Centre for Intelligent Systems and their Applications

THE UNIVERSITY of EDINBURGH informatics

Provenance & Semantics in Configuration Languages

Paul Anderson

dcspaul@ed.ac.uk

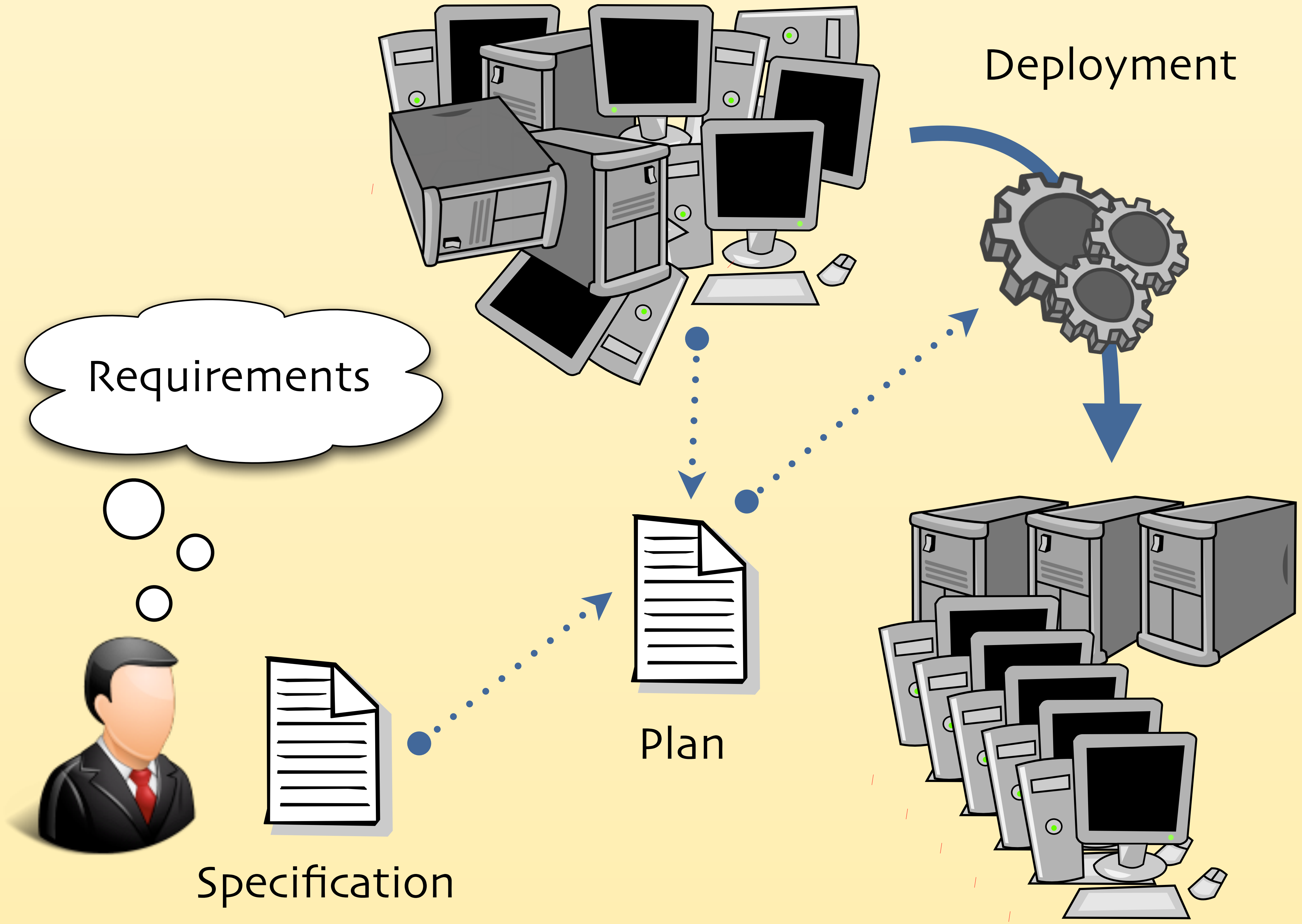
http://homepages.inf.ed.ac.uk/dcspaul

http://homepages.inf.ed.ac.uk/dcspaul/publications/newcastle1-2014.pdf

System Configuration

"Programming the infrastructure"

- corporate IT infrastructure, "grid", "datacentre", "cloud service", distributed application, ...
- virtual machines & networks mean that everything is now "soft"


Requirements
Specification
Plan
Deployment

A Traditional Approach

The traditional approach is to use “imperative” scripts

- these are created by a human to implement a workflow which they have designed to achieve the desired state
- workflows may run in response to “events” (eg. a failure)

But ..

- there is no often explicit specification of the desired state
  - even if there is, it is not easy to prove that the workflow achieves it
- a new workflow is needed for every new initial state
  - and/or the workflow includes complex hand-coded conditionals
  - for use in autonomic recovery, the number of possible states is large

A Declarative Approach

We advocate a more "declarative" approach

- the human specifies the desired state
- a monitoring system determines the current state
- a planner automatically creates a workflow
- a deployment engine executes this and validates the result

So ..

- the user provides (only) a specification of the final, desired state
  - and possibly some declarative constraints on the intermediate states
  - this is clearly separated from the actions required to achieve it
- the system can achieve this state from any starting point
  - if this is possible
- we can prove properties of the final (and intermediate) state

Configuration Languages

Imperative configuration uses conventional scripting languages

- or a DSL with a roughly equivalent power
  - they describe the process (computation) of changing the configuration

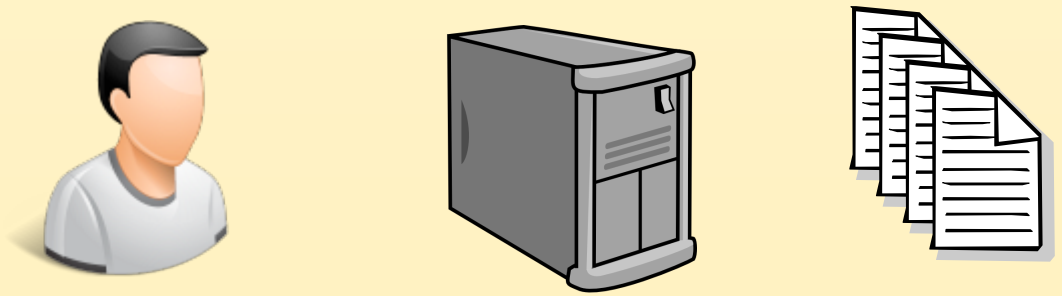
Declarative configuration languages are quite different

