Recursive inventory management

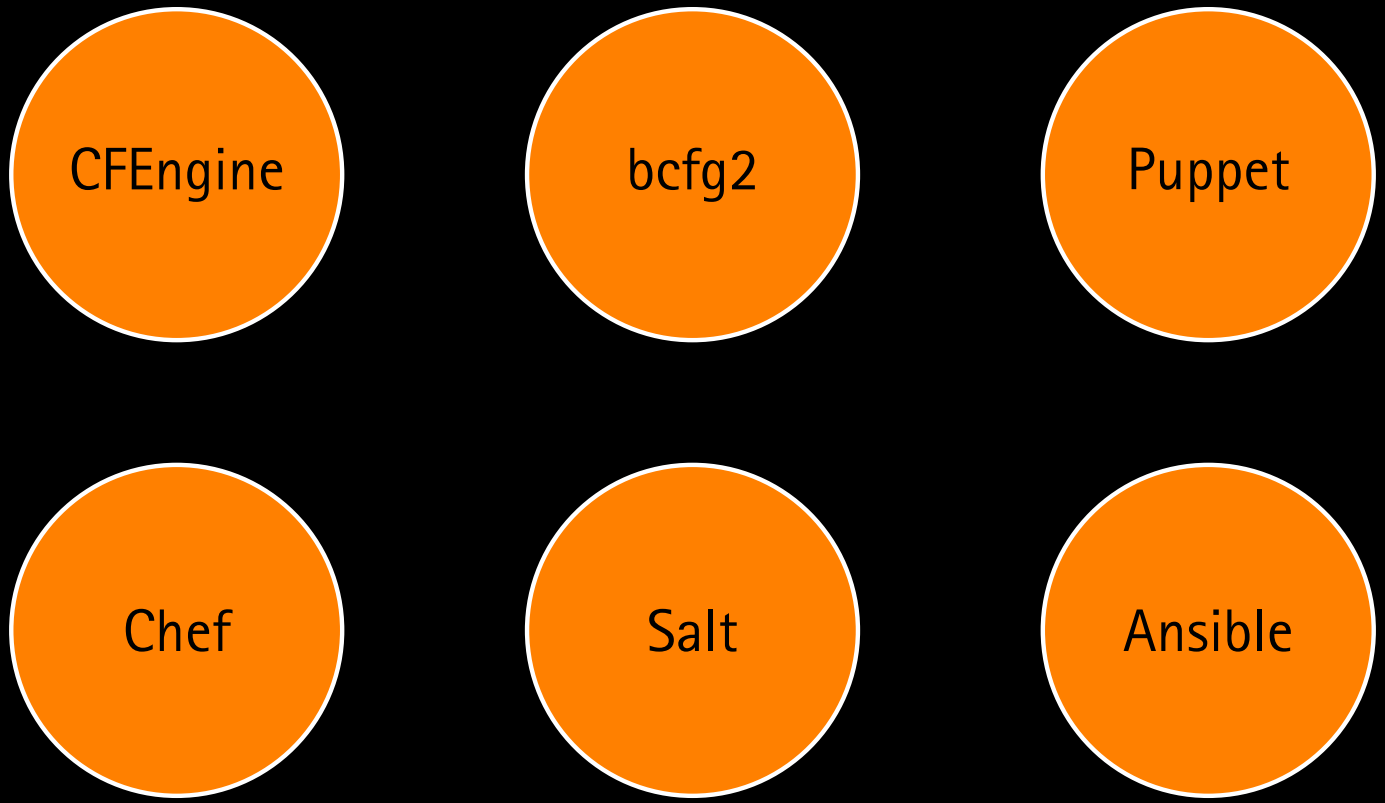
Martin F. Krafft
madduck@debian.org

13 Aug 2013 @ DebConf 13, Vaumarcus, Switzerland

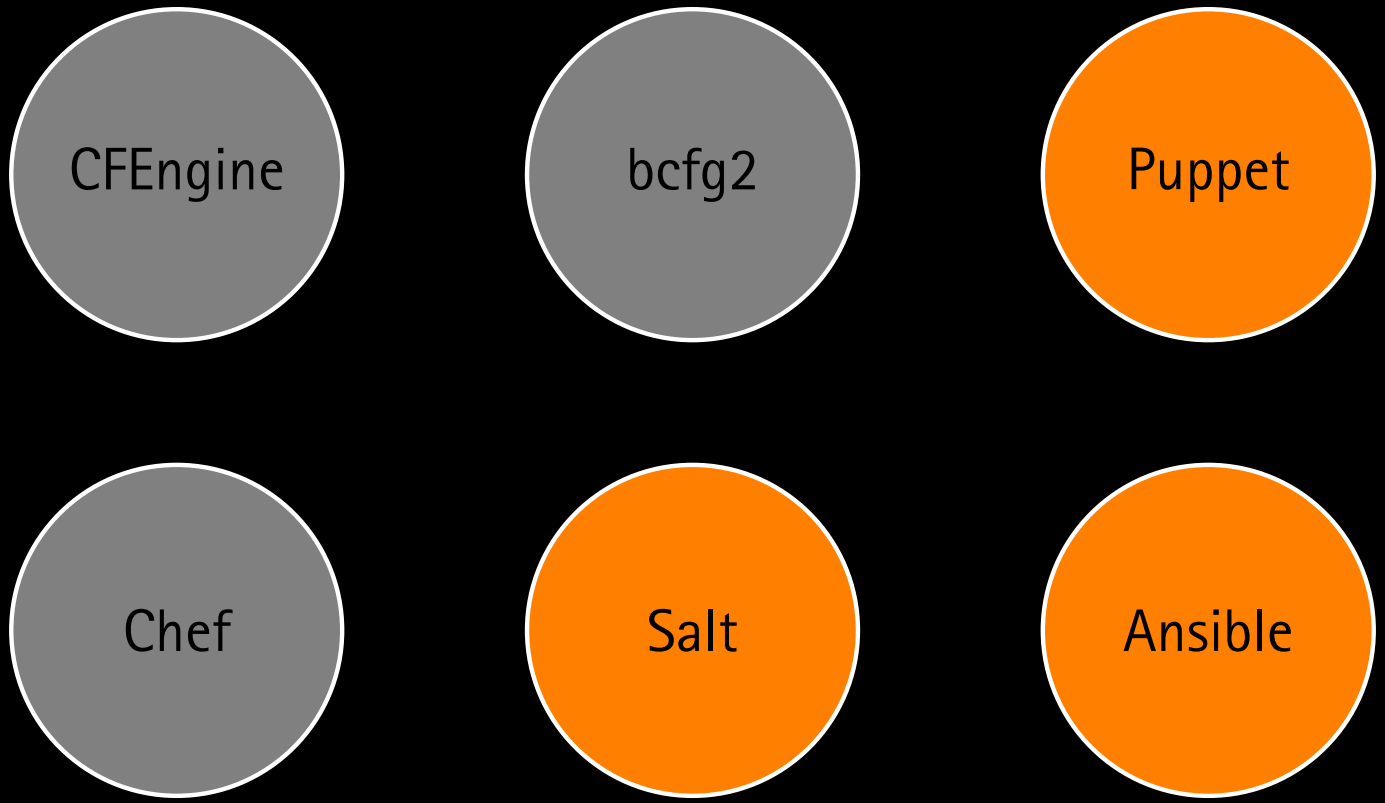
Configuration management

(system administration)

Configuration management



Configuration management



Two approaches

- Cloud provisioning
- System administration (classical)

Configuring the cloud

- Unprecedented ease!
- Scalability
- „Orchestration“
- *Ad-hoc* provisioning
- Homogeneous

Classical system administration

Classical system administration

- Longevity

Classical system administration

- Longevity
- Heterogeneous

Classical system administration

- Longevity
- Heterogeneous
- Themed hostnames, not canonical names and numbers

Classical system administration

- Longevity
- Heterogeneous
- Themed hostnames, not canonical names and numbers
- Laziness (*vs.* unprecedented ease)

Classical system administration

- Longevity
- Heterogeneous
- Themed hostnames, not canonical names and numbers
- Laziness (*vs.* unprecedented ease)
- Orchestration

