Centre for Intelligent Systems and their Applications

THE UNIVERSITY of EDINBURGH informatics

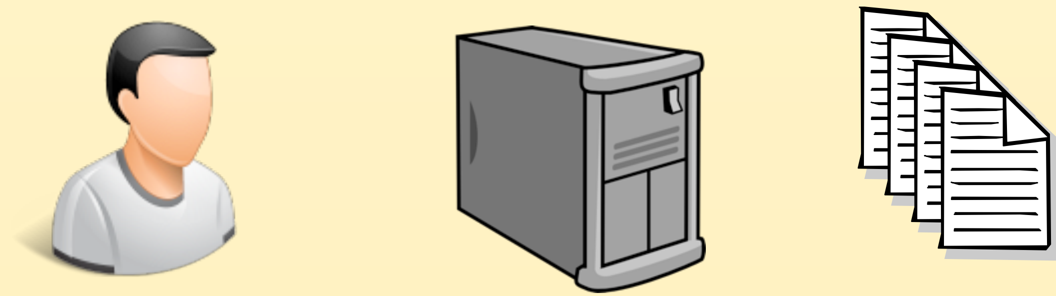
Collaborative Configurations

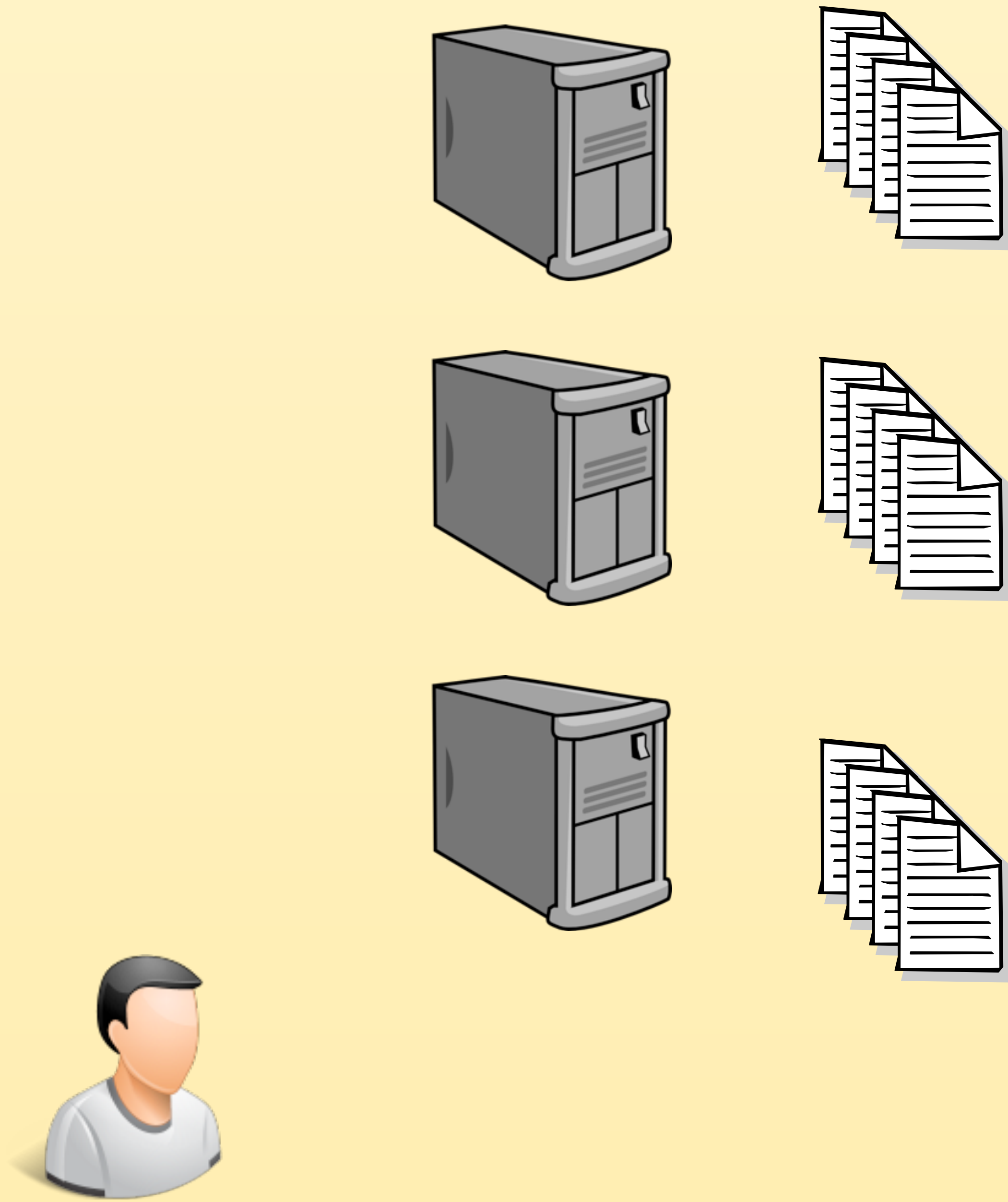
Paul Anderson

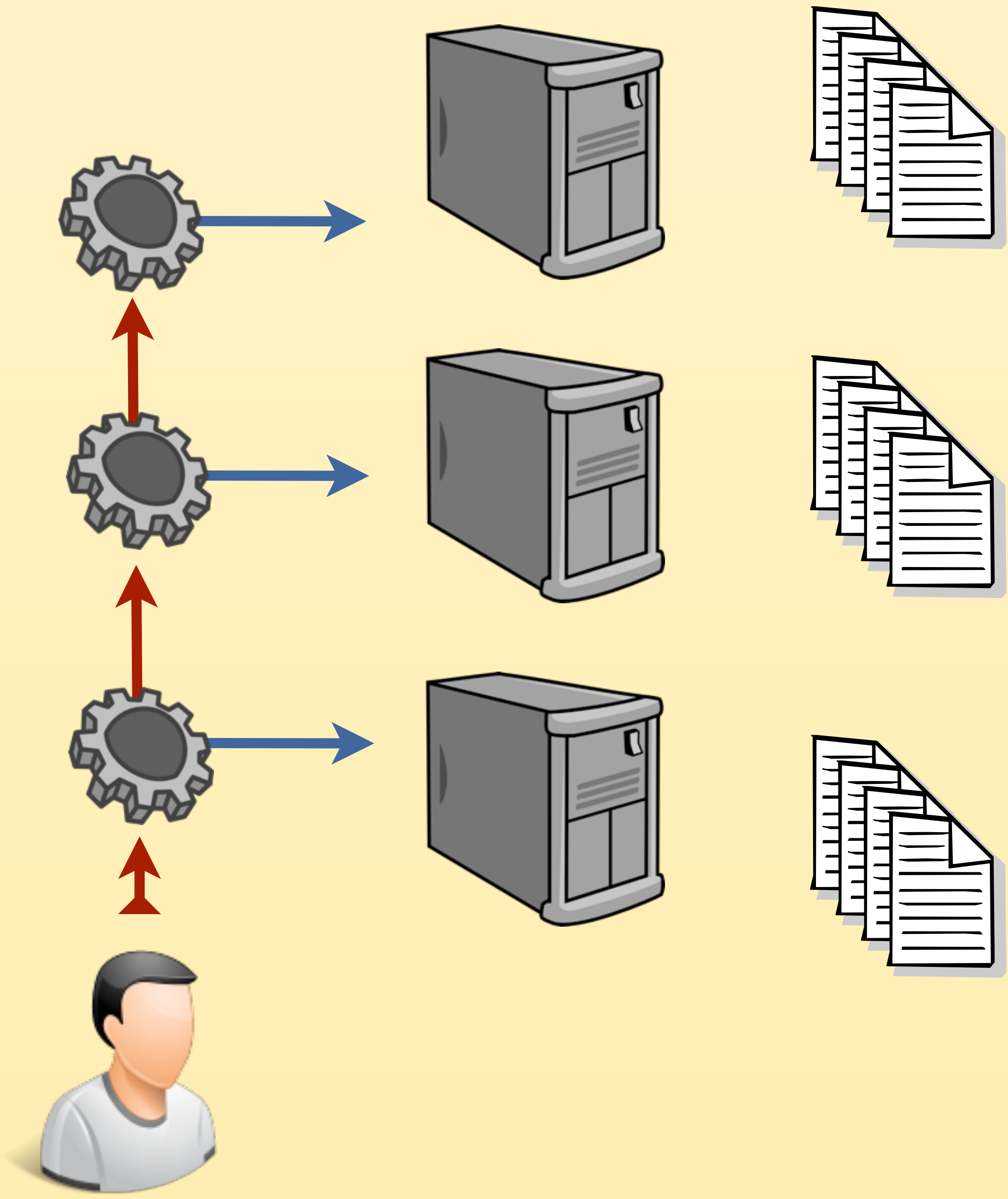
dcspaul@ed.ac.uk

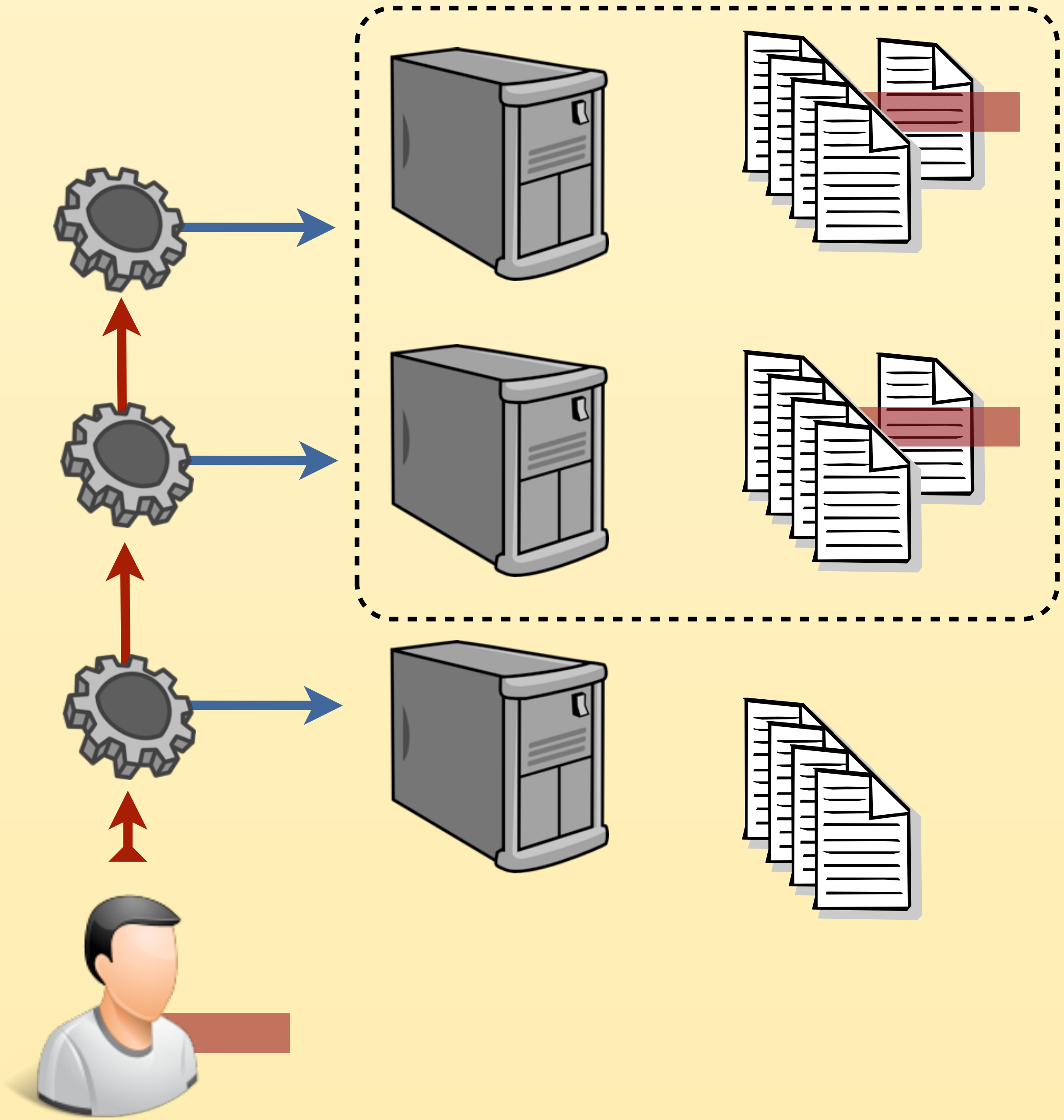
http://homepages.inf.ed.ac.uk/dcspaul

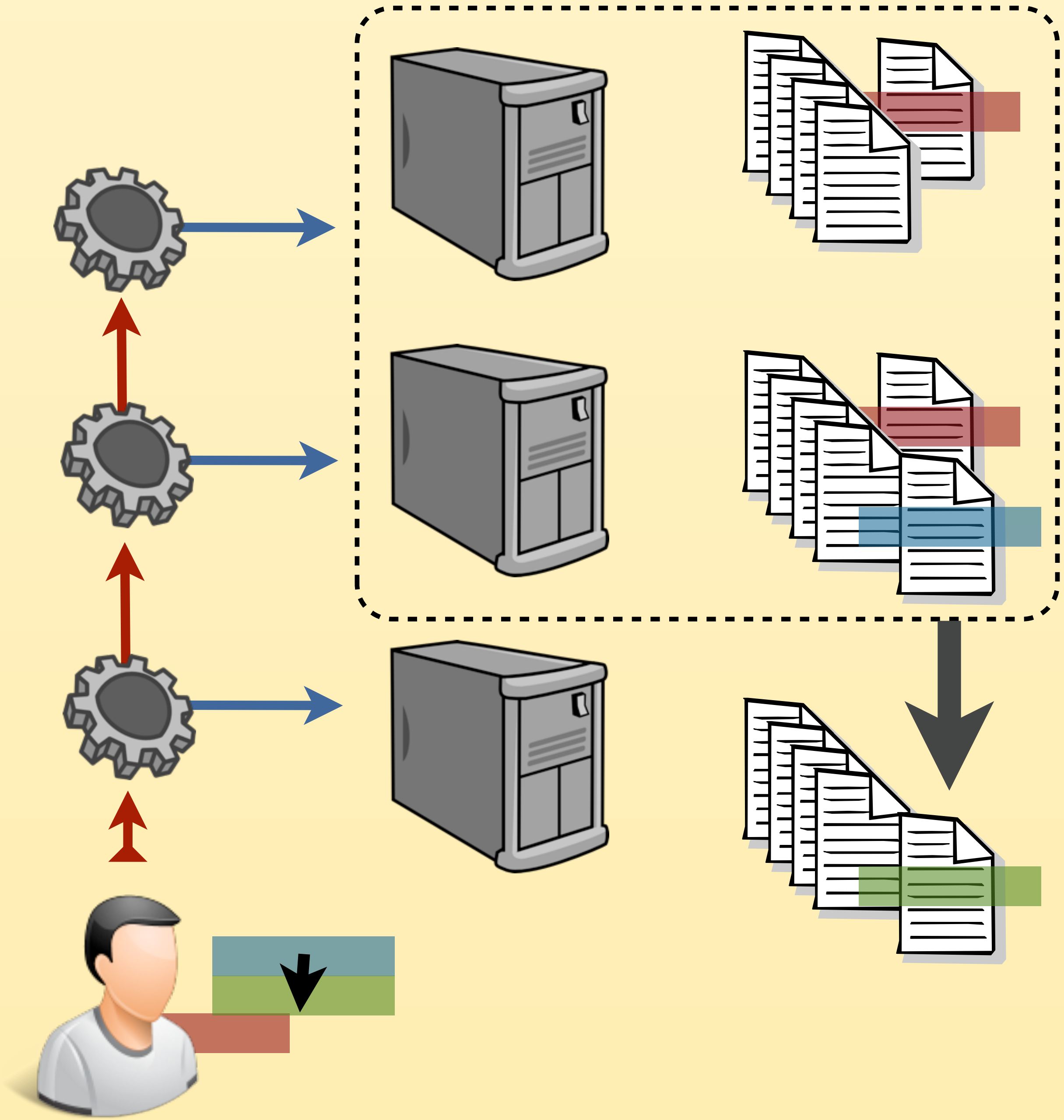
http://homepages.inf.ed.ac.uk/dcspaul/publications/google-2012.pdf

