Languages and Models for Automatic Deployment of Cloud Applications

Gianluigi Zavattaro

University of Bologna - Italy
FoCUS research team INRIA - France

Based on joint work with:
Roberto Di Cosmo and **Stefano Zacchiroli** PPS/Paris Diderot
Tudor A. Lascu and **Jacopo Mauro** Univ. of Bologna

NOVEL OPPORTUNITIES ~~OPEN PROBLEMS~~ IN FOR CONCURRENCY THEORY

Languages and Models for Automatic Deployment of Cloud Applications

Gianluigi Zavattaro

University of Bologna - Italy
FoCUS research team INRIA - France

Based on joint work with:
Roberto Di Cosmo and **Stefano Zacchiroli** PPS/Paris Diderot
Tudor A. Lascu and **Jacopo Mauro** Univ. of Bologna

Aeolus: Mastering the cloud complexity

- Models, languages and tools for the administration of cloud applications
  - Cloud computing offers the possibility to build **sophisticated** software systems on virtualized infrastructures at a **fraction** of the cost necessary just few years ago…
  - …but the administration of such software systems is a serious **challenge**, especially if one wants to take advantage of all the cloud potentialities

New models and languages: an industrial need

- Several industrial initiatives pursue the definition of **high-level** languages for the management of applications deployed on virtualized infrastructures

New models and languages: an industrial need

CLOUD FOUNDRY™ BETA
DEPLOY & SCALE YOUR APPLICATIONS IN SECONDS

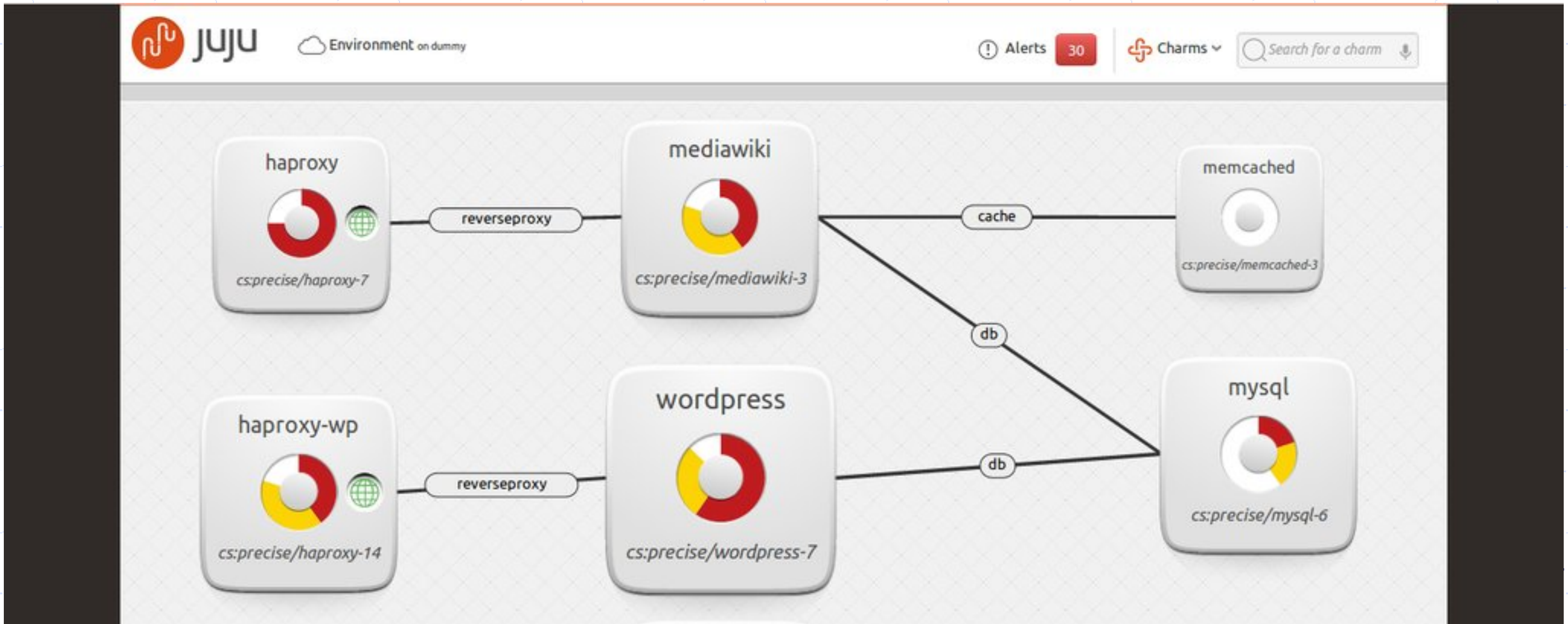
- Cloud Foundry (launched by VMware) provides a PaaS with high-level primitives for **service creation** and **binding**

```
$ cf create-service
What kind?> 1
Name?> cleardb-e2006
Creating service cleardb-e2006... OK
```

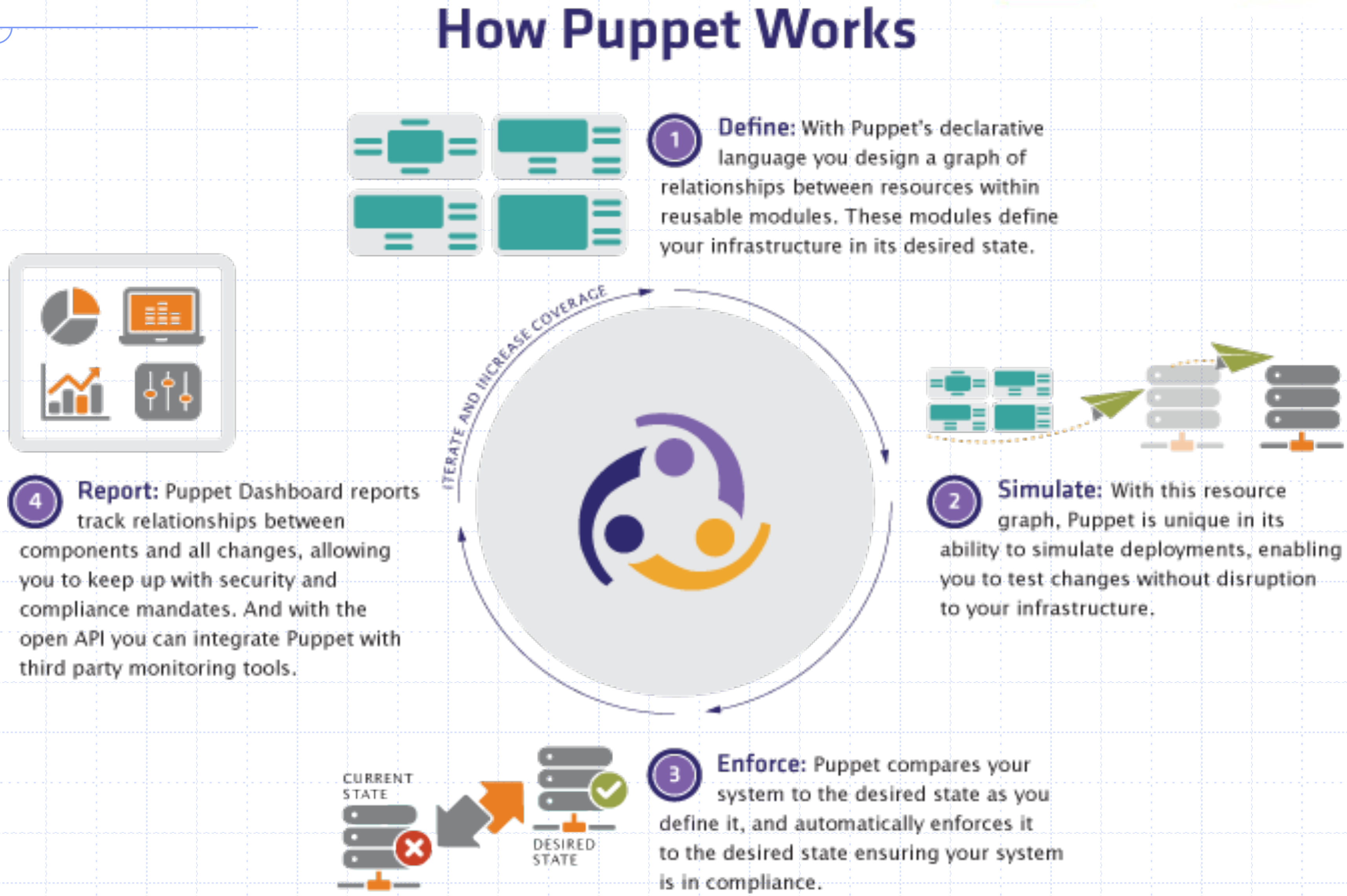
```
$ cf bind-service
1: myapp
Which application?> 1
1: cleardb-e2006
Which service?> 1
Binding cleardb-e2006 to myapp... OK
```

New models and languages: an industrial need

- Juju (an Ubuntu initiative) provides similar primitives
  - service **replication** and scaling supported
  - includes **GUI** for application management



New models and languages: an industrial need



New models and languages: an industrial need

- Declarative language: three kinds of **resources**

```
package
{ 'openssh-server':
   ensure => installed,}
```

```
service
{ 'sshd':
   ensure => running,
   enable => true,
   hasstatus => true,
   hasrestart => true,}
```

```
file
{ '/etc/ssh/sshd_config':
  source  => 'puppet:///modules/sshd/sshd_config',
  owner   => 'root',
  group   => 'root',
  mode    => '640',
  notify  => Service['sshd'],
  require => Package['openssh-server'],}
```

New models and languages: an industrial need

puppet labs

- Declarative language: three kinds of **resources**

```
service
{ 'sshd':
    ensure => running,
    enable => true,
    hasstatus => true,
    hasrestart => true,}
```

```
package
{ 'openssh-server':
   ensure => installed,}

file
{ '/etc/ssh/sshd_config':
  source  => 'puppet:///modules/sshd/sshd_config',
  owner   => 'root',
  group   => 'root',
  mode    => '640',
  notify  => Service['sshd'],
  require => Package['openssh-server'],}
```

New models and languages: an industrial need

- Declarative language: three kinds of **resources**

```
package
{ 'openssh-server':
   ensure => installed,}
```

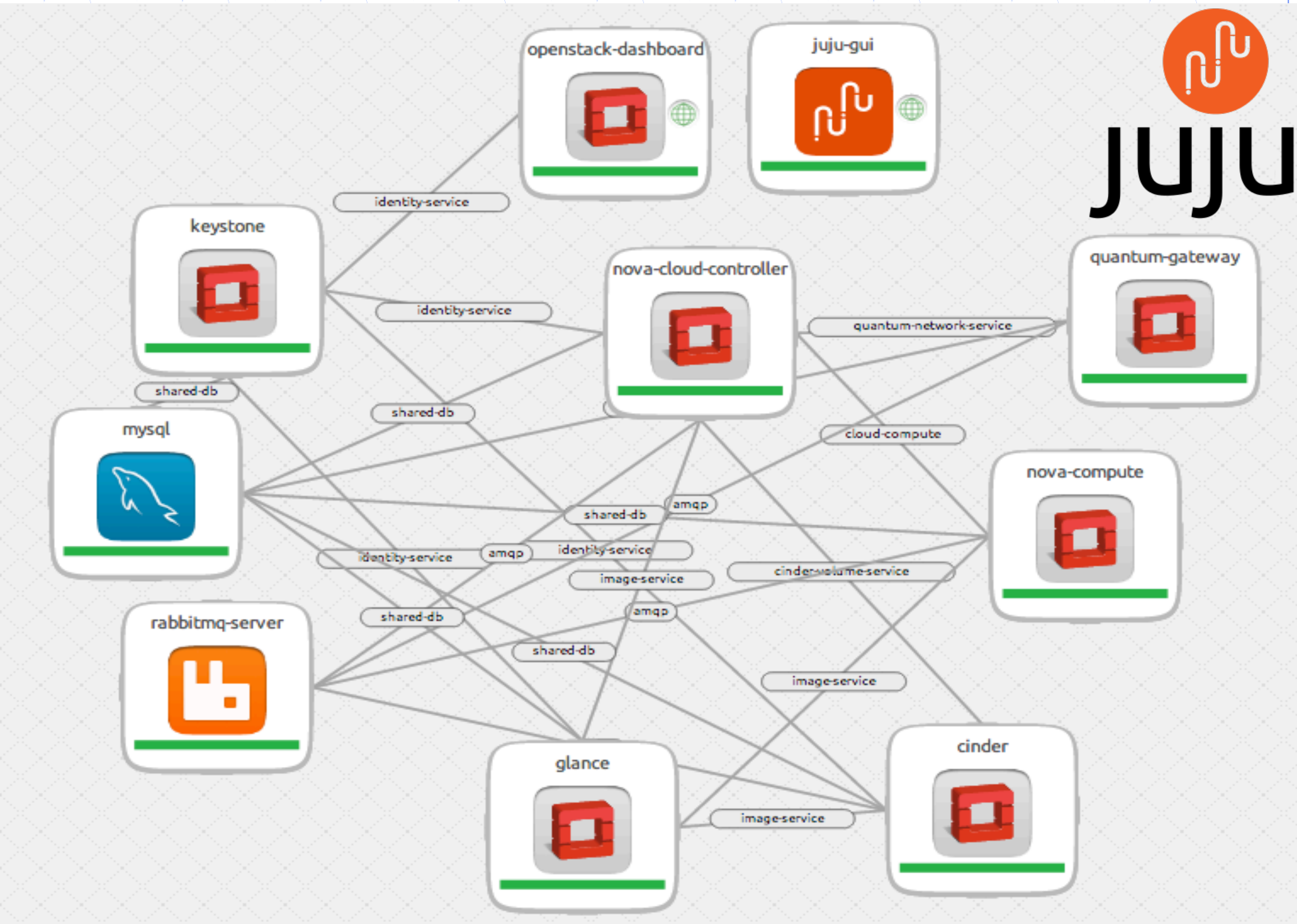
```
service
{ 'sshd':
   ensure => running,
   enable => true,
```

```
file
{ '/etc/ssh/sshd_config':
  source  => 'puppet:///modules/sshd/sshd_config',
  owner   => 'root',
  group   => 'root',
  mode    => '640',
  notify  => Service['sshd'],
  require => Package['openssh-server'],}
```