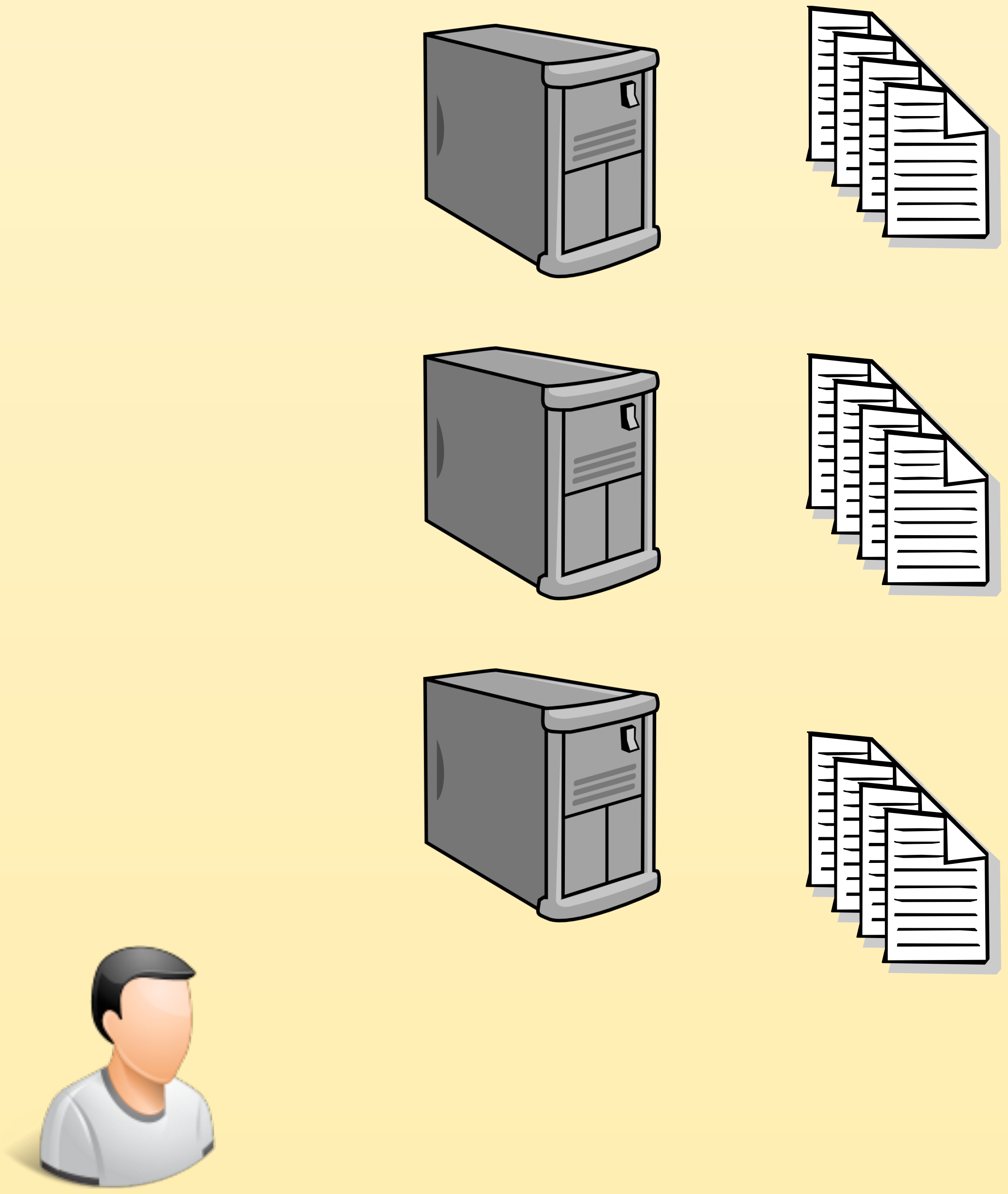
- they describe the desired state - not a computation
  - in theory, they should have a simpler semantics
  - and be easier to reason about
- they describe the requirements at a higher level
  - these are translated into explicit, detailed configuration parameters
- they compose the requirements from many independent people
  - the declarative nature allows us to do this composition …
- the deployment of the configuration is a separate problem
  - we won't cover that here