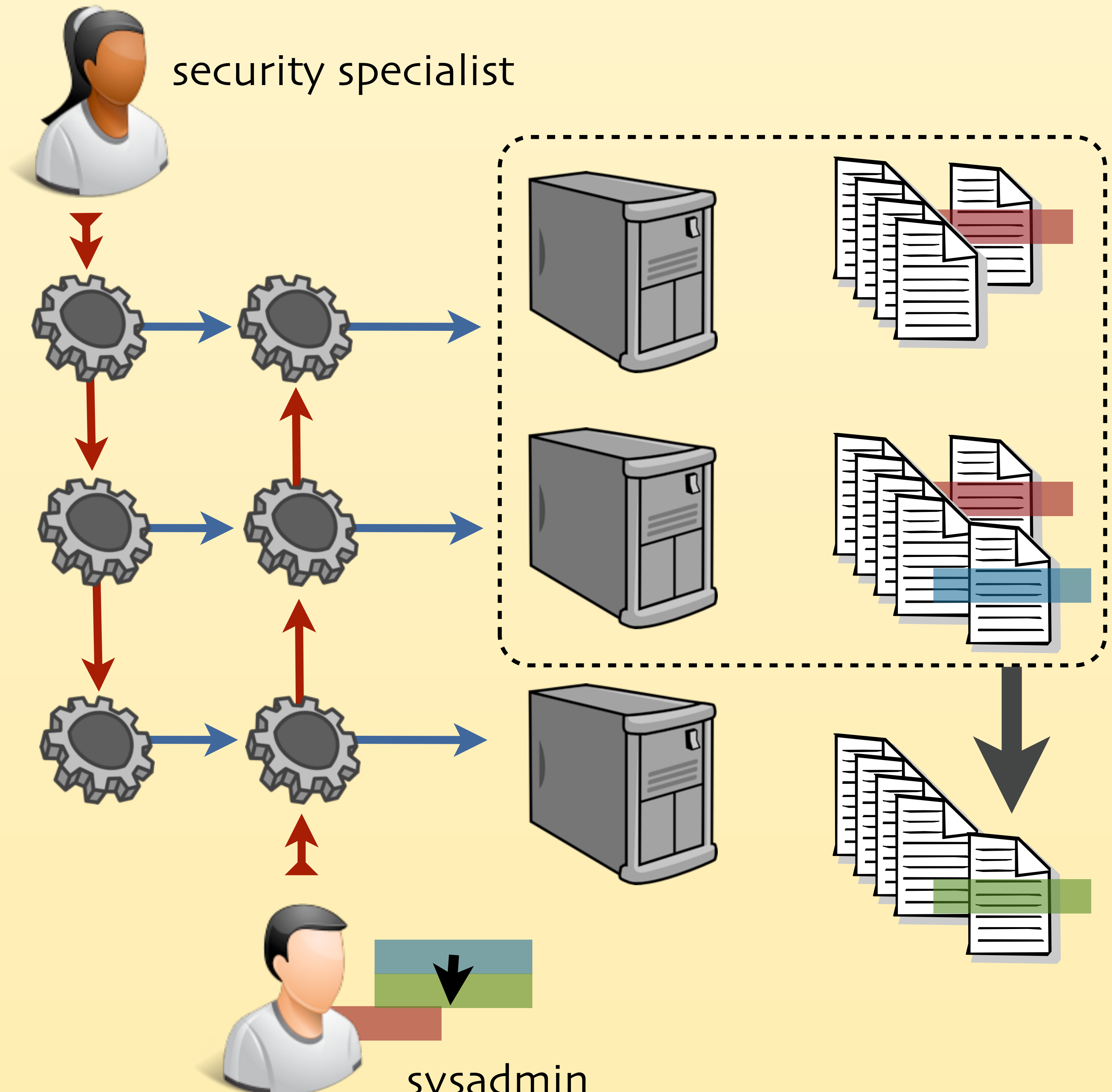


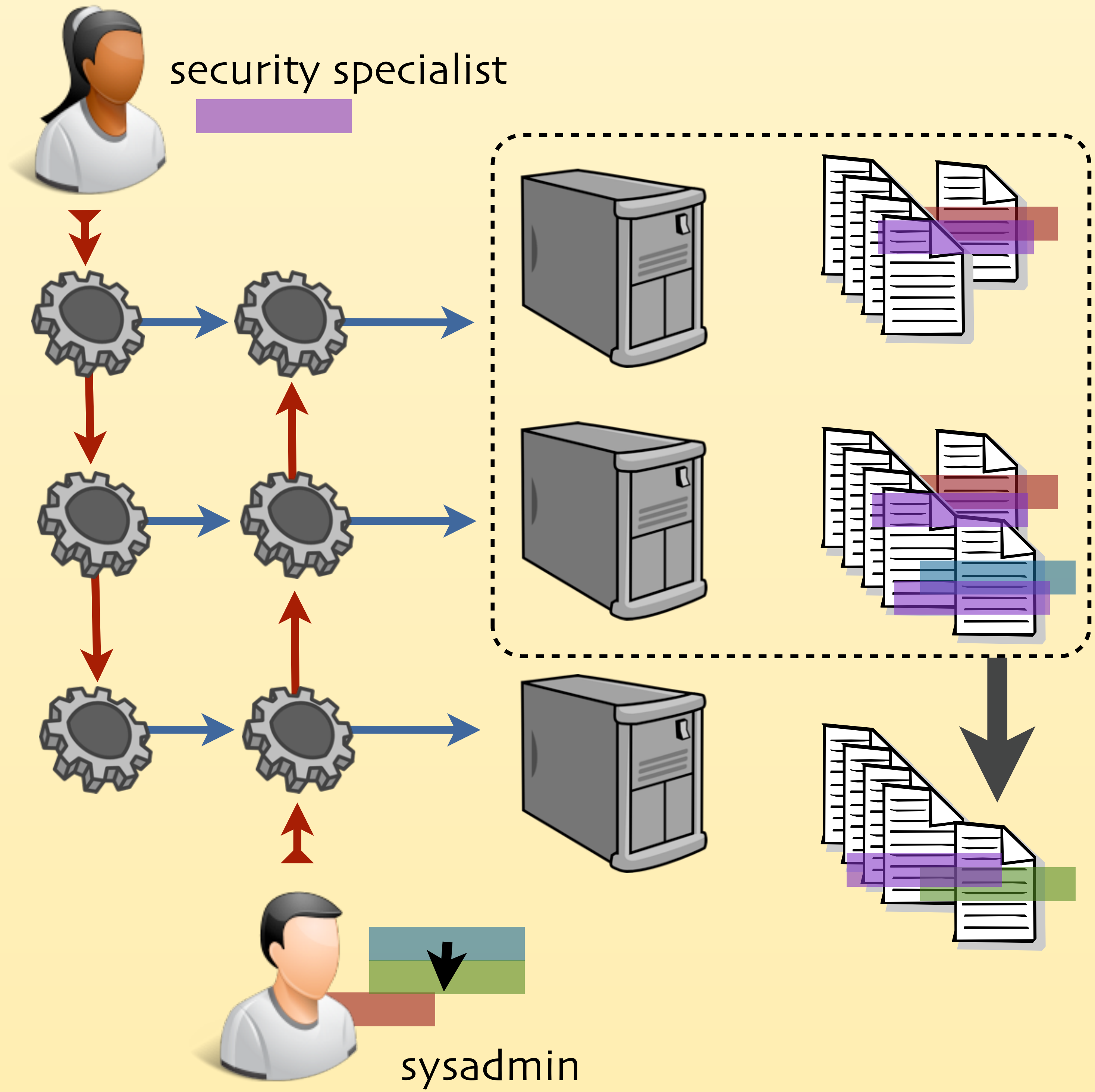


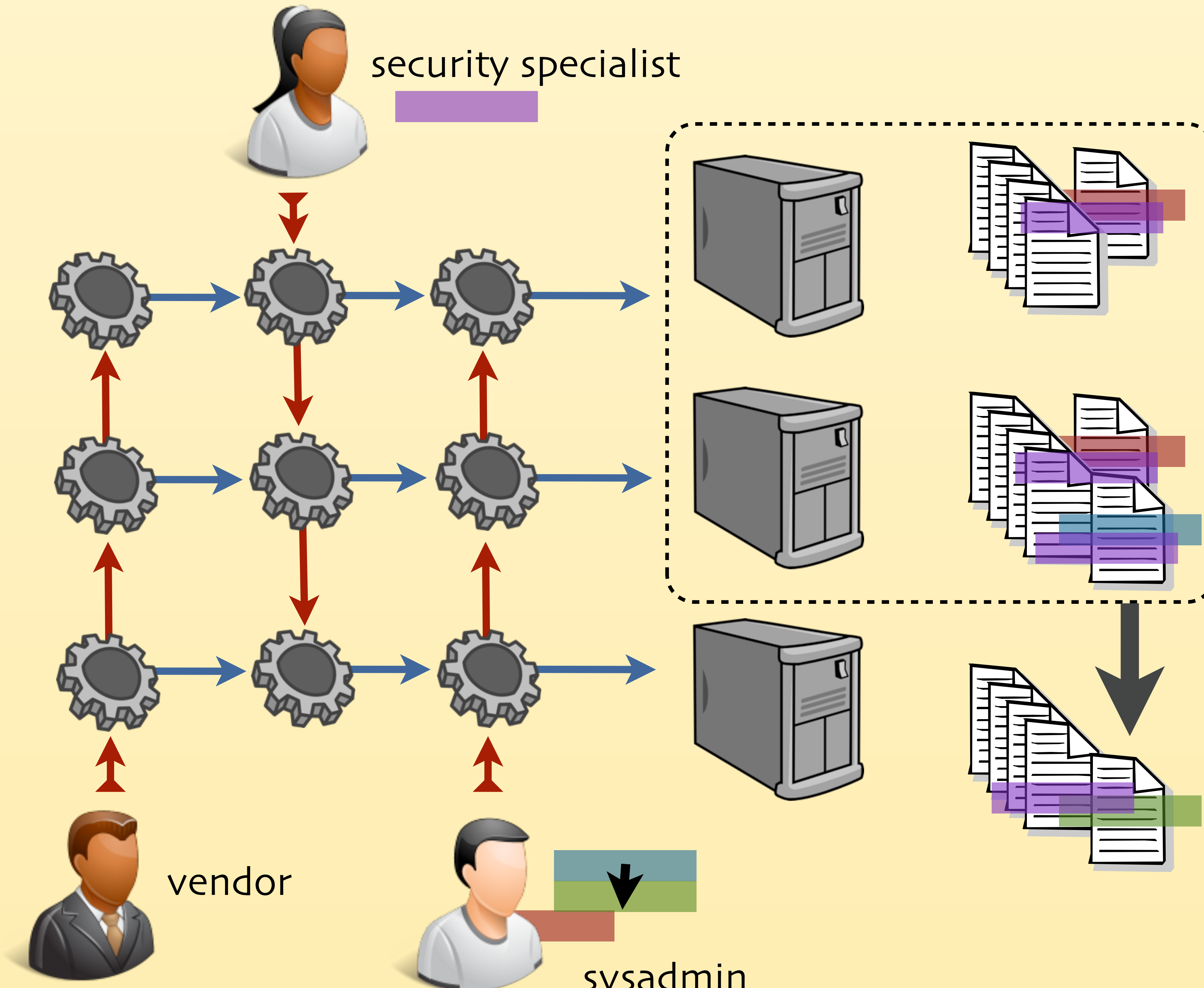


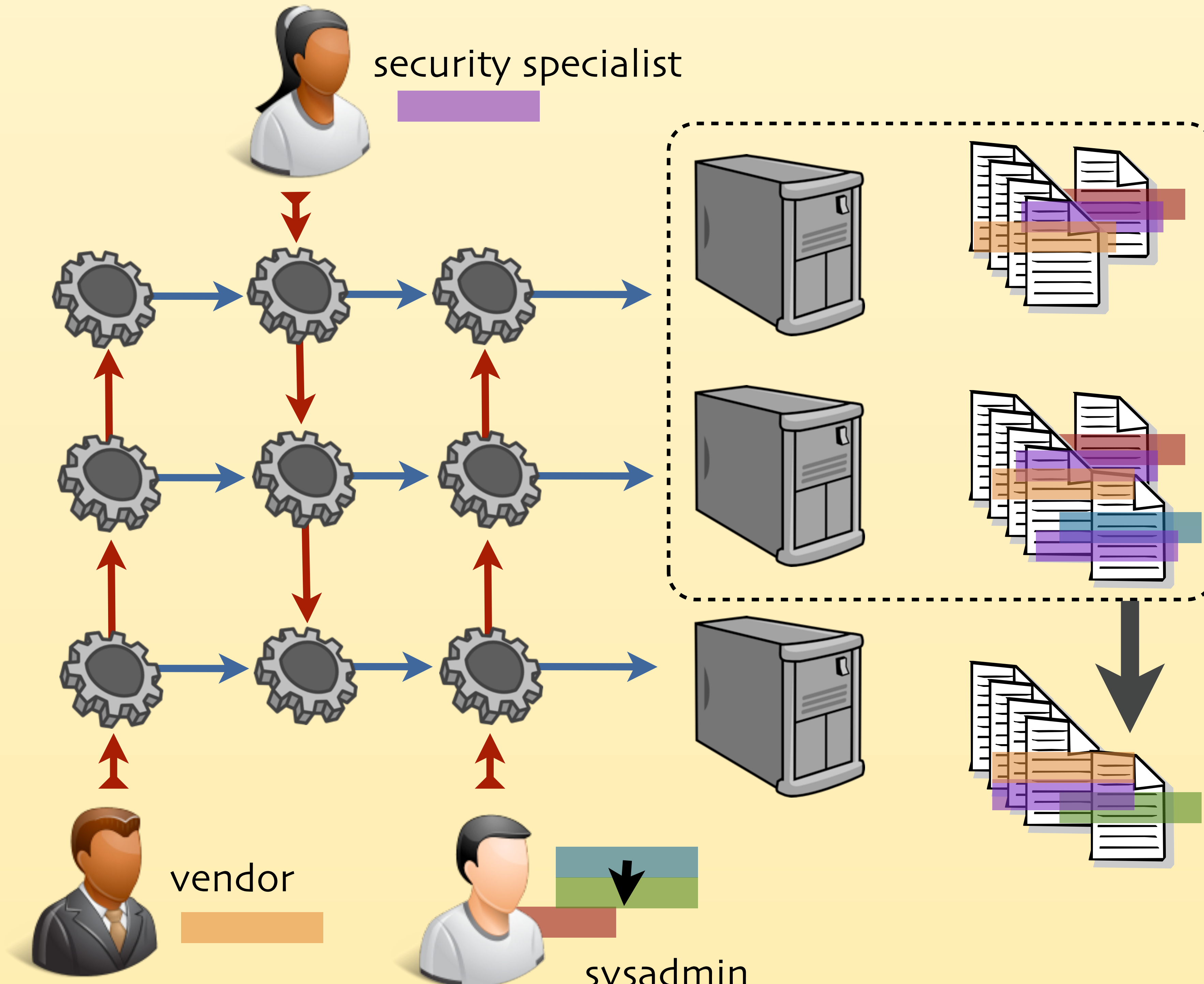


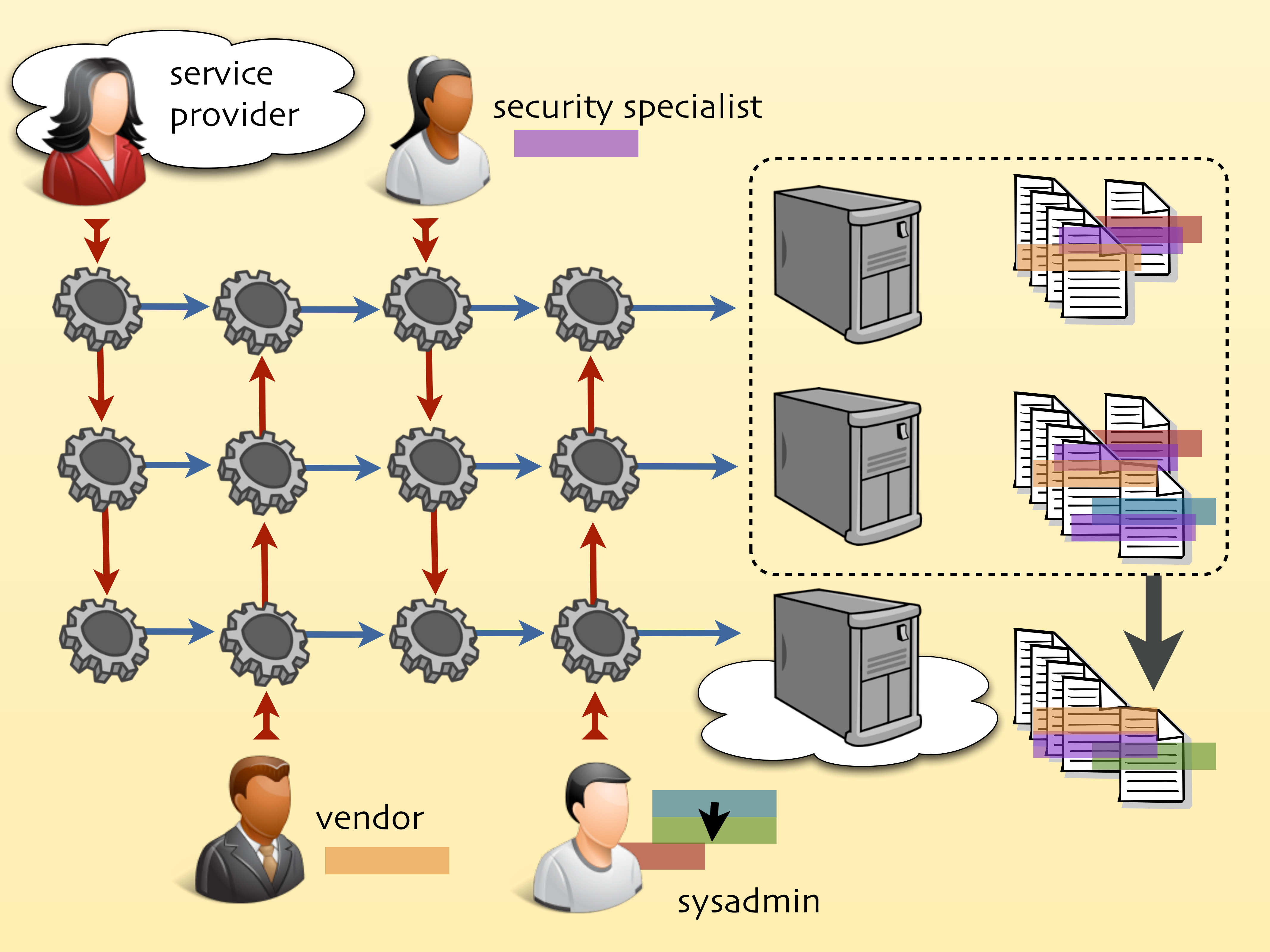



security specialist
sysadmin


security specialist
sysadmin


security specialist
vendor
sysadmin


security specialist
vendor
sysadmin


service provider
security specialist
vendor
sysadmin

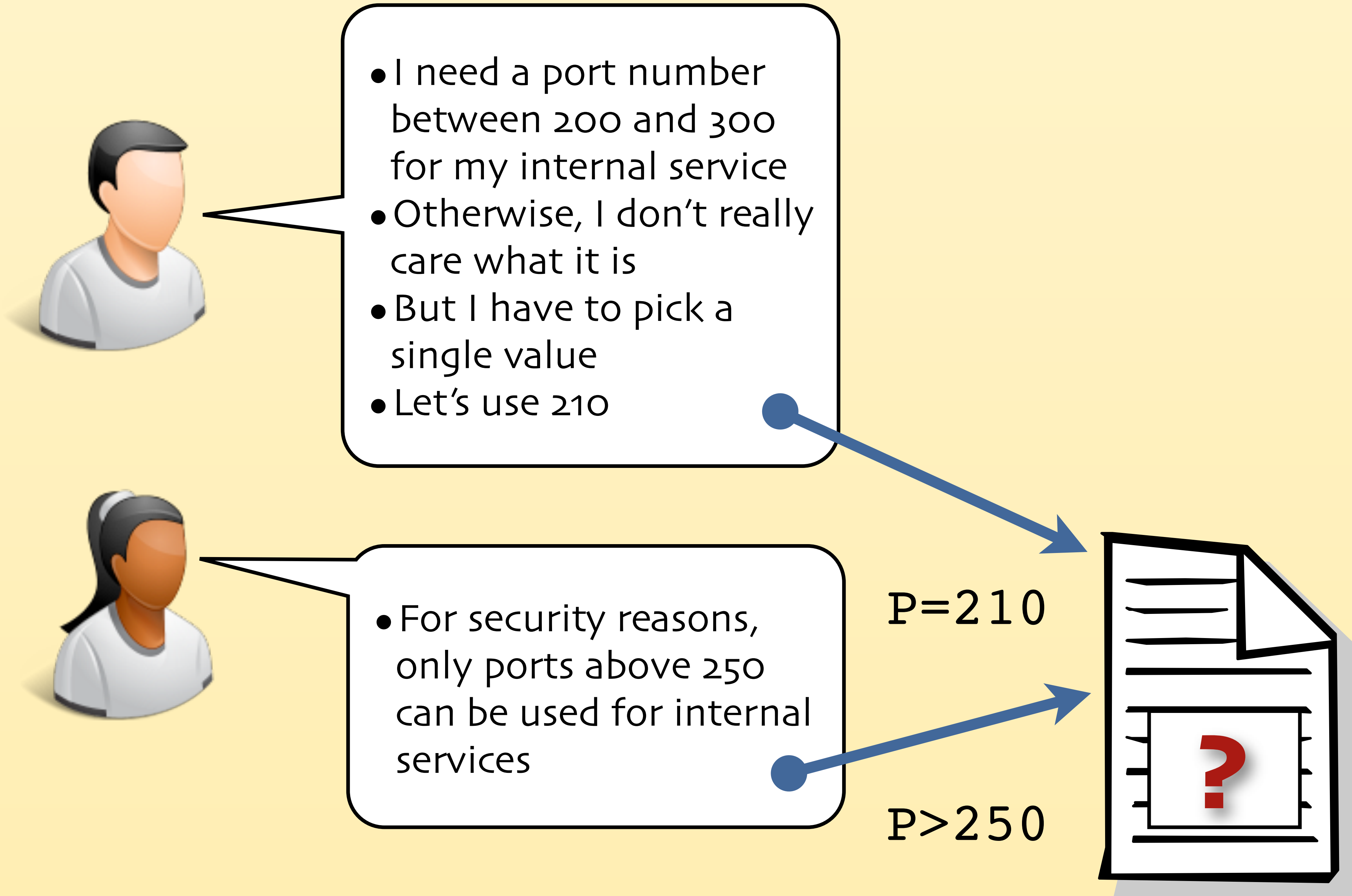
Aspect Composition

The problem is to compose these independent "aspects" to form a consistent specification

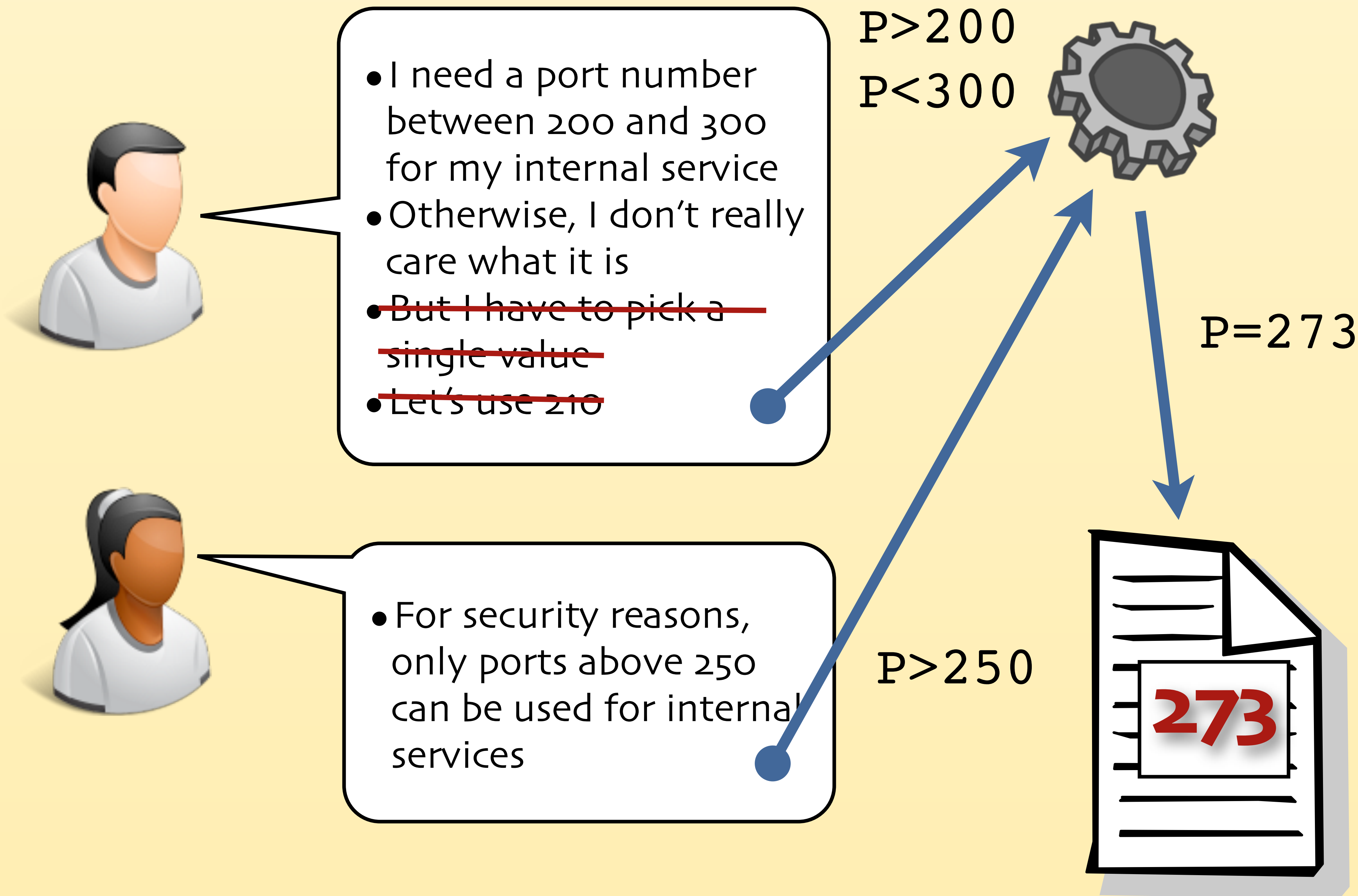
- with no unnecessary human negotiation
  - rapid configuration changes may be necessary to repair a failed system

We also need to be able to understand the "provenance" of the resulting configuration parameters

- how was the value of that parameter computed ?