Configuration management with reclass

- reclass comes from classical system administration

Configuration management with reclass

- reclass comes from classical system administration
- It might well suit your cloud needs

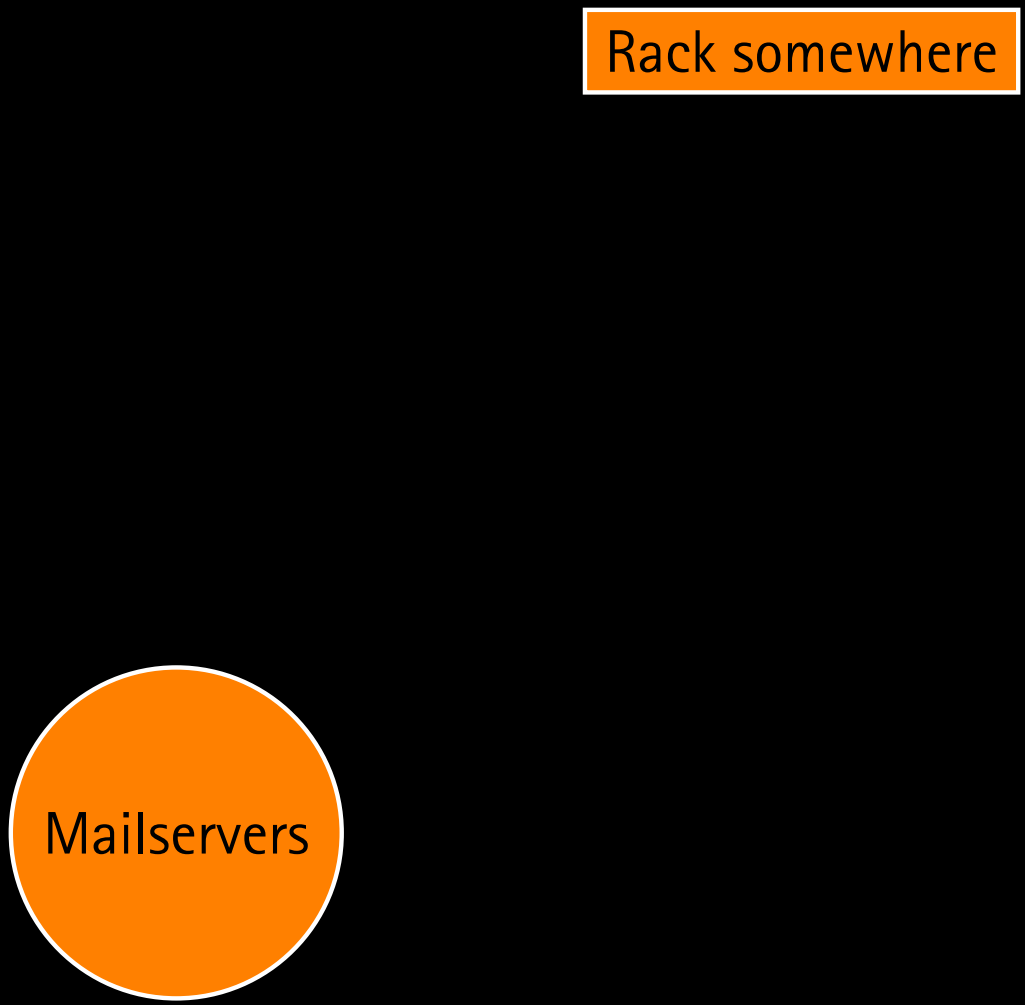
Configuration management with reclass

- reclass comes from classical system administration
- It might well suit your cloud needs (might require rethinking)

