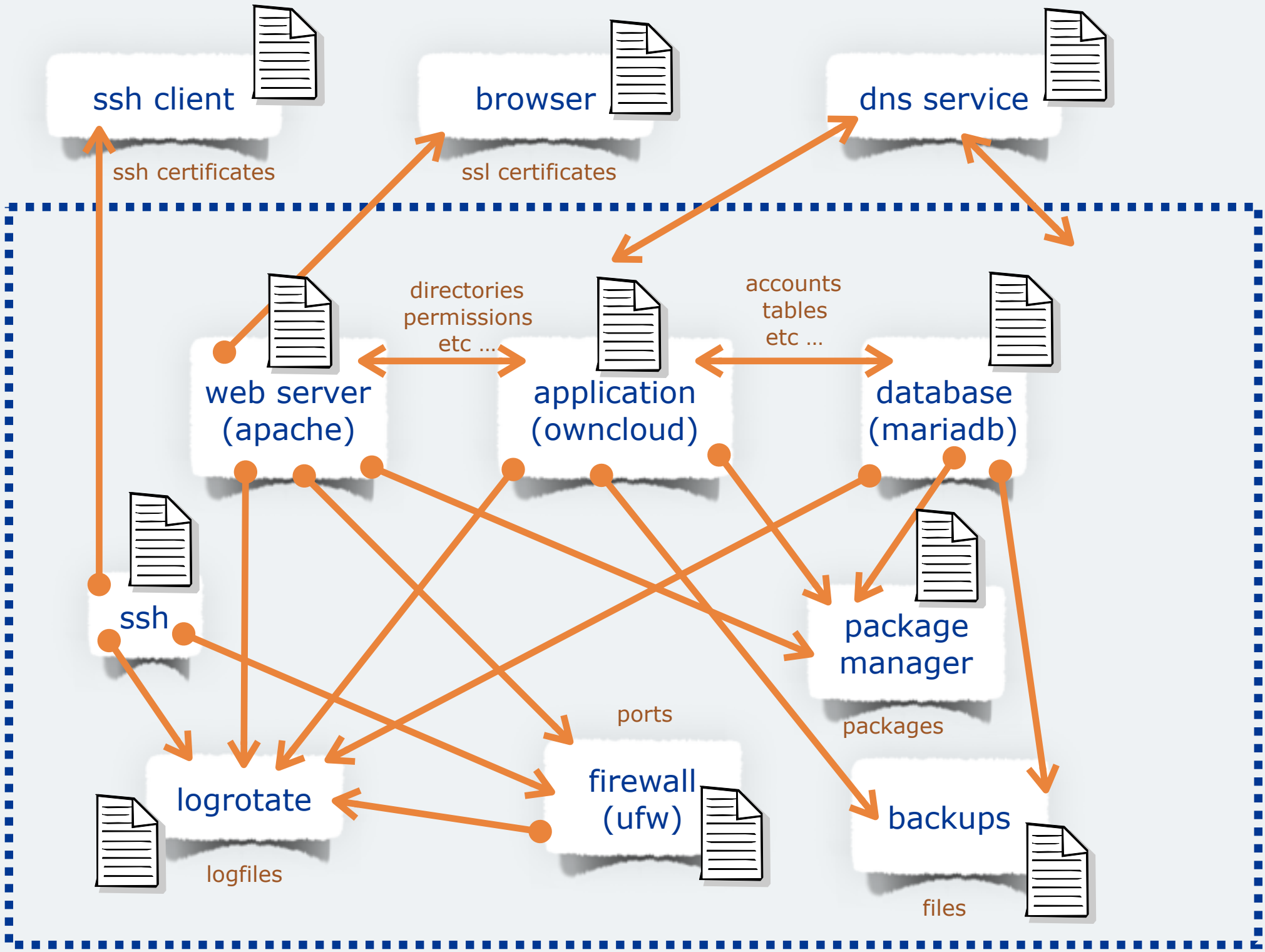
THE UNIVERSITY of EDINBURGH
informatics

Some thoughts on
Composition & References
in declarative configurations

Paul Anderson

dcspaul@ed.ac.uk

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

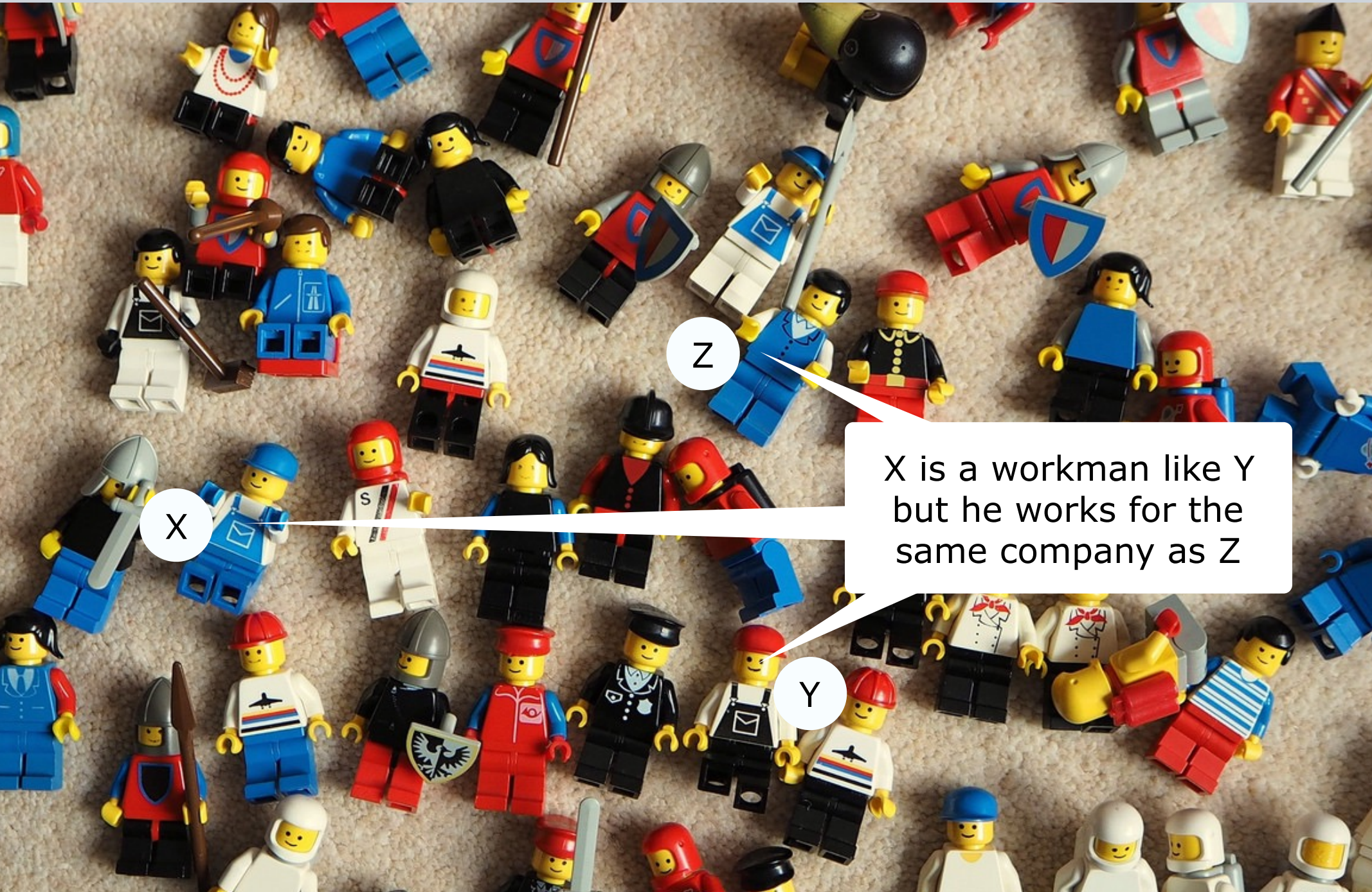

ssh client
browser
dns service
ssh certificates
ssl certificates
directories
permissions
etc ...
accounts
tables
etc ...
web server
(apache)
application
(owncloud)
database
(mariadb)
ssh
package
manager
packages
ports
logrotate
logfiles
firewall
(ufw)
backups
files


