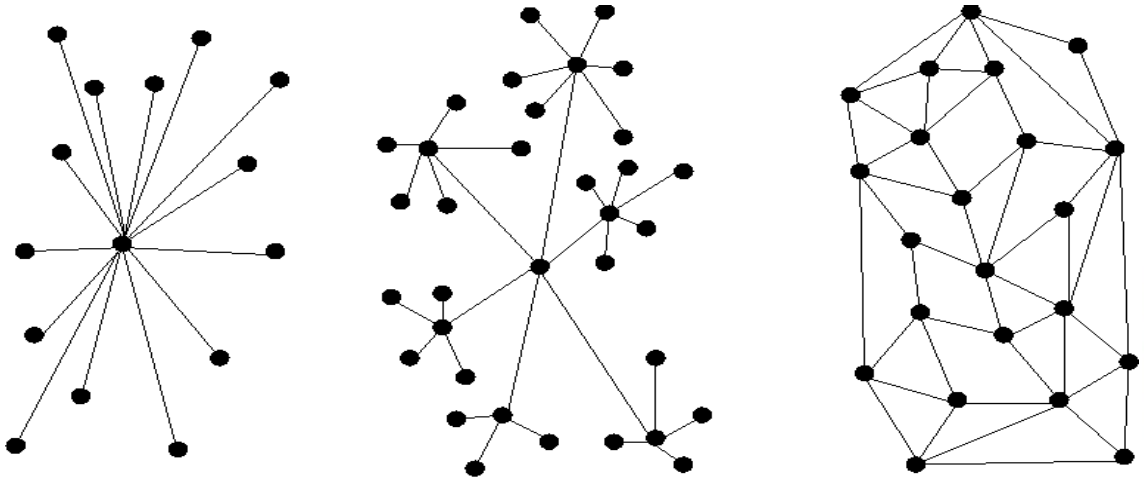
Configuration files

```
control:

 param = ( value )

files:

 /path/file  attributes=value

processes:

  “httpd” restart “/etc/init.d/apache2 start”

# .. etc
```

1. Organization and Management

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”.



(a) (b) (c)

Description by promises

- Voluntary cooperation = service paradigm
  - “Promises”
- A configuration specification is a list of promises to be kept by the resources of the network
- Promises can be grouped into classes of machine or individual machines

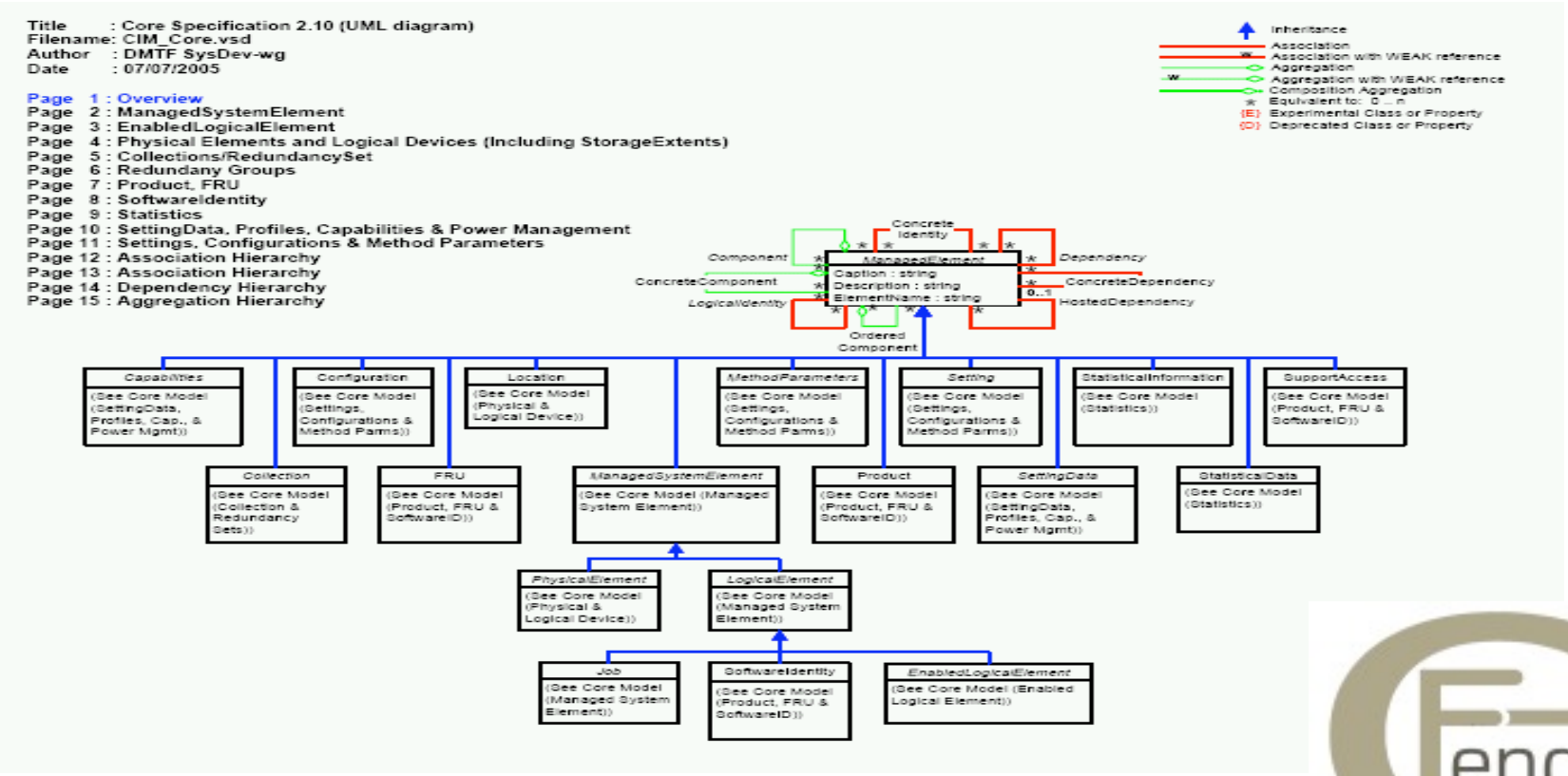
Modelling organizational complexity

- Cfengine is good at capturing complexity
- Use classes and pattern matching
  - Classify systems – solaris, linux, macos
  - Hide complexity automatically
  - Adopt legacy systems by gradual documentation

```
classes:

 newclass = ( oldclass FileExists(/etc/tag) .. )
```