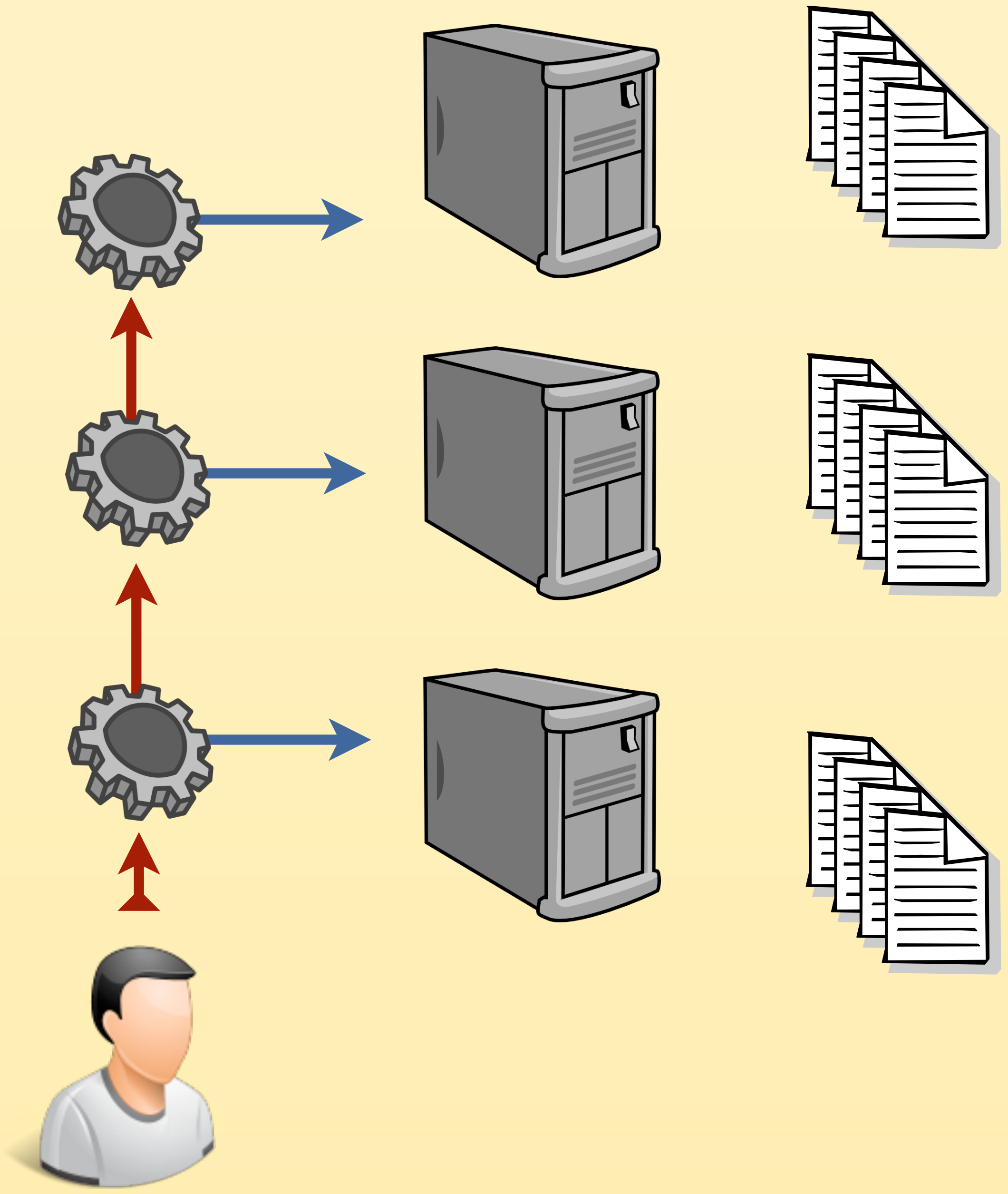
Aspects & Composition

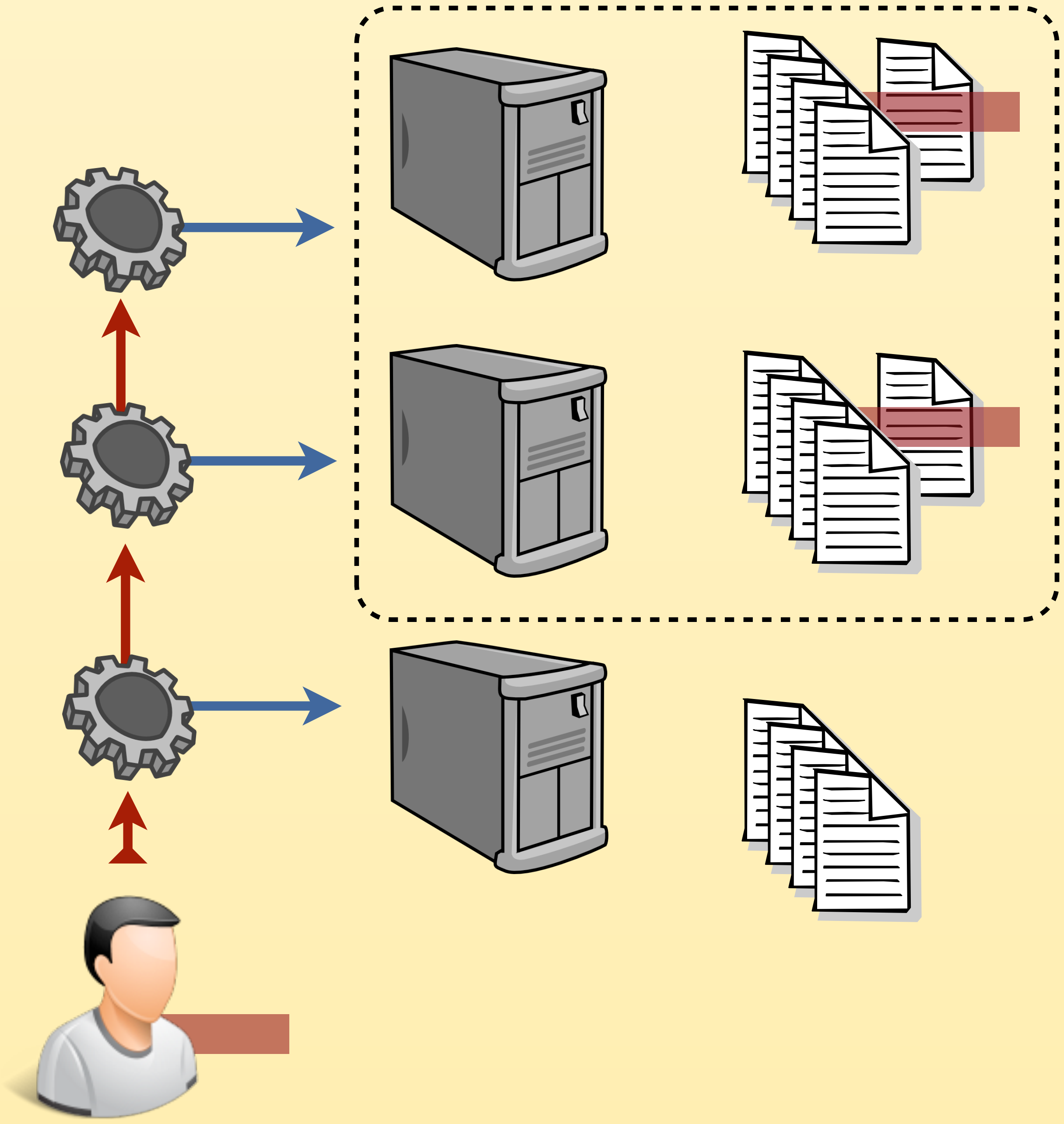
We are going to talk about this feature of configuration languages

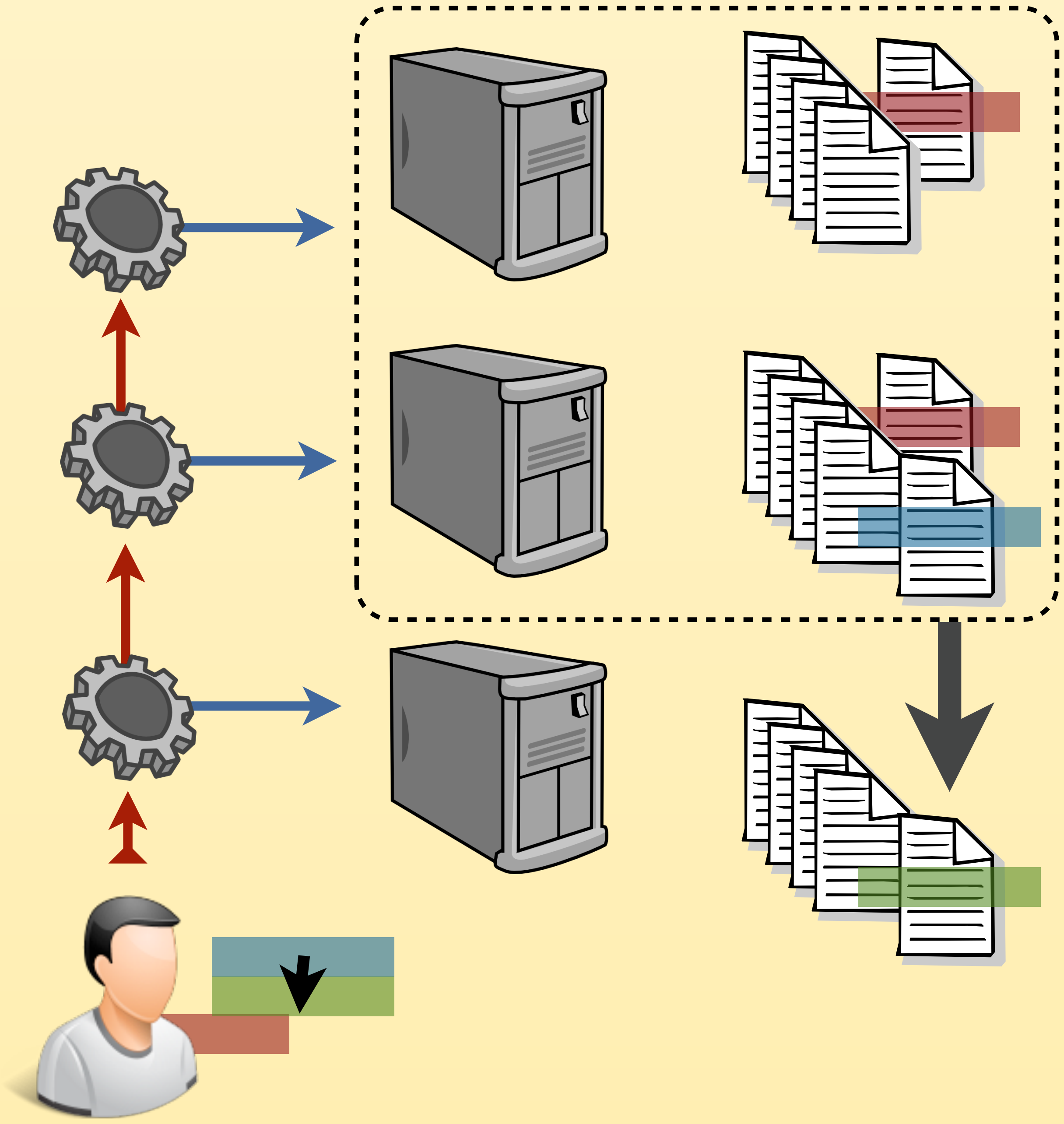
- which has no real equivalent in most programming languages