https://www.flickr.com/photos/foilman/15844421582

Declarative configuration

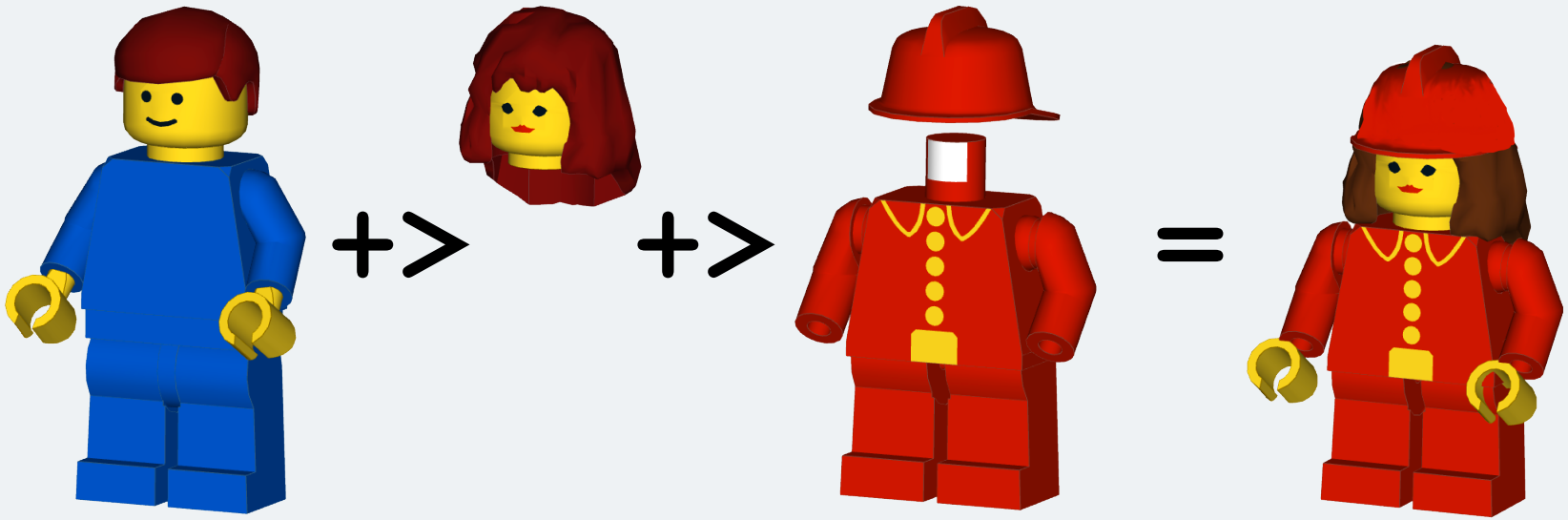
```
figure: {
 head: {
  face: "male"
  hair: {
   style: "short"
   colour: “brown"
  }
  hat: none
 }
 clothing: {
  ...
 }
}
```

Composition



Specialisation (instance inheritance)

This is a typical operation ...



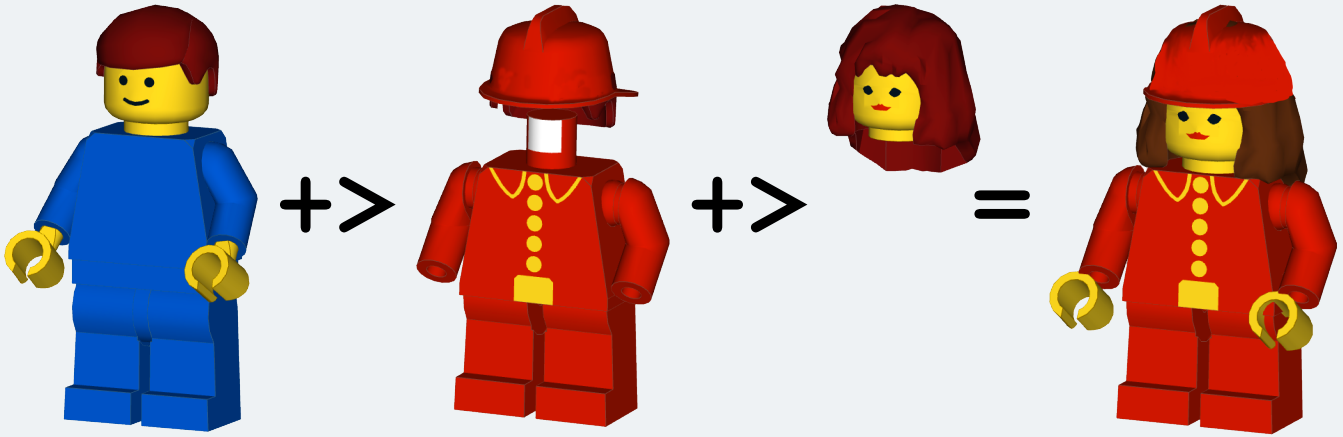
Or ...