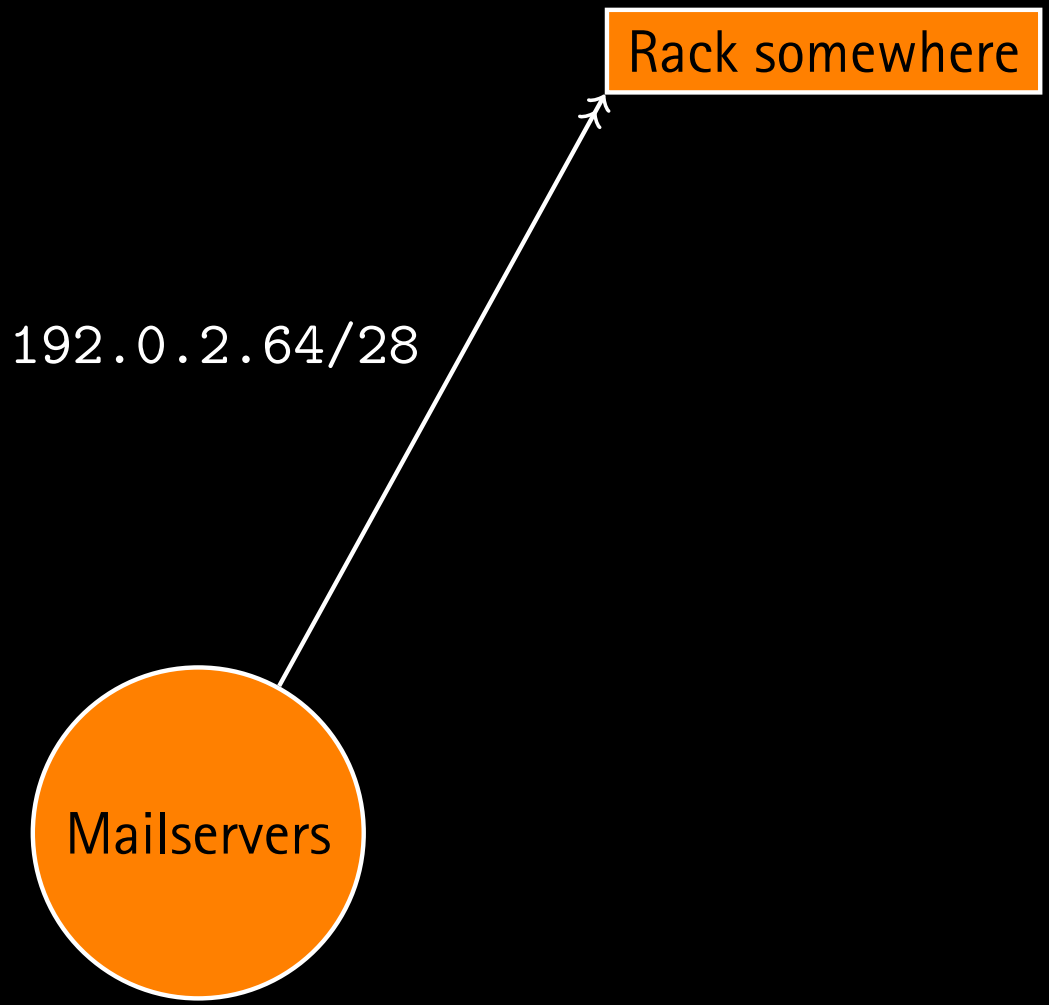
Targeting nodes
vs.
classifying hosts

Targeting nodes

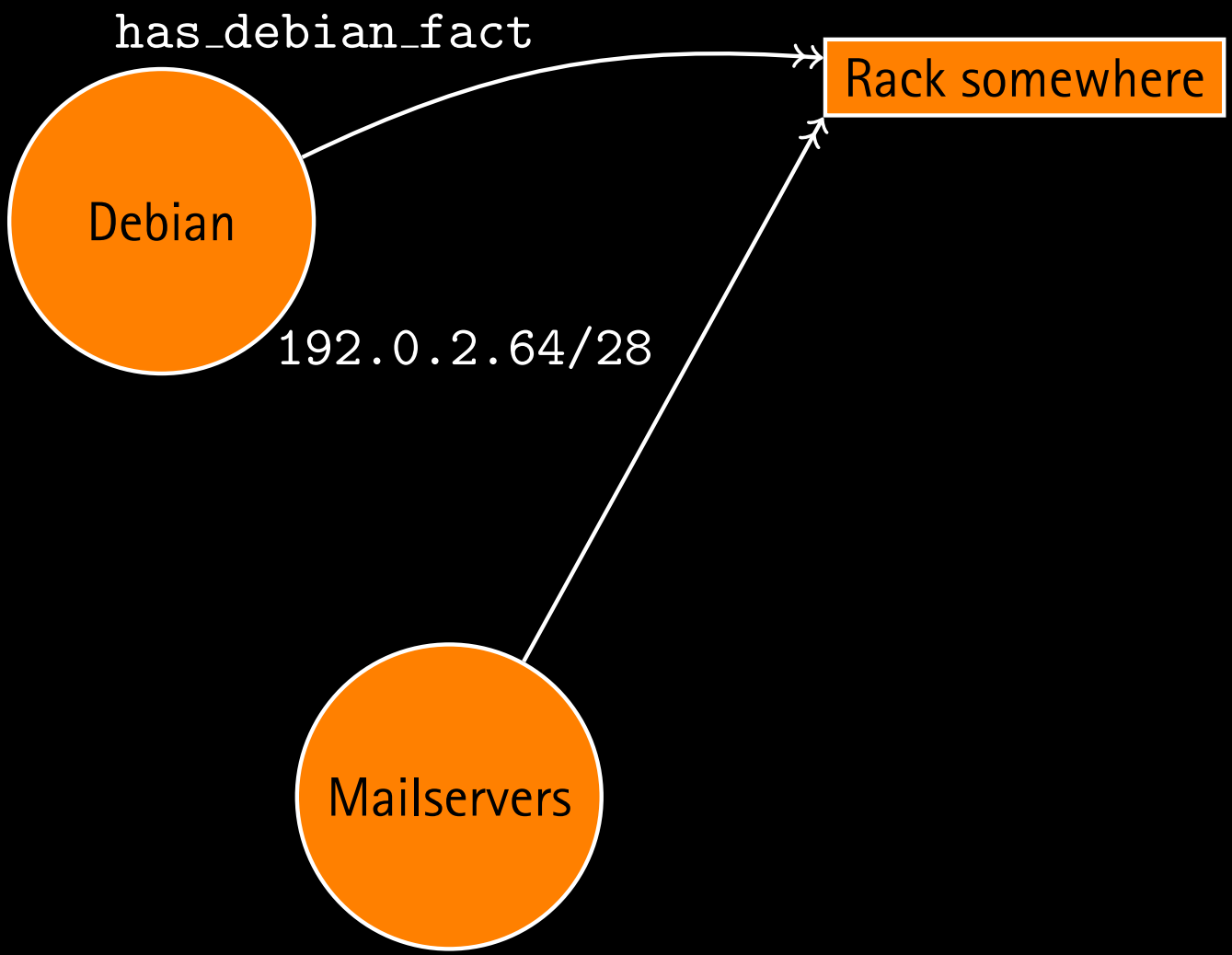
Targeting nodes



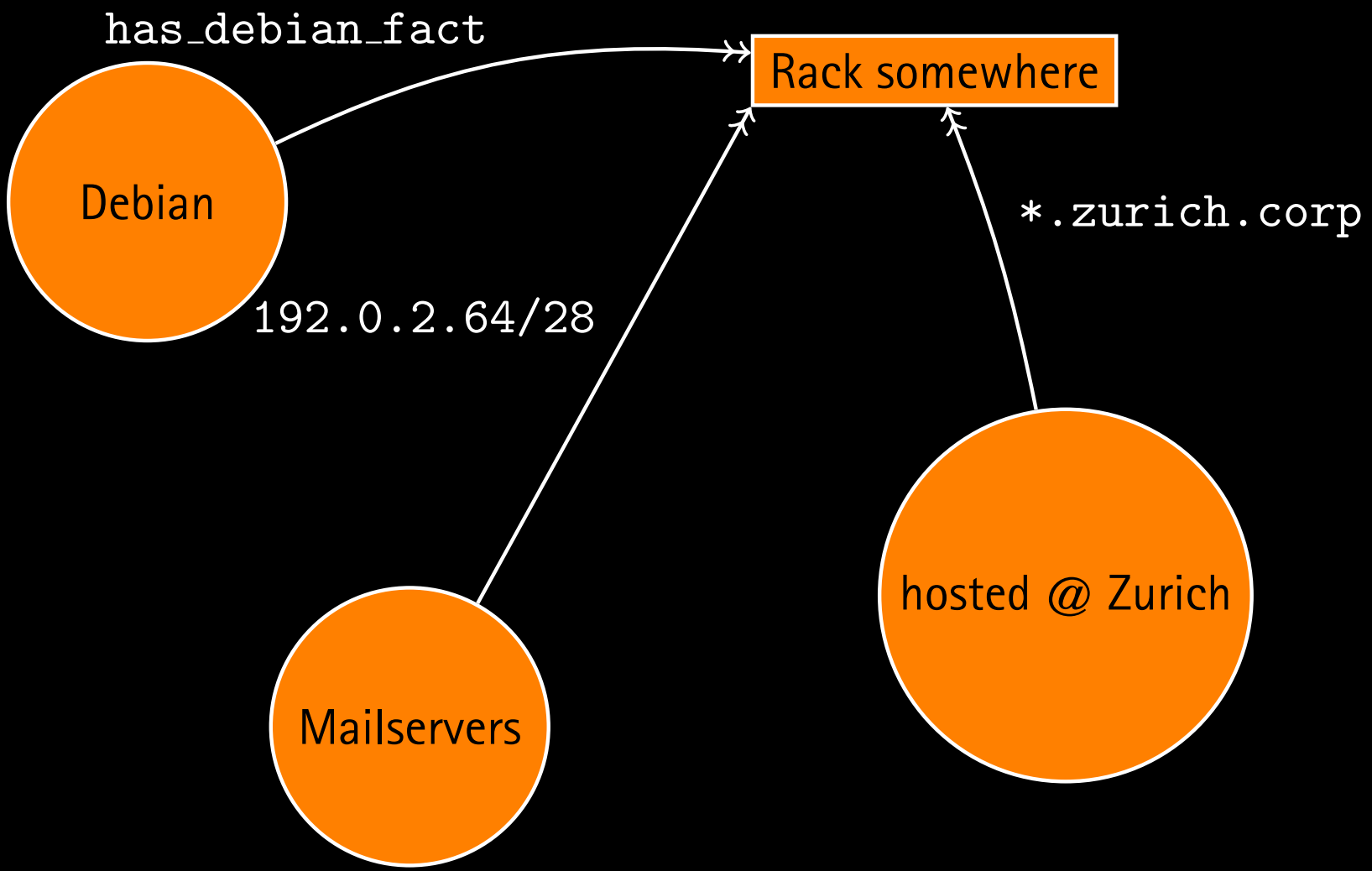
Targeting nodes