- if a particular parameter is wrong ...
  - who needs to change what to fix it ?
- if a particular parameter requires special authorisation ...
  - who was involved in contributing to its value, and are they authorised ?



• I need a port number between 200 and 300 for my internal service
• Otherwise, I don't really care what it is
• But I have to pick a single value
• Let's use 210
• For security reasons, only ports above 250 can be used for internal services
P=210
P>250
?



• I need a port number between 200 and 300 for my internal service
• Otherwise, I don't really care what it is
• ~~But I have to pick a single value~~
• ~~Let's use 210~~
P>200
P<300
P=273
• For security reasons, only ports above 250 can be used for internal services
P>250
273

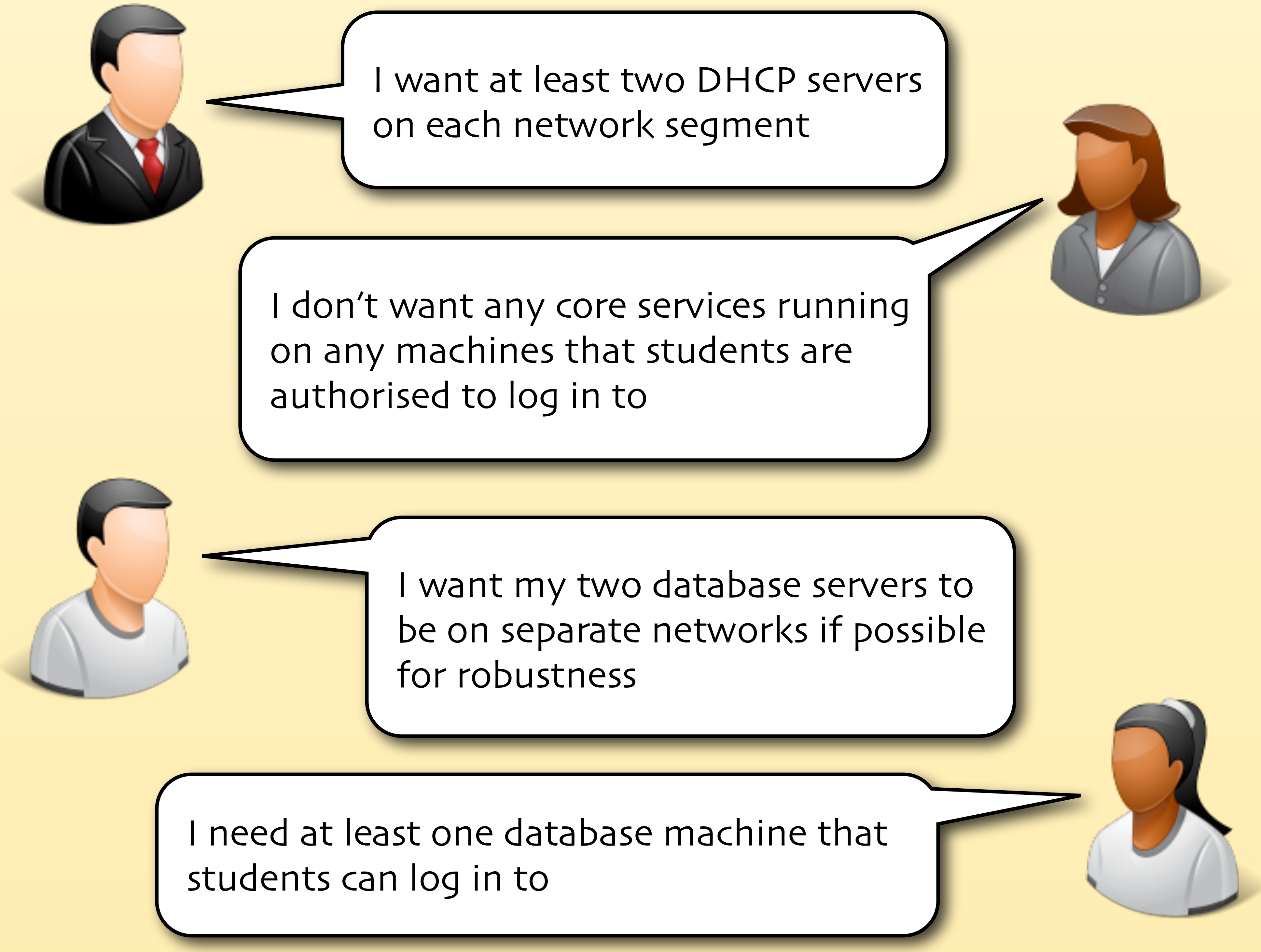
Constraints

Using constraint solvers for configuration problems is not new

- Alloy for network configuration
- Cauldron (HP)
- VM allocation (Google challenge)

But we have a different motivation which changes the emphasis

- we want to integrate the constraints with a (usable) configuration language to support a separation of concerns
- the constraint problems are often comparatively simple to solve, but they are embedded in large volumes of "constant" configuration data
- some specific properties are important (see later) ...
  - preferences (soft constraints)
  - stability


I want at least two DHCP servers on each network segment
I don’t want any core services running on any machines that students are authorised to log in to
I want my two database servers to be on separate networks if possible for robustness
I need at least one database machine that students can log in to

Modelling

The most popular practical configuration languages ..