New models and languages: an industrial need

- In all these approaches a lot of **human** intervention is needed for
  - **Service** selection
  - Deciding the service **bindings**

    (see next slide)



openstack-dashboard
juju-gui
juju
keystone
identity-service
nova-cloud-controller
quantum-gateway
quantum-network-service
shared-db
mysql
cloud-compute
nova-compute
amqp
identity-service
image-service
cinder-volume-service
rabbitmq-server
shared-db
image-service
glance
cinder

New models and languages: an industrial need

- In all these approaches a lot of **human** intervention is needed for
  - **Service** selection
  - Deciding the service **bindings**
    (see next slide)
- The challenge:
  - **automatize** as much as possible the management of such applications

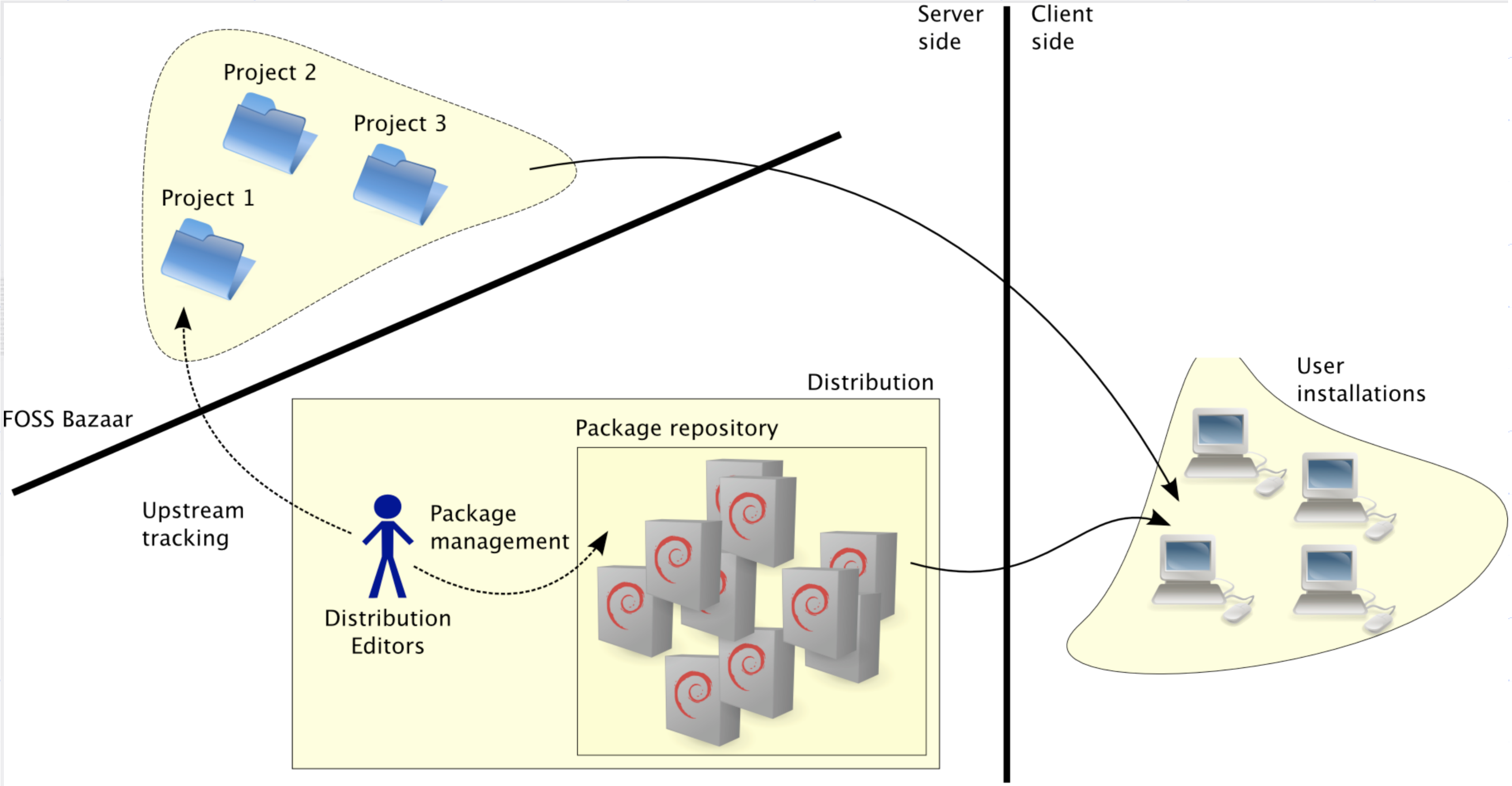
Structure of the talk

- The Aeolus starting point
- Formalizing the “deployment” problem
- Solving the “deployment” problem
  - Ackermann-hard in the general case
  - PolyTime without conflicts
- Open issues and related work

Structure of the talk

- **The Aeolus starting point**
- Formalizing the “deployment” problem
- Solving the “deployment” problem
  - Ackermann-hard in the general case
  - PolyTime without conflicts
- Open issues and related work

Automatic management of package-based software systems



- Developed rather sophisticated **tools** for FOSS (free and open-source software)

The dependency/conflict model

- Tools are based on the **dependency/conflict** model

```
Package: apache2
Version: 2.4.1-2
Maintainer: Debian Apache Maintainers <debian-apache@...>
Depends: lsb-base, procps, perl, mime-support, apache2-bin (= 2.4.1-2)
 apache2-data (= 2.4.1-2)
Conflicts: apache2.2-common
Provides: httpd
Description: Apache HTTP Server
```

Package configuration as a SAT problem

- One boolean **variable** for each package
  - TRUE – installed
  - FALSE – not installed
- Conflicts/dependencies can be formalized as boolean **formulae**
- Finding a correct configuration is mapped to a **satisfaction** problem

Package configuration as a SAT problem

- One boolean **variable** for each package
  - TRUE – installed
- C
  fo

Advanced configuration tools exploit state-of-the-art SAT solvers

- Finding a correct configuration is mapped to a **satisfaction** problem

Structure of the talk

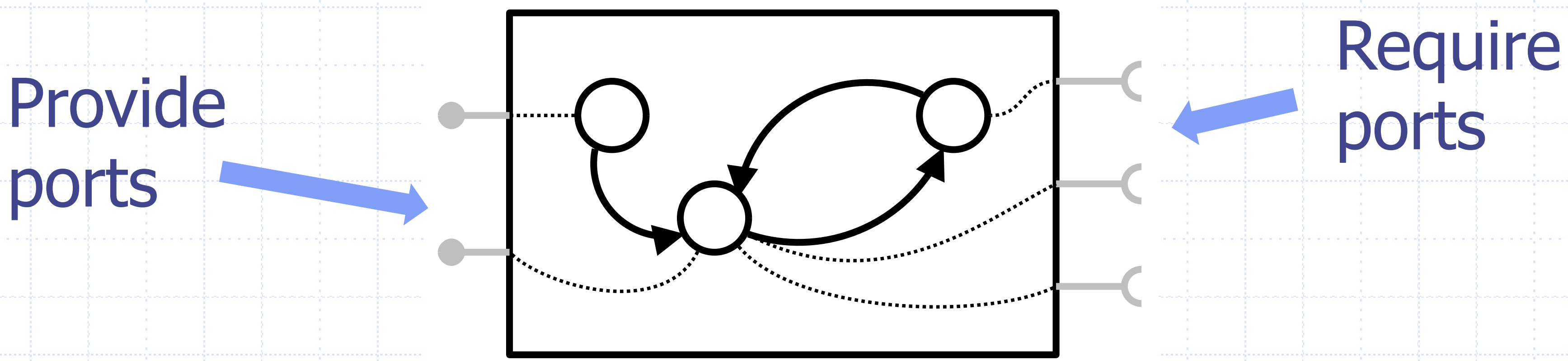
- The Aeolus starting point
- **Formalizing the “deployment” problem**
- Solving the “deployment” problem
  - Ackermann-hard in the general case
  - PolyTime without conflicts
- Open issues and related work

The Aeolus component model

- A component has **provide** and **require** ports

The Aeolus component model

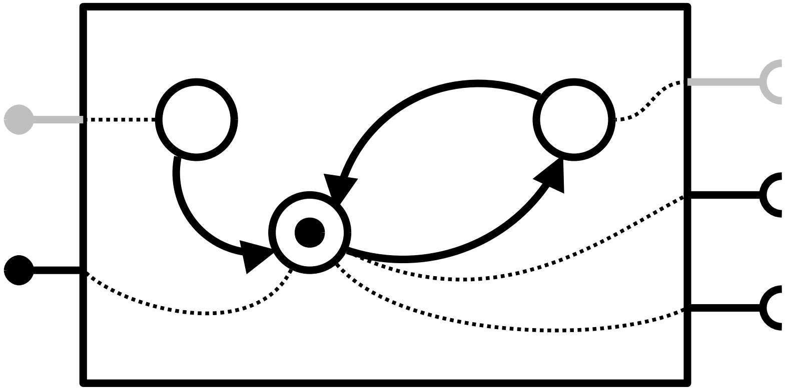
- A component has **provide** and **require** ports
- A component has an internal **state machine**



The Aeolus component model

- A component has **provide** and **require** ports
- A component has an internal **state machine**
- Ports are **active** or **inactive** according to the current internal state

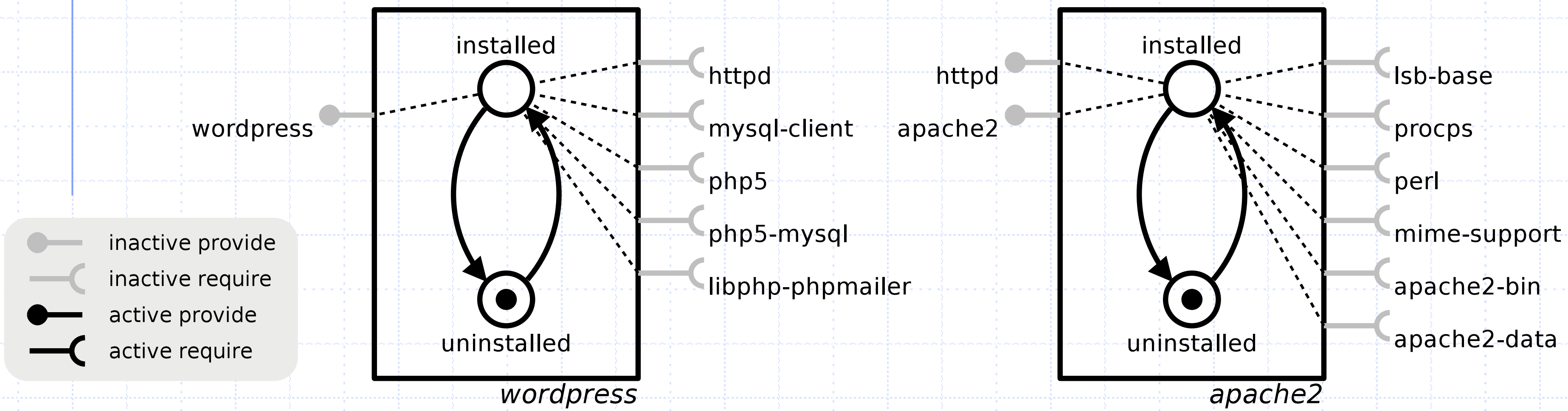
Provide ports



Require ports

Packages in the Aeolus model

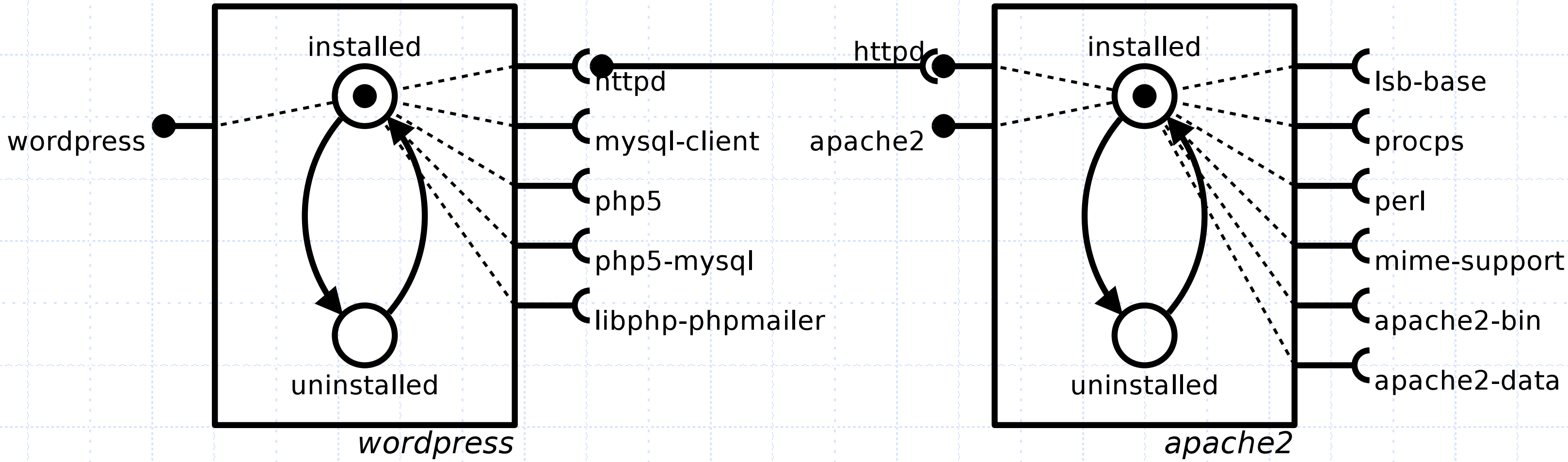
- The **packages** example



Package: wordpress
Version: 3.0.5+dfsg-0+squeeze1
Depends: httpd, mysql-client, php5, php5-mysql, libphp-phpmailer (>= 1.73-4),