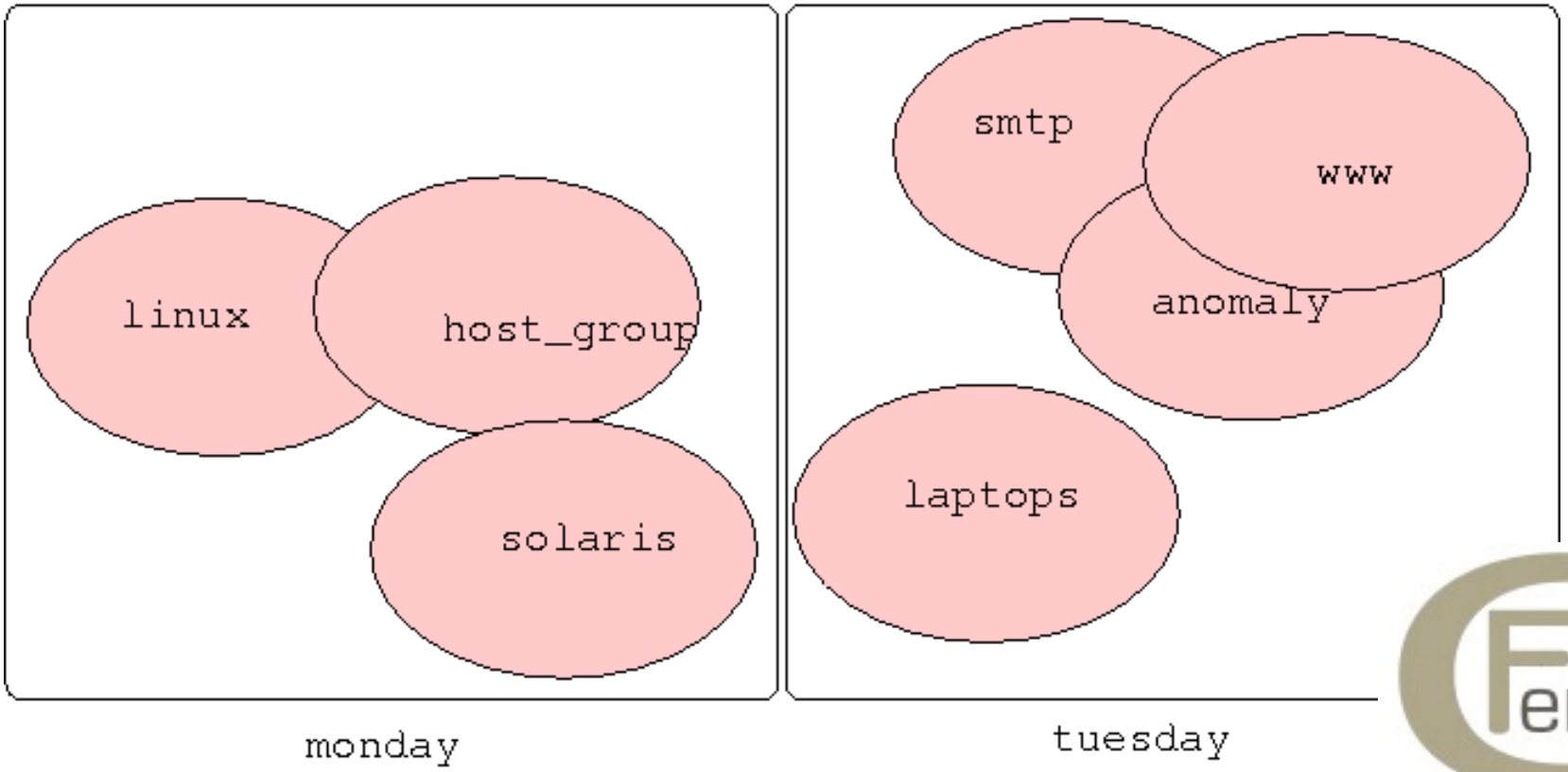
Organizational models

- Most information models for management are hierarchical



Cfengine class model

- The cfengine model is based on patches or overlapping sets



Advantages of the cf model

- Hierarchical organization => tree structure
- Alternatives are mutually exclusive
- In cfengine (like Francis Bacon) we observe and characterize
  - characters can overlap
  - characters can change dynamically
  - context dependence
  - copes with “legacy” complexity

Organization by class

- The obvious set is the set of all machines
  - Why not use it?
  - Everything the same
- We aim for clean separation of concerns
  - Divide and conquer (logical independence)
  - Delegate responsibility (load sharing)
  - Labelling/indexing makes things easier to find

Class belonging

- A cfengine configuration promise belongs to a (sub)set of classes like this:

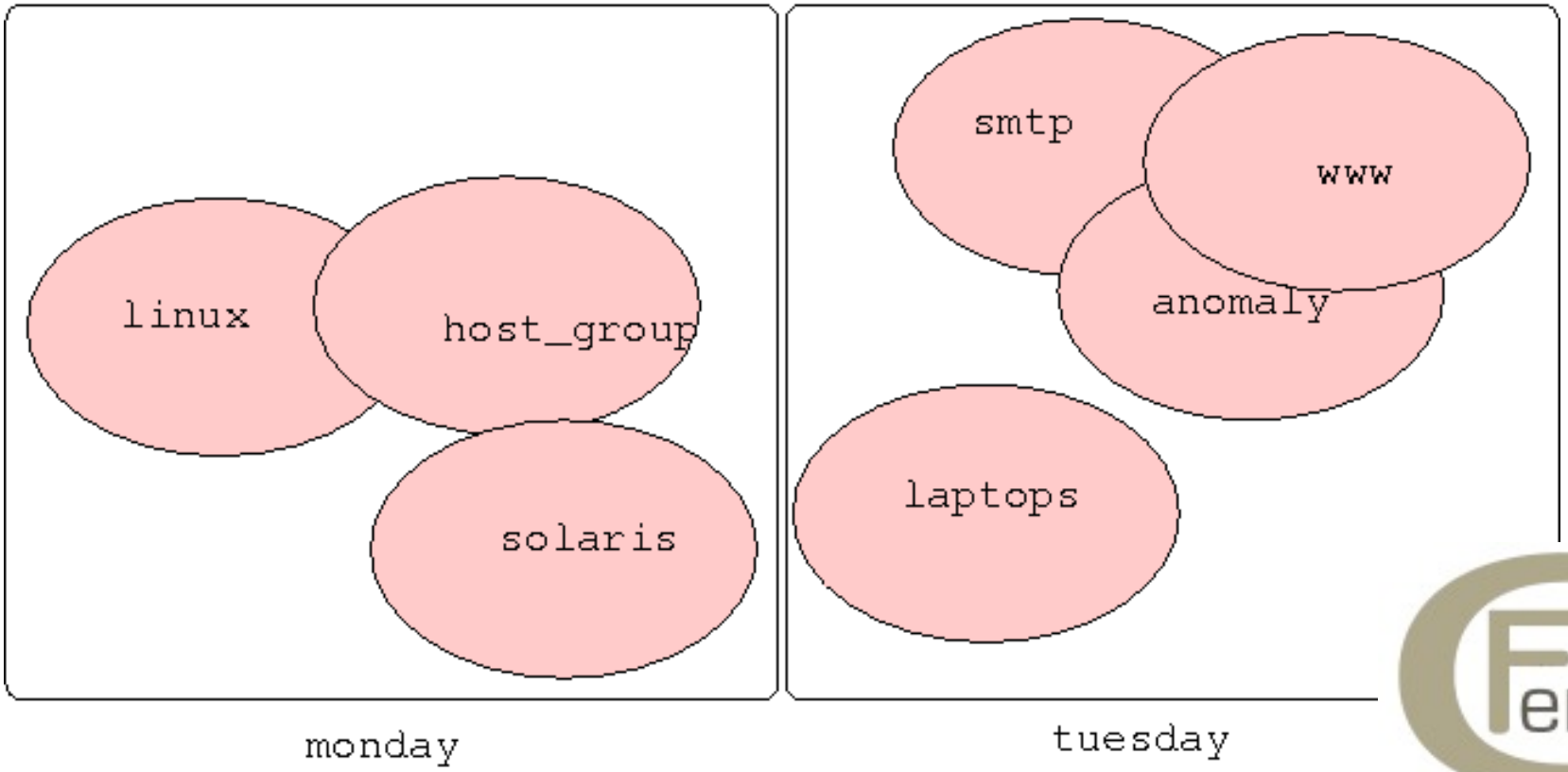
```
type:

  class1|class2&(class3|class4)::

  "object"

    attributes
```

context

promise

Divide up the issues

- We map the issues to files, responsible parties, resources etc.



Identification and Naming

- Consistent naming is the key to knowledge categorization
- Classes allow us make patches within patches
  - the most general possible model

```
classes:

  patch = ( range of classes )
```

Aggregation of similar systems

```
classes:
web = ( web01_dev web01_qa web01_stg
        web01_sd web02_sd web03_sd web04_sd
        web05_sd web06_sd web07_sd web08_sd
        web09_sd web10_sd web02_qa )

web = ( ClassMatch(^web[0-9]+_.*) )
```

Regular language

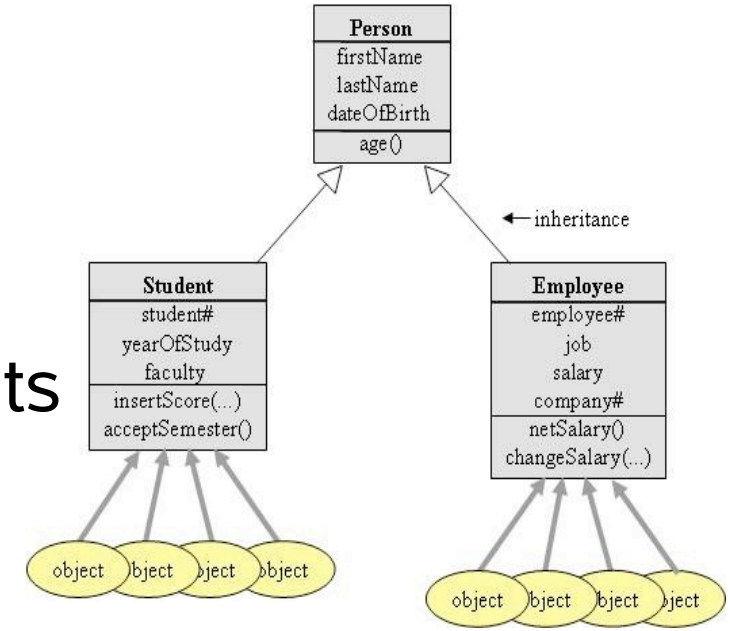
Example – more aggregation

```
classes:


inrange = ( IPRange(128.39.89.10-15) )
CIDR = ( IPRange(128.39.89.10/24) )
compute_nodes = ( HostRange(cpu-,01-32)
```

Enumerated language

Object orientation

- We are taught to think in terms of objects now
- OO thinking is a subset of the overlapping class model
  - It adds some additional short-cuts (overriding)
  - It adds some limitations (mutual exclusion)
  - Some limitations are “fixed” by “aspects”



Object orientation & cfengine

- Cfengine can make containers
  - Class and sub-class relationships
- It does not support overriding
  - Because overriding assumes an ordering of authority
  - This contradicts cfengine's autonomous flat model

```
debian::

debian.ubuntu::
```

Example

- How can we represent a general container relationship?
- We are used to this:

```
class Unix  # Base class

class Solaris extends_or_inherits Unix

class Freebsd overrides Unix
```

Extending (refining) a class

```
# a generic baseclass for operating systems

baseclass::
            cf.basedefaults

baseclass.solaris::
            cf.base-solaris

baseclass.linux::
            cf.base-linux

baseclass.linux.centos::
            cf.centos
```

Extending (refining) a class

- Just a naming convention
- The AND "." operator handles the mutual exclusion of the children
- No rocket science here!