"I want a Redhat Linux machine running Apache and Wordpress"

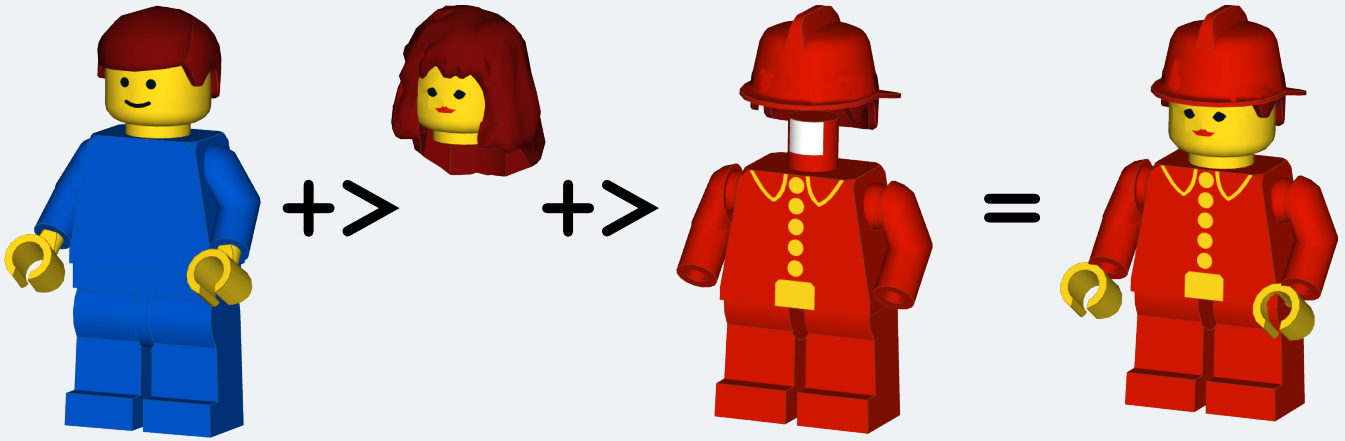
This works fine if the "aspects" are disjoint

Conflicts

"*Hair will not extend beyond the bottom of the earlobe*"
International Association of Women in Fire and Emergency Services
http://bit.ly/1Jt0Mz5



A female firefighter ?

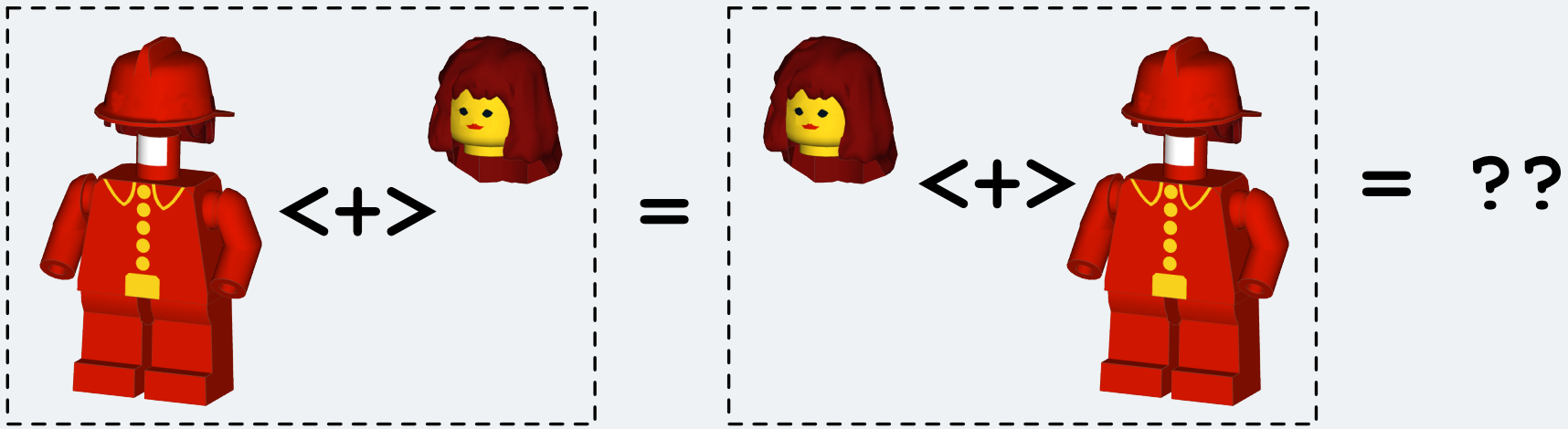


Or a firefighting female ?