Packages in the Aeolus model

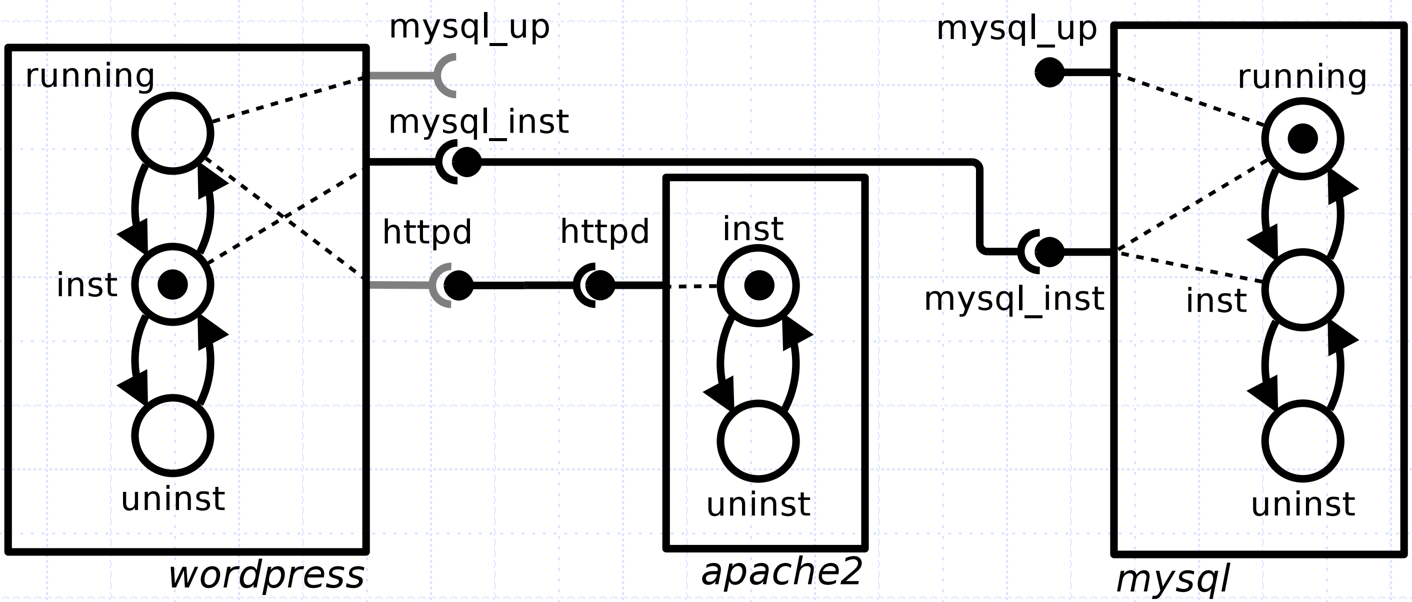
- **Binding** between two components



Package: wordpress
Version: 3.0.5+dfsg-0+squeeze1
Depends: httpd, mysql-client, php5, php5-mysql, libphp-phpmailer (>= 1.73-4),

Services in the Aeolus model

- At the service level, also a running state becomes relevant:
  - wordpress need to know the network address of a **running** MySQL instance



Conflicts in the Aeolus model

- Conflicts are expressed as **special** ports
  - The apache web server is in **conflict** with the lighttpd web server

Formalizing the “deployment” problem

Definition 1 (Component type). *The set Γ of* component types *of the* Aeolus core *model, ranged over by $\mathscr{T}, \mathscr{T}_1, \mathscr{T}_2, \ldots$ contains 4-ples $\langle Q, q_0, T, D \rangle$ where:*

- *Q is a finite set of states containing the initial state q_0;*
- *$T \subseteq Q \times Q$ is the set of* transitions*;*
- *D is a function from Q to a 3-ple $\langle \mathbf{P}, \mathbf{R}, \mathbf{C} \rangle$ of interface names (i.e. $\mathbf{P}, \mathbf{R}, \mathbf{C} \subseteq \mathscr{I}$) indicating the provide, require, and conflict ports that each state activates. We assume that the initial state q_0 has no requirements and conflicts (i.e. $D(q_0) = \langle \mathbf{P}, \emptyset, \emptyset \rangle$).*

Definition 2 (Configuration). *A* configuration *$\mathscr{C}$ is a 4-ple $\langle U, Z, S, B \rangle$ where:*

- *$U \subseteq \Gamma$ is the finite* universe *of the available component types;*
- *$Z \subseteq \mathscr{Z}$ is the set of the currently deployed* components*;*
- *S is the component* state description, *i.e. a function that associates to components in Z a pair $\langle \mathscr{T}, q \rangle$ where $\mathscr{T} \in U$ is a component type $\langle Q, q_0, T, D \rangle$, and $q \in Q$ is the current component state;*
- *$B \subseteq \mathscr{I} \times Z \times Z$ is the set of* bindings, *namely 3-ple composed by an interface, the component that requires that interface, and the component that provides it; we assume that the two components are different.*

Formalizing the “deployment” problem

Definition 5 (Actions). *The set* $\mathscr{A}$ *contains the following actions:*

- $stateChange(\langle z_1, q_1, q'_1 \rangle, \ldots, \langle z_n, q_n, q'_n \rangle)$ *where* $z_i \in \mathscr{Z}$ *and* $\forall i \neq j \,.\, z_i \neq z_j$;
- $bind(r, z_1, z_2)$ *where* $z_1, z_2 \in \mathscr{Z}$ *and* $r \in \mathscr{I}$;
- $unbind(r, z_1, z_2)$ *where* $z_1, z_2 \in \mathscr{Z}$ *and* $r \in \mathscr{I}$;
- $newRsrc(z : \mathscr{T})$ *where* $z \in \mathscr{Z}$ *and* $\mathscr{T} \in U$ *is the component type of* z;
- $delRsrc(z)$ *where* $z \in \mathscr{Z}$.

Formalizing the "deployment" problem

Definition 6 (Reconfigurations). *Reconfigurations are denoted by transitions* $\mathscr{C} \xrightarrow{\alpha} \mathscr{C}'$ *meaning that the execution of* $\alpha \in \mathscr{A}$ *on the configuration* $\mathscr{C}$ *produces a new configuration* $\mathscr{C}'$*. The transitions from a configuration* $\mathscr{C} = \langle U, Z, S, B \rangle$ *are defined as follows:*

$$\mathscr{C} \xrightarrow{stateChange(\langle z_1,q_1,q_1' \rangle,\dots,\langle z_n,q_n,q_n' \rangle)} \langle U, Z, S', B \rangle$$
$$\text{if } \forall i \,.\, \mathscr{C}[z_i].\texttt{state} = q_i$$
$$\text{and } \forall i \,.\, (q_i, q_i') \in \mathscr{C}[z_i].\texttt{trans}$$
$$\text{and } S'(z') = \begin{cases} \langle \mathscr{C}[z_i].\texttt{type}, q_i' \rangle & \text{if } \exists i \,.\, z' = z_i \\ \mathscr{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathscr{C} \xrightarrow{bind(r,z_1,z_2)} \langle U, Z, S, B \cup \langle r, z_1, z_2 \rangle \rangle$$
$$\text{if } \langle r, z_1, z_2 \rangle \notin B$$
$$\text{and } r \in \mathscr{C}[z_1].\texttt{req} \cap \mathscr{C}[z_2].\texttt{prov}$$

$$\mathscr{C} \xrightarrow{unbind(r,z_1,z_2)} \langle U, Z, S, B \setminus \langle r, z_1, z_2 \rangle \rangle \quad \text{if } \langle r, z_1, z_2 \rangle \in B$$

$$\mathscr{C} \xrightarrow{newRsrc(z:\mathscr{T})} \langle U, Z \cup \{z\}, S', B \rangle$$
$$\text{if } z \notin Z,\ \mathscr{T} \in U$$
$$\text{and } S'(z') = \begin{cases} \langle \mathscr{T}, \mathscr{T}.\texttt{init} \rangle & \text{if } z' = z \\ \mathscr{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathscr{C} \xrightarrow{delRsrc(z)} \langle U, Z \setminus \{z\}, S', B' \rangle$$
$$\text{if } S'(z') = \begin{cases} \bot & \text{if } z' = z \\ \mathscr{C}[z'] & \text{otherwise} \end{cases}$$
$$\text{and } B' = \{ \langle r, z_1, z_2 \rangle \in B \mid z \notin \{z_1, z_2\} \}$$

“Deployment” problem

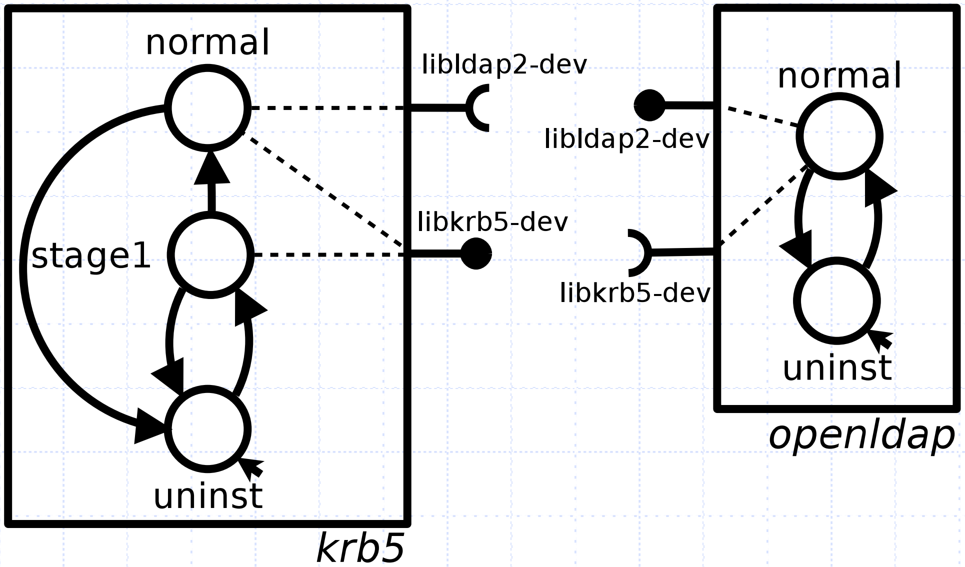
- Input:
  - A set of component types (called **Universe**)
  - One **target** component type-state pair
- Output:
  - **Yes**, if there exists a **deployment plan**
  - **No**, otherwise

Deployment plan:

a sequence of actions leading to a final configuration containing at least one component of the given target type, in the given target state

Deployment problem: example

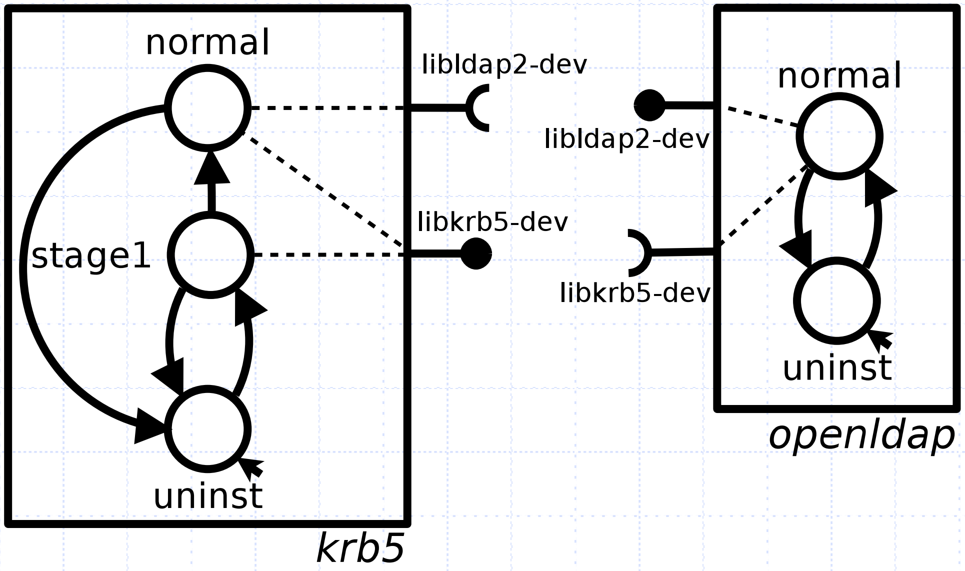
- Consider the problem of installing kerberos with ldap support in Debian
  - Universe: packages krb5 and openldap
  - Target: krb5 in normal state



Deployment problem: example

- Deployment plan:

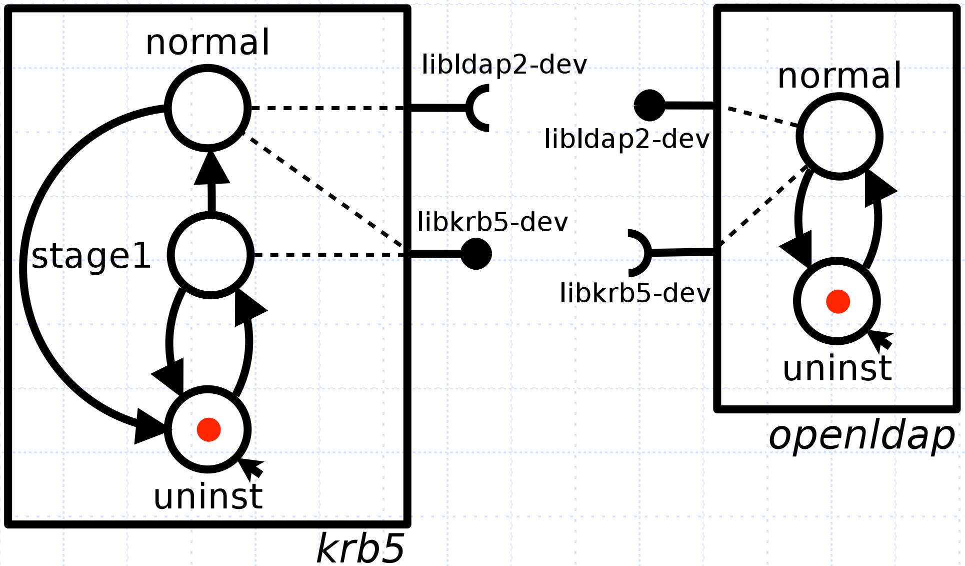
```
newRsrc(krb5), newRsrc(openldap),
stage1(krb5), bind(libkrb,openldap,krb5),
normal(openldap), bind(libldap,krb5,openldap),
normal(krb5)
```



Deployment problem: example

- Deployment plan:

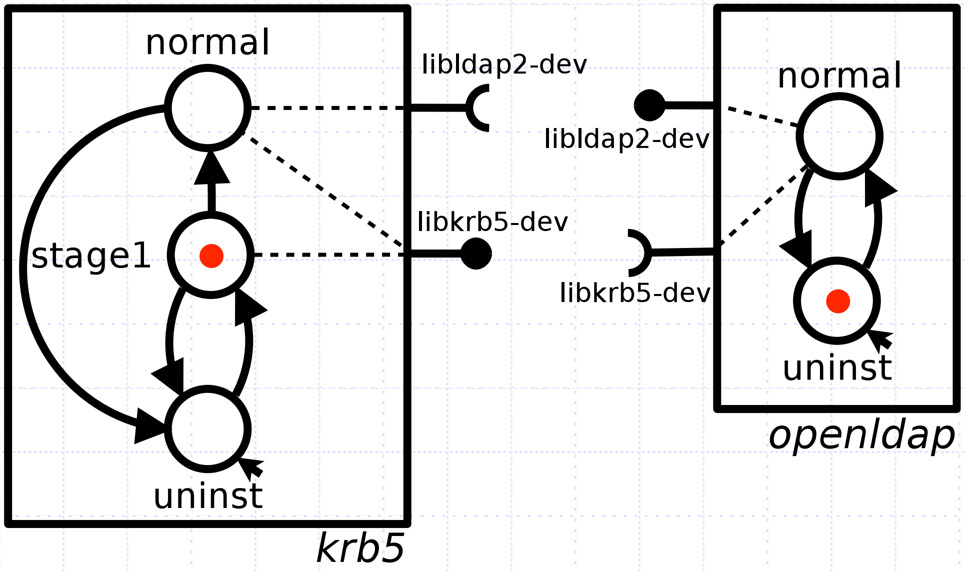
  newRsrc(krb5), newRsrc(openldap),
  stage1(krb5), bind(libkrb,openldap,krb5),
  normal(openldap), bind(libldap,krb5,openldap),
  normal(krb5)



Deployment problem: example

- Deployment plan:

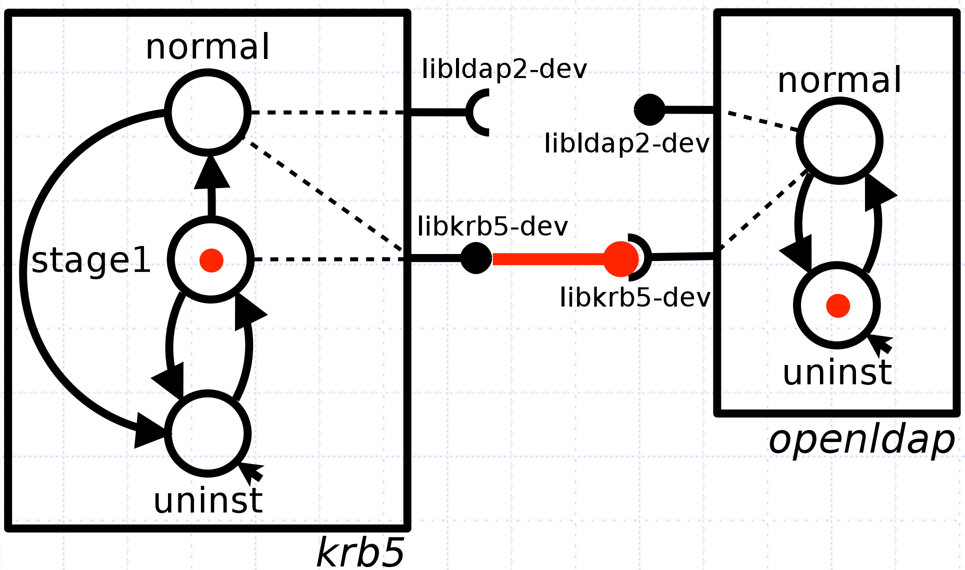
`newRsrc(krb5), newRsrc(openldap),`
`stage1(krb5), bind(libkrb,openldap,krb5),`
`normal(openldap), bind(libldap,krb5,openldap),`
`normal(krb5)`



Deployment problem: example

- Deployment plan:

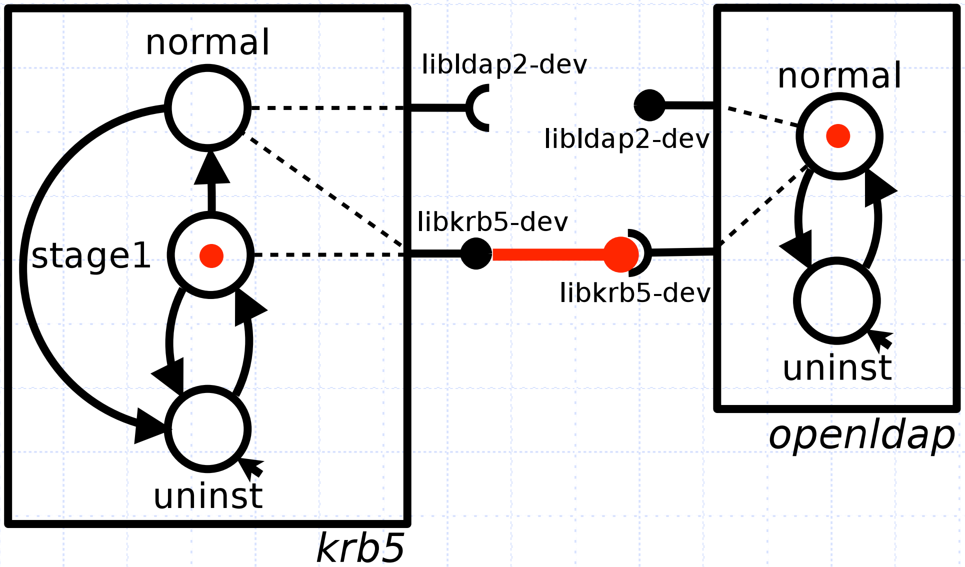
```
newRsrc(krb5), newRsrc(openldap),
stage1(krb5), bind(libkrb,openldap,krb5),
normal(openldap), bind(libldap,krb5,openldap),
normal(krb5)
```



Deployment problem: example

- Deployment plan:

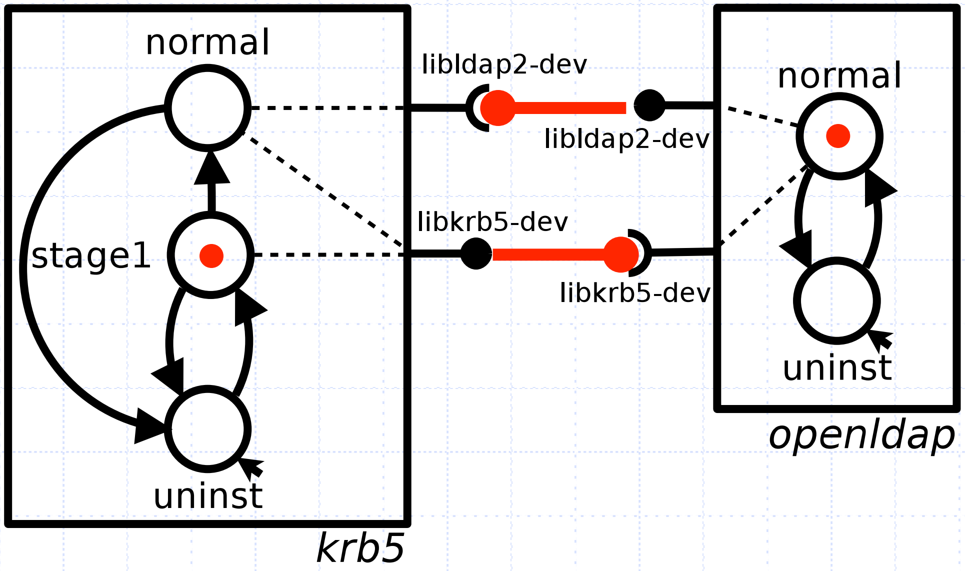
  newRsrc(krb5), newRsrc(openldap), stage1(krb5), bind(libkrb,openldap,krb5), normal(openldap), bind(libldap,krb5,openldap), normal(krb5)



Deployment problem: example

- Deployment plan:

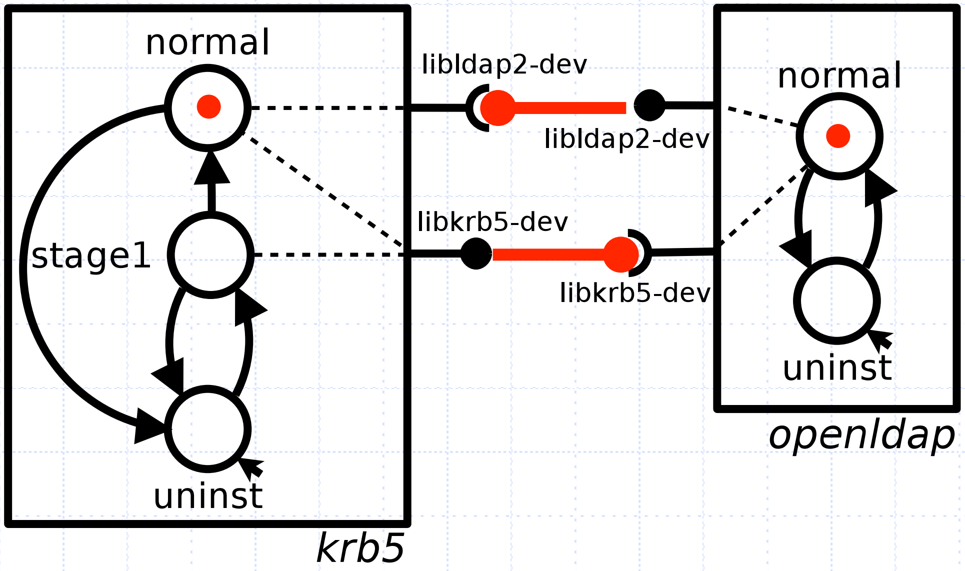
  newRsrc(krb5), newRsrc(openldap), stage1(krb5), bind(libkrb,openldap,krb5), normal(openldap), bind(libldap,krb5,openldap), normal(krb5)



Deployment problem: example

- Deployment plan:

newRsrc(krb5), newRsrc(openldap), stage1(krb5), bind(libkrb,openldap,krb5), normal(openldap), bind(libldap,krb5,openldap), normal(krb5)

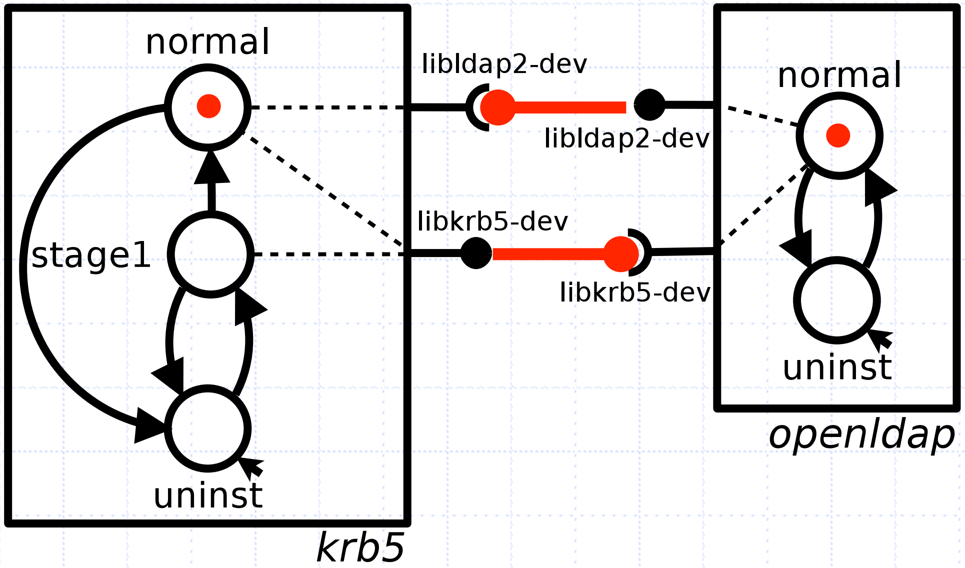


Structure of the talk

- The Aeolus starting point
- Formalizing the “deployment” problem
- **Solving the “deployment” problem**
  - Ackermann-hard in the general case
  - PolyTime without conflicts
- Open issues and related work