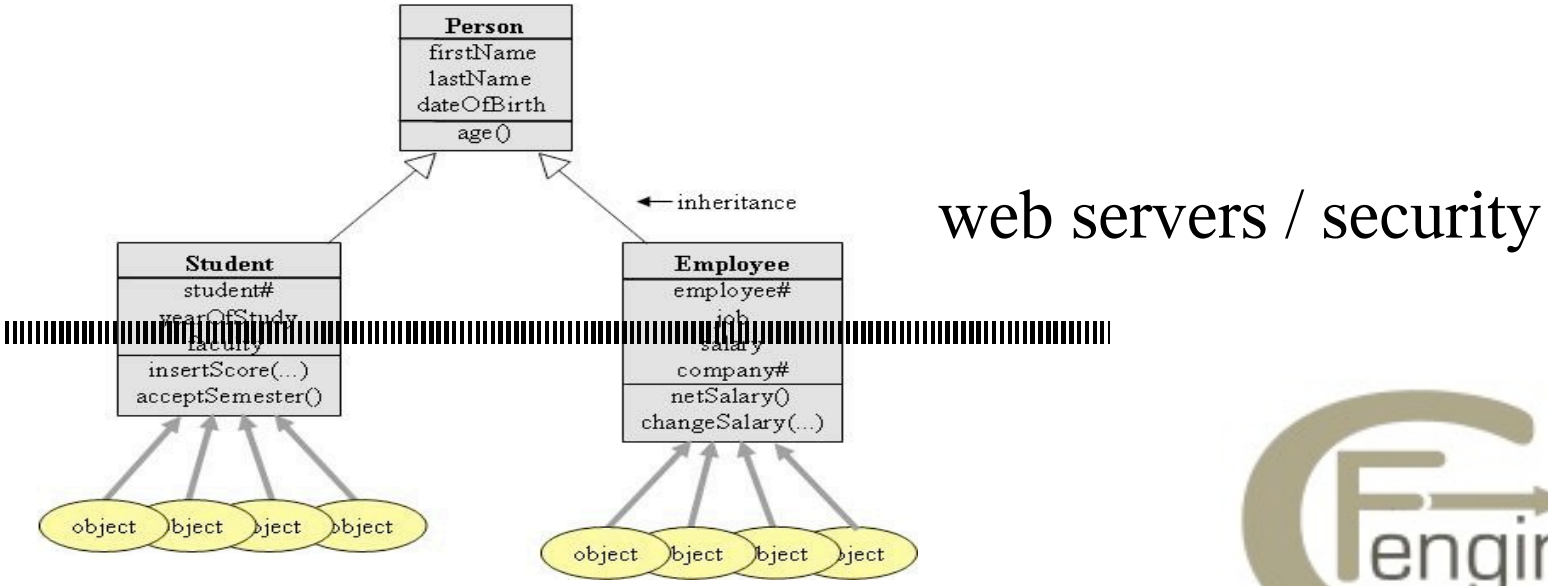
Overriding → voluntary cooperation

- Overriding implies a power to enforce over others – contravenes cfengine model
- A class has to yield in order to be overridden:

```
baseclass::                    # spans all
     "base_policy.cf

baseclass.solaris::            # special child
     "solaris_policy.cf"

baseclass.!solaris::           # default
     "default_policy.cf"
```

Aspect orientation

- One way of fixing OO's 1-dimensional exclusion principle
  - Add new code/rules across all branches of a heriarchical tree



Aspect orientation

- This is a non-issue in cfengine since we can always opt out of a class:

```
baseclass.subclass:: # Start in a special context
 "some specialist rules..."
# Cancel the specificity to span branches
any::
  "This rule cuts across all other classes"
linux::
  "This rule cuts across only the sub-classes of
linux"
```

2. Roles

Role based organization

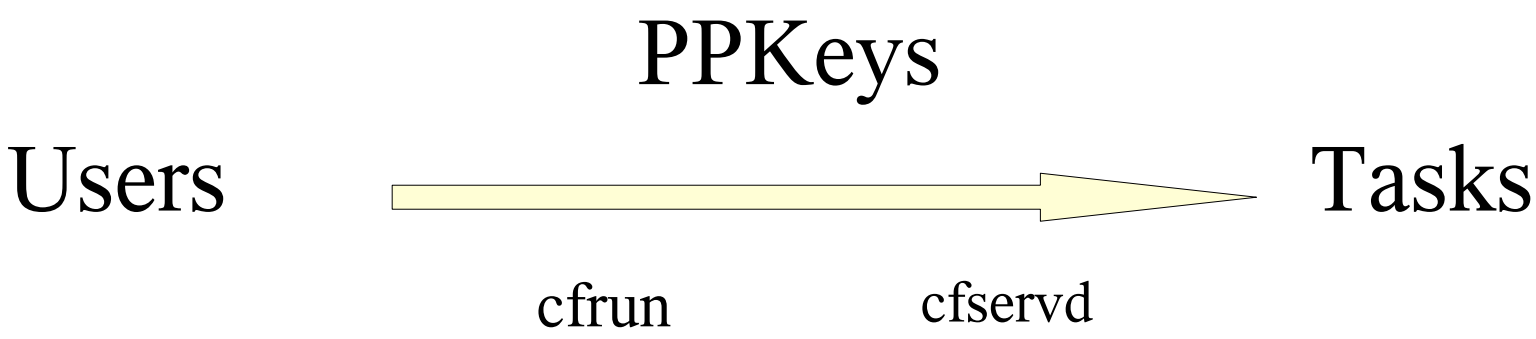
- An aspect of context-sensitive management
  - Who should have what characteristics
  - Who should I accept input from?
- Role based control can be achieved entirely with voluntary cooperation
  - Pull only model gives strong integrity
  - Grant exceptions for certain classes of behaviour

Class based organization

- If we have used classes to label rules then we have already defined roles
  - classes where
  - classes when
  - classes what
- Recall (start of this module) how to delegate by class

Role based access control

- Expressivity limited in cfengine 2
- Would like to say who can ask cfagent to activate which classes of action on demand
  - Broker through cfrun, cfservd



Roles implemented by

- User identified by ppkey and already trusted requests class XXX be activated
- Server grants the right of user to connect and activate cfagent with class roles
  - classes label the action roles
- Future: server grants fine grain control over which classes for which users

Roles requested by cfrun

```
cfrun actionhost -- -- -Dmy_role

cfrun actionhost --  -- linux -Dmy_role
```

Voluntary cooperation
handles the access control

Roles granted by cfservd.conf

```
control:

AllowConnectionsFrom = ( hosts )
AllowUsers = ( users )
cfruncommand = ( /var/cfengine/bin/cfagent)

grant:
 $(cfruncommand) hosts...
```

3. “Best practices”

What is ITIL?

- A heuristic set of “best practices” for business alignment of IT services
- Not based on theory or technology
- Process-oriented approach to human-centric management of IT systems
- A number of books
- Currently at version 3 (also)