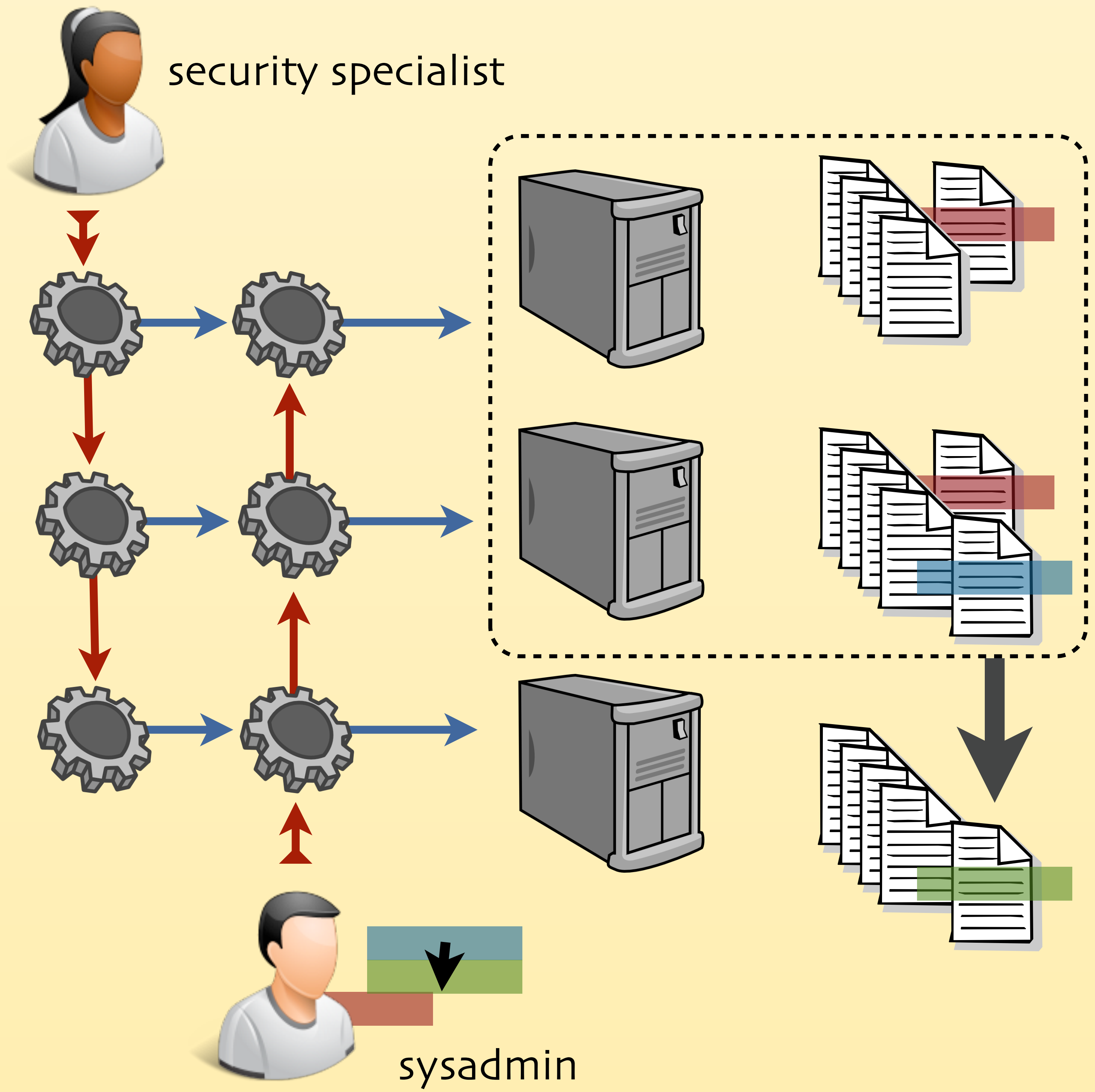


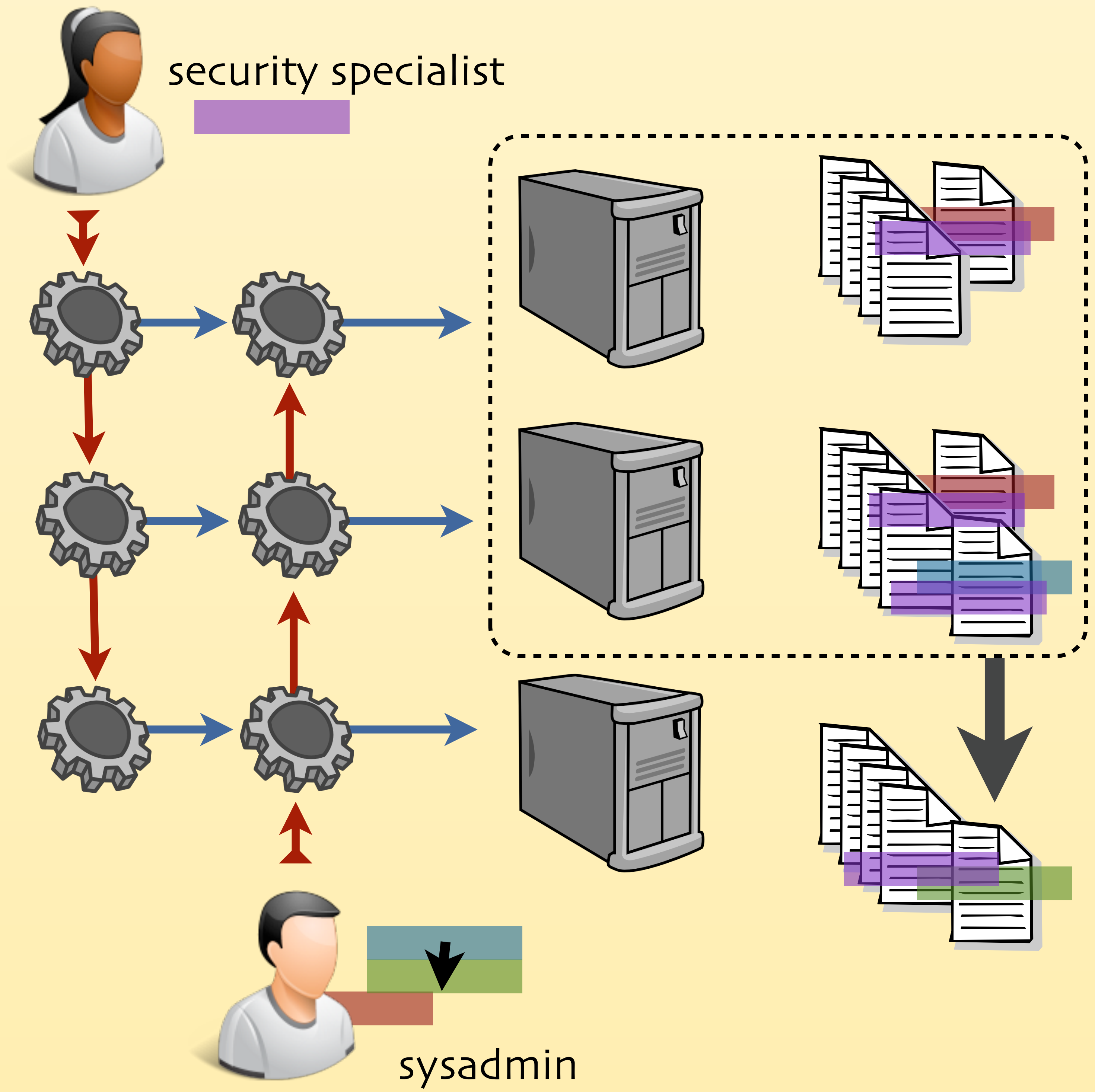


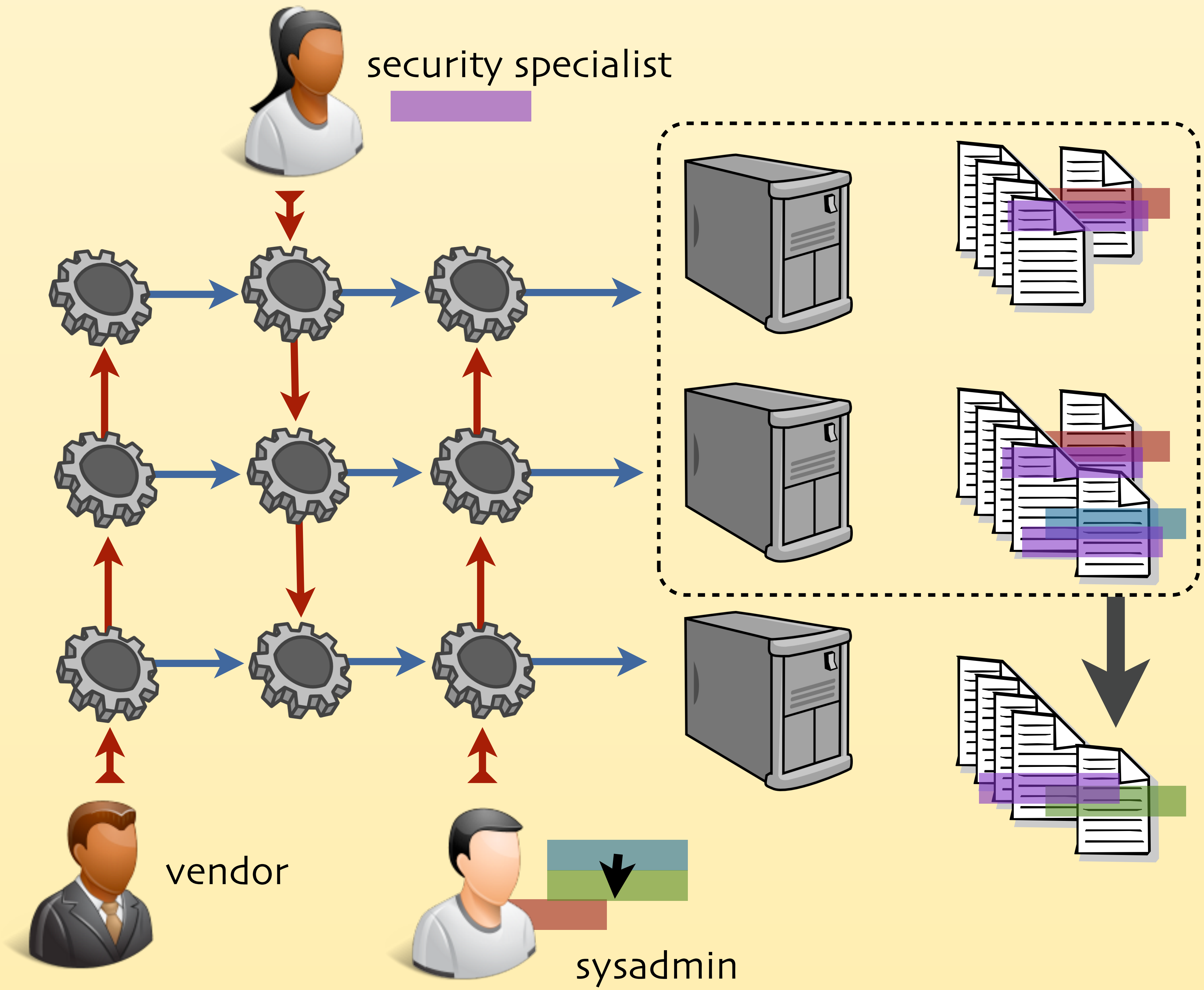


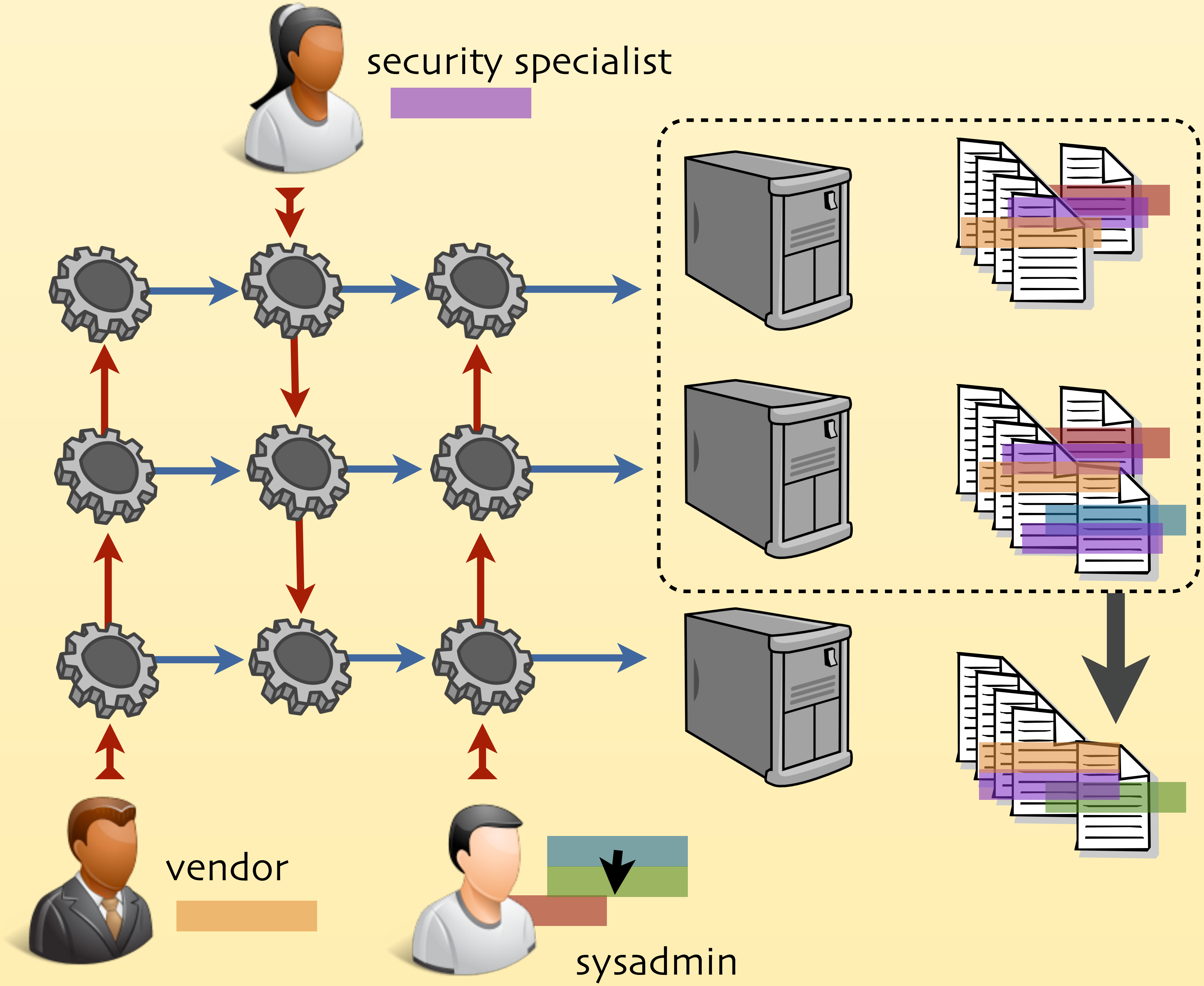