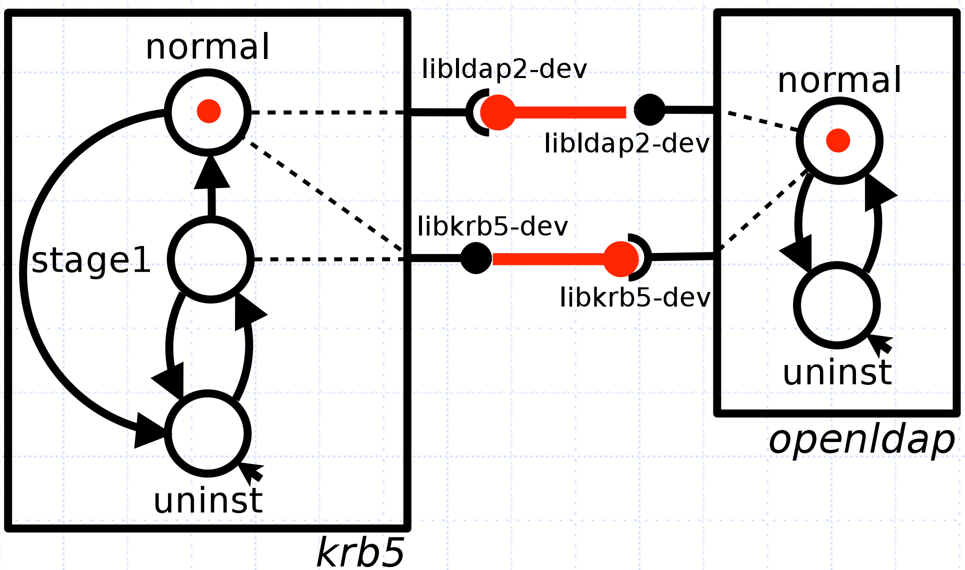
Reduce to well-known concurrent models? (as SAT for packages)

- Deployment plans recall firing sequences in **Petri nets**:
  - Tokens are moved from source places to target places by transitions



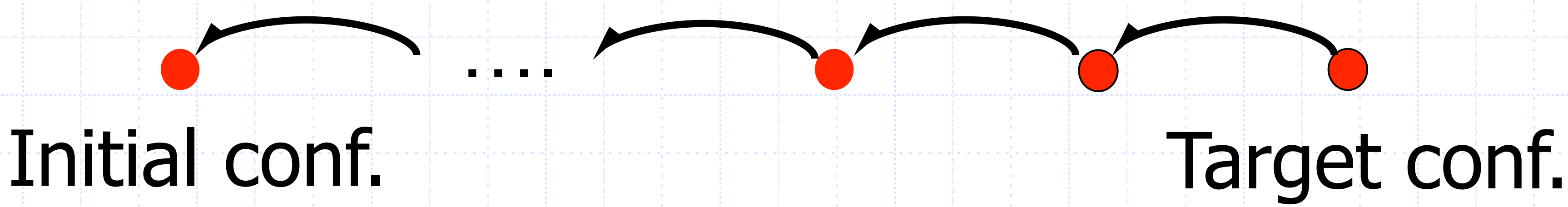
Reduce to well-known concurrent models? (as SAT for packages)

- ...but reachability problems in Petri nets are **undecidable** in the presence of inhibitor arcs (necessary for conflicts)



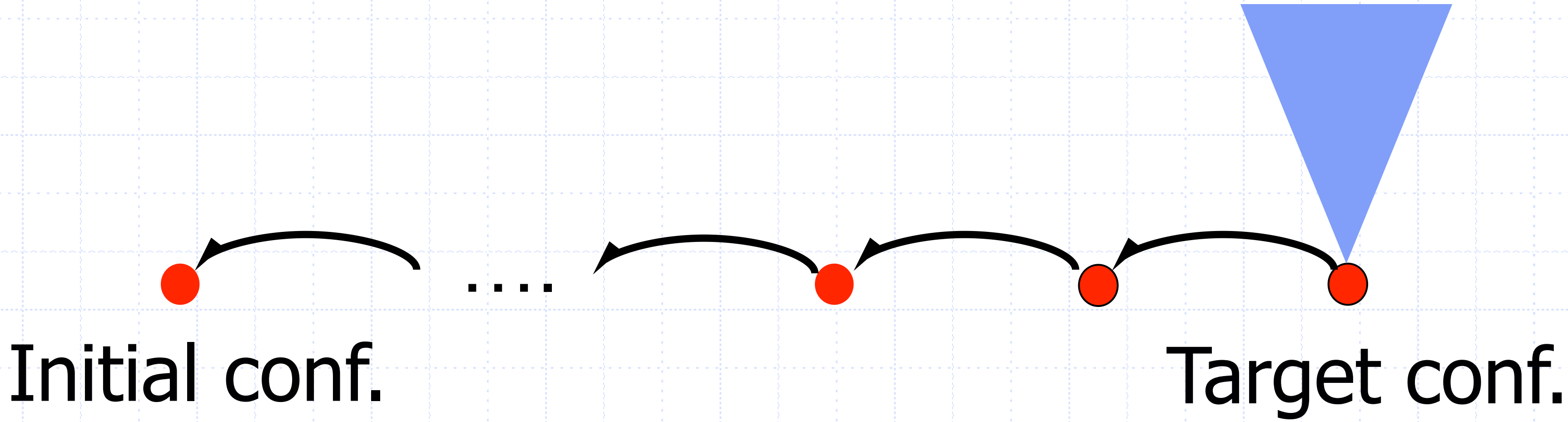
Decidability of the "deployment" problem [ICALP'13]

- **Backward** search algorithm based on the theory of WSTS (Well-Structured Transition Systems)
  - WSTS are popular in the context of infinite state concurrent systems verification



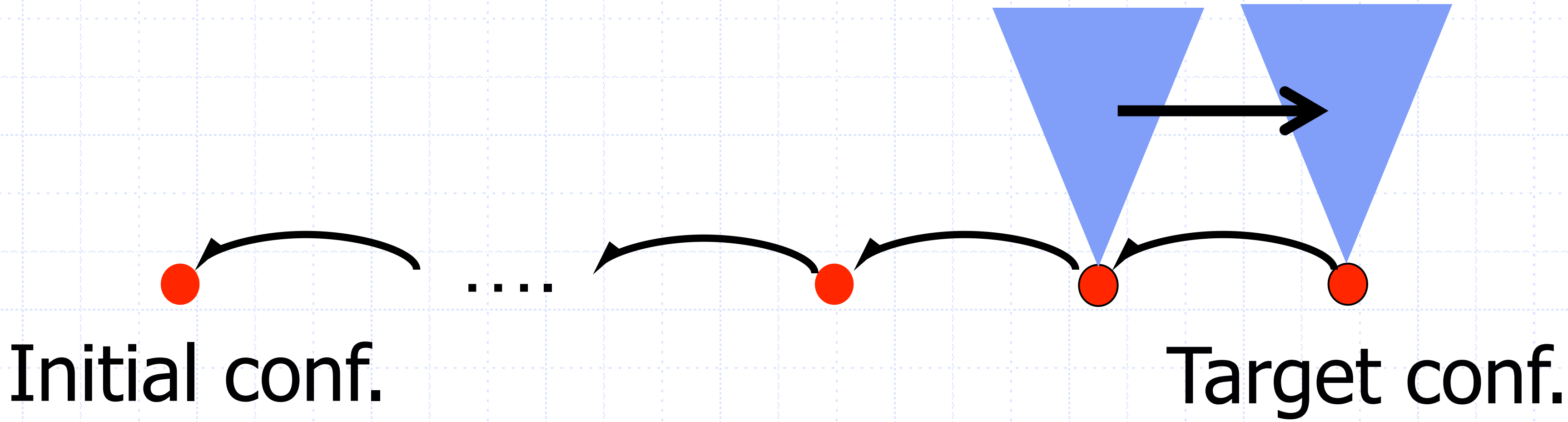
Decidability of the "deployment" problem [ICALP'13]

- Key point:
  ordering $C_1 \leq C_2$ on configurations s.t.
  - if C_1 has a given component, also C_2 has it



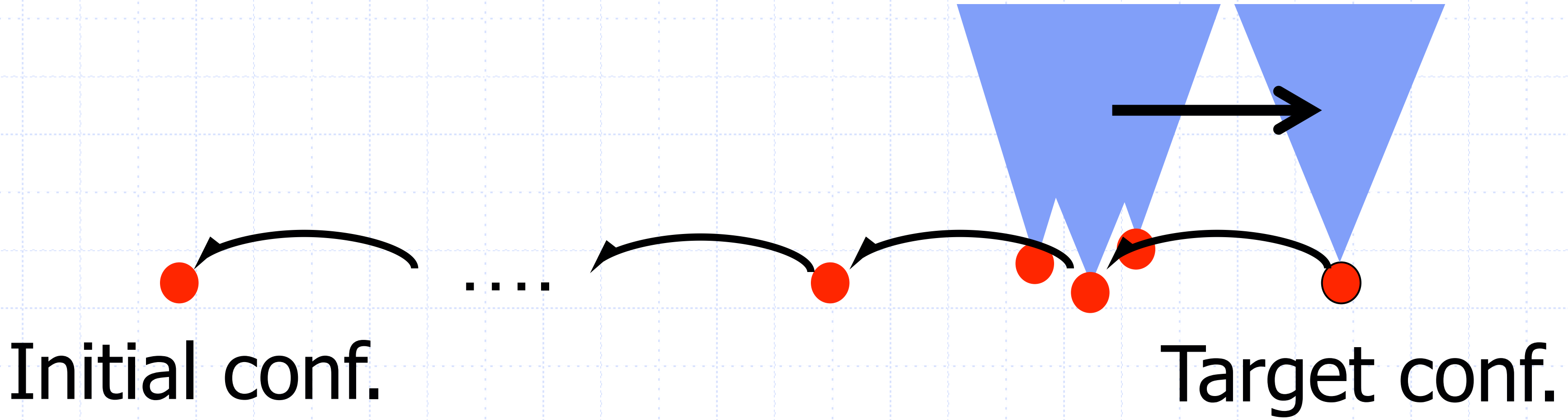
Decidability of the "deployment" problem [ICALP'13]

- Key point:
  ordering $C_1 \leq C_2$ on configurations s.t.
  - if C_1 has a given component, also C_2 has it
  - if $C_1 \leq C_2$ and $C_1 \rightarrow C_1'$ then $C_2 \rightarrow C_2'$ with $C_1' \leq C_2'$



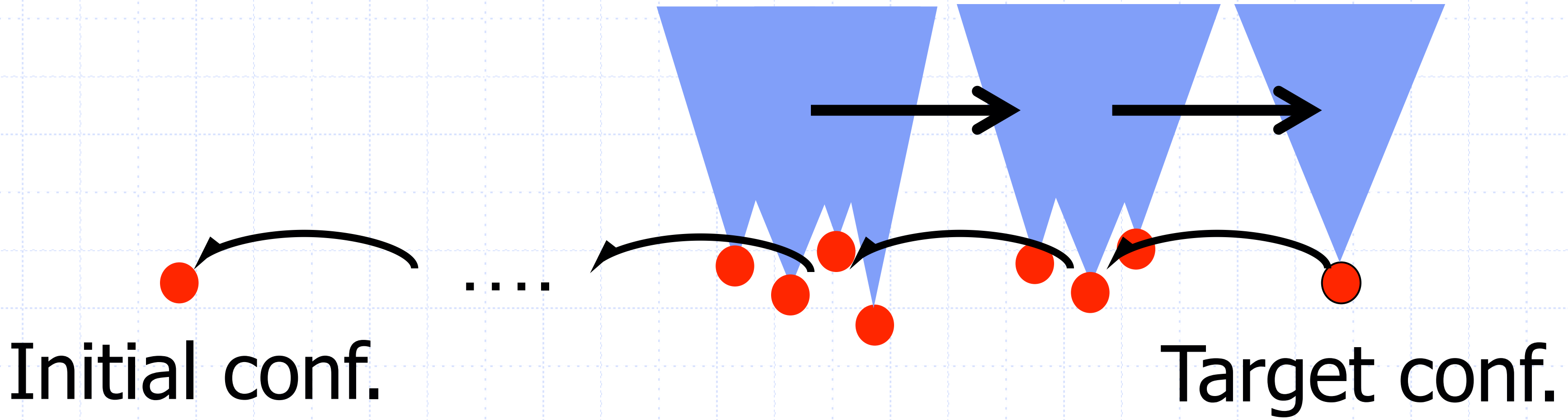
Decidability of the "deployment" problem [ICALP'13]

- Key point:
  ordering $C_1 \leq C_2$ on configurations s.t.
  - if C_1 has a given component, also C_2 has it
  - if $C_1 \leq C_2$ and $C_1 \rightarrow C_1'$ then $C_2 \rightarrow C_2'$ with $C_1' \leq C_2'$
  - $\leq$ is a wqo: finite basis and finite antichains