ITIL – IT Infrastructure library

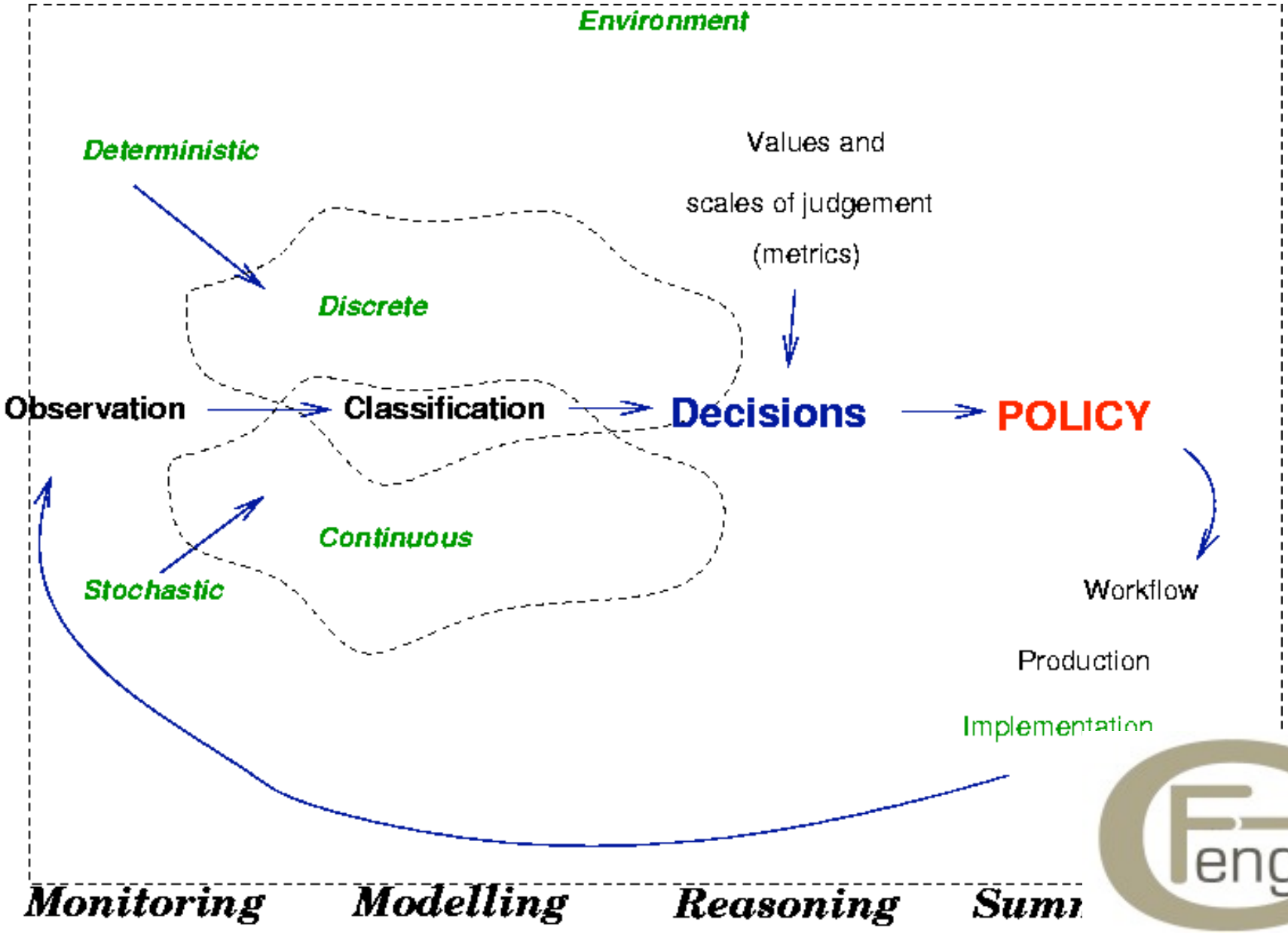
- **Service support / delivery**
  - Incident management
  - Problem management
  - Configuration management
  - Change management
  - Release management
  - Service Level Management

"Best practices!!"

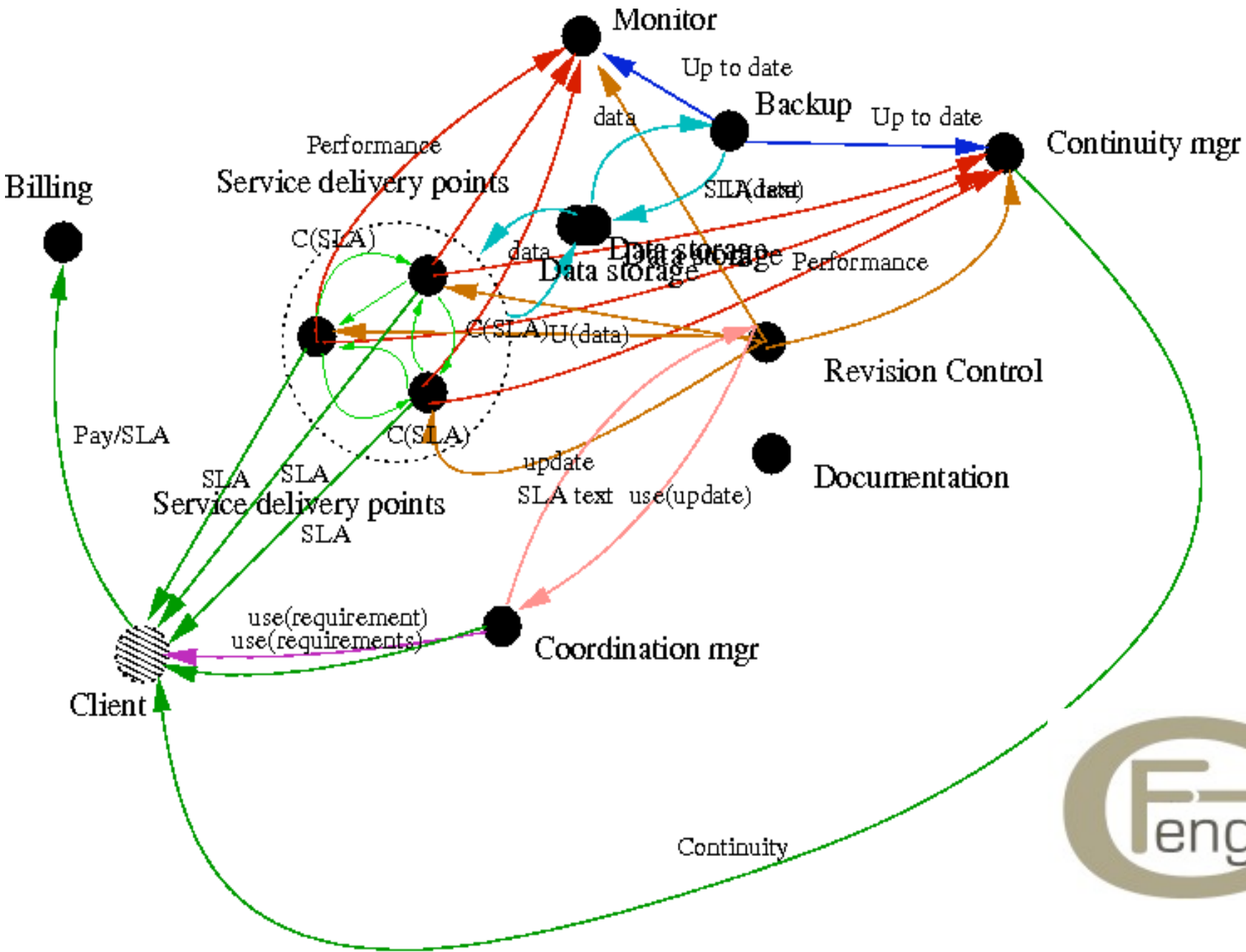
NB - terminology

- As always, secret codes hinder communication...
- In ITIL **CM** means
  - An asset + relationship database...(SQL)
- In server management **CM** means
  - Deployments, maintenance, updates...(c

Improvement cycle



ITIL relies on service promises



Promises

- Business stability is not so much about the details of what we do but what we promise
  - Do we keep promises?
  - How often do we check compliance?
- Agreements are acceptance of bundles of promises between parties
  - Service level agreements (SLA)
  - License agreements
  - etc.

Promising compliance

- With laws
  - Sarbanes-Oxley (SOX)
  - EU 8th company law data directives (EUROSOX)
- With standards
  - ITIL / BS15000/ISO20000
  - COBIT (Control Objectives for Information and related Technology)
  - eTOM (enhanced Telecom Operations Map)
  - BS17799/ISO17799

Cfengine -- low-level promises

- Each cfengine policy is a promise that is made by the host(s) “executing the rule”:

```
files:

 linux.Monday::

     /etc/passwd   mode=644
                   owner=root
                   action=fix
```

- *All linux hosts promise that the permissions of file* `/etc/passwd` *will be checked on Mondays*

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%
```

How does it look in cfengine?

mode=>644

/etc/services → *cfagent*

```
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"
      perms => myrules;

   "/tmp"
      delete   => mymask;

commands:

  "/usr/bin/updatedb";
}

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

Toolsets

- Not many of these standards have tools for "implementing"
  - ITIL kit?
  - SOX in a BOX?

4. Compliance

What does it mean?