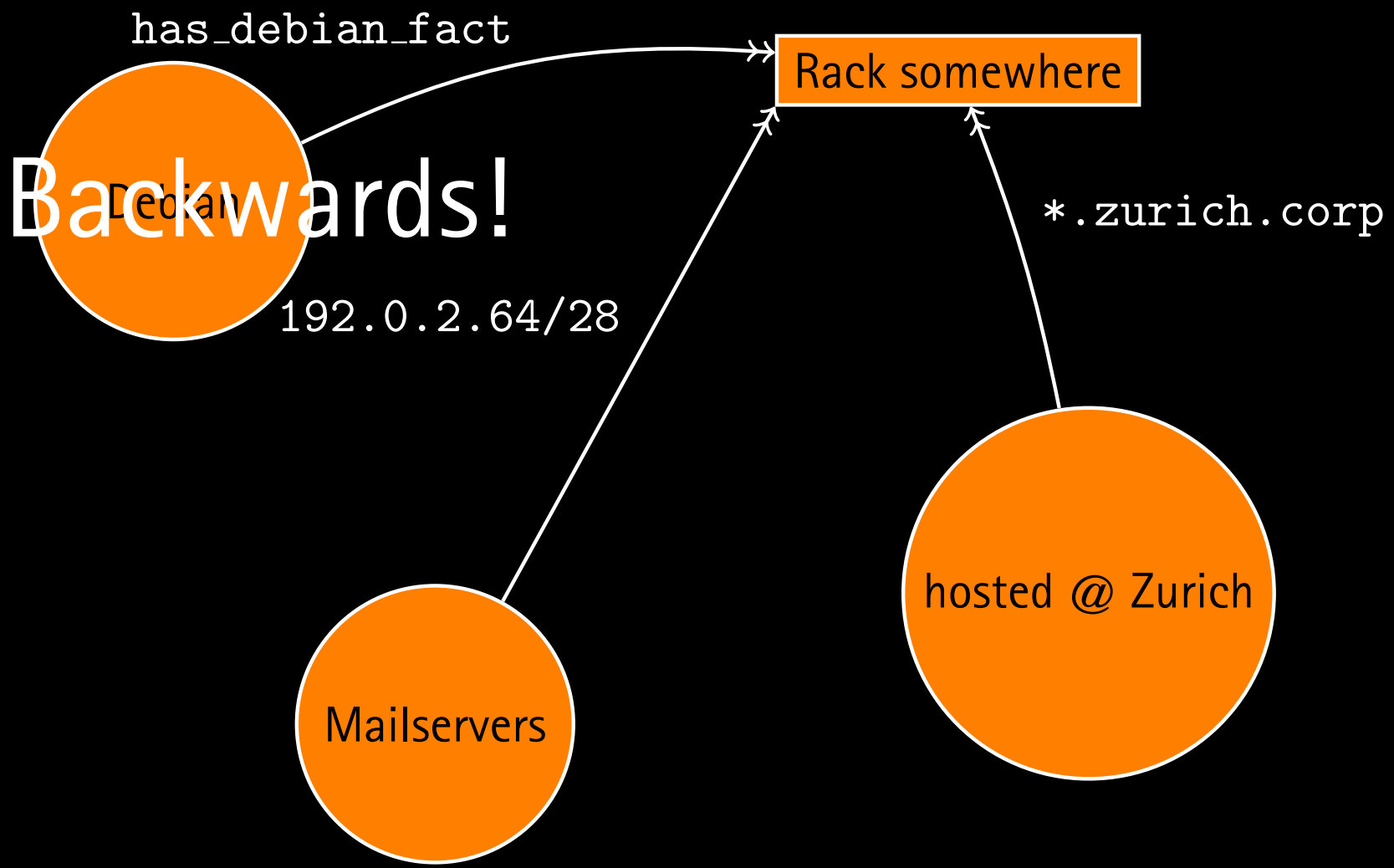
Targeting nodes



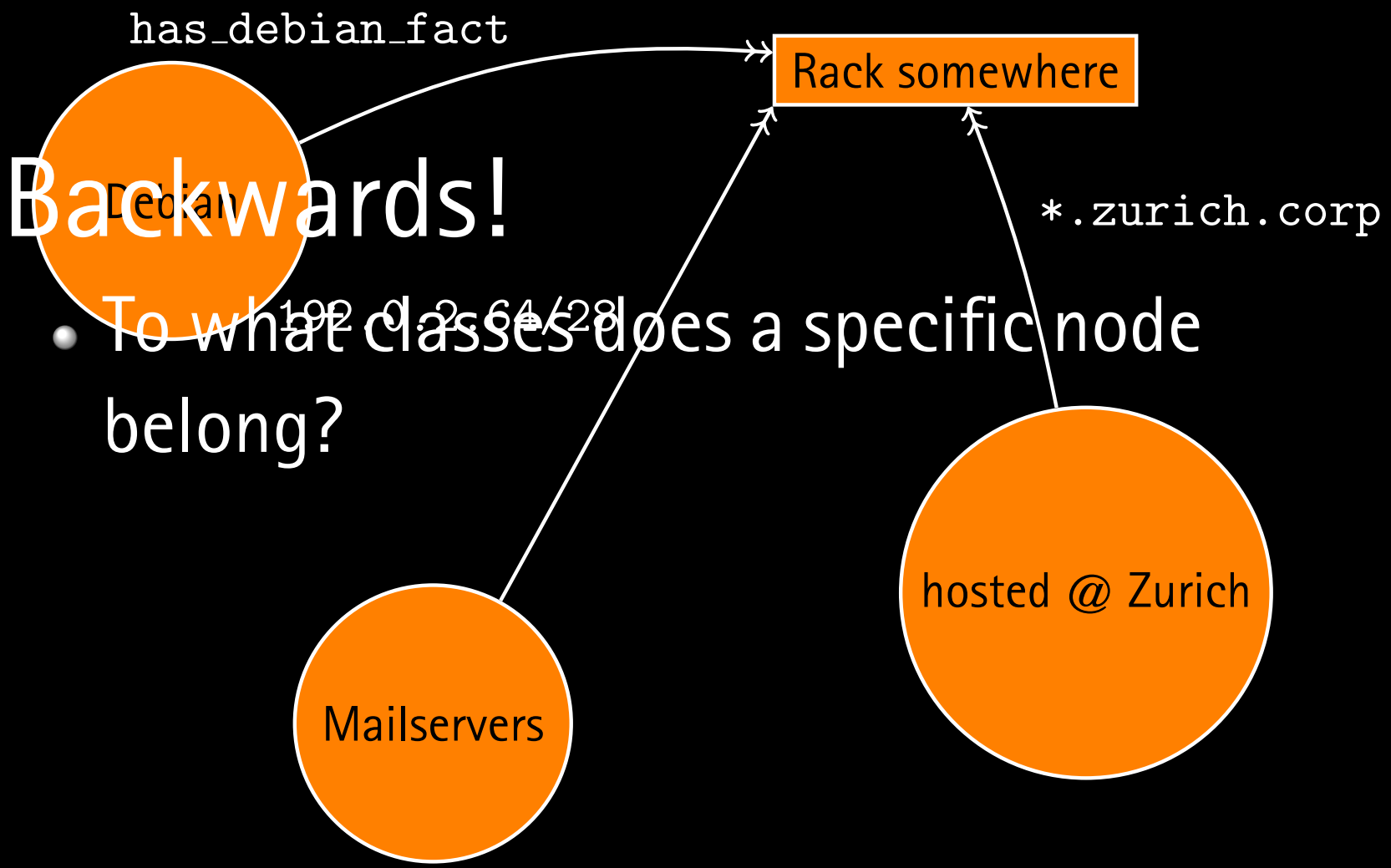
Targeting nodes



Targeting nodes



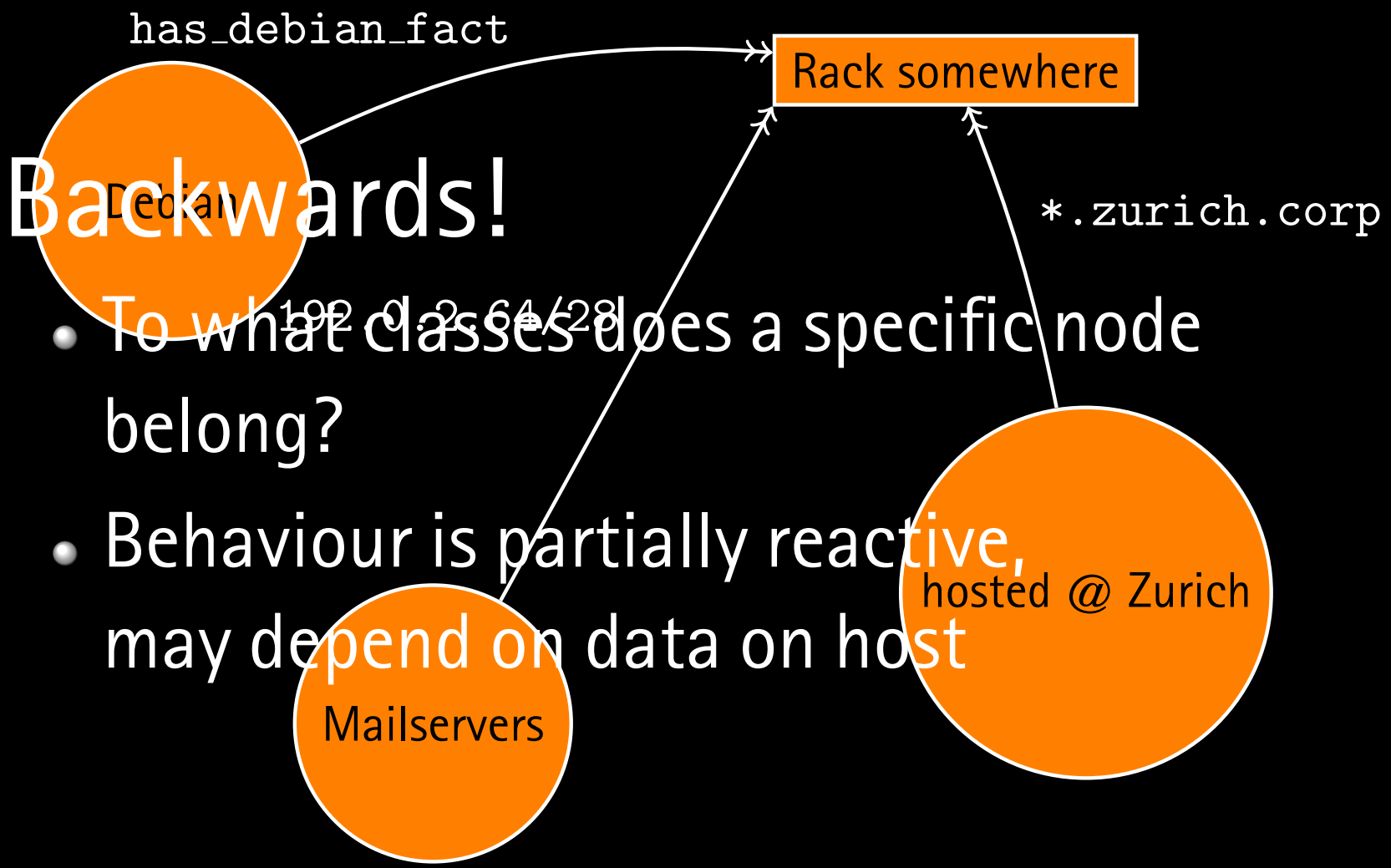
Targeting nodes



Backwards!