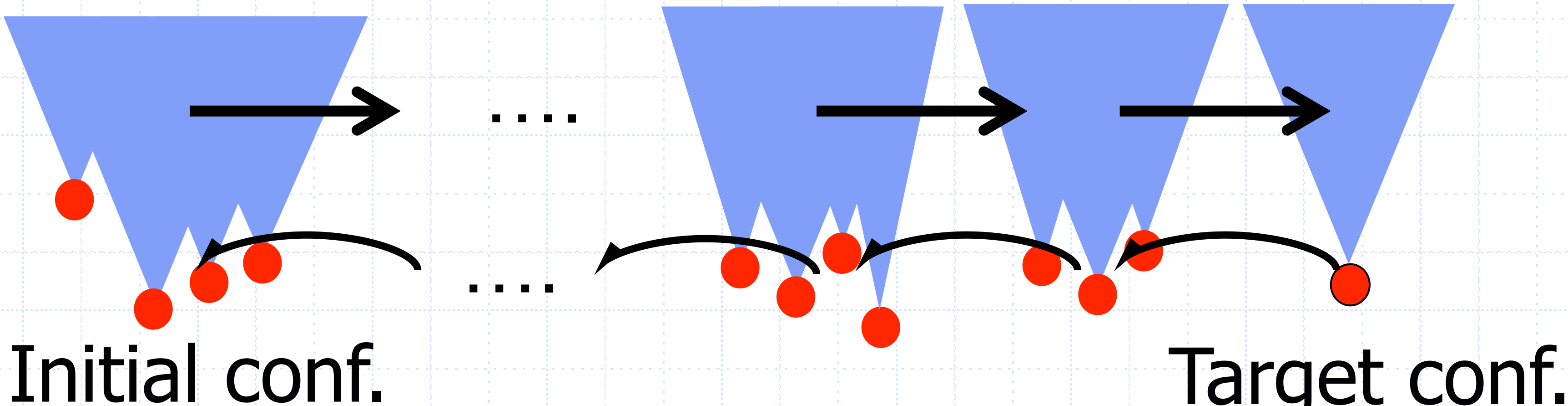
Decidability of the "deployment" problem [ICALP'13]

- Key point:
  ordering $C_1 \leq C_2$ on configurations s.t.
  - if C_1 has a given component, also C_2 has it
  - if $C_1 \leq C_2$ and $C_1 \rightarrow C_1'$ then $C_2 \rightarrow C_2'$ with $C_1' \leq C_2'$
  - $\leq$ is a wqo: finite basis and finite antichains



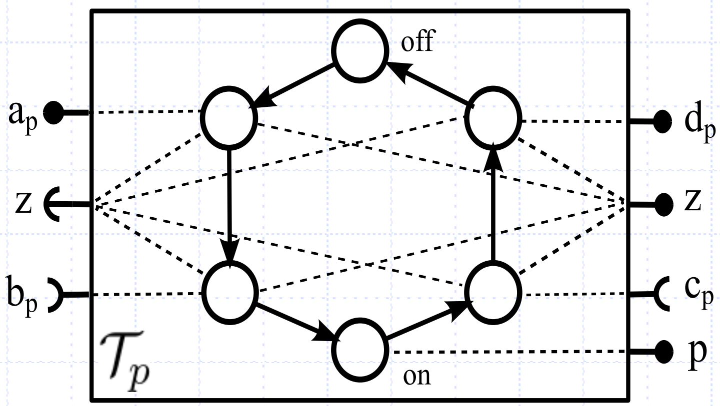
Decidability of the "deployment" problem [ICALP'13]

- Key point:
  ordering $C_1 \leq C_2$ on configurations s.t.
  - if C_1 has a given component, also C_2 has it
  - if $C_1 \leq C_2$ and $C_1 \rightarrow C_1'$ then $C_2 \rightarrow C_2'$ with $C_1' \leq C_2'$
  - $\leq$ is a wqo: finite basis and finite antichains

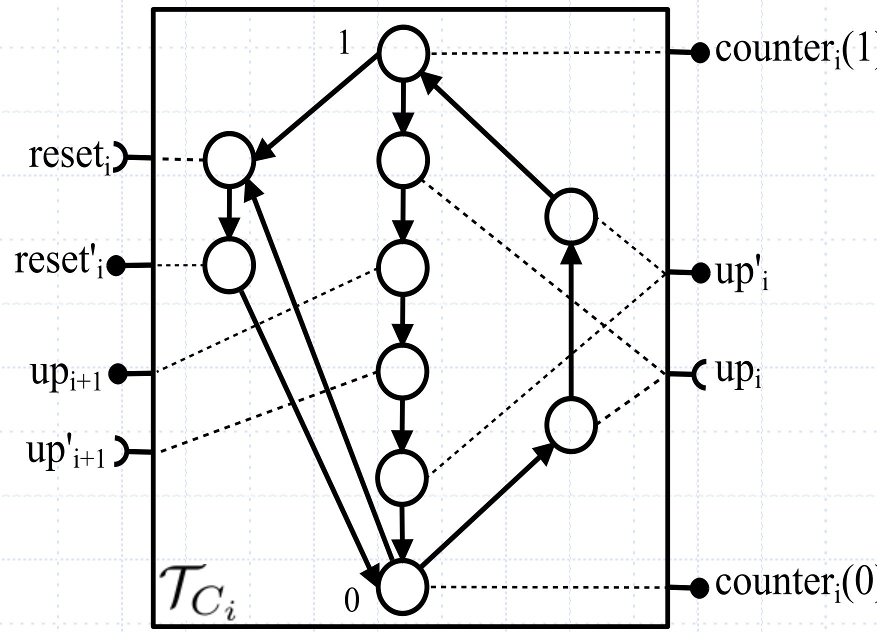


Complexity [ICALP'13]

- The complexity of the problem is Ackermann-hard (reduction from **coverability** in reset Petri nets)



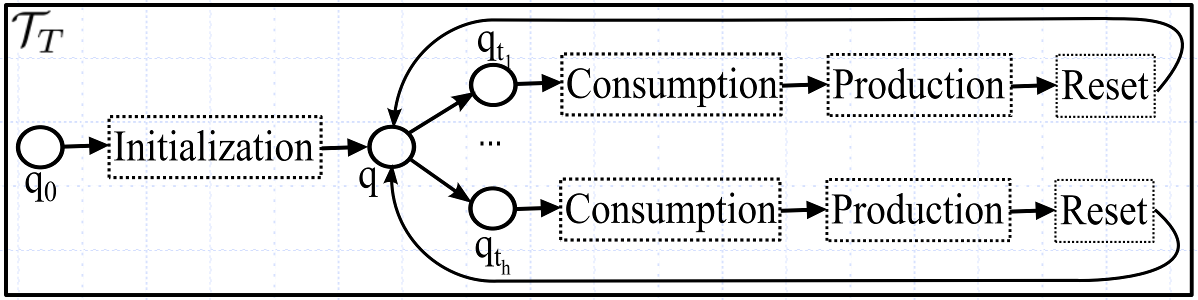
(a) Token in place p.



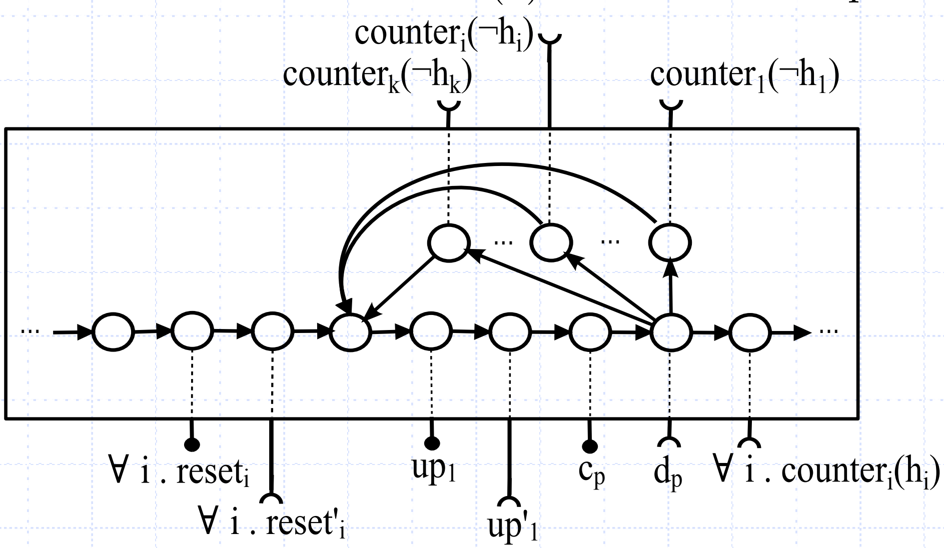
(b) i-th bit counter.

Complexity

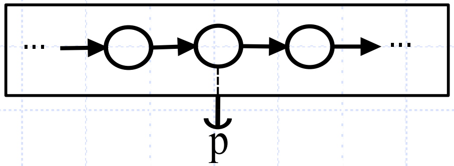
[ICALP'13]



(a) Transitions component.



(b) Consumption phase of n tokens from place p for a transition t ($k = \lceil \log(n) \rceil$ and h_i is the i-th least significative bit of the binary representation of n).



(c) Encoding of a reset arc for the place p.

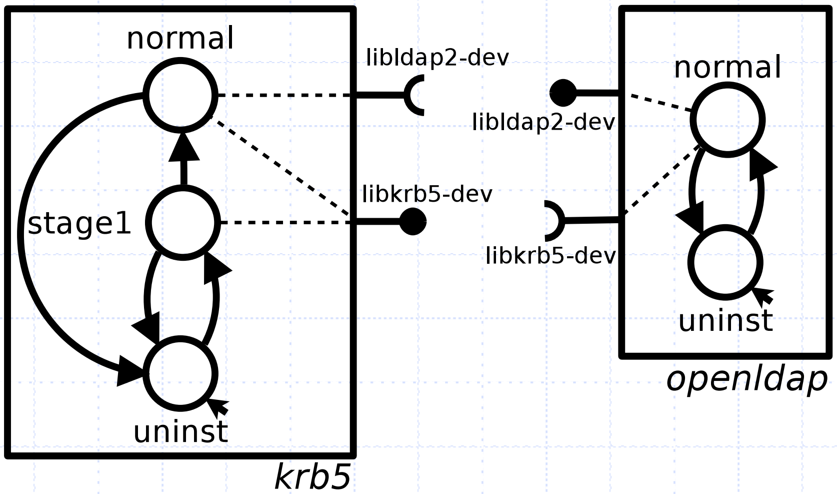
Quadratic algorithm (without conflicts) [SEFM'12]

- **Forward** reachability algorithm
  - all reachable states computed by saturation

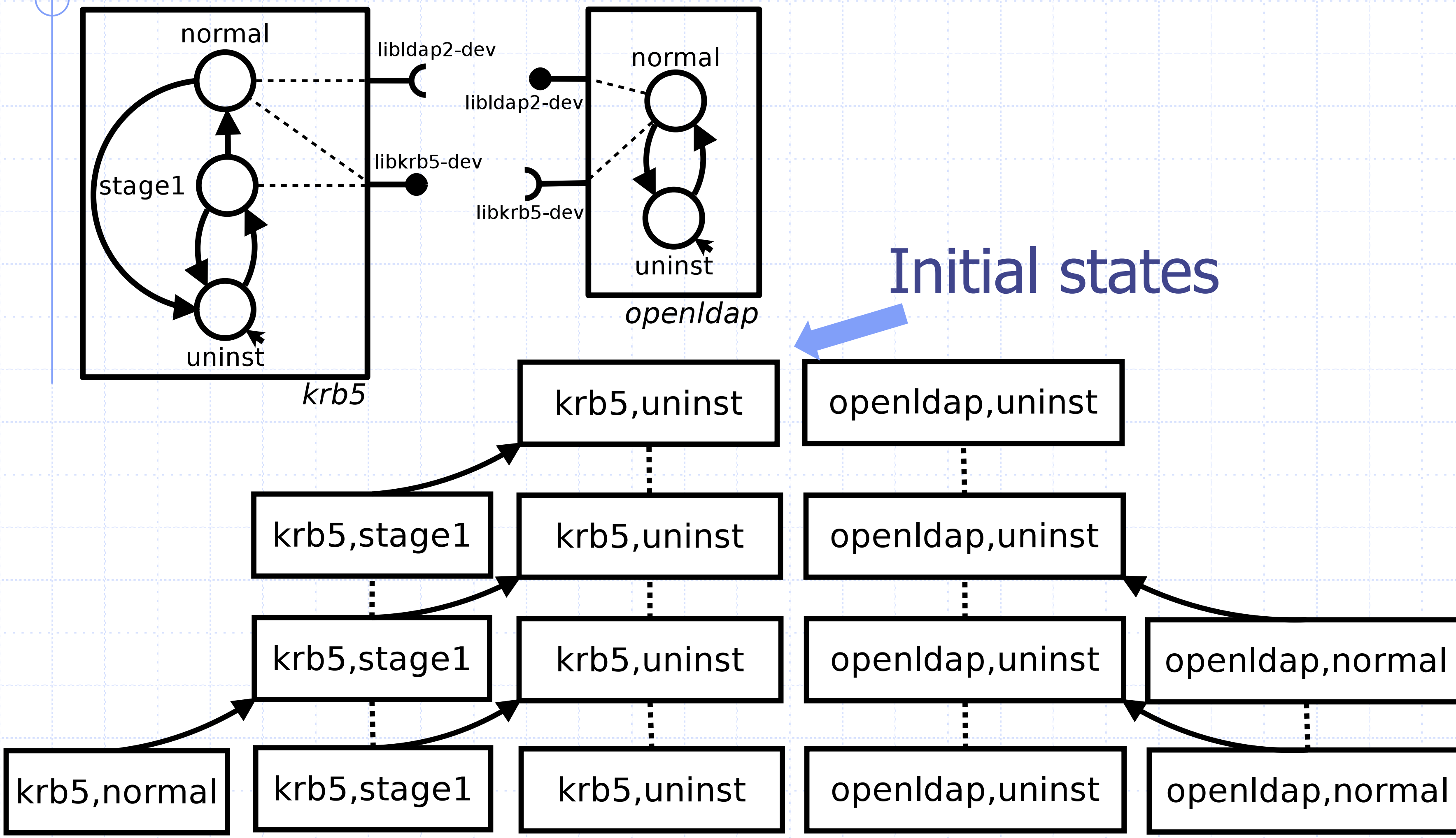
Algorithm 1 Checking achievability in the Aeolus$^-$ model

function ACHIEVABILITY(U, $\mathcal{T}$, q)
  $absConf := \{\langle \mathcal{T}', \mathcal{T}'.\texttt{init}\rangle \mid \mathcal{T}' \in U\}$
  $provPort := \bigcup_{\langle \mathcal{T}', q'\rangle \in absConf} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
  repeat
    $new := \{\langle \mathcal{T}', q'\rangle \mid \langle \mathcal{T}', q''\rangle \in absConf, (q'', q') \in \mathcal{T}'.\texttt{trans}\} \setminus absConf$
    $newPort := \bigcup_{\langle \mathcal{T}', q'\rangle \in new} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
    while $\exists \langle \mathcal{T}', q'\rangle \in new \,.\, dom(\mathcal{T}'.\mathbf{R}(q')) \not\subseteq provPort \cup newPort$ **do**
      $new := new \setminus \{\langle \mathcal{T}', q'\rangle\}$
      $newPort := \bigcup_{\langle \mathcal{T}', q'\rangle \in new} \{dom(\mathcal{T}'.\mathbf{P}(q'))\}$
    end while
    $absConf := absConf \cup new$
    $provPort := provPort \cup newPort$
  until $new = \emptyset$
  if $\langle \mathcal{T}, q\rangle \in absConf$ **then return true**
  else return false
  end if
end function