- Most standards are loose descriptions of intent, not technical specifications
- It is up to auditors to say what compliance means
- Usually some kind of “due diligence”
- (No one really knows)
  - User management
  - Security hardening
  - Change management

Managing users

```
control:
  actionsequence = (  methods )

classes:
  ok = ( PrepModule("module:getusers","") )

methods:

  # iterate over users

  FixUser("$(user)")
      action=fixuser.cf
```

```
editfiles:

 specialhosts.do:: # Add only special users

  { $(temppasswd) # copy master to this temp file & edit

  # $(listfile) contains a list of users whom we want to
  # have accounts on this subset of machines
  # So get rid of all the accounts that are not in our
  # special list

  DeleteLinesNotStartingFileItems "$(listfile)"
  }
  { $(realpasswd)

  # Add the restricted list to the password file, if the user
  # does not already exist there...

  DeleteLinesStartingFileItems "$(listfile)"
  AppendIfNoSuchLinesFromFile  "$(temppasswd)"
  }
```

If users are removed

- If users are removed from the list, they will not be deleted
- Need to clean up after these
  - Cfengine can help point these out once we have removed them from `/etc/passwd`
  - Pick a method for removing, e.g. delete all users after some well-known system account with editfiles
  - and start again

Checking users are gone

```
control:

SpoolDirectories =
   (
   /var/spool/cron/crontabs
   /var/spool/cron/atjobs
   )

WarnNonOwnerFiles = ( true )

#DeleteNonOwnerFiles
```

Other controls

- deletenonuserfiles
- deletenonownerfiles
- deletenonusermail
- deletenonownermail
- spooldirectories
- warnnonuserfiles
- warnnonownerfiles
- warnnonusermail
- warnnonownermail

SSH key distribution

```
#Master configuration

control:

 actionsequence  = ( methods )
 authuserlist    = ( user1:user2:user3:user4 )

methods:

  CopyKey("$(userlist)")
      action=cf.sshkey
```

```
control:

  actionsequence = ( editfiles  )

  MethodName = ( CopyKey )
  MethodParameters = ( user )

  source = ( "/home/$(user)/id_dsa.pub" )

editfiles:

  { /root/.ssh/authorized_keys

  AutoCreate

  BeginGroupIfNoLineContaining "$(user)"
    InsertFile "$(source)"
  EndGroup
  }
```

General file security

- Permissions and file ownership
  - basis of all computer security
- Continuous monitoring
  - How often do we have to schedule checks to satisfy auditors?
- Fewer changes by hand means greater consistency

5. Some ITIL terminology

ITIL versus cfengine

- Baseline
- Change record
- Config item (CI)
- Config managmt database
- Incident
- Incident response
- Problem

- Initial state
- Audit log
- Config object
- Asset database + promises
- Config error
- Maintenance run
- Root cause of repeated errors

ITIL versus cfengine ctd.

- Availability
  - Hours operational/ agreed service hrs
- Service Level Agreement

- Intermittency
  - Successful attempts/ Total attempts
- Config promises + execution schedule

ITIL versus cfengine ctd.

- Change
- Release

- Config alteration
- New version of config promises implemented on all current hardware

6. Change Management

Change Management

- Cfengine deals with change management
  - Implementation
  - Verification
- Business alignment is a nice goal, but we rarely get close to “optimization”
  - Avoiding incidents is more common
  - Change is the root cause of many incidents
- We can be proactive to avoid inciden using a tool like cfengine

Detecting change in files

```
files:

  /path/to/watch

    checksum=best # md5,sha1
    owner=root,other,mark
    group=root,other,privileged
    action=warnall
```

Change alerts

```
SECURITY ALERT:
Checksum (md5) for /usr/bin/passwd changed!
```

Neighbourhood watch

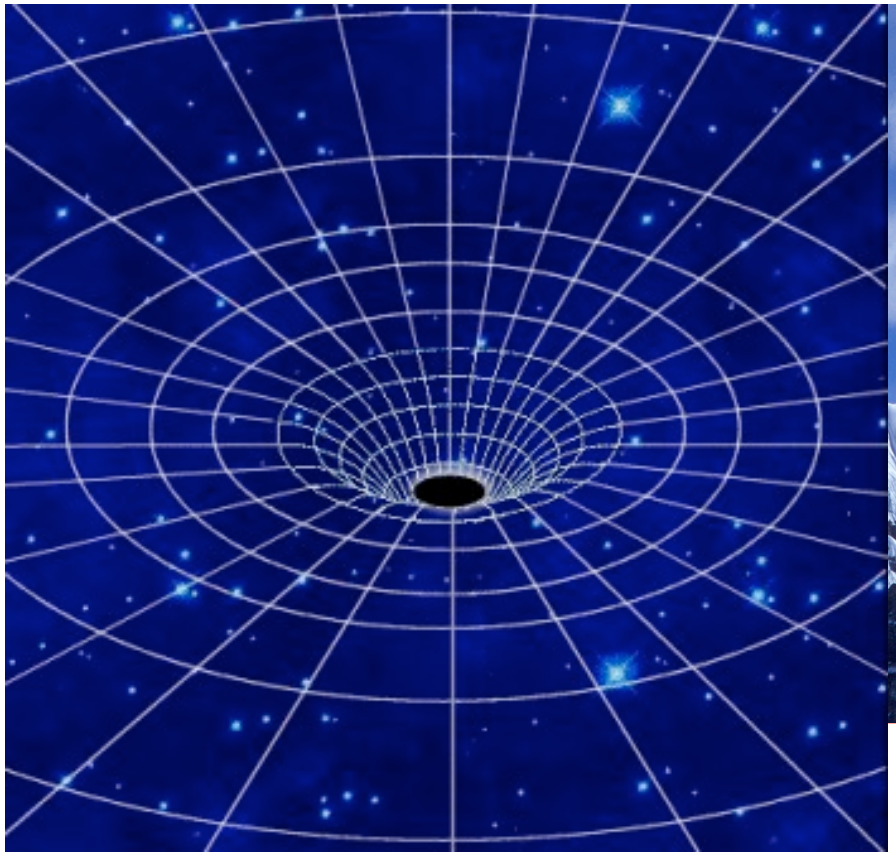
- To foil advanced hackers
  - Make sure there is a distributed backup of the hash database
  - Make neighbours report when they see changes to the whole database
  - Thus a tampering with the change database on one host unleashes a report from the whole neighbourhood

```
# Neighbourhood watch

  control:

  allpeers =
( SelectPartitionNeighbours(/path/cfrun.hosts,#,random,4) )

  copy:

  /var/cfengine/checksum_digests.db
         dest=/safekeep/chkdb_$(this)
         type=checksum
         server=$(allpeers)
         inform=true # warn of copy
         backup=timestamp
         define=tampering

 alert:
   tampering::
     'Digest tampering detected on a peer'
```

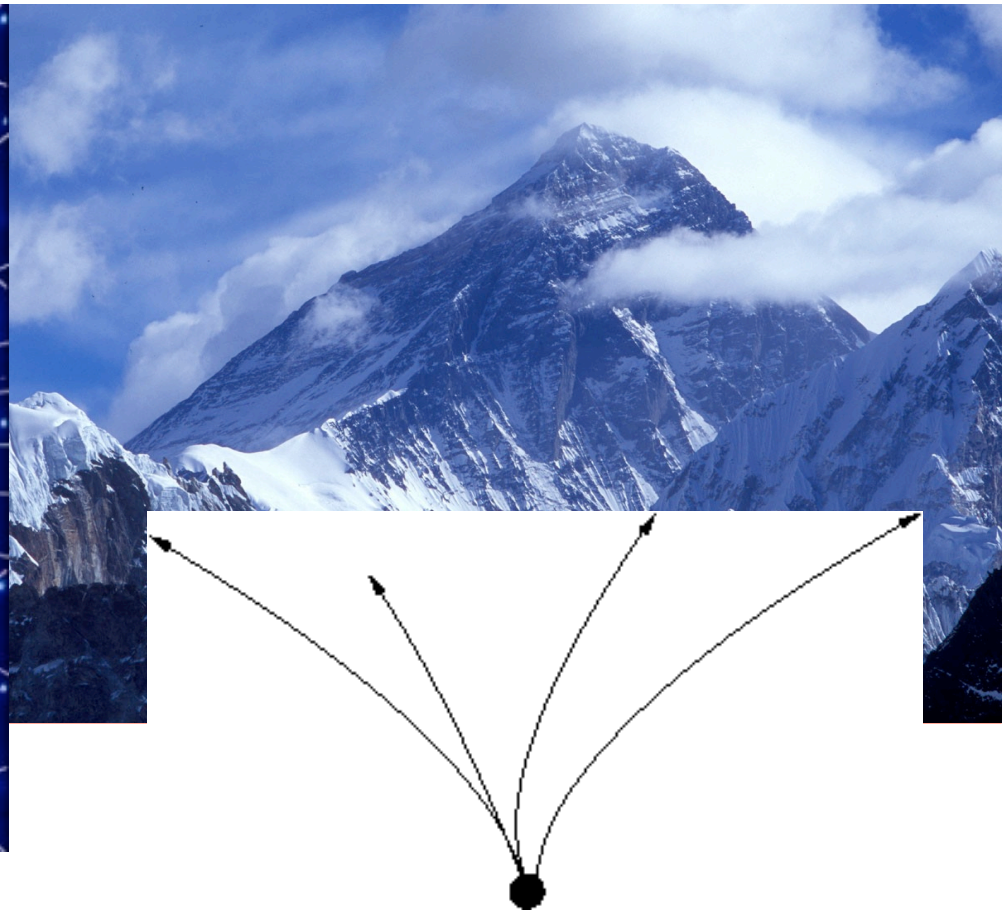
Instigating change

- Hands-free automation makes change reliable and consistent
  - Installing (copy, packages, create files)
  - Deleting (tidy,disable)
  - Editing (editfiles,change permissions etc)
  - Processes (start, stop, monitor)

Two kinds of change



Convergence to end state



Baseline and grow

ITIL “gold server”

- ITIL recommends “baselining” from a gold server
  - Exceeds its technical competence in this!
  - Baseline approach ok for some things
  - Post-install customization better for other things
- Cfengine places no restrictions on approach
- Consider the <u>end result</u> “the release”
  - In cfengine this is equivalent to a sufficiently comprehensive config policy.