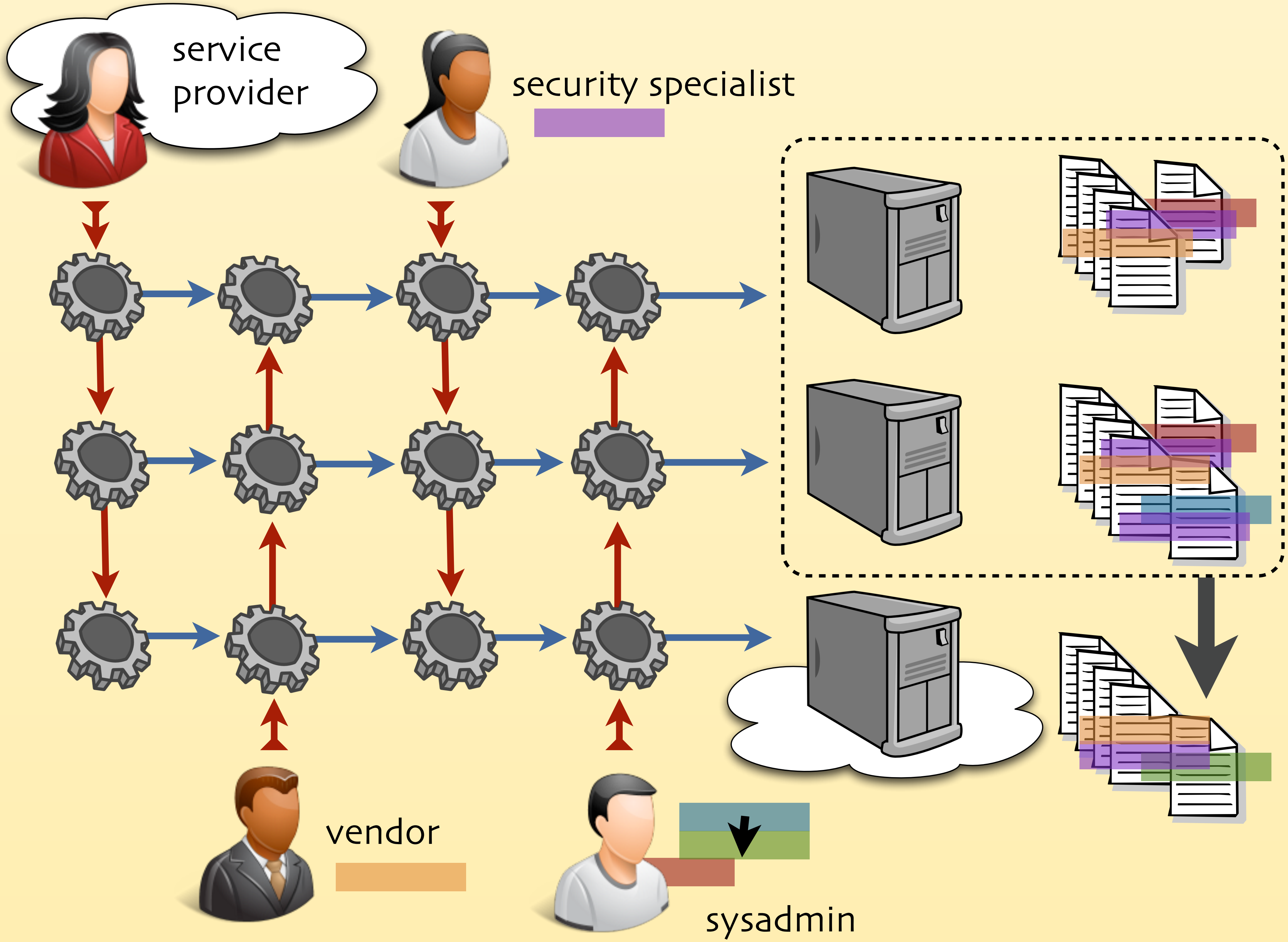



security specialist
sysadmin


security specialist
sysadmin


security specialist
vendor
sysadmin


security specialist
vendor
sysadmin


service provider
security specialist
vendor
sysadmin

Aspect Composition

Many different people are responsible for different "aspects"