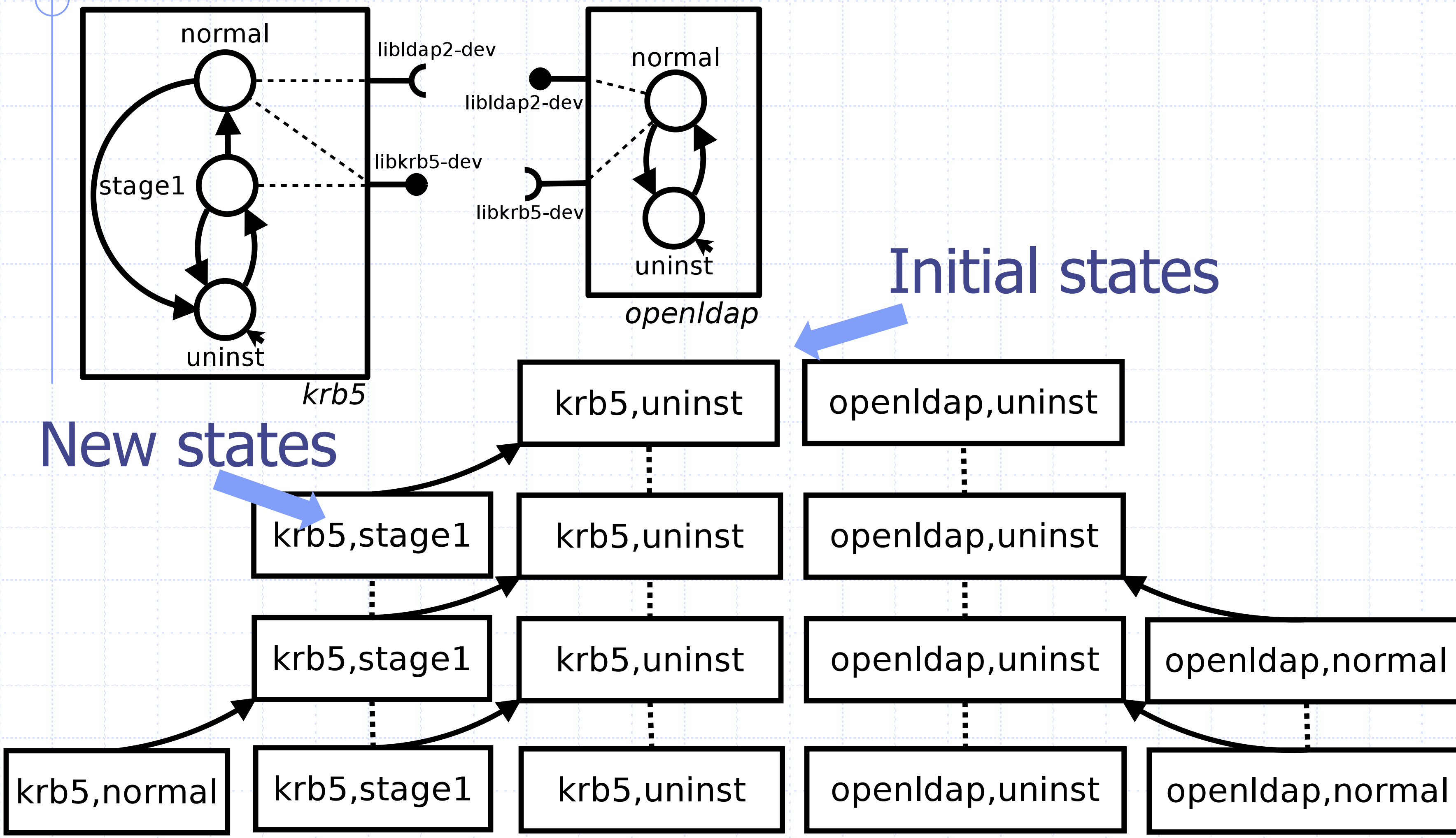
Example: the kerberos case-study



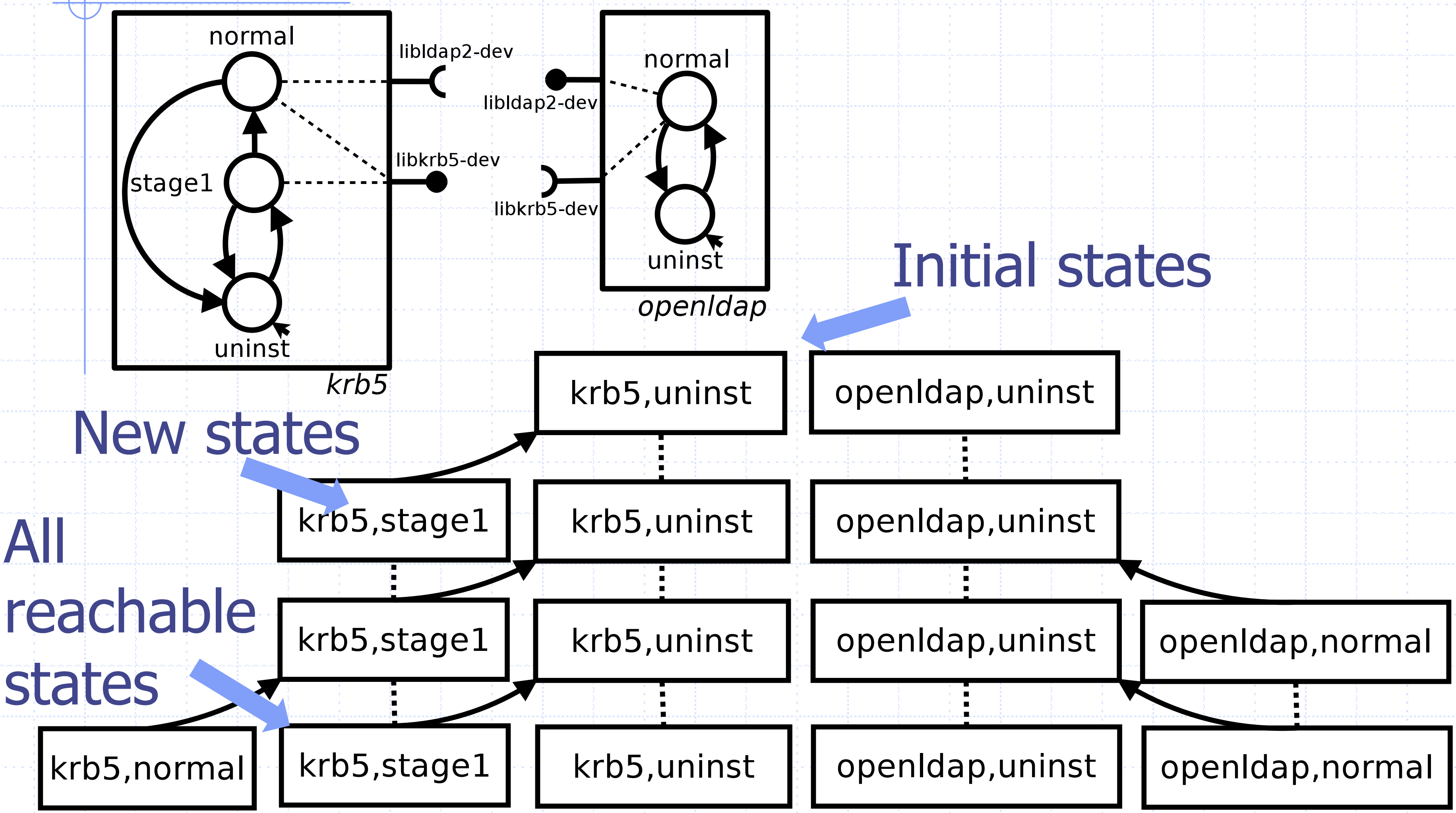
Example: the kerberos case-study



Example: the kerberos case-study



Example: the kerberos case-study



Structure of the talk

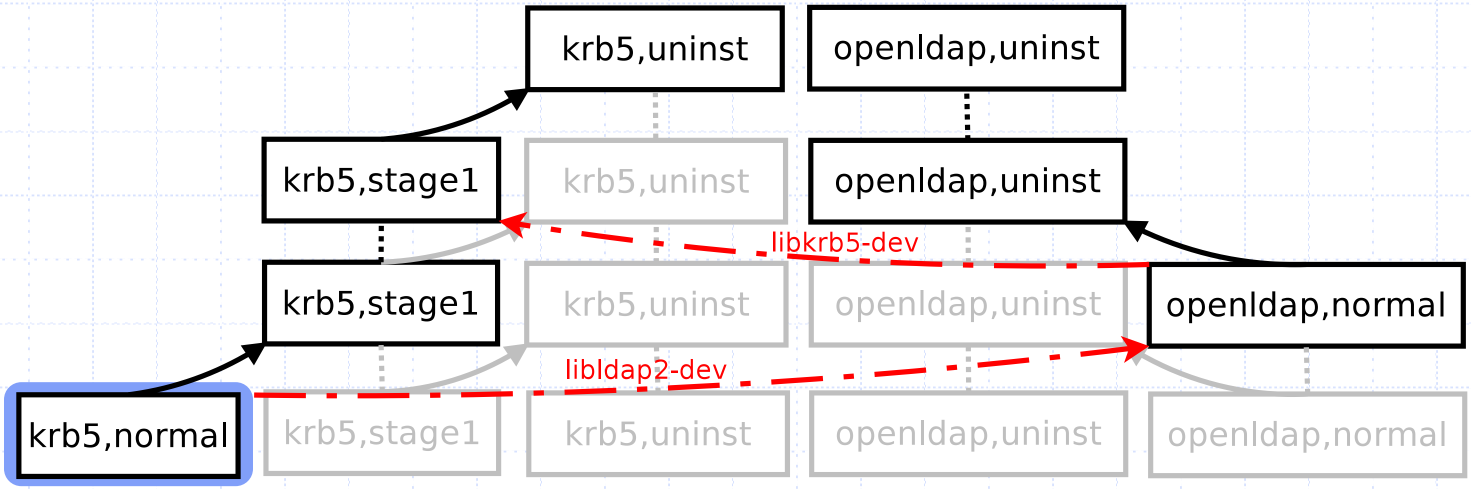
- The Aeolus starting point
- Formalizing the "deployment" problem
- Solving the "deployment" problem
  - Ackermann-hard in the general case
  - PolyTime without conflicts
- **Open issues and related work**

Real-life deployment tools

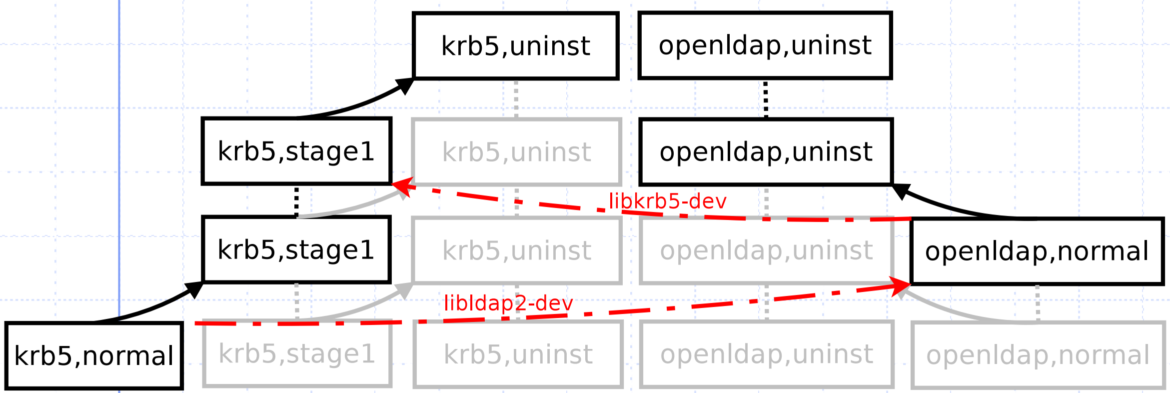
- The deployment problem simply replies **yes** / **no**
- A real deployment tool needs to know how to **reach** the target configuration
  - In other words, an actual deployment **plan** should be computed
- We have preliminary results…