Phase 1: substrate

- Start with some kind of standard image to start
  - It is does not necessarily matter what it is as long as it *behaves predictably*
  - e.g. install from known DVD
  - e.g. install from netboot or gold server

Phase 2: customize

- Two approaches:
  - Copy constant “gold” overlays or patches into place from a trusted source
    - Additional packages
    - Special files (config, data etc)
    - Run post-processing scripts
  - Edit system directly with cfengine
    - Documented automatically by cfagent promises
    - Can always customize after that too (phase 2)!

Customize by overlay

```
copy:

/Source/file

        dest=/dest/file
        server=gold_server
```

Customize by overlay template

```
copy:

/Source/file
      dest=/tmp/file
      server=gold_server

editfiles:

{ /dest/file

EmptyEntireFilePlease
InsertFile “/tmp/file”
ExpandVariables              # like m4
}
```

Customize directly

```
editfiles:

{ /dest/file

ReplaceAll “X” With “Y”
AppendIfNoSuchLine “ABC”
}
```

Open or closed world?
Unplanned changes

- ITIL does not want this kind of change!
  - unauthorized
  - pretend it does not happen
- Not wanting and not having are different!
  - Making it a security *incident* does not stop it from happening
- Cfengine assumes this kind of change

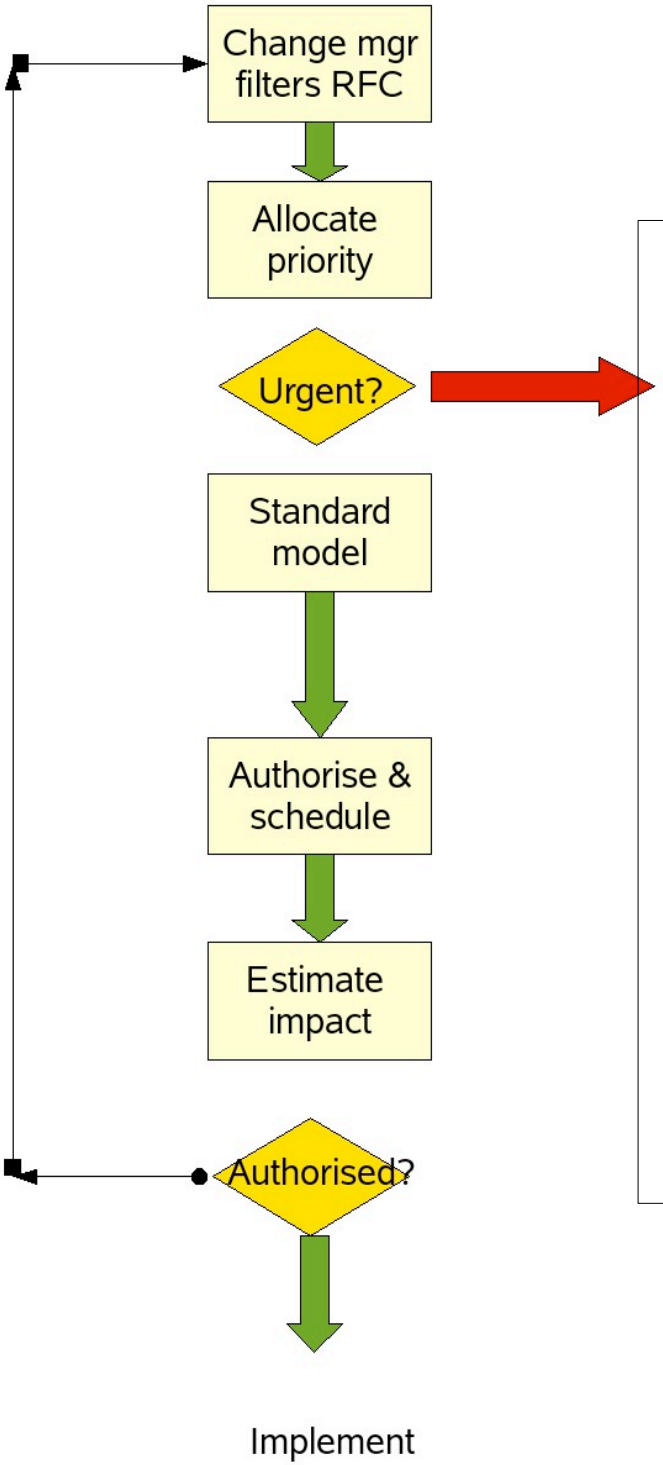
Example: unplanned change

```
files:

  /etc/passwd
        owner=root
        mode=0644        # correct
        checksum=best    # warn
        action=fixall
tidy:

  /tmp
        pattern=*        # garbag
        age=7
```

Think convergence

- Balance between stability and responsiveness
  - **Stability:** hands off, trust convergent maintenance
  - **Responsiveness:** Many small stable changes rather than big planned revisions
- Incorporate convergent thinking and you don't have to worry less about unplanned change
- Forward thinking, not backward defense

Change Process

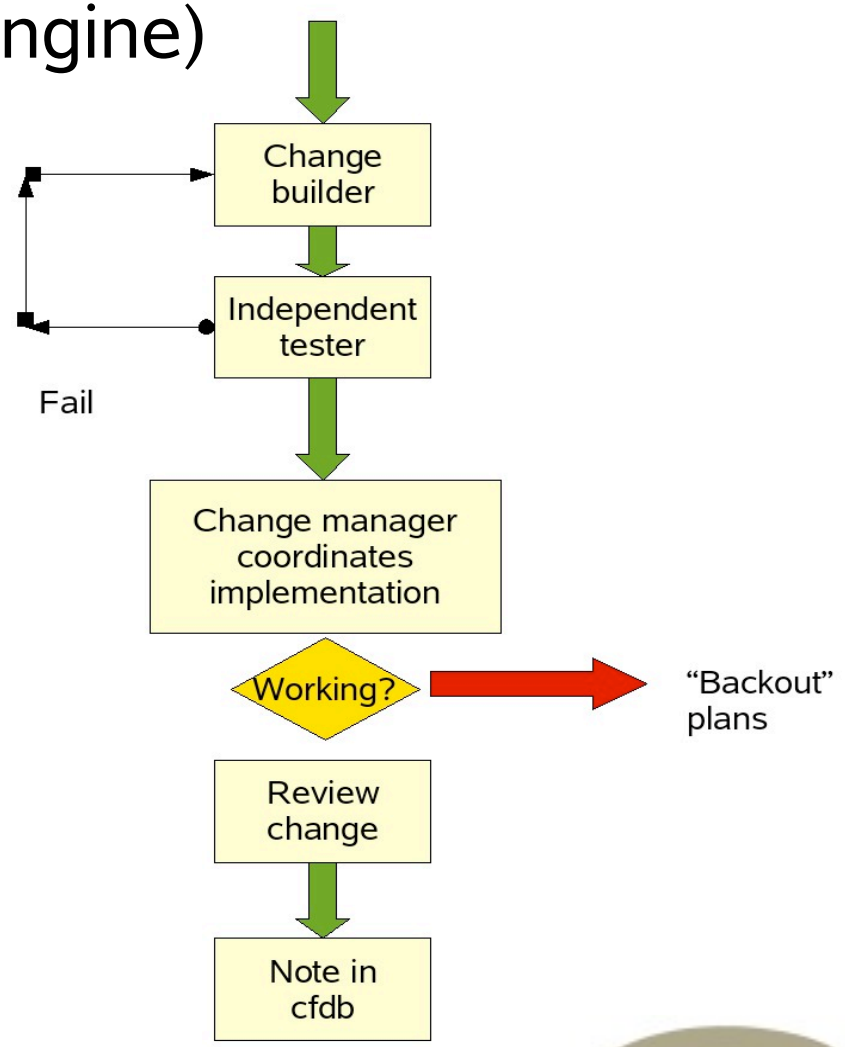


- Begins with a
  - Request for Change (RFC)
- Priority
  - Urgency + Impact
  - U: fire fighting?
  - I: return on investment

Change implementation

(enter cfengine)