- one of our goals for a configuration language is to help people collaborate & compose their requirements without unnecessary conflict
- A configuration tool composes the independent "aspects" to form a consistent specification

Different tools support different languages and approaches

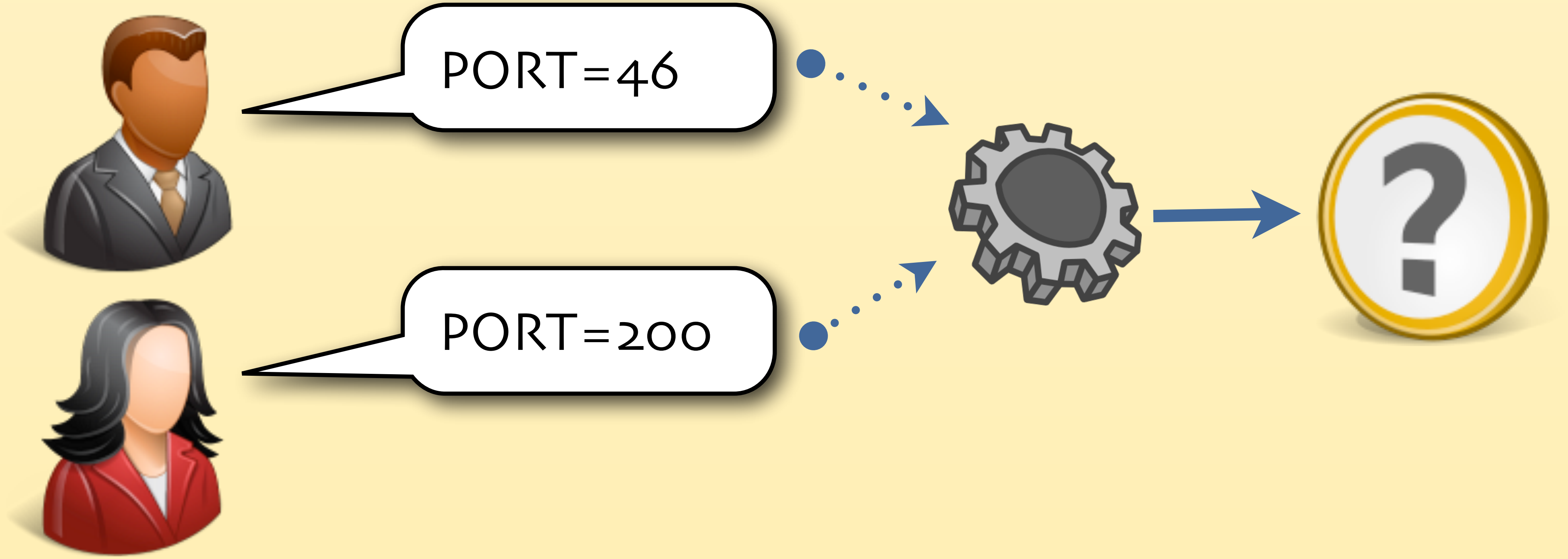
- "prototypes" and "instance inheritance" are common
- simple order precedence
- explicit composition functions

Arbitrary constraints

- 'ConfSolve" supports arbitrary constraints …

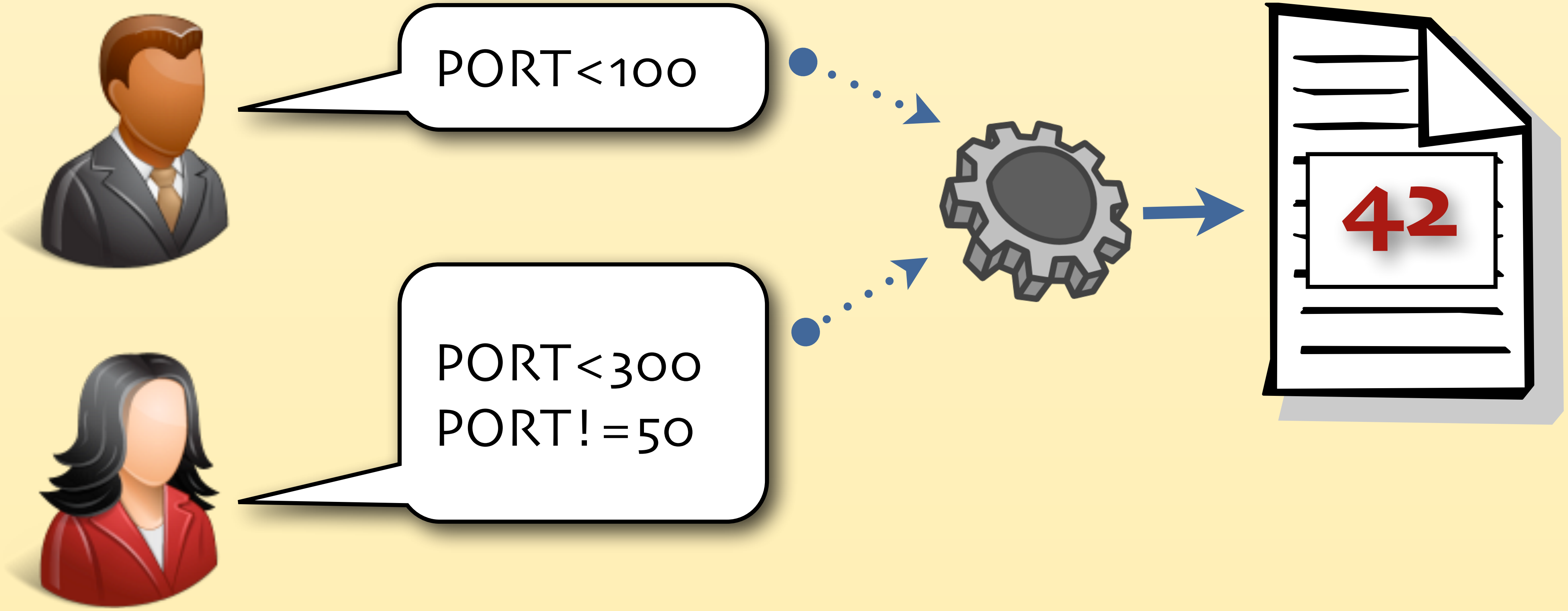
People's real requirements are often quite loose:

- "configure one machine as a web server" (but I don't care which)
- but most systems force the user to specify an arbitrary value



With a declarative approach, we can specify loose constraints ..