Commutative composition

The “user” is forced to make this decision

- But they don’t usually have the information to do this
- And neither order may be correct if there are multiple conflicts!



The user needs a commutative composition operation

- And the authors of the components need to specify how they should be composed

Resolving conflicts

What do we mean when we specify a value for a resource ?

- *"The value really must be 42".*
- *"I don't really care what the value is, but I can't leave it empty, so I'll give it the value 0".*
- *"36 would be a good value, but I don't care if someone else would rather have something different".*
- *"I think it should be 46, but if Jane thinks it should be different, then believe her".*
- *"The value must be between 100-200, but I can't specify a range, so I'll say 150".*

Tags & constraints

In “L3”, we can tag resource values ...

```
a: { colour: "red" #aliceSays }
b: { colour: "blue" #bobSays }
```

And we can specify precedence between the tags

```
c: ( $a <+> $b ) #aliceSays >> #bobSays
d: ( $a <+> $b ) #aliceSays << #bobSays
```

This supports requirements such as ...

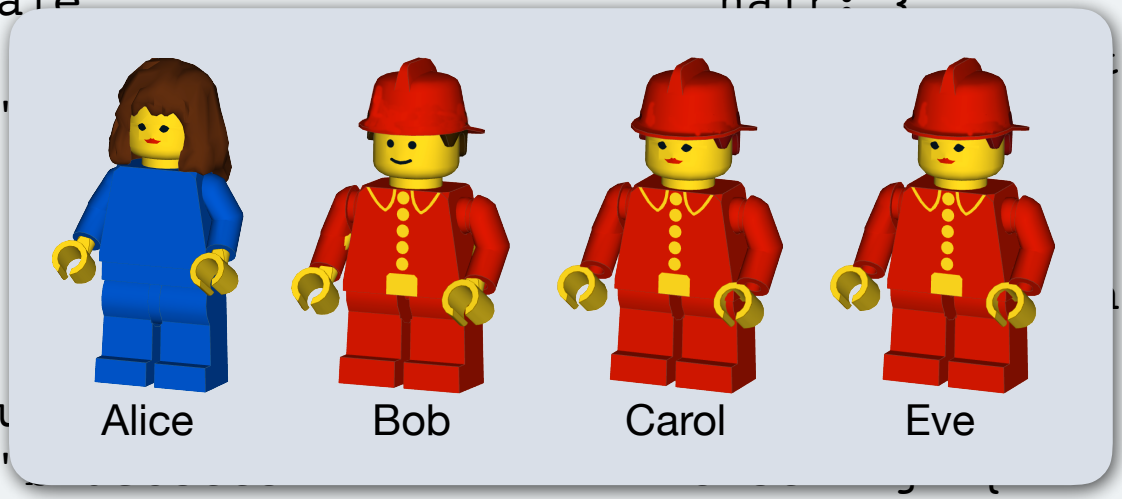
“I think it should be 46, but if Jane thinks it should be different, then believe her”.

“Parameters specified at a departmental level should override those set at a corporate level”.

Composition example

```
figure: {
 head: {
  face: "male"
  hair: {
   style: "
   colour:
  }
 }
 clothing:
  top: "blu
  bottom: "
 }
} #default

female: $figure <> {
 head: {
  face: "female" #final
  hair: {
   style: "long"
  }
 }
}
```