- To what classes does a specific node belong?

Targeting nodes



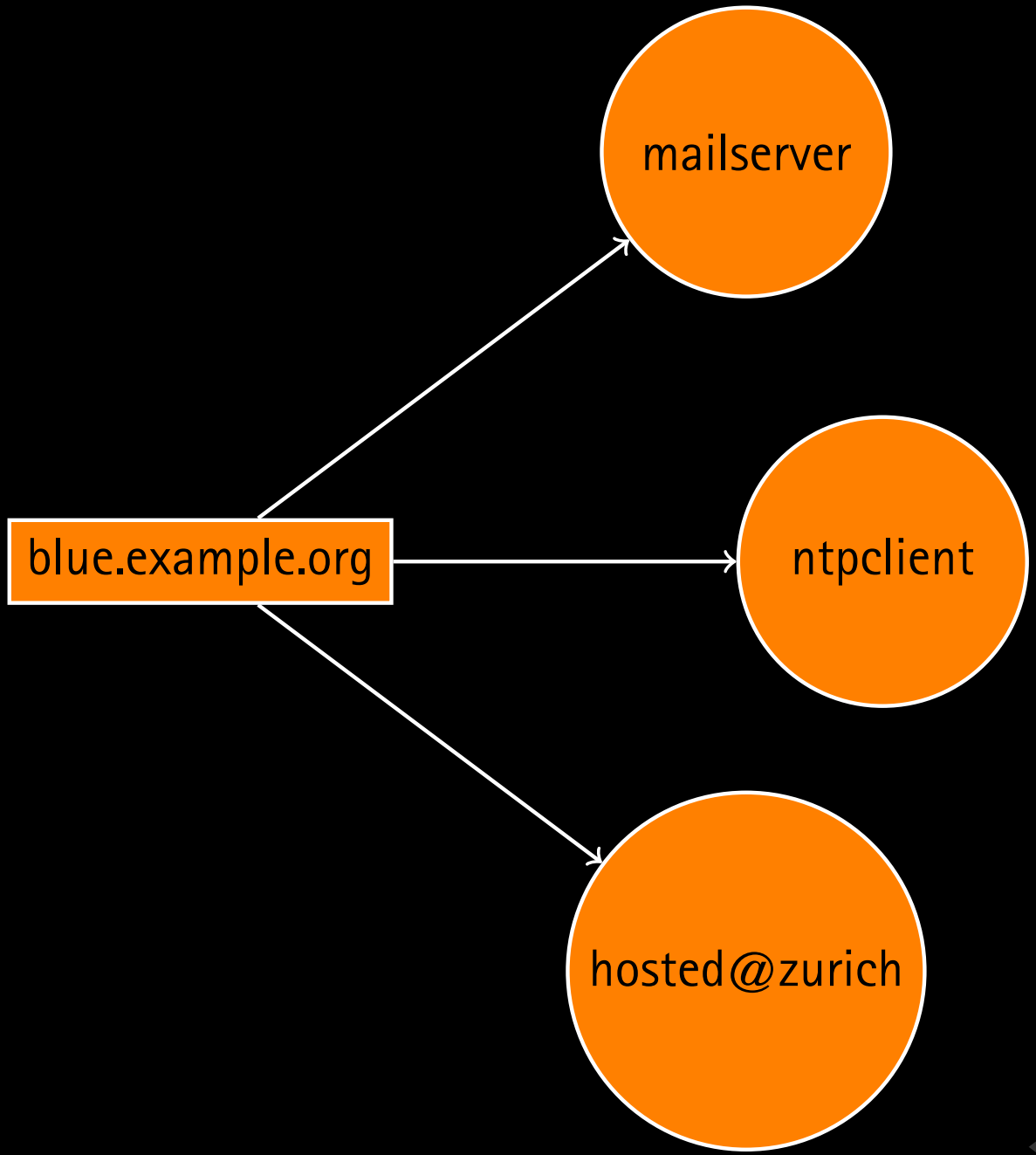
Backwards!

- To what classes does a specific node belong?
- Behaviour is partially reactive, may depend on data on host