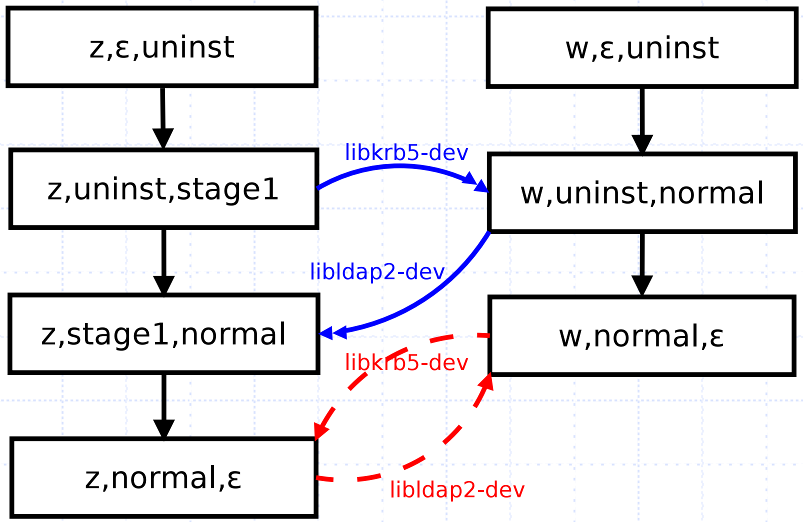
Ad-hoc planning [FACS'13,ICTAI'13]

- Use the reachability graph **bottom-up** from the target state
  - select the **bindings** (red arrows)
  - select the **predecessors** (black arrows)



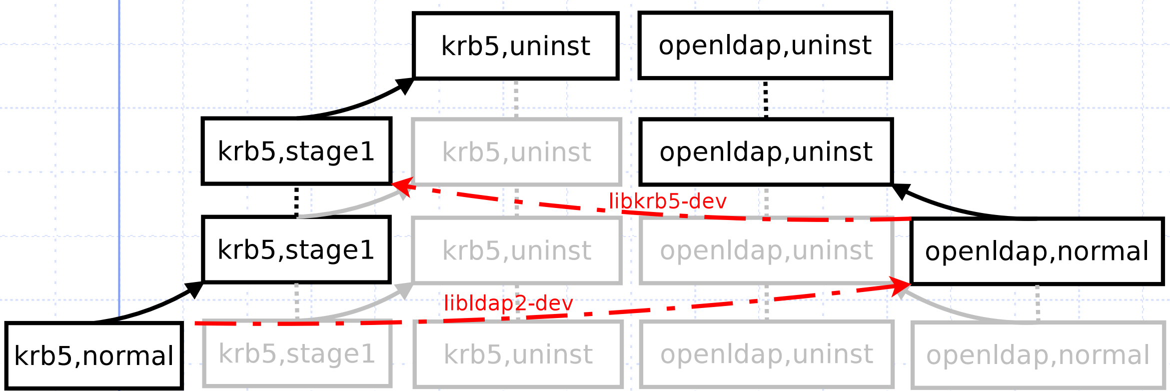
Ad-hoc planning [FACS'13,ICTAI'13]

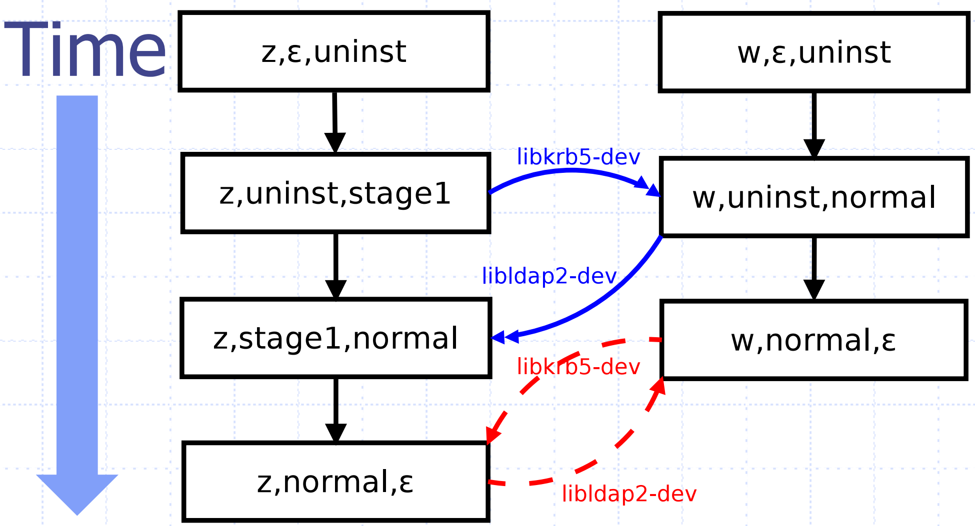




- Generate an **abstract plan** (one component for each maximal path)

Ad-hoc planning [FACS'13,ICTAI'13]

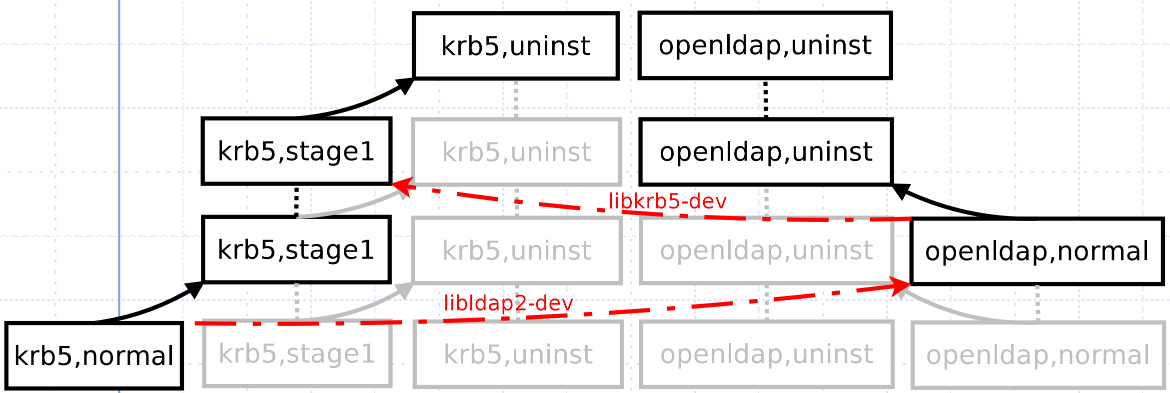




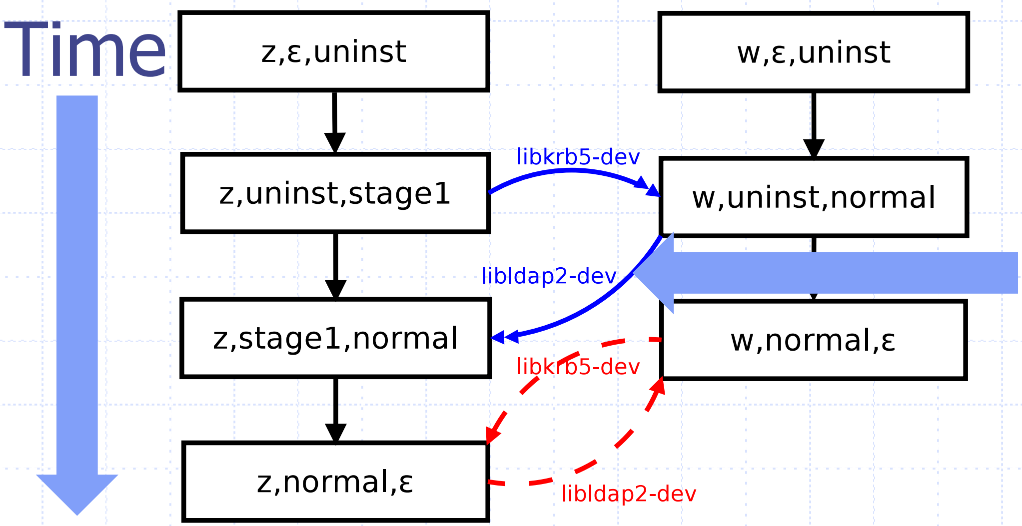
- Generate an **abstract plan** (one component for each maximal path)

Ad-hoc planning

[FACS'13,ICTAI'13]



- Generate an **abstract plan** (one component for each maximal path)



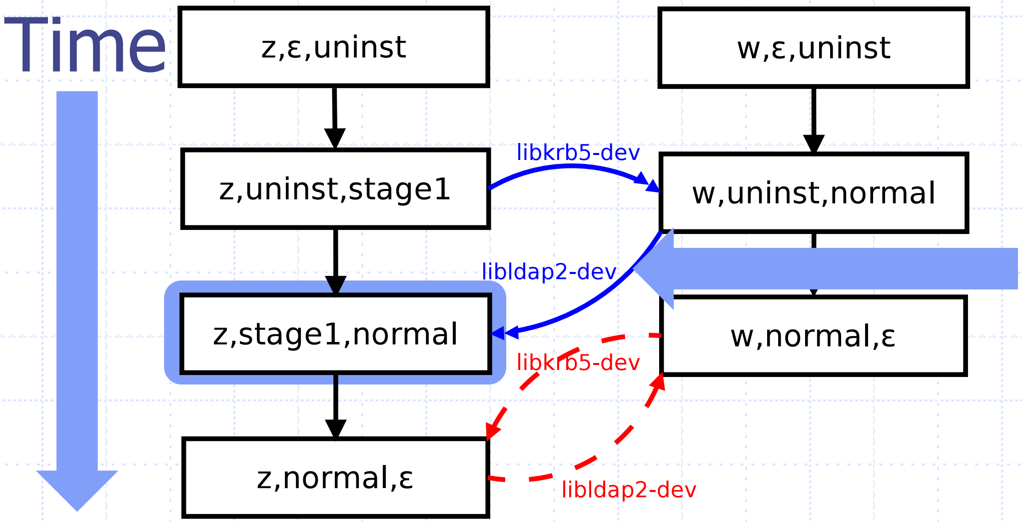
Arrows represent a **precedence** relation:

- **blue**: start requirement
- **red**: end requirement

Ad-hoc planning [FACS'13,ICTAI'13]

- Plan as a **topological** visit until target:

```
newRsrc(krb5), newRsrc(openldap),
stage1(krb5), bind(libkrb,openldap,krb5),
normal(openldap), bind(libldap,krb5,openldap),
normal(krb5)
```



Arrows represent a **precedence** relation:

- **blue**: start requirement
- **red**: end requirement

Reconfiguration vs. Deployment

- **Reconfiguration** problem:
  - same as deployment,
    but with non empty initial configuration
- We recently proved that reconfiguration is PSpace-complete
  (relation with 1-safe Petri nets)
- **Open issue**:
  - Find restrictions to the model that make reconfiguration tractable
    (seems very useful in practice)

Other open issues

- In real systems there is a flow of configuration data among components:
  - Room for **name-passing** models?
- Hierarchical modeling (virtual machines, administrative domains, geographical areas,…):
  - Room for **higher-order** models?
- Services consume resources:
  - Room for **resource-aware** models?

Related work

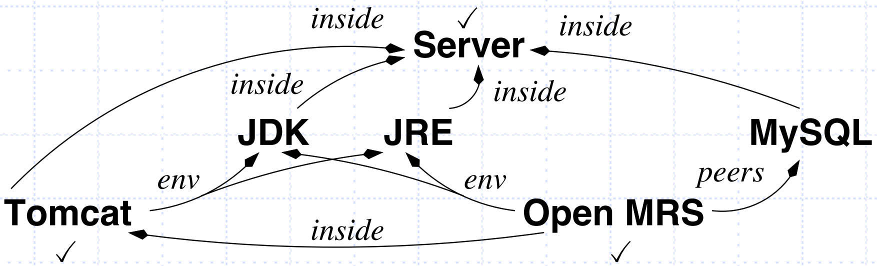
- ConfSolve [J.A.Hewson, P.Anderson, A.D.Gordon - LISA'12]
  - **Object-oriented language** for services and machines
  - **Type system** for checking configuration correctness
  - **Constraint solver** for automatic placement of services on machines

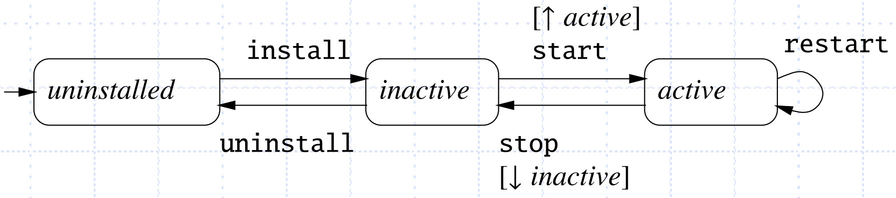
```
class Machine {
  var os as OperatingSystem;
  var cpus as 1..4;
  var memory as int;
}
```

```
class DatabaseServer extends Role {
  var role as DatabaseRole;

  // slave or master
  var peer as ref DatabaseServer;

  // the peer cannot be itself
  peer != this;

  // a master's peer must be a slave,
  // and a slave's peer must be a master
  role != peer.role;
}
```

Related work

- Engage [J.Fischer, R.Majumdar, S.Esmaeilsabzali - PLDI'12]
  - **Architectural specification** in terms of inside / peer / environment relationships
  - Automata with resource **lifecycle** and transient **dependencies**
  - Assumption on **acyclic** relationships (to always guarantee topological visit)





Publications and project web site

- Roberto Di Cosmo, Stefano Zacchiroli, Gianluigi Zavattaro. *Towards a Formal Component Model for the Cloud.* Proc. of SEFM'12: 156-171. LNCS 7504, Springer.
- Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, Gianluigi Zavattaro. *Component Reconfiguration in the Presence of Conflicts.* Proc. of ICALP'13: 187-198. LNCS 7966, Springer.
- Tudor A. Lascu, Jacopo Mauro, Gianluigi Zavattaro. *Automatic Component Deployment in the Presence of Circular Dependencies.* Proc. of FACS'13. LNCS to appear, Springer.
- Tudor A. Lascu, Jacopo Mauro, Gianluigi Zavattaro. *A Planning Tool Supporting the Deployment of Cloud Applications.* Proc. of ICTAI'13: 213-220. IEEE Press.
- http://www.aeolus-project.org