- are very good at reliably deploying large numbers of configuration parameters to large numbers of machines
- but they are not good at modelling higher-level abstractions such as those on the previous slide
- they have "evolved" gradually without a clear semantics
- and they have implementations which are not amenable to experimental extensions

Confsolve is an experimental constraint-based configuration language

- supports the necessary modelling
- generates an intermediate language which can be transformed fairly easily into an existing configuration language

Confsolve

An experimental constraint-based configuration language

- by John Hewson<john.hewson@ed.ac.uk>
  http://homepages.inf.ed.ac.uk/s0968244/
  (Sponsored by Microsoft Research)
- a general-purpose configuration language
  - no domain-specific knowledge
  - output can easily be transformed into some other language (eg. Puppet)
- the data model is an object-oriented hierarchy
  - constraints are possible at all levels
- compiles down to a standard constraint solver (MiniZinc)
- supports soft constraints and optimisation
- has a formal semantics for the translation
- supports "change minimisation"

Some Confsolve Classes

```
class Service {
    var host as ref Machine
    ...
}
class Datacenter {
    var machines as Machine[8]
    ....
}
class Machine { }
class Web_Srv extends Service { }
class Worker_Srv extends Service { }
class DHCP_Srv extends Service { }
```

Two Datacenters & Three Services

```
var cloud as Datacenter
var enterprise as Datacenter

var dhcp as DHCP_Service[2]
var worker as Worker_Service[3]
var web as Web_Service[3]
```

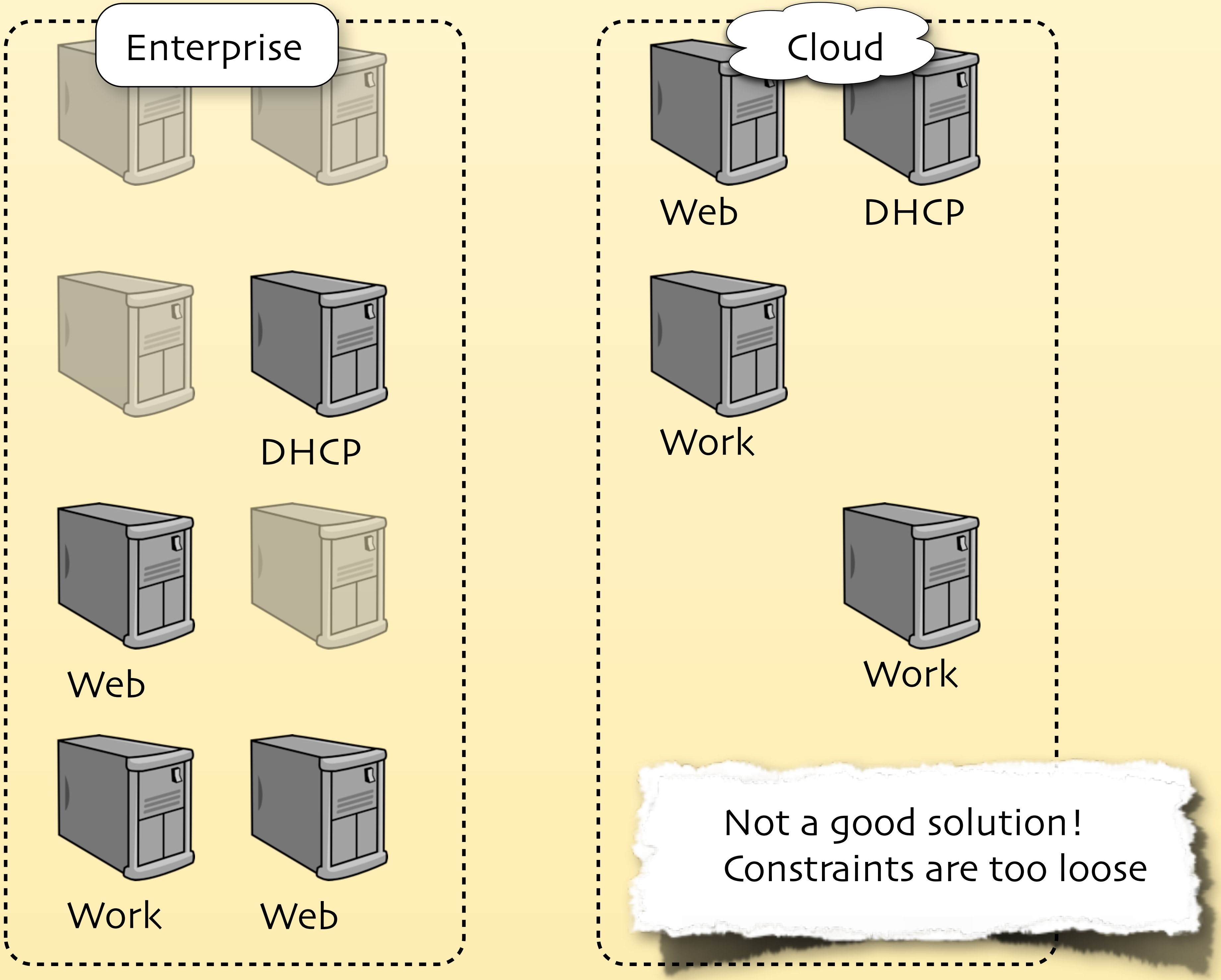
A Constraint

```
var services as ref Service[7]

where foreach (s1 in services) {
    foreach (s2 in services) {
        if (s1 != s2) {
            s1.host != s2.host
        }
    }
}
```

No two services on the same machine:

- this generates a correct configuration
    - no explicit assignment at all
    - not just validation
- this can be independently authored
    - no collaboration with the service authors, or system managers is required


Enterprise
DHCP
Web
Work
Web
Cloud
Web
DHCP
Work
Work
Not a good solution!
Constraints are too loose

An Optimisation Constraint

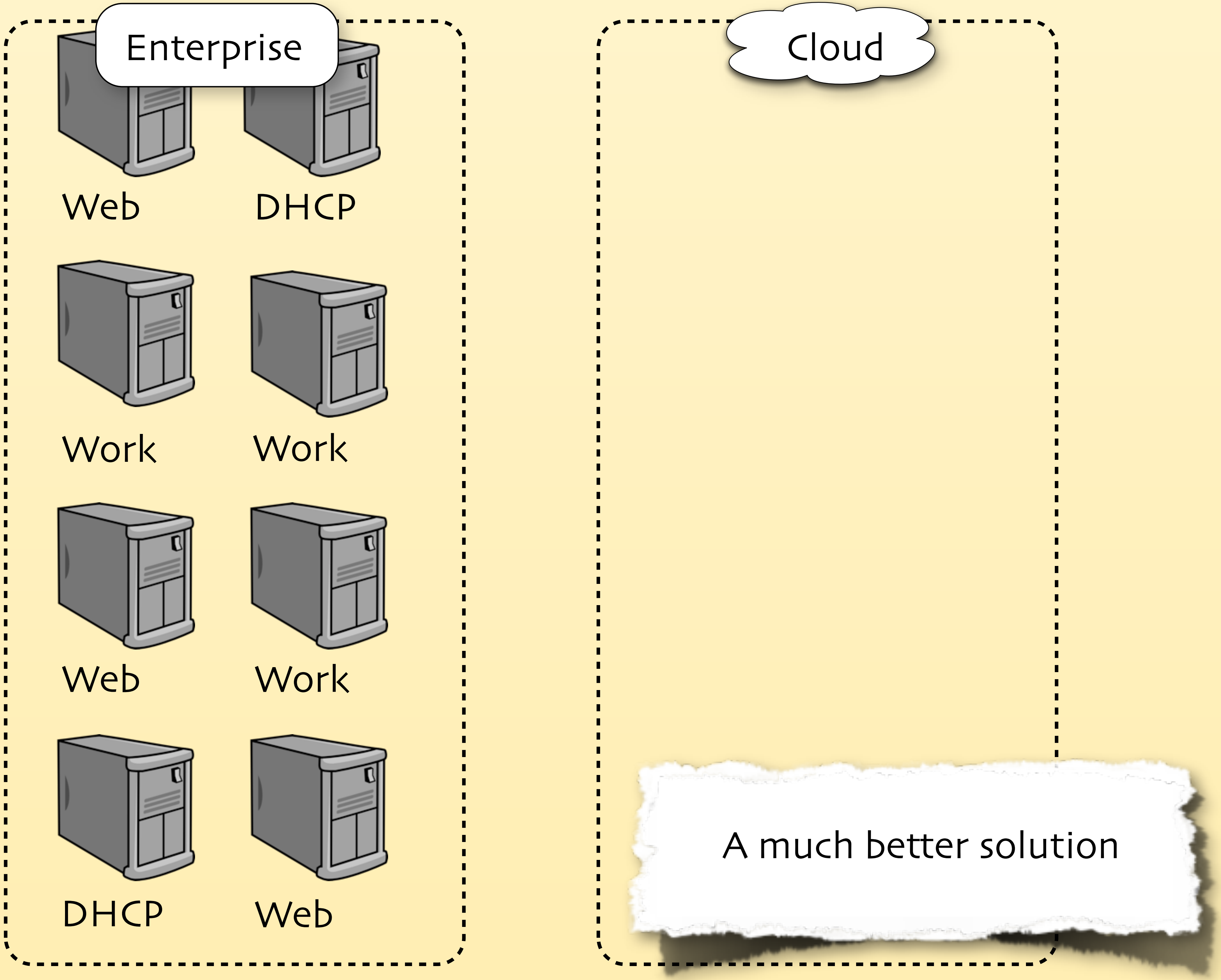
```
var utilisation as int

where utilisation == count (
   s in services
   where s.host in enterprise.machines)

maximize utilisation
```

“Favour placement of machines in the enterprise”

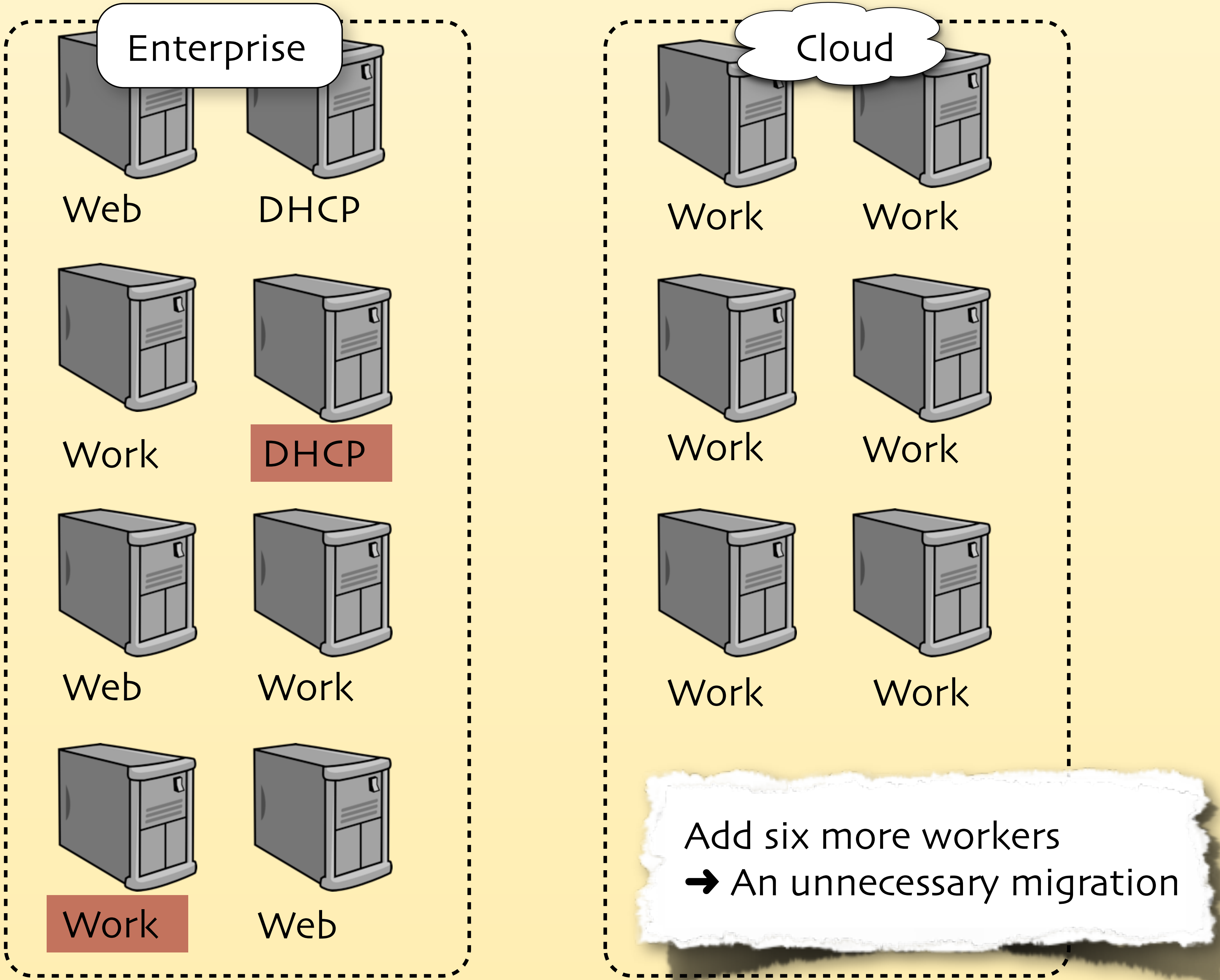
- this policy can be defined completely independently