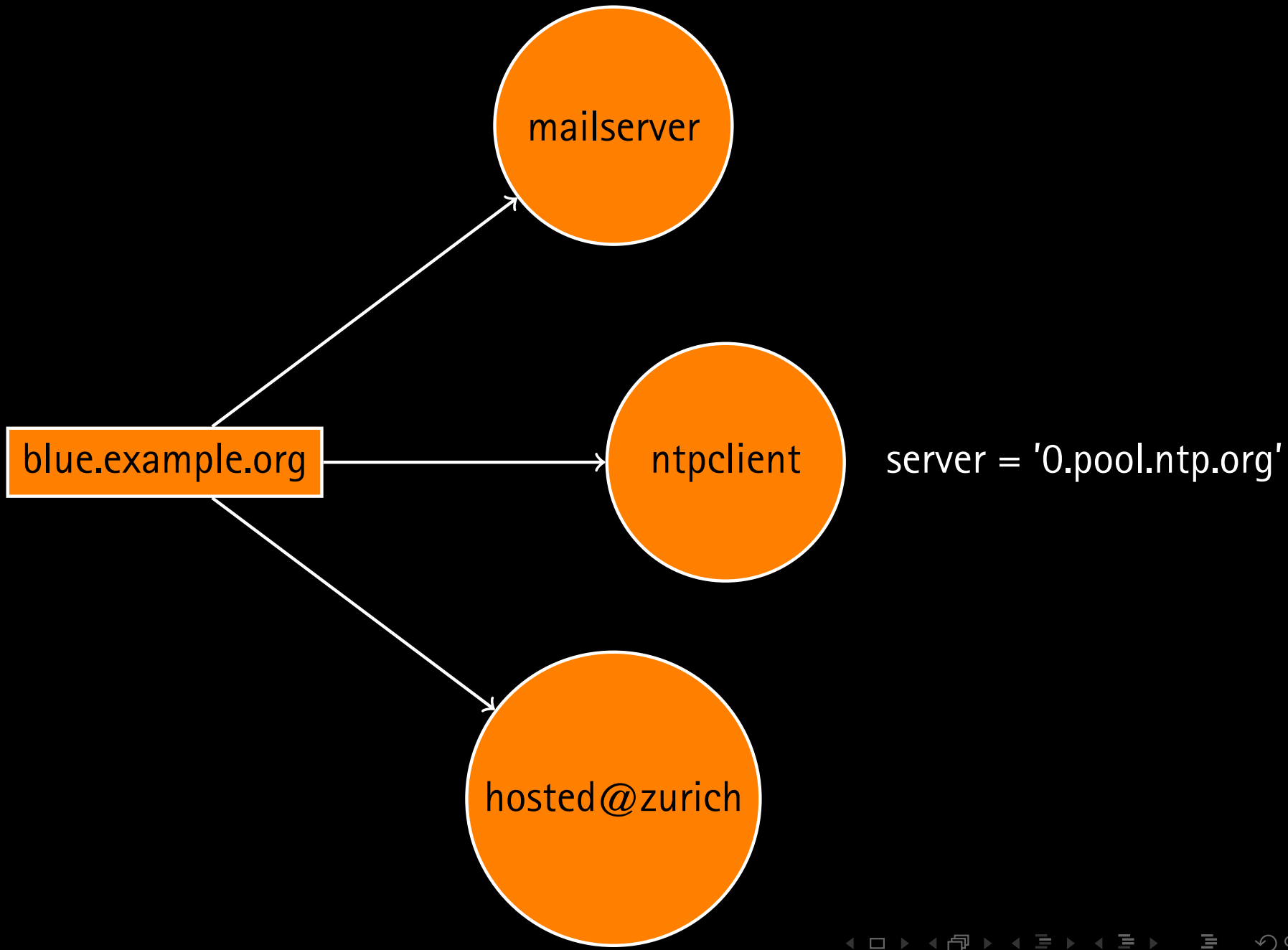
Classifying hosts

blue.example.org

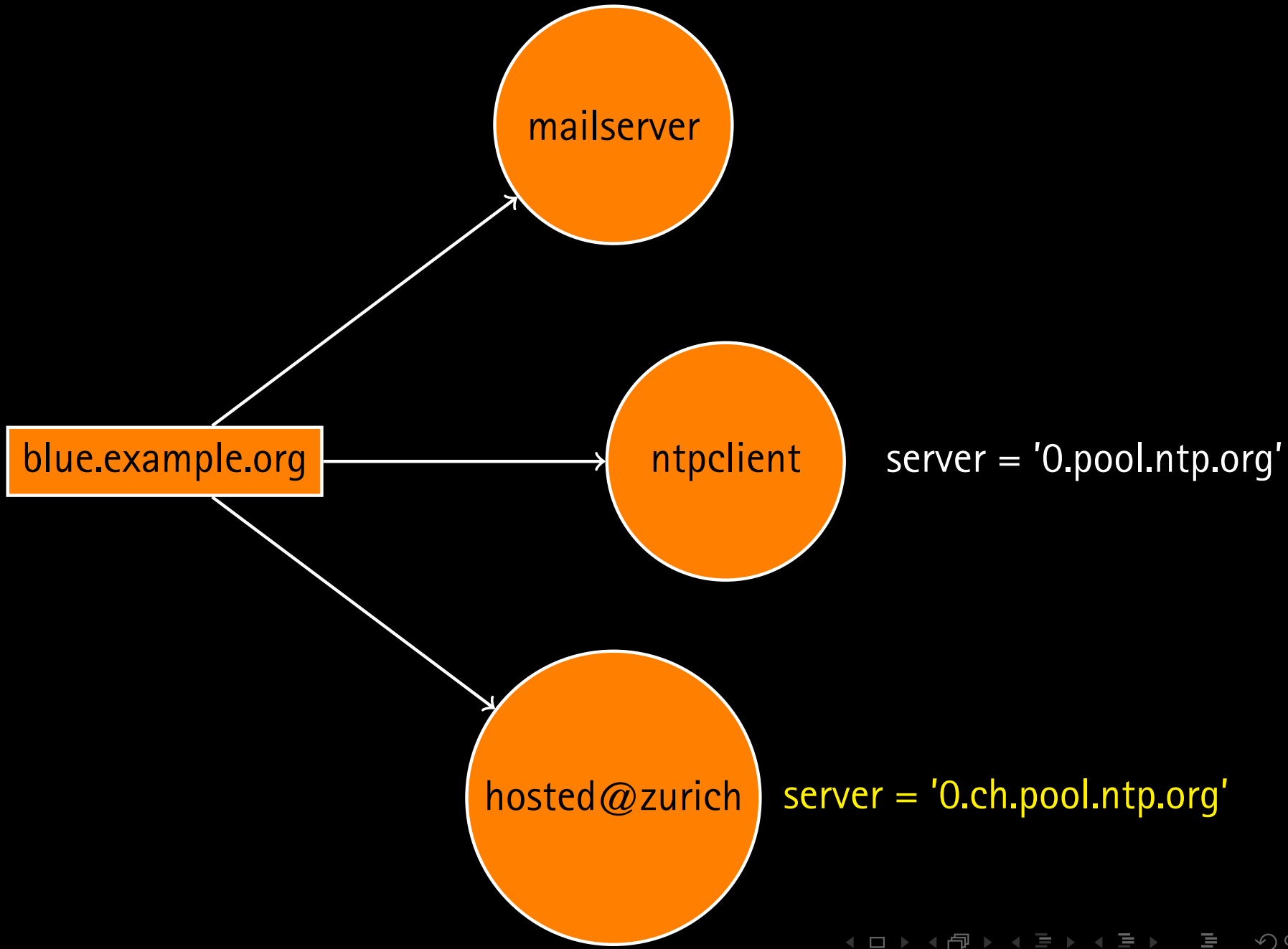
Classifying hosts



Classifying hosts



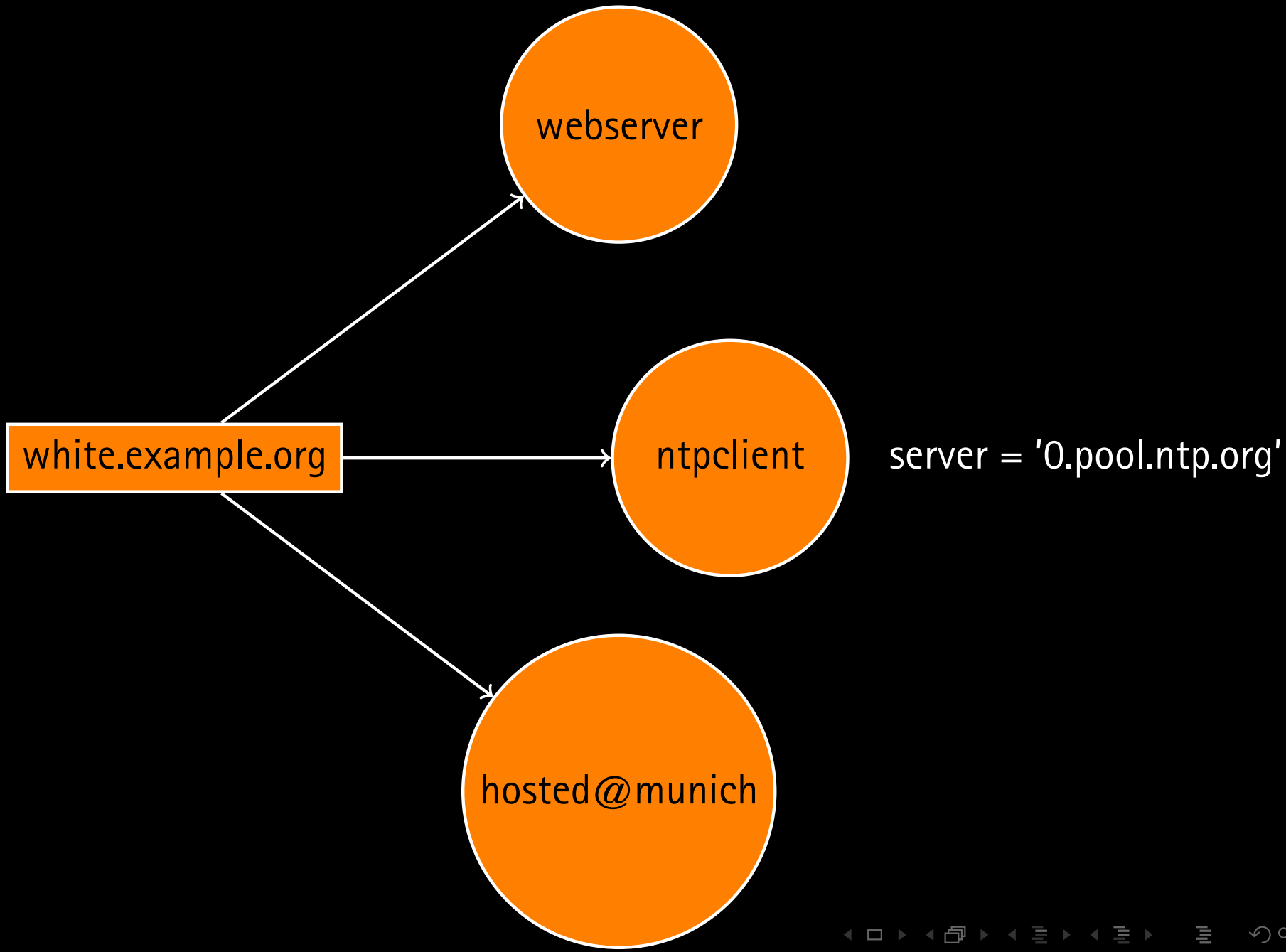
Classifying hosts



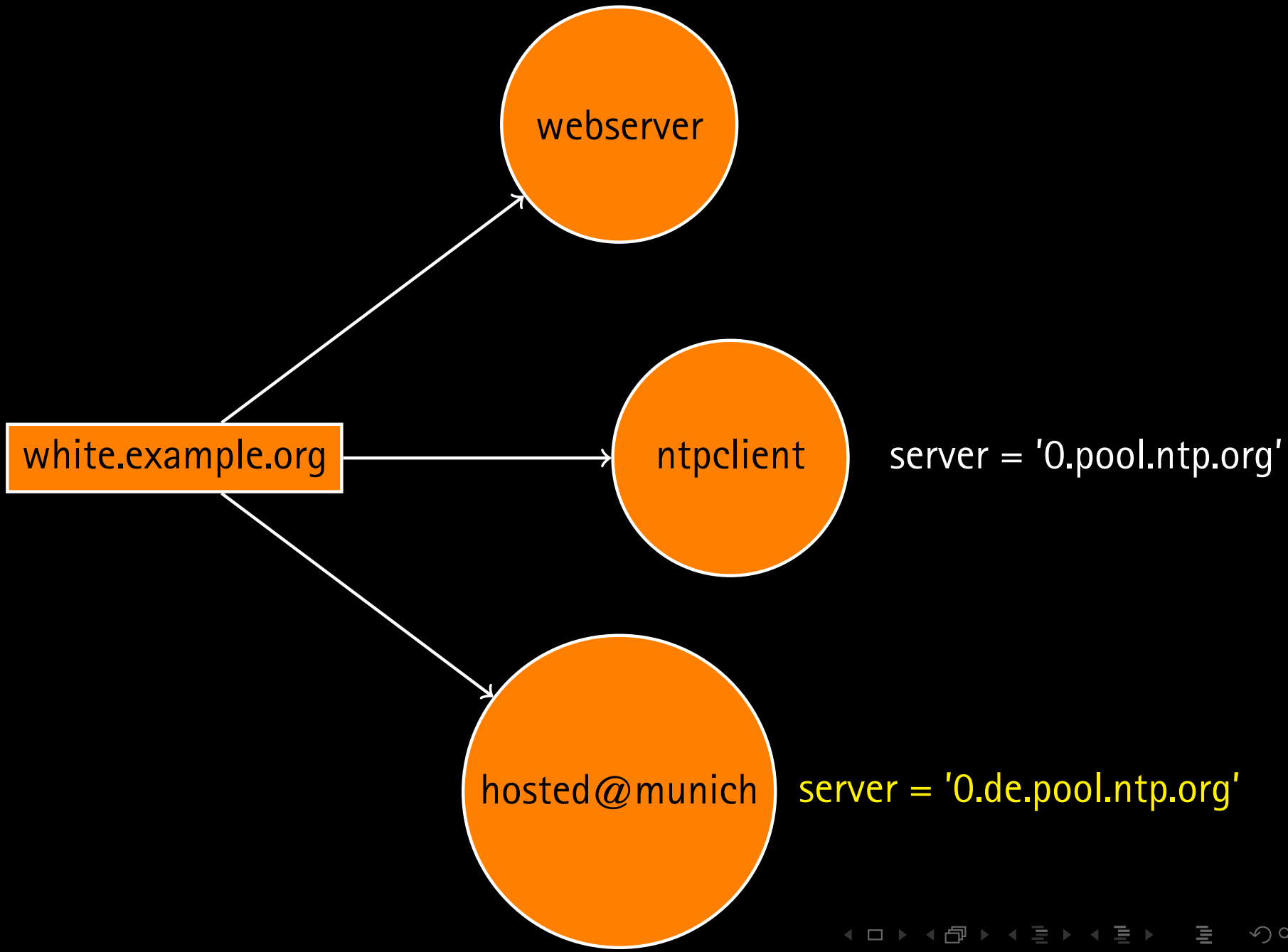
Classifying hosts

white.example.org

Classifying hosts



Classifying hosts



Puppet

```
node 'blue.example.org' {
  $ntpserver = 'red.example.org'
  include common
  include ntp
}
```

Puppet

```
node 'common' {
  $ntpserver = '0.pool.ntp.org'
  include common
  include ntp
}

node 'blue.example.org' inherits 'common' {

}
```

Puppet

```
node 'common' {
  $ntpserver = '0.pool.ntp.org'
  include common
  include ntp
}

node 'blue.example.org' inherits 'common' {
  $ntpserver = 'red.example.org'
}
```

Puppet

Anger!

```
node 'common' {
  $ntpserver = '0.pool.ntp.org'
  include common
  include ntp
}

node 'blue.example.org' inherits 'common' {
  $ntpserver = 'red.example.org'
}
```

Puppet

Anger!

- No multiple inheritance

```
node ’common’ {
  $ntpserver = ’0.pool.ntp.org’
  include common
  include ntp
}

node ’blue.example.org’ inherits ’common’ {
  $ntpserver = ’red.example.org’
}
```

Puppet

Anger!

```
node 'common' {
  $ntpserver = '0.pool.ntp.org'
  include common
  include ntp
}

node 'blue.example.org' inherits 'common' {
  $ntpserver = 'red.example.org'
}
```

- No multiple inheritance
- Inheritance generally discouraged (cf. docs)

Puppet

```
node 'common' {
  $ntpserver = '0.pool.ntp.org'
  include common
  include ntp
}

node 'blue.example.org' inherits 'common' {
  $ntpserver = 'red.example.org'
}
```

Anger!

- No multiple inheritance
- Inheritance generally discouraged (cf. docs)
- Parametrisation unnecessarily difficult

System administration automation principles (abridged)

System administration automation principles (abridged)

- Central control, versioned

System administration automation principles (abridged)

- Central control, versioned
- Parametrisation