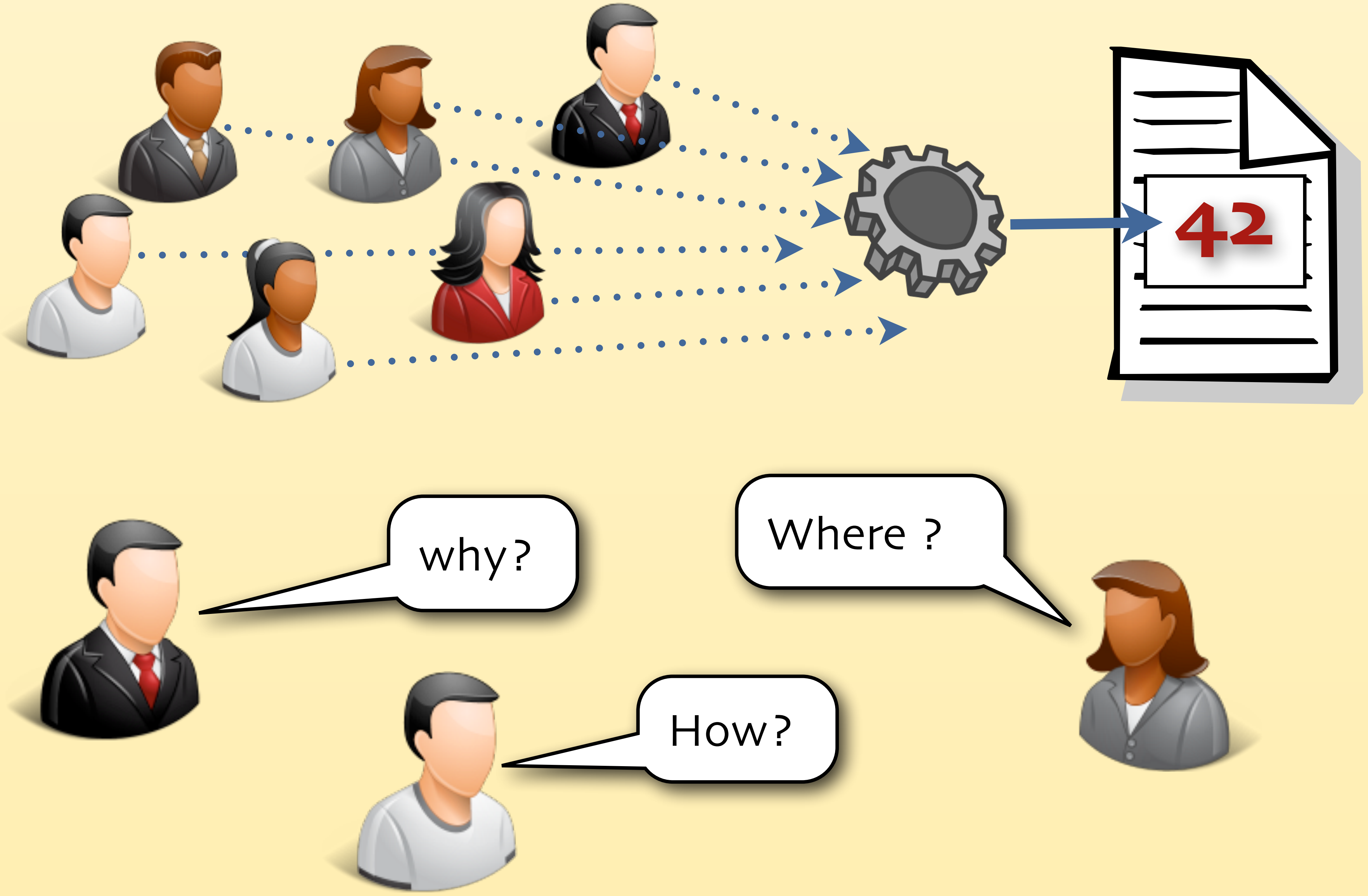
- this allows us to compose aspects without conflict or unnecessary negotiation



Provenance

The “provenance” of the resulting configuration is not clear

- the composition process is complex
- who was “responsible” for what?


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 rules 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

Were they all authorised to specify this?

Who needs to fix it?

- and how?

Does this have analogies with provenance issues in databases?

- James Cheney <jcheney@inf.ed.ac.uk> & I would like to explore this
- we have a Microsoft Phd award for this topic

A Typical Problem ...

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 ...
}

c


}

c




}

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 {



}
n


}
```

- 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 {



}

c

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

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

Dave Is The Technician

Alice

Bob

Carol

Dave

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

}
c

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

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

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

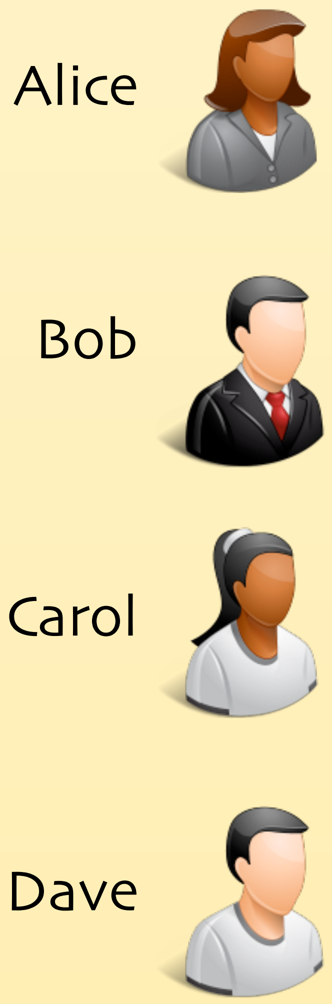
Bob

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



```
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

Provenance Semantics

A work in progress!

An Example …

{Alice}	`X=2`
{Bob}	`Y=3`
{Carol}	`if X==2 then`
{Dave}	`  Y=4`
{Carol}	`else`
{Erin}	`  Y=5`
{Carol}	`fi`

The value of Y is 4

Because Dave said so

But Alice had a say in this

If she changed her line, the result would be different

So did Carol

P = {D,A,C} ?

But what about Erin?

If her value was 4, then it would no longer matter what Alice said!

Some Research Questions ...

Can we provide a provenance semantics in parallel to the value semantics?

- could this help us to solve problems such as the preceding example?
- will this help us to design better configuration languages?

What are the values?

- a set of people? a more complex expression?

Is the history 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"

More Research Questions ...

Perhaps we need multiple notions of provenance for different purposes?

- using the result for security (allow/disallow changes) ?

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 ?

Is it possible to assign a meaningful provenance to existing configuration languages ?

- or do we need new languages ?
- perhaps the provenance is always “explosive”

Practical Issues

We need to create new compilers

- we need to explore both branches of conditionals, for example

A special-purpose editor may be necessary/helpful

- we need to attribute semantic changes - not just syntactic ones
- Line-based attribution is not sufficient for most languages

Some Preliminary Work

We have been looking at formal (value) semantics for some configuration languages

- "ConfSolve" (Hewson)
- SmartFrog & Nuri (Herry)

We would like to work with real production languages (Puppet?)

- it is important to understand how the features are used in practice
- but these usually have very informal semantics (and even syntax)
- and they often include imperative constructs & other pragmatics

We have been analysing historical configuration data in LCFG

- we have large historical repository (CVS)
- a simple language with line-based syntax
  - makes attribution easier

cisa Centre for Intelligent Systems and their Applications

THE UNIVERSITY *of* EDINBURGH informatics

Provenance & Semantics in Configuration Languages

Paul Anderson

dcspaul@ed.ac.uk

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

http://homepages.inf.ed.ac.uk/dcspaul/publications/newcastle1-2014.pdf