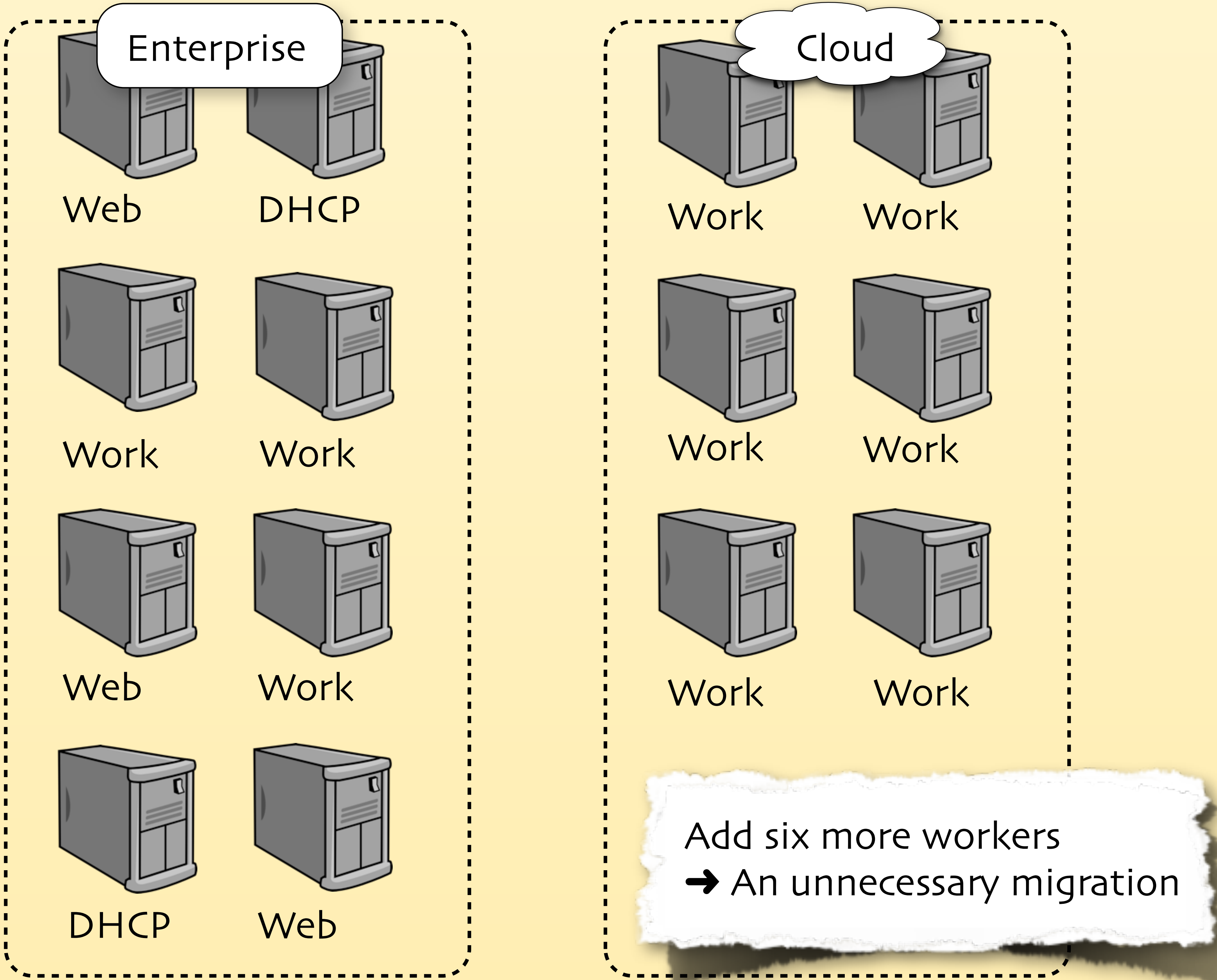

Enterprise
Web
DHCP
Work
Work
Web
Work
DHCP
Web
Cloud
A much better solution

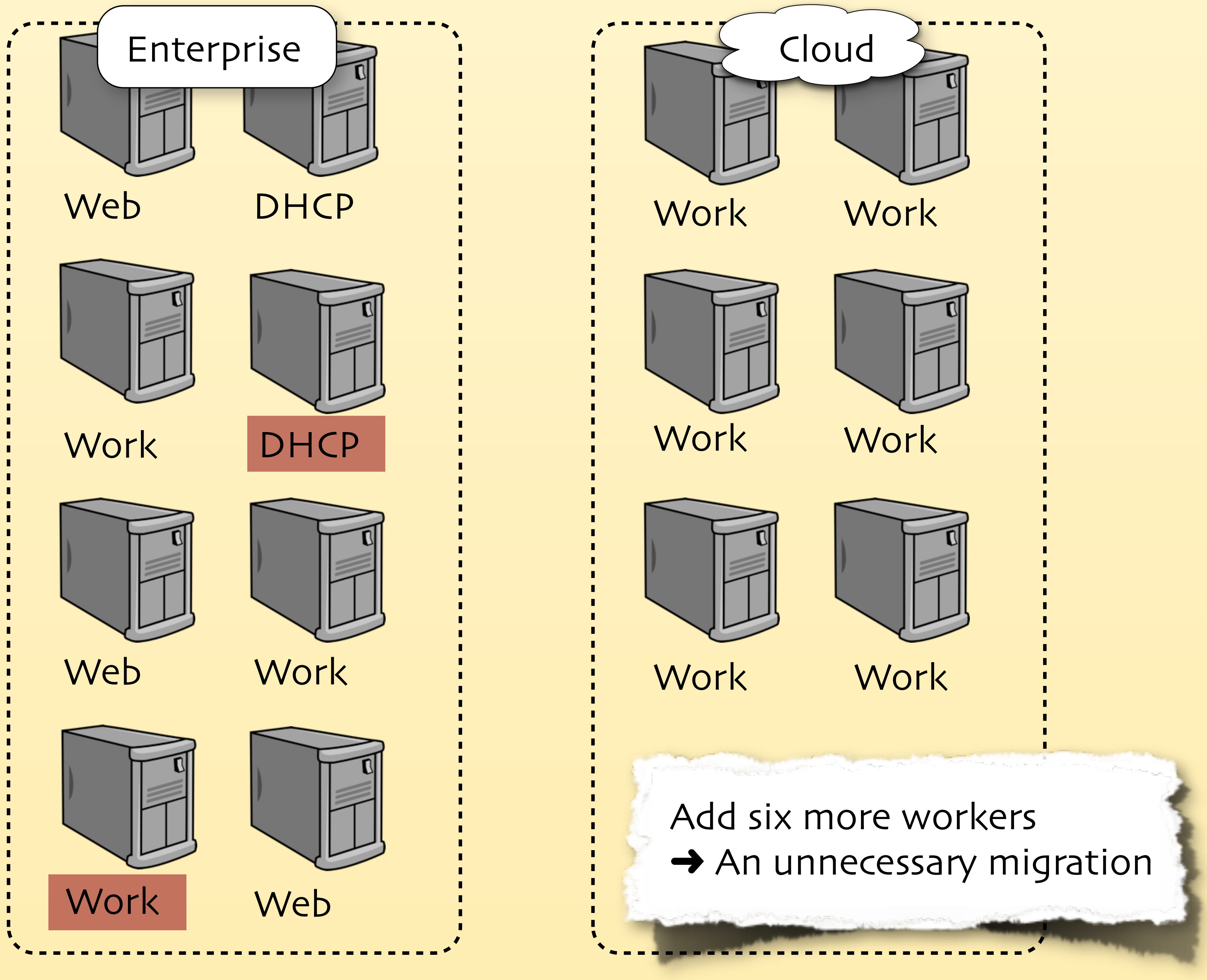
Add Six More Workers

```
var cloud as Datacenter
var enterprise as Datacenter

var dhcp as DHCP_Service[2]
```
~~var worker as Worker_Service[3]~~
```
var worker as Worker_Service[9]
var web as Web_Service[3]
```


Enterprise
Web
DHCP
Work
DHCP
Web
Work
Work
Web
Cloud
Work
Work
Work
Work
Work
Work
Add six more workers
➜ An unnecessary migration


Enterprise
Web
DHCP
Work
Work
Web
Work
DHCP
Web
Cloud
Work
Work
Work
Work
Work
Work
Add six more workers
➜ An unnecessary migration

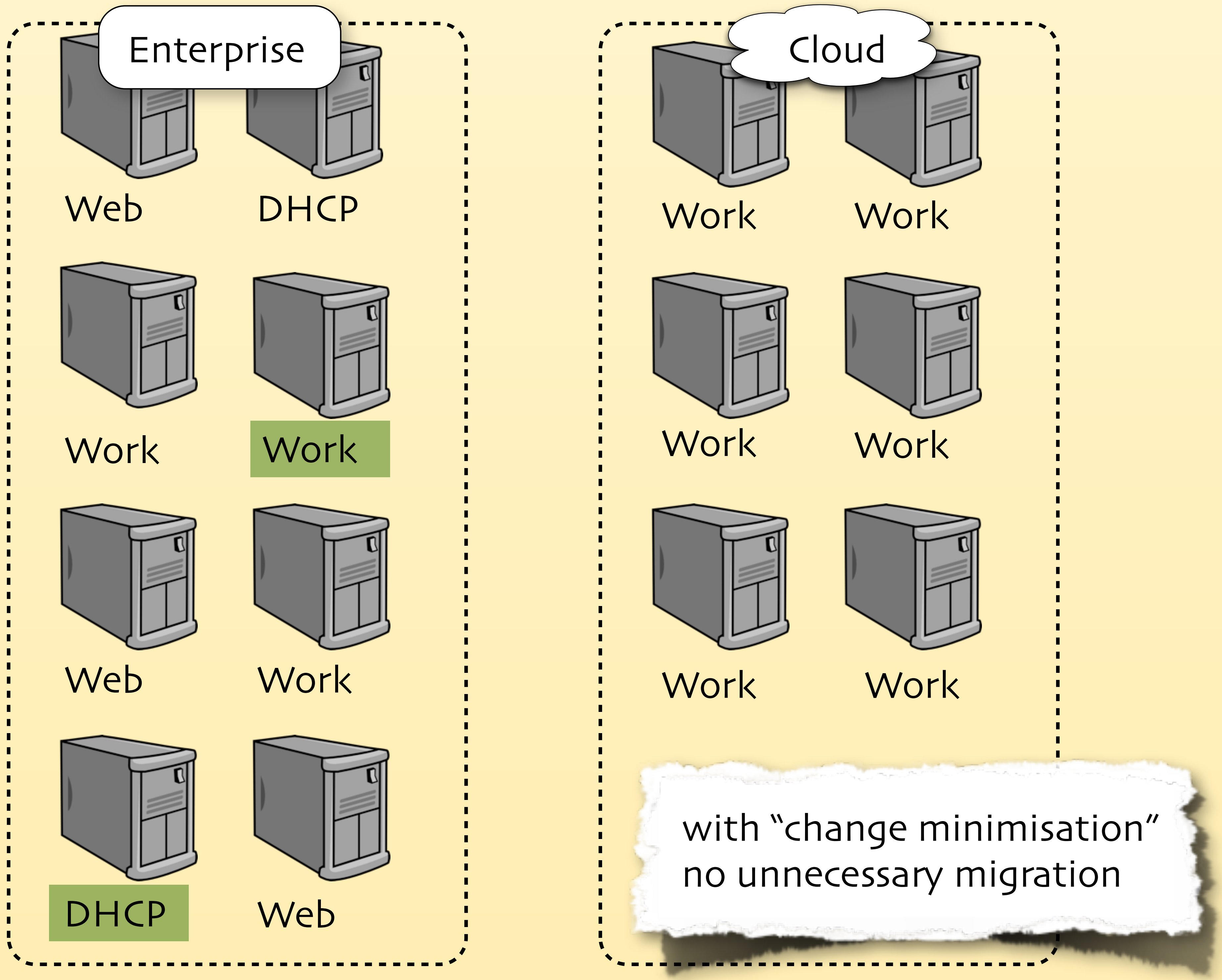

Enterprise
Web
DHCP
Work
DHCP
Web
Work
Work
Web
Cloud
Work
Work
Work
Work
Work
Work
Add six more workers
➜ An unnecessary migration