  (no special casing in code)

System administration automation principles (abridged)

- Central control, versioned
- Parametrisation
  (no special casing in code)
- No redundancy

System administration automation principles (abridged)

- Central control, versioned
- Parametrisation
  (no special casing in code)
- No redundancy
- Use node information as parameters,
  don't rely on them for behaviour selection

recursive
external
(node)
classifier

Data use by the CMS

- What applications should a role have?
  (aka. modules, states, playbooks)

Data use by the CMS

- What applications should a role have?
  (aka. modules, states, playbooks)
- How does this node differ from all other nodes that have the same application?
  (aka. parameters, pillar, variables)

Data use by the CMS

- What applications should a role have?
  (aka. modules, states, playbooks)
- How does this node differ from all other nodes that have the same application?
  (aka. parameters, pillar, variables)
- Which nodes belong to a group?
  (aka. nodegroups, clusters)

Usage by the system administrator

Usage by the system administrator

- Deploy and manage site-wide configuration changes

Usage by the system administrator

- Deploy and manage site-wide configuration changes
- Upgrade all nodes tagged debian@stable

Usage by the system administrator

- Deploy and manage site-wide configuration changes
- Upgrade all nodes tagged debian@stable
- Update `/etc/motd` on all hosts hosted@zurich

Usage by the system administrator

- Deploy and manage site-wide configuration changes
- Upgrade all nodes tagged debian@stable
- Update `/etc/motd` on all hosts hosted@zurich
- Fetch logs of all hosts tagged mailserver

Usage by the system administrator

- Deploy and manage site-wide configuration changes
- Upgrade all nodes tagged debian@stable
- Update `/etc/motd` on all hosts hosted@zurich
- Fetch logs of all hosts tagged mailserver

reclass just assembles/provides the data

Configuration management

vs.

remote execution

?