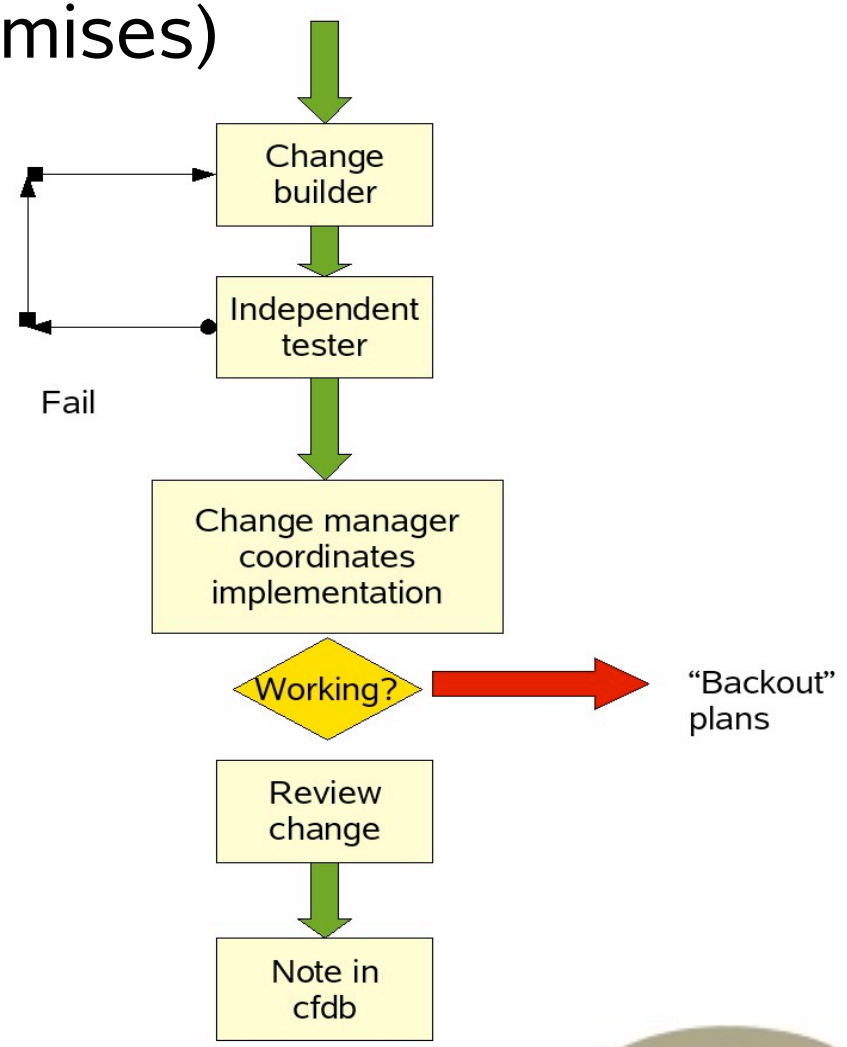
- Implementation plan
- Testing of proposal
- Introduce when/how?
- If at first you don't succeed...
- Observe effect
- Record change



cfengine interpretation

(enter promises)

- Describe promises
- Testing configuration
- Determine classes
- Go back to previous version or repair
- Observe + record
  - Redundant
  - Already documented as convergent promises

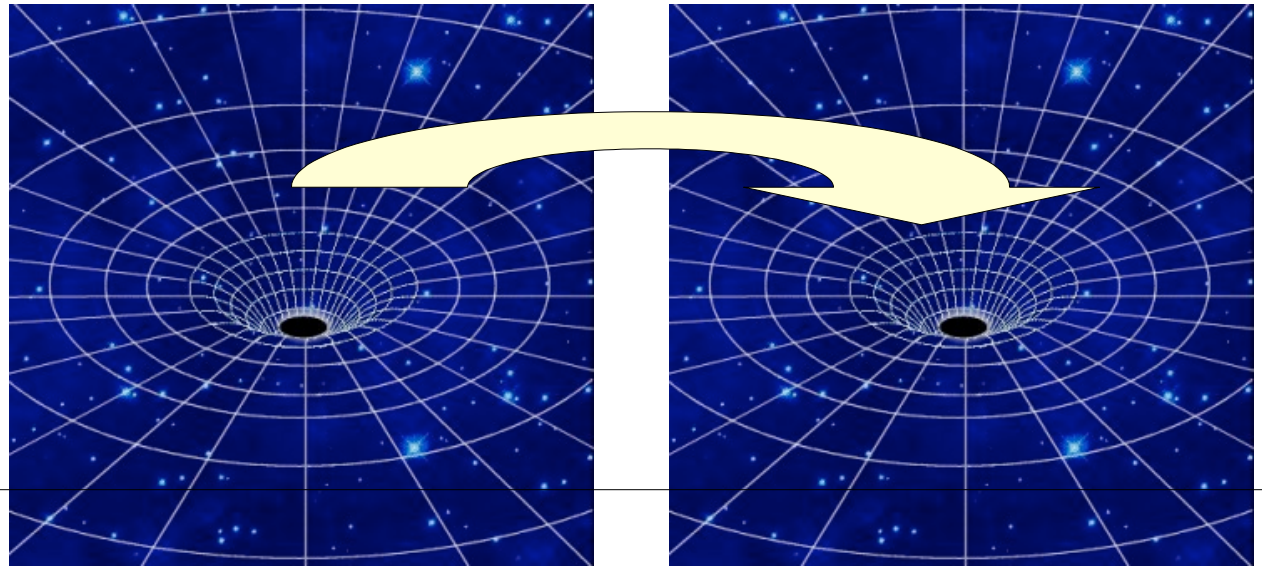


The backout / “rollback” issue

- Basic facts:
  - You can roll-back your planned changes but not the actual system state
  - You can partially restore state from back-up
  - Some blunders will cause irreversible loss or damage that “rollback” cannot fix

The roll-forward manoeuvre

- Convergence suggests that rollback thinking is partly misguided
  - Always think of moving forward to properly implement the promises you need to make
  - Placed the black-hole incorrectly? Move it and try again



7. Release Management

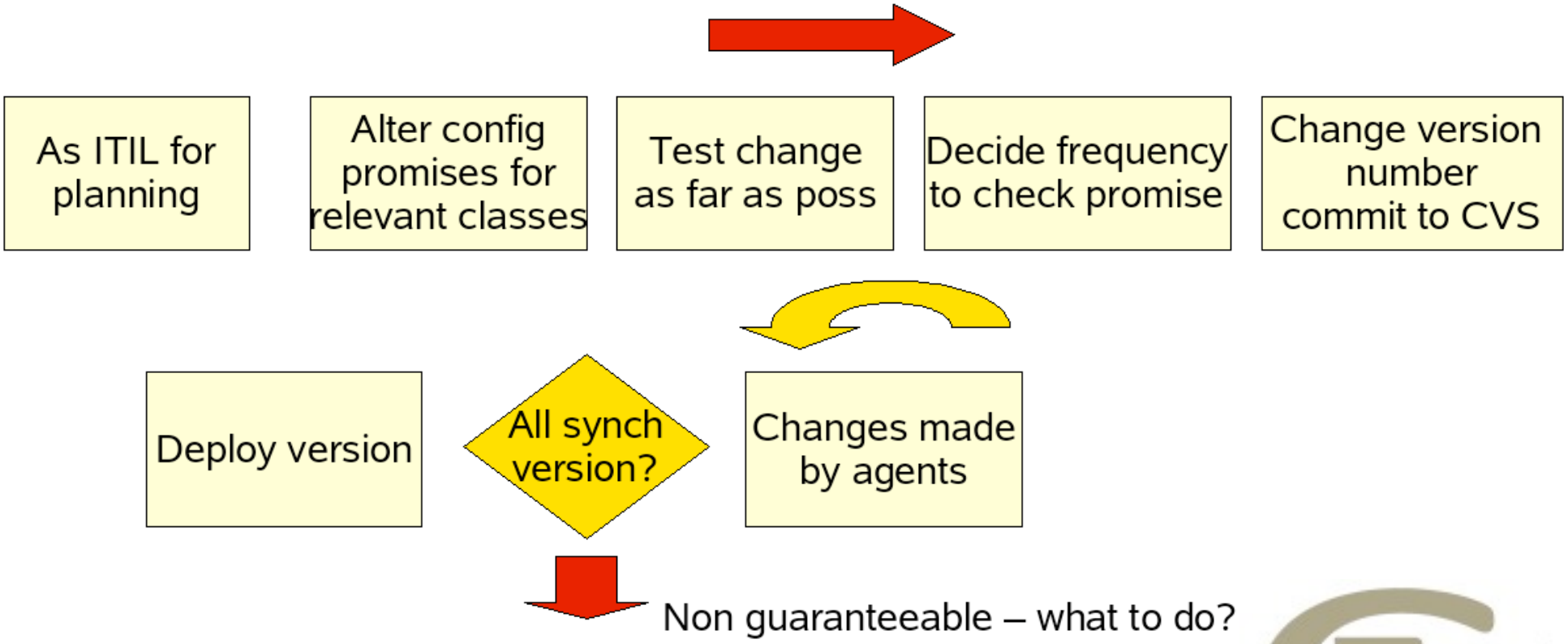
There should be a release policy

- A meta-policy for your configuration
  - What qualities should it have?
  - What is the release date/schedule for improvements?
  - Who is reponsible for collecting inputs and verifying their quality?
- Simply put: a continuous improvement cycle

Versioning

- Even if rollback is an over-simplification it can be important to trace changes for release management
  - Which ITIL process led to which change?
  - Which version of the process was enacted?