Minimising Changes

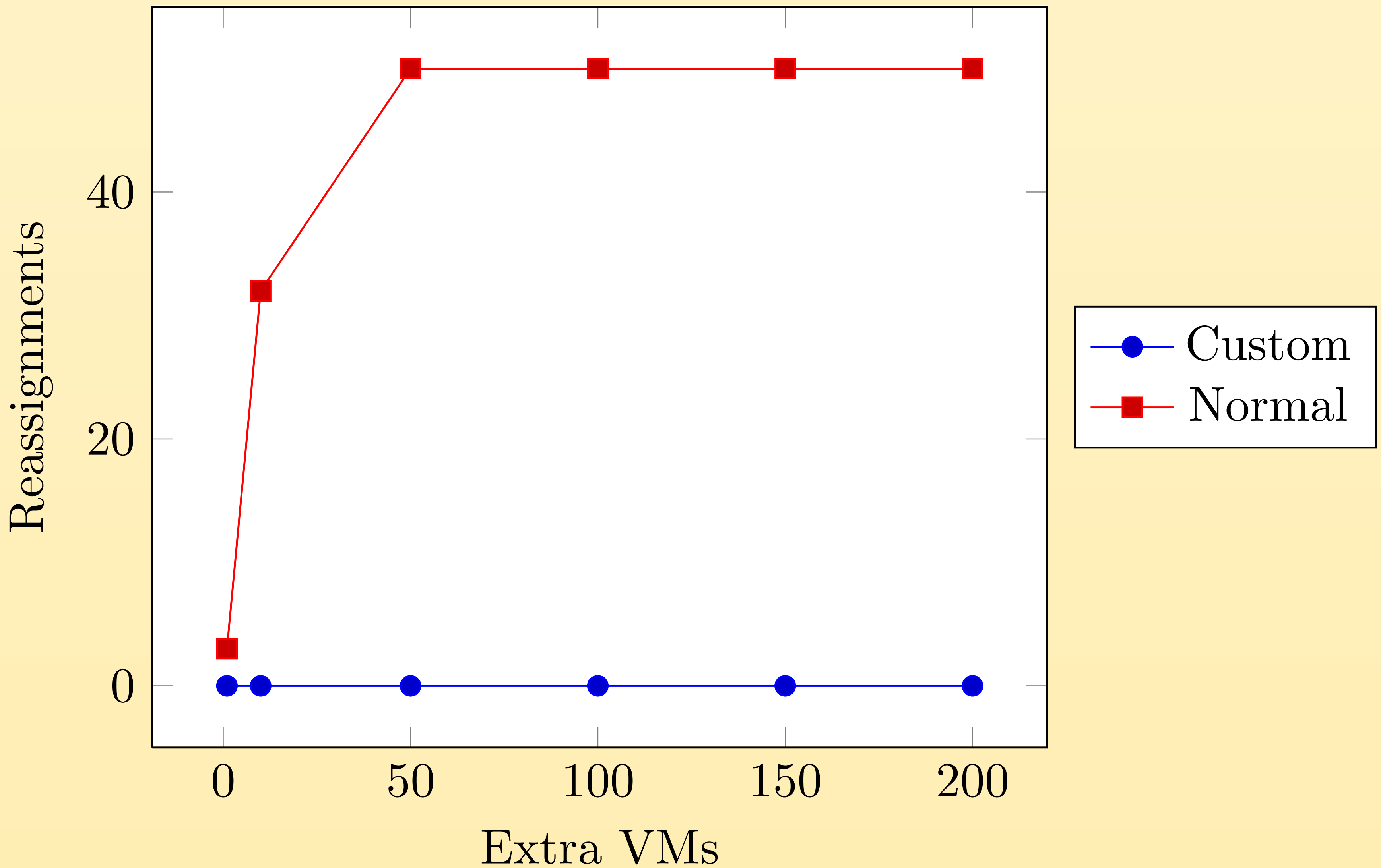
```
change {
  forall s in services {
    s.host = ~s.host;
  };
}
```

"Don't move machines once they have been allocated"

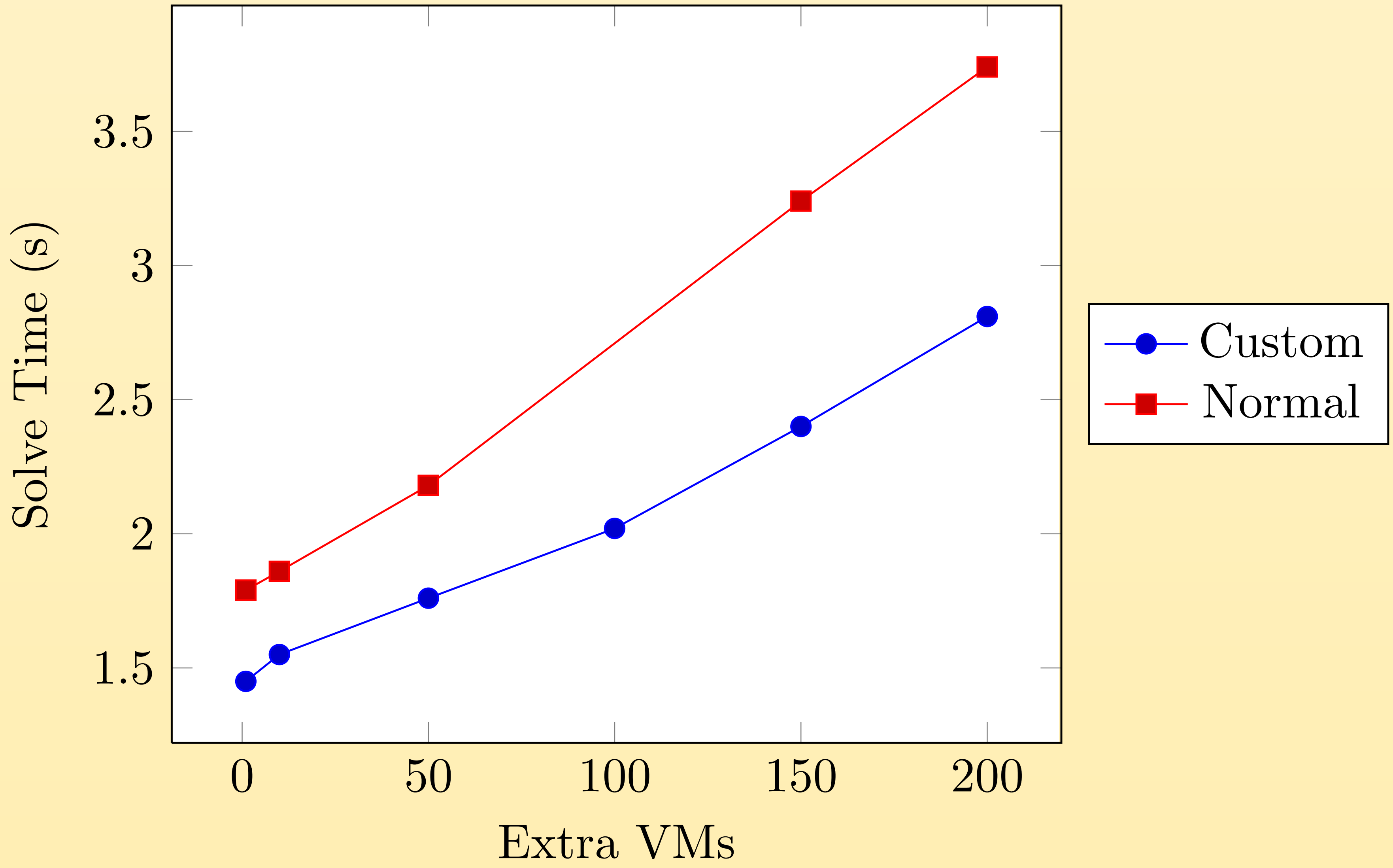
- "change" block is only valid when we have a previous configuration
- **~s** is the "previous" value
- this is a "hard constraint"
  - it could also have been a maximise/minimise constraint


Enterprise
Web
DHCP
Work
Work
Web
Work
DHCP
Web
Cloud
Work
Work
Work
Work
Work
Work
with “change minimisation”
no unnecessary migration

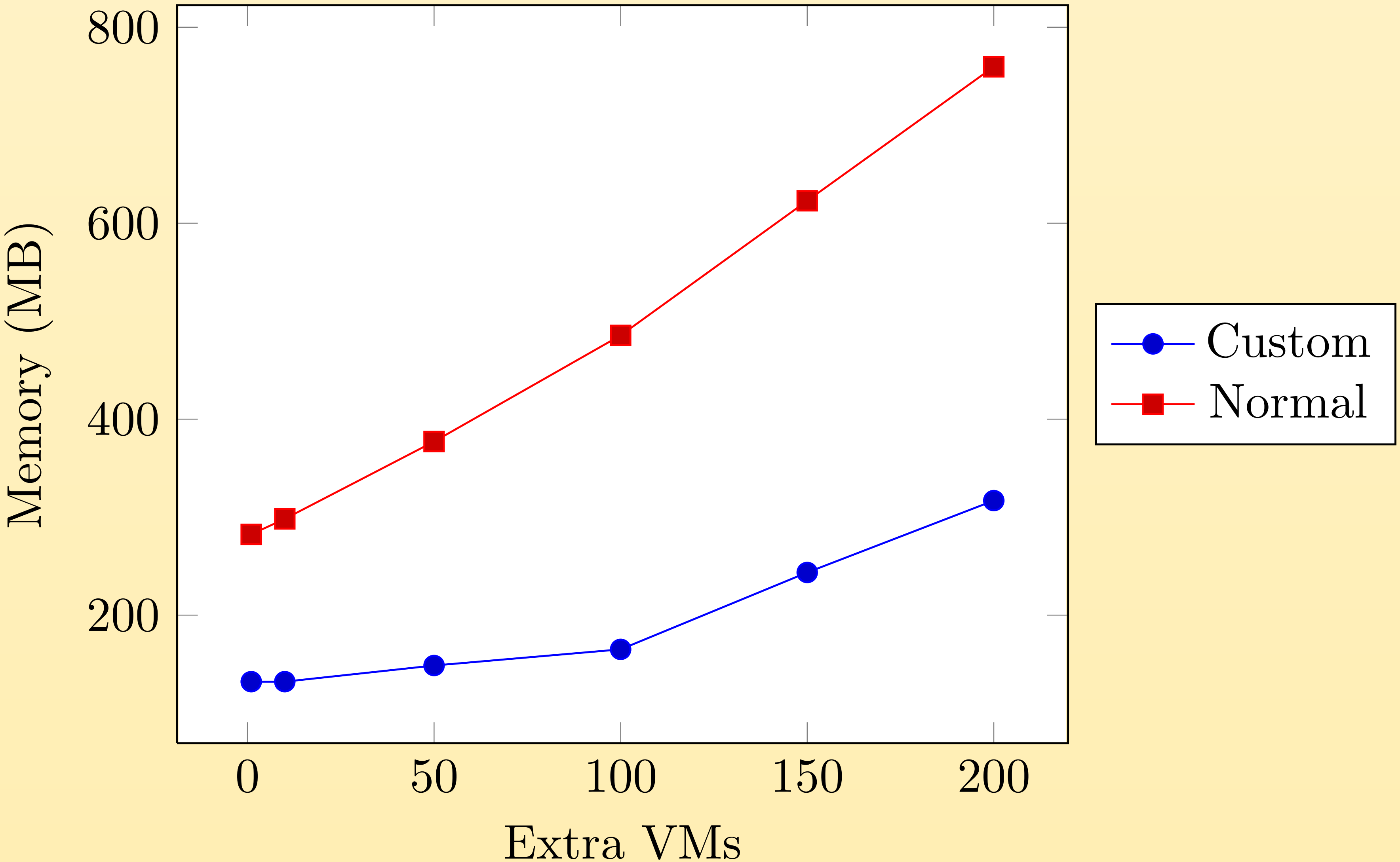
Reassignments



Time


3.5
3
2.5
2
1.5
Solve Time (s)
0
50
100
150
200
Extra VMs
Custom
Normal

Memory



800
600
400
200
Memory (MB)
0
50
100
150
200
Extra VMs
Custom
Normal

What's Good ?

Users can specify and change their own requirements completely independently

- and the resulting configurations are guaranteed to match the requirements

If some constraint changes, the system can automatically generate a new valid configuration (if one exists)

- things may change because of requirement changes
- or, for example, failures
- the deployment of the new configuration can be scheduled with automated planning tools

When the system reconfigures, it can do so with the minimum disruption necessary to meet the final requirements

What's Not So Good ?

It is very hard to specify comparative "costs"

- I could leave one service unnecessarily in the cloud, or I could move it back into the datacenter, but I would need to shuffle ten other servers to do so - which is best?