Configuration management & remote execution !

Single data source

reclass adapters

Adapters interface between configuration management system and reclass

reclass adapters

Adapters interface between configuration management system and reclass:

- Mode of invocation (module, command-line switches, etc.)

reclass adapters

Adapters interface between configuration management system and reclass:

- Mode of invocation (module, command-line switches, etc.)
- Output (YAML, JSON, ...)

reclass adapters

Adapters interface between configuration management system and reclass:

- Mode of invocation (module, command-line switches, etc.)
- Output (YAML, JSON, ...)

Adapters provided:

- Puppet (*not yet re-implemented*)

reclass adapters

Adapters interface between configuration management system and reclass:

- Mode of invocation (module, command-line switches, etc.)
- Output (YAML, JSON, ...)

Adapters provided:

- Puppet (*not yet re-implemented*)
- Salt (module)

reclass adapters

Adapters interface between configuration management system and reclass:

- Mode of invocation (module, command-line switches, etc.)
- Output (YAML, JSON, ...)

Adapters provided:

- Puppet (*not yet re-implemented*)
- Salt (module)
- Ansible (`exec` required)

reclass and Puppet

- Original reclass written for Puppet

reclass and Puppet

- Original reclass written for Puppet
- ... out of frustration

reclass and Puppet

- Original reclass written for Puppet
- ... out of frustration
- Rage-quit Puppet two years ago

reclass and Puppet

- Original reclass written for Puppet
- ... out of frustration
- Rage-quit Puppet two years ago
- Rewritten reclass from scratch since then

reclass and Puppet

- Original reclass written for Puppet
- ... out of frustration
- Rage-quit Puppet two years ago
- Rewritten reclass from scratch since then
- Could not be bothered to reimplement

reclass and Puppet

- Original reclass written for Puppet
- ... out of frustration
- Rage-quit Puppet two years ago
- Rewritten reclass from scratch since then
- Could not be bothered to reimplement
- Trivially done through adapter plugin

reclass and Salt

- Provides `top` and `pillar` data

reclass and Salt

- Provides `top` and `pillar` data
- Adapter is a Salt module, since 0.16.0

reclass and Salt

- Provides `top` and `pillar` data
- Adapter is a Salt module, since 0.16.0
- nodegroups not yet implemented (Salt issue #5787)

reclass and Ansible

- Provides inventory and node information

reclass and Ansible

- Provides inventory and node information
- Implemented as external script

reclass and Ansible

- Provides inventory and node information
- Implemented as external script
- Does not yet support batched calls (recent Ansible feature)

Enough of this boring stuff!

Parametrisation is key

- Parametrise modules as much as possible

Parametrisation is key

- Parametrise modules as much as sensible

Parametrisation is key

- Parametrise modules as much as sensible
- At all cost, avoid special-casing in module code

Parametrisation is key

- Parametrise modules as much as sensible
- At all cost, avoid special-casing in module code
- Reclass allows you to keep your parameters modular

Parametrisation is key

- Parametrise modules as much as sensible
- At all cost, avoid special-casing in module code
- Reclass allows you to keep your parameters modular
- Define your data in one place only (no redundancy)

reclass node definition

```
blue.example.org.yaml:
  ---
  applications:
  - postfix
  - ntp
  parameters:
    ntp:
      server: 0.pool.ntp.org
```

reclass node definition

```
blue.example.org.yaml:
  ---
  applications:
  - postfix
  - ntp
  parameters:
    ntp:
      server: 0.pool.ntp.org
```

But wait!
You promised recursion!

yaml_fs

- YAML files for nodes in `$inventory_base_uri/nodes`

yaml_fs

- YAML files for nodes in `$inventory_base_uri/nodes`
- YAML files defining classes in `$inventory_base_uri/classes`

yaml_fs

- YAML files for nodes in `$inventory_base_uri/nodes`
- YAML files defining classes in `$inventory_base_uri/classes`
- Nodes and classes files may specify classes to inherit

yaml_fs

- YAML files for nodes in `$inventory_base_uri/nodes`
- YAML files defining classes in `$inventory_base_uri/classes`
- Nodes and classes files may specify classes to inherit
- You can think of classes as tags, too!

yaml_fs

- YAML files for nodes in `$inventory_base_uri/nodes`
- YAML files defining classes in `$inventory_base_uri/classes`
- Nodes and classes files may specify classes to inherit
- You can think of classes as tags, too!
- Smart (deep) merging on return from recursive descent walk

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - common
  - mailserver
  parameters:
    ntp:
      server: 0.ch.pool.ntp.org

classes/common.yaml:
  applications:
  - ntp
  parameters:
    ntp:
      server: 0.pool.ntp.org
```

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - common
  - mailserver
  parameters:
    ntp:
      server: 0.ch.pool.ntp.org

classes/common.yaml:
  applications:
  - ntp
  parameters:
    ntp:
      server: 0.pool.ntp.org
```

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - common
  - mailserver
  parameters:
    ntp:
      server: 0.ch.pool.ntp.org

classes/common.yaml:
  applications:
  - ntp
  parameters:
    ntp:
      server: 0.pool.ntp.org
```

More specific classes override data defined in less specific classes

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - common
  - mailserver
  - hosted@zurich

classes/hosted@zurich.yaml:
  parameters:
    ntp:
      server: 0.ch.pool.ntp.org
```

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - common
  - mailserver
  - hosted@zurich

classes/hosted@zurich.yaml:
  parameters:
    ntp:
      server: 0.ch.pool.ntp.org
```

Multiple inheritance well-defined order

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - ssh-server


classes/ssh-server.yaml
  parameters:
    permit_root_login: no
```

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - ssh-server
  - backup-client

classes/ssh-server.yaml
  parameters:
    permit_root_login: no

classes/backup-client.yaml
  parameters:
    permit_root_login: without-password
```

reclass node definition

```
nodes/blue.example.org.yaml:
  classes:
  - ssh-server
  - backup-client

classes/ssh-server.yaml
  parameters:
    permit_root_login: no

classes/backup-client.yaml
  parameters:
    permit_root_login: without-password
  classes:
  - ssh-server
```

Parameter interpolation

```
nodes/diamond.example.org.yaml:
  classes:
  - motd
```

```
classes/motd.yaml
  parameters:
    motd:
      message: ${floyd_reference}
```

Parameter interpolation

```
nodes/diamond.example.org.yaml:
  classes:
  - motd
  parameters:
  - floyd_reference: Shine on, you crazy diamond

classes/motd.yaml
  parameters:
    motd:
      message: ${floyd_reference}
```

reclass future work

- Package it.

reclass future work

- Package it. Doh!

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?
- Policy classification (regexp $\rightarrow$ class mappings)

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?
- Policy classification (regexp $\rightarrow$ class mappings)
- Membership lists

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?
- Policy classification (regexp $\rightarrow$ class mappings)
- Membership lists
- Other data sources?

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?
- Policy classification (regexp $\rightarrow$ class mappings)
- Membership lists
- Other data sources?
- Better unit testing (without philosophical debates)

reclass future work

- Package it. Doh!
- `preseed.cfg`/`d-i` adapter?
- Policy classification (regexp $\rightarrow$ class mappings)
- Membership lists
- Other data sources?
- Better unit testing (without philosophical debates)
- Your idea here!