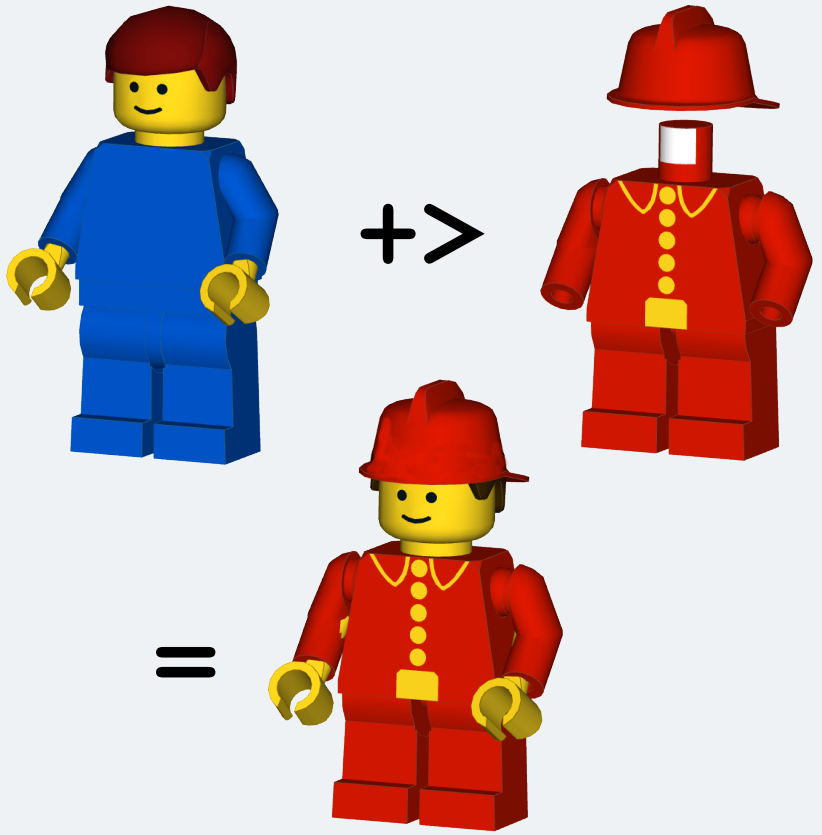



```
fireperson: $figure <> {
 head: {
  hair: {
                    "
                    t"
  top: "firetop"
  bottom: "redbottom"
 }
} #final

alice:   $female
bob:     $fireperson
carol:   $female <> ^fireperson
eve:     $fireperson <> ^female
```

Specialisation

```
fireperson:
$figure +> {
    head: {
      hat: "firehat"
    }
  }
  clothing: {
    top: "firetop"
    bottom: "redbottom"
  }
}
```



This can now be defined in terms of composition ...

```
(X +> Y) ≡ (X #tag1 <+> Y #tag2) #tag1 << #tag2
```

References

References

Some motivations for references …

- "Cloning" prototypes (usually to be specialised)
- Ensuring consistency between related resources

```
bob:    $fireperson
carol:  $female <+> $fireperson
eve:    $fireperson <+> $female

service: { port: 45; … }
server:  { port: $service.port; … }
client:  { port: $service.port; … }
```

Absolute references are unambiguous

- But there are different possible semantics for relative references
- The interaction with composition is "interesting"

Relative references

In this example, neither a purely “late” binding of the references, nor a purely “early” binding yields the “obviously” expected result:

```
service: {
  port: 25 #default
  client:  { port: ^^port, ... }
  server: { port: ^^port, ... }
}

myservice: ^service +> { port: 26 }

machineA: ^myservice.client +> { ... }
machineB: ^myservice.server +> { ... }
```

Disambiguating references

We could provide multiple types of reference

- LCFG has “early” and “late” references with different notations
- This is error-prone and very difficult for the user to get right

Humans are used to disambiguating references

"Divorcee and former air hostess Zsuzsi Starkloff talks on camera for the first time about her relationship with Prince William of Gloucester, the Queen's cousin and pageboy at her wedding”

(The Independent newspaper, Thursday 27th August 2015)

L3 currently has an experimental semantics

- Using composition to disambiguate multiple possible reference interpretations …

References in L3

```
service: {
  port: 25 #default
  client:  { port: ^^port, ... }
  server: { port: ^^port, ... }
}

myservice: ^service +> { port: 26 }

machineA: ^myservice.client +> { ... }
machineB: ^myservice.server +> { ... }
```

We compose all of the possible interpretations ...

`machineA.port = (25 #default) <+> 26 <+> null`

That's it ...

No coherent language proposal (yet)

- Just thinking about these kind of features, and ...

Issues ...

- Basic semantics
  - e.g. tag & constraint inheritance
- Usability
  - do we get unexpected results?
  - how easy is it to express existing configurations?
  - simplicity vs expressiveness
- Additional features
  - block parameters
  - lists, composition, ordering, map, filter, etc ..
- Implementation
  - Evaluation & efficiency