Promise versions in cfengine



Versioning

- Use `cfinputs_version` variable
- Keep separate ITIL processes in separate files
- Use auditing to follow implementation

```
control:

  cfinputs_version = ( 1.2.3 )

  Auditing = ( on )
```

Audit logs – if you have space!

Audit log nexus

Scan convergence	Observed	Promise made	Promise originates in	Promise version	line
Cycle complete Mon Oct 8 10:11:36 2007, lock acquired, was applied but performed no required actions	Cfagent starting	Mon Oct 8 10:11:36 2007	schedule		0
[Mon Oct 8 10:11:38 2007] op files/ Prepare./tmp/, was applied but performed no required actions	Commence checking file(s) in /usr/dt/bin	Mon May 14 16:02:13 2007	/local/iu/cfengine/inputs/cf.solaris	1.0	103
[Mon Oct 8 10:35:29 2007] op shellcommand. /local/iu/bin/BuildPasswdFiles/, was a regular (repeatable) maintenance task	Finished script /local/iu/bin/BuildPasswdFiles	Fri Aug 17 17:35:21 2007	/var/cfengine/inputs/cf.site	1.0	153
[Mon Oct 8 10:35:30 2007] op shellcommand. /iu/nexus/ua/mysql/GetAliases/php/>& 1dev/null/, was a regular (repeatable) maintenance task	Finished script /iu/nexus/ua/mysql/GetAliases.php > /dev/null 2>&1	Tue Feb 20 08:31:28 2007	/var/cfengine/inputs/cf.mail	1.0	145
[Mon Oct 8 10:35:34 2007] op shellcommand. /iu/nexus/ua/mysql/GetRejectStudents/ php/, was a regular (repeatable) maintenance task	Finished script /iu/nexus/ua/mysql/GetRejectStudents.php	Tue Feb 20 08:31:28 2007	/var/cfengine/inputs/cf.mail	1.0	146
[Mon Oct 8 10:35:39 2007] op shellcommand. /iu/nexus/ud/listmgr/Updates/alle/php/ , was a regular (repeatable) maintenance task	Finished script /iu/nexus/ud/listmgr/Updates-alle.php	Tue Feb 20 08:31:28 2007	/var/cfengine/inputs/cf.mail	1.0	147
[Mon Oct 8 10:35:40 2007] op shellcommand. /usr/lib/sendmail/, was a regular (repeatable) maintenance task	Finished script /usr/lib/sendmail -q -Ac	Tue Feb 20 08:31:28 2007	/var/cfengine/inputs/cf.mail	1.0	162

Check what process led to which cha

Packages

- Cfengine does not have a package format
- Native software package formats can be used to arrange for versioned package management

```
control:
  redhat::
     DefaultPkgMgr = ( rpm )

packages:
     redhat_8_0::
        “make” version=0:4.5-2
               cmp=ge
```

8. Service Level Management

A future challenge

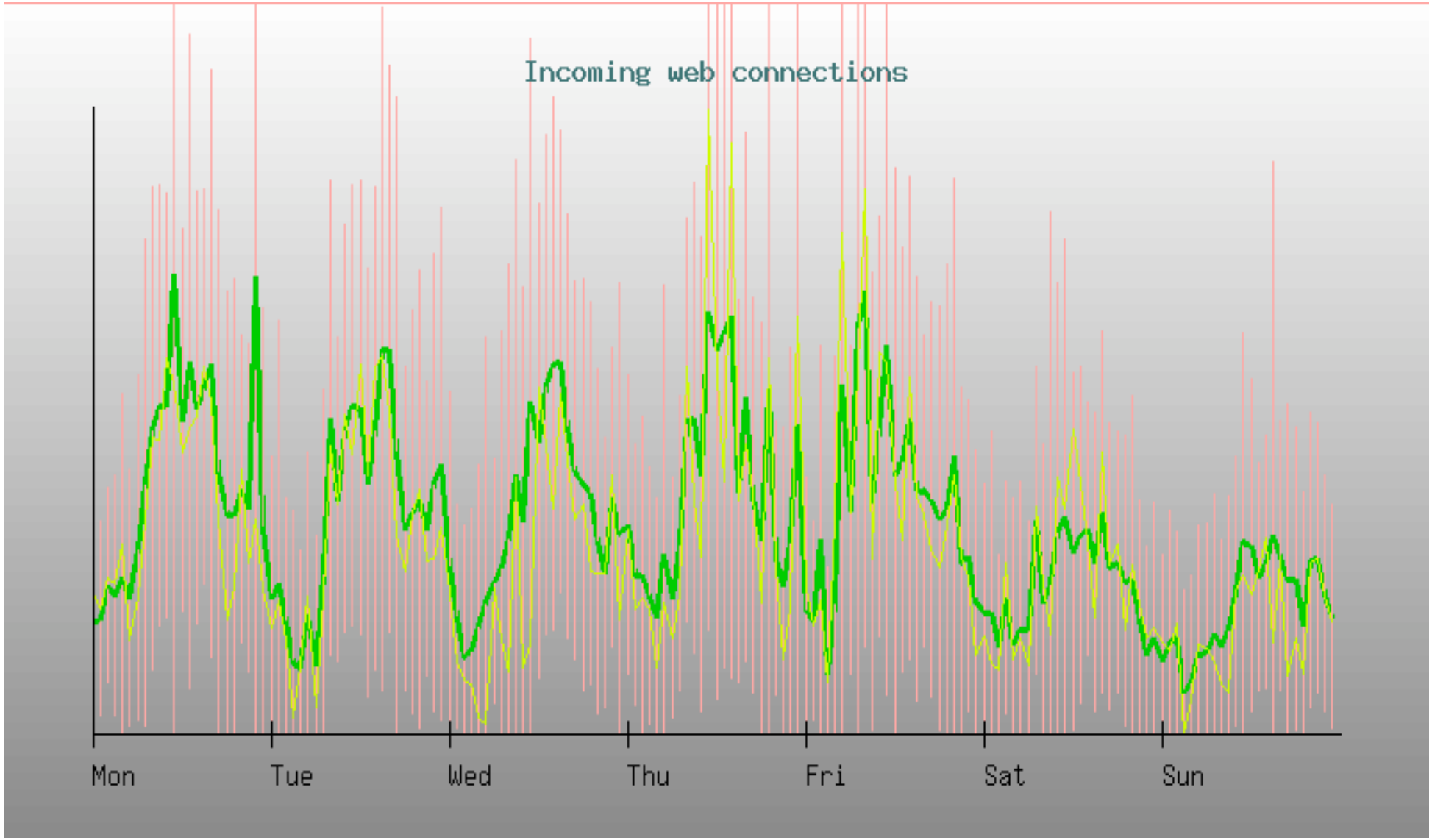
Performance indicators

- Service level agreements – wrt process implementation

Peformance recently measured on eternity

Copy(localhost:/iu/eternity/cfengine/inputs > /var/cfengine/inputs)	last performed at Wed Sep 19 13:45	completed in 0.0001 mins	Av 0.0001 mins	± 0.0001 mins
Copy(localhost:/iu/eternity/cfengine/methods > /var/cfengine/modules)	last performed at Wed Sep 19 13:45	completed in 0.0001 mins	Av 0.0000 mins	± 0.0001 mins
Copy(localhost:/iu/eternity/masterfiles/VIEWCVS_styles.css > /srv/viewcvs/doc/styles.css)	last performed at Wed Sep 19 13:30	completed in 0.0000 mins	Av 0.0000 mins	± 0.0001 mins
Copy(localhost:/iu/eternity/masterfiles/Wiki-LocalSettings.php > /iu/eternity/htdocs/wiki/LocalSettings.php)	last performed at Mon Sep 10 09:45	completed in 0.0000 mins	Av 0.0000 mins	± 0.0005 mins
Copy(localhost:/iu/eternity/masterfiles/ca.crt > /etc/apache2/ssl.crt/ca.crt)	last performed at Wed Sep 19 13:30	completed in 0.0000 mins	Av 0.0000 mins	± 0.0001 mins
Copy(localhost:/iu/eternity/masterfiles/ca.key > /etc/apache2/ssl.key/ca.key)	last performed at Wed Sep 19 13:30	completed in 0.0000 mins	Av 0.0000 mins	± 0.0001 mins

Service levels – another story



Summary

- Configuration is a service
  - Release mangement
  - Change management
  - Incident and problem management
- Configuration management
  - Not quite what we are used to (assets + relationships)
  - Promise model of cfengine encompasses all these things quite simply with a little abstraction

Some “best practices”?

- Don't make planned or unplanned changes by hand
- Create a versioned process for making convergent changes via cfagent
- Use “gold installation” and authorized packages to make granular substrate
- Build customizations as needed
  - Either from additional overlays
  - Direct editing
- Do everything via cfengine and you'll have an audit trail

Questions?

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