It is quite hard to avoid over-specifying or under-specifying constraints

- we either miss good solutions, or deploy bad ones

It can be hard for humans to predict the effects

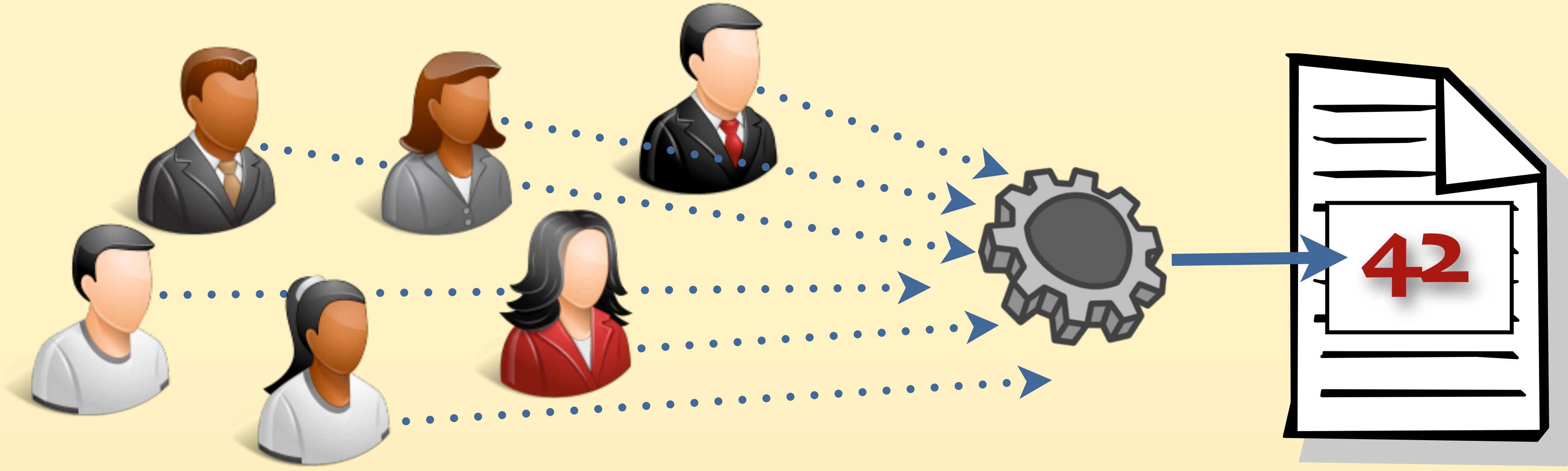
- sysadmins are very nervous with this degree of automation

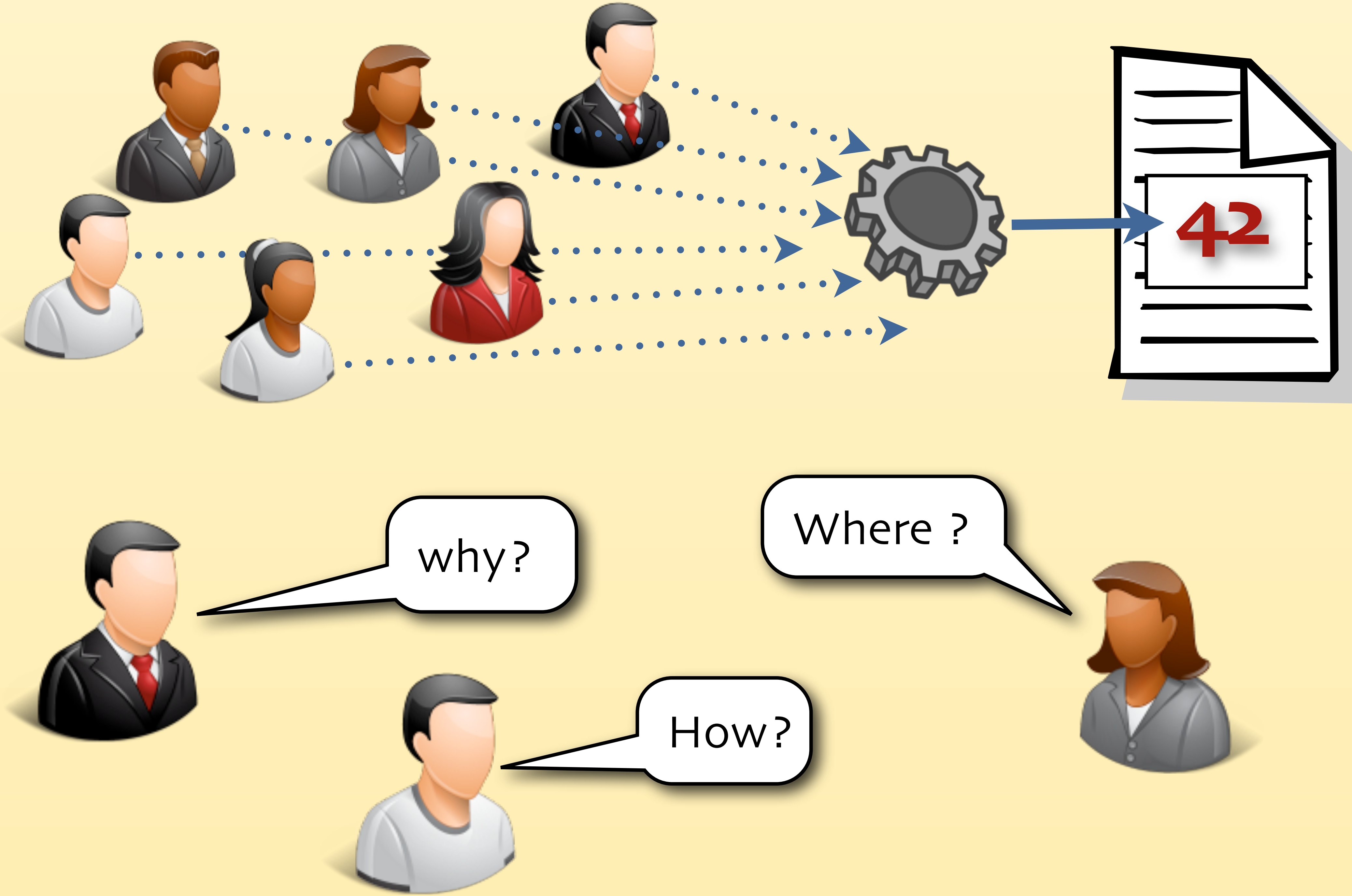
Sometimes there may be no solution

- and it is difficult to understand why

Performance can be unpredictable

- it is not always obvious what is computationally expensive


42


42
why?
Where ?
How?

Provenance

Who is responsible for the fact that service X is running in the cloud when it shouldn't be? !

- many people may have specified constraints contributing to this
- perhaps it was the fault of someone who said nothing at all!
  - i.e. there should have been a constraint preventing this

Who needs to fix it?

- and how?

We have started to look at provenance in configuration languages

- with James Cheney <jcheney@inf.ed.ac.uk>
  http://homepages.inf.ed.ac.uk/jcheney/

This is very complex when we allow full constraints

- but the problems exist in much simpler practical situations ...

Value Inheritance

Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@reliable.com
  ... 742 more parameters ...
}
class widgetServer isa genericServer {
  ...
}
class salesServer isa widgetServer {
  ...
   ...
}
node serverA isa salesServer {
   ip = 1.2.3.4
   ...
}
```

Alice Works For The Tool Vendor

Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@reliable.com
  ... 742 more parameters ...
}
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

- Alice develops generic templates
- this one is for a generic server
- it specifies the default "timeserver"
- this is set to some reliable public service

Bob Is The Senior Admin For widgets.com

Alice


Bob


Carol


Dave


```
class genericServer {
   timeServer = ts@reliable.com
   ... 742 more parameters ...
}

class widgetServer isa genericServer {
   ...
}

class salesServer isa widgetServer {

}
```

- Bob develops local templates
- these inherit from the generic ones
- Bob overrides some parameters
- but not the default timeserver

Carol Is The Admin For The Sales Dept

Alice

Bob

Carol

Dave

```
class genericServer {
```

- Carol inherits Bob's templates
- she overrides some parameters
- but not the default timeserver

```
}

class salesServer isa widgetServer {
  ...
    ...
}

node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave Is The Technician

Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@reliable.com
  ... 742 more parameters ...
}
```

- Dave configures the individual machines
- he assigns one of Carol's templates
- overriding a few machine-specific values

```
    ...
}
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Carol Adds A Local Timeserver

Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@reliable.com
  ... 742 more parameters ...
}

class widgetServer isa genericServer {
  ...
}

class salesServer isa widgetServer {
  timeServer = ts@sales.widget.com
  ...
}

node serverA isa salesServer {
   ip = 1.2.3.4
   ...
}
```

Alice Ships A New Template

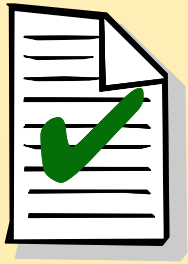
Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@unreliable.com
  ... 742 more parameters ...
}
class widgetServer isa genericServer {
  ...
}
class salesServer isa widgetServer {
  timeServer = ts@sales.widget.com
  ...
}
node serverA isa salesServer {
   ip = 1.2.3.4
   ...
}
```



Carol Withdraws Her Change

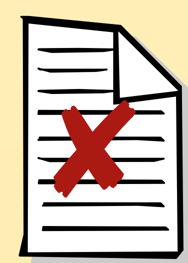
Alice

Bob

Carol

Dave

```
class genericServer {
  timeServer = ts@unreliable.com
  ... 742 more parameters ...
}
class widgetServer isa genericServer {
  ...
}
class salesServer isa widgetServer {
  timeServer = ts@sales.widget.com
  ...
}
node serverA isa salesServer {
   ip = 1.2.3.4
   ...
}
```



Whose "Fault" Is This?

Dave's server broke and he got the blame from the users

- in fact, all of the machines in the Sales Department are broken!
- but he says he didn't change anything at all

Carol says she just put the parameter back to the default

- so it can't be her fault - this is exactly the same as it was before

Bob says he carefully checked the new default configuration

- in fact, he ran some regression tests and the new configuration produced exactly the same results as the old one on all of the Sales Department machines

Alice says that she changed this default ages ago

- and it is up to the users to check these changes are appropriate
- although it is Alice's value which appears in the final configuration

Who Should Fix It? And How?

Alice probably isn't going to change this

- she presumably had a good reason for the new value
- and she doesn't work for us anyway, so she may break it again ...

Dave doesn't want to set it on his individual machines

- although he might do this as an interim fix!
- which will of course cause problems later, if it doesn't get removed

Carol just wants the same value as the rest of the company

- although she could make an interim fix too

But, it is probably Bob who needs to make a company-wide change ?

- even though he was not responsible for any of the changes which exposed the problem

Tracking Provenance Is Hard

We need to know who authored what

- relating source text diffs to semantic changes is not reliable

Every value must have a corresponding provenance expression

- the language needs a "provenance semantics" as well as the conventional "value semantics"
- there may be multiple different interpretations for different purposes

The provenance tends to be "explosive"

- "everyone had their fingers in this"
- we may need to evaluate (for example) both branches of a conditional

This needs to be implemented in the configuration compiler

Some Questions

Perhaps the history is important to understanding ?

- when Alice changed the default value, the configuration started to "smell bad", even though there was no immediate consequences
- even though the specification is entirely declarative, it may be useful to know "how we got here"

Perhaps we can assign some degree of "robustness" ?

- the above configuration is less robust in some sense, because it is more likely to break when things change
- is it right that things should break if I back out a change ?
- can I be warned when that situation is likely to occur ?

Provenance for a constraint-based language seems very hard

- can we still do something meaningful ?

Some Conclusions

Constraint-based (declarative) configuration languages seem promising

- they are capable of supporting the automatic composition of intersecting aspects
- but a fully-general constraint-solver is probably not appropriate for production use
- some human-factors research would be very useful to determine typical usage patterns which could be incorporated into a production language in a more usable way

A better understanding of configuration language provenance seems important

- for security
- and for debugging / problem fixing
- we may be able to learn from work in database provenance

More Conclusions

We need better configuration languages & implementations

- which support higher-level modelling
- and have clearer semantics
- and extensible implementations

Publications

A Declarative Approach to Automated Configuration
John Hewson & Andrew Gordon & Paul Anderson
Large Installation Systems Administration Conference (LISA '12)
`http://homepages.inf.ed.ac.uk/dcspaul/publications/lisa12.pdf`

Toward Provenance-Based Security for Configuration Languages
Paul Anderson & James Cheney
The 4th Usenix Workshop on the Theory and Practice of Provenance
`http://homepages.inf.ed.ac.uk/dcspaul/publications/tapp12-final15.pdf`

Modelling System Administration Problems with CSPs
John Hewson & Paul Anderson
The 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011)
`http://homepages.inf.ed.ac.uk/dcspaul/publications/ConfSolve-ModRef2011.pdf`

cisa | Centre for Intelligent Systems and their Applications

THE UNIVERSITY of EDINBURGH informatics

Questions

Use Cases

Comments

We are looking for a Phd student to work on provenance

Paul Anderson

dcspaul@ed.ac.uk

http://homepages.inf.ed.ac.uk/dcspaul

http://homepages.inf.ed.ac.uk/dcspaul/publications/google-2012.pdf